CP_ACarbone_z5260477

April 18, 2021

1 Lab 1: Coupled Pendulums

PHYS3112 - Experimental and Computational Physics

Anthony Carbone (z5260477)

I have done the Coupled Pendula lab in 2020 in PHYS2113 (Classical Mechanics), but online due to COVID. I will refer back to my some of my conclusions from that report throughout this report for deeper discussion of the results.

1.1 Aim

1. Obtain the spring constant used
2. Obtain the precise length of each pendulum from its uncoupled motion
3. Demonstrate the below theory in the case of coupled pendula
4. Obtain the spring constant by simple vertical harmonic motion

1.2 Theory

Derived from Pre-Work: - **In phase** the pendulums can be modeled as

$$\phi_1 = \phi_2 = \phi_{\max} \cos \sqrt{\frac{g}{L}} t \tag{1}$$

- π **out of phase** the pendulums can be modeled as

$$\phi_1 = -\phi_2 = \phi_{\max} \cos \sqrt{\frac{g}{L} + \frac{2l^2\kappa}{L^2}} t \tag{2}$$

- In **Beat mode** where $\phi_1(0) = \phi_{\max}$ and $\phi_2(0) = 0$ the pendulums can be modeled as

$$\phi_1 = \frac{\phi_{\max}}{2} \left[\cos\left(\sqrt{\frac{g}{L} + \frac{2l^2\kappa}{L^2}} t \right) + \cos\left(\sqrt{\frac{g}{L}} t \right) \right] \tag{3}$$

$$\phi_2 = \frac{\phi_{\max}}{2} \left[\cos\left(\sqrt{\frac{g}{L} + \frac{2l^2\kappa}{L^2}} t \right) - \cos\left(\sqrt{\frac{g}{L}} t \right) \right] \tag{4}$$

1.3 1. Spring Constant κ

1.3.1 Experimental Method

1. I hung spring from a point and set-up a metre ruler behind for measurements and recorded the initial displacement x_0 from the bottom of the spring.

2. I then added the 10g mass to the spring and recorded the displacement x from x_0.
3. I repeated this for each 10g mass incremement up to 60g.

1.3.2 Analysis

Hooke's Law in the simple case of a spring with a mass attached simplifies to $mg = \kappa x$, or rather $x = \frac{g}{\kappa}m$. Thus performing a linear fit and rearranging will result in κ.

Note, I considered the uncertainty in κ to be the standard error of the linear fit.

[1]:
```
#%% Import Modules
import pandas as pd
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from scipy.constants import g, pi
```

[2]:
```
#%% Import data

# Mass in kg and Displacement in m
df = pd.DataFrame(
    {'Mass': np.arange(0, .061, .01),
     'Displacement': np.array([0, 0.033, 0.065, 0.096, 0.127, 0.158, 0.191])}
    )

mass_abs_unc = 0.0005 # = 0.5g
disp_abs_unc = 0.005 # = 0.5cm

print('Raw data with mass in kg and displacement in m:')
print(df)
```

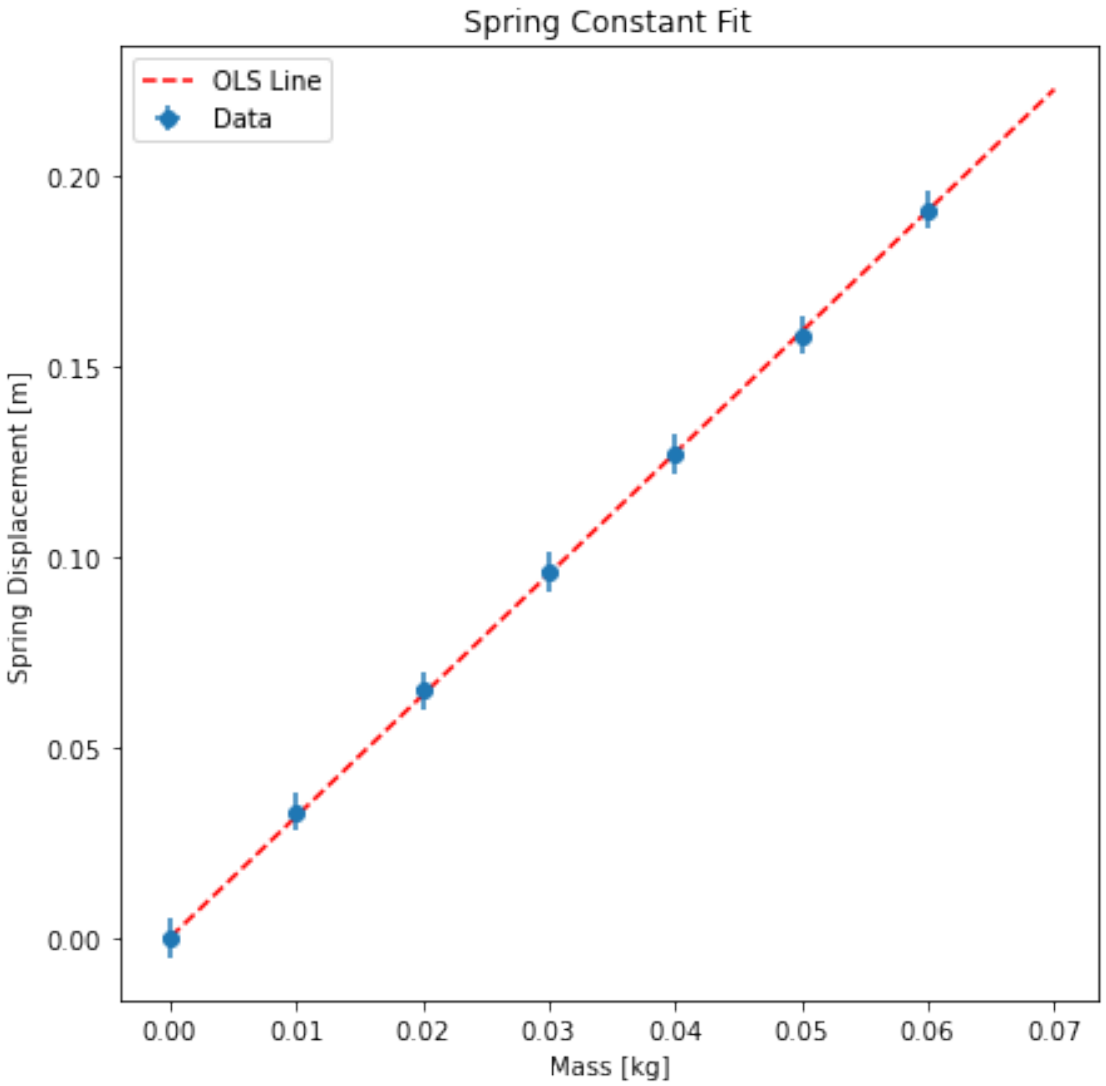
```
Raw data with mass in kg and displacement in m:
   Mass  Displacement
0  0.00         0.000
1  0.01         0.033
2  0.02         0.065
3  0.03         0.096
4  0.04         0.127
5  0.05         0.158
6  0.06         0.191
```

