Thermal Motion Experiment

Abstract:

The purpose of this experiment is to use the theory of Brownian motion to determine Boltzmann's constant and Avogadro's number. There are two results obtained for both Boltzmann's constant and Avogadro's number, from two different methods of estimation. For Boltzmann's constant, $(2.4 \pm 0.2) \times 10^{-23} m^2\ kg\ s^{-2}\ K^{-1}$ was obtained from maximum likelihood method, while $(1.5 \pm 0.1) \times 10^{-23} m^2\ kg\ s^{-2}\ K^{-1}$ was obtained from curve fitting in Python. Using these values, Avogadro's number is found to be $(3.5 \pm 0.3) \times 10^{23}\ mol^{-1}$ and $(5.4 \pm 0.4) \times 10^{23}\ mol^{-1}$ respectively. While the curve fitting is much closer to the accepted values, it should be noted that changing the bins size has significant impact on the output value. Therefore, it can be concluded that the maximum likelihood method is more reliable.

Introduction:

The goal of this experiment is to find Boltzmann's constant using Brownian motion. Then, use the Boltzmann's constant to get Avogadro's number. Einstein showed that theoretically, mean squared distance of Brownian particle will increase linearly with time:

$$< x^2 > = 2Dt$$

where $D = \frac{kT}{\gamma}, \gamma = 6\pi\eta r$. k is the Boltzmann's constant, η is the viscosity of water, and r is radius of the particle. Alternatively, Boltzmann's constant can be obtained by using probability distribution of steps in constant time interval:

$$P(x,t) = \frac{e^{-\frac{x^2}{4Dt}}}{\sqrt{4\pi Dt}}$$

Since the experiment will be done on a 2D surface, the equation above needs to be rederived into the Rayleigh distribution:

$$p(r,t) = \frac{r}{2Dt} e^{-\frac{r^2}{4Dt}}$$

However, the Python curve fitting function will violate some assumptions of least square, and it is more appropriate to use the maximum likelihood estimation for Rayleigh distribution:

$$(2Dt)_{est} = \frac{1}{2N} \sum_{i=1}^{N} r_i^2$$

Finally, to get to Avogadro's number using Boltzmann's constant:

$$N_A = \frac{R}{k}$$

where R is the gas constant.

Method and Equipment:

The Brownian motion will be simulated through microscopic sized beads and will be captured by a microscope with the assistance of computer software to find out the distance travelled.

List of equipment:

- Microscope
- Solution with beads
- Pipette
- Tape
- Grease
- Microscope slide
- Glass cover
- Fluorescence illumination box

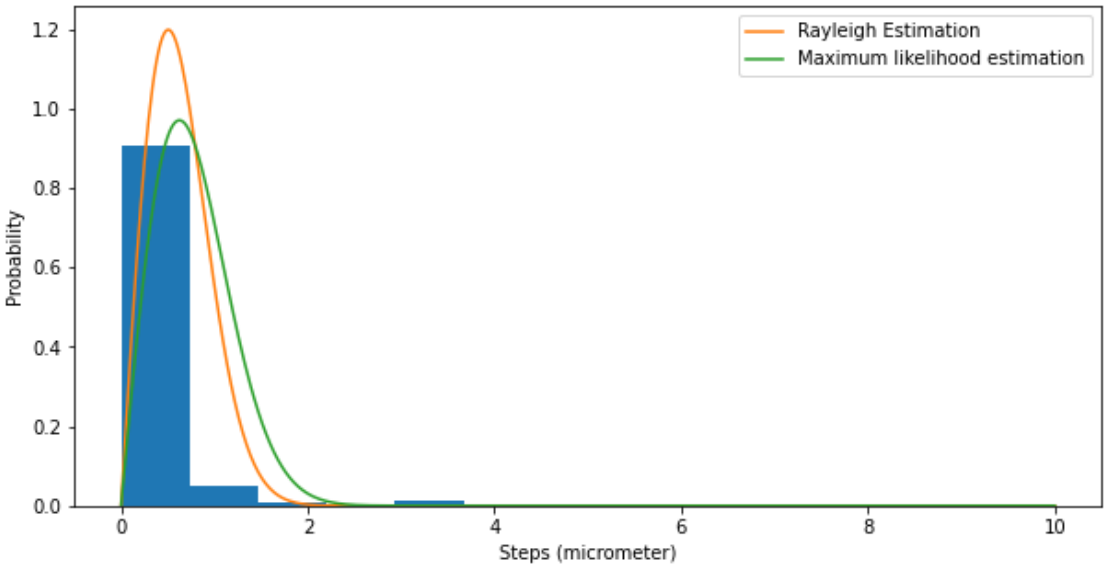
Procedure:

