Lab 06: Millikan Oil Drop Experiment

Import Libraries & Setup

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# rounding to 4th decimal place for easier reading
pd.options.display.float_format = '{:.2e}'.format
```

Reflection

I had no lab partner for this lab, so I was getting accustomed to doing a lab on my own. As well it would have been beneficial to consider how best to gather data before starting the experiment, as I changed methods in order to collect a reasonable ammount of data.

Data Analysis

Start by importing our data

```
day1 = pd.read_csv('data/day1.csv')
day1_long = pd.read_csv('data/day1_long.csv')
# day2 data is already in fall/rise format so convert it
day2_fall = pd.read_csv('data/day2_fall.csv').to_numpy()[:,1:]
day2_rise = pd.read_csv('data/day2_rise.csv').to_numpy()[:,1:]
```

`day1` and `day1_long` contain timestamps for when the oil drop hits a reticle. We need to calculate the fall and rise times from these timestamps.

```
# convert dataframes to numpy arrays
day1 = day1.to_numpy()
day1_long = day1_long.to_numpy()

# replace the Drop # column with zeroes to indicate t0
day1[:,0] = 0
day1_long[:,0] = 0

# calculate fall & rise times
day1_times = day1[:, 1:] - day1[:,:-1]
day1_fall = day1_times[:, ::2]
day1_rise = day1_times[:, 1::2]

day1_long_times = day1_long[:, 1:] - day1_long[:,:-1]
day1_long_fall = day1_long_times[:, ::2]
day1_long_rise = day1_long_times[:, 1::2]
```

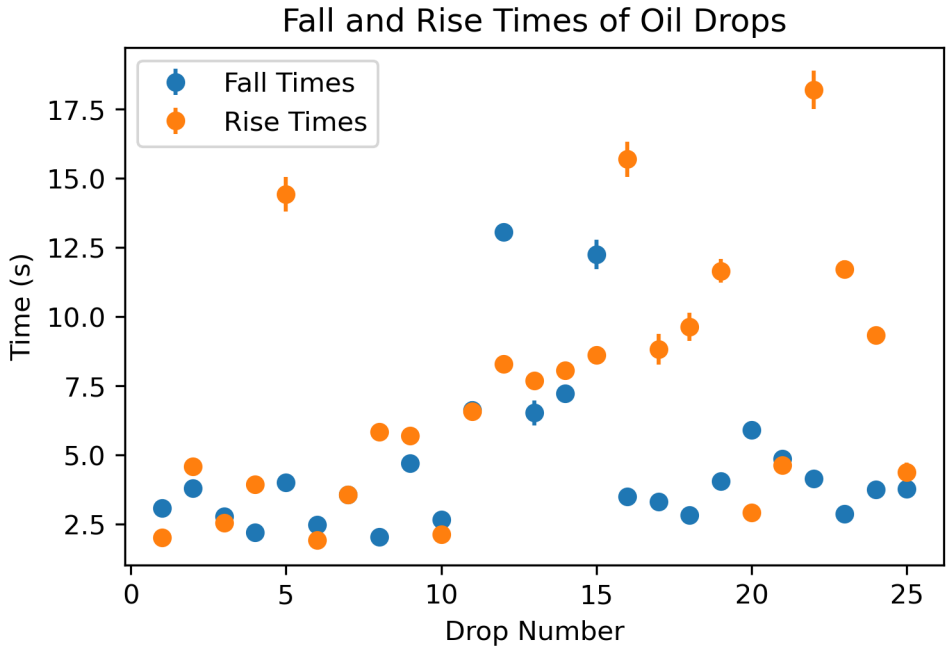
Now we get the mean fall and rise times for each drop

```
def combine_times(fall, rise, day):
    fall_means = np.mean(fall, axis=1)
    fall_stds = np.std(fall, axis=1)
    rise_means = np.mean(rise, axis=1)
    rise_stds = np.std(rise, axis=1)
    day_array = np.full(fall.shape[0], day)
    return np.stack((fall_means, fall_stds, rise_means, rise_stds, day_array), axis=1)
day1_combined = combine_times(day1_fall, day1_rise, 1)
day1_long_combined = combine_times(day1_long_fall, day1_long_rise, 1)
day2_combined = combine_times(day2_fall, day2_rise, 2)
drops = np.concatenate((day1_combined, day1_long_combined, day2_combined), axis=0)