[3]:
```
#%% Fit

# OLS fit without intercept as x_0 = 0
fit_result = sm.OLS(df['Displacement'], df['Mass']).fit()

kappa = g / fit_result.params['Mass']
kappa_std = fit_result.bse['Mass']
```

```
print(f'kappa = {kappa:.2f} +/- {kappa_std:.2f} N/m')
print(f'R^2 = {fit_result.rsquared:.2f}')
```

```
kappa = 3.08 +/- 0.01 N/m
R^2 = 1.00
```

[4]:
```
#%% Plot fit on data
# Predicted from fit
mass_for_pred = np.linspace(0, 0.07, 201)
displacement_pred = g / kappa * mass_for_pred

plt.figure(figsize=[7,7])
plt.errorbar(df['Mass'], df['Displacement'], xerr=mass_abs_unc,␣
 ↪yerr=disp_abs_unc, fmt='o', label='Data')
plt.plot(mass_for_pred, displacement_pred, 'r--', label='OLS Line')
plt.legend()
plt.title('Spring Constant Fit')
plt.xlabel('Mass [kg]')
plt.ylabel('Spring Displacement [m]')
plt.show()
```



As seen above, the linear fit is exceptionally convincing with $R^2 = 1.00$.

Herein κ shall be considered to be $3.08 \pm 0.01 Nm^{-1}$.

This seems reasonable in comparison to my 2020 report where κ was found to be $3.10 \pm 0.02 Nm^{-1}$.

1.4 2. Pendulum Length L

I did measure the pendulums with a metre ruler to be 0.970 ± 0.05m.

As a precursor for the coupled analysis, I wanted to see if I could confirm this, perhaps with better accuracy, by recording the natural frequency f of their oscillation, and calculating their length as

$$f = \frac{1}{2\pi}\sqrt{\frac{g}{L}} \implies L = \frac{g}{(2\pi f)^2}.$$

1.4.1 Experimental Method

I simply recorded the uncoupled motion of each pendulum for minute. I did these measurements seperately to avoid any weak coupling between the desk and the fixtures.

1.4.2 Analysis

[5]:
```
#%% Import and prepare data
df1 = pd.read_csv('Data/uncoupled_1.txt', sep='\t', skiprows=[0, 1],␣
 ↪usecols=[0, 1], names=['Time', 'Phi1'])
df2 = pd.read_csv('Data/uncoupled_2.txt', sep='\t', skiprows=[0, 1],␣
 ↪usecols=[0, 2], names=['Time', 'Phi2'])

# Prepare data for Fourier Transform, time has to be linearly spaced, however I␣
 ↪made the
# mistake of not extending the decimal places provided in exporting from the␣
 ↪lab software,
# so there are duplicate time values with non-duplicate transducer voltages.

# I achieved this by taking the mean of all voltages for each time duplicate␣
 ↪and setting that
# as the new voltage for that time.
df1 = df1.groupby(['Time']).mean().reset_index()
df2 = df2.groupby(['Time']).mean().reset_index()
```

[6]:
```
#%% Fourier transform
df1_fftd = pd.DataFrame(
    {'Freq': np.fft.rfftfreq(df1.Time.size, d=0.01)[1:],
     'Amp': np.abs(np.fft.rfft(df1.Phi1))[1:]}
    )
df2_fftd = pd.DataFrame(
    {'Freq': np.fft.rfftfreq(df2.Time.size, d=0.01)[1:],
     'Amp': np.abs(np.fft.rfft(df2.Phi2))[1:]}
    )
```

[7]:
```
#%% Obtain natural frequency
# Note the absolute uncertainty is considered to be half the frequency bin width
freq1_index = df1_fftd.Amp.idxmax() # Find Freq corresponding
freq1 = df1_fftd.Freq[freq1_index]  # to max Amp.
freq1_abs_unc = (df1_fftd.Freq[freq1_index + 1] - df1_fftd.Freq[freq1_index -␣
 ↪1]) / 2

freq2_index = df2_fftd.Amp.idxmax()
freq2 = df2_fftd.Freq[freq2_index]
freq2_abs_unc = (df2_fftd.Freq[freq2_index + 1] - df2_fftd.Freq[freq2_index -␣
 ↪1]) / 2
```

[8]:
```
#%% Calculate pendulum length
L1 = g / (2 * pi * freq1)**2
L1_abs_unc = 2 * L1 * freq1_abs_unc / freq1

L2 = g / (2 * pi * freq2)**2
L2_abs_unc = 2 * L2 * freq2_abs_unc / freq2

print(f'Pendulum 1: L = {L1:.2f} +/- {L1_abs_unc:.2f} m')
print(f'Pendulum 2: L = {L2:.2f} +/- {L2_abs_unc:.2f} m')
```

```
Pendulum 1: L = 0.97 +/- 0.07 m
Pendulum 2: L = 0.98 +/- 0.07 m
```

Whilst the predicted value does agree with both the ruler-measured value and my 2020 report, the uncertainties are much larger than anticipated given its far greater than that of a metre ruler (± 0.005m). These uncertainties are also reasonably larger than the same analysis performed on the in-phase motion in my 2020 report which found $L = 0.965 \pm 0.003$m.

