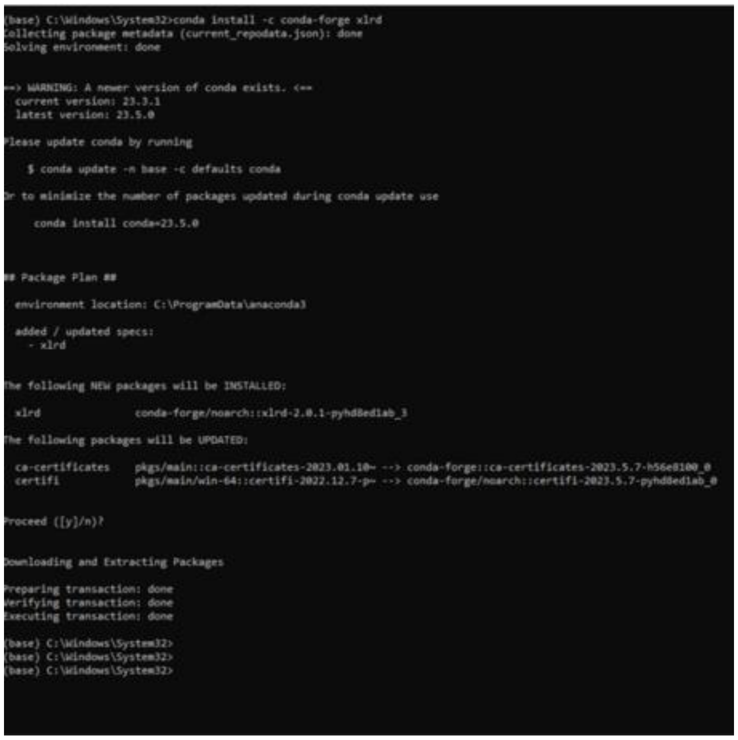
An understanding of how physics interacts with human movement is essential in the testing and application of proper exercise methods. Gravity, inertia, velocity, acceleration, friction, momentum, work, power, and torque are all aspects of physics that must be considered in proper exercise application and measurement [1]. In this problem set I did three exercises routine where I took advantage of the physics behind it to be able to gather data and represent it through coding in Jupyter Notebook. The three routines I did were, running to show the relationship between acceleration and time in three dimensions. Secondly, the jumping rope represents the oscillation and the relation of autocorrelation of sum and time shift. Lastly, arm rotation represents the accumulated data of angular velocity in a given time.

Method:

To execute the experiment I have to consider using a sensor app. I used Phyphox for this matter and utilized its capabilities to record simple movements and gather data. This app also has a feature in which it can export the obtained data.

For my running routine, I used the typical accelerometer to record my acceleration in a span of time. For jumping rope, I used the Spring feature of the app to record the frequency and period to represent the oscillation. While for the arm rotation, I used the Gyroscope feature to record the rotational movement of my arm.

After I successfully gathered the data, I exported it and downloaded it in my desktop. I import the file in the jupyter notebook but installation of "xlrd" is required. Thus, I installed it va conda cmd which shows below [2].

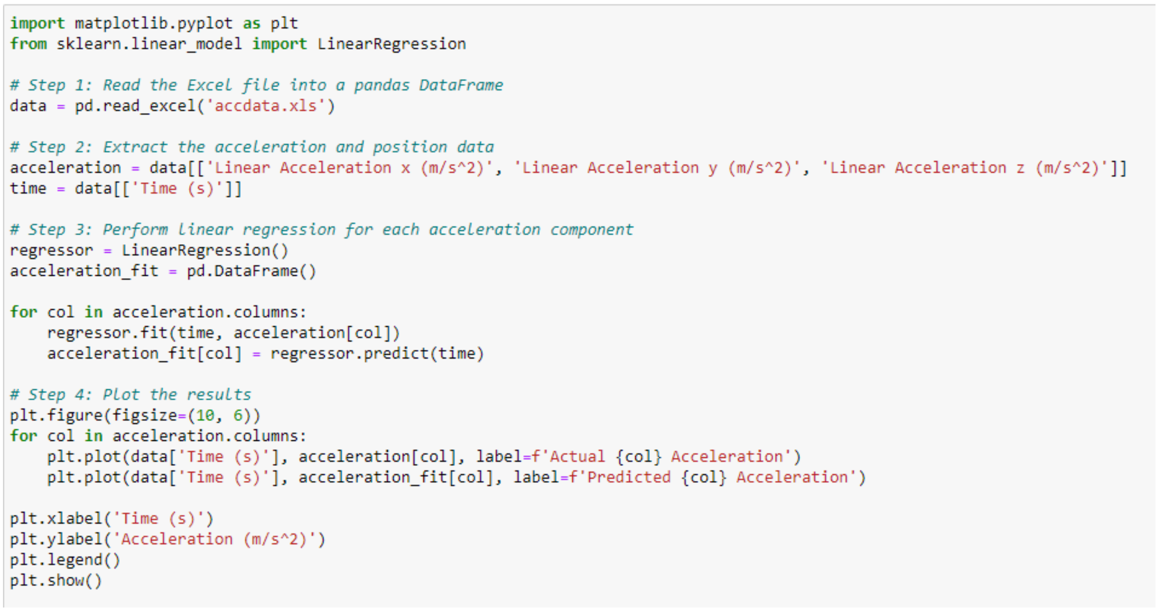


```
(base) C:\Windows\System32>conda install -c conda-forge xlrd
Collecting package metadata (current_repodata.json): done
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 23.3.1
  latest version: 23.5.0

Please update conda by running

    $ conda update -n base -c defaults conda

Or to minimize the number of packages updated during conda update use

     conda install conda=23.5.0



## Package Plan ##

  environment location: C:\ProgramData\anaconda3

  added / updated specs:
    - xlrd


The following NEW packages will be INSTALLED:

  xlrd               conda-forge/noarch::xlrd-2.0.1-pyhd8ed1ab_3

The following packages will be UPDATED:

  ca-certificates    pkgs/main::ca-certificates-2023.01.10~ --> conda-forge::ca-certificates-2023.5.7-h56e8100_0
  certifi            pkgs/main/win-64::certifi-2022.12.7-p~ --> conda-forge/noarch::certifi-2023.5.7-pyhd8ed1ab_0


Proceed ([y]/n)?


Downloading and Extracting Packages

Preparing transaction: done
Verifying transaction: done
Executing transaction: done

(base) C:\Windows\System32>
(base) C:\Windows\System32>
(base) C:\Windows\System32>
```

