Smartphone - Based Sensing of Pendulum Vibrations

Table of contents

Instructor: Mohammad Talebi-Kalaleh
Course: Sensing Techniques and Data Analytics — Fall 2025

Student ID: `STUDENT_ID`
Name: Your Name Here
Date: YYYY-MM-DD

Tool Chains: `Jupyter->Quarto->PDF`

1 Background

1.1 Assignment Outline

1. Setup & Imports
2. Utility functions (data loading, peak detection, damping fit, stats)
3. Part A — Short Pendulum (L 0.3 m)
4. Part B — Long Pendulum (L 1.0 m)
5. Part C — Nonlinear Regime (large angles)
6. Part D — Noise discussion
7. Figures & Data saving helpers
8. Submission checklist

Each analysis part includes: data loading, visualization, statistical descriptors, period estimation, damping estimation, and discussion cells.

```
import os
import glob
import numpy as np
import pandas as pd
import scipy.signal as signal
```

```
import scipy.optimize as optimize
from scipy.stats import skew, kurtosis
import matplotlib.pyplot as plt
from IPython.display import display, Markdown

# Ensure figures render in notebook
%matplotlib inline

# Reproducible plotting size policy
plt.rcParams['figure.figsize'] = (10,4)

# Paths (edit STUDENT_ID below)
STUDENT_ID = "STUDENT_ID"
BASE_DIR = f"."
DATA_DIR = os.path.join(BASE_DIR, 'Data')
FIG_DIR = os.path.join(BASE_DIR, 'Figures')
```

1.2 Experiment Preparation

1.2.1 Materials

1. Smartphone: Type: Xiaomi 10 Target sensor: accelerator
2. String
3. Toilet paper roll
4. Camera: Another smartphone
5. Softer & Version: PhyPhox 1.2.0
6. Cross beam: 1 Chopstick
7. Angle measurement: Compass App of Smartphone

1.2.2 Structure

The pendulum device was constructed as follows:

- A chopstick was used as the cross beam and fixed horizontally to serve as the support.
- Four lengths of string were cut and tied to the chopstick, spaced evenly to ensure stability.
- The lower ends of the strings were attached to a toilet paper roll, which acted as the pendulum bob holder.
- A smartphone (Xiaomi 10) was placed securely inside the toilet paper roll, with its main axis (y) aligned along the swing direction.
- The PhyPhox app (version 1.2.0) was installed on the smartphone to record acceleration data.

- A second smartphone was used as a camera to record the motion for reference and angle measurement.
- The initial angle of displacement was measured using the compass app on the smartphone before release.

This setup allowed for accurate measurement of pendulum motion using the smartphone's accelerometer, while the camera provided visual confirmation and angle calibration.

1.3 Theoretical Model

1.3.1 Small-Angle Pendulum Period

For small angular displacements ($\theta \leq 15^\circ$), the period of a simple pendulum is given by Equation 1:

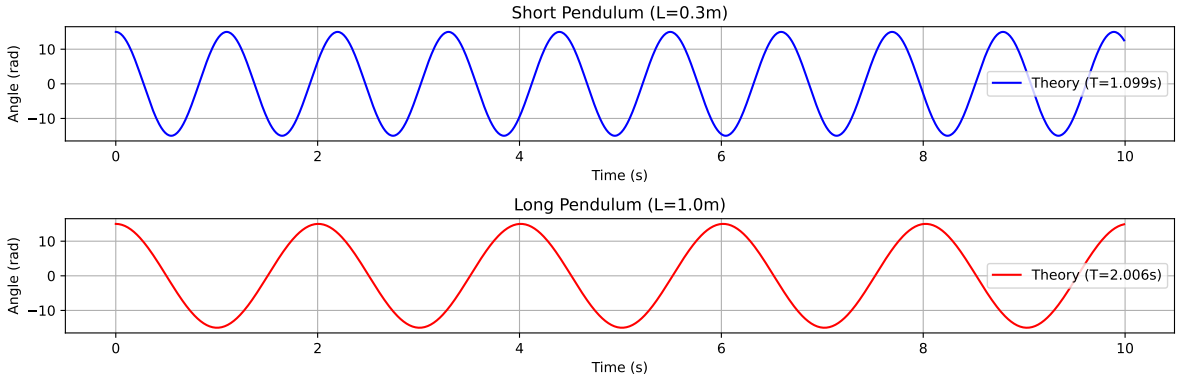
$$T_n = 2\pi\sqrt{\frac{L}{g}} \tag{1}$$

where:

- T_n = oscillation period (seconds)
- L = pendulum length (meters)
- g = gravitational acceleration $(9.81, m/s^2)$

```python
def generate_theoretical_pendulum(L, t_max=10, dt=0.01, theta0=15, g=9.81):
    """Generate theoretical pendulum motion for small angles
    Args:
        L (float): pendulum length in meters
        t_max (float): simulation duration in seconds
        dt (float): time step in seconds
        theta0 (float): initial angle in radians
        g (float): gravitational acceleration (m/s^2)
    Returns:
        t, theta: time and angle arrays
    """
    t = np.arange(0, t_max, dt)
    omega = np.sqrt(g/L)  # natural frequency
    theta = theta0 * np.cos(omega * t)  # solution for small angles
    return t, theta

def theoretical_period(L, g=9.81):
    """Calculate theoretical period of a simple pendulum"""
    return 2 * np.pi * np.sqrt(L / g)
```

```
# Generate and plot theoretical data for both pendulum lengths
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 4))

# Short pendulum
L_short = 0.3 # part A
t_short, theta_short = generate_theoretical_pendulum(L_short)
T_short = theoretical_period(L_short)
ax1.plot(t_short, theta_short, 'b-', label=f'Theory (T={T_short:.3f}s)')
ax1.set_title(f'Short Pendulum (L={L_short}m)')
ax1.set_xlabel('Time (s)')
ax1.set_ylabel('Angle (rad)')
ax1.grid(True)
ax1.legend()

# Long pendulum
L_long = 1.0 # part B
t_long, theta_long = generate_theoretical_pendulum(L_long)
T_long = theoretical_period(L_long)
ax2.plot(t_long, theta_long, 'r-', label=f'Theory (T={T_long:.3f}s)')
ax2.set_title(f'Long Pendulum (L={L_long}m)')
ax2.set_xlabel('Time (s)')
ax2.set_ylabel('Angle (rad)')
ax2.grid(True)
ax2.legend()

plt.tight_layout()
plt.show()
print(f'Theoretical periods:\nShort pendulum: {T_short:.3f}s\nLong pendulum: {T_long:.3f}s')
```



Theoretical periods:

```
Short pendulum: 1.099s
Long pendulum: 2.006s
```

1.3.2 Damped Oscillation Model

The amplitude envelope of a damped pendulum follows exponential decay, as shown in Equation 2:

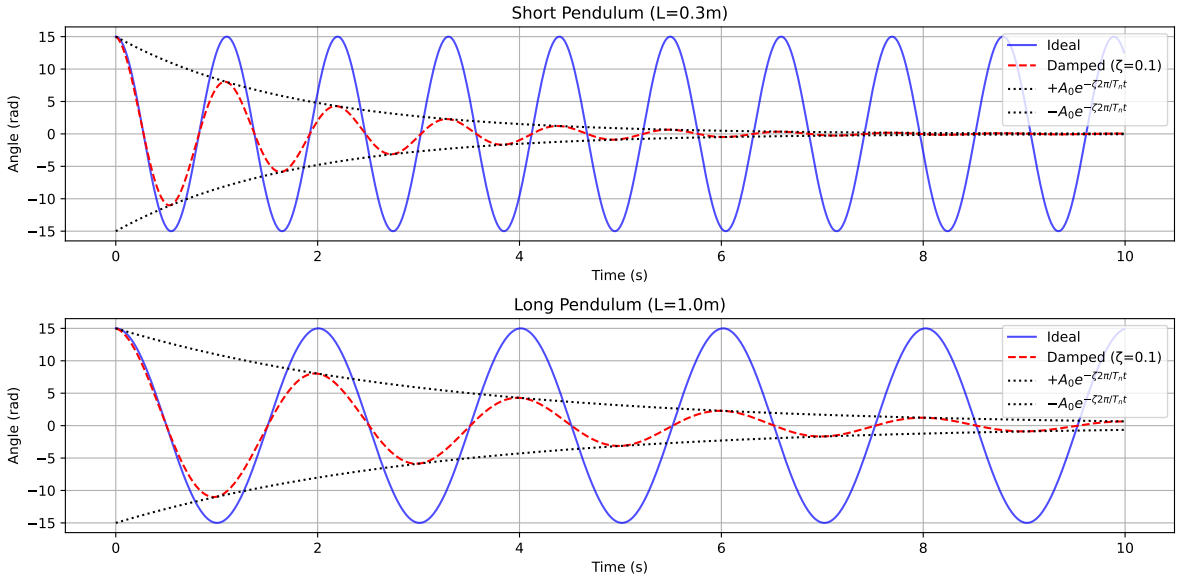
$$A(t) = A_0 \exp(-\zeta \frac{2\pi}{T_n} t) \tag{2}$$

where:

- $A(t)$ = amplitude at time t
- A_0 = initial amplitude
- ζ = damping ratio (dimensionless)
- t = elapsed time (seconds)

```python
def generate_damped_pendulum(L, t_max=10, dt=0.01, theta0=15, zeta=0.1, g=9.81):
    """Generate damped pendulum motion
    Args:
        L (float): pendulum length in meters
        t_max (float): simulation duration in seconds
        dt (float): time step in seconds
        theta0 (float): initial angle in radians
        zeta (float): damping ratio (dimensionless)
        g (float): gravitational acceleration (m/s^2)
    Returns:
        t, theta: time and angle arrays
    """
    t = np.arange(0, t_max, dt)
    omega = np.sqrt(g/L)  # natural frequency
    # Damped solution
    theta = theta0 * np.exp(-zeta * omega * t) * np.cos(omega * t)
    return t, theta

# Plot comparison of ideal vs damped oscillation
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 6))

# Short pendulum comparison
t_short_damped, theta_short_damped = generate_damped_pendulum(L_short, zeta=0.1)
ax1.plot(t_short, theta_short, 'b-', label='Ideal', alpha=0.7)
ax1.plot(t_short_damped, theta_short_damped, 'r--', label='Damped ( =0.1)')
ax1.set_title(f'Short Pendulum (L={L_short}m)')
```

```
ax1.set_xlabel('Time (s)')
ax1.set_ylabel('Angle (rad)')
ax1.grid(True)

# Long pendulum comparison
t_long_damped, theta_long_damped = generate_damped_pendulum(L_long, zeta=0.1)
ax2.plot(t_long, theta_long, 'b-', label='Ideal', alpha=0.7)
ax2.plot(t_long_damped, theta_long_damped, 'r--', label='Damped ( =0.1)')
ax2.set_title(f'Long Pendulum (L={L_long}m)')
ax2.set_xlabel('Time (s)')
ax2.set_ylabel('Angle (rad)')
ax2.grid(True)

# Add the +- A_0 exp(-zeta 2pi/T_n t) lines
for ax, t, theta0, Tn in [
    (ax1, t_short_damped, 15, theoretical_period(L_short)),
    (ax2, t_long_damped, 15, theoretical_period(L_long))
]:
    envelope = theta0 * np.exp(-0.1 * 2 * np.pi / Tn * t)
    ax.plot(t, envelope, 'k:', label=r'$+A_0 e^{-\zeta 2\pi/T_n t}$')
    ax.plot(t, -envelope, 'k:', label=r'$-A_0 e^{-\zeta 2\pi/T_n t}$')

ax1.legend(loc='upper right')
ax2.legend(loc='upper right')

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



2 Data Preprocessing

```python
import zipfile
import os
import glob
import pandas as pd

def extract_raw_data_from_zip_files(folder_path, experiment):
    """
    Finds all zip files in a folder, extracts 'Raw Data.csv' from each,
    and saves it with the zip file's name in a new 'extracted_data' folder.

    Args:
        folder_path (str): The path to the folder containing zip files.
    """
    # Define the name of the CSV file and the output directory
    csv_filename = "Raw Data.csv"
    output_dir = os.path.join(folder_path,"")

    # Create the output directory if it doesn't exist
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
        print(f"Created output directory: {output_dir}")
```

```
    # Find all zip files in the specified folder
    zip_files = glob.glob(os.path.join(output_dir,experiment, "*.zip"))

    if not zip_files:
        print("No zip files found in the specified folder. Exiting.")
        return

    print(f"Found {len(zip_files)} zip files to process.")

    for zip_filepath in zip_files:
        try:
            with zipfile.ZipFile(zip_filepath, 'r') as zip_ref:
                # Get the base name of the zip file without the extension
                base_name = os.path.splitext(os.path.basename(zip_filepath))[0]

                # Find the path to the CSV file inside the zip, accounting for nesting
                csv_member_path = None
                for member in zip_ref.namelist():
                    if member.endswith(csv_filename):
                        csv_member_path = member
                        break

                if not csv_member_path:
                    print(f"Warning: '{csv_filename}' not found in '{os.path.basename(zip_fi
                    continue

                # Extract the file to a temporary location
                temp_extract_dir = os.path.join(output_dir, base_name)
                zip_ref.extract(csv_member_path, temp_extract_dir)

                # The extracted file's path
                extracted_file_path = os.path.join(temp_extract_dir, csv_member_path)

                # Define the new filename and path
                new_csv_filename = f"{base_name}.csv"
                new_csv_path = os.path.join(output_dir, new_csv_filename)

