**Purpose of the experiment:**

In this lab, we will be considering the free diffusion of a particle in 2D and calculate its entropy as a function of time.

**Problem Description:**



*(Equation 1 – probability of finding the state of a system at certain time)*

The script will model an equation (1) and calculate the entropy of a closed system. P\_i is the probability of finding the system in state I, and the sum is over all possible states. For this lab where it focuses on the rate of diffusion of 2D, each state will be defined with respect to the a position in the x-y plane. P\_i at each time iteration, n, is the probability of finding a random walker that has n steps at this position.

In lab 8, the particles we will be start at the origin of an x-y plane (coordinates (0,0)). We then assume the motion is random walk. Each step will update the position of the x and y component of the particle randomly and independently.

Before I did anything, I imported the libraries necessary for plotting, mathematical functions and the random function as shown in figure (1).

The first part of the script is where I declared the following constants num\_steps = 500, num\_walkers = 1000. num\_walkers represented the number of times to repeat the situation and the number of walkers. num\_steps represented the length of the random walks.

import numpy as np

from numpy.random import random as rng

import matplotlib.pyplot as plt

num\_steps = 500

num\_walkers = 1000

x\_coor = np.zeros([num\_walkers, num\_steps])

y\_coor = np.zeros([num\_walkers, num\_steps])

*(Figure (1) – Importing the important libraries useful for this script and declaring constants)*

I then created two arrays to store the coordinates of each walker over time, x\_coor and y\_coor. To create a random walk for each walker, a loop is defined and executed.

In the loop, two arrays, x\_step and y\_step are created which hold the values of the random steps taken by each walker in each direction. Both arrays will have 500 elements with either zero or one as each value for each element. To create a binary distribution, I had to identify when each element is greater or less than 0.5. If the element is greater than .5, it will move in the positive direction. If it is smaller than 0.5, it will be in the negative direction.

x\_step and y\_step represent the steps taken by the random walker. I add the previous values of each step together to find the position of the random walker at each time step. I then plotted the position at each x and y value.

x and y will represent the coordinates of the random walker at each time. We will store these coordinates into the corresponding arrays x\_coor and y\_coor at each walker j. I then plot the random walk of each walker in such a way to not reproduce too many results. The resulting plot is shown below of the first part of the calculations done for the loop in figure (2).

for j in range(num\_walkers):

x\_step = rng(num\_steps)

y\_step = rng(num\_steps)

x\_step = 2 \* (x\_step >= 0.5) - 1

y\_step = 2 \* (y\_step >= 0.5) - 1

x = np.cumsum(x\_step)

y = np.cumsum(y\_step)

x\_coor[j,:] = x

y\_coor[j,:] = y

plt.plot(x\_coor[:,num\_steps-1],y\_coor[:,num\_steps-1],'o')

*(Figure 2 – the first part of the calculations done in the loop)*

The entropy is then calculated in the loop. The number of walkers on the surface is need to be known. Calculating the distribution is achieved by first dividing the 2D surface into a 64x64 grid and use then using the histogram2d function to calculate the histogram of the walker’s position at each time step. I then plotted random walk in two dimensions which was done outside the loop.

I defined the following variables in figure 3.

nbins = 64

xedgeds = np.linspace(-200,200,nbins+1)

yedgeds = np.linspace(-200,200,nbins+1)

*(Figure 3 – defining variables for the calculation of entropy)*

Two variables xedges and yedges store the coordinates of the grid points in x and y dimensions. It can be defined in terms of num\_steps.  
  
In the script, within the loop, the array Entropy is then defined at each time step and initialized to be zero with 500 elements. I then defined a loop as show in figure 4:

for s in range(num\_steps):

h = np.histogram2d(x\_coor[:,s], y\_coor[:,s], bins=(xedges, yedges))

p = h[0]/sum(sum(h[0]))

v = 0

for i in range(nbins):

for j in range(nbins):

if (p[i,j]>0):

v+=-p[i,j]\*np.log(p[i,j])

Entropy[s] = v

*(Figure 4 – the for loop to calculate the Entropy and store it in appropriate variables)*

The first command within the for loop as shown in figure 4 calculates the histogram and writes it into a variable called h. h consists of three arrays. The last two elements contain the coordinates of the bins in x and y directions, respectively. In this case, they each have 64 elements. h[1] and h[2] contain the coordinates of the bins in x and y directions respectively. Each element in h[0] will represent the number of walkers, and thus, we can define probability of finding the walkers at position [i,j] as shown in figure 4. v is the entropy of the system at time step and it is stored in Entropy at the very last line in figure 4.