1. Turn on microscope. Turn up illumination intensity and turn phase ring into Ph2.
2. Turn on fluorescence illumination (X-Cite box), rotate GFP fluorescence cube into position 2.
3. Prepare the sample as follow:
    a. On microscope slide, delimit a rectangular space (3 cm by 1 cm) with tape
    b. Drop some grease on the edge of the tape
    c. Pipette 50 μl of beads solution onto the space. Seal with glass cover
4. Move microscope slide in place.
5. Turn on the LabView camera program.
6. Setup Microscope Camera Controller on Multiple Image capture with 120 frames and 2 frames per second.
7. Take 10 sets of data
8. Use Image Object tracker to analyze the displacement for each frame for the selected bead in each set of data.

Results:

The following is the graph of the data, and the probability distribution function using parameter from curve fitting and parameter from maximum likelihood estimation:

Figure. 1
Probability distributions vs. histogram of the data

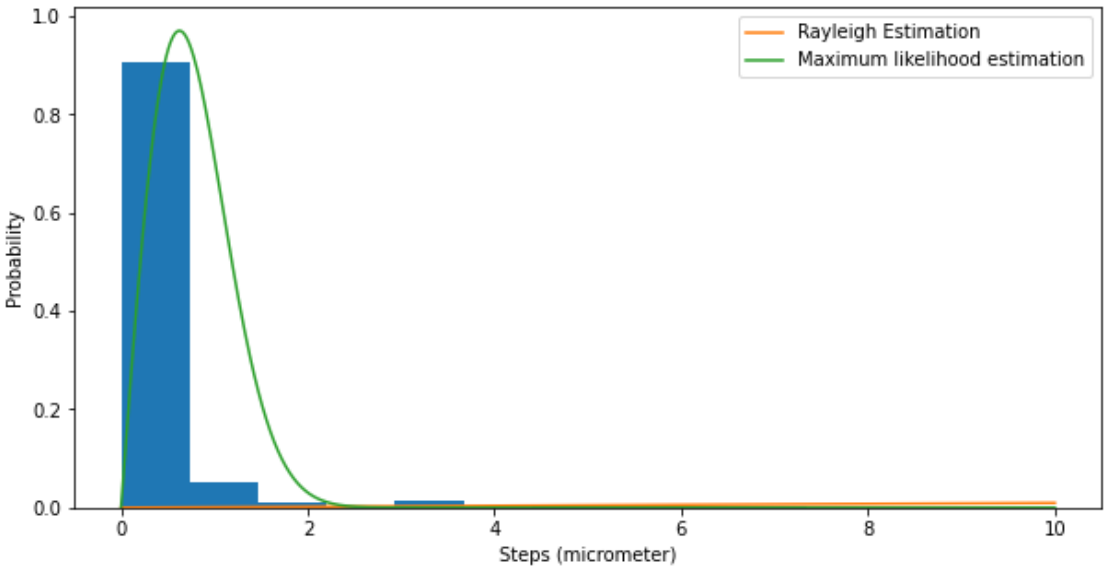


By applying the curve fitting function over the data, the value of 2*Dt* is obtained and used to calculate Boltzmann's constant. With this method, Boltzmann's constant is $(1.5 \pm 0.1) \times 10^{-23} m^2\ kg\ s^{-2}\ K^{-1}$, which has a 12% difference with the accepted value of Boltzmann's constant. Maximum likelihood is also used to estimate 2*Dt*, and the Boltzmann's constant is found to be $(2.4 \pm 0.2) \times 10^{-23} m^2\ kg\ s^{-2}\ K^{-1}$, which has a 71% difference with the accepted value.

Using curve fitting, the Boltzmann's constant obtained is closer to the accepted value. This is also shown on the plotting of the graphs in figure 1, in which the plot that uses the *2Dt* value from curve fitting of Rayleigh distribution is more resemblant to the data than the plot that uses *2Dt* from maximum likelihood. Although, bins sizes are very important when using the curve fitting function. It is known that the function violates some assumptions of least square when data is outputted onto a histogram. If one were to change the bins sizes, the result can be very different.

The following is a plot that illustrates this problem:

Figure 2.
Probability distributions vs. histogram of the data in another bins size



In this plot, it is clear that the curve fitting function did not get the correct probability distribution function for the data. The Boltzmann's constant using *2Dt* from this plot is $6.0 \times 10^{-20}\ m^2\ kg\ s^{-2}\ K^{-1}$, which is significantly larger than the accepted value. Therefore, due to this issue with the curve fitting function, the maximum likelihood estimation is more reliable.