a. Acceleration

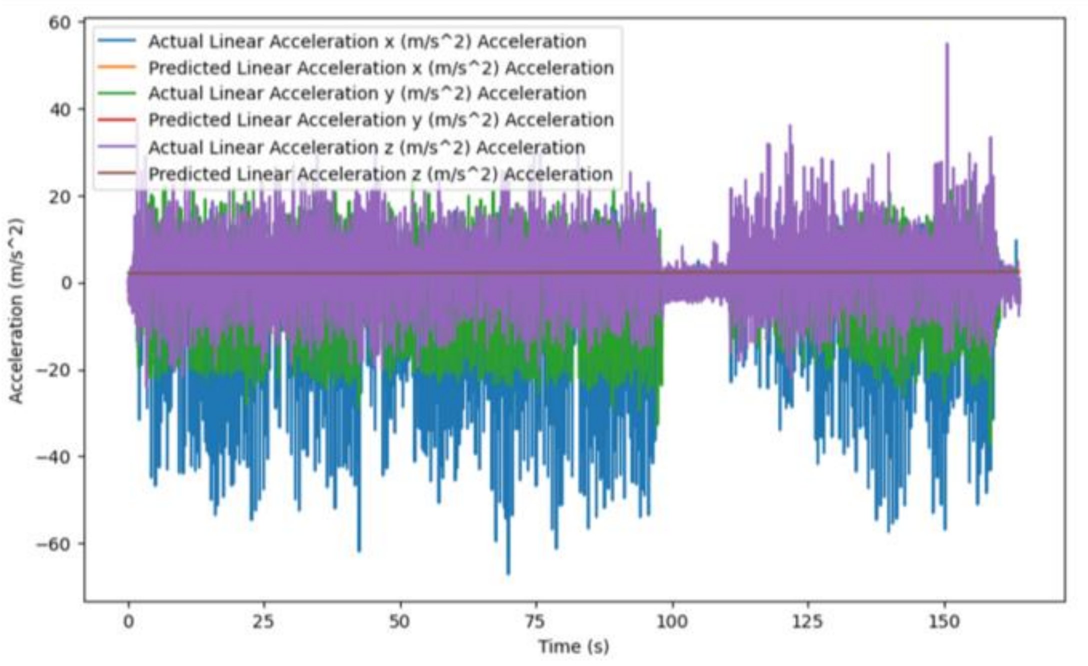
a.1 representation of the accumulated data

Using accelerometer feature of Phyphox app, I held my phone and started to run in 2-3 lapses covering more than 100m seconds. I prefer not to lengthen the time too much in so that I can process the data and export it quickly.



```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Step 1: Read the Excel file into a pandas DataFrame
data = pd.read_excel('accdata.xls')

# Step 2: Extract the acceleration and position data
acceleration = data[['Linear Acceleration x (m/s^2)', 'Linear Acceleration y (m/s^2)', 'Linear Acceleration z (m/s^2)']]
time = data[['Time (s)']]

# Step 3: Perform linear regression for each acceleration component
regressor = LinearRegression()
acceleration_fit = pd.DataFrame()

for col in acceleration.columns:
    regressor.fit(time, acceleration[col])
    acceleration_fit[col] = regressor.predict(time)

# Step 4: Plot the results
plt.figure(figsize=(10, 6))
for col in acceleration.columns:
    plt.plot(data['Time (s)'], acceleration[col], label=f'Actual {col} Acceleration')
    plt.plot(data['Time (s)'], acceleration_fit[col], label=f'Predicted {col} Acceleration')

plt.xlabel('Time (s)')
plt.ylabel('Acceleration (m/s^2)')
plt.legend()
plt.show()
```

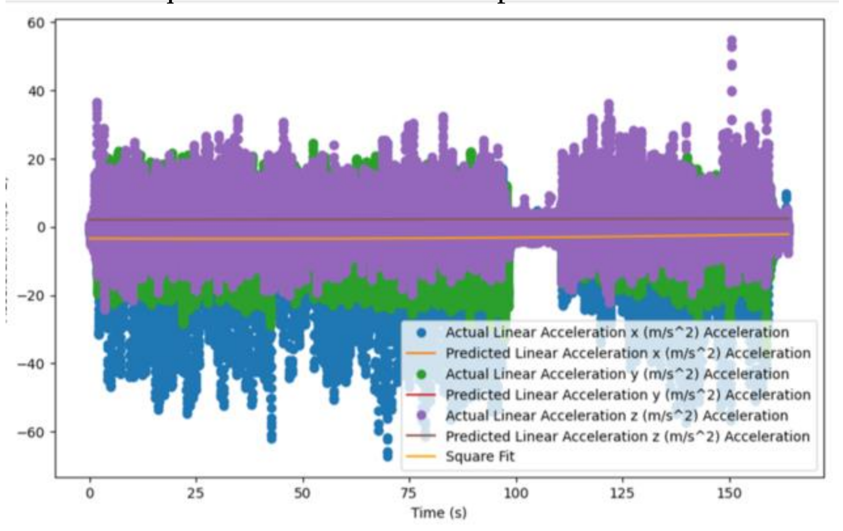
Using this code, it will extract the acceleration data for each component ('Linear Acceleration x', 'Linear Acceleration y', 'Linear Acceleration z') along with the corresponding time data and perform linear regression for each component. The results shown below, showing the actual and predicted acceleration values over time [3].



a.2 Representation using Linear square fit

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression

# Step 1: Read the Excel file into a pandas DataFrame
data = pd.read_excel('accdata.xls')

# Step 2: Extract the acceleration and position data
acceleration = data[['Linear Acceleration x (m/s^2)', 'Linear Acceleration y (m/s^2)', 'Linear Acceleration z (m/s^2)']]
time = data[['Time (s)']]

# Step 3: Perform linear regression for each acceleration component
regressor = LinearRegression()
acceleration_fit = pd.DataFrame()

for col in acceleration.columns:
    regressor.fit(time, acceleration[col])
    acceleration_fit[col] = regressor.predict(time)

# Step 4: Plot the results
plt.figure(figsize=(10, 6))
for col in acceleration.columns:
    plt.plot(data['Time (s)'], acceleration[col], 'o', label=f'Actual {col} Acceleration')
    plt.plot(data['Time (s)'], acceleration_fit[col], label=f'Predicted {col} Acceleration')

# Add square fit line
x = np.array(data['Time (s)']).reshape(-1, 1)
x_square = np.hstack((x, x**2))  # Add squared term
regressor_square = LinearRegression()
regressor_square.fit(x_square, acceleration['Linear Acceleration x (m/s^2)'])
y_square = regressor_square.predict(x_square)
plt.plot(data['Time (s)'], y_square, label='Square Fit', color='orange')