I believe this to be down to the time resolution I mentioned earlier, where the time values were truncated to two decimal points. To improve on this uncertainty range, I would increase the decimal points exported from the lab software to achieve a far more defined Fourier Transformed dataset.

One possibility I thought to do was assume each time duplicate was linearly split and alter the time duplicates to conform, ie `[0.55, 0.55, 0.55] -> [0.553, 0.556, 0.559]`. Unfortunately, lack of time saw to it this did not happen.

1.5 3. Coupling

I could have calculated a theoretical f_n using the ruler-measured L and the simple pendulum formula but as Section 2 revealed, the mistake of exporting truncated data would make it unsensible to compare the experimental conclusions with its larger uncertainties.

Therefore, throughout this Section I shall assume the natural frequency f_n of the uncoupled pendula to be equal to one another and be the mean of what was found in Section 2 with the largest uncertainty range between the two.

[9]:
```
fn = np.mean([freq1, freq2])
fn_lower = freq2 - freq2_abs_unc
fn_upper = freq1 + freq1_abs_unc

print(f'fn = {fn:.3f} with uncertainty range ({fn_lower:.3f}, {fn_upper:.3f})')
```

```
fn = 0.505 with uncertainty range (0.487, 0.525)
```

Experimental Method The method used in each mode and spring height is identical, aside from starting condition and spring height; 1. Measure spring length from pendulum pivot point l, ensuring its equal for both pendula 1. Hold the pendula at the starting conditions, ensuring small angles of approximately 10° 2. Release and ensure expected motion continues 3. Begin recording transducer recording for one minute 4. Export data

Note, I considered the uncertainty $\Delta l = 0.5$cm.

Below is the imported data as a dictionary.

[10]:
```
#%% Import Data
ls = [10, 25, 53, 75, 90] # List of spring lengths from top of pendulum [cm]
modes = ['inphase', 'outofphase', 'beatmode'] # List of modes

df_dict = {} # Create dictionary of all data, level 0 = mode, level 1 = spring␣
 ↪length
for mode in modes:
    df_dict[f'{mode}'] = {}
    for l in ls:
        df = pd.read_csv(f'Data/l{l}cm_{mode}.txt', sep='\t', skiprows=[0, 1],
                         usecols=[0, 1, 2], names=['Time', 'Phi1', 'Phi2'])
        df = df.groupby(['Time']).mean().reset_index() # Collapse time␣
 ↪duplicates as means
        df_fftd = pd.DataFrame(
            {'Freq': np.fft.rfftfreq(df.Time.size, d=0.01)[1:],
             'Amp1': np.abs(np.fft.rfft(df.Phi1))[1:],
             'Amp2': np.abs(np.fft.rfft(df.Phi2))[1:]}
            )
        df_dict[f'{mode}'][l] = df_fftd
del l, mode, df, df_fftd
```

