```
!pip install pwlf
```

```
Collecting pwlf
  Downloading pwlf-2.5.1-py3-none-any.whl.metadata (6.3 kB)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.11/dist-packages (from pwlf) (2.0.2)
Requirement already satisfied: scipy>=1.8.0 in /usr/local/lib/python3.11/dist-packages (from pwlf) (1.15.3)
Downloading pwlf-2.5.1-py3-none-any.whl (17 kB)
Installing collected packages: pwlf
Successfully installed pwlf-2.5.1
```

```
import numpy as np
import scipy.optimize as sp
from scipy.signal import butter, filtfilt
from scipy.fft import fft, fftfreq
import matplotlib.pyplot as plt
import pwlf
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

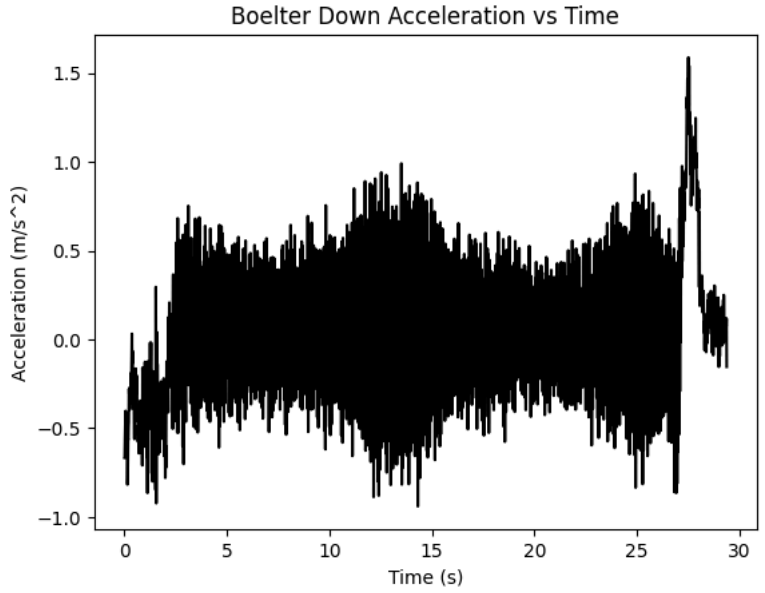
```
filepath = "/content/drive/MyDrive/Physics 4AL Table 2/Project/Data/"

# BOELTER ELEVATORS
boelter_down = np.loadtxt(filepath + "boelter_down8.txt", delimiter=',')
boelter_up = np.loadtxt(filepath + "boelter_up5.txt", delimiter=',')
# down
boelter_down_t = boelter_down[:,0]
boelter_down_t = boelter_down_t - boelter_down[0,0]
boelter_down_t = boelter_down_t / 1000
boelter_down_z = boelter_down[:,4]
# up
boelter_up_t = boelter_up[:,0]
boelter_up_t = boelter_up_t - boelter_up[0,0]
boelter_up_t = boelter_up_t / 1000
boelter_up_z = boelter_up[:,4]

# GEOLOGY ELEVATORS
geology_down = np.loadtxt(filepath + "geology_down7.txt", delimiter=',')
geology_up = np.loadtxt(filepath + "geology_up7.txt", delimiter=',')
# down
geology_down_t = geology_down[:,0]
geology_down_t = geology_down_t - geology_down[0,0]
geology_down_t = geology_down_t / 1000
geology_down_z = geology_down[:,4]
# up
geology_up_t = geology_up[:,0]
geology_up_t = geology_up_t - geology_up[0,0]
geology_up_t = geology_up_t / 1000
geology_up_z = geology_up[:,4]

# KNUDSEN ELEVATORS
knudsen_down = np.loadtxt(filepath + "knudsen_down8.txt", delimiter=',')
knudsen_up = np.loadtxt(filepath + "knudsen_up8.txt", delimiter=',', encoding='utf-8-sig')
# down
knudsen_down_t = knudsen_down[:,0]
knudsen_down_t = knudsen_down_t - knudsen_down[0,0]
knudsen_down_t = knudsen_down_t / 1000
knudsen_down_z = knudsen_down[:,4]
# up
knudsen_up_t = knudsen_up[:,0]
knudsen_up_t = knudsen_up_t - knudsen_up[0,0]
knudsen_up_t = knudsen_up_t / 1000
knudsen_up_z = knudsen_up[:,4]

# PAB ELEVATORS
pab_down = np.loadtxt(filepath + "pab_down6.txt", delimiter=',')
pab_up = np.loadtxt(filepath + "pab_up6.txt", delimiter=',')
# down
pab_down_t = pab_down[:,0]
pab_down_t = pab_down_t - pab_down[0,0]
pab_down_t = pab_down_t / 1000
pab_down_z = pab_down[:,4]
```

```
# up
pab_up_t = pab_up[:,0]
pab_up_t = pab_up_t - pab_up[0,0]
pab_up_t = pab_up_t / 1000
pab_up_z = pab_up[:,4]

# PRITZKER ELEVATORS
pritzker_down = np.loadtxt(filepath + "pritzker_down8.txt", delimiter=',')
pritzker_up = np.loadtxt(filepath + "pritzker_up8.txt", delimiter=',')
# down
pritzker_down_t = pritzker_down[:,0]
pritzker_down_t = pritzker_down_t - pritzker_down[0,0]
pritzker_down_t = pritzker_down_t / 1000
pritzker_down_z = pritzker_down[:,4]
# up
pritzker_up_t = pritzker_up[:,0]
pritzker_up_t = pritzker_up_t - pritzker_up[0,0]
pritzker_up_t = pritzker_up_t / 1000
pritzker_up_z = pritzker_up[:,4]
```

```
# The function that will convert the accelerometer calibration readings to actual acceleration values (in m/s^2).
# Accelerometer calibration for everything that isn't knudsen
def output_to_accel_z(output):
  accel = (6.27884288 * 10**(-4)) * output + (0.2)
  return accel

# Accelerometer calibration for just knudsen
def output_to_accel_z_k(output):
  accel = (6.27884288 * 10**(-4)) * output + (-0.8)
  return accel
```

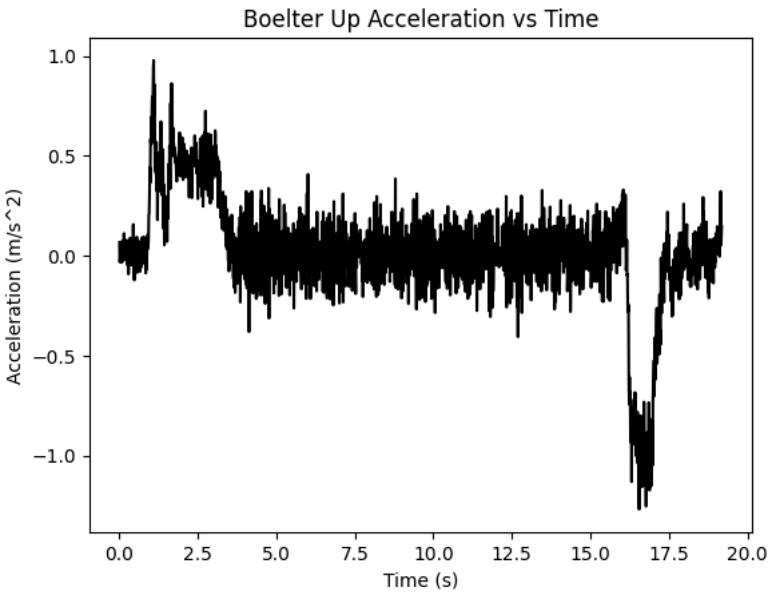
```
# BOELTER ELEVATORS
plt.plot(boelter_down_t, output_to_accel_z(boelter_down_z) - 9.81, c="black", label="Data")
plt.xlabel("Time (s)")
plt.ylabel("Acceleration (m/s^2)")
plt.title("Boelter Down Acceleration vs Time")
```

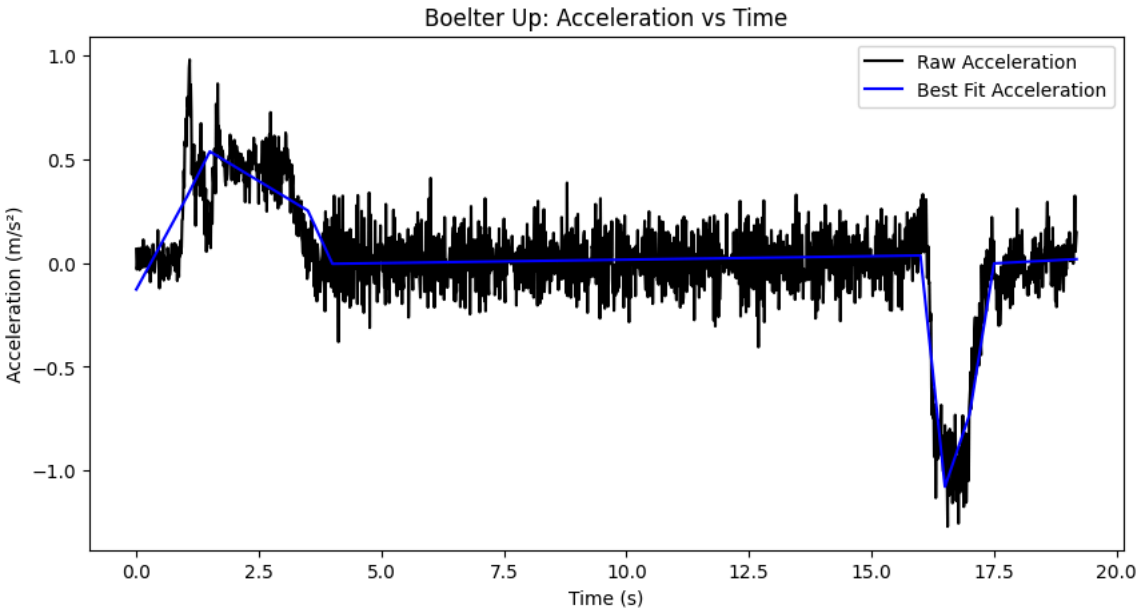
```
Text(0.5, 1.0, 'Boelter Down Acceleration vs Time')
```




