H01e Pendulum as an accelerated Frame of Reference

October 30, 2024

Author(s)

- **Carla Rotzoll**, 50% contribition
- **Mirzokhid Ganiev**, 50% contribution

Group Number: 03

0.0.1 Abstract

This research aims to calculate the acceleration ($\vec{a}$) and linear acceleration without gravity ($\vec{a_g}$) of a pendulum using measured angular velocity ($\vec{\omega_z}$). The calculated values are then compared to direct measurements of $\vec{a}$ and $\vec{a_g}$. Additionally, the study determines the angular frequency (f_d) and the damping constant (δ) to facilitate comparisons between the two setups. The experiment uses a smartphone equipped with the **PhyPhox** application as a measurement tool, attached to a ruler serving as a pendulum. The ruler is fixed at a pivot point located 27 cm from its center of mass. The damping constant (δ) is derived through Python data-fitting methods to account for frictional effects in the calculations of $\vec{a}$ and $\vec{a_g}$. The results show that both the calculated and measured values of $\vec{a}$ and $\vec{a_g}$ align with the expected behavior based on the fundamental principles of pendulum motion, reinforcing the preliminary understanding of the system.

1 Introduction

In this experiment we use our smartphone to measure key parameters of a physical pendulum.

Namely:

- Angular velocity ($\vec{\omega_z}$)
- Acceleration ($\vec{a}$)
- Acceleration Without g ~ Linear Acceleration ($\vec{a_g}$).

The primary goals of the experiment are to determine the frequency of the pendulum and to illustrate the effect of friction in order to determine the damping constant, and calculate the theoratical values to compare them to the experimental ones of acceleration. All in all to show insides of how the physical pendulum works and compare the accuracy of the calculated values over the measured ones. The pendulum was set up using a phone as the mass point, while running an application 'PhyPhox' to measure the accelertion/(s) and angular velocity of the phone in it's frame of reference.

2 Theoretical Background

2.1 Equation of Motion of Mathematical Pendulum

The derivations for equations **(1)** to **(10)** below were taken from the Professor's Provided Aid (H01ePendulum as an accelerated Frame of Reference)

Before deriving the equation of acceleration utilised in our research, we need to derive a more concrete simple case of the system ~ the Mathematical Pendulum

For a Mathematical Pendulum, the effects of friction and viscosity of air are neglacted and assumed that the system does not lose energy from its initial release. With the assumption that the pendulum has a Moment of Inertia of J_p, with respect to the point of rotation (which is set up as l distance away from the centre of mass of our oscillating object) and taking into account that the change in angular moment, L, equals to the torque of the system, M, the equation of motion can be stated as:

$$J_p\phi = -mgl\sin\phi \tag{1}$$

.

Where $L = J_p\dot{\phi}$ and the torque is given by, $M = -mglsin(\phi)$

Solving this nonlinear equation, while taking $\vec{\omega}_0^2 = \frac{mgl}{J_p}$ and a small angle approximation of $sin(\phi) \approx \phi$, we get the equation of motion of a harmonic oscillator as:

$$\ddot{\phi} + \omega_0^2\phi = 0 \tag{2}$$

$$\phi = \phi_0 cos(\omega_0 t + \beta)) \tag{3}$$

, where β is some phase change which we can take to be zero. From our $\vec{\omega}_0$, we can extrapolate our amplitude frequency (the eigenfrequency) as $f_{md} = \frac{\omega_0}{2\pi}$.

The accelerations, $\vec{a}$ or $\vec{a}_g$ (where the latter is just the former but without $\vec{g}$), derived for this ideal set up is due to the acceleration of gravity, $\vec{g}$, and additional accelerations from the accelerated frame of reference of the smartphone. Which namely are the Coriolois acceleration, $\vec{a}_c$, the centrifugal acceleration $\vec{acf}$, and the Euler acceleration $\vec{a}_\alpha$. The acceleration measured by the smartphone in the end would be:

$$\vec{a} = \vec{g} + \vec{a}_{cf} + \vec{a}_\alpha = \vec{g} - \vec{\omega} \times (\vec{\omega} \times \vec{r}) - \vec{\alpha} \times \vec{r} \tag{4}$$

The vector $\vec{r}$ would be the length from the Point of Rotation and the Centre of Mass of our oscillating object (the phone). However, other than the acceleration formula, only f_{md} will be utilised from this Ideal Pendulum as a comparative tool. While the acceleration equation will be expandted upon, taking into account fricition, in the below section.

2.2 Equation of Motion of Real Pendulum

For a real pendulum, the above **equation (3)**, can amended by taking into account the influence of viscous drag, as such the influence of friction due to air, by adding a δ component for the expected exponential decrease in amplitude of a pendulum. Yielding:

$$\ddot{\phi} + 2\delta\dot{(\phi)} + \omega_0^2\phi = 0 \tag{5}$$

Which, when solved, provides an exponentially damped amplitude (i.e equation of motion for our Real Pendulum):

$$\phi = \phi_0 exp(-\delta t) cos(2\pi f_d + \beta)) \tag{6}$$

And a shift frequency of $f_d = \frac{\sqrt{\omega_0^2 - \delta^2}}{2\pi}$. The value of δ will be determined using a curve_fit() function in python, utilising the measured angular velocity from our experiment.

Utilising equation **(5)**, with our paramaters $\vec{r}$, $\vec{\omega}$ and additionally $\vec{\alpha}$ set as:

$$\vec{r} = \begin{pmatrix} 0 \\ -l \\ 0 \end{pmatrix} \tag{7}$$

$$\vec{\omega} = \begin{pmatrix} 0 \\ 0 \\ \omega \end{pmatrix} \tag{8}$$

$$\vec{\alpha} = \begin{pmatrix} 0 \\ 0 \\ \alpha \end{pmatrix} \tag{9}$$

Note: From the orientation of our phone, the $\vec{\omega}$ and $\vec{\alpha}$ would be along the z-axis, and as such the final cross multiplied result for our acceleration being:

$$\vec{a} = \begin{pmatrix} g\sin\phi + l\alpha \\ -g\cos\phi \ - l\omega^2 \\ 0 \end{pmatrix} \tag{10}$$

For equation **(10)**, taking the second derivative of our equation of motion **(6)**, $\frac{d^2\phi}{(d\phi)^2}$, provides us with the $\vec{\alpha}$.

$$\vec{\alpha} = \phi_0 e^{-\delta t} \left(2\delta\sqrt{\omega_0^2 - \delta^2}\ \sin(\sqrt{\omega_0^2 - \delta^2}\ t) + (2\delta^2 - \omega_0^2)\cos(\sqrt{\omega_0^2 - \delta^2}\ t) \right) \tag{11}$$

The fact that we only need the z component of angular velocity means we will be only utilising the z-component of our angular velocity for further calculations.

3 Physical Exploration

3.1 Materials

- Phone with the application PhyPhox installed
- 30 cm Ruler
- Tape
- A stick like aparatus for centre of rotation, i.e the crochet sticks seen in [Image 1] and [Image 2]

3.1.1 3.2 Set Up

In order to build our physical pendulum we use a Smartphone ($0.193\text{Kg} \pm 0.001\text{Kg}$) which is taped to a ruler ($0.007\text{Kg} \pm 0.001\text{Kg}$) , so that the lower edge of the phone is ($0.270\text{m} \pm 0.005\text{m}$) away from the fixed pivot point. The ruler acts as our pendulum arm by inserting a wooden crochet hook through the top hole in the ruler, which has a slightly smaller diameter than the hole, which allows the ruler to rotate freely.The crochet hook is held flat on the table near the edge so that the ruler hangs down vertically with the phone attached to it.

[53]:
```python
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
from scipy.optimize import curve_fit
import warnings
from IPython.display import display, Image
import matplotlib.image as mpimg
warnings.filterwarnings('ignore')
```

[40]:
```python
#nbconverter was not able to render images in markdown, latex or html when␣
↪converting to pdf. As such python was used to print images
img1 = mpimg.imread("set_up_1.png")
img2 = mpimg.imread("set_up_2.png")
fig, axes = plt.subplots(1, 2, figsize=(10, 5))
axes[0].imshow(img1)
axes[0].axis("off")
axes[0].set_title("Image 1", y=-0.09)
axes[1].imshow(img2)
axes[1].axis("off")
axes[1].set_title("Image 2", y=-0.09)
plt.tight_layout()
plt.show()
```

Image 1

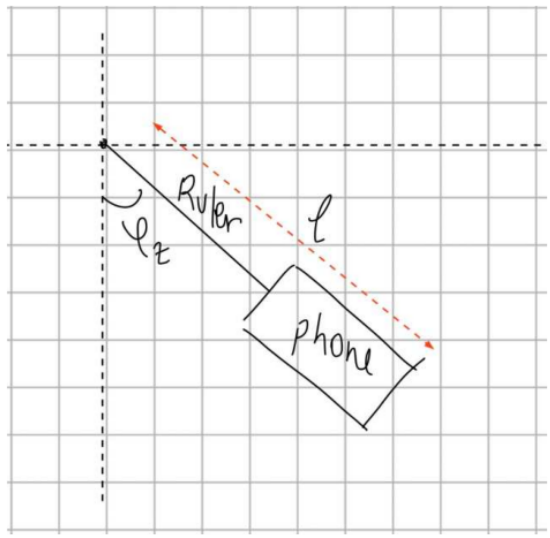
Image 2

The pendulum (smartphone attached to the ruler) is held at 45 degrees to the ruler. To measure the parameters we utilised the app Phyphox which measures the components of the angular acceleration and angular velocity. After we left go of the phone, the phone starts to oscillate, with a clearly visible damping effect taking place due to the viscousity of air. The initial effects of friction is strong and rapid, with a much sharper decrease, but as time goes on and the pendulum is closer to the standstill . Decrease isn't as noticable anymore. We measure the data for a minute. Since the release does not happen at the same time as the measurement starts, our 3.5 seconds (release time) is used as the initial start time, running until 63.5 seconds ((60s $\pm$ 0.01s $\sim$ 1 minute).

