PHY224 Thermal Motion Lab Report

Thursday, the 28th of November, 2019.

Victor Chu (1004004043), Spencer Ki (1003165031)

Abstract:

This experiment attempted to determine Boltzmann's constant by analyzing the thermal motion of fluorescent polystyrene beads in an aqueous solution. Thermal (or Brownian) motion is the random motion of particles in a fluid due to the transfer of kinetic energy between the particles of the fluid as they collide. This motion was observed experimentally by digitally recording the motion of 22 fluorescent particles in a fluid solution at 2 frames per second for 120 frames using a video microscope. The displacements of the particles from their original positions in two dimensions were then extrapolated from the recordings using the *Image Object Tracker* software. Using a two-dimensional generalization of the one-dimensional walk displacement relation, $< r^2 > = 4Dt$ (for the diffusion constant $D = kT/\gamma$ and displacement $r^2 = x^2 + y^2$), the value of Boltzmann's constant was determined to be $1.1139 \times 10^{-23} \pm 9.8411 \times 10^{-22}\, J/K$ using Einstein's relation. The experimentally determined value of Boltzmann's constant, $k_b = 1.3806 \times 10^{-23} J$/K, is within the margins of error of the experimentally determined value.

Introduction:

The purpose of this experiment was to determine Boltzmann's constant by analyzing the random motion of beads in an aqueous solution due to random collisions with water particles. This was determined using the expression $< r^2 > = 4t\left(\frac{kT}{\gamma}\right)$ relating the two-dimensional mean-squared distance of a particle in Brownian motion to Boltzmann's constant.

Methods and Materials:

- Phase Video Microscope (10x ocular objective, 40x objective lens)
- Microscope Slide
- Double Sticky Tape
- Petroleum Jelly
- P200 Micropipette
- Diluted GFP-Polystyrene Bead Solution
- X-Cite Fluorescence Illumination Source

Procedure:

The microscope was switched on and the illumination intensity was set to its maximum value. The phase ring was then set to *Ph1*, the X-Cite fluorescence illumination box was switched on, and the GFP cube was set to Position 2. A slide was then prepared by demarcating a ~2.0 cm wide square space using sticky tape and petroleum jelly. 50 μL of dilute bead solution was then dispensed into the square space using a P200 micropipette. A slide cover was then placed atop the square. The microscope stage was set to its lowest setting using the coarse focus. The prepared slide was then placed onto the microscope stage.

The stage of the microscope was adjusted upwards using the coarse focus until the slide approached the lens closely. The slide was then adjusted on the stage using the stage adjustment knobs until the square of bead solution was roughly centered under the 40x objective lens. From there, the fine focus knob was used until an image of the fluorescent beads was visible on the screen. The image was then manually scanned using the stage adjustment knobs until a small group of discrete beads were seen on the screen. This group was then recorded for one minute at 2 images per second. This data was then exported to the Image Object Tracker for quantitative analysis. This process was repeated nine more times (for a total of ten).

Results:

While the *Image Object Tracker* software provided quantitative measurements for the positions of the beads relative to a fixed axis, further analysis was necessary in order to determine the displacement of beads from frame to frame (from their original position to their position in the next frame). These values were obtained via data manipulation in Python for each axis of motion. The total motion of each bead per frame (the square-root of the sum of the squares of horizontal and vertical motion) was calculated from the resulting displacements. These resultant set of two-dimensional 'step sizes' were saved in a single-dimensional array. This information data can then be used to calculate an estimate of Boltzmann's constant in three different ways. Firstly, Boltzmann's constant is estimated by utilizing a model function representing a Rayleigh distribution is written in Python, then passing the cleaned data through the *curve_fit()* function of Scipy's optimize package, allowing for a parameter representative of the value of *D* to be estimated. Boltzmann's constant can also be extrapolated by using the cleaned data to calculate the maximum likelihood estimator of *D*, $(2Dt)_{est} = \frac{1}{2N}\sum_{i=1}^{N} r_i^2$, in Python. Finally, Boltzmann's constant is estimated by utilizing the orthogonality of the basis vectors of the Cartesian coordinate system, giving the system the property that each axis of motion is independent of each other. This allows for the relationship Brownian motion exhibits in one dimension, $< x^2 >= 2Dt$, to be considered in multiple dimensions by a vector representation of the displacement, $< r^2 > = < x^2 > + < y^2 >$, allowing for the two-dimensional step sizes to be calculated from the relation $< x^2 >= 4Dt$.