```
# RAW DATA
plt.plot(boelter_up_t, output_to_accel_z(boelter_up_z) - 9.81, c="black", label="Data")
plt.xlabel("Time (s)")
plt.ylabel("Acceleration (m/s^2)")
plt.title("Boelter Up Acceleration vs Time")

# BEST FIT PIECEWISE LINEAR FUNCTION
breakpoints = np.array([min(boelter_up_t), 1, 1.5, 3.5, 4, 16, 16.5, 17, 17.5, max(boelter_up_t)])
# initialize piecewise linear fit with your x and y data
my_pwlf = pwlf.PiecewiseLinFit(boelter_up_t, output_to_accel_z(boelter_up_z) - 9.81)
# initialize piecewise linear fit with your x and y data
my_pwlf.fit_with_breaks(breakpoints)
```

```
# predict for the determined points
xHat = np.linspace(min(boelter_up_t), max(boelter_up_t), num=10000)
yHat = my_pwlf.predict(xHat)
# plot the results
plt.figure(figsize=(10, 5))
plt.plot(boelter_up_t, output_to_accel_z(boelter_up_z) - 9.81, c="black", label="Raw Acceleration")
plt.plot(xHat, yHat, c="blue", label="Best Fit Acceleration")
plt.xlabel("Time (s)")
plt.ylabel("Acceleration (m/s²)")
plt.title("Boelter Up: Acceleration vs Time")
plt.legend()


plt.figure(figsize=(10, 5))
plt.plot(boelter_up_t, output_to_accel_z(boelter_up_z) - 9.81, c="black", label="Raw Acceleration")  # still showing raw
bestfit_jerk = np.gradient(yHat, xHat)
plt.plot(xHat, bestfit_jerk, c="red", label="Best Fit Jerk")
raw_jerk = np.gradient(output_to_accel_z(boelter_up_z) - 9.81, boelter_up_t)
# plt.plot(boelter_up_t, raw_jerk, c="green", label="Raw Jerk")
plt.xlabel("Time (s)")
plt.ylabel("Jerk (m/s³)")
plt.title("Boelter Up: Jerk vs Time")
plt.legend()

# Sampling frequency (Hz)
fs = 100

# Define bandpass filter (0.1 - 2 Hz)
lowcut = 0.1
highcut = 2
order = 4

nyq = 0.5 * fs
low = lowcut / nyq
high = highcut / nyq

# Create filter
b, a = butter(order, [low, high], btype='band')

# Apply filter
filtered_signal = filtfilt(b, a, output_to_accel_z(boelter_up_z))

# Plot original and filtered signal
plt.figure(figsize=(12, 6))
plt.plot(output_to_accel_z(boelter_up_z), label='Original Signal')
plt.plot(filtered_signal, label='Filtered Signal', linewidth=2)
plt.legend()
plt.title("Boelter Up: Bandpass Filtered Signal (0.1 - 2 Hz)")
plt.xlabel("Sample")
plt.ylabel("Amplitude")
plt.grid(True)
plt.show()

# Only positive frequencies
mask = (xf >= 0) & (xf <= 25)  # zoom into 0–10 Hz

plt.plot(xf[mask], np.abs(yf)[mask] * 2 / N)
plt.title("Boelter Up: Frequency Spectrum (0–25 Hz)")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Amplitude")
plt.show()
```

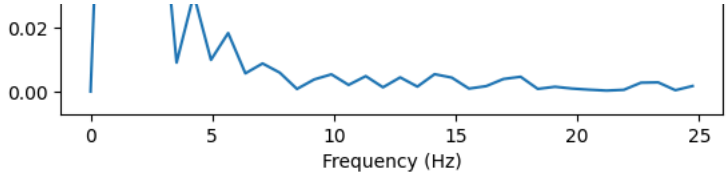




Start coding or generate with AI.

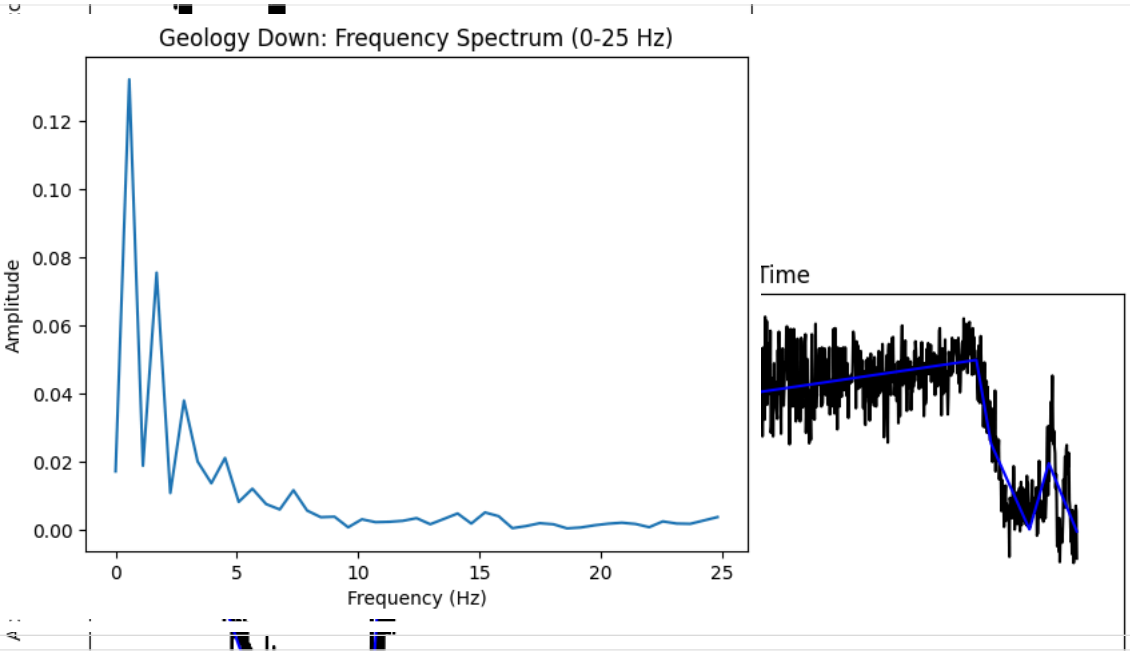
```
plt.plot(geology_down_t, output_to_accel_z(geology_down_z) - 9.81, c="black", label="Data")
plt.xlabel("Time (s)")
plt.ylabel("Acceleration (m/s^2)")
plt.title("Geology Down Acceleration vs Time")

breakpoints = np.array([min(geology_down_t), 1, 2.3, 4.8, 5.1, 12.5, 12.7, 17.4, 17.7, 18.5, 18.9, 20.0, max(geology_down_