The value of $(2.4 \pm 0.2) \times 10^{-23} m^2\ kg\ s^{-2}\ K^{-1}$ does not agree with the accepted value of $1.38\ \times\ 10^{-23} m^2\ kg\ s^{-2}\ K^{-1}$. This could be due to the lack of data from the experiment. While 10 sets of data were taken, the Image Object Tracker only manage to process the displacement of beads for 4 sets. If more data were available to be processed, the value obtained should be closer to the accepted value. It should also be noted that the sizes of the beads are not all the same, some are larger than the others. For the calculations of Boltzmann's constant, the radiuses are assumed to be the same with small uncertainty. This discrepancy for the radius could be another source of error that affected the Boltzmann's constant calculated.

The other goal of the experiment is to get to Avogadro's number. Using the equation $N_A = \frac{R}{k}$, the number is found to be $(3.5 \pm 0.3) \times 10^{23}\ mol^{-1}$ using the maximum likelihood value, with 41% difference to the accepted value. While the curve fitting value gives $(5.4 \pm 0.4) \times 10^{23}\ mol^{-1}$, with 11% difference to the accepted value.

Conclusion:

From the results of the experiment, using the data available, it is found that the Boltzmann's constant and Avogadro's constant are $(2.4 \pm 0.2) \times 10^{-23} m^2\ kg\ s^{-2}\ K^{-1}$ and $(3.5 \pm 0.3) \times 10^{23}\ mol^{-1}$ respectively, using a method that is more reliable for this instance. The Python's curve fitting function does not always produce the best results, and alternatives are sometimes required. If more data could be processed, the results could be better.

Appendix:

Sample calculation:

Boltzmann's constant using curve fitting:

$$2Dt = 0.256005\mu m^2 = 2.56005 \times 10^{-13} m^2$$

$$\eta = 1\frac{g}{cms} = 0.001\frac{kg}{ms}$$

$$r = 0.95\mu m = 9.5 \times 10^{-7} m$$

$$T = 296.5K, t = 0.5s$$

$$\frac{2kTt}{6\pi\eta r} = \frac{kTt}{3\pi\eta r} = 2Dt$$

$$k = \frac{3\pi\eta r(2Dt)}{Tt}$$

$$k = \frac{3\pi\left(0.001\frac{kg}{ms}\right)(9.5 \times 10^{-7} m)(2.56005 \times 10^{-13} m^2)}{(296.5K)(0.5s)}$$

$$k = 1.546 \times 10^{-23} m^2\ kg\ s^{-2}\ K^{-1}$$

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

x_pos1, y_pos1 = np.loadtxt('data4 tracker.txt', skiprows = 2, unpack = True)
x_pos2, y_pos2 = np.loadtxt('data6 tracker.txt', skiprows = 2, unpack = True)
x_pos3, y_pos3 = np.loadtxt('Data7 tracker.txt', skiprows = 2, unpack = True)
x_pos4, y_pos4 = np.loadtxt('data9 tracker.txt', skiprows = 2, unpack = True)


def rayleigh (r, a):
    return r/(a)*np.exp(-r**2/(2*a))
```

```
def boltzmann (x):
    '''
    k = 3pi*r*η*(2Dt)/(T*t)
    '''
    x = x*10**-12 #1 micrometer^2 = 1*10^-12 meter^2
    k = 3*np.pi*(9.5*10**-7)*0.001*x/(296.5*0.5) #1g/cm-s = 0.001 kg/m-s; 0.95 micrometer = 9.5*10^-7 meter
    return k


#Calculate uncertainty for k
def boltzmann_err(x, u_x):
    '''
    u_r = 0.05 micrometer
    u_n = 0.05 centipoise
    u_T = 0.5K
    u_t = 0.03s
    '''
    x = x*10**-12
    u_x = u_x*10**-12
    u_r = 0.05*10**-7
    u_n = 0.05/1000
    u_T = 0.5
    u_t = 0.03
    rn = (9.5*10**-7)*0.001
    Tt = 296.5*0.5
    u_rn = rn*np.sqrt((u_r/9.5*10**-7)**2+(u_n/0.001)**2)
    u_Tt = Tt*np.sqrt((u_T/296.5)**2+(u_t/0.5)**2)
    rnTt = rn/Tt
```

```
    u_rnTt = rnTt*np.sqrt((u_rn/rn)**2+(u_Tt/Tt)**2)
    a = rnTt*x
    u_a = a*np.sqrt((u_x/x)**2+(u_rnTt/rnTt)**2)
    return 3*np.pi*u_a