A diagram of the set up with the axis relative to a stationary point of view and the angle to which the angular velocity is measured is presented below.

[41]:

```
#nbconverter was not able to render images in markdown, latex or html when␣
↪converting to pdf. As such python was used to print images
img1 = mpimg.imread("diagram_1.jpg")
img2 = mpimg.imread("diagram_2.png")
fig, axes = plt.subplots(1, 2, figsize=(10, 5))
axes[0].imshow(img1)
axes[0].axis("off")
axes[0].set_title("Graph 1",  y=-0.09)
axes[1].imshow(img2)
axes[1].axis("off")
axes[1].set_title("Image 3",  y=-0.09)
plt.tight_layout()
plt.show()
```



Graph 1

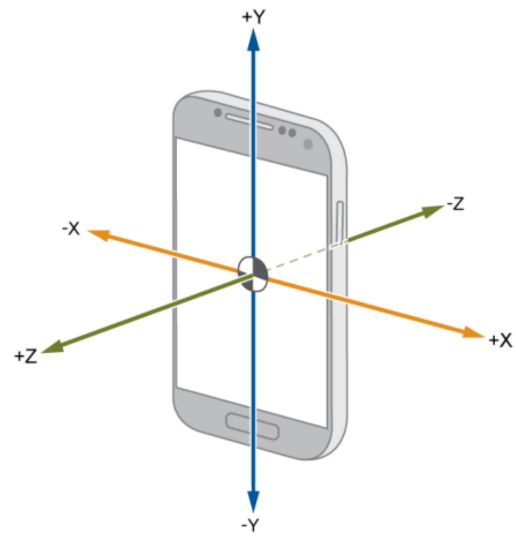


Image 3

Some specific values, which will be required in further calculations:

$J_p = 0.014 \pm 0.0006485 kgm^2$, retrieved by applying Parallel Axis Theorem on the rotating mass (phone).

$\omega_0 = 6.022 \pm 0.00001 \frac{raddians}{sec}$, retrieved from the data base at the time element of 3.5 seconds, which corresponds to the first peak.

4 Results

4.1 Data Collection and determining δ and f_d

4.1.1 Data Collection and determining δ

For the determining of δ, the curve_fit() function from the Python scipy library was utilised. The results of the graphing of δ determination fitting and the data is seen below in Figure 1.

Figure 1 (a) presents the Angular Velocity (along the z-axis, as stated in the derivation of acceleration), $\omega_z(Hz)$ against Time, t(s). While Figure 1 (b) in the subplot presents a data fitting algorithim to find a suitable δ for our database of $\omega_z(Hz)$. The actual exact fitting of the data to cover each point is ignored, as such exact solution leads to the commutaional limit of the python code. However, the retrieved value for δ is directly graphed on an exponential curve to see the accuracy of how good is damping coefficient relative to the actual data base.

In setting up the code, the approximation of $\exp(-\delta t) \approx (1 - \delta t)$ has been used, as on a closer inspection, the exponential decrease of the oscillation is nearly linear. This set up leads to a more accurace retrival of a delta function. Additionally, the time range of 3.45 to 63.5 seconds (an additional 0.05 seconds from the one minute mark stated earlier) was used as the red fitting line was barely covering any amount of data within the one minute time scope without the additional 0.05 seconds. We could not figure out the exact reasoning behind this issue, and we suspsect is part of how curve_fit() algorithim processess the given initial guess and the provided equation.

However, for any further calcualtions, only the values of ω_z between 3.5 and 63.5 seconds were used. .

If any information regarding the ω_x, ω_y and $|\omega|$ are required, the graphs can be found in the **Section 8**. As they are irrelevant to the current needed set of values, they are ignored here as to remove clutter.

[55]:
```
#We are taking only the z axis as only the angle agaoinst the z axis is␣
↪changing when it is rotating in the manner seen in the diagram
data_gyroscope = r"Gyroscope.csv"
phi_0 = np.radians(45)
g=9.81

data_gy_evaluated = pd.read_csv(data_gyroscope)
#filetring data within the scope that is when the pendulum was run
filtered_data_gyroscope = data_gy_evaluated[(data_gy_evaluated.iloc[:, 0] >= 3.
↪45) & (data_gy_evaluated.iloc[:, 0] <= 63.5)]

t = filtered_data_gyroscope.iloc[:, 0]  #time
av_x = filtered_data_gyroscope.iloc[:, 1]  #x-axis
av_y = filtered_data_gyroscope.iloc[:, 2]  #y-axis
av_z = filtered_data_gyroscope.iloc[:, 3]  #z-axis

def delta_fit(time, delta, omega):
    omega_delta_fit = (1-delta*time) * phi_0 * (-delta * np.cos((omega * time))␣
↪- omega * np.sin(omega * time))
    return omega_delta_fit

initial_guess = [0.001, 6]
params, covariance = curve_fit(delta_fit, t, av_z, p0=initial_guess)
delta_test, omega_fit = params

SE = np.sqrt(np.diag(covariance))
SE_A = SE[0]

print("The value for our delta: ", delta_test)
print(F'Standard error of {SE_A:.5f}.')
fit = delta_fit(t, delta_test, omega_fit)
fig, (ax1, ax2) = plt.subplots(2, figsize=(10, 6))
fig.suptitle('Angular Velocity, ${\omega_z}$ against Time, t(s)')
ax1.plot(t, av_z, linestyle='dotted', color='b',  markersize=5, label="Measured␣
↪Data, ${\omega_z}$")
ax1.set_xlabel('Time, t(s)')
ax1.set_ylabel('Angular Velocity, ${\omega_z}~~(Hz)$')
ax1.legend(loc='upper right')
ax2.plot(t, av_z, linestyle='dotted', color='b',  markersize=5, label="Measured␣
↪Data")
```

```
ax2.set_xlabel('Time, t(s)')
ax2.set_ylabel('Angular Velocity, ${\omega_z}~~(Hz)$')
ax2.plot(t, -fit, linestyle='-', color='r', label="Fitted Equation to find ${\delta}$" )
ax2.plot(t, 7*np.exp(-delta_test*t), color='g', label="${\delta}$ validation")
ax2.legend()
ax2.grid(True)
ax1.grid(True)
plt.show()
fig.tight_layout()
```

```
The value for our delta:  0.026236885964776028
Standard error of 0.00014.
```