                # Move and rename the extracted file
                os.rename(extracted_file_path, new_csv_path)

                # Clean up the temporary directory structure
                os.rmdir(os.path.dirname(extracted_file_path))
```

```
                print(f"Successfully extracted and renamed '{os.path.basename(zip_filepath)}

        except (zipfile.BadZipFile, FileNotFoundError) as e:
            print(f"Error processing '{os.path.basename(zip_filepath)}': {e}")
        except Exception as e:
            print(f"An unexpected error occurred with '{os.path.basename(zip_filepath)}': {e

if __name__ == "__main__" and False:
    folder_path = "./Data"
    experiment_type = "/Raw/partC"

    # Run the function to process the files
    extract_raw_data_from_zip_files(folder_path, experiment=experiment_type)
```

```
def load_trial_csv(path,start=500,end=10000):
    """Load CSV exported from PhyPhox.
    Assumes time column 'time' (s) and acceleration axis 'accel' aligned with swing directio
    Adjust column names below if your CSV uses different headers."""
    df = pd.read_csv(path)
    time_col = df.columns[0]
    # choose accel axis heuristically
    accel_col = df.columns[2]
    df = df.rename(columns={time_col: 'time', accel_col: 'accel'})
    return df[['time','accel']].iloc[start:end]  # trim start/end
```

```
def estimate_period_via_peaks(time, signal_data, prominence=0.5, distance=None):
    # find peaks (positive swings)
    peaks, props = signal.find_peaks(signal_data, prominence=prominence, distance=distance)
    if len(peaks) < 2:
        return np.nan, peaks
    peak_times = time[peaks]
    periods = np.diff(peak_times)
    return periods, peaks
```

```
def fit_exponential_decay(peak_times, peak_amps, Tn=None):
    """Fit A(t) = A0 * exp(-zeta * 2*pi / Tn * t)
    If Tn is None, fit exponent parameter directly as A0 * exp(-k t) and compute zeta from k
    Returns dict with A0, zeta (if possible), k (decay rate), covariance.
    """
```

```
    # fit ln(A) = ln(A0) - k t  where k = zeta * 2*pi / Tn
    positive_mask = peak_amps > 0
    t = np.array(peak_times)[positive_mask]
    A = np.array(peak_amps)[positive_mask]
    if len(A) < 2:
        return {'A0': np.nan, 'k': np.nan, 'zeta': np.nan}
    logA = np.log(A)
    coef = np.polyfit(t, logA, 1)
    k = -coef[0]  # because logA = ln(A0) - k t
    A0 = np.exp(coef[1])
    zeta = np.nan
    if Tn is not None and Tn>0:
        zeta = k * Tn / (2*np.pi)
    return {'A0': A0, 'k': k, 'zeta': zeta}
```

```
def theoretical_period(L, g=9.81):
    return 2*np.pi*np.sqrt(L/g)
```

3 Part A — Short Pendulum (L 0.3 m)

Instructions: Place CSV files for Part A into `HW1_STUDENTID/Data/` with names like `partA_trial01.csv` … `partA_trial10.csv`. The code cells below will loop through trials, compute descriptors, estimate period, fit damping, and summarize results.

3.1 Data Collection

3.2 Statistical Analysis

For part A, I collect 11 trails, For each trial, compute: • Mean (μ) • Variance (2) • Root Mean Square (RMS) • Skewness (1) • Kurtosis (2)

List all Part A trials and compute descriptors, output as Table 1, for assuring the data consistence, exclude the begining and the ending data, only keep the middle (from t=500 to t=10000)

Table 1: Summary of Part A

	file	Mean (μ)	Variance (^{2})	Root Mean Square (RMS)	Skewnes
0	partA_trial01_2025-09-16 15-55-52.csv	-0.024352	1.334183	1.155264	-0.00366

Table 1: Summary of Part A

	file	Mean (μ)	Variance (²)	Root Mean Square (RMS)	Skewnes
1	partA_trial02_2025-09-16 15-59-44.csv	-0.027097	1.109700	1.053716	-0.00506
2	partA_trial03_2025-09-16 16-02-50.csv	-0.022610	1.665985	1.290860	-0.00242
3	partA_trial04_2025-09-16 16-06-16.csv	-0.006346	2.023282	1.422360	-0.02675
4	partA_trial05_2025-09-16 16-10-40.csv	-0.012029	2.386752	1.544877	-0.01472
5	partA_trial06_2025-09-16 16-15-06.csv	0.000310	2.444235	1.563323	-0.02677
6	partA_trial07_2025-09-16 16-19-42.csv	-0.009750	2.326952	1.525386	-0.02319
7	partA_trial08_2025-09-16 16-22-50.csv	-0.011337	1.915803	1.384099	-0.01847
8	partA_trial09_2025-09-16 16-25-21.csv	0.005203	2.635221	1.623259	-0.03238
9	partA_trial10_2025-09-16 16-27-49.csv	-0.014440	1.509034	1.228448	-0.02180
10	partA_trial11_2025-09-16 16-30-35.csv	-0.007640	2.238456	1.496088	-0.02591

3.3 Period Estimation

```
L_short = 0.3  # meters (adjust if measured)
file_pattern = os.path.join(DATA_DIR, 'partA_trial*.csv')
print(file_pattern)
files = sorted(glob.glob(file_pattern))
print(f"Found {len(files)} files for Part A analysis.")
results = []

for fpath in files:
    df = load_trial_csv(fpath)
    t = df['time'].values
    x = df['accel'].values
    # basic plot
    plt.figure(); plt.plot(t, x); plt.title(f'Raw accel: {os.path.basename(fpath)}'); plt.xla
    # descriptors
    desc = compute_descriptors(x)
    # smoothing for peak detection (optional)
    x_smooth = signal.savgol_filter(x, 11, 3)
    periods, peaks = estimate_period_via_peaks(t, x_smooth, prominence=np.std(x_smooth)*0.5)
    if isinstance(periods, np.ndarray):
        mean_period = np.mean(periods)
        std_period = np.std(periods, ddof=1)
    else:
        mean_period = np.nan
        std_period = np.nan
```