After the loop, I then plotted entropy. When it came to the graphing and calculation of the particles in a closed system, I changed the code above into 2 functions. The first one was function calc2D which included part of the code from figure 1 and the code from figure 2. It also included the graphing blocks of code where it would graph the histograms and the random walks. The block of code I added in the function is shown in figure 5. I have a conditional statement where it checks to see if the system is closed or not. When it is a closed system, it executes a for while loop that would cause the particles to “bounce” off the walls by multiplying x\_steps and y\_steps to negative one.

if (closed == 1):

for i in range(num\_steps):

while (x[i] < -100 or x[i] > 100 or y[i] < -100 or y[i] > 100):

x\_step[i:num\_steps] = -1 \* x\_step[i:num\_steps]

x[i:num\_steps] = np.cumsum(x\_step[i:num\_steps]) +x[i-1]

y\_step[i:num\_steps] = -1 \* y\_step[i:num\_steps]

y[i:num\_steps] = np.cumsum(y\_step[i:num\_steps]) + y[i-1]

*(Figure 5 – shows the calculations used to define behavior of walkers in closed system)*

I then created a function for entropy. It is like figure 4 and 3 except it has a conditional statement on whether it will be executed or not. It is contained within the function Entropy as shown in figure 6 by changing the edges from -100 to 100 if it were a closed system. It then graphed the results.

if (closed == 1):

bounds = 100

else:

bounds = num\_steps

*(Figure 6 - The code included to check if its closed or open system for entropy)*

I then created a plotting function to automate most of my graphing needs as shown in figure 7.

def plot2d(title):

plt.figure(title)

plt.xlabel('Distance in the x - direction')

plt.ylabel('Distance in the y - direction')

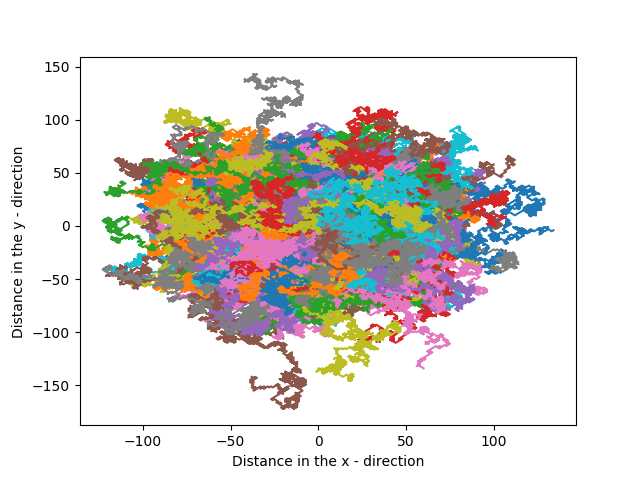
*(Figure 7 - plotting function)*

At the end the script, I then called the function to calculate the entropy and particles movement and passed arguments specifying if it is a closed system or open system.

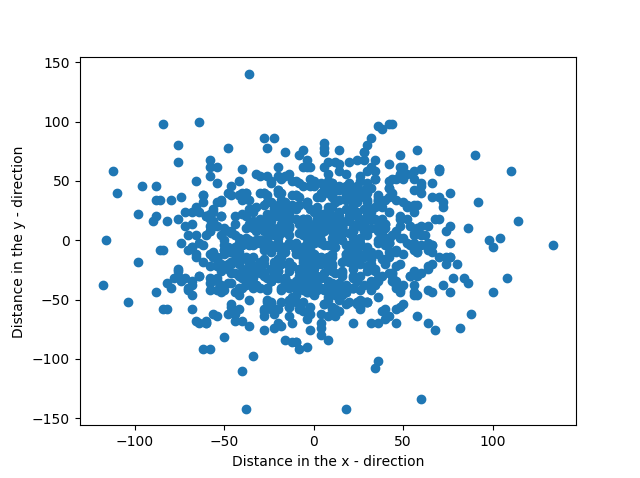
**Equations solved and algorithms used:**