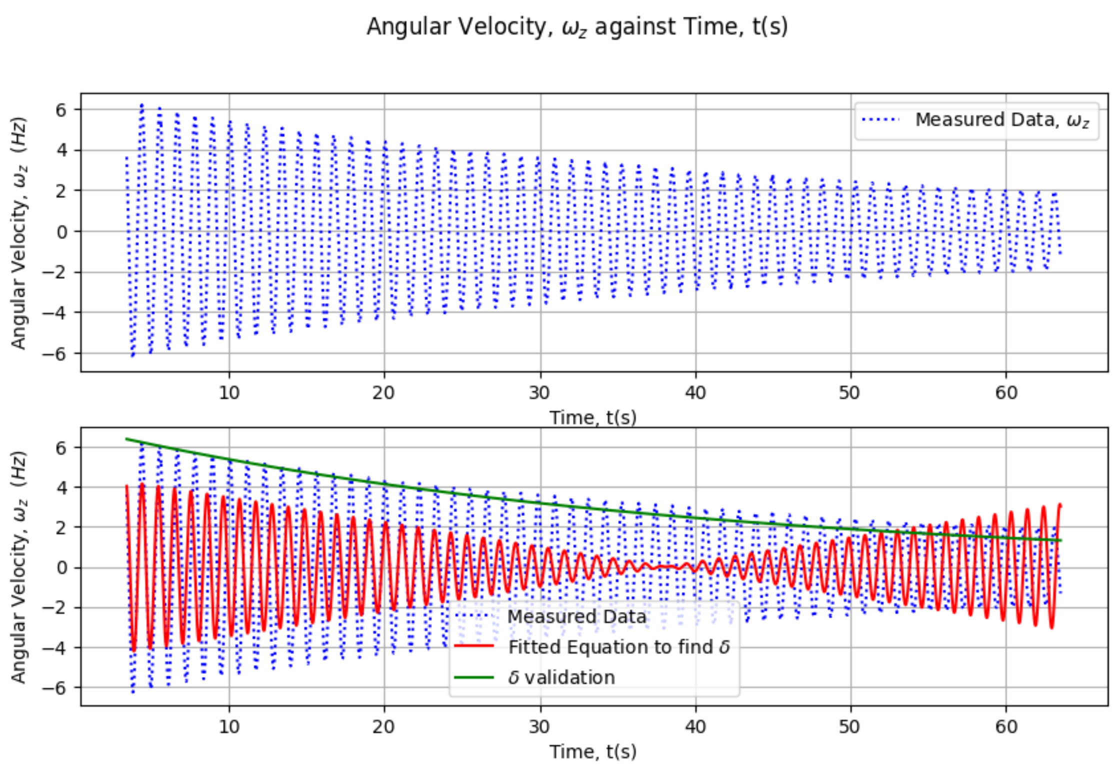


Figure 2: (a) Angular Velocity ω_z against Time, t(s) (b) δ Approximation over the former graph

As explained before, the second subplot is used to determine δ for our data base of angualr velocity, and as such, the code yields a δ of $= 0.026236885964776028 \approx 0.026 \pm 0.00014$. Which seen applied with the green line of Figure 1 (b), and is nearly close to the actual exponential decrease of the oscillation of our pendulum.

4.1.2 Determing f_d:

The mentioned value of Eigenfrequency, or more well understood as the Pendulum frequency, can be utilised to determine the damping constant of our system. However, as we already are determining such value through a curve fitting algaorithim, we are going to use this absolute numerical calcualtions as a means to compare our found δ from before to a more concrete δ found through the below calculations. The f_d is determined using the equtions briefly mentioned in Section 2. The equations, for the theoratical (Natural Frequency) will be calculted through the equation:

$$f_{md_t} = \frac{\omega_0}{2\pi} = \frac{1}{2\pi}\sqrt{\frac{mgl}{J_p}} \tag{12}$$

while the Pendulum's Damped Natural Frequency for our system, will be determined through:

$$f_{md_e} = \frac{\sqrt{\omega_0^2 - \delta^2}}{2\pi} \approx \frac{\omega_0}{2\pi} \tag{13}$$

f_{md_e} has been estimated towards a just an expression in terms of ω_0 (as, seen from the ω_z graph the damping is close to linear), as a means to find the Damped Natural Frequency. The extent of the accuracy of this estimation is acknolwedged, but as other means to calcualte the natural damped frequency were not feasible (e.g the PhyPhox did not provide an option to measure the acceleration(s), angular velocity and *frqeuency* all in one run), we chose to do such approximation to be able to do the comparative analysis. Applying them, we get the respective values of $f_{md_e} = 0.954929658551 \approx 0.955$ Hz and $f_{md_t} = 0.958801317038 \approx 0.959$ Hz.

With these known values, we determined the damping constant by using the damping ratio $\zeta = \frac{\delta}{2mf_{md_t}}$ with the relationship between f_{md_t} and f_{md_e} as $f_{md_e} = f_{md_t}\sqrt{1-\zeta}$. Which, if plugged in and solved for δ, yielded $0.00245655438474 \approx 0.0245$, a value close to our δ seen from the fitting algorithim.

Uncertainity for our f_{md_e} is derived from the smallest possible measurement of the angular velocity, as only that value is measurable data from the calculation. As such $f_{md_e} = 0.955 \pm (0.00001)$. While the uncertainity in f_{md_t} is culculated using the uncertainity arithimitics with the already mentioned uncertainties (from Section 3). As such $f_{md_t} = 0.959 \pm (0.0415) Hz$

4.1.3 The Acceleration (Measured):

Utilising the data retrived from the experiment, we can also graph the measured $\vec{a}$ and $\vec{a}_g$, which will be utilised in the Analysis section to compare to the calculated acceleration(s). The repsective uncertainities are shown on the label axis

The Absolute Acceleration for $\vec{a}\frac{m}{s^2}$ and $\vec{a}_g\frac{m}{s^2}$

[57]:
```
data_accelerometer = pd.read_csv(r'Accelerometer.csv')

#filetring data within the scope that is when the pendulum was run
filtered_data_acceleration = data_accelerometer[(data_accelerometer.iloc[:, 0]␣
↪>= 3.5) & (data_accelerometer.iloc[:, 0] <= 63.5)]
```

```python
a_x = filtered_data_acceleration.iloc[:, 1]  # x-axis
a_y = filtered_data_acceleration.iloc[:, 2]  # y-axis
a_z = filtered_data_acceleration.iloc[:, 3]  # z-axis

t2 = filtered_data_acceleration.iloc[:, 0]  # x-axis

absolute_acceleration = np.sqrt(a_x**2 + a_y**2 + a_z**2)
absolute_acceleration_g = absolute_acceleration-g
plt.figure(figsize=(10, 6))
plt.plot(t2, absolute_acceleration, label='Absolute Acceleration', color='b')
plt.plot(t2, absolute_acceleration_g, label='Absolute Acceleration Without G',␣
 ↪color='g')
plt.xlabel('Time, t(s) $\pm 0.001$')
plt.ylabel('Absolute Acceleration, $a\\frac{m}{s^2} \pm 0.00001$')
plt.title('Absolute Acceleration,  $a\\frac{m}{s^2}$ vs. Time, t(s)')
plt.grid()
plt.legend()
plt.show()
```

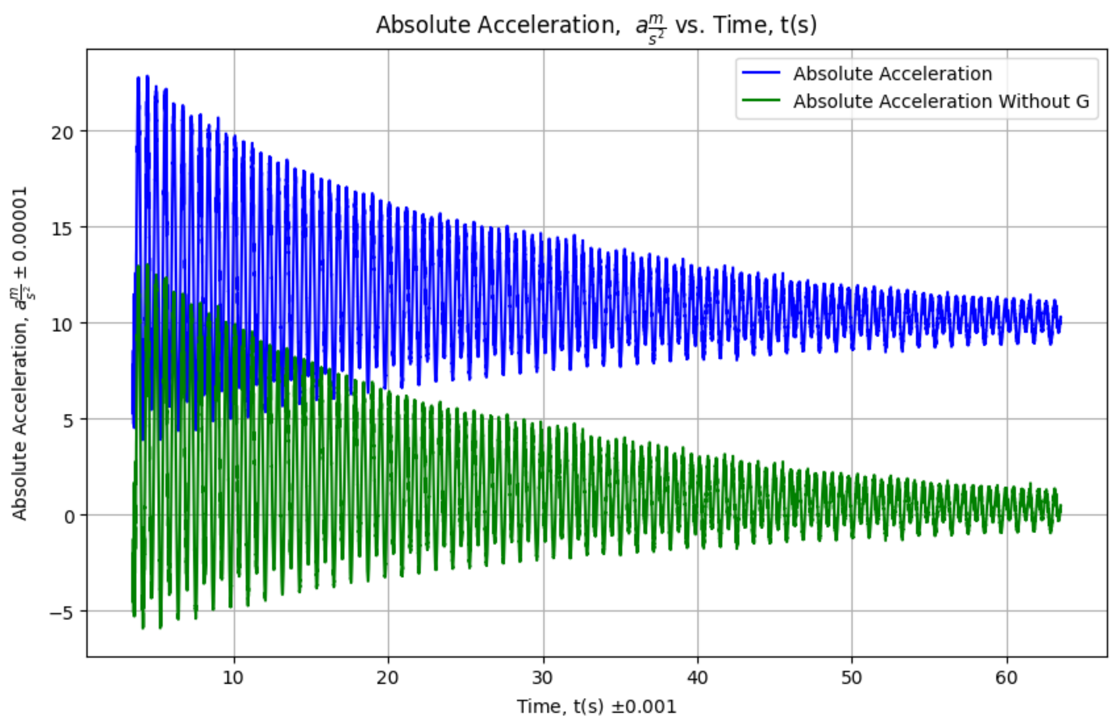


Figure 2 Absolute Acceleration(s) a $\frac{m}{s^2}$ vs. Time, t(s)

The Acceleration for $\vec{a}\frac{m}{s^2}$ and $\vec{a}_g\frac{m}{s^2}$ saperately for each axis (Measured):