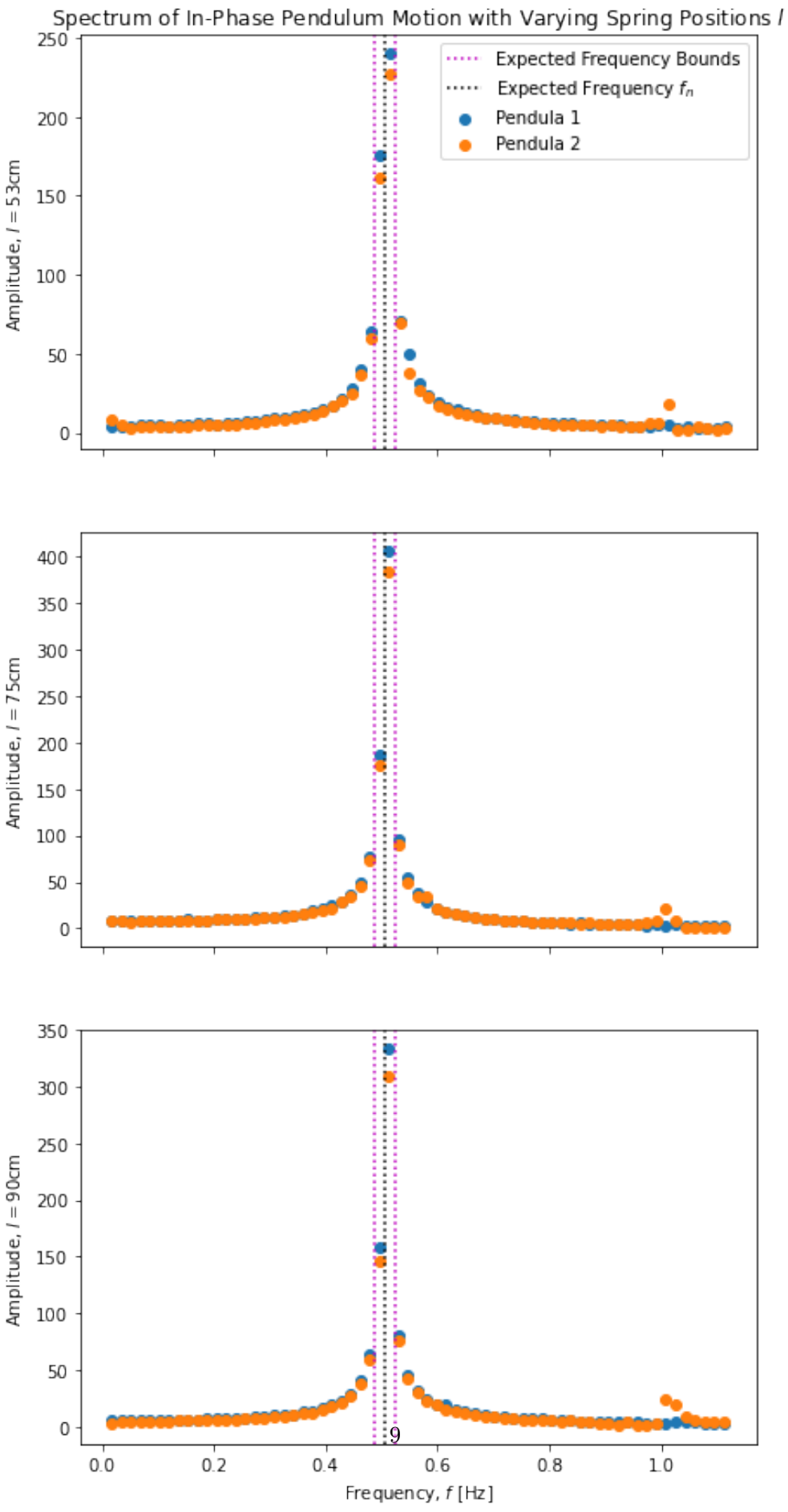
1.5.1 3.1. In-Phase

In this Subsection, I merely investigate that the in-phase oscillation is indeed that of the uncoupled natural frequency f_n.

Analysis

[11]:
```
#%% Plot spectrum and predicted frequency

# Calculate uncoupled frequency and uncertainty bounds
f_theo = np.mean([freq1, freq2])
f_theo_lower = freq2 - freq2_abs_unc
f_theo_upper = freq1 + freq1_abs_unc

# Plot
interval = 65 # Region of interest
fig, (ax1, ax2, ax3) = plt.subplots(nrows = 3, sharex = True, figsize = [7, 15])
ax1.set_title('Spectrum of In-Phase Pendulum Motion with Varying Spring␣
 ↪Positions $l$')

ax1.scatter(df_dict['inphase'][53].Freq[:interval], df_dict['inphase'][53].
 ↪Amp1[:interval], label='Pendula 1')
ax1.scatter(df_dict['inphase'][53].Freq[:interval], df_dict['inphase'][53].
 ↪Amp2[:interval], label='Pendula 2')
```

```
ax1.axvline(x = f_theo_lower, c = 'm', linestyle = ':', label = 'Expected␣
 ↪Frequency Bounds')
ax1.axvline(x=f_theo, c='k', linestyle=':', label = 'Expected Frequency $f_n$')
ax1.axvline(x = f_theo_upper, c = 'm', linestyle = ':')
ax1.set_ylabel('Amplitude, $l = 53$cm')
ax1.legend()

ax2.scatter(df_dict['inphase'][75].Freq[:interval], df_dict['inphase'][75].
 ↪Amp1[:interval])
ax2.scatter(df_dict['inphase'][75].Freq[:interval], df_dict['inphase'][75].
 ↪Amp2[:interval])
ax2.axvline(x = f_theo_lower, c = 'm', linestyle = ':')
ax2.axvline(x=f_theo, c='k', linestyle=':')
ax2.axvline(x = f_theo_upper, c = 'm', linestyle = ':')
ax2.set_ylabel('Amplitude, $l = 75$cm')

ax3.scatter(df_dict['inphase'][90].Freq[:interval], df_dict['inphase'][90].
 ↪Amp1[:interval])
ax3.scatter(df_dict['inphase'][90].Freq[:interval], df_dict['inphase'][90].
 ↪Amp2[:interval])
ax3.axvline(x = f_theo_lower, c = 'm', linestyle = ':')
ax3.axvline(x=f_theo, c='k', linestyle=':')
ax3.axvline(x = f_theo_upper, c = 'm', linestyle = ':')
ax3.set_ylabel('Amplitude, $l = 90$cm')
ax3.set_xlabel('Frequency, $f$ [Hz]')
plt.show()
```


Spectrum of In-Phase Pendulum Motion with Varying Spring Positions l
Expected Frequency Bounds
Expected Frequency f_n
Pendula 1
Pendula 2
Amplitude, l = 53cm
Amplitude, l = 75cm
Amplitude, l = 90cm
Frequency, f [Hz]
0.0
0.2
0.4
0.6
0.8
1.0

As can be seen in the above figure, the two pendula oscillate at frequency $\approx f_n$, confirming the hypothesis that in-phase identical coupled pendula oscillate at their natural frequency.