plt.xlabel('Time (s)')
plt.ylabel('Acceleration (m/s^2)')
plt.legend()
plt.show()
```

A square fit line is added to the plot using the squared term of the time data. The "x_square array" is created by stacking the original time array x with its squared values x**2. Then, a new linear regression model "regressor_square" is fitted to x_square and the acceleration data for the 'Linear Acceleration x' component. The predicted values y_square based on the square fit model are then plotted

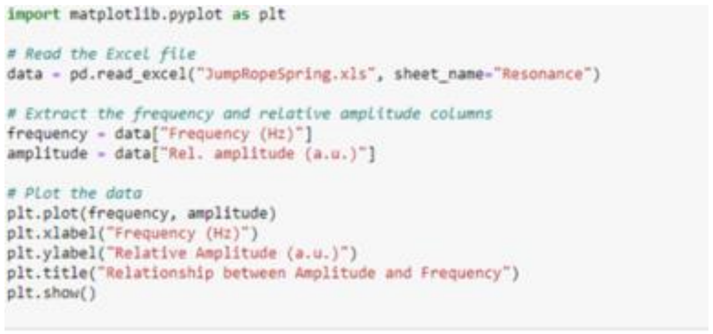


The linear and square fits in this context provide an approximation of the relationship between time and acceleration. The relationship between time and acceleration is represented by a straight line in the linear fit. It makes the assumption that there is a linear relationship and that the change in time is proportional to the change in acceleration. The best-fit line between the predicted and actual acceleration values is determined by the linear regression model, which also finds the best-fit line overall. We can determine the general direction and relationship between time and acceleration using the linear fit. A quadratic relationship between time and acceleration is represented by the square fit. It is predicated on the notion that the acceleration will increase over time in a parabolic manner. The square fit captures the curvature in the data by incorporating the squared term of time into the regression model. In comparison to a simple linear fit, the square fit line's quadratic function approximates the relationship between time and acceleration while allowing us to identify more complex patterns in the data [4].

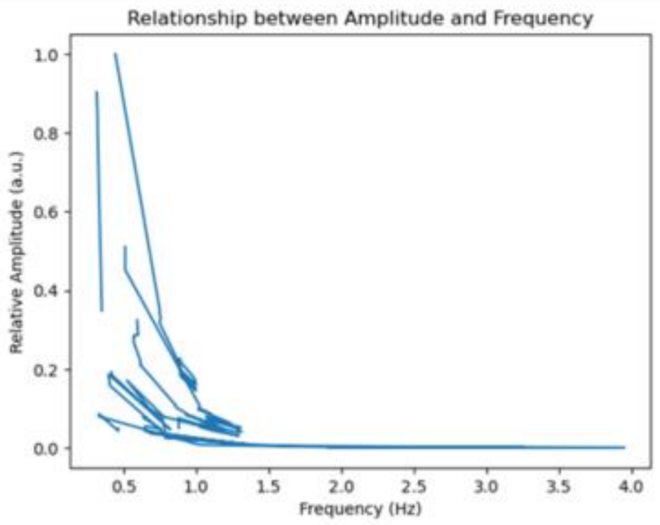
b. Oscillation

In this routine, I think of an activity that would imitate the way spring behaves. I initally thought of jumping jacks but to make it more funa nd complex I used a jumping rope. I put the phone inside my pocket as a sensor while I jump, however, this method seems distracting because of the movement of the phone inside. And also the space inside my pocket affects the way the phone moves. Thus, I tried to handle the phone on my left arm as I swing the rope. It added the difficulty but the result was much better than the previous method. I utilized Phyphox "Spring" feature as a sensor [5].

b.1 Amplitude and frequecy representation of the gathered data

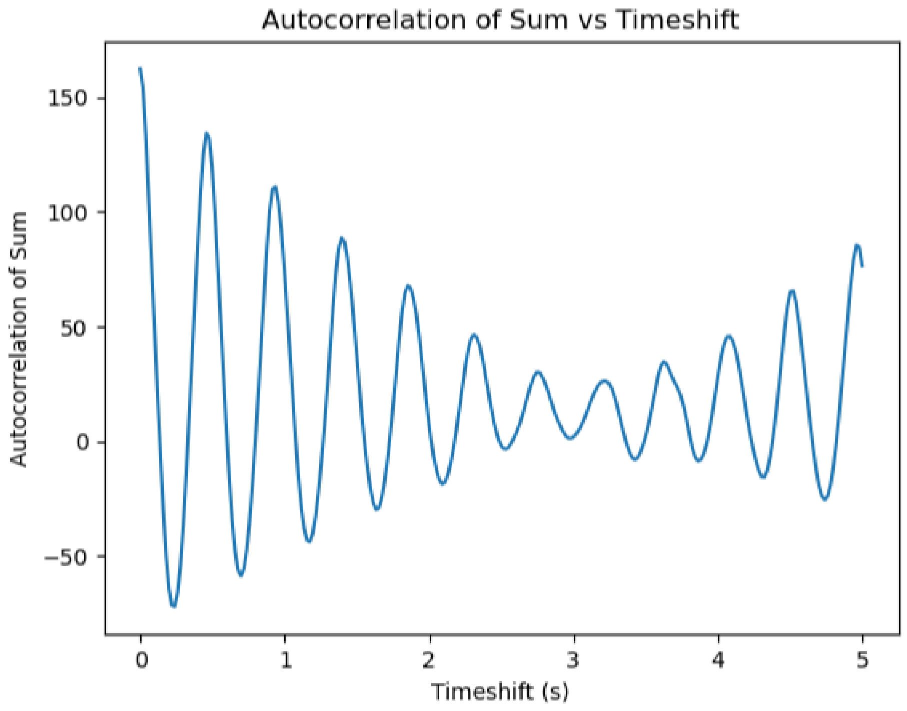


```
import matplotlib.pyplot as plt

# Read the Excel file
data = pd.read_excel("JumpRopeSpring.xls", sheet_name="Resonance")

# Extract the frequency and relative amplitude columns
frequency = data["Frequency (Hz)"]
amplitude = data["Rel. amplitude (a.u.)"]

# Plot the data
plt.plot(frequency, amplitude)
plt.xlabel("Frequency (Hz)")
plt.ylabel("Relative Amplitude (a.u.)")
plt.title("Relationship between Amplitude and Frequency")
plt.show()
```

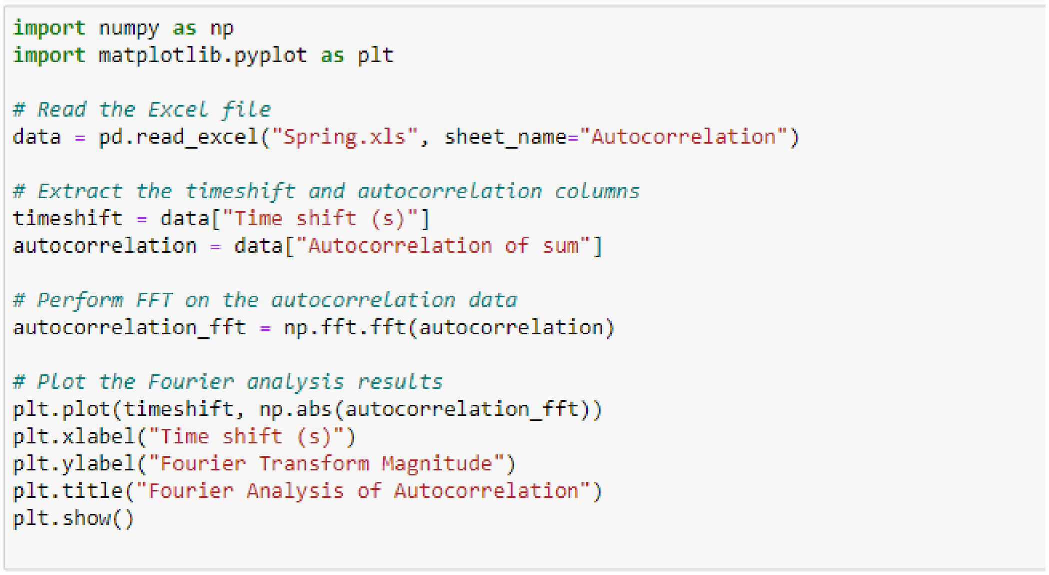


b.2 The relationship of autocorrelation of sum and time shift

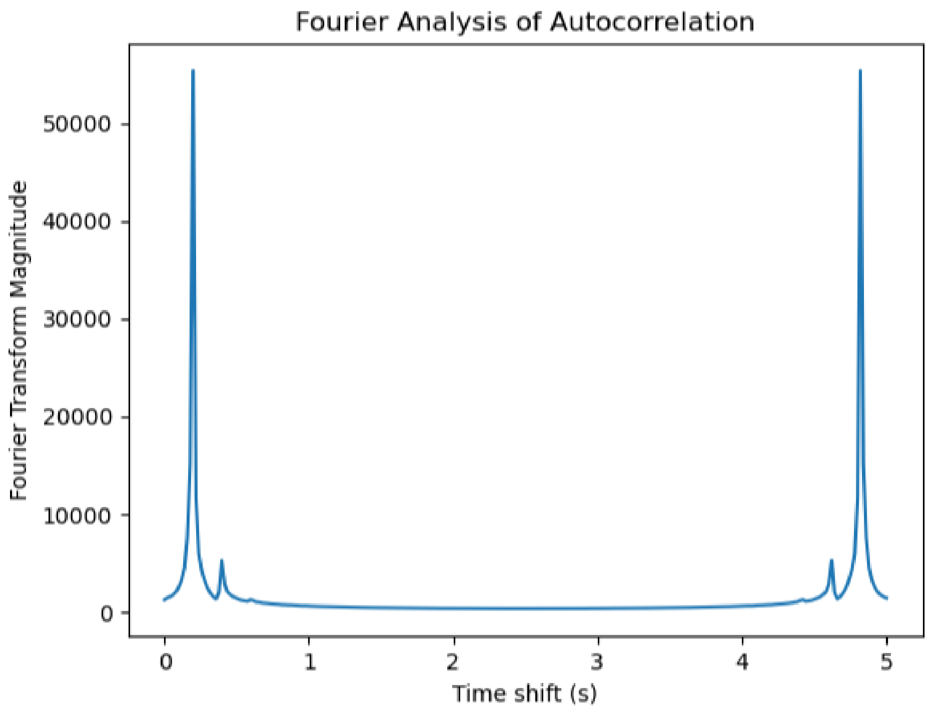
```
import matplotlib.pyplot as plt