# initialize piecewise linear fit with your x and y data
my_pwlf = pwlf.PiecewiseLinFit(geology_down_t, output_to_accel_z(geology_down_z) - 9.81)

# initialize piecewise linear fit with your x and y data
my_pwlf.fit_with_breaks(breakpoints)

# predict for the determined points
xHat = np.linspace(min(geology_down_t), max(geology_down_t), num=10000)
yHat = my_pwlf.predict(xHat)

# plot the results
plt.figure(figsize=(10, 5))
plt.plot(geology_down_t, output_to_accel_z(geology_down_z) - 9.81, c="black", label="Raw Acceleration")
plt.plot(xHat, yHat, c="blue", label="Best Fit Acceleration")
plt.xlabel("Time (s)")
plt.ylabel("Acceleration (m/s²)")
plt.title("Geology Down: Acceleration vs Time")
plt.legend()
```

```
plt.legend()

plt.figure(figsize=(10, 5))
jerk = np.gradient(yHat, xHat)
plt.plot(geology_down_t, output_to_accel_z(geology_down_z) - 9.81, c="black", label="Raw Acceleration")  # still showing
plt.plot(xHat, jerk, c="red", label="Best Fit Jerk")
plt.xlabel("Time (s)")
plt.ylabel("Jerk (m/s³)")
plt.title("Geology Down: Jerk vs Time")
plt.legend()

# Sampling frequency (Hz)
fs = 100

# Define bandpass filter (0.1 - 2 Hz)
lowcut = 0.1
highcut = 2
order = 4

nyq = 0.5 * fs
low = lowcut / nyq
high = highcut / nyq

# Create filter
b, a = butter(order, [low, high], btype='band')

# Apply filter
filtered_signal = filtfilt(b, a, output_to_accel_z(geology_down_z))

# Plot original and filtered signal
plt.figure(figsize=(12, 6))
plt.plot(output_to_accel_z(geology_down_z), label='Original Signal')
plt.plot(filtered_signal, label='Filtered Signal', linewidth=2)
plt.legend()
plt.title("Geology Down: Bandpass Filtered Signal (0.1 - 2 Hz)")
plt.xlabel("Sample")
plt.ylabel("Amplitude")
plt.grid(True)
plt.show()

se = my_pwlf.standard_errors()
print(se)
```

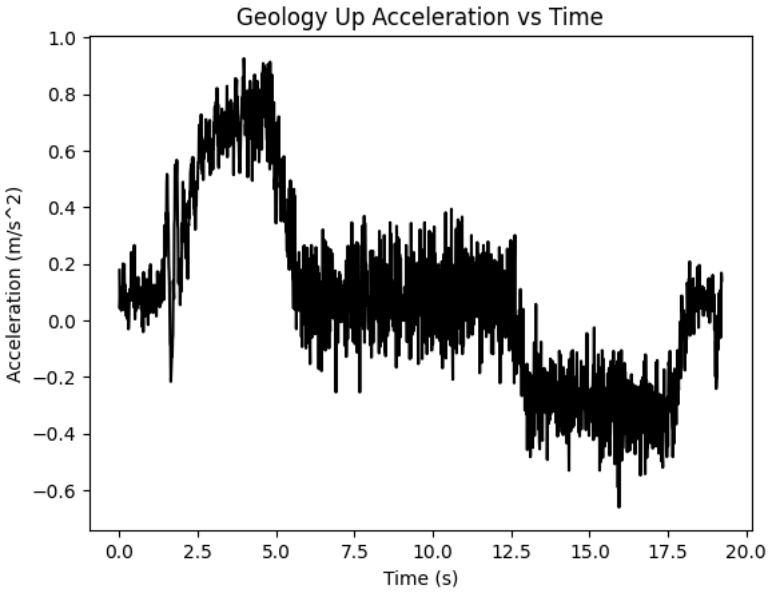


```
N = len(geology_down_z)
fs = 1000  # sampling frequency

# Remove DC offset
signal = output_to_accel_z(geology_down_z) - np.mean(output_to_accel_z(geology_down_z))
```

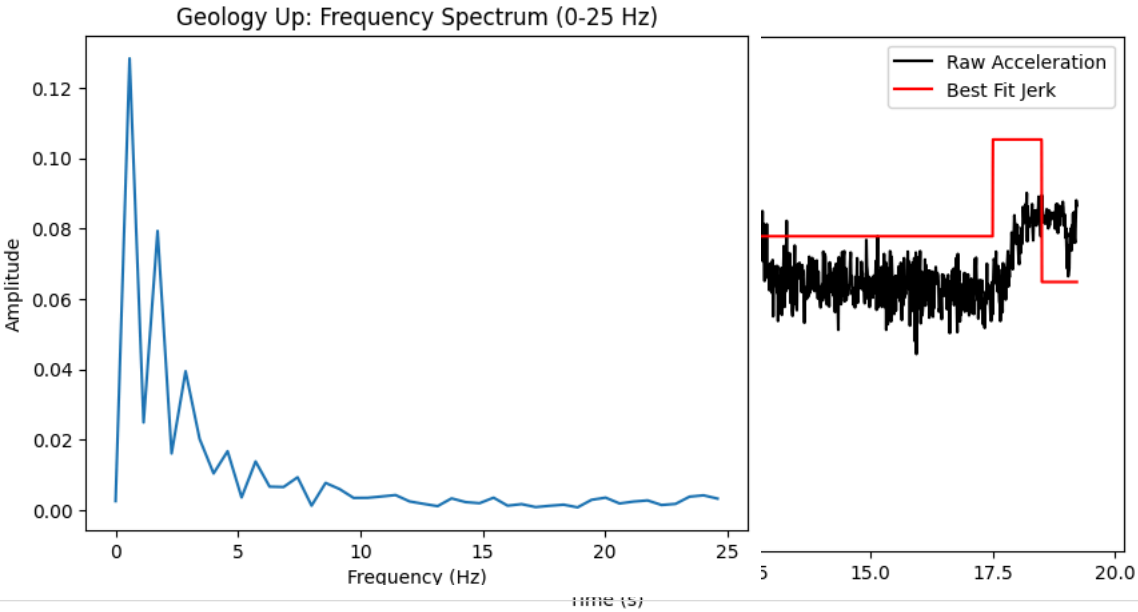
```
# Windowing
window = np.hanning(N)
yf = fft(signal * window)

xf = fftfreq(N, 1/fs)

# Only positive frequencies
mask = (xf >= 0) & (xf <= 25)  # zoom into 0–10 Hz

plt.plot(xf[mask], np.abs(yf)[mask] * 2 / N)
plt.title("Geology Down: Frequency Spectrum (0–25 Hz)")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Amplitude")
plt.show()
```




```
plt.plot(geology_up_t, output_to_accel_z(geology_up_z) - 9.81, c="black", label="Data")
plt.xlabel("Time (s)")
plt.ylabel("Acceleration (m/s^2)")
plt.title("Geology Up Acceleration vs Time")

breakpoints = np.array([min(geology_up_t), 1.5, 3.5, 4.9, 6, 12.5, 12.7, 17.5, 18.5, 20.0, max(geology_up_z)])

# initialize piecewise linear fit with your x and y data
my_pwlf = pwlf.PiecewiseLinFit(geology_up_t, output_to_accel_z(geology_up_z) - 9.81)

# initialize piecewise linear fit with your x and y data
my_pwlf.fit_with_breaks(breakpoints)

# predict for the determined points
xHat = np.linspace(min(geology_up_t), max(geology_up_t), num=10000)
yHat = my_pwlf.predict(xHat)