# output to csv for use in lab report table
drops_df = pd.DataFrame(np.round(drops, 2), columns=['Fall Time', 'Fall Std', 'Rise Time', '
drops_df.insert(0, 'Drop #', np.arange(1, len(drops_df)+1))
drops_df.to_csv('data/drop_stats.csv', index=False)
```

Plot the fall and rise times of drops with error bars

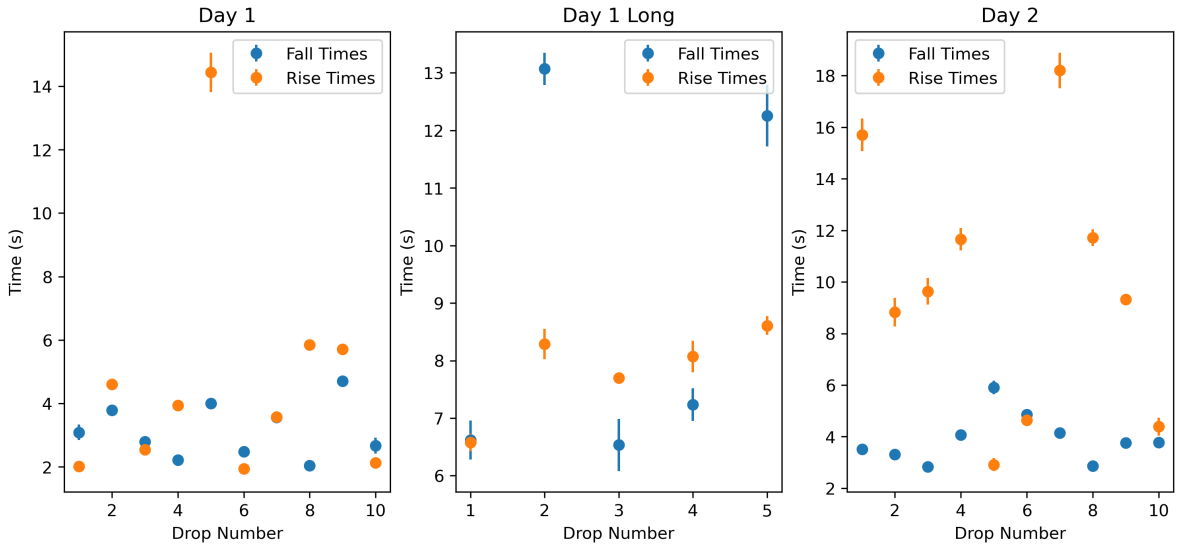
```
X = np.arange(0, len(drops), 1) + 1
plt.figure
plt.errorbar(X, drops[:,0], yerr=drops[:,1], fmt='o', label='Fall Times')
```

```
plt.errorbar(X, drops[:,2], yerr=drops[:,3], fmt='o', label='Rise Times')
plt.xlabel('Drop Number')
plt.ylabel('Time (s)')
plt.title('Fall and Rise Times of Oil Drops')
plt.legend()
plt.show()
```



As well here are two graphs of all three datasets separately to see if there are any differences between them.

```
titles = ['Day 1', 'Day 1 Long', 'Day 2']
plt.figure(figsize=(12, 5))
for i, curr_drops in enumerate([day1_combined, day1_long_combined, day2_combined]):
  plt.subplot(1, 3, i+1)
  X = np.arange(0, len(curr_drops), 1) + 1
  plt.errorbar(X, curr_drops[:,0], yerr=curr_drops[:,1], fmt='o', label='Fall Times')
  plt.errorbar(X, curr_drops[:,2], yerr=curr_drops[:,3], fmt='o', label='Rise Times')
  plt.xlabel('Drop Number')
  plt.ylabel('Time (s)')
  plt.title(titles[i])
  plt.legend()
plt.show()
```



Calculating Charge

Now we can calculate the charge on each drop using the formula provided in the lab manual. We will need to define some constants first.