```
    # peak amplitudes and times
    peak_times = t[peaks] if len(peaks)>0 else np.array([])
    peak_amps = np.abs(x_smooth[peaks]) if len(peaks)>0 else np.array([])
    decay = fit_exponential_decay(peak_times, peak_amps, Tn=mean_period)
    result = {
        'file': os.path.basename(fpath),
        'mean_period': mean_period,
        'std_period': std_period,
        'A0': decay['A0'],
        'k': decay['k'],
        'zeta': decay['zeta'],
        **desc
    }
    results.append(result)

results_df = pd.DataFrame(results)
results_df
```

```
.\Data\partA_trial*.csv
Found 11 files for Part A analysis.
```

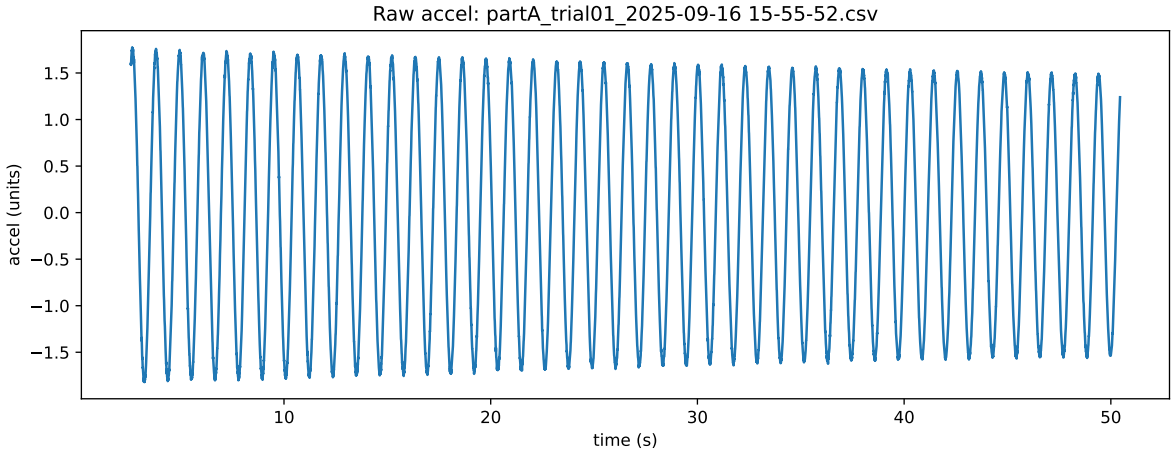
::: {#part-a-analysis-loop-(short-pendulum .cell-output .cell-output-display execution_count=26}

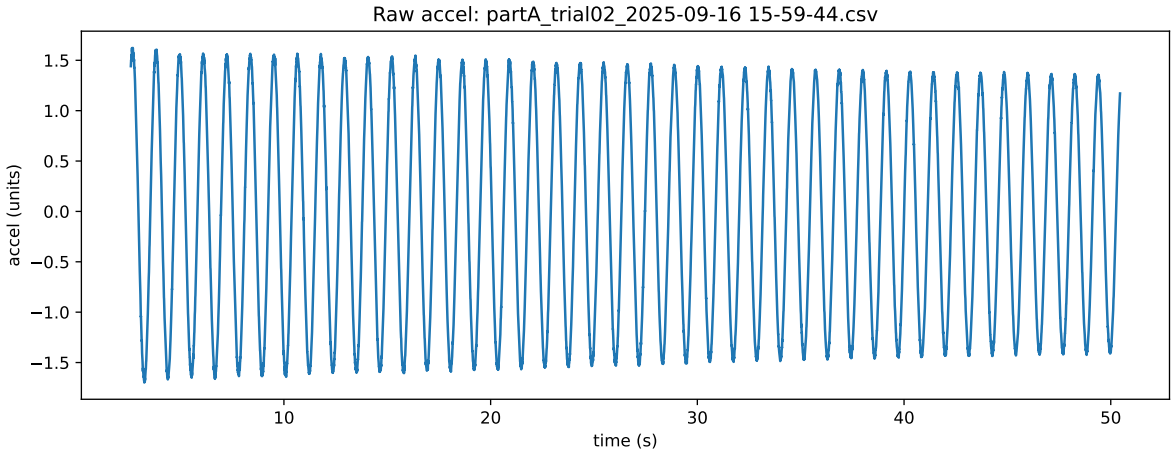
	file	mean_period	std_period	A0	k	zeta	M
0	partA_trial01_2025-09-16 15-55-52.csv	1.140048	0.010835	1.755053	0.003487	0.000633	-0
1	partA_trial02_2025-09-16 15-59-44.csv	1.139430	0.013493	1.597898	0.003552	0.000644	-0
2	partA_trial03_2025-09-16 16-02-50.csv	1.140452	0.011391	1.966872	0.003497	0.000635	-0
3	partA_trial04_2025-09-16 16-06-16.csv	1.141591	0.010255	2.137263	0.002973	0.000540	-0
4	partA_trial05_2025-09-16 16-10-40.csv	1.142225	0.009813	2.334413	0.003044	0.000553	-0
5	partA_trial06_2025-09-16 16-15-06.csv	1.142144	0.010936	2.358323	0.002996	0.000545	0
6	partA_trial07_2025-09-16 16-19-42.csv	1.142098	0.012245	2.298571	0.003101	0.000564	-0
7	partA_trial08_2025-09-16 16-22-50.csv	1.141347	0.013735	2.079646	0.003034	0.000551	-0
8	partA_trial09_2025-09-16 16-25-21.csv	1.142880	0.012110	2.444768	0.002981	0.000542	0
9	partA_trial10_2025-09-16 16-27-49.csv	1.140544	0.013974	1.846905	0.003120	0.000566	-0
10	partA_trial11_2025-09-16 16-30-35.csv	1.142019	0.011892	2.247809	0.003039	0.000552	-0

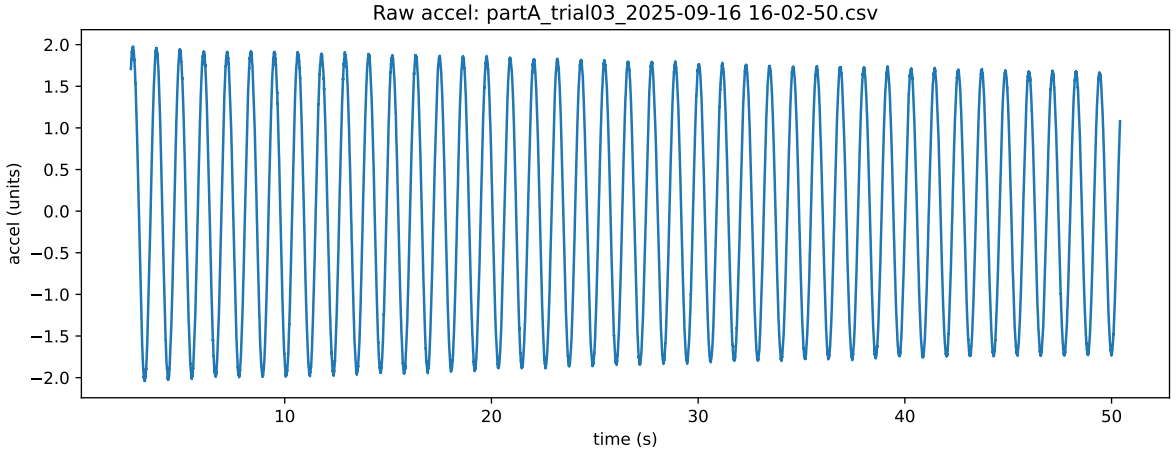
:::

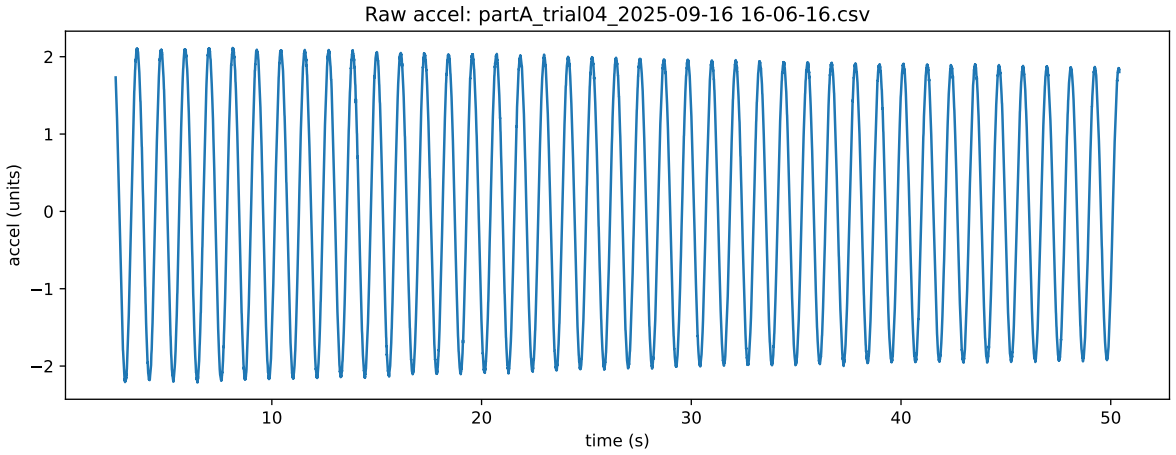
```
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 10))

