Pierre Ishak (1007897457), Joaquin Arcilla (1007930820)

Motion in Fluids Lab

1. Introduction

In this experiment, we aim to study the fluid mechanics of terminal velocity for spherical objects in different flow types. To simulate low Reynolds number, Teflon beads of various sizes are used and dropped in glycerine. For high Reynolds number simulations, Nylon beads are utilized and dropped in water. The movement of each bead is captured using cameras and video tracking software to determine its terminal velocity. The aim is to evaluate the accuracy of various models for terminal velocity based on bead radius in both high and low Reynolds number flow types.

The Reynolds number is a crucial and powerful concept in the realm of fluid dynamics and plays a significant role in the behaviour of fluid flow. More specifically it is a dimensionless number that describes the ratio of inertial and viscous forces in fluid flow. It was first introduced and created by Osborne Reynolds in 1883 as a tool to classify and understand different types of fluid flow. The two main ones being laminar and turbulent. Intuitively, the Reynolds number represents the balance between the forces that keep fluid particles in moving in straight lines vs the forces that cause the particles to change direction and interact with each other.

Inertial forces can be defined through Newton's Second law of motion (1). While Viscous forces can be defined by an object's ability to resist displacement against the direction of fluid flow (2). It is a function of fluid viscosity μ, the cross-sectional area between the fluid and the object A, the velocity v, and the characteristic length l of the object. In this lab the characteristic length is defined by the volume of the object compared to surface area (3).

$$F_i = \rho V a = \frac{\rho l A v}{t} \quad (1) \qquad\qquad F_v = \frac{\mu A v}{l} \quad (2)$$

$$l = \frac{Volume_{body}}{A_{surface}} \quad (3) \Rightarrow l_{sphere} = \frac{r}{3}$$

The ratio of the inertial to viscous forces is the infamous Reynolds number (4). A high Reynolds number means that inertial forces overcome the viscous. While a low Reynolds number describe fluids where viscous forces are impeding on inertial forces.

$$R_e = \frac{F_i}{F_v} = \frac{\rho l v}{\mu} \quad (4)$$

In our investigation of terminal velocity (v_{term}) of the spheres occurs when the inertial and viscous forces are in balance ($F_i = F_v$). The inertial force produced by gravity ($F_i = F_g = mg$) would be equal to the viscous force of drag ($F_v = F_d$). When this happens, the sphere is no longer able to accelerate further. At high Reynold numbers F_d is modeled as (5), where C_d is an empirically calculated value.

$$F_d = \frac{1}{2}\rho C_d A v^2 \quad (5)$$

Substituting $F_d = mg$, solving the differential equation in terms of v_{term} you are able to conclude that the terminal velocity at high Reynold numbers is equal to the square root of the

radius. At low Reynold numbers, drag force is described at $F_d = 6\pi\mu vr$, and by following a similar process, the terminal velocity is equal to the square of the radius.

Low Reynolds: $v_{term} \propto r^2$ High Reynold: $v_{term} \propto r^{1/2}$

The constant of proportionality C_d can be modeled by $\frac{2g}{9\mu}(\rho_{object} - \rho_{fluid})$ [1]. Additionally, when an object falls in fluids close to a boundary layer such as a wall an effect is formed called the no-slip condition. This condition causes objects falling near the edge of the walls have a greater viscous effect compared to if they were dropped exactly in the center. This causes the particles in center to have greater terminal velocities [2]. To correct for the discrepancies of the terminal velocities of the beads Equation 6 was presented in the lab. The equation corrects for the different diameters of the beads (D) and the wall length (d).

$$v_{corr} = \frac{v_{term,mean}}{1 - 2.104\left(\frac{d}{D}\right) + 2.089\left(\frac{d}{D}\right)^2} \quad (6)$$

2. Procedure

2.1 Equipment

- Digital calipers ± 0.02mm [3]
- Tweezers
- Camera chamber (including camera)
- Labview code "Motion through Fluids" application
- Spherical Teflon and Nylon beads of varying sizes (Table 2 shows all dimensions)
- Rectangular clear containers of Water and Glycerol

2.2 Apparatus

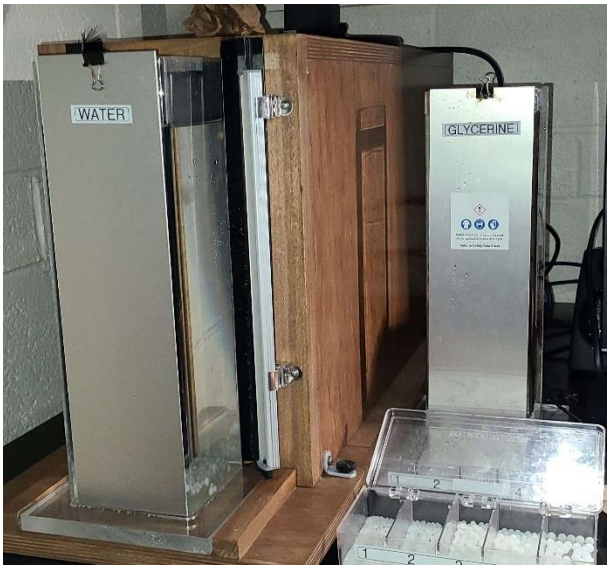

Figure 1: The Experimental Apparatus showing the equipment we used, including the beads, the two containers of fluid, and the camera box.