```
# Constants
g = 9.81  # m/s^2
b = 8.23e-6  # Pa.m (constant for air viscosity correction)
eta_1 = 18.52e-6  # N.s/m^2 (viscosity of air at day 1 temp)
eta_2 = 18.48e-6  # N.s/m^2 (viscosity of air at day 2 temp)
P_1 = 93.83  # kPa (pressure day 1)
P_2 = 93.26  # kPa (pressure day 2)
P_err = 0.01 # kPa (error in pressure measurements)
dist_1 = (18.14-6.20-6.12)*10e-3  # m (distance between plates measured day 1)
dist_2 = (18.16-6.05-6.12)*10e-3 # m (distance between plates measured day 2)
raw_dist_err = np.sqrt((0.01e-3)**2 * 3) # m (error in the distance measurements)
dist = (dist_1 + dist_2) / 2 # Weighted average of distances
dist_err = raw_dist_err / np.sqrt(2) # Weighted error of distances
d = 0.5e-3  # m (distance between reticles)
V = 400  # V (voltage across plates)
rho_oil = 890  # kg/m^3 (density of oil)
```

Now we can calculate the charge on each drop

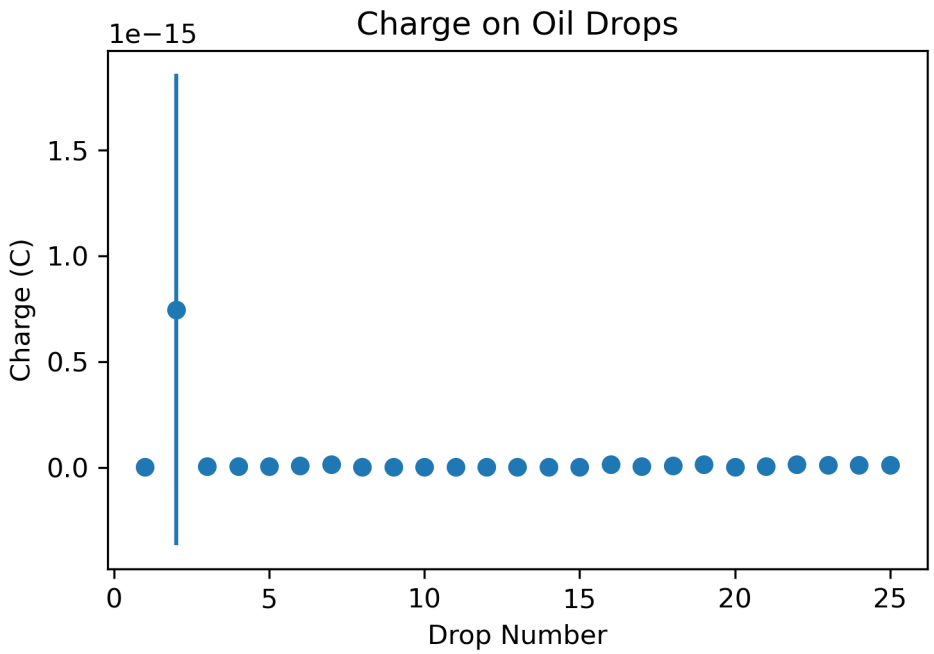
```
fall_time = drops[:,1]
fall_err = drops[:,1]
rise_time = drops[:,2]
rise_err = drops[:,3]
day = drops[:,4]
eta = np.where(day == 1, eta_1, eta_2)
P = np.where(day == 1, P_1, P_2)
dist_used = np.where(day == 1, dist_1, dist_2)
v_fall = d/fall_time
v_fall_err = d/(fall_time**2) * fall_err
v_rise = d/rise_time
v_rise_err = d/(rise_time**2) * fall_err
r = -b/(2*P) + np.sqrt((b**2)/(4*P**2) + (9 * eta * v_fall)/(2 * rho_oil * g))
r_err = np.sqrt(
  (P_err*(9*eta)/(8*rho_oil*g*(r+b/(2*P))))**2 +
  (v_fall_err * (b/(2*P**2) - (b**2)/(4*(P**3)*(r+b/(2*P)))))**2
)
E = V / dist
q = ((v_fall + v_rise)*4*np.pi*rho_oil*g*r**3)/(3*E)
q_err = q * np.sqrt(
  (v_rise_err/(v_fall+v_rise))**2 +
  (1/(2*v_fall) + 1/(v_fall + v_rise))**2 * v_fall_err**2
)

