Iowa State University
Aerospace Engineering

AER E 322 Lab 01
Practice Experiment and Data Analysis

Matthew Mehrtens, Peter Mikolitis, and Natsuki Oda

February 3, 2023

Aerospace Structures Laboratory Report
Lab 01 Practice Experiment and Data Analysis
Section 4 Group 2
Matthew Mehrtens, Peter Mikolitis, and Natsuki Oda
AER E 322 February 3, 2023 **Spring 2023**

Contents

Chapter 1

Pre-Lab

1.1. Introduction

Aircraft wings undergo oscillations and other random forces while in flight. This lab replicates and analyzes some of the forces and oscillations a wing will experience in flight while also serving as an introduction to the PASCO tool kits and data processing. To simulate the wing, we used a cantilevered aluminum beam, and to generate and measure the oscillations, we used a PASCO tool kit—specifically the PASCO wave driver, displacement sensor and motion sensor. There were three rounds of testing; each additional round of testing introduced a new variable into the beam movement that changed the shape of the data. The data was collected using the PASCO tool kit and software provided. After the lab, we analyzed and processed the data in Python to how each variable effected the oscillation of the beam.

1.2. Objectives

During this lab, our primary objectives were to:

1. Learn how to record data under dynamic conditions and analyze or post-process the data.
2. Observe approximately how a common aerospace structural material might respond to oscillatory forces.

3. Gain familiarity with the PASCO tool kit and the PASCO Capstone software.

1.3. Hypothesis

1.3.1. Test 1

We predict this test will provide the cleanest data of the three tests. Since the only force acting on the beam should be from the wave driver, we expect the displacement graph to shown a uniform and steady wave—matching the oscillations of the wave driver. The data from this test should closely match the oscillations of an airplane wing in very steady flight.

1.3.2. Test 2

This test adds a spring with a hanging weight to the cantilevered beam. Due to the oscillations of the spring-weight system, we expect to see sudden highs and lows in the data corresponding with when the spring is in compression or tension respectively. The data from this test should demonstrate the oscillations of the wing in steady flight if there is an additional oscillatory or vibrational force simultaneously acting on the wing.

1.3.3. Test 3

This test is similar to Test 2 (see Section 1.3.2) except a third significant force has been introduced. Due to the addition of arbitrary impulses being applied by hand to the free end of the beam, we expect the data to show large peaks and dips in the data correlated with the timing of the impulses. The data from this test should demonstrate the oscillations of real flight as described in Section 1.3.2 but also how the wing might react during periods of high turbulence where sudden, large impulses may act on the wing.

Chapter 2

Lab Work

2.1. Variables

2.1.1. Independent Variables

Wave Driver Frequency

The frequency of the wave generated by the PASCO wave driver.

Wave Driver Amplitude

The maximum voltage the wave driver will use when oscillating, proportional to the displacement of the wave driver arm.

Sampling Frequency

The time intervals at which the PASCO motion sensor or displacement sensor poll the position or displacement of the beam. Sensor with a higher sampling frequency will collect more data in the same amount of time.

Smoothing Span

The number of elements after a given element used to calculate a rolling mean while data processing. A small smoothing span will better preserve the shape of the data; whereas, a

larger smoothing span will better reduce noise and outliers.

Time

Each test was run for exactly 15 s, measured in 0.01 s intervals.

2.1.2. Dependent Variables

Displacement/Position

The change in location of the cantilevered beam from its equilibrium position, measured closer to the free end along a perpendicular vertical axis.

Best Fit Curve

The line of best fit or fit curve is a normalized curve matching the raw or smoothed data and depends on the shape and magnitude of the displacement data.

2.2. Work Assignments

Refer to Table 2.1 for the distribution of work during this lab.

2.3. Materials

The materials required for this lab include: cantilever aluminum beam, spring, weight, Styrofoam pad, tape, round point-tip "needle" displacement sensor, C-clamps, wooden 2"x4", bench vice, string, computer with PASCO Capstone software, clamp stand with clamps, PASCO 850 Universal Interface, PASCO mechanical wave driver, and a PASCO non-contact motion sensor.

2.4. Apparatus

Figure 2.1 shows the fully assembled lab apparatus. For Test 1, the weight was removed from the spring. For Tests 2 and 3, the weight was added back to the apparatus and the displacement sensor—the needle-like device on top of the beam—was removed.

Table 2.1: Work assignments for AER E 322 Lab 01.