2.3 Experimental Method

The experimental method was followed according to the lab manual [4].

The Teflon and Nylon beads were measured and recorded using the digital calipers 5 times for each respective ball size. The beads were also measured accurately by ensuring that calipers weren't excessively tight, due the nature of the polymers the beads were made from, it was possible to deform them and get a wrong measurement. The width of the containers for the glycerol and the water were also measured.

Then the lab computer was started, and the LabView "Motion of Fluids" tracking software was opened. In order to set the frame rate, the time it took for each respective bead size to reach the bottom of the container was noted. For each trial the frame count and framerate were selected. Table 3&4 shows all the set frame counts we selected.

The glycerol container was placed in the camera chamber. Three trials were conducted for each bead size (for a total of 15 trials) where the beads were picked and suspended with tweezers after they were submerged under the glycerol. The button to start the video capture was clicked and a series of beeps commenced. Once the longest beep happened the bead was dropped to fall freely in the glycerol and the recording started and ended momentarily after the bead hit the bottom of the container.

This exact process was repeated, the only difference being the glycerol was replaced with the water container. Additionally for the water drops five trials were done for each respective bead size (for a total of 25 trials).

3. Results and Analysis

3.1 Uncertainties, Error propagation and Sources of Error

When collecting the data, measured values such as diameter and velocity have uncertainties. For the diameter, the uncertainty is from the digital calipers which only measure to a certainty of 0.02mm. For the velocity, because there were so many trials collected during the experiment, most of the uncertainty is formed from the differences between trials of the same type. So, the uncertainty from the velocity calculations is the standard deviation of the trails calculated using the following equation:

$$\sigma_x = \frac{1}{\sqrt{\sum \frac{1}{\Delta x^2}}} \quad (7)$$

For our calculations, we performed error propagation using our python code. This code implemented the following equations for sum calculations and product calculations respectively:

$$\Delta z = \sqrt{(\Delta x)^2 + (\Delta y)^2} \quad (8) \qquad \frac{\Delta z}{z} = \sqrt{\left(\frac{\Delta x}{x}\right)^2 + \left(\frac{\Delta y}{y}\right)^2} \quad (9)$$

For this experiment, there were many possible sources of error. One was that there was a great diversity in diameters for each size of Nylon and Teflon balls. This changes the Reynolds number for each case, as one of the variables in the equation (4) is dependent on the radius of the ball. One should try to get balls which are closer in size so that each trial is more like the ones in its group.

Another source of error is the fact that the containers used to conduct the drop-test were already filled with Nylon and Teflon balls from previous experiments. While impedance on the velocity caused by the walls were accounted for using equation (6), any turbulence caused by the surface balls are not accounted for. This may have changed the 'real' velocity values of the experiment.

A more technical error is the fact that the device that held our containers only had working lights on side of the apparatus as seen on Figure 2. This may have hindered the camera and LabVIEW's ability to track the Nylon and Teflon balls accurately. This can be seen in our raw data where the velocities of the balls don't remain constant and fluctuates.

Figure 2: The container for the water, with light only coming through the left side and no light coming through from the right.

3.2 Physical Reference Data

- Glycerol Density [5]: $1.26 \pm 0.005 \frac{g}{cm^3}$
- Glycerol Viscosity [5]: $9.34 \pm 0.005 \frac{g}{cm\,s}$
- Water Density [6]: $0.995 \pm 0.0005 \frac{g}{cm^3}$
- Water Viscosity [6]: $0.0089 \pm 0.00005 \frac{g}{cm\,s}$
- Teflon Density [7]: $2.2 \pm 0.05 \frac{g}{cm^3}$
- Nylon Density [8]: $1.12 \pm 0.005 \frac{g}{cm^3}$

Note: Physical quantities were checked with uncertainties using the sources provided at 25 degrees Celsius.

3.3 Bead Diameters

The diameter of the beads of each respective size and material are recorded in Table 2 in the appendix A.1.

3.4 Reynolds Number

The Terminal velocities of the beads in each trial was measured from the time position data provided through the LabView tracking software. The terminal velocities were agreed upon to be once the velocities calculated were stable. To account for the velocity fluctuations an average of 10 velocities after it had reached terminal were averaged.

Then to account for the no-slip condition caused by the walls of the container, Equation 6 was used where the respective container width for the water and glycerol was 99.75 mm and 99.22 mm.

Lastly, the Reynolds number was calculated using Equation 4 with constants like density and viscosity being provided from section 4.2 and the characteristic length being r/3 for the spherical beads which was derived in Equation 3.