# Read the Excel file
data = pd.read_excel("JumpRopeOsc.xls", sheet_name="Autocorrelation")

# Extract the timeshift and autocorrelation columns
timeshift = data["Time shift (s)"]
autocorrelation = data["Autocorrelation of sum"]

# Plot the autocorrelation
plt.plot(timeshift, autocorrelation)
plt.xlabel("Timeshift (s)")
plt.ylabel("Autocorrelation of Sum")
plt.title("Autocorrelation of Sum vs Timeshift")
plt.show()
```



b.2.1 Using Fourier Analysis Approach

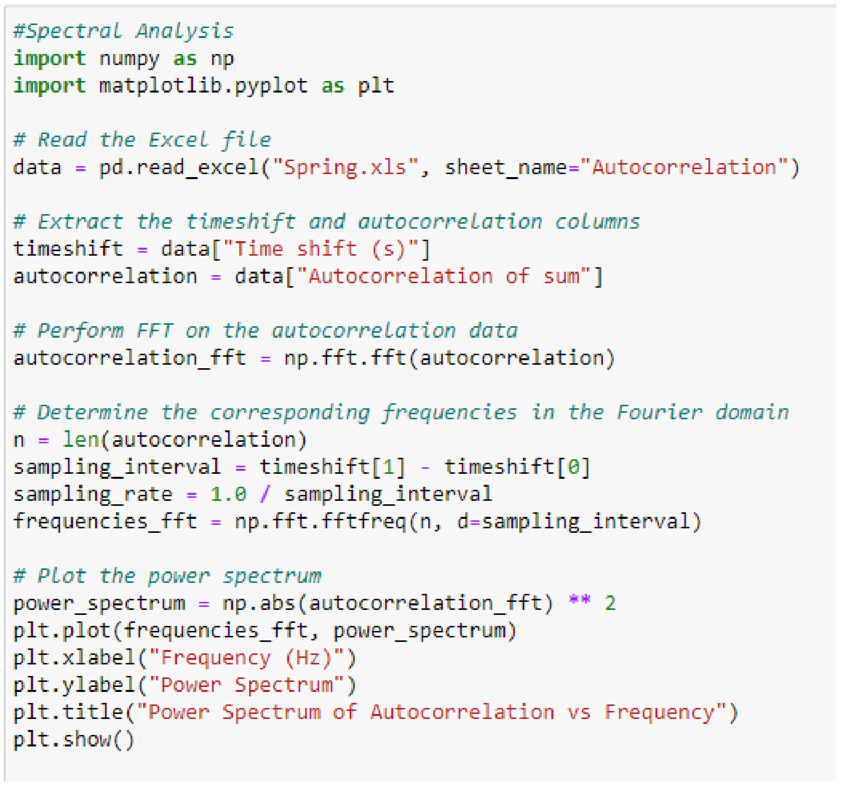


```
import numpy as np
import matplotlib.pyplot as plt

# Read the Excel file
data = pd.read_excel("Spring.xls", sheet_name="Autocorrelation")

# Extract the timeshift and autocorrelation columns
timeshift = data["Time shift (s)"]
autocorrelation = data["Autocorrelation of sum"]

# Perform FFT on the autocorrelation data
autocorrelation_fft = np.fft.fft(autocorrelation)

# Plot the Fourier analysis results
plt.plot(timeshift, np.abs(autocorrelation_fft))
plt.xlabel("Time shift (s)")
plt.ylabel("Fourier Transform Magnitude")
plt.title("Fourier Analysis of Autocorrelation")
plt.show()
```

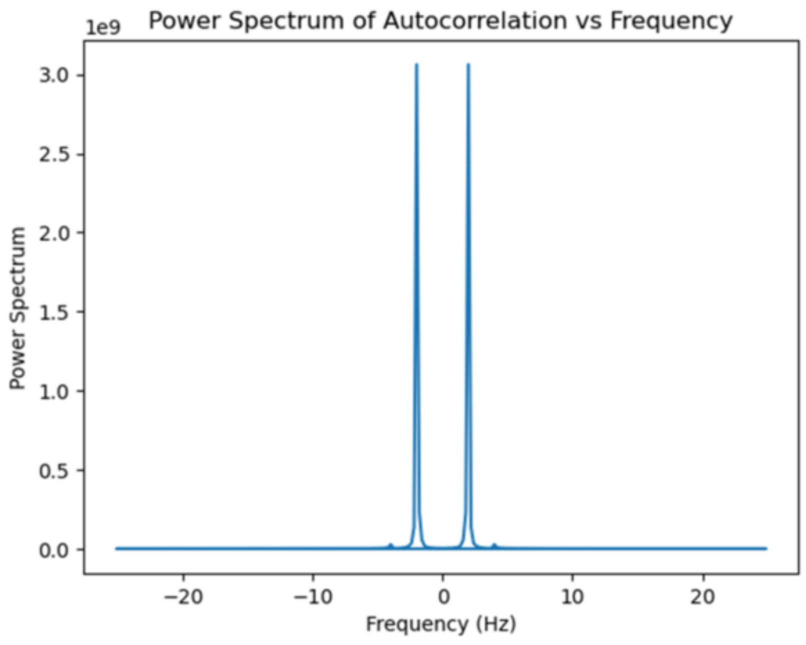


We can examine the frequency components present in the signal and gain understanding of the periodicity or rhythmic patterns within the data by applying the Fourier Transform to the autocorrelation data[6].The resulting plot helps to identify dominant frequencies or periodicities present in the autocorrelation of the sum data. It allows us to explore the frequency-domain characteristics of the signal and analyze its spectral content [7].

b.2.2 Using spectral analysis approach