1.5.2 3.2. Out-of-Phase

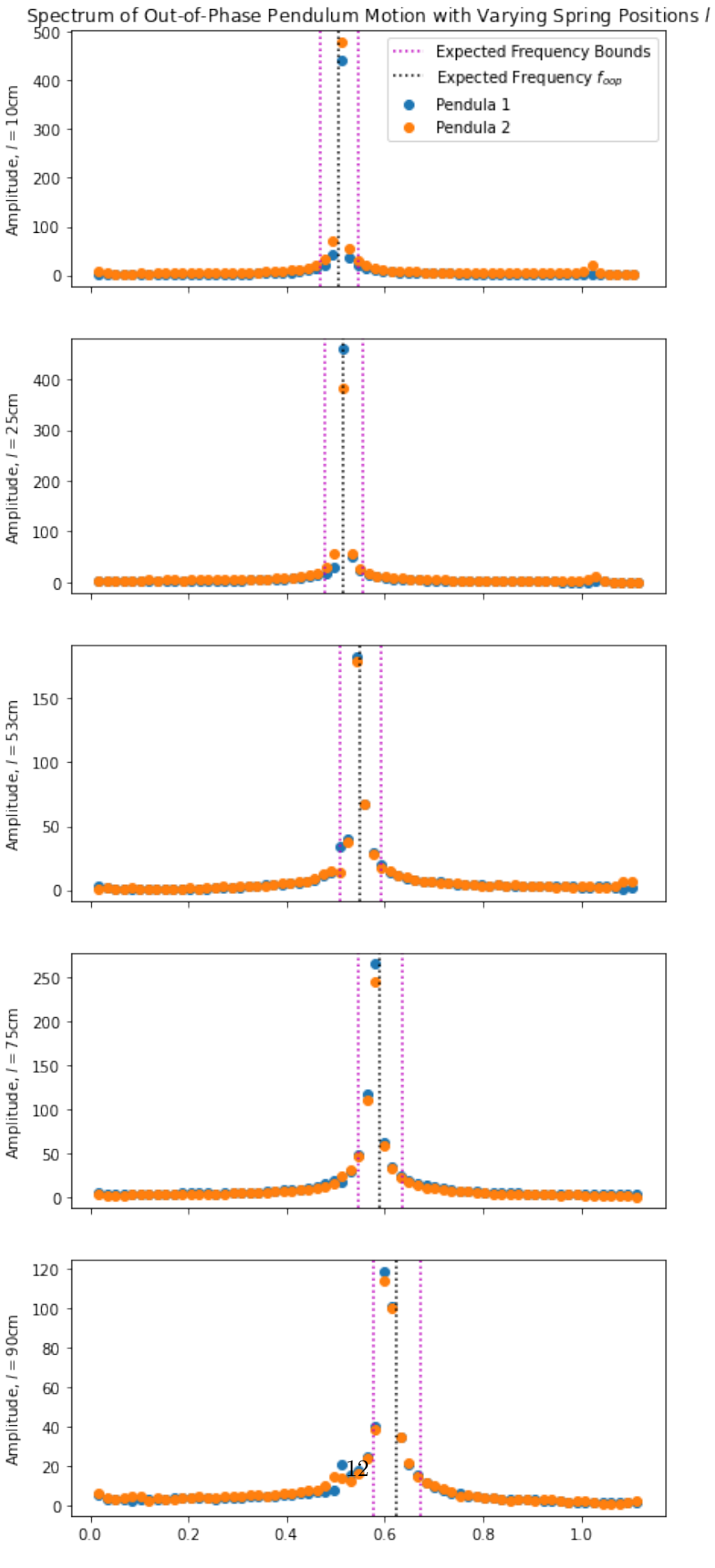
In this Subsection, I attempt to predict the out-of-phase (oop) frequency knowing $f_{oop} = \frac{1}{2\pi}\sqrt{\frac{g}{L} + \frac{2l^2\kappa}{L^2}}$ where f_n is solely used

$$f_{oop} = f_n\sqrt{1 + \frac{8\pi^2 l^2 \kappa f_n^2}{g^2}}.$$

Below I calculate these predicted f_{oop} values and propogated uncertainties.

[12]:
```
#%% Predict outofphase frequency
f_outofphase = {}
f_outofphase_lower = {}
f_outofphase_upper = {}

for l in ls:
    f = fn * np.sqrt(1 + (8 * pi**2 * (l / 100)**2 * kappa * fn**2) / g**2)
    fn_rel_unc = (fn_upper - fn_lower) / (2 * fn)
    l_rel_unc = 0.005 / l
    kappa_rel_unc = kappa_std / kappa
    abs_unc = f * (2 * fn_rel_unc + l_rel_unc + kappa_rel_unc / 2)

    f_outofphase[l] = f
    f_outofphase_lower[l] = f - abs_unc
    f_outofphase_upper[l] = f + abs_unc

del abs_unc, f, fn_rel_unc, l_rel_unc, l

#%% Plot
interval = 65 # Region of interest
fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(nrows=5, sharex=True, figsize=[7,␣
↪18])
ax1.set_title('Spectrum of Out-of-Phase Pendulum Motion with Varying Spring␣
↪Positions $l$')

ax1.scatter(df_dict['outofphase'][10].Freq[:interval],␣
↪df_dict['outofphase'][10].Amp1[:interval], label='Pendula 1')
ax1.scatter(df_dict['outofphase'][10].Freq[:interval],␣
↪df_dict['outofphase'][10].Amp2[:interval], label='Pendula 2')
ax1.axvline(x=f_outofphase_lower[10], c='m', linestyle=':', label='Expected␣
↪Frequency Bounds')
```

```
ax1.axvline(x=f_outofphase[10], c='k', linestyle=':', label='Expected Frequency␣
 ↪$f_{oop}$')
ax1.axvline(x=f_outofphase_upper[10], c='m', linestyle=':')
ax1.set_ylabel('Amplitude, $l = 10$cm')
ax1.legend()

ax2.scatter(df_dict['outofphase'][25].Freq[:interval],␣
 ↪df_dict['outofphase'][25].Amp1[:interval], label='Pendula 1')
ax2.scatter(df_dict['outofphase'][25].Freq[:interval],␣
 ↪df_dict['outofphase'][25].Amp2[:interval], label='Pendula 2')
ax2.axvline(x=f_outofphase_lower[25], c='m', linestyle=':')
ax2.axvline(x=f_outofphase[25], c='k', linestyle=':')
ax2.axvline(x=f_outofphase_upper[25], c='m', linestyle=':')
ax2.set_ylabel('Amplitude, $l = 25$cm')

ax3.scatter(df_dict['outofphase'][53].Freq[:interval],␣
 ↪df_dict['outofphase'][53].Amp1[:interval], label='Pendula 1')
ax3.scatter(df_dict['outofphase'][53].Freq[:interval],␣
 ↪df_dict['outofphase'][53].Amp2[:interval], label='Pendula 2')
ax3.axvline(x=f_outofphase_lower[53], c='m', linestyle=':')
ax3.axvline(x=f_outofphase[53], c='k', linestyle=':')
ax3.axvline(x=f_outofphase_upper[53], c='m', linestyle=':')
ax3.set_ylabel('Amplitude, $l = 53$cm')

ax4.scatter(df_dict['outofphase'][75].Freq[:interval],␣
 ↪df_dict['outofphase'][75].Amp1[:interval], label='Pendula 1')
ax4.scatter(df_dict['outofphase'][75].Freq[:interval],␣
 ↪df_dict['outofphase'][75].Amp2[:interval], label='Pendula 2')
ax4.axvline(x=f_outofphase_lower[75], c='m', linestyle=':')
ax4.axvline(x=f_outofphase[75], c='k', linestyle=':')
ax4.axvline(x=f_outofphase_upper[75], c='m', linestyle=':')
ax4.set_ylabel('Amplitude, $l = 75$cm')

ax5.scatter(df_dict['outofphase'][90].Freq[:interval],␣
 ↪df_dict['outofphase'][90].Amp1[:interval], label='Pendula 1')
ax5.scatter(df_dict['outofphase'][90].Freq[:interval],␣
 ↪df_dict['outofphase'][90].Amp2[:interval], label='Pendula 2')
ax5.axvline(x=f_outofphase_lower[90], c='m', linestyle=':')
ax5.axvline(x=f_outofphase[90], c='k', linestyle=':')
ax5.axvline(x=f_outofphase_upper[90], c='m', linestyle=':')
ax5.set_ylabel('Amplitude, $l = 90$cm')
plt.show()
```