Bead	Diameter (mm)	Terminal Velocity (mm/s)	Corrected Velocity (mm/s)	Reynolds Number
Nylon 1	2.34±0.01	76.02±0.72	79.87±0.76	0.0004±1.039
Nylon 2	3.14±0.01	93.08±4.59	99.46±4.93	0.0006±6.965
Nylon 3	3.82±0.09	110.75±1.77	120.07±3.47	0.0009±0.000
Nylon 4	4.69±0.06	125.83±11.40	138.95±12.73	0.0032±5.782
Nylon 5	6.29±0.01	147.79±4.65	168.78±5.31	0.0074±0.001
Teflon 1	1.57±0.03	10.94±0.04	11.31±0.21	2.0900±0.022
Teflon 2	2.23±0.19	10.93±0.23	11.46±0.98	3.7048±0.186
Teflon 3	2.79±0.47	12.77±0.20	13.55±2.31	5.5933±0.246
Teflon 4	4.69±0.02	27.28±0.45	30.14±0.52	10.872±1.008
Teflon 5	6.28±0.05	46.09±0.24	52.67±0.55	17.666±0.562

Table 1: Measured and Corrected Terminal Velocity Values with Calculated Reynolds Number for each Bead.

3.5 Terminal Velocity and Bead Radius

Figures 2 and 3 demonstrate the proposed models of spherical terminal velocities in high and low Reynolds fluids as a function of bead radius against our actual gathered data points. The data was curve fitted using Python's NumPy Library and residual plots are provide below as a goodness of fit criteria, as well as a discussion of our data from an R^2 and χ^2.

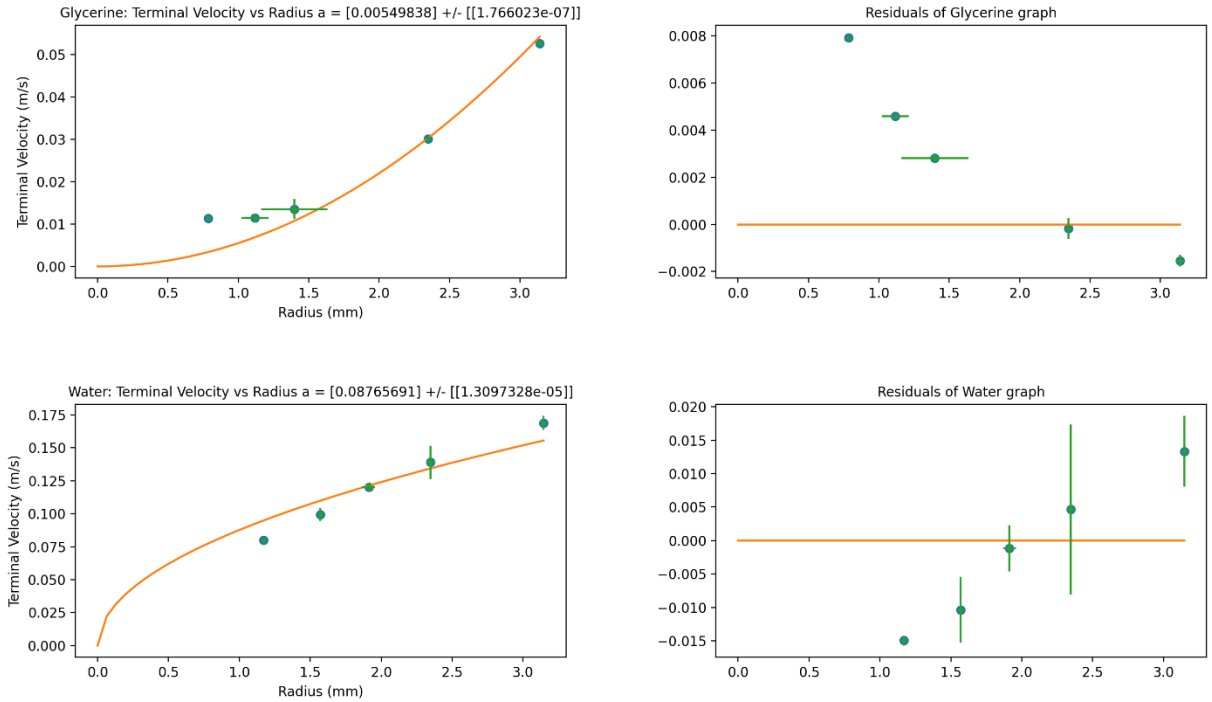

Figure 2: Beads in Glycerine and Water Corrected Terminal Velocities Graph

3.51 Low Reynolds number

The Teflon beads falling in glycerin is considered to be a low Reynolds number situation where the v_{term} can be modeled as equal to $v_{term} = C_d r^2$. We can model the theoretical C_d value using Equation mentioned in the Introduction ($\frac{2g}{9\mu}(\rho_{object} - \rho_{fluid})$) which would be equal to 2.2 ± 0.1 $\frac{1}{mm\ s}$ in this case. Experimentally, the value we found for C_d was 5.49 ± 1.76 x 10^{-4} $\frac{1}{mm\ s}$. These values are not very close to each other but still remain within the same order of magnitude. This difference in C_d could be related to the flow not being fully laminar in glycerin since we saw the fluctuations in the velocities. This could be due to the spheres or glycerin having imperfections impeding the flow described.