# plot the results
plt.figure(figsize=(10, 5))
plt.plot(geology_up_t, output_to_accel_z(geology_up_z) - 9.81, c="black", label="Raw Acceleration")
plt.plot(xHat, yHat, c="blue", label="Best Fit Acceleration")
plt.xlabel("Time (s)")
plt.ylabel("Acceleration (m/s²)")
plt.title("Geology Up: Acceleration vs Time")
plt.legend()

plt.figure(figsize=(10, 5))
jerk = np.gradient(yHat, xHat)
plt.plot(geology_up_t, output_to_accel_z(geology_up_z) - 9.81, c="black", label="Raw Acceleration")  # still showing raw
plt.plot(xHat, jerk, c="red", label="Best Fit Jerk")
plt.xlabel("Time (s)")
plt.ylabel("Jerk (m/s³)")
plt.title("Geology Up: Jerk vs Time")
plt.legend()

# Sampling frequency (Hz)
```

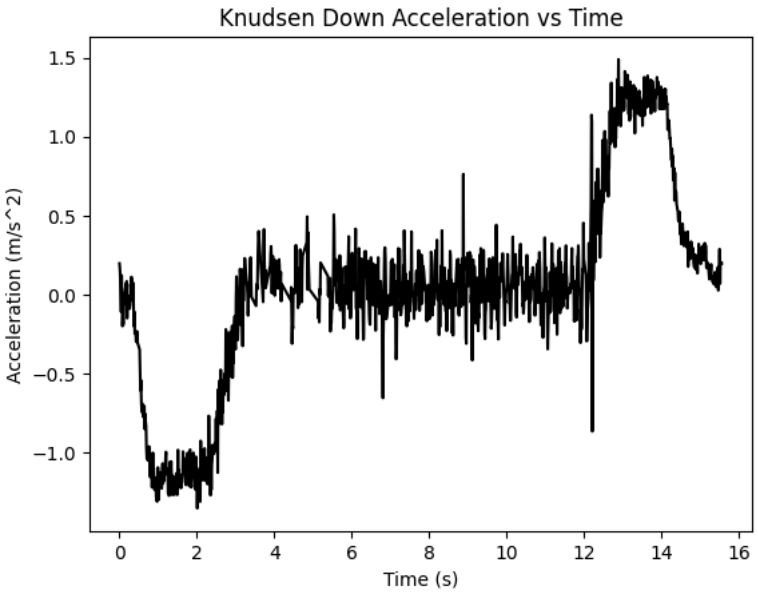


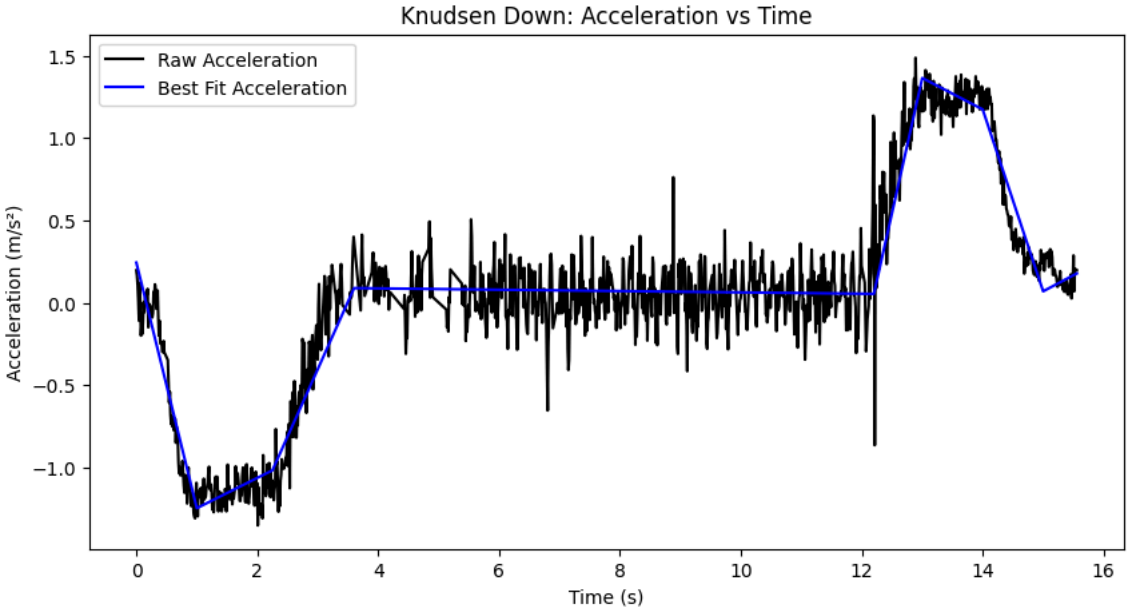

```
N = len(geology_up_z)
fs = 1000  # sampling frequency

# Remove DC offset
signal = output_to_accel_z(geology_up_z) - np.mean(output_to_accel_z(geology_up_z))

# Windowing
window = np.hanning(N)
yf = fft(signal * window)

xf = fftfreq(N, 1/fs)

# Only positive frequencies
mask = (xf >= 0) & (xf <= 25)  # zoom into 0–10 Hz

plt.plot(xf[mask], np.abs(yf)[mask] * 2 / N)
plt.title("Geology Up: Frequency Spectrum (0–25 Hz)")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Amplitude")
plt.show()
```



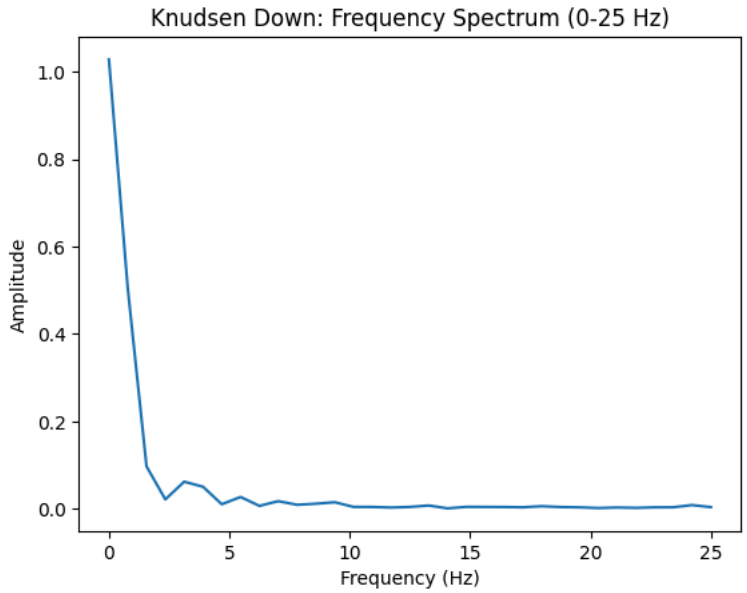

```
plt.plot(knudsen_down_t, output_to_accel_z_k(knudsen_down_z) - 9.81, c="black", label="Data")
plt.xlabel("Time (s)")
plt.ylabel("Acceleration (m/s^2)")
```

```
plt.title("Knudsen Down Acceleration vs Time")

breakpoints = np.array([min(knudsen_down_t), 1, 2.25, 3.6, 12.2, 13, 14, 15, max(knudsen_down_z)])

# initialize piecewise linear fit with your x and y data
my_pwlf = pwlf.PiecewiseLinFit(knudsen_down_t, output_to_accel_z_k(knudsen_down_z) - 9.81)

# initialize piecewise linear fit with your x and y data
my_pwlf.fit_with_breaks(breakpoints)

# predict for the determined points
xHat = np.linspace(min(knudsen_down_t), max(knudsen_down_t), num=10000)
yHat = my_pwlf.predict(xHat)

# plot the results
plt.figure(figsize=(10, 5))
plt.plot(knudsen_down_t, output_to_accel_z(knudsen_down_z) - 10.81, c="black", label="Raw Acceleration")
plt.plot(xHat, yHat, c="blue", label="Best Fit Acceleration")
plt.xlabel("Time (s)")
plt.ylabel("Acceleration (m/s²)")
plt.title("Knudsen Down: Acceleration vs Time")
plt.legend()

plt.figure(figsize=(10, 5))
jerk = np.gradient(yHat, xHat)
plt.plot(knudsen_down_t, output_to_accel_z(knudsen_down_z) - 10.81, c="black", label="Raw Acceleration")  # still showin
plt.plot(xHat, jerk, c="red", label="Best Fit Jerk")
plt.xlabel("Time (s)")
plt.ylabel("Jerk (m/s³)")
plt.title("Knudsen Down: Jerk vs Time")
plt.legend()