# Plot 1: Raw vs Smoothed Signal
ax1.plot(t, x, 'b-', alpha=0.5, label='Raw')
ax1.plot(t, x_smooth, 'r-', label='Smoothed')
ax1.set_title('Signal Comparison')
ax1.set_xlabel('Time (s)')
ax1.set_ylabel('Acceleration')
ax1.grid(True)
ax1.legend()

# Plot 2: Peak Detection
ax2.plot(t, x_smooth, 'b-')
if len(peaks) > 0:
    ax2.plot(t[peaks], x_smooth[peaks], 'ro', label='Peaks')
ax2.set_title('Peak Detection')
ax2.set_xlabel('Time (s)')
ax2.set_ylabel('Acceleration')
ax2.grid(True)
ax2.legend()

# Plot 3: Period Analysis
if isinstance(periods, np.ndarray) and len(periods) > 0:
    ax3.plot(periods, 'g.-')
    ax3.axhline(y=np.mean(periods), color='r', linestyle='--',
                label=f'Mean: {np.mean(periods):.3f}s')
    ax3.set_title('Period Variation')
    ax3.set_xlabel('Peak Index')
    ax3.set_ylabel('Period (s)')
    ax3.grid(True)
    ax3.legend()

# Plot 4: Peak Amplitude Decay
if len(peak_times) > 0:
    ax4.plot(peak_times, peak_amps, 'b.', label='Peak Amplitudes')
    ax4.set_title('Amplitude Decay')
    ax4.set_xlabel('Time (s)')
    ax4.set_ylabel('Peak Amplitude')
    ax4.grid(True)
    ax4.legend()