[44]:

```
fig, (ax1, ax2, ax3) = plt.subplots(3, figsize=(10, 10))

a_x_g = a_x-g
a_y_g = a_y-g
a_z_g = a_z-g

fig.suptitle('Acceleration (Measured) indivual per axis')
ax1.plot(t2, a_x, label="Acceleration, $\\vec{a_x}\\frac{m}{s^2}")
ax1.plot(t2, a_x_g, color="g", label="Linear Acceleration,␣
↪$\\vec{a_{xg}}\\frac{m}{s^2}$")
ax1.set_xlabel('Time, t(s) $\pm 0.001$')
ax1.set_ylabel('Acceleration, $\\vec{a_x}\\frac{m}{s^2}\pm 0.00001$')
ax1.title.set_text('Measured Acceleration(s) for X-Axis')
ax1.legend()
ax2.plot(t2, a_y, label="Acceleration, $\\vec{a_y}\\frac{m}{s^2}$")
ax2.plot(t2, a_y_g, color="g", label="Linear Acceleration,␣
↪$\\vec{a_{yg}}\\frac{m}{s^2}$")
ax2.set_xlabel('Time, t(s), $\pm 0.001$')
ax2.set_ylabel('Acceleration, $\\vec{a_y}\\frac{m}{s^2} \pm 0.00001$')
ax2.title.set_text('Measured Acceleration for Y-Axis')
ax2.legend()
ax3.plot(t2, a_z, label="Acceleration, $\\vec{a_z}\\frac{m}{s^2}$")
ax3.plot(t2, a_z_g, color="g",  label="Linear Acceleration,␣
↪$\\vec{a_{zg}}\\frac{m}{s^2}$")
ax3.set_xlabel('Time, t(s) $\pm 0.001$')
ax3.set_ylabel('Acceleration, $\\vec{a_z}\\frac{m}{s^2}\pm 0.00001$')
ax3.title.set_text('Measured Acceleration for Z-Axis')
ax3.legend()
fig.tight_layout()

ax2.grid(True)
ax1.grid(True)
ax3.grid(True)
plt.show()
fig.tight_layout()
```

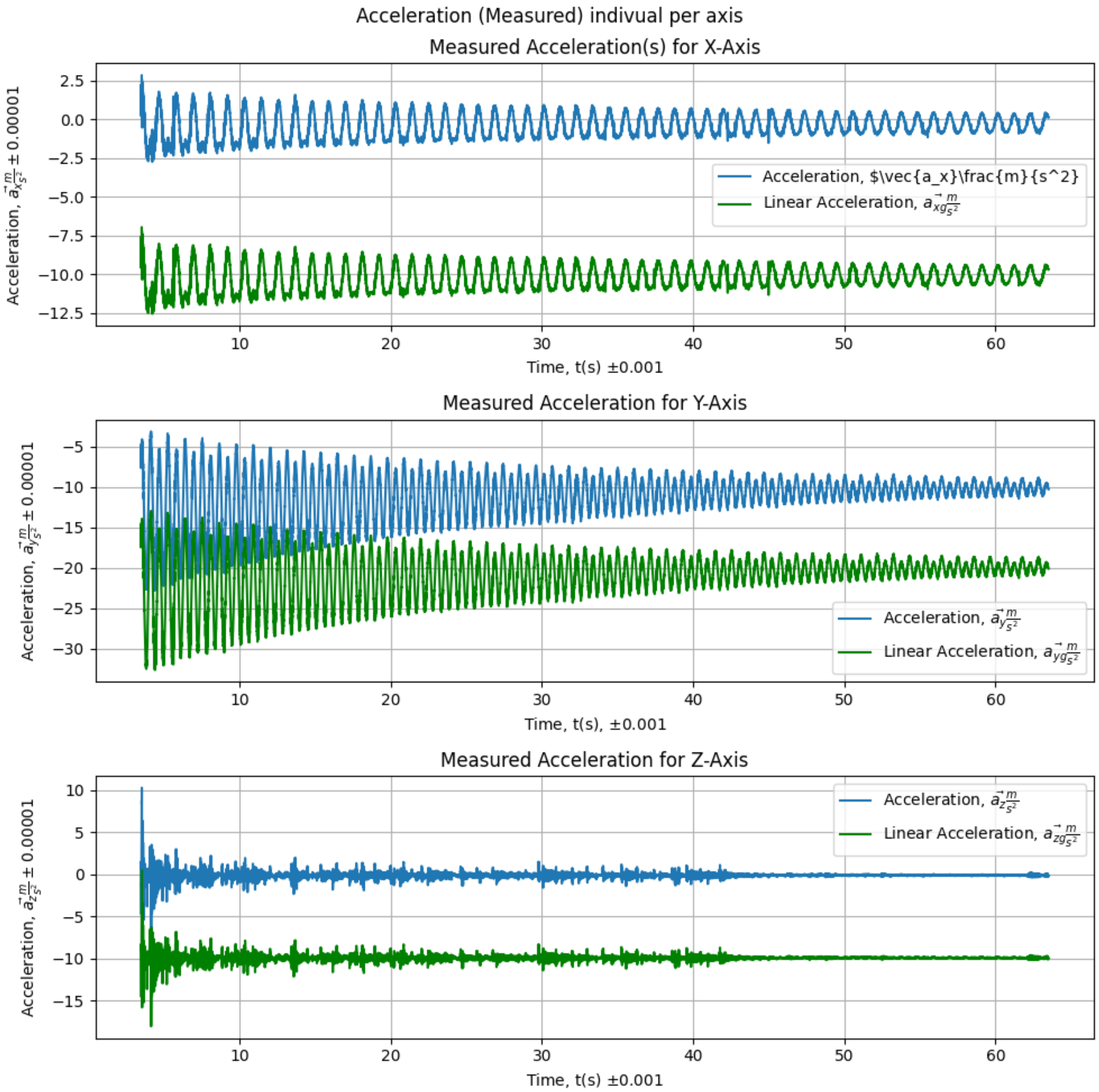


Figure 3: Acceleration(s) (a) $\vec{a}_x\dfrac{m}{s^2}$ (b) $\vec{a}_y\dfrac{m}{s^2}$ (c) $\vec{a}_z\dfrac{m}{s^2}$

4.1.4 The Acceleration (s) Calculated:

Utilising equations (**10**) and (**11**), the acceleration of a system and the angular acceleration respectively, we are able to plot the results for the Calculated Acceleration for both $\vec{a}$ and $\vec{a}_g$. The uncertainities would be as follows: Time, t ±0.001 (s) and Acceleration $\vec{a}_i\frac{m}{s^2} \pm 0.00001$. As the error margins are very small relative to the graph, error bars cannot be graphed to be seen properly.

[45]:
```
g = 9.81
m = 0.2 #+-0.001
l = 0.27 #+-0.005
```

```
phi_0 = np.radians(45)
delta = 0.002

filtered_data_gyroscope2 = data_gy_evaluated[(data_gy_evaluated.iloc[:, 0] >= 3.
↪5) & (data_gy_evaluated.iloc[:, 0] <= 63.5)]
omega_z = filtered_data_gyroscope2.iloc[:, 3].to_numpy()
t_values = np.linspace(3.5, 63.5, num=28115)

omega_0 = omega_z[0]

acceleration_values = []

for i,t in enumerate(t_values):
    omega = omega_z[i]
    sh1 = np.sqrt(omega_0**2 - delta**2)
    phi = phi_0*np.exp(-delta*t)*np.cos(sh1*t)
    #alpha = (-1*((m*g*l)/(mi)))*np.sin(phi)
    alpha = phi_0*np.exp(-delta*t)*(2*delta*sh1*np.sin(sh1*t) +␣
↪(((2*delta**2)-omega_0**2)*np.cos(sh1*t)))
    a_x_c = g * np.sin(phi) + (l*alpha)
    a_y_c = -g * np.cos(phi) - (l*omega**2)
    a_z_c = 0
    acceleration_values.append((a_x_c, a_y_c, a_z_c))

acceleration_calculated = pd.DataFrame(acceleration_values, columns=['a_x_c',␣
↪'a_y_c', 'a_z_c'])
acceleration_calculated['time'] = t_values

acceleration_calculated['absolute_acceleration'] = np.
↪sqrt(acceleration_calculated['a_x_c']**2 +␣
↪acceleration_calculated['a_y_c']**2 + acceleration_calculated['a_z_c']**2)
acceleration_calculated_withoutg = acceleration_calculated - g

plt.figure(figsize=(10, 6))
plt.plot(acceleration_calculated['time'],␣
↪acceleration_calculated['absolute_acceleration'], label='Absolute␣
↪Acceleration', color='b')
plt.plot(acceleration_calculated['time'],␣
↪acceleration_calculated_withoutg['absolute_acceleration'], label='Absolute␣
↪Linear Acceleration', color='g')
plt.xlabel('Time, t(s), $\pm 0.001$')
plt.ylabel('Absolute Calculated Acceleration, $a\\frac{m}{s^2}\pm 0.00001$')
plt.title('Absolute Calculated Acceleration, $a\\frac{m}{s^2}$ vs. Time, t(s)')
plt.grid()
plt.legend()
plt.xlim(1, 60)
```

```
plt.gca().yaxis.set_major_locator(MaxNLocator(nbins=10))
plt.show()
```