# Sampling frequency (Hz)
fs = 100

# Define bandpass filter (0.1 - 2 Hz)
lowcut = 0.1
highcut = 2
order = 4

nyq = 0.5 * fs
low = lowcut / nyq
high = highcut / nyq

# Create filter
b, a = butter(order, [low, high], btype='band')

# Apply filter
filtered_signal = filtfilt(b, a, output_to_accel_z(knudsen_down_z))

# Plot original and filtered signal
plt.figure(figsize=(12, 6))
plt.plot(output_to_accel_z(knudsen_down_z), label='Original Signal')
plt.plot(filtered_signal, label='Filtered Signal', linewidth=2)
plt.legend()
plt.title("Knudsen Down: Bandpass Filtered Signal (0.1 - 2 Hz)")
plt.xlabel("Sample")
plt.ylabel("Amplitude")
plt.grid(True)
plt.show()

se = my_pwlf.standard_errors()
print(se)
```



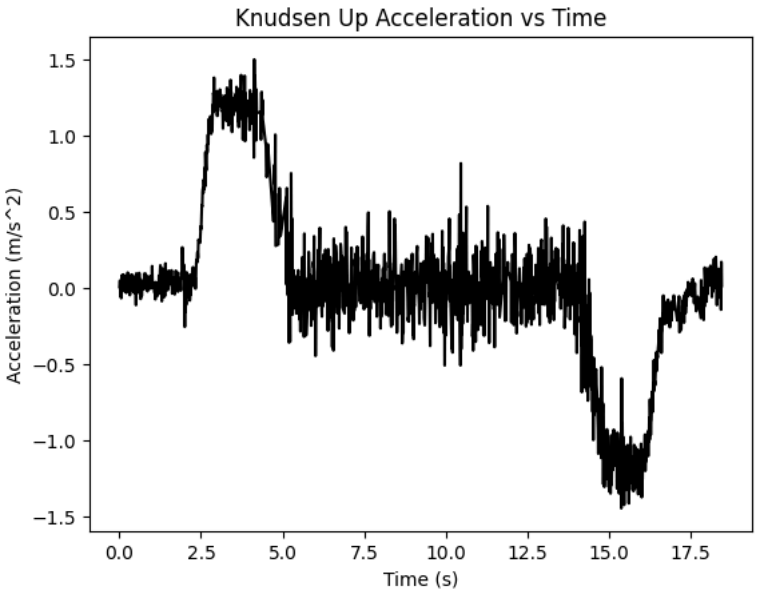


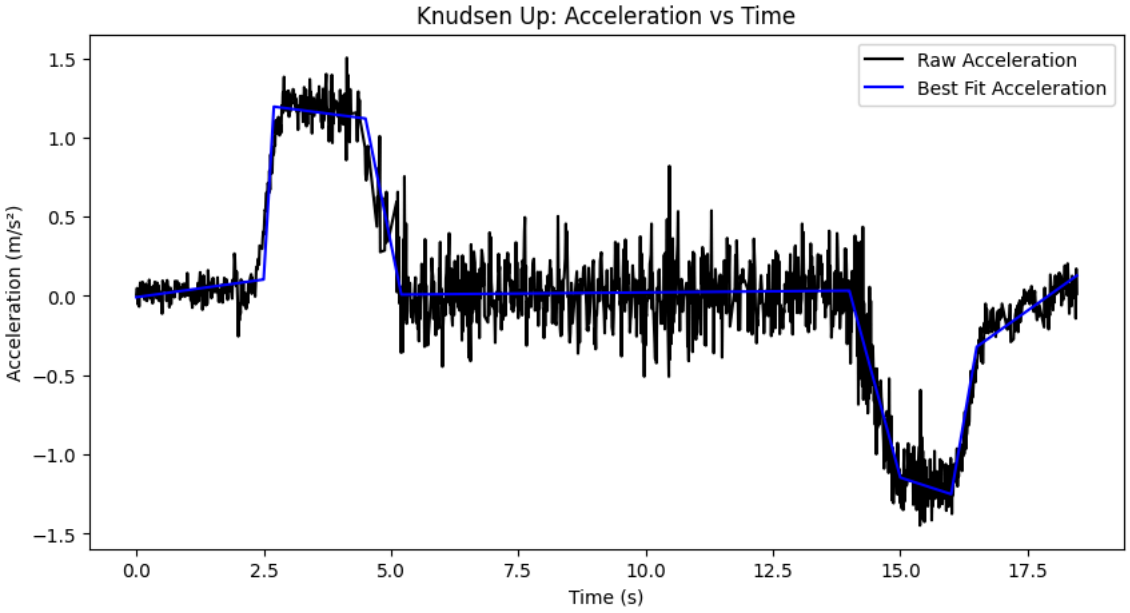

```
N = len(knudsen_down_z)
fs = 1000  # sampling frequency

# Remove DC offset
signal = output_to_accel_z(knudsen_down_z) - np.mean(output_to_accel_z_k(knudsen_down_z))

# Windowing
window = np.hanning(N)
yf = fft(signal * window)

xf = fftfreq(N, 1/fs)

# Only positive frequencies
mask = (xf >= 0) & (xf <= 25)  # zoom into 0–10 Hz

plt.plot(xf[mask], np.abs(yf)[mask] * 2 / N)
plt.title("Knudsen Down: Frequency Spectrum (0–25 Hz)")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Amplitude")
plt.show()
```

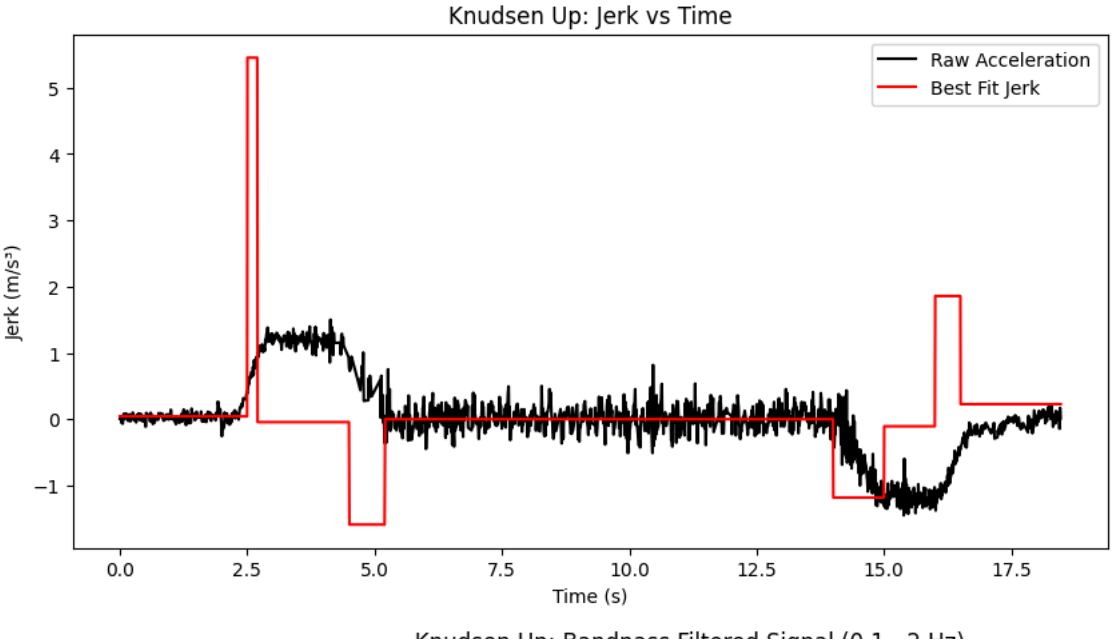



```
plt.plot(knudsen_up_t, output_to_accel_z_k(knudsen_up_z) - 9.81, c="black", label="Data")
plt.xlabel("Time (s)")
plt.ylabel("Acceleration (m/s^2)")
plt.title("Knudsen Up Acceleration vs Time")

breakpoints = np.array([min(knudsen_up_t), 2.5, 2.7, 4.5, 5.2, 14.0, 15.0, 16, 16.5, max(knudsen_up_z)])