Task	**Matthew**	**Peter**	**Natsuki**
Lab Work			
Date Recording		X	
Exp. Setup	X		
Exp. Work			X
Exp. Clean-Up	X	X	X
Data Processing			
Data Import	X		
Smoothing	X	X	
Line of Best Fit	X	X	X
Report			
Introduction			X
Objectives		X	
Hypothesis	X		
Variables		X	
Materials		X	
Apparatus		X	
Procedures	X		
Data	X		
Analysis	X	X	X
Conclusion			X
References			X
Appendix			X
Revisions	X	X	X
Editing	X		

As shown in Figure 2.1, the left side of the beam is fixed to the board and the right side is attached to the wave driver with string. The rectangular, gray box in the foreground is the PASCO 850 Universal Interface, and the blue device in the midground is the motion sensor. The motion sensor must be positioned directly underneath the piece of white foam taped to the bottom of the beam with at least 15 cm between the sensor and the foam.

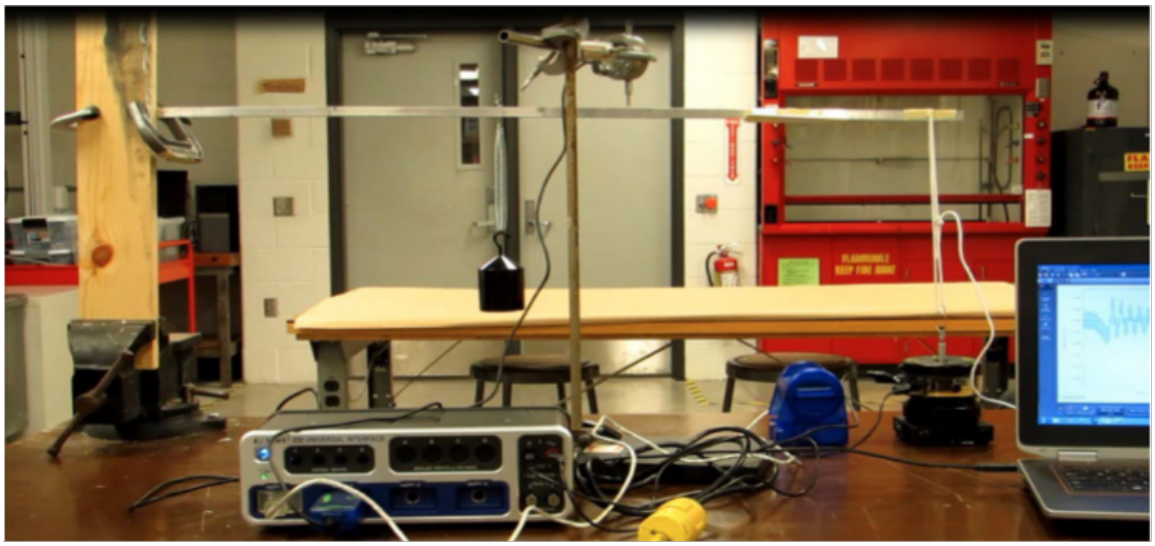

Figure 2.1: Lab apparatus completely set up, including the spring, weight, and displacement sensor. Image from the *Lab 1 Practice Experiment and Data Analysis* manual [1]. Copyright 2023 by Thomas Chiou.

2.5. Procedures

2.5.1. Setup

Assemble the apparatus as described in Section 2.4. Ensure that the styrofoam reflector plate is at least 15 cm away from the motion sensor before running any tests. Launch the PASCO Capstone software, and turn on the PASCO 850 Universal Interface. Check that both the "needle" displacement sensor and motion sensors are recognized by the software. Once the sensors are recognized, configure the software to display two graphs "by selecting the corresponding icon from the on-screen template" [1]. Next, click on each axis to label the graphs, and set the y-label of the displacement sensor graph to "displacement" and "position" for the motion sensor graph. For both graphs, the x-label will be "time". Refer to Figure 2.2 to see how the graph layout should look.

Set up the wave driver by clicking the "Signal Generator" icon on the left-side of the screen. Set the waveform to "Sine", sweep type to "Off", frequency to 10 Hz, amplitude to 10 V, and select "Auto" at the bottom of the window. For reference, compare your "Signal Generator"

Figure 2.2: The graph screen of the PASCO Capstone software fully configured for this lab. Image from the *Lab 1 Practice Experiment and Data Analysis* manual [1]. Copyright 2023 by Thomas Chiou.

window to Figure 2.3.

Next, at the bottom of the screen select "Recording Condition", then select "Stop Condition". Set "Condition Type" to "Time Based", and set the time to 15 s. The "Recording Condition" window will look similar to Figure 2.4.