The glycerin's curve fitting yields an R^2 was 0.93 (using the equation in Appendix B) which is a good value as an R2 value greater than 0.9 is considered good. This seems to indicate what is visually apparent, that the curve of best fit, fits the data well, meaning the theory matches the experiment well. However, the χ^2 value (calculated using the equation in Appendix B) was 495 which is not close to the desired value. This number could be an error with the code computing this value.

3.52 High Reynolds number

The Nylon beads falling through water is considered to be a high Reynolds number scenario where the v_{term} can be modeled as equal to $v_{term} = C_d r^{1/2}$. In high Reynold number situations modeling the C_d doesn't work due to the turbulent flow taking over once the Reynolds number hits around 24 [1]. As well as in theory "Experimental data place the crossover near Re ~ 5, at which point cd ~ 2", between low and high-speed flows.

For this experiment, the χ^2 and R^2 values (calculated using the equations in Appendix B) are not as good as the Low Reynolds number ones. The χ^2 value for this case was 132.7 while the R^2 value was –137.9. These values don't make sense since the curve of best fit, fits the data visually based on Figure 2. The residuals as well show that the curve fits within the uncertainties.

4. Conclusion

Based on our data, the theoretical approximations stating that a low Reynolds number would have a terminal velocity proportional to r^2, is a correct approximation, since the data for the Teflon balls was able to fit into a curve of that form. Similarly, the approximation stating that a high Reynolds number would have a terminal velocity proportional to $r^{1/2}$ is also a correct approximation since the Nylon ball data was able to fit a curve of the theoretical form. This shows that the Reynolds number is a good indicator for the study of fluid mechanics and physics.

References

[1] "Art of approximation," 2010. http://web.mit.edu/6.055/old/S2008/notes/apr30.pdf.

[2] "EngArc - L - No-Slip Condition," *www.engineeringarchives.com*. http://www.engineeringarchives.com/les_fm_noslip.html\

[3] "Digital Caliper H-7352 - Uline," *www.uline.ca*. https://www.uline.ca/Product/Detail/H-7352/Tools/Digital-Caliper

[4] R. Serbanescu and L. Avramidis, “Motion in fluids,” 2019. https://q.utoronto.ca/courses/290339/pages/experiment-list-includes-links-to-all-manuals?module_item_id=4359319.

[5] “Glycerol.” https://pubchem.ncbi.nlm.nih.gov/compound/Glycerol.

[6] “Water.” https://pubchem.ncbi.nlm.nih.gov/compound/Water.

[7] “Tetrafluoroethylene.” https://pubchem.ncbi.nlm.nih.gov/compound/tetrafluoroethylene.

[8] “The online materials information resource.” https://www.matweb.com/search/DataSheet.aspx?MatGUID=8d78f3cfcb6f49d595896ce6ce6a2ef1&ckck=1.

Appendix

A Data Tables:

A.1 Bead Diameters

	Diameter Measurements (mm), ± 0.02mm					
Bead	*Trial 1*	*Trial 2*	*Trial 3*	*Trial 4*	*Trial 5*	Mean Diameter
Nylon 1	2.34	2.34	2.34	2.26	2.37	2.34±0.01
Nylon 2	3.15	3.13	3.12	3.13	3.12	3.14±0.01
Nylon 3	3.89	3.76	3.93	3.91	3.86	3.82±0.09
Nylon 4	4.74	4.65	4.63	4.74	4.72	4.69±0.06
Nylon 5	6.29	6.29	6.28	6.28	6.30	6.29±0.01
Teflon 1	1.59	1.55	1.57	1.56	1.57	1.57±0.03
Teflon 2	2.37	2.10	2.21	2.18	2.21	2.23±0.19
Teflon 3	3.13	2.46	2.84	2.46	3.18	2.79±0.47
Teflon 4	4.68	4.71	4.73	4.67	4.71	4.69±0.02
Teflon 5	6.32	6.24	6.29	6.25	6.27	6.28±0.05

Table 2: Diameter of the beads measurement trials, and their average.

A.2 Frame Counts

Total Number of Frames			
Nylon 1	80	Teflon 1	1000
Nylon 2	70	Teflon 2	740
Nylon 3	60	Teflon 3	420
Nylon 4	60	Teflon 4	220
Nylon 5	60	Teflon 5	140

Table 3&4: Set Total Frame Rate counts for each respective bead size.

B Formulae:

$$\chi^2 = \sum_{i=1}^{N} \frac{[y_i - f(x_i)]^2}{\sigma_{y_i}^2}$$

Formula 1: $\chi 2$ squared Formula

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

Formula 2: R^2 Formula

C python code:

```python
#Imports
import numpy as np
import matplotlib.pyplot as plt
from pylab import loadtxt
import scipy.optimize as optimize
from scipy.stats import chisquare

# File Constants:
gly = "Glycerine/"
wat = "Water/"
vel = "Velocities/"
rad = "Radii/"

# Other measurements:
gly_height = 99.22
wat_height = 99.75

#Uncertainties
def uncertainty_sum(dx, dy):
    return np.sqrt(dx**2 + dy**2)

def uncertainty_prod(x, dx, y, dy, z):
    return np.sqrt((dx/x)**2 + (dy/y)**2) * z

def import_data(filename,skip_rows=2, cols=(0,1)):
    data=loadtxt(filename, usecols=cols, skiprows=skip_rows, unpack=True)

    return data[0], data[1]

def mean_same_uncern(list):
    return sum(list)/len(list)

def standard_deviation(value, list):
    return np.sqrt((1/(len(list)-1))*sum((value - list)**2))

def mean_diff_uncern(list, uncern_list):
    return sum(list/uncern_list**2) / sum(1/uncern_list**2)

def chi_squared(func, a, x_data, y_data, dy_data): # Goodness of Fit
    fx = func(x_data, a)
    N = len(x_data)
    sum = 0

    for i in range(N):
        sum += (((y_data[i] - fx[i])**2) / (dy_data[i]**2))

    v = N-2

    return sum / v

def r_squared(y_data, y_graph):
    y_mean = np.mean(y_data)
    ss_total = np.sum((y_data - y_mean)**2)
    ss_residual = np.sum((y_data - y_graph)**2)
    r_squared = 1 - (ss_residual / ss_total)
    return r_squared

def residuals(func, a, x, y): # Residuals
    fx = func(x, a)
    resid = y - fx

    return x, resid


# Functions for the lab
def calc_velo(time, position): # Calculate velocity from raw data
    velocities = []
    for i in range(1, len(time)): # iterate through positions and divide by time
        value = (position[i] - position[i-1])/(time[i]-time[i-1])
        if value != 0:
            velocities.append(value)

    return velocities

def curate_list(list): # Take data after terminal velocity
    list = list[13: len(list)-4]
    #print(list)
    return list

def terminal_velo(time,position): # Calculate terminal velocities
    velocities = calc_velo(time, position)
    velocities = curate_list(velocities)
    terminal_velo = mean_same_uncern(velocities) # find the terminal velocity
    terminal_velo_std_dev = standard_deviation(terminal_velo, velocities) # standard devation of the mean

    return terminal_velo, terminal_velo_std_dev

def Calculate_Velocities(): # Full function for calculating terminal velocities
    # Calculate Velocities from raw data
    gly_calc = False
    file = gly + "Gly_5_3.txt"
    if gly_calc == True:
        gly1_time, gly1_pos = import_data(file, 2)
        terminal_velocity, term_vel_std_dev = terminal_velo(gly1_time, gly1_pos)
        print(f'{terminal_velocity}  +/- {term_vel_std_dev}')
```

```
    wat_calc = False
    file = wat + "Wat_3_1.txt"
    if wat_calc == True:
        files = [wat + "Wat_5_1.txt",
                 wat + "Wat_5_2.txt",
                 wat + "Wat_5_3.txt",
                 wat + "Wat_5_4.txt",
                 wat + "Wat_5_5.txt"]

        for file in files:
            wat_time, wat_pos = import_data(file, 2)
            terminal_velocity, term_vel_std_dev = terminal_velo(wat_time, wat_pos)
            print(f'{file}: {terminal_velocity}  +/- {term_vel_std_dev}')

    # Final values are put into .txt files and used below

def Load_Velocities(files):
    # Load calculated velocities
    terminal_velocities = []
    terminal_velocities_uncern = []

    for file in files:
        # Get average and standard deviation
        term_velo, term_velo_uncern = import_data(file, 1)
        term_velo = np.array(term_velo)/1000
        term_velo_uncern = np.array(term_velo_uncern)/1000
        avg_term_velo = mean_diff_uncern(term_velo, term_velo_uncern)
        avg_term_velo_uncern = standard_deviation(avg_term_velo, term_velo)

        #append to the list of velocities
        terminal_velocities.append(avg_term_velo)
        terminal_velocities_uncern.append(avg_term_velo_uncern)

    return terminal_velocities, terminal_velocities_uncern