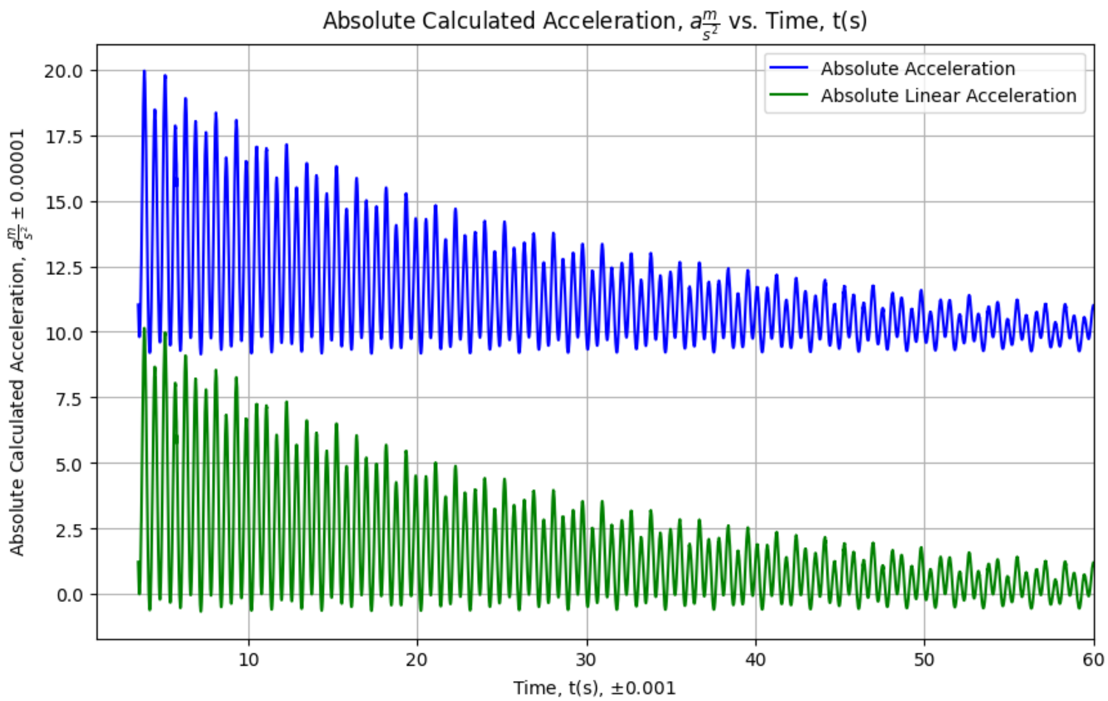


Figure 4 Absolute Calculated Acceleration(s) a $\frac{m}{s^2}$ vs. Time, t(s)

[46]:
```
a_cx_g = acceleration_calculated['a_x_c']-g
a_cy_g = acceleration_calculated['a_y_c']-g
a_cz_g = acceleration_calculated['a_z_c']-g

fig, (ax_c1, ax_c2, ax_c3) = plt.subplots(3, figsize=(10, 10))
fig.suptitle('Acceleration (Calculated) indivual per axis')
ax_c1.plot(acceleration_calculated['time'], acceleration_calculated['a_x_c'],␣
↪label="Calculated Acceleration, $\\vec{a_x}\\frac{m}{s^2}$")
ax_c1.plot(acceleration_calculated['time'], a_cx_g, label="Linear Calculated␣
↪Acceleration, $\\vec{a_{xg}}\\frac{m}{s^2}$", color='g')
ax_c1.title.set_text('Calculated Acceleration for X-Axis')
ax_c1.set_xlabel('Time, t(s)')
ax_c1.set_ylabel('Calculated Acceleration, $\\vec{a_x}\\frac{m}{s^2}$')
ax_c1.legend()
ax_c2.plot(acceleration_calculated['time'], acceleration_calculated['a_y_c'],␣
↪label="Calculated Acceleration, $\\vec{a_x}\\frac{m}{s^2}$")
ax_c2.plot(acceleration_calculated['time'], a_cy_g,  label="Linear Calculated␣
↪Acceleration, $\\vec{a_{xg}}\\frac{m}{s^2}$", color='g')
```

```
ax_c2.title.set_text('Calculated Acceleration for Y-Axis')
ax_c2.set_xlabel('Time, t(s)')
ax_c2.set_ylabel('Calculated Acceleration, $\\vec{a_y}\\frac{m}{s^2}$')
ax_c2.legend()
ax_c3.plot(acceleration_calculated['time'], acceleration_calculated['a_z_c'],label="Calculated Acceleration, $\\vec{a_x}\\frac{m}{s^2}$")
ax_c3.plot(acceleration_calculated['time'], a_cz_g,  label="Linear Calculated Acceleration, $\\vec{a_{xg}}\\frac{m}{s^2}$", color='g')
ax_c3.title.set_text('Calculated Acceleration for Z-Axis')
ax_c3.set_xlabel('Time, t(s)')
ax_c3.set_ylabel('Calculated Acceleration, $\\vec{a_x}\\frac{m}{s^2}$')
ax_c3.legend()
fig.tight_layout()

ax_c1.grid(True)
ax_c2.grid(True)
ax_c3.grid(True)
plt.show()
fig.tight_layout()
```

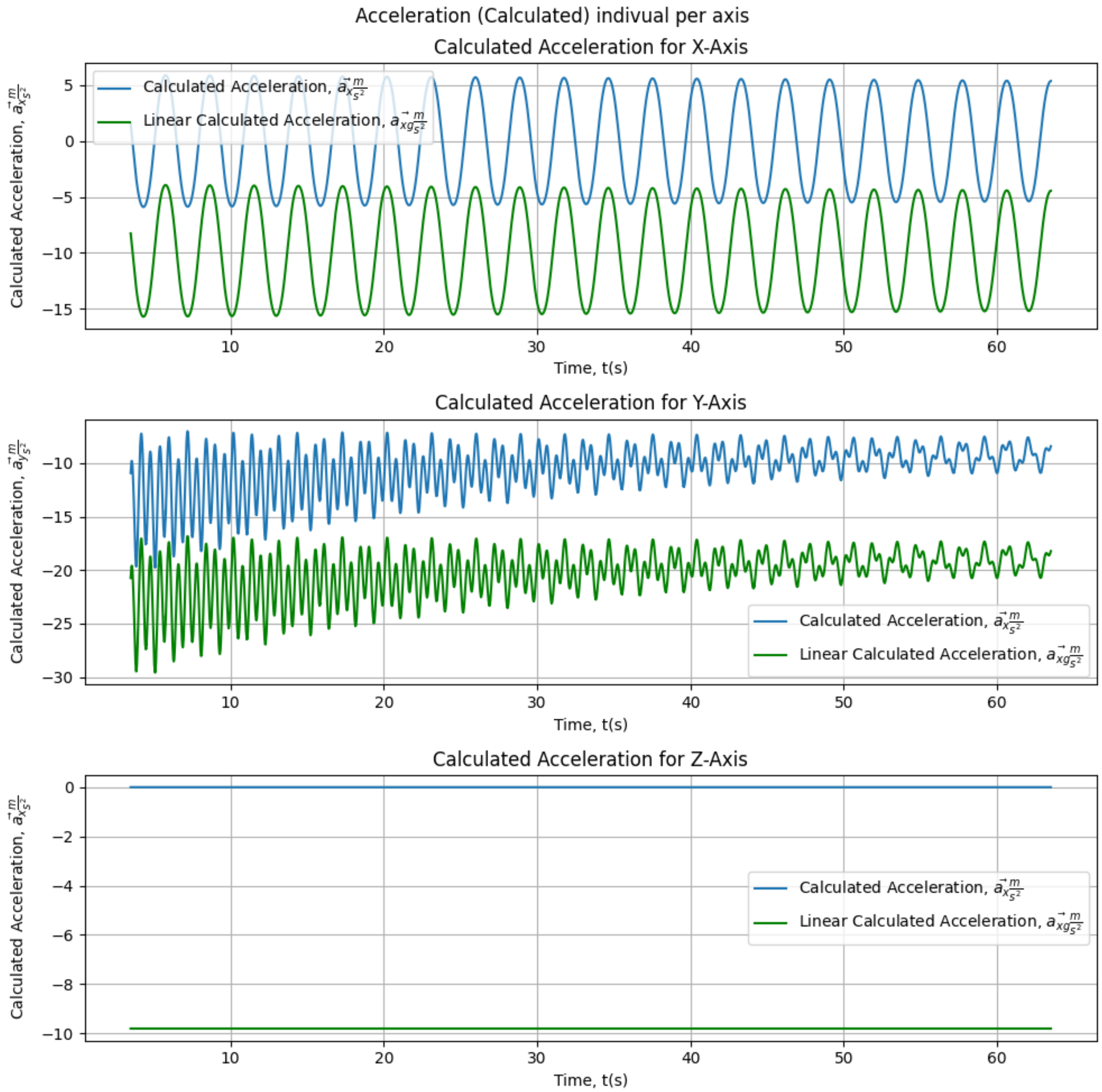


Figure 5: Acceleration(s) (a) $\vec{a}_x \dfrac{m}{s^2}$ (b) $\vec{a}_y \dfrac{m}{s^2}$ (c) $\vec{a}_z \dfrac{m}{s^2}$