Spectrum of Out-of-Phase Pendulum Motion with Varying Spring Positions l
Expected Frequency Bounds
Expected Frequency f_{oop}
Pendula 1
Pendula 2
Amplitude, l = 10cm
Amplitude, l = 25cm
Amplitude, l = 53cm
Amplitude, l = 75cm
Amplitude, l = 90cm
0.0
0.2
0.4
0.6
0.8
1.0

The above figure is particularly interesting given it illustrates that as the pendula become more coupled (as l becomes larger), the frequency f_{oop} becomes larger. Therfore, the more coupled the pendula, the faster they oscillate.

Aside from this, the actual frequencies fit well within the predicted range, confirming the above relationship.

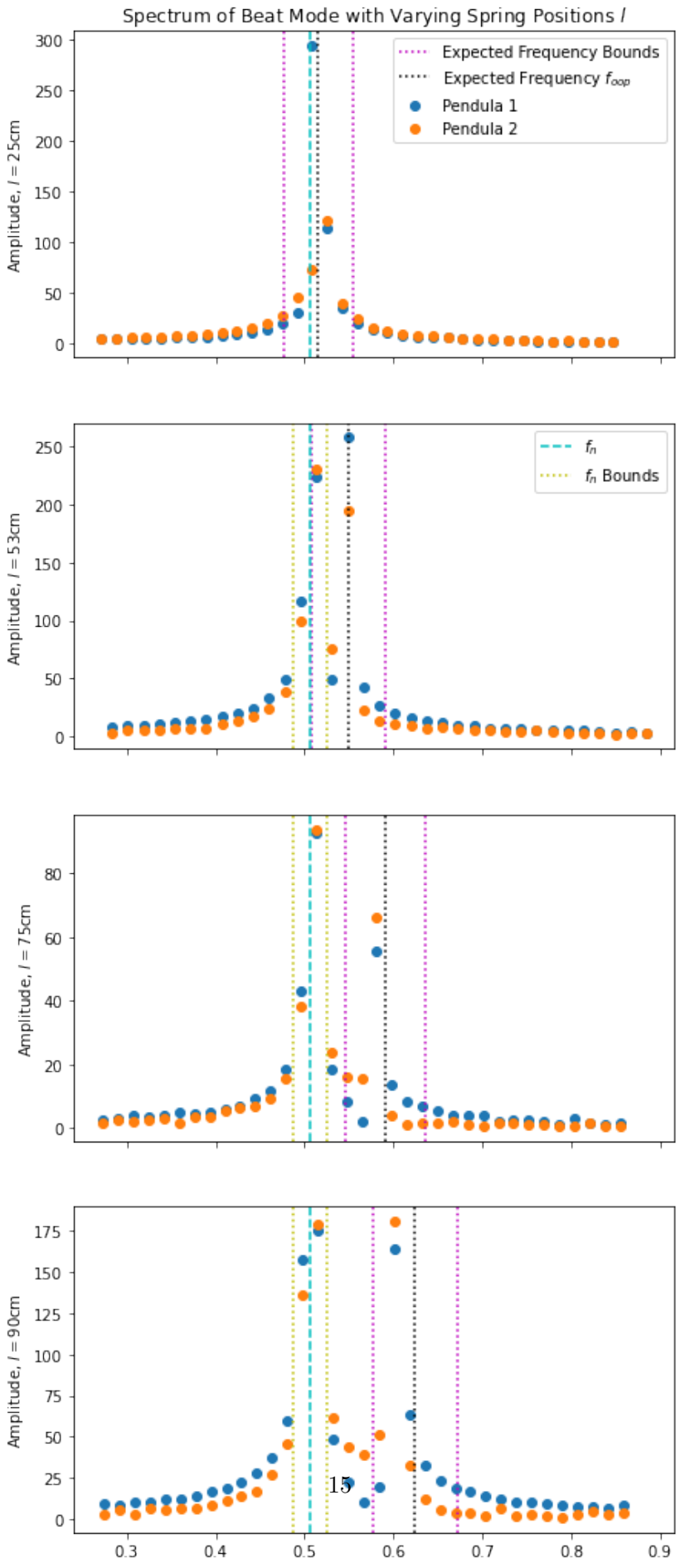
1.5.3 3.3. Beat Mode

With these initial conditions, two predominant frequencies should become apparent, f_n and f_{oop}.