def Calculate_Radius(files):
    # Load Radii
    radii = []
    radii_uncern = []

    for file in files:
        # Get average and standard deviation
        diameters = import_data(file, 1, (0))
        radiuss = np.array(diameters)/2
        avg_radius = mean_same_uncern(radiuss)
        avg_radius_uncern = standard_deviation(avg_radius, radiuss)

        # Append to the list of radii
        radii.append(avg_radius)
        radii_uncern.append(avg_radius_uncern)

    return radii, radii_uncern

def gly_fit_func(r, a):
    r = np.array(r)
    return a*r**2

def wat_fit_func(r, a):
    r = np.array(r)
    return a*r**(1/2)

def calc_curve_of_fit(func, xdata, ydata, init_guess):
    a, a_uncern = optimize.curve_fit(func, xdata, ydata, p0=init_guess)
    x_graph = np.linspace(0, xdata[len(xdata)-1])
    y_graph = func(x_graph, a)

    return x_graph, y_graph, a, a_uncern

def wall_correciton(vm, r, D):
    # turn radius to diameter
    d = r*2
    return vm / (1 - 2.104*(d/D) + 2.089*(d/D)**2)

def wall_correction_uncertainty(vm, dvm, r, dr, D, z):
    # function: vm / (d/D)
    d = r*2 # turn radius to diameter
    dd = dr*2
    dD = 0.02
    doverD = d/D
    doverD_uncern = uncertainty_prod(d, dd, D, dD, doverD)
    correction_uncern = uncertainty_prod(vm, dvm, doverD, doverD_uncern, z)

    return correction_uncern

# Calculate terminal velocities and Radii
gly_velo_files = [vel + "gly_1.txt",
                  vel + "gly_2.txt",
                  vel + "gly_3.txt",
                  vel + "gly_4.txt",
                  vel + "gly_5.txt"]
gly__radius_files = [rad + "gly_1.txt",
                  rad + "gly_2.txt",
                  rad + "gly_3.txt",
                  rad + "gly_4.txt",
                  rad + "gly_5.txt"]
wat_velo_files = [vel + "wat_1.txt",
```

```
            vel + "wat_2.txt",
            vel + "wat_3.txt",
            vel + "wat_4.txt",
            vel + "wat_5.txt"]
wat__radius_files = [rad + "wat_1.txt",
            rad + "wat_2.txt",
            rad + "wat_3.txt",
            rad + "wat_4.txt",
            rad + "wat_5.txt"]

gly_radii, gly_radii_uncern = Calculate_Radius(gly__radius_files)
gly_terminal_velocities, gly_terminal_velocities_uncern = Load_Velocities(gly_velo_files)
wat_radii, wat_radii_uncern = Calculate_Radius(wat__radius_files)
wat_terminal_velocities, wat_terminal_velocities_uncern = Load_Velocities(wat_velo_files)

### No Correction:

# Curve fitting
gly_x_no_corr, gly_y_no_corr, gly_a_no_corr, gly_a_no_corr_uncern = calc_curve_of_fit(gly_fit_func, gly_radii, gly_terminal_velocities, 0.5)
wat_x_no_corr, wat_y_no_corr, wat_a_no_corr, wat_a_no_corr_uncern = calc_curve_of_fit(wat_fit_func, wat_radii, wat_terminal_velocities, 0.5)

# Residuals
gly_resid_x_no_corr, gly_resid_y_no_corr = residuals(gly_fit_func, gly_a_no_corr, gly_radii, gly_terminal_velocities)
wat_resid_x_no_corr, wat_resid_y_no_corr = residuals(wat_fit_func, wat_a_no_corr, wat_radii, wat_terminal_velocities)
y_zeroline = [0,0]
gly_x_zeroline = [0, gly_resid_x_no_corr[len(gly_resid_x_no_corr)-1]]
wat_x_zeroline = [0, wat_resid_x_no_corr[len(wat_resid_x_no_corr)-1]]

# Chi squared and R squared:
gly_chi_no_corr = chi_squared(gly_fit_func, gly_a_no_corr, np.array(gly_radii), np.array(gly_terminal_velocities), np.array(gly_terminal_velocities_uncern))
wat_chi_no_corr = chi_squared(wat_fit_func, wat_a_no_corr, np.array(wat_radii), np.array(wat_terminal_velocities), np.array(wat_terminal_velocities_uncern))

# Graphs without correction:
#Plot 1 Glycerine Graph without correction
plt.title("Glycerine and Water Graphs with no wall correction")
plt.subplot(2, 2, 1) # Set up graph
plt.plot(gly_radii, gly_terminal_velocities, 'o') # Raw points
plt.plot(gly_x_no_corr, gly_y_no_corr) # Curve of best fit
plt.errorbar(gly_radii, gly_terminal_velocities, xerr=gly_radii_uncern, yerr=gly_terminal_velocities_uncern, fmt=".") #error bars
plt.xlabel("Radius (mm)") # Labels
plt.ylabel("Terminal Velocity (m/s)")
plt.title(f"Glycerine: Terminal Velocity vs Radius a = {gly_a_no_corr} +/- {gly_a_no_corr_uncern}", fontsize=10) # Title
plt.tight_layout()

#Plot 2 residuals
plt.subplot(2, 2, 2)