```
#Spectral Analysis
import numpy as np
import matplotlib.pyplot as plt

# Read the Excel file
data = pd.read_excel("Spring.xls", sheet_name="Autocorrelation")

# Extract the timeshift and autocorrelation columns
timeshift = data["Time shift (s)"]
autocorrelation = data["Autocorrelation of sum"]

# Perform FFT on the autocorrelation data
autocorrelation_fft = np.fft.fft(autocorrelation)

# Determine the corresponding frequencies in the Fourier domain
n = len(autocorrelation)
sampling_interval = timeshift[1] - timeshift[0]
sampling_rate = 1.0 / sampling_interval
frequencies_fft = np.fft.fftfreq(n, d=sampling_interval)

# Plot the power spectrum
power_spectrum = np.abs(autocorrelation_fft) ** 2
plt.plot(frequencies_fft, power_spectrum)
plt.xlabel("Frequency (Hz)")
plt.ylabel("Power Spectrum")
plt.title("Power Spectrum of Autocorrelation vs Frequency")
plt.show()
```



Spectral analysis examines how a signal's energy or power is distributed across various frequencies.

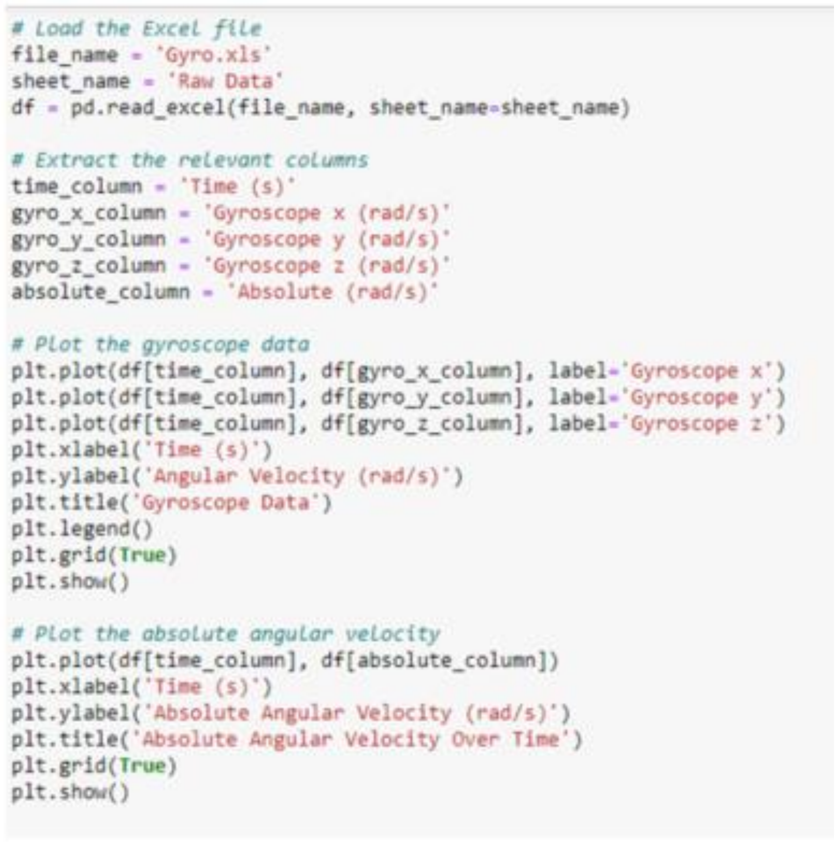
The power spectrum is calculated within the code by taking the absolute value squared of the Fourier Transform of the autocorrelation data. The intensity or power at various frequencies present in the autocorrelation signal is represented by the power spectrum that results [8].

The power spectrum plot illustrates the relationship between frequency and the strength of the autocorrelation signal at each frequency. It provides information about the spectral content of the signal which helps in identifying the dominant frequencies or periodic components in the autocorrelation data [9].

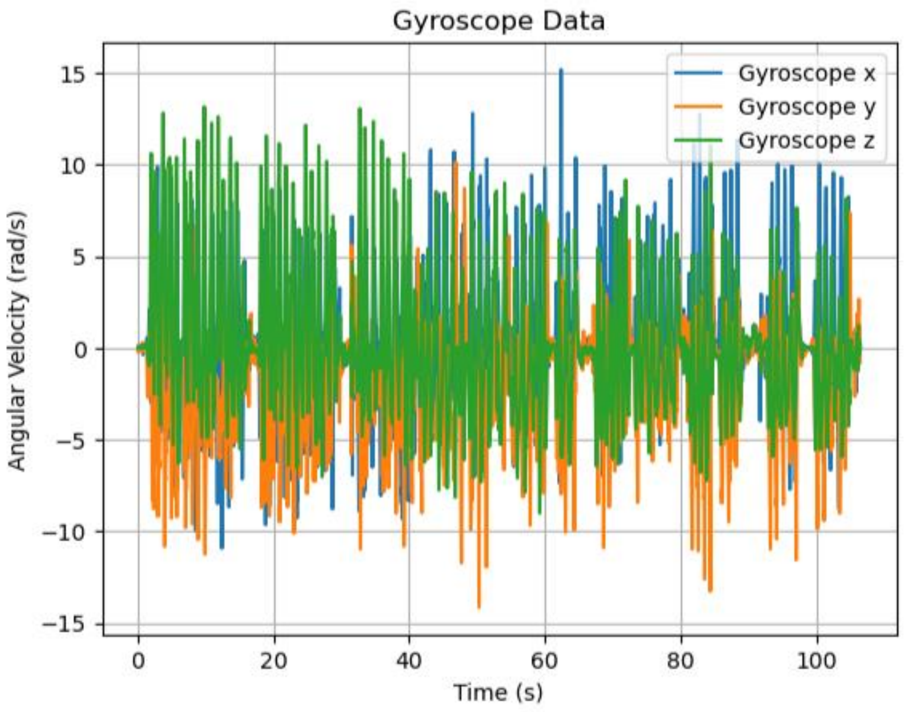
c. Angular Velocity

In this Physics concept I did a simple exercise routine to measure angular velocity- an arm rotation. We were doing tons of arm rotation before when I had my training in volleyball when I was in highschool. I thought this exercise would be a good example to measure angular velocity. In this activity, I used the "Gyroscope" feature in Phyphox app to show the relationship of angular velocity over time using different methods to represent this relation.

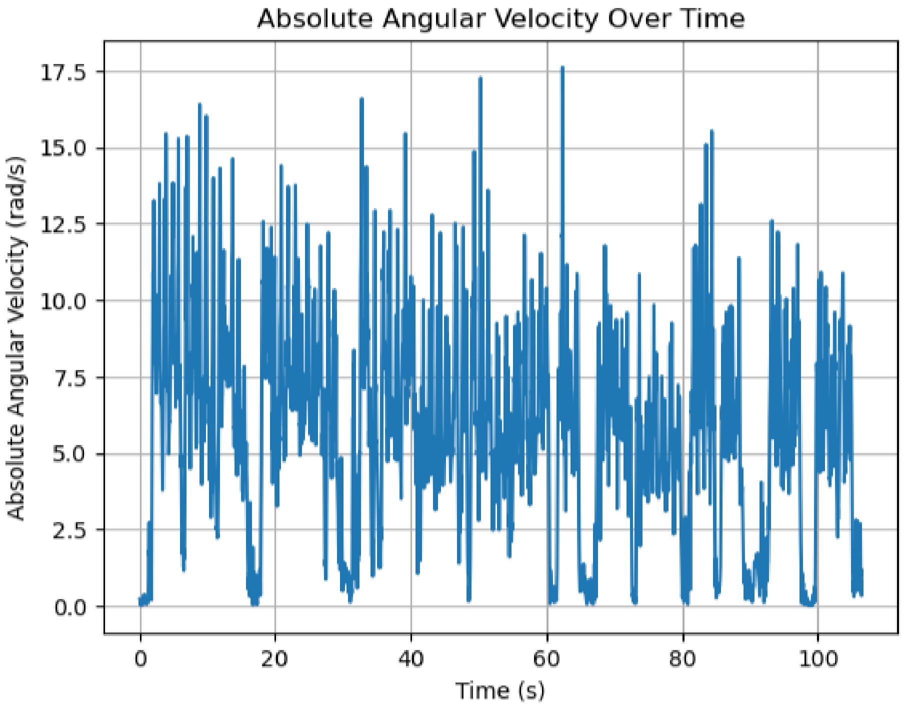
c.1 Representation of the gathered data:



```
# Load the Excel file
file_name = 'Gyro.xls'
sheet_name = 'Raw Data'
df = pd.read_excel(file_name, sheet_name=sheet_name)

# Extract the relevant columns
time_column = 'Time (s)'
gyro_x_column = 'Gyroscope x (rad/s)'
gyro_y_column = 'Gyroscope y (rad/s)'
gyro_z_column = 'Gyroscope z (rad/s)'
absolute_column = 'Absolute (rad/s)'

# Plot the gyroscope data
plt.plot(df[time_column], df[gyro_x_column], label='Gyroscope x')
plt.plot(df[time_column], df[gyro_y_column], label='Gyroscope y')
plt.plot(df[time_column], df[gyro_z_column], label='Gyroscope z')
plt.xlabel('Time (s)')
plt.ylabel('Angular Velocity (rad/s)')
plt.title('Gyroscope Data')
plt.legend()
plt.grid(True)
plt.show()

# Plot the absolute angular velocity
plt.plot(df[time_column], df[absolute_column])
plt.xlabel('Time (s)')
plt.ylabel('Absolute Angular Velocity (rad/s)')
plt.title('Absolute Angular Velocity Over Time')
plt.grid(True)
plt.show()
```

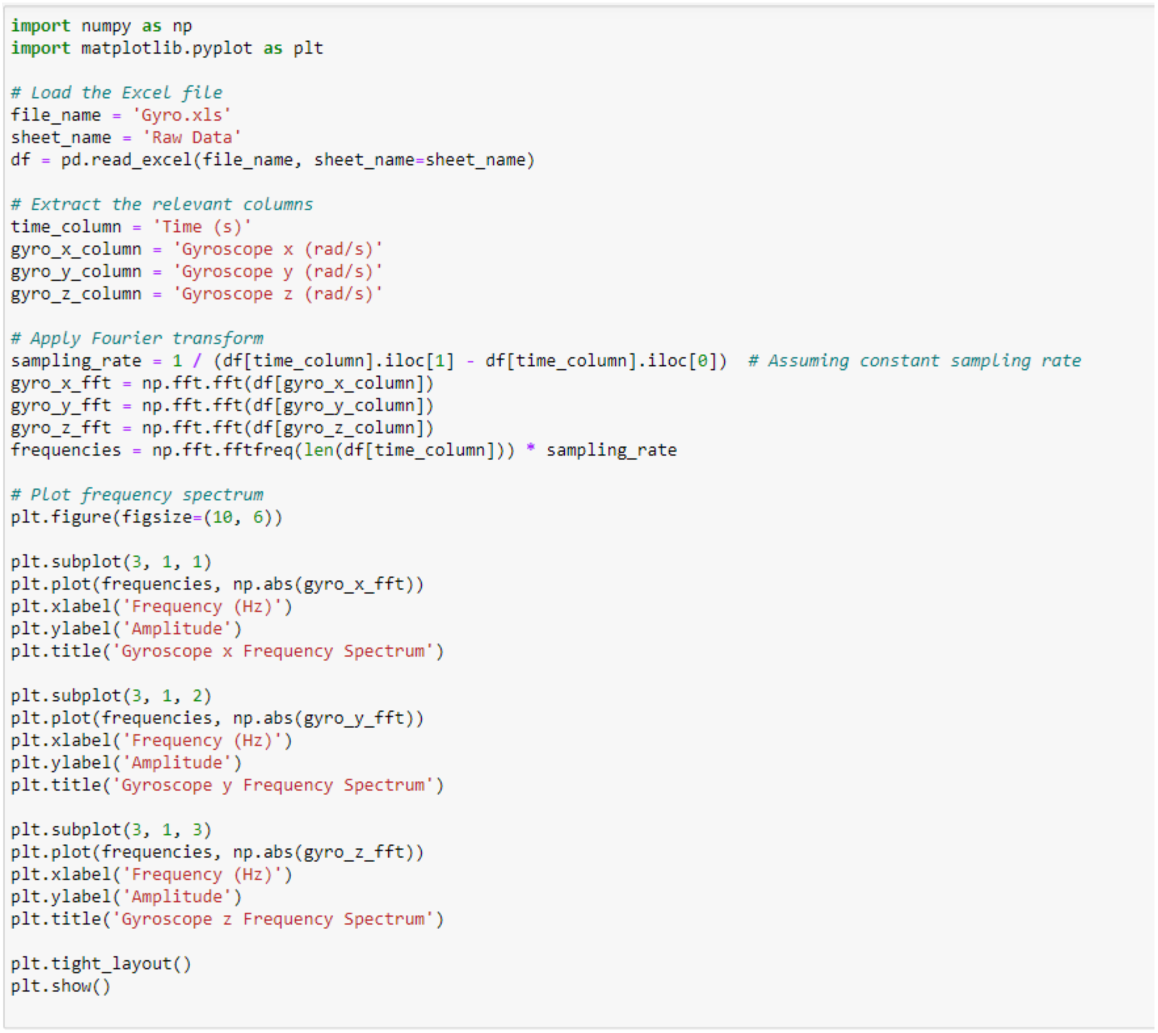


Gyroscope Data
Gyroscope x
Gyroscope y
Gyroscope z
Angular Velocity (rad/s)
Time (s)

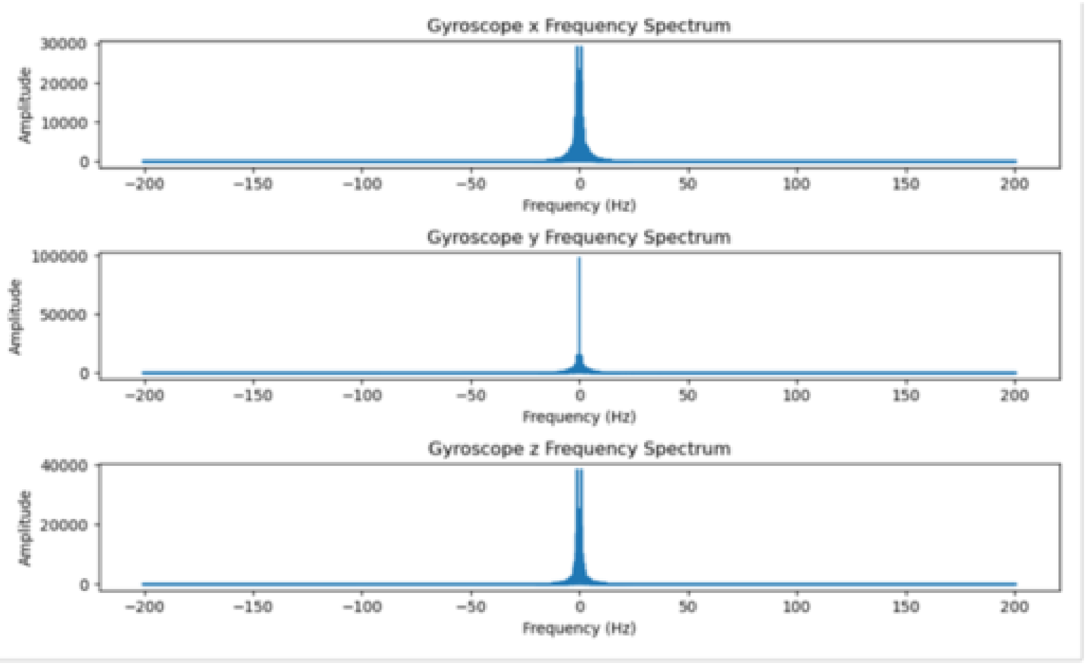


Absolute Angular Velocity Over Time
Absolute Angular Velocity (rad/s)
Time (s)

c.2 Representation using Fourier Transform