# initialize piecewise linear fit with your x and y data
my_pwlf = pwlf.PiecewiseLinFit(knudsen_up_t, output_to_accel_z_k(knudsen_up_z) - 9.81)

# initialize piecewise linear fit with your x and y data
my_pwlf.fit_with_breaks(breakpoints)

# predict for the determined points
xHat = np.linspace(min(knudsen_up_t), max(knudsen_up_t), num=10000)
yHat = my_pwlf.predict(xHat)

# plot the results
plt.figure(figsize=(10, 5))
plt.plot(knudsen_up_t, output_to_accel_z(knudsen_up_z) - 10.81, c="black", label="Raw Acceleration")
plt.plot(xHat, yHat, c="blue", label="Best Fit Acceleration")
plt.xlabel("Time (s)")
plt.ylabel("Acceleration (m/s²)")
plt.title("Knudsen Up: Acceleration vs Time")
plt.legend()

plt.figure(figsize=(10, 5))
jerk = np.gradient(yHat, xHat)
plt.plot(knudsen_up_t, output_to_accel_z(knudsen_up_z) - 10.81, c="black", label="Raw Acceleration")  # still showing ra
plt.plot(xHat, jerk, c="red", label="Best Fit Jerk")
plt.xlabel("Time (s)")
plt.ylabel("Jerk (m/s³)")
plt.title("Knudsen Up: Jerk vs Time")
plt.legend()

# Sampling frequency (Hz)
fs = 100

# Define bandpass filter (0.1 - 2 Hz)
lowcut = 0.1
highcut = 2
order = 4

nyq = 0.5 * fs
low = lowcut / nyq
high = highcut / nyq

# Create filter
b, a = butter(order, [low, high], btype='band')

# Apply filter
```

```
filtered_signal = filtfilt(b, a, output_to_accel_z(knudsen_up_z))

# Plot original and filtered signal
plt.figure(figsize=(12, 6))
plt.plot(output_to_accel_z(knudsen_up_z), label='Original Signal')
plt.plot(filtered_signal, label='Filtered Signal', linewidth=2)
plt.legend()
plt.title("Knudsen Up: Bandpass Filtered Signal (0.1 - 2 Hz)")
plt.xlabel("Sample")
plt.ylabel("Amplitude")
plt.grid(True)
plt.show()

se = my_pwlf.standard_errors()
print(se)
```

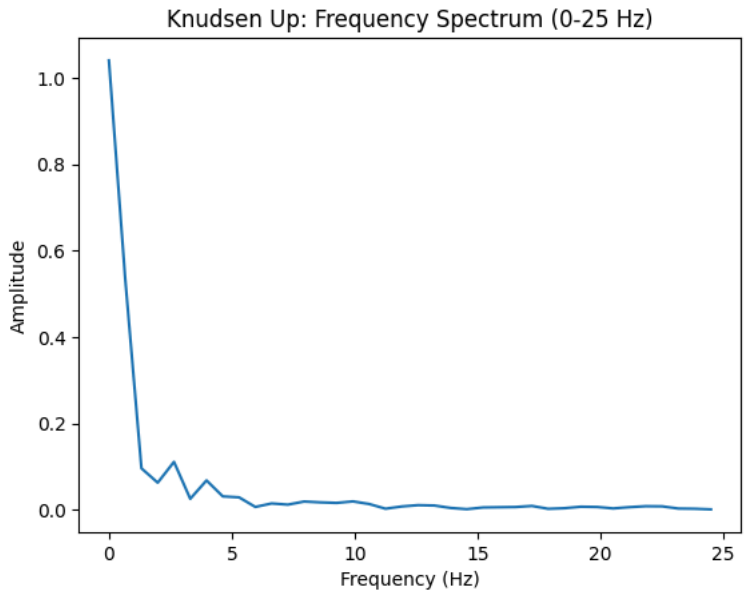

Knudsen Up Acceleration vs Time
Acceleration (m/s^2)
Time (s)


Knudsen Up: Acceleration vs Time
Raw Acceleration
Best Fit Acceleration
Acceleration (m/s²)
Time (s)


Knudsen Up: Jerk vs Time
Raw Acceleration
Best Fit Jerk
Jerk (m/s³)
Time (s)

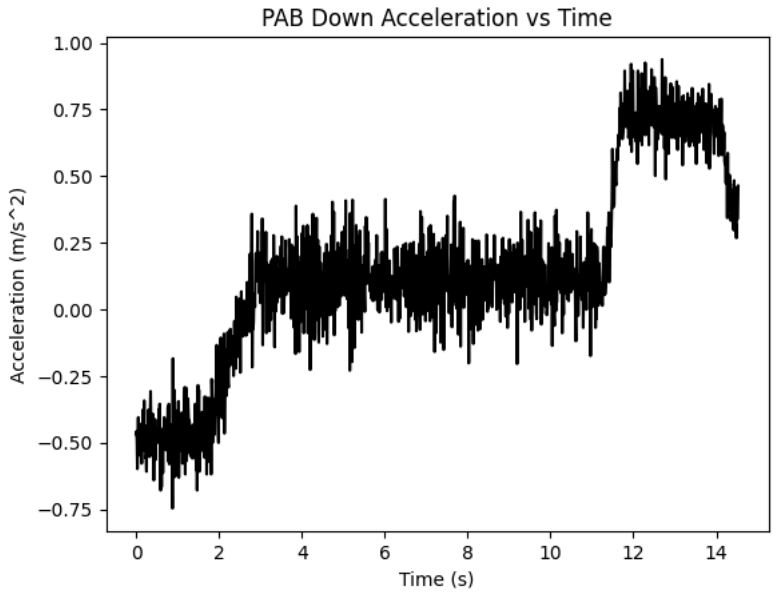
Start coding or generate with AI.

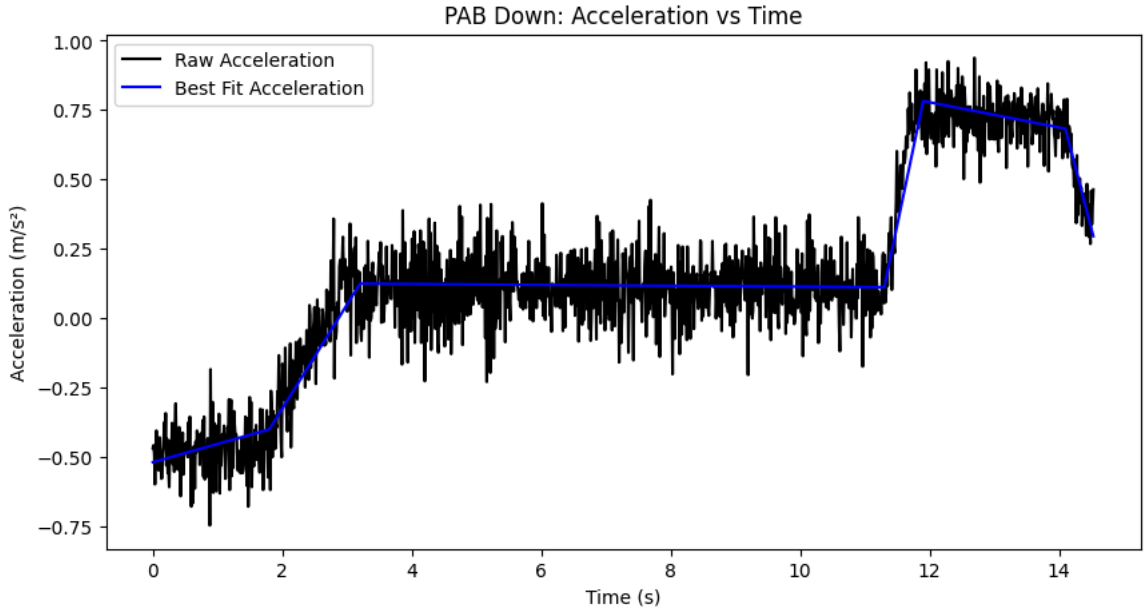
```
N = len(knudsen_up_z)
fs = 1000  # sampling frequency

# Remove DC offset
signal = output_to_accel_z(knudsen_up_z) - np.mean(output_to_accel_z_k(knudsen_up_z))

# Windowing
window = np.hanning(N)
yf = fft(signal * window)

xf = fftfreq(N, 1/fs)