plt.tight_layout()
```

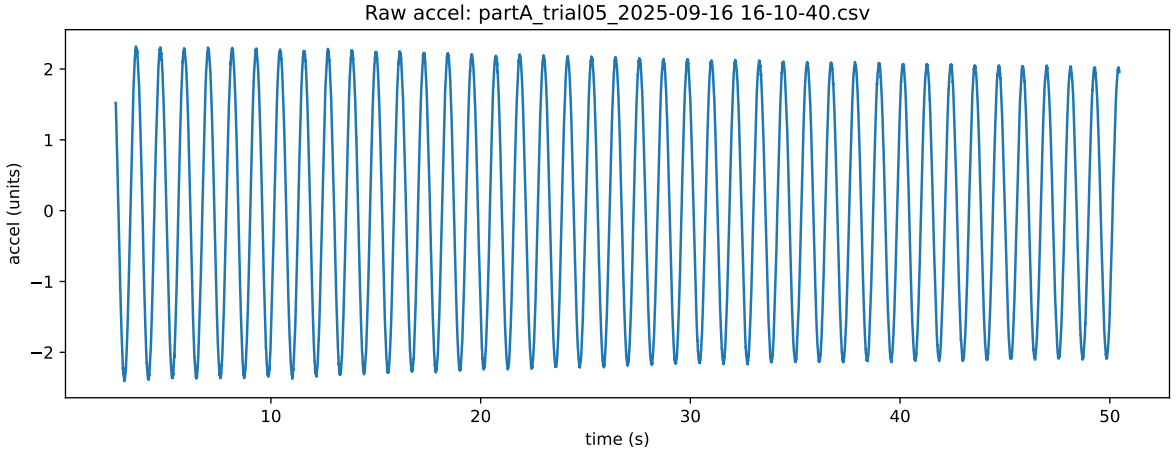
```
plt.show()
```

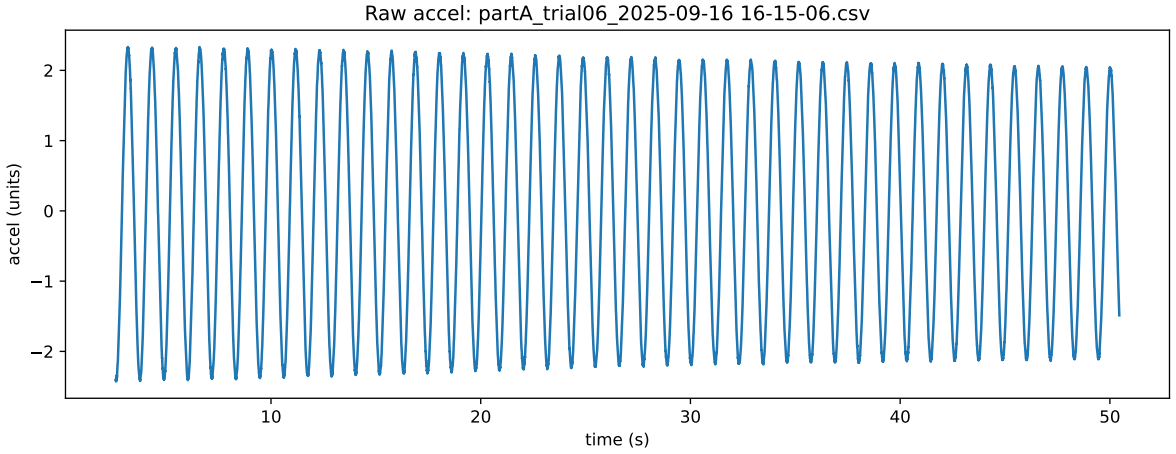

Raw accel: partA_trial01_2025-09-16 15-55-52.csv
accel (units)
time (s)

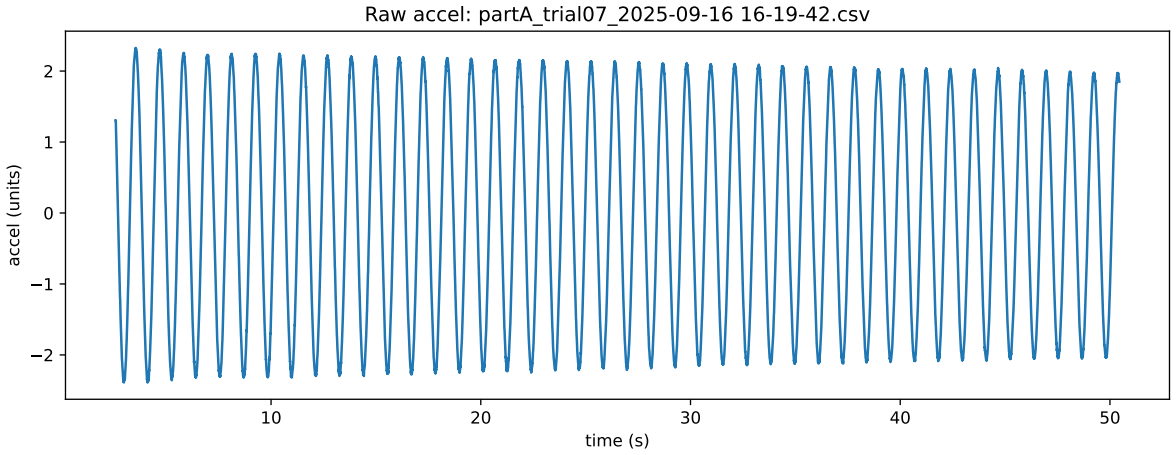

Raw accel: partA_trial02_2025-09-16 15-59-44.csv
accel (units)
time (s)

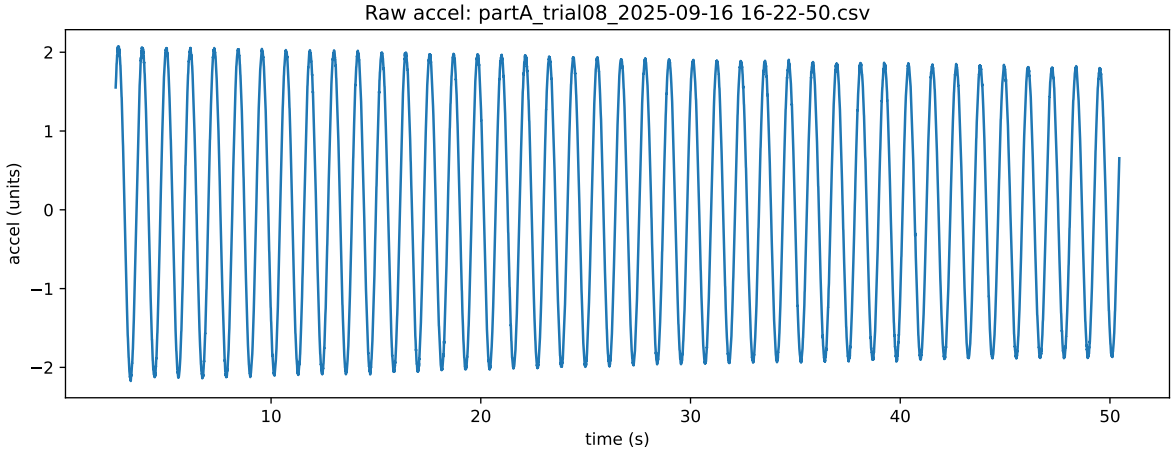

Raw accel: partA_trial03_2025-09-16 16-02-50.csv
accel (units)
time (s)

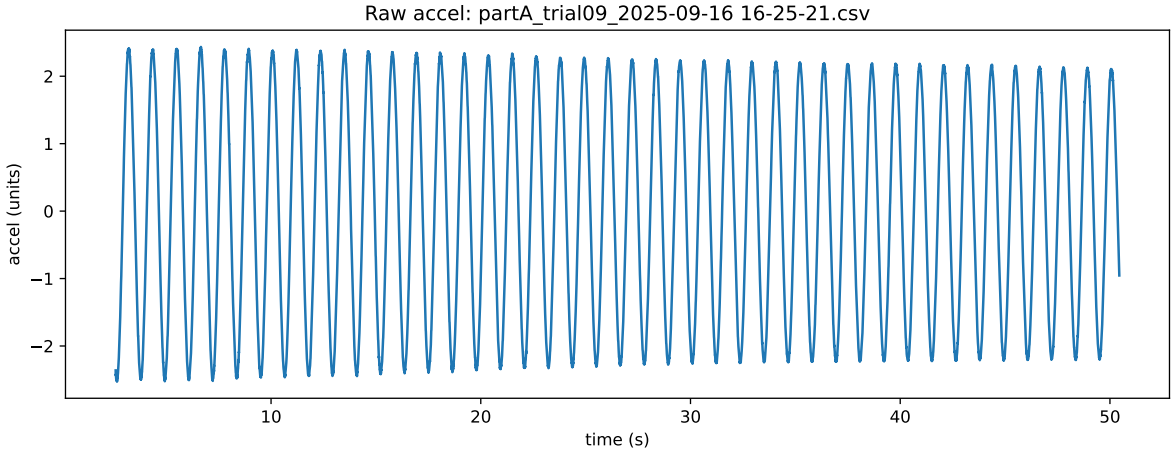

Raw accel: partA_trial04_2025-09-16 16-06-16.csv
accel (units)
time (s)
2
1
0
−1
−2
10
20
30
40
50

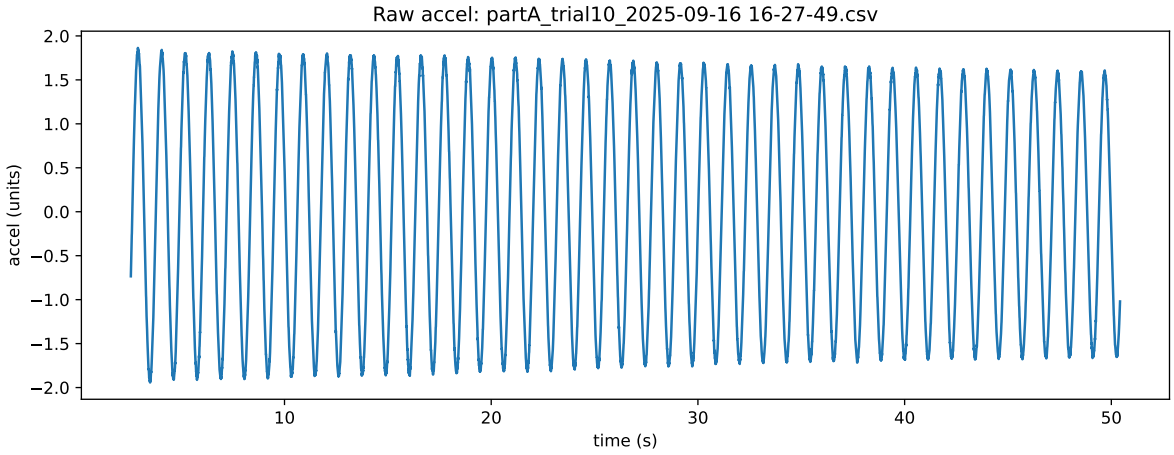

Raw accel: partA_trial05_2025-09-16 16-10-40.csv
accel (units)
time (s)
2
1
0
−1
−2
10
20
30
40
50

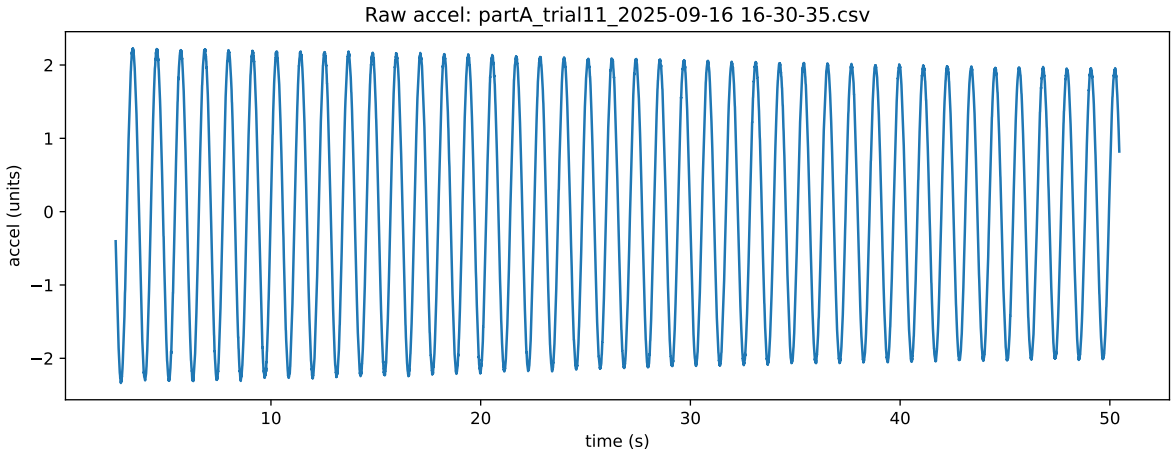

Raw accel: partA_trial06_2025-09-16 16-15-06.csv
accel (units)
time (s)
2
1
0
−1
−2
10
20
30
40
50

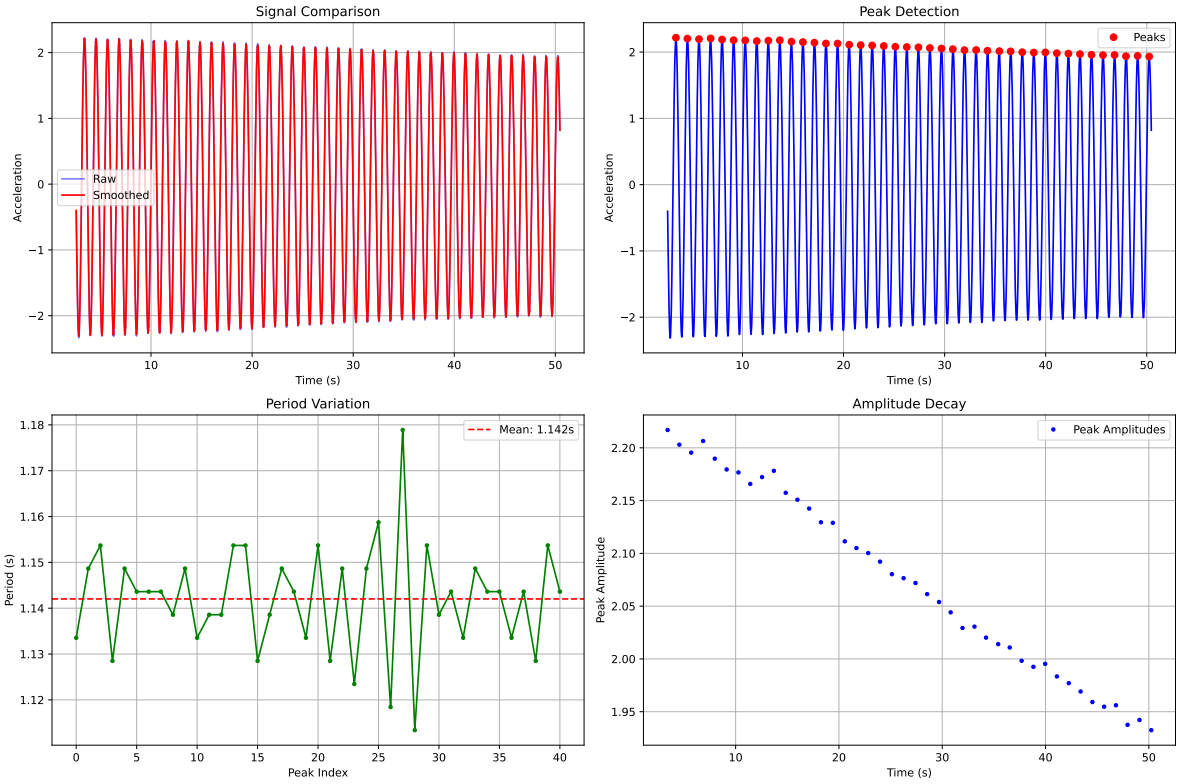

Raw accel: partA_trial07_2025-09-16 16-19-42.csv
accel (units)
time (s)
2
1
0
−1
−2
10
20
30
40
50


Raw accel: partA_trial08_2025-09-16 16-22-50.csv
accel (units)
time (s)
2
1
0
−1
−2
10
20
30
40
50


Raw accel: partA_trial09_2025-09-16 16-25-21.csv
accel (units)
time (s)
2
1
0
−1
−2
10
20
30
40
50


Raw accel: partA_trial10_2025-09-16 16-27-49.csv
accel (units)
time (s)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
10
20
30
40
50


Raw accel: partA_trial11_2025-09-16 16-30-35.csv
accel (units)
time (s)
2
1
0
−1
−2
10
20
30
40
50



3.4 Damping Analysis

3.5 Discussion

4 Part B — Long Pendulum (L 1.0 m)

Repeat analysis from Part A for `partB_trialXX.csv` files. Adjust `L_long` below if your measured length differs.

4.1 Statistical Analysis

4.2 Damping Analysis

4.3 Discussion

4.4 Comparative Study