In this lab, I solved for equation 1 by creating a nested for loop. The for loop I used to solve the equation is described in figure 4. I found the finding the probability of finding the state, the probability of finding a random walker that has n steps at a certain position and multiplied them together for each iteration. I then found the sum of all the of each iteration from 0 to 1500. I then modified the code so that it would be able to calculate both closed and open system. For the closed system, I created a for while loop multiplying the x and y steps by negative one to bounce off the borders.

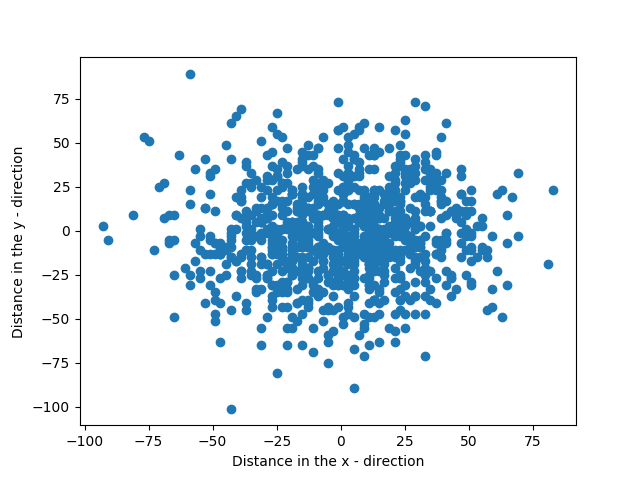
**Results & Analysis**



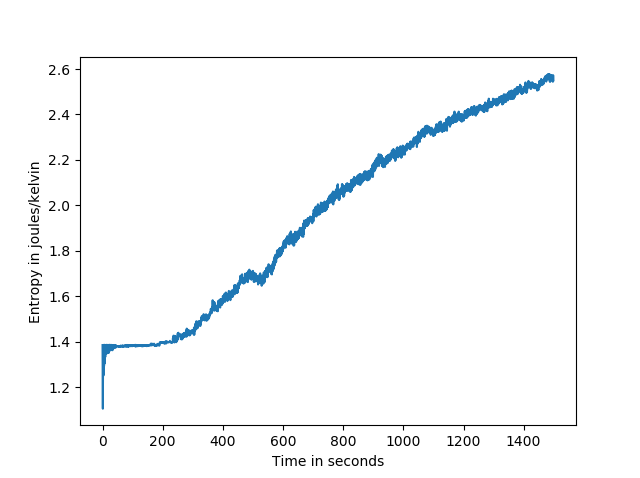
*(Figure 9 – Random walks in an open system)*



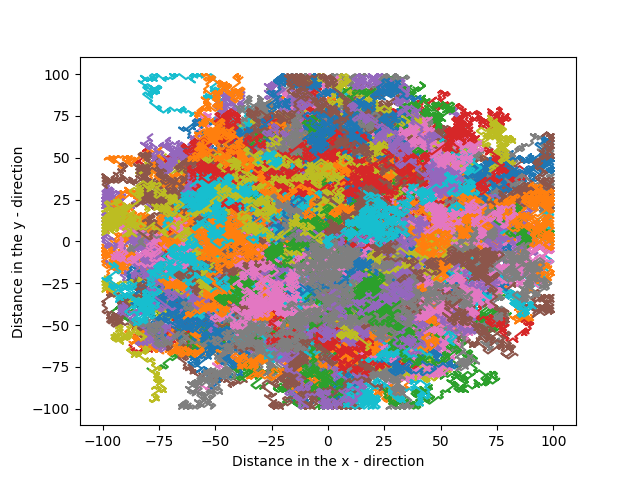
*(Figure 10 – Random Walk with open system at iteration num\_steps)*