# Only positive frequencies
mask = (xf >= 0) & (xf <= 25)  # zoom into 0–10 Hz

plt.plot(xf[mask], np.abs(yf)[mask] * 2 / N)
plt.title("Knudsen Up: Frequency Spectrum (0–25 Hz)")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Amplitude")
plt.show()
```

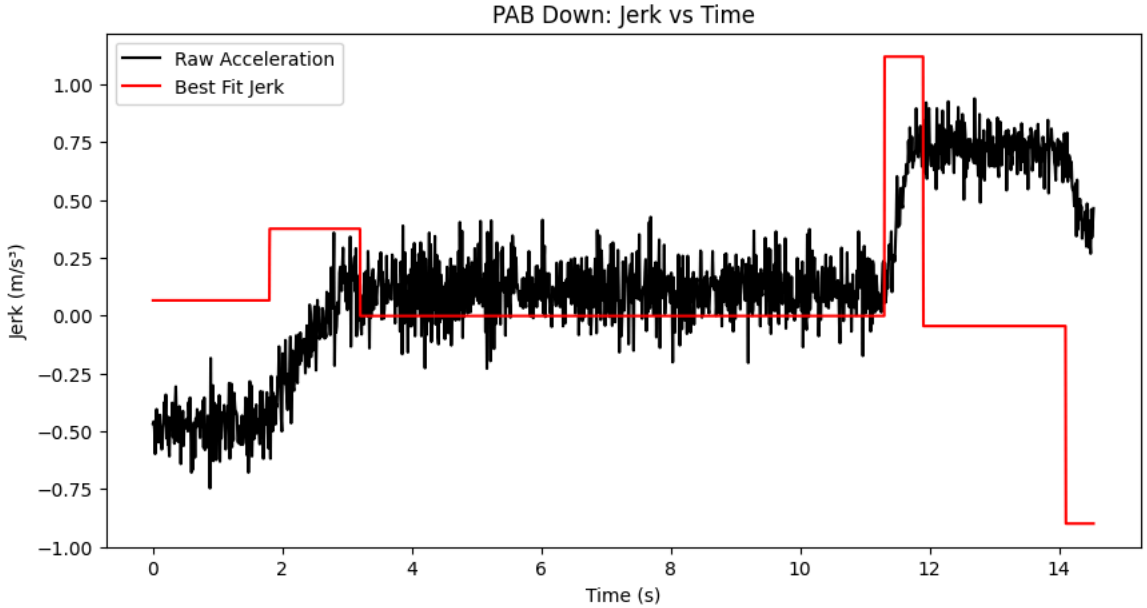



```
plt.plot(pab_down_t, output_to_accel_z(pab_down_z) - 9.81, c="black", label="Data")
plt.xlabel("Time (s)")
plt.ylabel("Acceleration (m/s^2)")
plt.title("PAB Down Acceleration vs Time")

breakpoints = np.array([min(pab_down_t), 1.8, 3.2, 11.3, 11.9, 14.1, max(pab_down_z)])

# initialize piecewise linear fit with your x and y data
my_pwlf = pwlf.PiecewiseLinFit(pab_down_t, output_to_accel_z(pab_down_z) - 9.81)

# initialize piecewise linear fit with your x and y data
my_pwlf.fit_with_breaks(breakpoints)

# predict for the determined points
xHat = np.linspace(min(pab_down_t), max(pab_down_t), num=10000)
yHat = my_pwlf.predict(xHat)

# plot the results
plt.figure(figsize=(10, 5))
plt.plot(pab_down_t, output_to_accel_z(pab_down_z) - 9.81, c="black", label="Raw Acceleration")
plt.plot(xHat, yHat, c="blue", label="Best Fit Acceleration")
plt.xlabel("Time (s)")
plt.ylabel("Acceleration (m/s²)")
plt.title("PAB Down: Acceleration vs Time")
plt.legend()

plt.figure(figsize=(10, 5))
jerk = np.gradient(yHat, xHat)
```

```
plt.plot(pab_down_t, output_to_accel_z(pab_down_z) - 9.81, c="black", label="Raw Acceleration")  # still showing raw dat
plt.plot(xHat, jerk, c="red", label="Best Fit Jerk")
plt.xlabel("Time (s)")
plt.ylabel("Jerk (m/s³)")
plt.title("PAB Down: Jerk vs Time")
plt.legend()

# Sampling frequency (Hz)
fs = 100

# Define bandpass filter (0.1 - 2 Hz)
lowcut = 0.1
highcut = 2
order = 4

nyq = 0.5 * fs
low = lowcut / nyq
high = highcut / nyq

# Create filter
b, a = butter(order, [low, high], btype='band')

# Apply filter
filtered_signal = filtfilt(b, a, output_to_accel_z(pab_down_z))

# Plot original and filtered signal
plt.figure(figsize=(12, 6))
plt.plot(output_to_accel_z(pab_down_z), label='Original Signal')
plt.plot(filtered_signal, label='Filtered Signal', linewidth=2)
plt.legend()
plt.title("PAB Down: Bandpass Filtered Signal (0.1 - 2 Hz)")
plt.xlabel("Sample")
plt.ylabel("Amplitude")
plt.grid(True)
plt.show()

se = my_pwlf.standard_errors()
print(se)
```

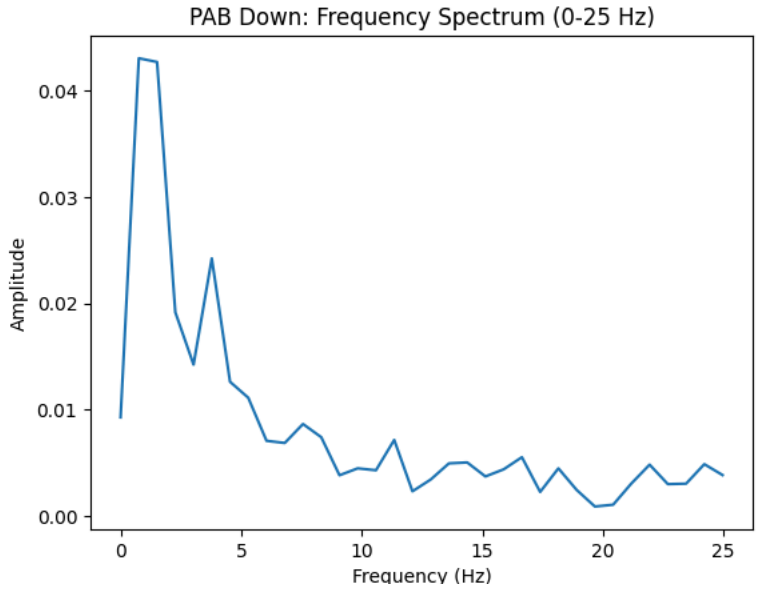






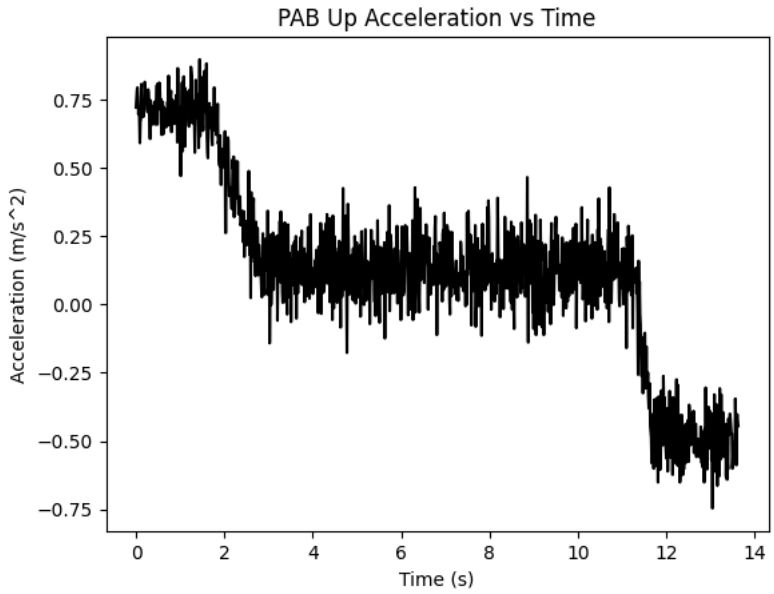
Start coding or generate with AI.

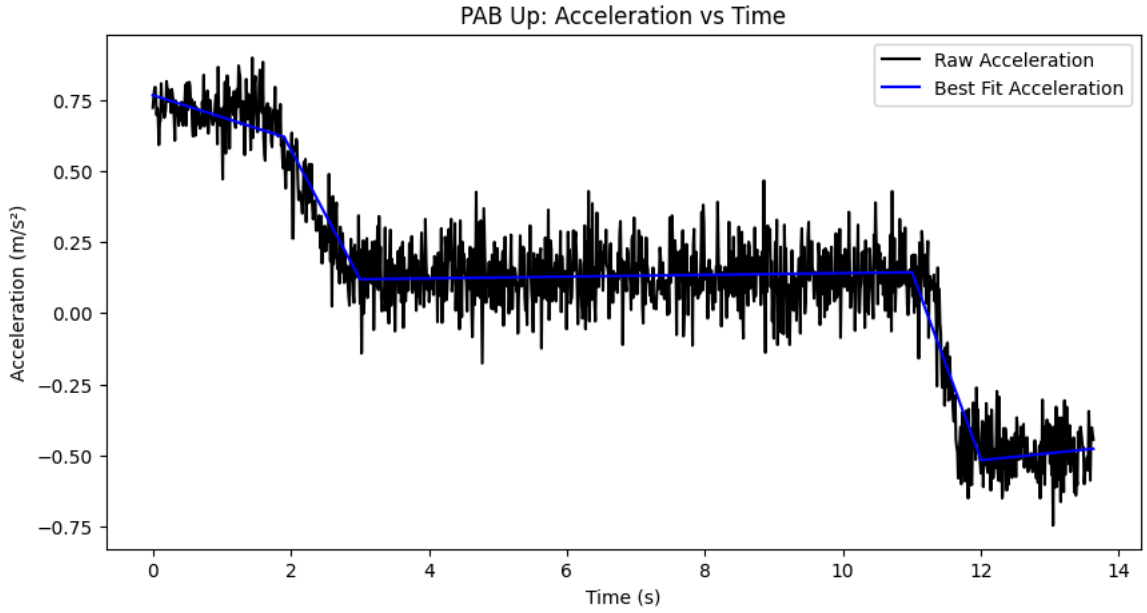
```
N = len(pab_down_z)
fs = 1000  # sampling frequency