# output to csv for use in lab report table
charges_df = pd.DataFrame({ "Drop #": drops_df["Drop #"], "Charge": q, "Error": q_err})
charges_df.to_csv('data/drop_charges.csv', index=False, float_format='%.2e')
charges_df
```

	Drop #	Charge	Error
0	1	1.08e-18	1.50e-18
1	2	7.44e-16	1.11e-15
2	3	5.49e-18	7.99e-18
3	4	5.82e-18	8.55e-18
4	5	3.60e-18	5.36e-18
5	6	8.29e-18	1.20e-17
6	7	1.26e-17	1.86e-17
7	8	2.33e-18	3.42e-18
8	9	2.13e-18	3.14e-18
9	10	9.34e-19	1.30e-18
10	11	4.22e-19	6.12e-19

	Drop #	Charge	Error
11	12	6.69e-19	9.81e-19
12	13	2.09e-19	3.01e-19
13	14	6.56e-19	9.62e-19
14	15	1.36e-19	1.95e-19
15	16	1.52e-17	2.27e-17
16	17	4.49e-18	6.67e-18
17	18	7.37e-18	1.10e-17
18	19	1.37e-17	2.05e-17
19	20	8.10e-19	1.15e-18
20	21	5.39e-18	7.95e-18
21	22	1.51e-17	2.26e-17
22	23	9.55e-18	1.43e-17
23	24	1.15e-17	1.72e-17
24	25	9.23e-18	1.36e-17

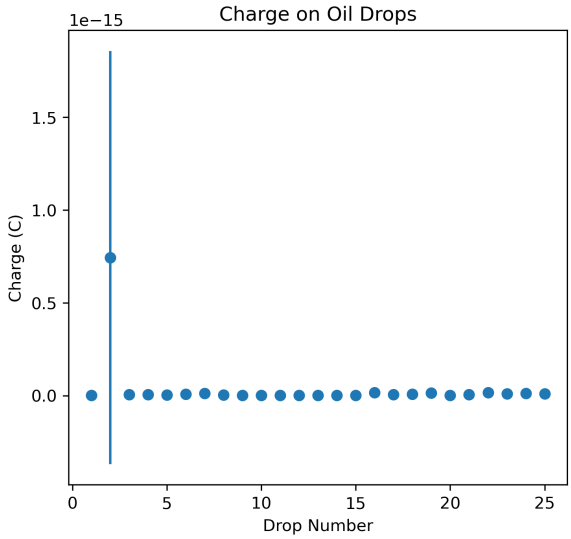
Now we can plot q

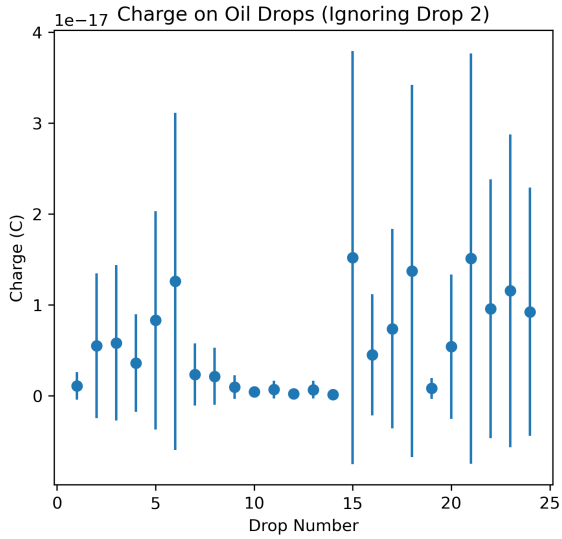
```
plt.figure()
X = np.arange(0, len(q), 1) + 1
plt.errorbar(X, q, yerr=q_err, fmt='o')
plt.xlabel('Drop Number')
plt.ylabel('Charge (C)')
plt.title('Charge on Oil Drops')
plt.show()
```



For some reason, we had a really large uncertainty on the 2nd drop, which means that it is hard to see the quantization of charge in this plot. However, if we ignore that point can see the quantization more clearly. Note we only ignore this point for visualization, not for any calculations.