At the bottom of the screen, select "Motion Sensor", and set the sampling frequency to 100 Hz. The sensor can be zeroed by selecting the icon to the right of the sampling frequency field. Use the same process to set up the displacement sensor, except set the sampling frequency to 5 Hz. To zero the displacement sensor, click the physical O/I button on the sensor. The sensors should be zeroed before each round of testing.

Finally, navigate through the following menus: "Data Summary" icon on the left column of the screen → Motion Sensor → Position (m) → select the gear symbol → Numerical Format → change Number Style to Significant Figures → set the Number of Significant Figures to 5. Follow the exact same process for the displacement sensor.

Figure 2.3: The wave driver or "Signal Generator" configuration window. Image from the *Lab 1 Practice Experiment and Data Analysis* manual [1]. Copyright 2023 by Thomas Chiou.

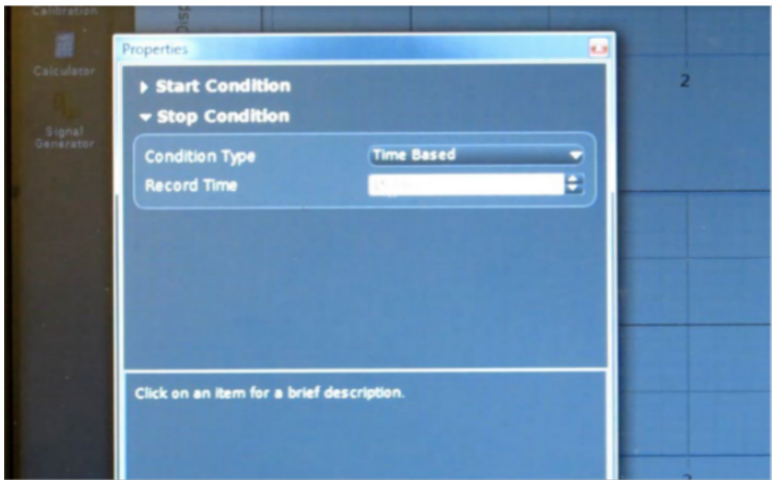

Figure 2.4: The "Stop Conditions" configuration window. Image from the *Lab 1 Practice Experiment and Data Analysis* manual [1]. Copyright 2023 by Thomas Chiou.

2.5.2. Test 1

For Test 1, we will only utilize the wave driver. Take the hanging weight off the beam before starting and remain clear of the apparatus during operation. Ensure the displacement sensor is making good contact with the beam and approximately halfway depressed. Check the motion sensor is directly under the styrofoam reflector plate with at least 15 cm of buffer. The string that attaches the beam to the axle socket of the wave driver should be taut. Lastly, double-check that the connector tip of the wave driver "is firmly seated inside the axle socket" [1]. Zero both sensors before running the test, then, in the PASCO Capstone software, click "Record" to run the test. Let the test run for the entire 15 s duration.

2.5.3. Test 2

Before beginning Test 2, remove the displacement sensor from the beam and add the hanging weight to the spring. Begin the test as described in Section 2.5.2. As the test begins, a group member will pull the weight down and release it, allowing the spring-weight system to oscillate for the duration of the test. Be careful not to pull the weight down too hard.

2.5.4. Test 3

For Test 3, leave the apparatus as it was configured for Test 2 in Section 2.5.3. Start the test as described in Section 2.5.2. After beginning the test, in addition to pulling down and releasing the hanging weight, a group member will randomly tap the free end of the cantilevered beam up or down every 2 s to 3 sfor the duration of the test.

2.5.5. Cleanup

Using the "File" menu in the PASCO Capstone software, save the experiment data to a `.txt` or `.csv` file. Transfer the data to a flash drive for post-processing and analysis.

2.6. Data

All data analysis was done with Python. The following packages were used for importing, smoothing, and processing the data:

- `pandas` (https://pandas.pydata.org/)

- `numpy` (https://numpy.org/)
- `matplotlib` (https://matplotlib.org/)
- `scipy` (https://scipy.org/)

The data was in a .csv format which the `pandas` Python package had no difficulty importing and parsing. To smooth the data, we used the `pandas.DataFrame.rolling()` and `pandas.DataFrame.mean()` functions. The `.rolling()` function selects a subset of the elements, and the `.mean()` function calculates the mean of that subset. See Listing 2.1 for the function from our Python script that smoothed the data.

```
# Smooths a DataFrame
# Input Args:
#   df: 1 col DataFrame
#   smooth: list of smoothing windows, e.g., [5, 3, 3]
def smooth_df(df, smooth):
    smoothed_data = df
    for win in smooth:
        # min_periods=1 stops the NaNs
        smoothed_data = smoothed_data.rolling(win, min_periods=1).mean()

    return smoothed_data
```

Listing 2.1: The `smooth_df()` function from our Python data processing script. This function smooths a `pandas.DataFrame` object once or in multiple passes as defined by the `smooth` argument.