```
L_long = 1.0
file_pattern = os.path.join(DATA_DIR, 'partB_trial*.csv')
files = sorted(glob.glob(file_pattern))
# Copy-paste the same loop for Part B (or refactor into a function). For brevity we reuse co
results_B = []
for fpath in files:
    df = load_trial_csv(fpath)
    t = df['time'].values
    x = df['accel'].values
    x_smooth = signal.savgol_filter(x, 11, 3)
    periods, peaks = estimate_period_via_peaks(t, x_smooth, prominence=np.std(x_smooth)*0.5)
    if isinstance(periods, np.ndarray):
        mean_period = np.mean(periods)
        std_period = np.std(periods, ddof=1)
    else:
        mean_period = np.nan
        std_period = np.nan
    peak_times = t[peaks] if len(peaks)>0 else np.array([])
    peak_amps = np.abs(x_smooth[peaks]) if len(peaks)>0 else np.array([])
    decay = fit_exponential_decay(peak_times, peak_amps, Tn=mean_period)
    desc = compute_descriptors(x)
    result = {
        'file': os.path.basename(fpath),
        'mean_period': mean_period,
        'std_period': std_period,
        'A0': decay['A0'],
        'k': decay['k'],
        'zeta': decay['zeta'],
        **desc
    }
    results_B.append(result)
pd.DataFrame(results_B)
```

	file	mean_period	std_period	A0	k	zeta	M
0	partB_trial01_2025-09-16 17-05-44.csv	1.180521	0.015039	7.031579	0.006889	0.001294	0.5
1	partB_trial02_2025-09-16 17-07-57.csv	1.181938	0.017244	7.473388	0.010552	0.001985	0.9
2	partB_trial03_2025-09-16 17-10-47.csv	1.184906	0.013286	7.472619	0.006202	0.001170	0.4
3	partB_trial04_2025-09-16 17-13-57.csv	1.185939	0.014329	7.591668	0.006245	0.001179	0.4
4	partB_trial05_2025-09-16 17-16-40.csv	1.182452	0.013770	7.170450	0.007182	0.001352	0.6
5	partB_trial06_2025-09-16 17-19-05.csv	1.183477	0.016683	7.515907	0.010483	0.001974	0.8
6	partB_trial07_2025-09-16 17-21-16.csv	1.181411	0.015671	7.377127	0.010014	0.001883	0.8
7	partB_trial08_2025-09-16 17-24-26.csv	1.181405	0.016501	7.142454	0.007075	0.001330	0.6
8	partB_trial09_2025-09-16 17-26-47.csv	1.185537	0.018647	7.818244	0.011255	0.002124	1.0
9	partB_trial10_2025-09-16 17-29-18.csv	1.183341	0.016820	7.552322	0.010731	0.002021	0.8

5 Part C — Nonlinear Regime (Large Angles)

Use `partC_trialXX.csv`. For large angle trials (0 > 45°), measure period variation vs amplitude and compare with small-angle theory.

5.1 Period Analysis

5.2 Comparison and Discussion