Using given physical characteristics of the experiment (temperature, density, viscosity, etc.), the Einstein's relationship $D = kT/\gamma$ was used to calculate Boltzmann's constant, *k*, via the

three aforementioned methods. Obtained values were $6.31 \times 10^{-8} \frac{m^2 kg}{s^2 K}$, $1.36 \times 10^{-7} \frac{m^2 kg}{s^2 K}$, and $1.67 \times 10^{-23} \frac{m^2 kg}{s^2 K}$ respectively. The percentage difference between these values and the known value of $1.38 \times 10^{-23} \frac{J}{K}$ are so large as to be meaningless in the first two cases, while the Boltzmann constant extrapolated by the third method differs from the theoretical value by 19.016% (calculated by the formula $\frac{\|V_1 - V_2\|}{(V_1 + V_2)/2}$)

Error was propagated in the third method by taking the error in the displacements as the standard deviation of the measurements of displacements in each dimension and then propagating this error through each operation leading from the displacement of the beads to Boltzmann's constant. As such, each step size has an error of 1.18×10^{-8}, the calculated value of γ has an error of 4.66×10^{-4}, the Boltzmann estimate from the maximum likelihood estimator has an error of 1.16×10^{-10}, and the Boltzmann estimate from Einstein's relation has an error of 9.8411×10^{-22}. The error for the Boltzmann estimate derived from the Rayleigh function *curve_fit()* could not be determined, due to a lack of transparency of the working of the *curve_fit()* function's calculations.

Discussion:

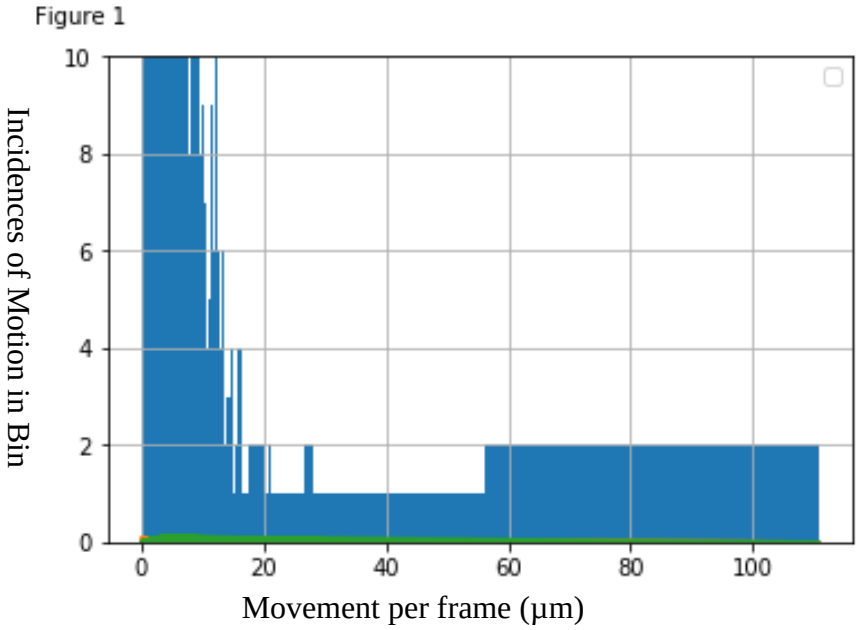
It is interesting to note that only the method using Einstein's relationship in vector form yielded a satisfactory extrapolation of Boltzmann's constant. This is likely due to unreasonable approximations required of the other two methods. In the case of estimating via *curve_fit()*, the "Rayleigh" Python function was fit under the assumption that the output would obey a Gaussian distribution. This assumption was made given that Brownian motion does ideally result in normally distributed measurements in each independent axis, as well as the fact that no observations of the expected output could be obtained experimentally. However, this assumption is tenuous when considered in conjunction with the visibly non-Gaussian spread of the data in the histogram plot, along with statistical anomalies of said data, such as negative variance (a theoretically impossible value). It is interesting to note that the as more displacements were included in the histogram, that it became more and more Gaussian (quantified by a Shapiro-Wilk score closer to one). Limitations in the processing power of computers used to run the Python program, and limitations in the number of data points observed limited the number of points eligible for use however.