plt.plot(gly_resid_x_no_corr, gly_resid_y_no_corr, 'o')
plt.plot(gly_x_zeroline, y_zeroline)
plt.errorbar(gly_resid_x_no_corr, gly_resid_y_no_corr, xerr=gly_radii_uncern, yerr=gly_terminal_velocities_uncern, fmt=".") #error bars
plt.title("Residuals of Glycerine graph", fontsize=10)
plt.tight_layout()

#Plot 3 Water Graph without correction
plt.subplot(2, 2, 3) # Set up graph
plt.plot(wat_radii, wat_terminal_velocities, 'o') # Raw points
plt.plot(wat_x_no_corr, wat_y_no_corr) # Curve of best fit
plt.errorbar(wat_radii, wat_terminal_velocities, xerr=wat_radii_uncern, yerr=wat_terminal_velocities_uncern, fmt=".") #error bars
plt.xlabel("Radius (mm)") # Labels
plt.ylabel("Terminal Velocity (m/s)")
plt.title(f"Water: Terminal Velocity vs Radius a = {wat_a_no_corr} +/- {wat_a_no_corr_uncern}", fontsize=10) # Title
plt.tight_layout()

#Plot 4 residuals
plt.subplot(2, 2, 4)
plt.plot(wat_resid_x_no_corr, wat_resid_y_no_corr, 'o')
plt.plot(wat_x_zeroline, y_zeroline)
plt.errorbar(wat_resid_x_no_corr, wat_resid_y_no_corr, xerr=wat_radii_uncern, yerr=wat_terminal_velocities_uncern, fmt=".") #error bars
plt.title("Residuals of Water graph", fontsize=10)
plt.tight_layout()
#plt.show()


### With Correction:
gly_terminal_velocities_with_corr = wall_correciton(np.array(gly_terminal_velocities), np.array(gly_radii), gly_height)
wat_terminal_velocities_with_corr = wall_correciton(np.array(wat_terminal_velocities), np.array(wat_radii), wat_height)

# Propogating uncertainty
gly_terminal_velocities_with_corr_uncern = wall_correction_uncertainty(np.array(gly_terminal_velocities), np.array(gly_terminal_velocities_uncern), np.array(
wat_terminal_velocities_with_corr_uncern = wall_correction_uncertainty(np.array(wat_terminal_velocities), np.array(wat_terminal_velocities_uncern), np.array(

# Curve Fitting
gly_x_with_corr, gly_y_with_corr, gly_a_with_corr, gly_a_with_corr_uncern = calc_curve_of_fit(gly_fit_func, gly_radii, gly_terminal_velocities_with_corr, 0.5
wat_x_with_corr, wat_y_with_corr, wat_a_with_corr, wat_a_with_corr_uncern = calc_curve_of_fit(wat_fit_func, wat_radii, wat_terminal_velocities_with_corr, 0.5

# Goodness of fit:
gly_chi_with_corr = chi_squared(gly_fit_func, gly_a_with_corr, np.array(gly_radii), np.array(gly_terminal_velocities_with_corr), np.array(gly_terminal_veloci
wat_chi_with_corr = chi_squared(wat_fit_func, wat_a_with_corr, np.array(wat_radii), np.array(wat_terminal_velocities_with_corr), np.array(wat_terminal_veloci

gly_r_squared = r_squared(gly_terminal_velocities_with_corr, gly_fit_func(gly_radii, gly_a_with_corr))
wat_r_squared = r_squared(wat_terminal_velocities_with_corr, gly_fit_func(wat_radii, wat_a_with_corr))

print(f'Glycerine: Chi^2: {gly_chi_with_corr}, R^2: {gly_r_squared}')
print(f'Water: Chi^2: {wat_chi_with_corr}, R^2: {wat_r_squared}')

# Residuals
gly_resid_x_with_corr, gly_resid_y_with_corr = residuals(gly_fit_func, gly_a_with_corr, gly_radii, gly_terminal_velocities_with_corr)
```

```
wat_resid_x_with_corr, wat_resid_y_with_corr = residuals(wat_fit_func, wat_a_with_corr, wat_radii, wat_terminal_velocities_with_corr)
gly_x_zeroline = [0, gly_resid_x_no_corr[len(gly_resid_x_with_corr)-1]]
wat_x_zeroline = [0, wat_resid_x_no_corr[len(wat_resid_x_with_corr)-1]]

# Graph with correction:
#Plot 1 Glycerine Graph with correction
plt.title("Glycerine and Water Graphs with wall correction")
plt.subplot(2, 2, 1) # Set up graph
plt.plot(gly_radii, gly_terminal_velocities_with_corr, 'o') # Raw points
plt.plot(gly_x_with_corr, gly_y_with_corr) # Curve of best fit
plt.errorbar(gly_radii, gly_terminal_velocities_with_corr, xerr=gly_radii_uncern, yerr=gly_terminal_velocities_with_corr_uncern, fmt=".") #error bars
plt.xlabel("Radius (mm)") # Labels
plt.ylabel("Terminal Velocity (m/s)")
plt.title(f"Glycerine: Terminal Velocity vs Radius a = {gly_a_with_corr} +/- {gly_a_with_corr_uncern}", fontsize=10) # Title
plt.tight_layout()

#Plot 2 residuals
plt.subplot(2, 2, 2)
plt.plot(gly_resid_x_with_corr, gly_resid_y_with_corr, 'o')
plt.plot(gly_x_zeroline, y_zeroline)
plt.errorbar(gly_resid_x_with_corr, gly_resid_y_with_corr, xerr=gly_radii_uncern, yerr=gly_terminal_velocities_uncern, fmt=".") #error bars