4.2 Analysis

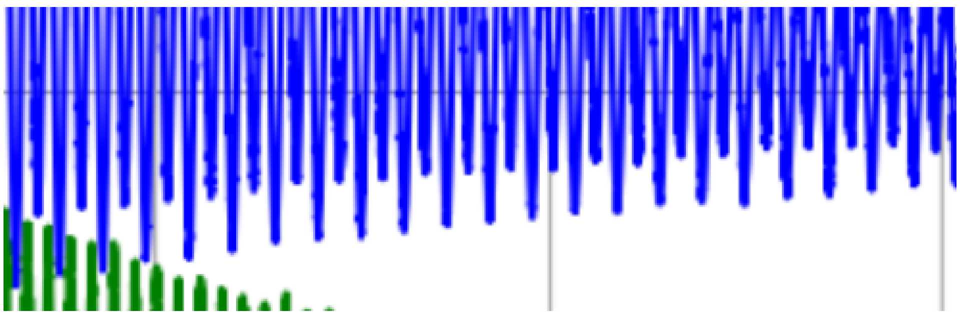
4.2.1 Accuracy of $\vec{a}$ and $\vec{a_g}$:

Our Measured Absolute Acceleration Graph **(Figure 2)** has the typical form of damped oscillation. These oscillations are characterized by a decreasing amplitude over time, indicating that energy is being lost in the system – in this case due to the viscous drag(air resistance). Air resistance counteracts motion and continuously drains energy from the system. Initially, the acceleration is relatively high, but due to air resistance, the amplitude decreases with each oscillation until the motion finally comes to a standstill. The theoretical course of the oscillation curve therefore shows an exponential decrease in the amplitude while the frequency of the oscillation is remaining constant.

Moreover, there seems to be a pattern of that every second lower trough of our graph, if we divide the troughs into pairs of two, seen for both $\vec{a}$ and $\vec{a_g}$, seems to be a little less then the prevoius one. After each pair, the trough again rises, and has this rugged pattern, as seen in **Graph 2** below:

[47]:
```
image_path = 'diagram_3.png'
img = mpimg.imread(image_path)
plt.imshow(img)
plt.axis('off')
plt.title("Graph 2")
plt.show()
```

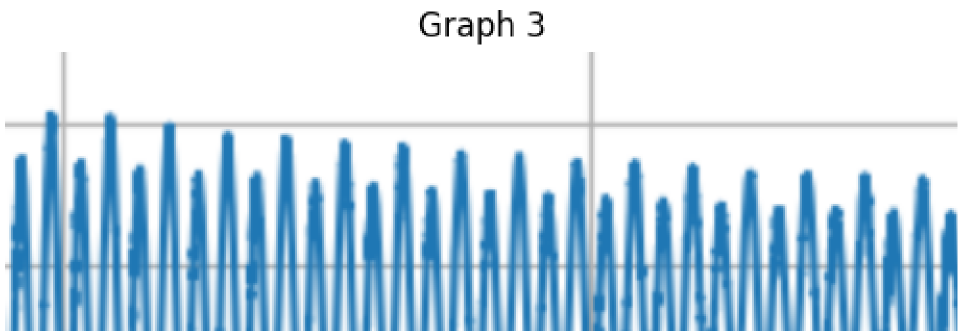
Graph 2



However, such patterns are also visible in the Y-axis renderation of our Measured Acceleration(s), as seen in **Figure 3 (b)**, which makes us believe that original issue from the absolute acceleration is emerging from the y component of our dataset.

The Caclaulted Acceleratoin Graph (**Figure 4**) values also present similar behaviour to the measured oscillation pattern, with an exponentially decreasing pattern, but with one crucial difference: the lower range of oscillation remains close to 0, and only the upper line of oscillation decreases exponentially. This is due to the absolute acceleration computation, which squares the negatives into positives (which also produces the pair wise lower and up patern ~ seen in **Graph 3**. This shifts the originally symmetrical sinusoidal oscillation completely into the positive region of the graph. This results in a curve that only the upper part represents the damped oscillation,with the decreasing amplitude.

[48]:
```
image_path = 'diagram_4.png'
img = mpimg.imread(image_path)
plt.imshow(img)
plt.axis('off')
plt.title("Graph 3")
plt.show()
```



For an Analysis of each indivual axis, we can overlay the Measured over the Calculated Values to see how close are the two sets, in terms of behavior but also just in magnitude. The first set, **Figure 6** below, presents the comparison between $\vec{a}$ for calculated and measured. The uncertainities would be as follows: Time, t ± 0.001 (s) and Acceleration $\vec{a}_i \frac{m}{s^2} \pm 0.00001$. As the error margins are very small relative to the graph, error bars cannot be graphed to be seen properly.

[49]:
```
fig, (a01, ax_o1, ax_o2, ax_o3) = plt.subplots(4, figsize=(10, 10))
fig.suptitle('Acceleration(s)')
a01.plot(t2, absolute_acceleration, label='Measured Acceleration', color='c')
a01.plot(acceleration_calculated['time'],␣
 ↪acceleration_calculated['absolute_acceleration'], label='Calculated␣
 ↪Acceleration', color='r')
a01.legend()
a01.set_ylabel('Absolute Calculated Acceleration, $a\\frac{m}{s^2} \pm 0.
 ↪00001$')
a01.set_xlabel('Time, t(s) $\pm 0.001$')

ax_o1.plot(t2, a_x, label=" Measured Acceleration, $\\vec{a_x}\\frac{m}{s^2}$",␣
 ↪color='c')
ax_o1.plot(acceleration_calculated['time'], acceleration_calculated['a_x_c'],␣
 ↪label="Calculated Acceleration, $\\vec{a_x}\\frac{m}{s^2}$", color='r')
ax_o1.set_xlabel('Time, t(s) ')
ax_o1.set_ylabel('Acceleration, $\\vec{a_x}\\frac{m}{s^2}$')
ax_o1.legend()

ax_o2.plot(t2, a_y, label=" Measured Acceleration, $\\vec{a_y}\\frac{m}{s^2}$",␣
 ↪color='c')
ax_o2.plot(acceleration_calculated['time'], acceleration_calculated['a_y_c'],␣
 ↪label="Calculated Acceleration, $\\vec{a_y}\\frac{m}{s^2}$", color='r')
ax_o2.set_xlabel('Time, t(s)')
ax_o2.set_ylabel('Acceleration, $\\vec{a_y}\\frac{m}{s^2}$')
ax_o2.legend()
```

```
ax_o3.plot(t2, a_z, label=" Measured Acceleration, $\\vec{a_z}\\frac{m}{s^2}$",␣
↪color='c')
ax_o3.plot(acceleration_calculated['time'], acceleration_calculated['a_z_c'],␣
↪label="Calculated Acceleration, $\\vec{a_z}\\frac{m}{s^2}$", color='r')
ax_o3.set_xlabel('Time, t(s)')
ax_o3.set_ylabel('Acceleration, $\\vec{a_z}\\frac{m}{s^2}$')
ax_o3.legend()

ax_o1.grid(True)
a01.grid(True)
ax_o2.grid(True)
ax_o3.grid(True)
fig.tight_layout()
plt.show()
```