```
# Plot both for easier comparison and to make the lab report look nicer
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
X = np.arange(0, len(q), 1) + 1
plt.errorbar(X, q, yerr=q_err, fmt='o')
plt.xlabel('Drop Number')
plt.ylabel('Charge (C)')
plt.title('Charge on Oil Drops')
plt.subplot(1,2,2)
Q = q[~(X==2)]
Q_err = q_err[~(X==2)]
X = np.arange(0, len(Q), 1) + 1
plt.errorbar(X, Q, yerr=Q_err, fmt='o')
plt.xlabel('Drop Number')
plt.ylabel('Charge (C)')
plt.title('Charge on Oil Drops (Ignoring Drop 2)')
plt.show()
```





from this it is easier to see a noticable pattern in the charges, which suggests that charge is quantized, despite the large uncertainties in some of the measurements. It seems that the more charged our oil drops are the larger the uncertainty in our measurements. This suggests that our error is systematic, likely due to some unaccounted for factor in our experiment, such as collisions with air molecules.

Let us divide by the smallest charge to see if we can find integer multiples

```
min_charge = np.min(q)
min_charge_err = q_err[np.argmin(q)]
ratios = q / min_charge
ratios_err = ratios * np.sqrt((q_err/q)**2 + (min_charge_err/min_charge)**2)
print(f"min_charge: {min_charge:.4e} ± {min_charge_err:.4e}")
ratios_df = pd.DataFrame({ "Drop #": drops_df["Drop #"], "Charge Ratios": ratios, "Error": ra
ratios_df.to_csv('data/ratios.csv', index=False, float_format='%.2f')
ratios_df
```

```
min_charge: 1.3559e-19 ± 1.9544e-19
```

	Drop #	Charge Ratios	Error
0	1	7.95e+00	1.59e+01
1	2	5.49e+03	1.14e+04
2	3	4.05e+01	8.29e+01
3	4	4.29e+01	8.83e+01
4	5	2.65e+01	5.50e+01

	Drop #	Charge Ratios	Error
5	6	6.12e+01	1.25e+02
6	7	9.28e+01	1.91e+02
7	8	1.72e+01	3.53e+01
8	9	1.57e+01	3.24e+01
9	10	6.89e+00	1.38e+01
10	11	3.11e+00	6.36e+00
11	12	4.93e+00	1.01e+01
12	13	1.54e+00	3.14e+00
13	14	4.84e+00	9.95e+00
14	15	1.00e+00	2.04e+00
15	16	1.12e+02	2.33e+02
16	17	3.31e+01	6.86e+01
17	18	5.44e+01	1.13e+02
18	19	1.01e+02	2.10e+02
19	20	5.97e+00	1.21e+01
20	21	3.98e+01	8.20e+01
21	22	1.11e+02	2.31e+02
22	23	7.05e+01	1.46e+02
23	24	8.51e+01	1.76e+02
24	25	6.81e+01	1.41e+02

We do not have very good data, so it is hard to see integer multiples in these ratios. If we divide by an integer we end up getting a worse fit for our estimated elementary charge, so we dont do that. Actually agrees well with the expected value of the elementary charge. Now we can try to estimate the elementary charge by dividing our charges by our ratios, rounded to the nearest integer, then we can look at the weigheted average of these estimates to get a better estimate of the elementary charge.

```
int_ratios = np.round(ratios)
e_estimates = q / int_ratios
e_estimates_err = e_estimates * np.sqrt((q_err/q)**2 + (ratios_err/ratios)**2)
weights = 1 / (e_estimates_err**2)
e_weighted_avg = np.sum(e_estimates * weights) / np.sum(weights)
e_weighted_err = np.sqrt(1 / np.sum(weights))
print(f"Estimated elementary charge: {e_weighted_avg:.4e} ± {e_weighted_err:.4e} C")
```

```
Estimated elementary charge: 1.3317e-19 ± 6.7343e-20 C
```

This actually agrees quite well with the accepted value of the elementary charge, which is approximately 1.602e-19 C. Overall, despite some large uncertainties in our measurements,

we were able to observe the quantization of charge and estimate the elementary charge with reasonable accuracy.