To generate the line of best fit, we first observed that the data in our graphs looked generally sinusoidal. To generate and graph a sinusoidal line of best fit, our best option was to use the `scipy.optimize()` function, but since the frequency of the beams displacement curve varied due to the forces we were applying to it—especially in Test 2 and Test 3—the `scipy.optimize()` function struggled to accurately generate an appropriate line of best fit. Our final solution for generating the line of best fit involved using the `scipy.optimize()` function and the `numpy.fft.fftfreq()` and `numpy.fft.fft()` functions which performs a Fast Fourier Transform (FFT) on a `numpy.array` object. See Listing 2.2 for our implementation of the sinusoidal line of best fit function.

```
# Uses a Fast Fourier Transform to generate an initial guess then uses a
    non-linear least squares algorithm to
# generate a matching sine curve
```

```python
def sin_lobf(data, t, stats=False):
    # Fit sin to the input time sequence, and return fitting parameters "amp", "omega", "phase", "offset", "freq", "period" and "fitfunc"
    t = np.array(t)
    data = np.array(data)
    ff = np.fft.fftfreq(len(t), (t[1]-t[0]))   # assume uniform spacing
    Fyy = abs(np.fft.fft(data))
    guess_freq = abs(ff[np.argmax(Fyy[1:])+1])   # excluding the zero frequency "peak", which is related to offset
    guess_amp = np.std(data) * 2.**0.5
    guess_offset = np.mean(data)
    guess = np.array([guess_amp, 2.*np.pi*guess_freq, 0., guess_offset])

    def sinfunc(t, A, w, p, c):  return A * np.sin(w*t + p) + c
    popt, pcov = curve_fit(sinfunc, t, data, p0=guess) # this is the least squares bit
    A, w, p, c = popt
    f = w/(2.*np.pi)
    fitfunc = lambda t: A * np.sin(w*t + p) + c
    dict = {"amp": A, "omega": w, "phase": p, "offset": c, "freq": f, "period": 1./f, "fitfunc": fitfunc, "maxcov": np.max(pcov), "rawres": (guess,popt,pcov)}

    # Print some stats
    if stats:
        print("Amplitude: %g mm\nFrequency: %g Hz\nLOBF: f(t) = %gsin(%g * t + %g) + %g\n----------------------------------------------------------" % (dict['amp'], dict['freq'], dict['amp'], dict['omega'], dict['phase'], dict['offset']))
    return dict
```

Listing 2.2: The `sin_lobf()` function from our Python data processing script. This function generates a sinusoidal line of best fit for data that is generally sinusoidal. It uses an FFT algorithm from the `scipy` package and an optimization function from the `numpy` package. The function was modified from unsym's post on StackOverflow [2].

We also tested using a piece-wise line of best fit generator, *i.e.*, the line of best was generated for smaller consecutive domains rather than the entire 0 s to 15 s of data. The graphs generated using this method can be found in Appendix A.2. The entirety of the data processing script is shown in Appendix B.1.

The following sections discuss the data from Tests 1–3. For conciseness, only the most pertinent graphs are discussed below. All the graphs are included in Appendix A for reference.

2.6.1. Test 1

Shown in Figures 2.5 and 2.6 is the raw data and smoothed data respectively for Test 1. The raw data shows the measurements from the displacement sensor and the motion sensor. The smoothed data omits the raw data from the displacement sensor.

The smoothed data was generated using a two-pass rolling mean algorithm. The first pass used a window of 5 and the second pass used a window of 3. For clarity, only the data from 5 s to 8 s is shown in Figures 2.5 and 2.6.

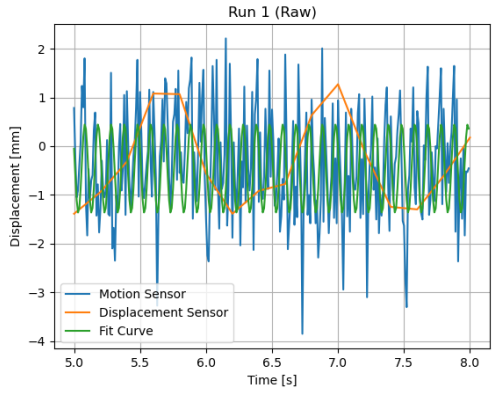

Figure 2.5: Raw data from Test 1, zoomed to only show 5 s to 8 s.