```
file_pattern = os.path.join(DATA_DIR, 'partC_trial*.csv')
files = sorted(glob.glob(file_pattern))
period_vs_amp = []
for fpath in files:
    df = load_trial_csv(fpath)
    t = df['time'].values
    x = df['accel'].values
    x_smooth = signal.savgol_filter(x, 11, 3)
    periods, peaks = estimate_period_via_peaks(t, x_smooth, prominence=np.std(x_smooth)*0.5)
    peak_times = t[peaks] if len(peaks)>0 else np.array([])
    peak_amps = np.abs(x_smooth[peaks]) if len(peaks)>0 else np.array([])
    if len(peak_amps)>0 and isinstance(periods, np.ndarray):
        period_vs_amp.append({'file': os.path.basename(fpath), 'mean_period': np.mean(periods
pd.DataFrame(period_vs_amp)
```

	file	mean_period	mean_amp
0	partC_trial01_2025-09-16 11-45-11.csv	1.981914	1.449503
1	partC_trial02_2025-09-16 11-47-53.csv	1.979063	0.702695
2	partC_trial03_2025-09-16 15-19-13.csv	1.983025	1.576847
3	partC_trial04_2025-09-16 15-22-33.csv	1.981946	1.659214
4	partC_trial05_2025-09-16 15-24-26.csv	1.983063	1.607294
5	partC_trial06_2025-09-16 15-25-01.csv	1.983063	1.607294
6	partC_trial07_2025-09-16 15-26-52.csv	1.982623	1.560053
7	partC_trial08_2025-09-16 15-29-56.csv	1.982623	1.530992
8	partC_trial09_2025-09-16 15-31-56.csv	1.978476	1.386825
9	partC_trial10_2025-09-16 15-34-03.csv	1.983739	1.566655
10	partC_trial11_2025-09-16 15-36-20.csv	1.981989	1.498077
11	partC_trial12_2025-09-16 15-38-39.csv	1.982213	1.657862
12	partC_trial13_2025-09-16 15-41-21.csv	1.982001	1.641470
13	partC_trial14_2025-09-16 15-43-53.csv	1.983316	1.834556

6 Part D — Noise Discussion

Visually inspect signals and write your qualitative assessment here. Use the examples shown in previous plots. Answer: What are possible noise sources? How do they affect period/damping estimation?

6.1 Qualitative Noise Observation

6.1.1 Discussion

7 Submission Checklist

- ☐ Export this notebook to PDF: `HW1_{STUDENT_ID}.pdf` (max 20 A4 pages)
- ☐ Create ZIP named `HW1_{STUDENT_ID}.zip` containing:
  - Executed `.ipynb`
  - `Data/` folder with CSVs
  - `Figures/` folder with generated plots
- ☐ Verify all tables are rendered as DataFrames (no screenshots)
- ☐ Confirm all equations and figures are numbered and captioned
- ☐ Ensure reproducibility: all code cells run from top to bottom without manual path edits (except `STUDENT_ID`)

```
# Helper: save summary tables and figures
summary_csv = os.path.join(BASE_DIR, f'summary_{STUDENT_ID}.csv')
if 'results_df' in globals():
    results_df.to_csv(os.path.join(BASE_DIR, 'summary_partA.csv'), index=False)
if 'results_B' in globals():
    pd.DataFrame(results_B).to_csv(os.path.join(BASE_DIR, 'summary_partB.csv'), index=False)
print('Saved summary CSVs to', BASE_DIR)
```

```
Saved summary CSVs to .
```