The second method of maximum likelihood estimation relied on many of the same statistical axioms as in the first methods (i.e.: the assumption that underlying data is Gaussian), and thus features the same faults. A Chi-squared test between the two faulty methods highlights their relationship yields a test statistic of 8.21, indicating a fairly good fit between the two sets. This is representative of a strong correlation between the two sets. Similarly, a Shapiro-Wilk score of 0.070 for the set of step size data indicates that this function is likely not drawn from a Gaussian parent population.

Conclusion:

The statistical axioms (Gaussian distribution of data) which underpin the calculations of Boltzmann's constant via the formula for a Rayleigh distribution and a maximum likelihood estimator do not fit the apparent nature of the observations of Brownian motion. Thus, these methods yielded very poor estimates of Boltzmann's constant. Only the derivation of the constant from the elementary vector relationship of Einstein's relation allowed a reasonable estimate of $k = 1.67 \times 10^{-23} \pm 9.8411 \times 10^{-22} J/K$; a mere 19.016% different with respect to the known value of 1.38×10^{-23}.

Magnitude of Brownian Motion Per Frame by Microbeads



Note: The *curve_fit()* Rayleigh distribution is plotted in green and the distribution based on the maximum likelihood estimator is in beige.

```
# -*- coding: utf-8 -*-
"""
General Code and Given Constants for the Lab
"""
import numpy as np
import scipy.optimize as opt
import scipy.stats as sp
import matplotlib.pyplot as plt
import os

#propagation of error function
def adsubprop(err_x, err_y): #err_x is the error in the argument x
    err_adsub=np.sqrt(err_x**2+err_y**2)
    return(err_adsub)

def muldivprop(x, y, z, err_x, err_y):
    err_muldiv=z*(np.sqrt((err_x/x)**2+(err_y/y)**2))
    return(err_muldiv)

def exprop(exponent, x, err_x):
    err_exp=exponent*(x**exponent)*(err_x/x)
    return(err_exp)

t = 60 #seconds, 120 frames, 2 frames per second
r_bee = 9.5e-7 #diamater of the bead
T = 296.5
temp_diff = 296.5 - 293.15 #difference from 20 degrees C to ambient temp
eta = (1 - 0.02*(temp_diff))*0.1 #in pascal seconds, converted from centipoise
gamma = 6*np.pi*eta*r_bee
"""
Code to Clean and Compile Data
"""
#importing the data collected from the observations of motion.

filelist = []

i = 0
longest = 0
#a placeholder array for data, shape of the number of entries
#we want by the number of measurements taken

#makes a list of the paths of each file in the desired directory
for root, dirs, files in os.walk(r"C:\Users\alexc\Documents\PHY224\Experiments\Bees\Positions"):
    for file in files:
        #takes only files of type txt
        if file.endswith(".txt"):
            i=i+1
            filelist.append(os.path.join(root, file))

xdata = [] #displacement in x in pixels
ydata = [] #displacement in y in pixels

for i in range(len(filelist)):
    xdata.append(np.loadtxt(filelist[i], skiprows = 2, usecols = 0))
    ydata.append(np.loadtxt(filelist[i], skiprows = 2, usecols = 1))

xdiffs = np.zeros((len(xdata), 119))
ydiffs = np.zeros((len(xdata), 119))*0.1155e-6 #correction factor, pixels to m

for a in range(len(xdata)):
    for b in range(len(xdata[a])-1):
        xdiffs[a][b]=xdata[a][b+1]-xdata[a][b]
        ydiffs[a][b]=xdata[a][b+1]-xdata[a][b]
```

```
xdiff_reshaped = xdiffs.reshape(1, 2618) #2618 = 22*119
ydiff_reshaped = ydiffs.reshape(1, 2618) #done to flatten array

xdiff = np.round(xdiff_reshaped, 1)