plt.title("Residuals of Glycerine graph", fontsize=10)
plt.tight_layout()

#Plot 3 Water Graph without correction
plt.subplot(2, 2, 3) # Set up graph
plt.plot(wat_radii, wat_terminal_velocities_with_corr, 'o') # Raw points
plt.plot(wat_x_with_corr, wat_y_with_corr) # Curve of best fit
plt.errorbar(wat_radii, wat_terminal_velocities_with_corr, xerr=wat_radii_uncern, yerr=wat_terminal_velocities_with_corr_uncern, fmt=".") #error bars
plt.xlabel("Radius (mm)") # Labels
plt.ylabel("Terminal Velocity (m/s)")
plt.title(f"Water: Terminal Velocity vs Radius a = {wat_a_with_corr} +/- {wat_a_with_corr_uncern}", fontsize=10) # Title
plt.tight_layout()

#Plot 4 residuals
plt.subplot(2, 2, 4)
plt.plot(wat_resid_x_with_corr, wat_resid_y_with_corr, 'o')
plt.plot(wat_x_zeroline, y_zeroline)
plt.errorbar(wat_resid_x_with_corr, wat_resid_y_with_corr, xerr=wat_radii_uncern, yerr=wat_terminal_velocities_with_corr_uncern, fmt=".") #error bars
plt.title("Residuals of Water graph", fontsize=10)
plt.tight_layout()
#plt.show()

print_values = False
if print_values == True:
    print('Glycerine')
    for i in range(5):
        print(f'Size: {i+1}, Diameter: {gly_radii[i]*2} +/- {gly_radii_uncern[i]*2}, Terminal Velocity: {gly_terminal_velocities[i]} +/- {gly_terminal_velocities_uncern[i]}')

    print('Water')
    for i in range(5):
        print(f'Size: {i+1}, Diameter: {wat_radii[i]*2} +/- {wat_radii_uncern[i]*2}, Terminal Velocity: {wat_terminal_velocities[i]} +/- {wat_terminal_velocities_uncern[i]}')

    print('\nWall Corrected')

    print('Glycerine')
    for i in range(5):
        print(f'Size: {i+1}, Diameter: {gly_radii[i]*2} +/- {gly_radii_uncern[i]*2}, Terminal Velocity: {gly_terminal_velocities_with_corr[i]} +/- {gly_terminal_velocities_with_corr_uncern[i]}')
    print('Water')
    for i in range(5):
        print(f'Size: {i+1}, Diameter: {wat_radii[i]*2} +/- {wat_radii_uncern[i]*2}, Terminal Velocity: {wat_terminal_velocities_with_corr[i]} +/- {wat_terminal_velocities_with_corr_uncern[i]}')


### Reynolds number
# Constants
gly_ρ = 1.26 #g/cm^3
gly_η = 9.34 #g/cm*s
wat_ρ = 1 #g/cm^3
wat_η = 0.01 #g/cm*s
teflon_ρ = 2.2
nylon_ρ = 0.005
gravity = 930 #cm/s^2

dgly_ρ = 0.005 #g/cm^3
dgly_η = 0.005 #g/cm*s
dwat_ρ = 0.0005 #g/cm^3
dwat_η = 0.00005 #g/cm*s
dteflon_ρ = 0.05
dnylon_ρ = 0.005


# Functions
def reynolds_number(density, char_length, velocity, viscosity):
    return (density*char_length*velocity)/viscosity

def reynolds_numbers_uncern(density, ddensity, char_length, dchar_length, velocity, dvelocity, viscosity, dviscosity, reynolds_numbers):
    # function: ρlv/η
    mul1 = uncertainty_prod(density, ddensity, char_length, dchar_length, density*char_length)
    mul2 = uncertainty_prod(density*char_length, mul1, velocity, dvelocity, density*char_length*velocity)
    mul3 = uncertainty_prod(density*char_length*velocity, mul2, viscosity, dviscosity, reynolds_numbers)

    return mul3

def r_calc_mean_std_dev(density, ddensity, radius, dradius, velocities, dvelocity, viscosity, dviscosity):
    reynolds_numbers = []
    reynolds_uncern = []
```

```
    for i in range(len(radius)):
        reynolds_numbers.append(reynolds_number(density, radius[i]/3, velocities[i], viscosity))
        reynolds_uncern.append(reynolds_numbers_uncern(density, ddensity, radius[i]/3, dradius[i]/3, velocities[i], dvelocity[i], viscosity, dviscosity, reyn

    #mean_reynolds = mean_diff_uncern(reynolds_numbers, np.array(reynolds_uncern))
    #uncern_reynolds = standard_deviation(np.array(mean_reynolds), np.array(reynolds_numbers))

    return reynolds_numbers, reynolds_uncern

reyn = False
if reyn == True:
    gly_reyn, gly_reyn_uncern = r_calc_mean_std_dev(gly_ρ, dgly_ρ, gly_radii, gly_radii_uncern, gly_terminal_velocities_with_corr, gly_terminal_velocities_wi
    wat_reyn, wat_reyn_uncern = r_calc_mean_std_dev(wat_ρ, dwat_ρ, gly_radii, wat_radii_uncern, wat_terminal_velocities_with_corr, wat_terminal_velocities_wi

    for i in range(5):
        print(f'Glycerine Reynolds Number {i}: {gly_reyn[i]} +/- {gly_reyn_uncern[i]}')
    for i in range(5):
        print(f'Water Reynolds Number {i}: {wat_reyn[i]} +/- {wat_reyn_uncern[i]}')


#To do list
# Chi squared
# r squared
```