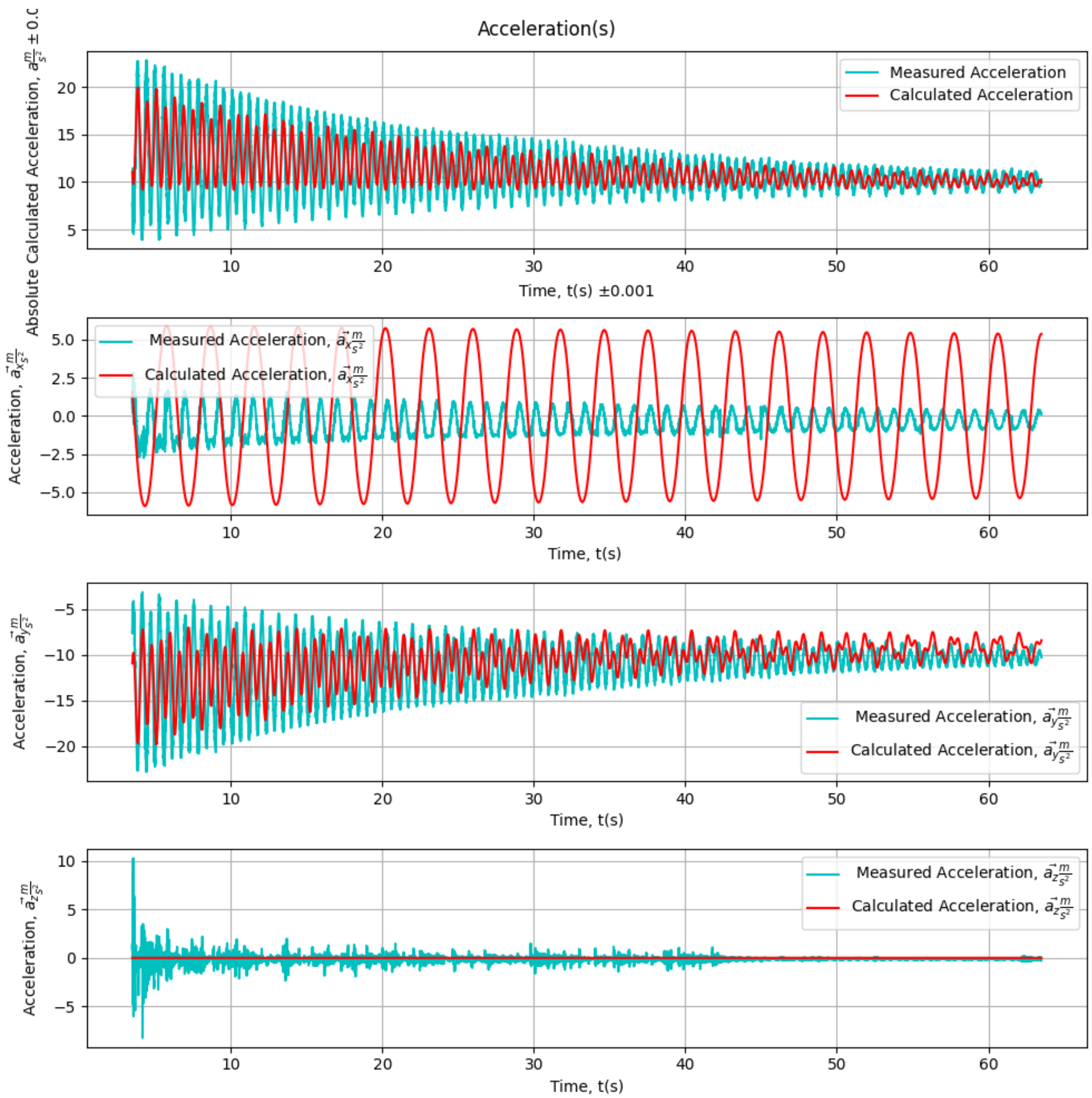


Figure 6: Acceleration(s) (a) |a| $\frac{m}{s^2}$ (b) $\vec{a_x}\frac{m}{s^2}$ (c) $\vec{a_y}\frac{m}{s^2}$ (d) $\vec{a_z}\frac{m}{s^2}$

As we can observe from our Figure 6, we can see that the general trend along each axis, including the absolute accelerations, are behaving in the similiar manner.

- For the absolute acceleration, |a|, there is this general slow degration towards the value of 10, which would be a close approximate of the magnitude of gravity. This makes sense, as seen from Images 1 and 2, and from Graph 1, the orientation of our phone yields a set up where the angle of measurements is against the z-axis, meaning the resultant acceleration would be towards the Y axis, where we experience the influence of gravity the most. The actual values are not consistent for the initial values, however the difference decreases as the pendulum starts to slow down.
- $\vec{a_x}$ $\vec{a_y}$ $\vec{a_z}$ $\frac{m}{s^2}$ Similair trend, values far apart initially but becomes closer to each other as the pendulum progresses and slows down. But the overal behavior is similiar. $\vec{a_z}$ stays close tries to stay close to zero (for the Measured Acceleration) while the Calculated Acceleration is consistently zero). $\vec{a_y}$ oscillated towards the value 10, as which is to the absolute acceleration. $\vec{a_x}$ oscillated with a more consistenten frequency change and a more stead decreasein ampltiude. Which maps the the back and forth movement of the phone as it rotates in the same plane as from a stationary viewpoint.

Possible Explanation Our calculated acceleration values were derived through some approximation, such as namely the small angle approximation. As our starting angle was $\frac{\pi}{4}$, which in retrospect is not a small angle, the similiarity between the calculated and measured accelerations become prevelent only when the measured values reach an oscillation spectrum where a small angle approximation is more accurate. Furthermore, the calcualted accelerations do not take into account many more lose of energy due to other sources (heat and sound) and additionally cannot take into account that the phone does not stay along a perfect oscillating line, but deviates away. These additional variables in the end end up causing the numerical differences between the two sets of data, even if the behaviorial trends are similiar.

On a lower scale of error influence, the mass distribution of the smartphone is not ideal, and the centre of mass is not in reality in the centre of the phone. This would lead to slight misalignment and imperfections in the rotation and acceleration of the mass, leading to possible minor errors.

The results had **Good accuracy but limited precision**

Additionally, here are the graphs for Linear Acceleration comparison. The uncertainities would be as follows: Time, t ± 0.001 (s) and Acceleration $\vec{a_i}\frac{m}{s^2} \pm 0.00001$. As the error margins are very small relative to the graph, error bars cannot be graphed to be seen properly.

[50]:
```
fig, (a01, ax_o1, ax_o2, ax_o3) = plt.subplots(4, figsize=(10, 10))
fig.suptitle('Acceleration(s)')
a01.plot(t2, absolute_acceleration, label='Measured Acceleration', color='c')
```

```
a01.plot(acceleration_calculated['time'],␣
↪acceleration_calculated_withoutg['absolute_acceleration'], label='Calculated␣
↪Acceleration', color='r')
a01.legend()
a01.set_ylabel('Absolute Acceleration, $a\\frac{m}{s^2}$')
a01.set_xlabel('Time, t(s)')

ax_o1.plot(t2, a_x_g, label=" Measured Acceleration,␣
↪$\\vec{a_x}\\frac{m}{s^2}$", color='c')
ax_o1.plot(acceleration_calculated['time'], a_cx_g, label="Calculated␣
↪Acceleration, $\\vec{a_x}\\frac{m}{s^2}$", color='r')
ax_o1.set_xlabel('Time, t(s)')
ax_o1.set_ylabel('Acceleration, $\\vec{a_x}\\frac{m}{s^2}$')
ax_o1.legend()

ax_o2.plot(t2, a_y_g, label=" Measured Acceleration,␣
↪$\\vec{a_y}\\frac{m}{s^2}$", color='c')
ax_o2.plot(acceleration_calculated['time'], a_cy_g, label="Calculated␣
↪Acceleration, $\\vec{a_y}\\frac{m}{s^2}$", color='r')
ax_o2.set_xlabel('Time, t(s)')
ax_o2.set_ylabel('Acceleration, $\\vec{a_y}\\frac{m}{s^2}$')
ax_o2.legend()

ax_o3.plot(t2, a_z_g, label=" Measured Acceleration,␣
↪$\\vec{a_z}\\frac{m}{s^2}$", color='c')
ax_o3.plot(acceleration_calculated['time'], a_cz_g, label="Calculated␣
↪Acceleration, $\\vec{a_z}\\frac{m}{s^2}$", color='r')
ax_o3.set_xlabel('Time, t(s)')
ax_o3.set_ylabel('Acceleration, $\\vec{a_z}\\frac{m}{s^2}$')
ax_o3.legend()

ax_o1.grid(True)
a01.grid(True)
ax_o2.grid(True)
ax_o3.grid(True)
fig.tight_layout()
plt.show()
```