Having already calculated and used these predicted values above, I will skip to the figures below.

[13]:
```
#%% Plot
(lower_int, upper_int) = (15, 50) # Region of interest
fig, (ax2, ax3, ax4, ax5) = plt.subplots(nrows=4, sharex=True, figsize=[7, 18])

ax2.set_title('Spectrum of Beat Mode with Varying Spring Positions $l$')

ax2.scatter(df_dict['beatmode'][25].Freq[lower_int:upper_int],␣
 ↪df_dict['beatmode'][25].Amp1[lower_int:upper_int], label='Pendula 1')
ax2.scatter(df_dict['beatmode'][25].Freq[lower_int:upper_int],␣
 ↪df_dict['beatmode'][25].Amp2[lower_int:upper_int], label='Pendula 2')
ax2.axvline(x=f_outofphase_lower[25], c='m', linestyle=':', label='Expected␣
 ↪Frequency Bounds')
ax2.axvline(x=f_outofphase[25], c='k', linestyle=':', label='Expected Frequency␣
 ↪$f_{oop}$')
ax2.axvline(x=f_outofphase_upper[25], c='m', linestyle=':')
ax2.axvline(x = fn, c = 'c', linestyle = '--')
# ax2.axvline(x = fn_lower, c = 'y', linestyle = ':') # Comment out if you wish
# ax2.axvline(x = fn_upper, c = 'y', linestyle = ':') # to see the fn bounds
ax2.set_ylabel('Amplitude, $l = 25$cm')
ax2.legend()

ax3.scatter(df_dict['beatmode'][53].Freq[lower_int:upper_int],␣
 ↪df_dict['beatmode'][53].Amp1[lower_int:upper_int])
ax3.scatter(df_dict['beatmode'][53].Freq[lower_int:upper_int],␣
 ↪df_dict['beatmode'][53].Amp2[lower_int:upper_int])
ax3.axvline(x=f_outofphase_lower[53], c='m', linestyle=':')
ax3.axvline(x=f_outofphase[53], c='k', linestyle=':')
ax3.axvline(x=f_outofphase_upper[53], c='m', linestyle=':')
ax3.axvline(x = fn, c = 'c', linestyle = '--', label = '$f_n$')
ax3.axvline(x = fn_lower, c = 'y', linestyle = ':', label='$f_n$ Bounds')
ax3.axvline(x = fn_upper, c = 'y', linestyle = ':')
ax3.set_ylabel('Amplitude, $l = 53$cm')
ax3.legend()
```

```
ax4.scatter(df_dict['beatmode'][75].Freq[lower_int:upper_int],␣
 ↪df_dict['beatmode'][75].Amp1[lower_int:upper_int])
ax4.scatter(df_dict['beatmode'][75].Freq[lower_int:upper_int],␣
 ↪df_dict['beatmode'][75].Amp2[lower_int:upper_int])
ax4.axvline(x=f_outofphase_lower[75], c='m', linestyle=':')
ax4.axvline(x=f_outofphase[75], c='k', linestyle=':')
ax4.axvline(x=f_outofphase_upper[75], c='m', linestyle=':')
ax4.axvline(x = fn, label = '$f_n$', c = 'c', linestyle = '--')
ax4.axvline(x = fn_lower, c = 'y', linestyle = ':')
ax4.axvline(x = fn_upper, c = 'y', linestyle = ':')
ax4.set_ylabel('Amplitude, $l = 75$cm')

ax5.scatter(df_dict['beatmode'][90].Freq[lower_int:upper_int],␣
 ↪df_dict['beatmode'][90].Amp1[lower_int:upper_int])
ax5.scatter(df_dict['beatmode'][90].Freq[lower_int:upper_int],␣
 ↪df_dict['beatmode'][90].Amp2[lower_int:upper_int])
ax5.axvline(x=f_outofphase_lower[90], c='m', linestyle=':')
ax5.axvline(x=f_outofphase[90], c='k', linestyle=':')
ax5.axvline(x=f_outofphase_upper[90], c='m', linestyle=':')
ax5.axvline(x = fn, label = '$f_n$', c = 'c', linestyle = '--')
ax5.axvline(x = fn_lower, c = 'y', linestyle = ':')
ax5.axvline(x = fn_upper, c = 'y', linestyle = ':')
ax5.set_ylabel('Amplitude, $l = 90$cm')
plt.show()
```


Spectrum of Beat Mode with Varying Spring Positions l
Expected Frequency Bounds
Expected Frequency f_{oop}
Pendula 1
Pendula 2
Amplitude, l = 25cm
Amplitude, l = 53cm
f_n
f_n Bounds
Amplitude, l = 75cm
Amplitude, l = 90cm
0.3
0.4
0.5
0.6
0.7
0.8
0.9

The above figure depicts the spectrum again of the motion but on a smaller horizontal (its zoomed in) to assist in discerning the two predominant peaks. Both the f_{oop} and f_n ranges are plotted as well, though the latter bounds are omitted from the $l = 25$cm given the peaks are so close together.

Whilst confidence in the accuracy of the prediction may benefit from better defined data (referring to the my data truncation mistake), it is reasonable to conclude the predominant frequencies are as predicted within the expected ranges. This further confirms the above theory, and that found in the pre-work.