# Remove DC offset
signal = output_to_accel_z(pab_down_z) - np.mean(output_to_accel_z(pab_down_z))

# Windowing
window = np.hanning(N)
yf = fft(signal * window)

xf = fftfreq(N, 1/fs)

# Only positive frequencies
mask = (xf >= 0) & (xf <= 25)  # zoom into 0–10 Hz

plt.plot(xf[mask], np.abs(yf)[mask] * 2 / N)
plt.title("PAB Down: Frequency Spectrum (0–25 Hz)")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Amplitude")
plt.show()
```

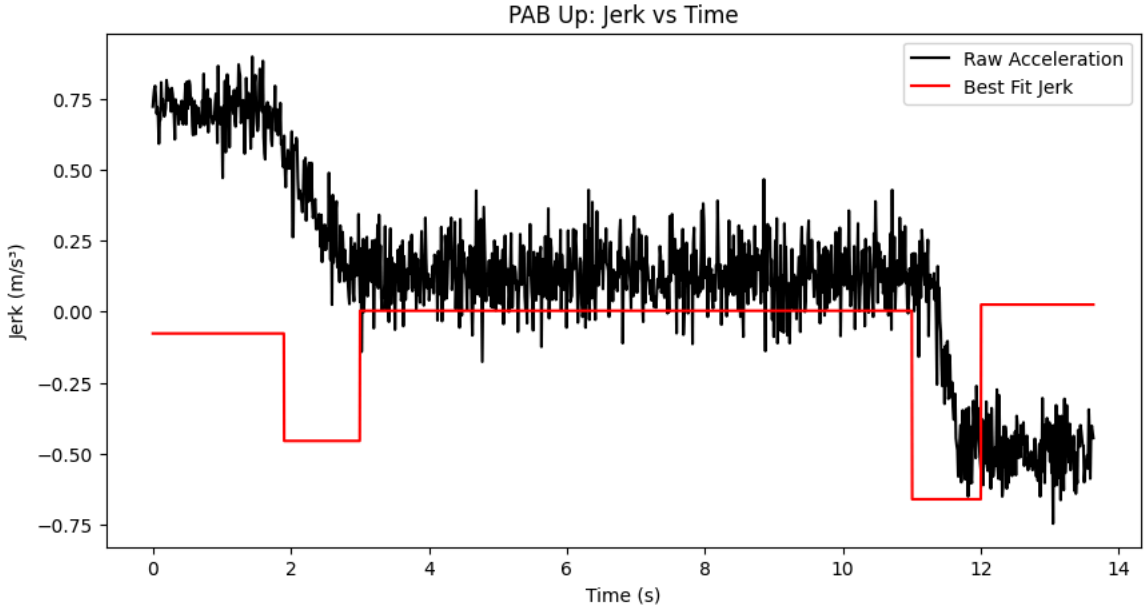



```
plt.plot(pab_up_t, output_to_accel_z(pab_up_z) - 9.81, c="black", label="Data")
plt.xlabel("Time (s)")
plt.ylabel("Acceleration (m/s^2)")
plt.title("PAB Up Acceleration vs Time")

breakpoints = np.array([min(pab_up_t), 1.9, 3, 11, 12, max(pab_up_z)])

# initialize piecewise linear fit with your x and y data
my_pwlf = pwlf.PiecewiseLinFit(pab_up_t, output_to_accel_z(pab_up_z) - 9.81)

# initialize piecewise linear fit with your x and y data
my_pwlf.fit_with_breaks(breakpoints)

# predict for the determined points
xHat = np.linspace(min(pab_up_t), max(pab_up_t), num=10000)
yHat = my_pwlf.predict(xHat)

# plot the results
plt.figure(figsize=(10, 5))
plt.plot(pab_up_t, output_to_accel_z(pab_up_z) - 9.81, c="black", label="Raw Acceleration")
plt.plot(xHat, yHat, c="blue", label="Best Fit Acceleration")
plt.xlabel("Time (s)")
plt.ylabel("Acceleration (m/s²)")
plt.title("PAB Up: Acceleration vs Time")
plt.legend()

plt.figure(figsize=(10, 5))
jerk = np.gradient(yHat, xHat)
plt.plot(pab_up_t, output_to_accel_z(pab_up_z)- 9.81, c="black", label="Raw Acceleration")  # still showing raw data
plt.plot(xHat, jerk, c="red", label="Best Fit Jerk")
```

```
plt.xlabel("Time (s)")
plt.ylabel("Jerk (m/s³)")
plt.title("PAB Up: Jerk vs Time")
plt.legend()

# Sampling frequency (Hz)
fs = 100

# Define bandpass filter (0.1 - 2 Hz)
lowcut = 0.05
highcut = 2
order = 4

nyq = 0.5 * fs
low = lowcut / nyq
high = highcut / nyq

# Create filter
b, a = butter(order, [low, high], btype='band')

# Apply filter
filtered_signal = filtfilt(b, a, output_to_accel_z(pab_up_z))

# Plot original and filtered signal
plt.figure(figsize=(12, 6))
plt.plot(output_to_accel_z(pab_up_z), label='Original Signal')
plt.plot(filtered_signal, label='Filtered Signal', linewidth=2)
plt.legend()
plt.title("PAB Up: Bandpass Filtered Signal (0.1 - 2 Hz)")
plt.xlabel("Sample")
plt.ylabel("Amplitude")
plt.grid(True)
plt.show()

se = my_pwlf.standard_errors()
print(se)
```


PAB Up Acceleration vs Time
Acceleration (m/s^2)
Time (s)


PAB Up: Acceleration vs Time
Raw Acceleration
Best Fit Acceleration
Acceleration (m/s²)
Time (s)


PAB Up: Jerk vs Time
Raw Acceleration
Best Fit Jerk
Jerk (m/s³)
Time (s)