```
import numpy as np
import matplotlib.pyplot as plt

# Load the Excel file
file_name = 'Gyro.xls'
sheet_name = 'Raw Data'
df = pd.read_excel(file_name, sheet_name=sheet_name)

# Extract the relevant columns
time_column = 'Time (s)'
gyro_x_column = 'Gyroscope x (rad/s)'
gyro_y_column = 'Gyroscope y (rad/s)'
gyro_z_column = 'Gyroscope z (rad/s)'

# Apply Fourier transform
sampling_rate = 1 / (df[time_column].iloc[1] - df[time_column].iloc[0])  # Assuming constant sampling rate
gyro_x_fft = np.fft.fft(df[gyro_x_column])
gyro_y_fft = np.fft.fft(df[gyro_y_column])
gyro_z_fft = np.fft.fft(df[gyro_z_column])
frequencies = np.fft.fftfreq(len(df[time_column])) * sampling_rate

# Plot frequency spectrum
plt.figure(figsize=(10, 6))

plt.subplot(3, 1, 1)
plt.plot(frequencies, np.abs(gyro_x_fft))
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.title('Gyroscope x Frequency Spectrum')

plt.subplot(3, 1, 2)
plt.plot(frequencies, np.abs(gyro_y_fft))
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.title('Gyroscope y Frequency Spectrum')

plt.subplot(3, 1, 3)
plt.plot(frequencies, np.abs(gyro_z_fft))
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.title('Gyroscope z Frequency Spectrum')

plt.tight_layout()
plt.show()
```



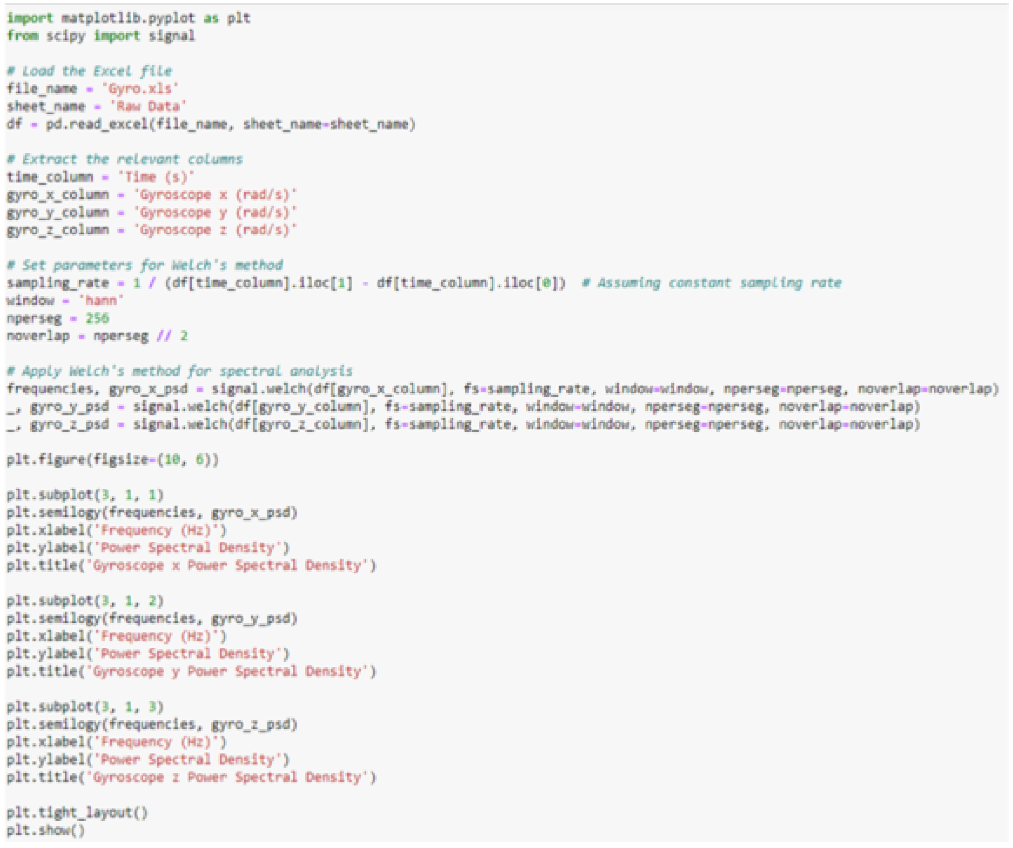
By performing the Fourier transform, we can convert the time-domain signals into the frequency domain and examine the different frequencies that contribute to the signal[10].

In the plotted values, the graph shows the frequency spectrum of each gyroscope axis. The x-axis represents frequency in Hz, and the y-axis represents the amplitude or magnitude of the frequency component. The Fourier transform provides information about the strength or contribution of each frequency to the overall signal.

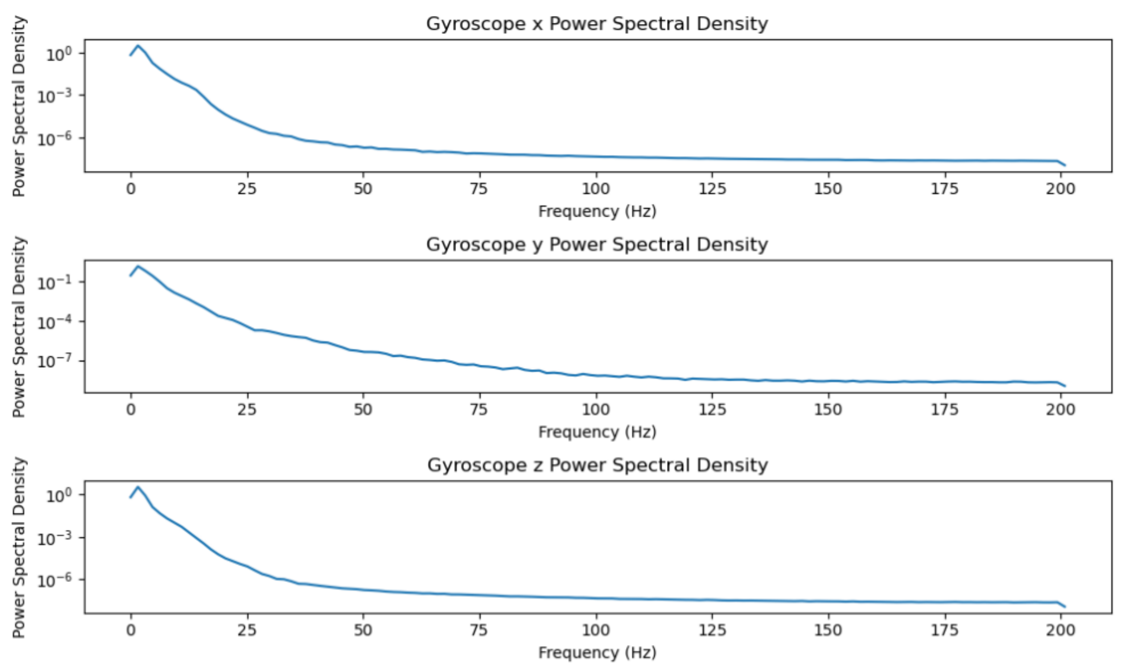
Peaks in the frequency spectrum indicate the presence of dominant frequencies in the gyroscope data. The height of a peak represents the magnitude or strength of that frequency component.

The x-axis displays the range of frequencies. Lower frequencies are located towards the left side, while higher frequencies are towards the right side.
The y-axis represents the amplitude or magnitude of each frequency component. A higher amplitude indicates a stronger contribution from that frequency.

c.3 Representation of data using Spectral analysis