#Uncertainty of NA
def avo_err(k, u_k):
    u_a = 1/k*(u_k/k)
    return 8.3145*u_a

def avogadro(k):
    return 8.3145/k

dx1 = np.zeros(len(x_pos1)-1)
dy1 = np.zeros(len(y_pos1)-1)

dr_sq1 = np.zeros(len(dx1))

dx2 = np.zeros(len(x_pos2)-1)
dy2 = np.zeros(len(y_pos2)-1)

dr_sq2 = np.zeros(len(dx2))

dx3 = np.zeros(len(x_pos3)-1)
dy3 = np.zeros(len(y_pos3)-1)

dr_sq3 = np.zeros(len(dx3))
```

```
dx4 = np.zeros(len(x_pos4)-1)
dy4 = np.zeros(len(y_pos4)-1)

dr_sq4 = np.zeros(len(dx4))



#0.1155 micrometer/pixel
for i in range (0, len(x_pos1)-1):
    dx1[i] = (x_pos1[i+1]-x_pos1[i])*0.1155
    dy1[i] = (y_pos1[i+1]-y_pos1[i])*0.1155
    dr_sq1[i] = dx1[i]**2+dy1[i]**2

for i in range (0, len(x_pos2)-1):
    dx2[i] = (x_pos2[i+1]-x_pos2[i])*0.1155
    dy2[i] = (y_pos2[i+1]-y_pos2[i])*0.1155
    dr_sq2[i] = dx2[i]**2+dy2[i]**2

for i in range (0, len(x_pos3)-1):
    dx3[i] = (x_pos3[i+1]-x_pos3[i])*0.1155
    dy3[i] = (y_pos3[i+1]-y_pos3[i])*0.1155
    dr_sq3[i] = dx3[i]**2+dy3[i]**2

for i in range (0, len(x_pos4)-1):
    dx4[i] = (x_pos4[i+1]-x_pos4[i])*0.1155
    dy4[i] = (y_pos4[i+1]-y_pos4[i])*0.1155
    dr_sq4[i] = dx4[i]**2+dy4[i]**2

dr_sq = np.append(dr_sq1, dr_sq2)
```

```
dr_sq = np.append(dr_sq, dr_sq3)
dr_sq = np.append(dr_sq, dr_sq4)
dr_sq = np.sort(dr_sq)
error = np.zeros(len(dr_sq))
for i in range(0, len(error)):
    error[i] = 0.1

dr = np.sqrt(dr_sq)

bins = np.linspace(0, 10, 477)
bin2 = np.linspace(0, 1, 477)

plt.figure(figsize = (10,5))
p, b = np.histogram(dr, bin2, density=True)
weights = np.ones_like(dr)/float(len(dr))
plt.hist(dr, weights=weights)

#Curve fitting
popt , pcov = curve_fit(rayleigh, dr, p, (1), error, True)

#max likelihood
x = 0
uncertainty = []
for i in dr:
    x += i**2
    uncertainty.append((2*i*0.1)**2)
x = x/(2*len(dr))
```

```
max_likeli = boltzmann(x)
ray_est = boltzmann(popt[0])
avo_max = avogadro(max_likeli)
avo_ray = avogadro(ray_est)
print(max_likeli)
print((max_likeli-1.38064852*10**-23)/(1.38064852*10**-23)*100, '%')
print(ray_est)
print((ray_est-1.38064852*10**-23)/(1.38064852*10**-23)*100, '%')
print(avo_max)
print((avo_max-6.0221409*10**23)/(6.0221409*10**23)*100, '%')
print(avo_ray)
print((avo_ray-6.0221409*10**23)/(6.0221409*10**23)*100, '%')

plt.plot(bins, rayleigh(bins, popt), label='Rayleigh Estimation')
plt.plot(bins, rayleigh(bins, x), label='Maximum likelihood estimation')
plt.xlabel("Steps (micrometer)")
plt.ylabel("Probability")
plt.legend()

##########################################################################
#######
u_max_likeli = np.sqrt(sum(uncertainty))/(2*len(dr))
u_x = boltzmann_err(x, u_max_likeli)
u_ray = boltzmann_err(popt[0], np.sqrt(pcov[0]))
print()
print('Errors')
print(u_x)
print(u_ray)
print(avo_err(max_likeli, u_x))
```

```
print(avo_err(ray_est, u_ray))


'''
Output:
2.35830889567778e-23
70.81167737591753 %
1.5461365405172676e-23
11.986252700815395 %
3.525619572244545e+23
-41.45571100396297 %
5.377597503269888e+23
-10.702894658776772 %

Errors
1.8584979778848652e-24
[1.21580549e-24]
2.7784133188899486e+22
[4.22867735e+22]
'''
```