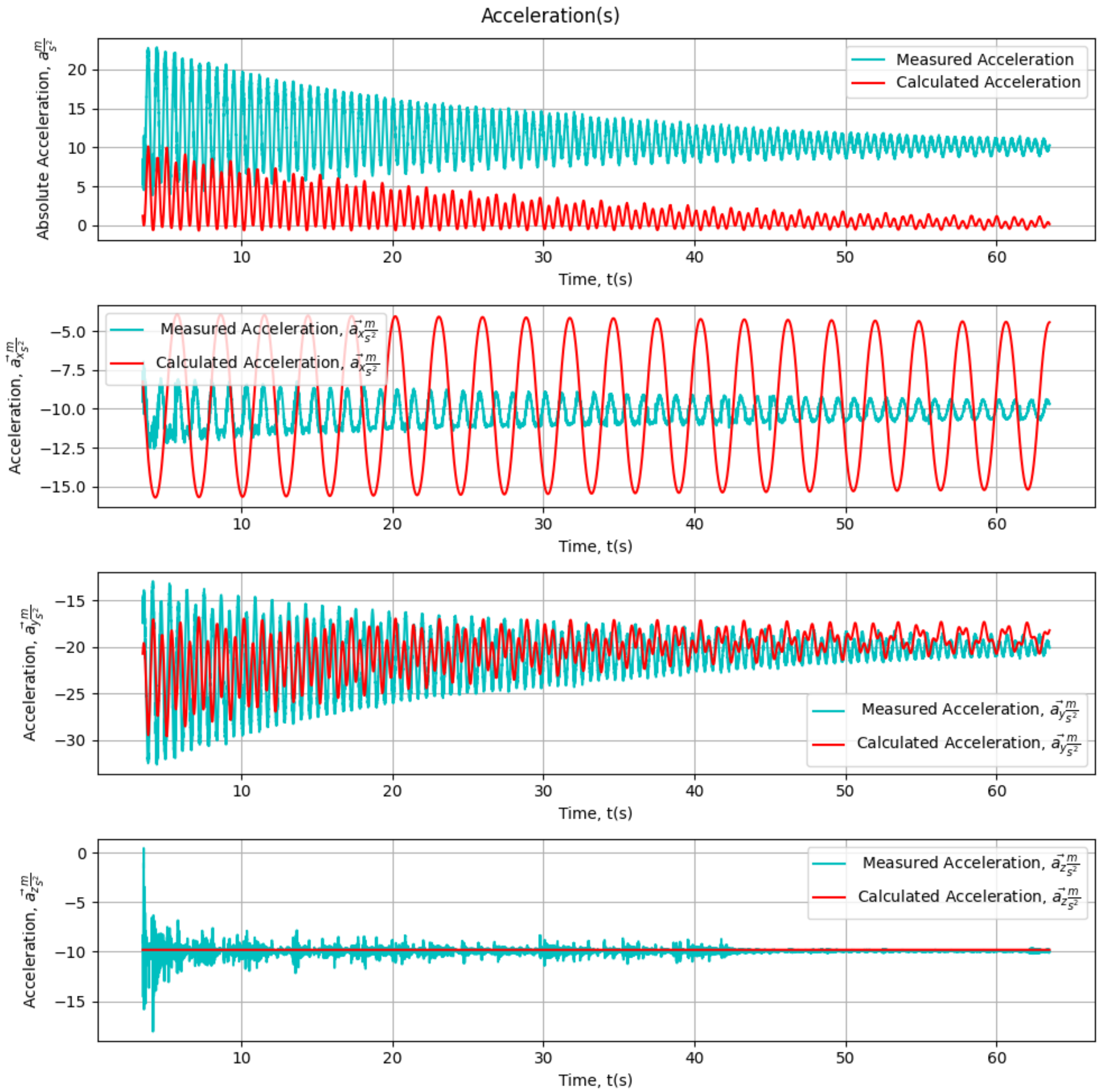


Figure 7: Linear Acceleration(s) (a) $|a| \ \frac{m}{s^2}$ (b) $\vec{a}_x \frac{m}{s^2}$ (c) $\vec{a}_y \frac{m}{s^2}$ (d) $\vec{a}_z \frac{m}{s^2}$

4.2.2 Error Caculations:

Overall Error: All the uncertainities are explained and calcualted as the paper progress, but as an overall summary, the error margin of the main values are:

Component	Value
Theoratical f_{md_t}	$\approx 0.959 \pm (0.00001) Hz$
Calcualted f_{md_e}	$\approx 0.955 \pm (0.0415) Hz$
Absolute Error	0.001 ± 0.04151
Theoratical δ	≈ 0.0245

Component	Value
Calculated δ	≈ 0.0262
Absolute Error	-0.0017

Table 1: Error

Type B Error Calculations: Outside the already mentioned uncertainities for each value, there are some unquantifiable unceratinities that emerge from our system. Such as systemic deviations. In the equation of motion of the pendulum from which the formular for the eigenfrequence is derived, we are using the small angle approximation. In this experiment our largest angle is 45 degree (our released angle).

Futher Inaccuracies in the measurement could possibly have been caused by factors such as a slight trembling of the hand when releasing the pendulum or a possible unintentional bumping of the ruler against the table top. These factors could potentially have introduced small deviations and disturbances into the oscillation.

4.2.3 Conclusion

The result has shown an expected outcome, where the behavior of the measured acceleration(s) are more chaotic, as external influences affect the stability and precision of the experiment.The unacounted extdernal influence to the system lead to the more varying values for the final magntiudes, yet the resutls were in the manner expected for a elementary approach such as ours.

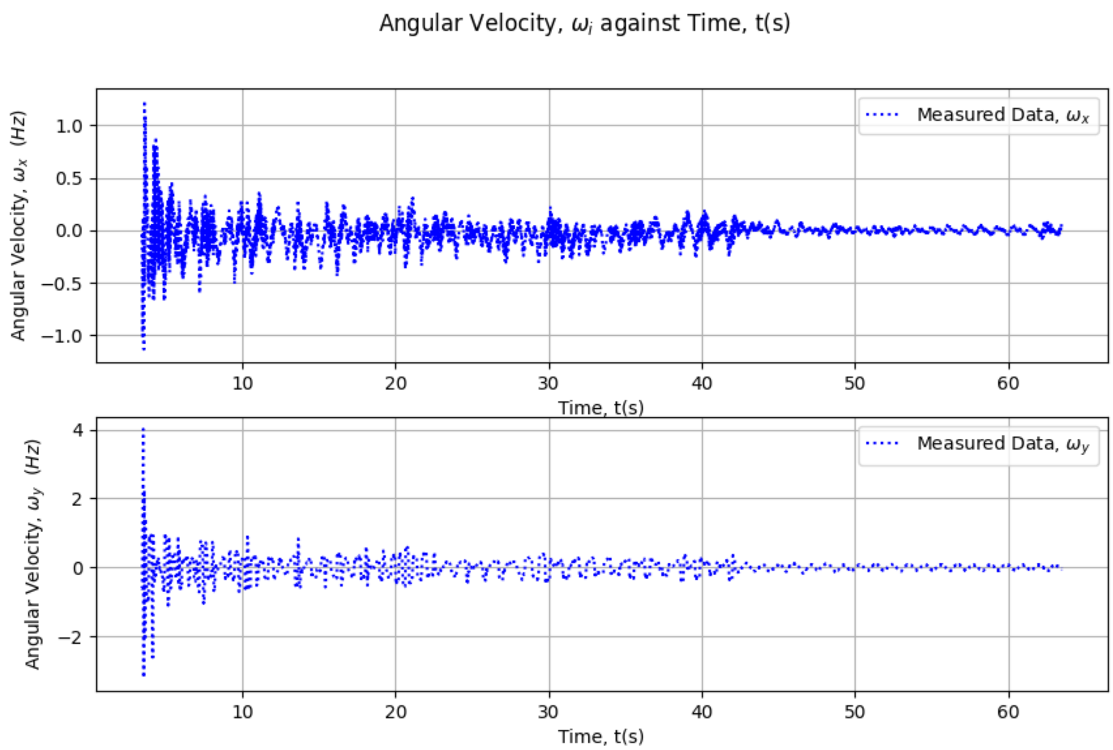
Overall, the experiment's utilisation of Angular Velocity to Measure the Calcualted Acceleration(s) was feasable, and provided a set of data that behaved in the manner expected, and seen, from the Measred Data set of Acceleration(s). With furthermore, the implementation of derived formulas to calculate values such as, but not limited to, δ or f_{md_i} were a success. In future developments, a more controlled environment for testing would increase the accuracy of the experiment, and additionally testing varying values (such as changing initial conditions) to see how they influence the system, would be a sufficient development for anyone interested.

4.2.4 Appendix:

Graphs for the ω_x and ω_y, as mentioned in an earlier section:

[52]:
```
t = filtered_data_gyroscope.iloc[:, 0]  #time
fig, (ax1, ax2) = plt.subplots(2, figsize=(10, 6))
fig.suptitle('Angular Velocity, ${\omega_i}$ against Time, t(s)')
ax1.plot(t, av_x, linestyle='dotted', color='b',  markersize=5, label="Measured␣
 ↪Data, ${\omega_x}$")
ax1.set_xlabel('Time, t(s)')
ax1.set_ylabel('Angular Velocity, ${\omega_x}~~(Hz)$')
ax1.legend(loc='upper right')
ax2.plot(t, av_y, linestyle='dotted', color='b',  markersize=5, label="Measured␣
 ↪Data, ${\omega_y}$")
ax2.set_xlabel('Time, t(s)')
```

```
ax2.set_ylabel('Angular Velocity, ${\omega_y}~~(Hz)$')
ax2.legend()
ax2.grid(True)
ax1.grid(True)
plt.show()
fig.tight_layout()
```



4.2.5 Bibliography:

- "Human Activity Recognition Using LSTMs on Android | TensorFlow for Hackers (Part VI)." Curiously, 2017, curiousily.com/posts/human-activity-recognition-using-lstms-on-android/.
- "H01e Pendulum as an accelerated Frame of Reference , Michael Ziese