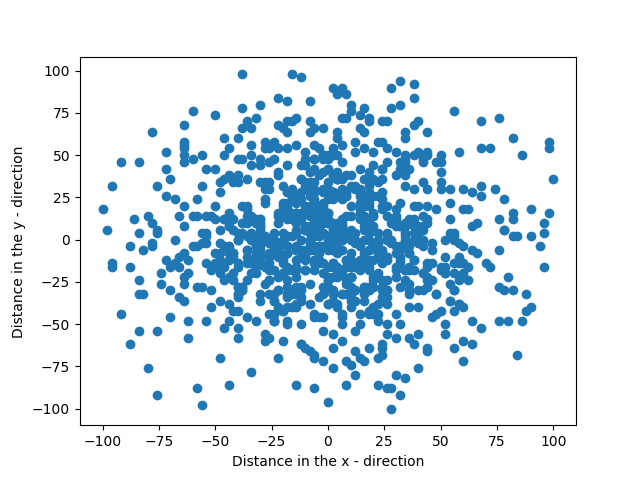
*(Figure 11 – Random walk with open system at iteration num\_steps/2)*



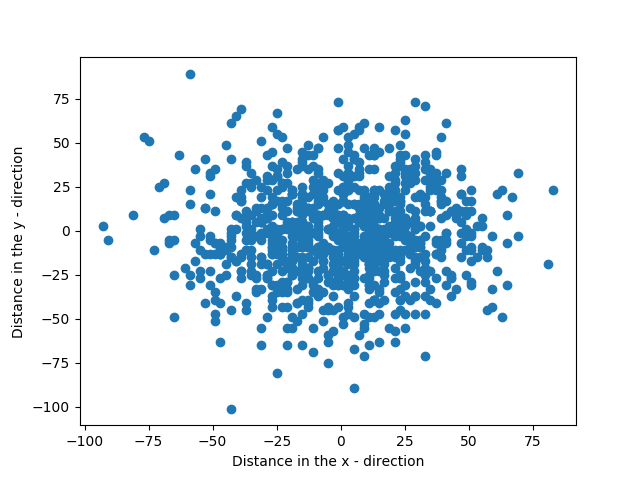
*(Figure 12 – Entropy in an open system)*



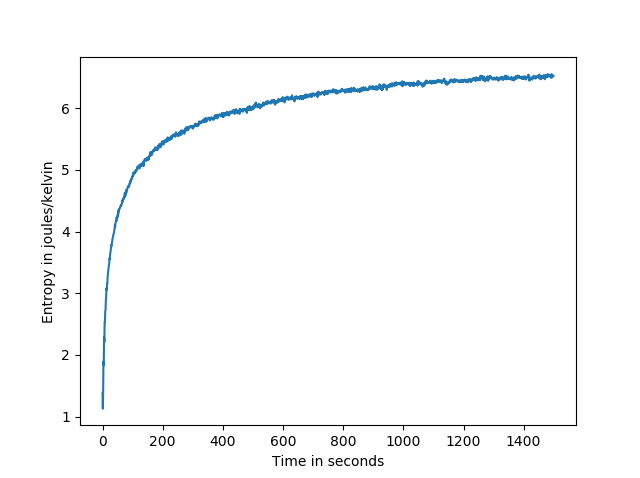
*(Figure 13 – Random walks in an open system)*



*(Figure 14 – Random Walk with open system at iteration num\_steps)*



*(Figure 15 – Random Walk with open system at iteration num\_steps/2)*



*(Figure 16 – Entropy in a closed system)*

In figures 9 – 11 it is shows that the particles at first are close to each other but as time progresses, the particles spread out and which can lead us to conclude that the particles would expand out infinitely since it’s an open system. This is because in Figure 12, as time increases, Entropy also increases, and the probability of finding a particle’s position will most likely be at a random point.

In figures 13 – 15 it is also shown that the particles are expanding away from each other but as time progresses, the particles will be evenly distributed amongst each other. This is because in figure 16, as time progresses towards infinity, the line that describes Entropy starts to have a slope that approaches zero and the probability of finding particles that will be evenly distributed in the system is high, thus reaching equilibrium. This is due to the existence of bounds, hence a closed system.