1.5.4 4. Spring Constant κ from SHM

Out of curiosity, I wanted to see whether using the simple harmonic motion of a spring with a hanging mass would yield a more accurate result of the spring constant κ.

From Hooke's Law it can be shown for a linear spring with a hanging mass oscillates with period

$$T^2 = \frac{4\pi^2}{\kappa} m$$

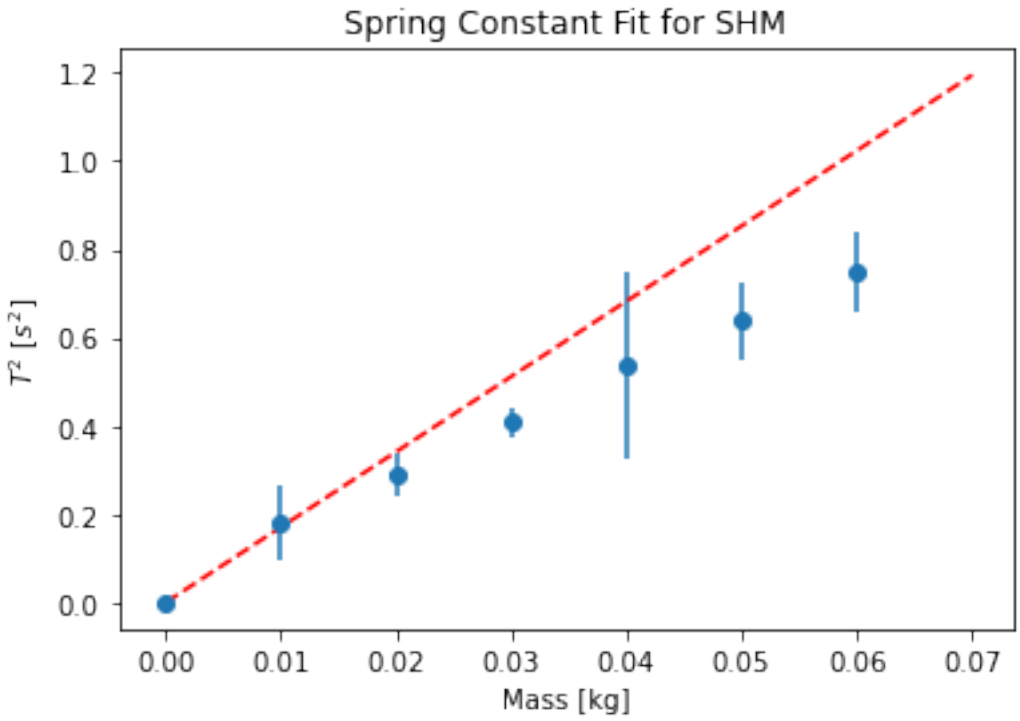
Experimental Method

1. Hang spring with 70g
2. Begin small oscillations
3. Time how long twenty oscillations took
4. Perform three such trials
5. Perform above steps for 10g decrements

[14]:
```
raw_data = {10: [8.55, 8.46, 8.38],
        20: [10.84, 10.77, 10.74],
        30: [12.80, 12.73, 12.80],
        40: [14.52, 14.92, 14.50],
        50: [15.85, 16.03, 16.02],
        60: [17.41, 17.30, 17.23]}

period = np.array([0])
mass = np.arange(0, .061, .01)
period_abs_unc = np.array([0])

for twenty_periods in raw_data.values():
    period_abs_unc = np.append(period_abs_unc, np.ptp(twenty_periods) / 2)
    period = np.append(period, np.mean(twenty_periods) / 20)
del twenty_periods

# OLS fit without intercept as x_0 = 0
fit_result = sm.OLS(period, mass).fit()

kappa = 4 * pi**2 / fit_result.params[0]
kappa_std = fit_result.bse[0]
```

```
print(f'So kappa = {kappa:.2f} +/- {kappa_std:.2f}.')

# Predicted
mass_for_pred = np.linspace(0, 0.07, 201)
period_pred = 4 * pi**2 / kappa * mass_for_pred

plt.errorbar(mass, period**2, xerr=0.0005, yerr=period_abs_unc, fmt='o',␣
↪label='Data') # TODO Make clearer
plt.plot(mass_for_pred, period_pred, 'r--', label='OLS Line')
plt.title('Spring Constant Fit for SHM')
plt.xlabel('Mass [kg]')
plt.ylabel('$T^2$ [$s^2$]')
plt.show()
```

```
So kappa = 2.31 +/- 1.65.
```



As can be seen in the above figure, the fit fails to fit within all the uncertainty ranges. The data itself was found to be reasonably inaccurate. More trials could be performed and may result in a more accurate spring constant that using the method in Section 1, but would require further investigation.

It does however agree with the previously obtained value of $\kappa = 3.08 \pm 0.01 Nm^{-1}$.

1.6 Conclusions

To conclude, the derived equations for the motion of two coupled pendulum accurately describe the experimental behaviour, given the motion satisfies the small angle approximation.

The spring constant κ from the spring characterisation data was calculated to be $\kappa = 3.08 \pm 0.01$N/m.

For in phase pendulums, the predominant frequency f_n was accurately predicted.

For out of phase pendulums, the predominant frequency f_{oop} too was accurately predicted.

For pendulums in beat mode, the two predominant frequencies were accurately predicted although may prompt further investigation to better differentiate the two. I recommend doing so by increasing the length of the pendulums as f_{oop} is inversely proportional to L which would separate the frequencies on the spectrum to permit more convincing conclusions.

Of course, the truncation mistake can be avoided by setting the software to export in a greater number of decimal places.

Regarding the Simple Harmonic Motion excercise to find the spring constant, more trials should be performed to obtain a reasonably accurate κ.