ydiff = np.round(ydiff_reshaped, 1)

xdiff_corrected = xdiff*0.1155e-6 #correction factor; pixels to m
ydiff_corrected = ydiff*0.1155e-6 #correction factor; pixels to m


"""
Code for Histogram, Rayleigh, and Maximum Likelihood Calculations
"""

step_size = np.sqrt(xdiff[0]**2 + ydiff[0]**2)
#step_size = np.round(step_size, decimals = 2)

def rayleigh(r, D, t):
    return (r/(2*D*t))*np.exp(-(r**2)/(4*D*t))

def maximum_likelihood(r):
    return np.sum(r**2)/(2*len(r))

#identifying the unique number of incidences of radiation in the time interval
unique_values = np.unique(step_size)
av_unique_values = np.average(step_size) #expected average emission
std_unique_values = np.std(step_size)

#defining the normal distribution
normal = sp.norm.pdf(step_size, loc=av_unique_values, scale=std_unique_values)
p_opt, p_cov = opt.curve_fit(rayleigh, step_size, normal)

rayleigh_k = p_opt[0]*gamma/T

#Histogram of incidence of counts per time interval
plt.figure(0)
plt.hist(step_size, bins=unique_values)
plt.plot(step_size, normal, label = 'Normal Distribution')
plt.plot(step_size, (step_size/(maximum_likelihood(step_size)))*np.exp\
         (-(step_size**2)/(2*maximum_likelihood(step_size))), label =\
         'Rayleigh Distribution')
plt.xlabel('Displacement (pixels)')
plt.ylabel('Counts')
plt.title('Histogram of Random Displacements of Particles due to Thermal \
          Interactions')
plt.figtext(0.05, 0.93, "Figure 1")
plt.legend()
plt.grid()
plt.savefig('ChuKi Random Motion Histogram.pdf')
plt.ylim(0,100)

"""
Code for Einstein Relationship Calculations
"""
meansquarex = np.average(xdiff_corrected**2)
meansquarey = np.average(ydiff_corrected**2)

einstein_k = (meansquarex + meansquarey)*gamma/(4*t*T)

"""
Code for Propagation of Error
"""
```

```
xdata_er = np.std(xdiff_corrected**2)
ydata_er = np.std(ydiff_corrected**2)

err_x_sq = muldivprop(np.average(xdiff_corrected), \
                      np.average(xdiff_corrected), xdata_er, xdata_er, np.average(xdiff_corrected)**2)
err_y_sq = muldivprop(np.average(ydiff_corrected),\
                      np.average(ydiff_corrected), \
                      ydata_er, ydata_er, np.average(ydiff_corrected)**2)

err_radius = 1e-10 #0.1 um
err_visc = 0.005 #g/m*s
err_temp = 0.5 #k

err_sumx = np.std(xdiff_corrected**2)
err_sumy = np.std(ydiff_corrected**2)

#error in the summ of the average, no error from division by scalar
for c in range(2618):
    err_sumx = adsubprop(err_sumx, xdata_er)
    err_sumy = adsubprop(err_sumy, ydata_er)

err_avx = muldivprop(np.sum(xdiff_corrected), 2618, err_sumx, 0, np.average(xdiff_corrected))
err_avy = muldivprop(np.sum(ydiff_corrected), 2618, err_sumy, 0, np.average(ydiff_corrected))

err_x_sq = muldivprop(xdiff_corrected, xdiff_corrected, err_avx, err_avx, xdiff_corrected**2)
err_y_sq = muldivprop(ydiff_corrected, ydiff_corrected, err_avy, err_avy, ydiff_corrected**2)

err_msqdisp = adsubprop(err_avx, err_avy)

err_gamma = muldivprop(eta, r_bee, err_visc, err_radius, eta*r_bee)

#error in multiplying displacement by gamma;
err_1 = muldivprop((meansquarex + meansquarey), gamma, err_msqdisp, err_gamma, einstein_k*4*t*T)
#error in dividing displacement*drag/temperature
err_2 = muldivprop((einstein_k*4*t*T), T, err_1, err_temp, einstein_k*4*t)
err_3 = muldivprop((einstein_k*4*t), t, err_2, 0, einstein_k*4)
err_4 = muldivprop((einstein_k*4), 4, err_3, 0, einstein_k)

tep_er = exprop(2, av_unique_values, xdata_er)
max_like_er = np.sqrt(len(step_size)*step_er**2)/(2*len(step_size))
```