```python
import matplotlib.pyplot as plt
from scipy import signal

# Load the Excel file
file_name = 'Gyro.xls'
sheet_name = 'Raw Data'
df = pd.read_excel(file_name, sheet_name=sheet_name)

# Extract the relevant columns
time_column = 'Time (s)'
gyro_x_column = 'Gyroscope x (rad/s)'
gyro_y_column = 'Gyroscope y (rad/s)'
gyro_z_column = 'Gyroscope z (rad/s)'

# Set parameters for Welch's method
sampling_rate = 1 / (df[time_column].iloc[1] - df[time_column].iloc[0])  # Assuming constant sampling rate
window = 'hann'
nperseg = 256
noverlap = nperseg // 2

# Apply Welch's method for spectral analysis
frequencies, gyro_x_psd = signal.welch(df[gyro_x_column], fs=sampling_rate, window=window, nperseg=nperseg, noverlap=noverlap)
_, gyro_y_psd = signal.welch(df[gyro_y_column], fs=sampling_rate, window=window, nperseg=nperseg, noverlap=noverlap)
_, gyro_z_psd = signal.welch(df[gyro_z_column], fs=sampling_rate, window=window, nperseg=nperseg, noverlap=noverlap)

plt.figure(figsize=(10, 6))

plt.subplot(3, 1, 1)
plt.semilogy(frequencies, gyro_x_psd)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Power Spectral Density')
plt.title('Gyroscope x Power Spectral Density')

plt.subplot(3, 1, 2)
plt.semilogy(frequencies, gyro_y_psd)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Power Spectral Density')
plt.title('Gyroscope y Power Spectral Density')

plt.subplot(3, 1, 3)
plt.semilogy(frequencies, gyro_z_psd)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Power Spectral Density')
plt.title('Gyroscope z Power Spectral Density')

plt.tight_layout()
plt.show()
```



The PSD represents the distribution of power across different frequencies in the gyroscope data. It provides information about the relative contribution of each frequency component to the overall power of the signal [12].

The graph indicates the power spectral density for each gyroscope axis. The x-axis represents frequency in Hz, and the y-axis represents the power spectral density. The power spectral density represents the power (or energy) of the signal at each frequency.

Peaks or spikes in the graph indicate dominant frequency components or significant power contributions at those specific frequencies.The height or magnitude of each peak represents the strength or amplitude of that particular frequency component. A larger peak indicates a higher power or energy associated with that frequency [13].

References:

[1] E. S. Ryan Hall, “Exercise science, LLC,” Specificity of Speed in Exercise | Exercise Science, LLC, http://exercisesciencellc.com/exercise-science-articles/specificity-exercise/specificity-exercise.html#:~:text=An%20understanding%20of%20how%20physics,proper%20exercise%20application%20and%20measurement. (accessed Jun. 14, 2023).

[2] "conda-forge," Xlrd :: Anaconda.org, https://anaconda.org/conda-forge/xlrd (accessed Jun. 14, 2023).

[3] "Phyphox file format," phyphox, https://phyphox.org/wiki/index.php/Phyphox_file_format (accessed Jun. 14, 2023).

[4] "What are acceleration vs. time graphs? (article)," Khan Academy, https://www.khanacademy.org/science/physics/one-dimensional-motion/acceleration-tutorial/a/what-are-acceleration-vs-time-graphs (accessed Jun. 14, 2023).

[5] S. Staacks, "Spring," phyphox, https://phyphox.org/experiment/spring/ (accessed Jun. 14, 2023).

[6] Real Python, "Fourier transforms with scipy.fft: Python Signal Processing," Real Python, https://realpython.com/python-scipy-fft/ (accessed Jun. 14, 2023).

[7] "What is Fourier analysis?," Collimator, https://www.collimator.ai/reference-guides/what-is-fourier-analysis (accessed Jun. 14, 2023).

[8] Z. West, "Autocorrelation of time series data in Python," αlphαrithms, https://www.alpharithms.com/autocorrelation-time-series-python-432909/ (accessed Jun. 14, 2023).

[9] Time series and spectral analysis - Stanford University, https://web.stanford.edu/class/earthsys214/notes/series.html (accessed Jun. 14, 2023).

[10] https://arxiv.org/pdf/2212.06949.pdf

[11] "Guide to FFT analysis (fast fourier transform)," Data Acquisition | Test and Measurement Solutions, https://dewesoft.com/blog/guide-to-fft-analysis (accessed Jun. 14, 2023).

[12] J. @Scicoding, "Calculating Power Spectral Density in python," Scicoding, https://scicoding.com/calculating-power-spectral-density-in-python/ (accessed Jun. 14, 2023).

[13] S. Hanly, "Why the Power Spectral Density (PSD) is the gold standard of vibration analysis," enDAQ Blog for Data Sensing and Analyzing, https://blog.endaq.com/why-the-power-spectral-density-psd-is-the-gold-standard-of-vibration-analysis (accessed Jun. 14, 2023).