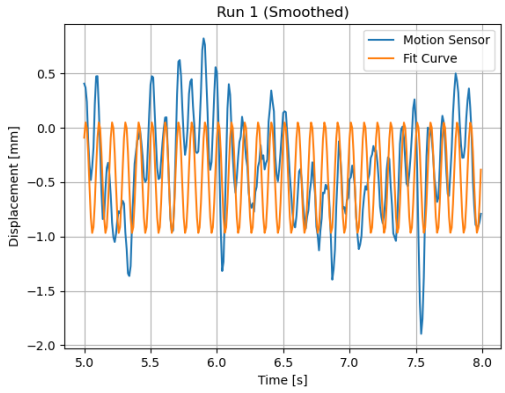

Figure 2.6: Smoothed data from Test 1, zoomed to only show 5 s to 8 s.

From these graphs, it is clear the displacement sensor's polling rate was too slow. In retrospect, the reason is clear. As shown from Figure 2.6 and as described in Section 2.5.1, the wave driver was forcing the beam to oscillate at a rate of approximately 10 Hz, but the polling rate of the displacement sensor was 5 Hz. In other words, the resolution of the displacement sensor was too low to accurately measure the true displacement of the beam. For this reason, the displacement sensor was omitted from the second and third tests.

The equations of the lines of best for Figures 2.5 and 2.6 are given in Equations 2.1 and 2.2 respectively.

$$f(t) = -0.9117 \sin(62.82t - 0.4239) - 0.4580 \tag{2.1}$$

$$f(t) = 0.5134 \sin(62.82t + 0.8422) - 0.4591 \tag{2.2}$$

2.6.2. Test 2

Shown in Figures 2.7 and 2.8 is the raw data and smoothed data respectively for Test 2. At the beginning of this run, a force was applied to a weight hanging from a spring connected to the beam, causing the weight to oscillate (see Section 2.4 and 2.5.1 for details).

The smoothed data was generated using a three-pass rolling mean algorithm. The first pass used a window of 5, the second pass used a window of 3, and the third pass used a window of 3. For clarity, only the data from 5 s to 8 s is shown in Figures 2.7 and 2.8.

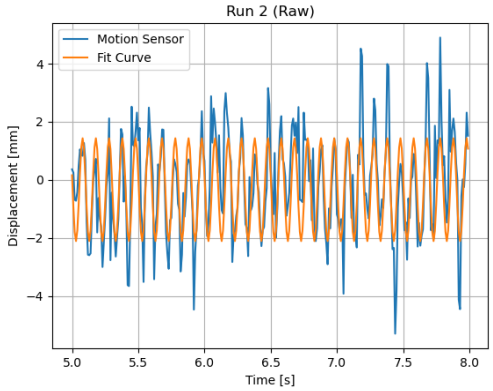

Figure 2.7: Raw data from Test 2, zoomed to only show 5 s to 8 s.

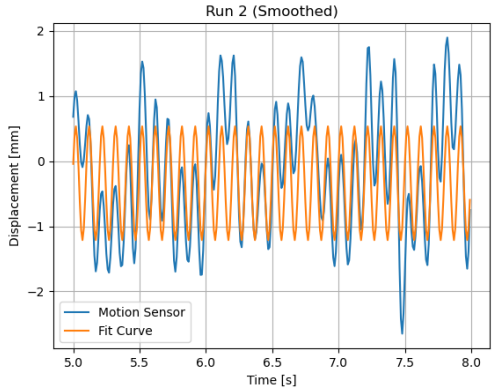

Figure 2.8: Smoothed data from Test 2, zoomed to only show 5 s to 8 s.

The line of best fit does not account for the secondary oscillations due to the hanging weight—it averages between the high and low secondary oscillations–but from looking at the motion sensor measurements shown in Figure 2.8, you can observe the periods of oscillation where the weight was applying a downwards force on the beam, *e.g.*, from 5.2 s to 5.5 s.

The equations of the lines of best for Figures 2.7 and 2.8 are given in Equations 2.3 and 2.4 respectively.

$$f(t) = -1.771 \sin(62.83t - 0.2651) - 0.3391 \tag{2.3}$$

$$f(t) = 0.8731 \sin(62.83t + 0.3724) - 0.3387 \tag{2.4}$$

2.6.3. Test 3

Shown in Figures 2.9 and 2.10 is the raw data and smoothed data respectively for Test 3. At the beginning of this run, a force was applied to a weight hanging from a spring connected to the beam causing the weight to oscillate. At random times during the run, a member from our group applied an arbitrary impulse with their hand to the end of the beam (see Section 2.5.4 for details).

The smoothed data was generated using a two-pass rolling mean algorithm. The first pass used a window of 5 and the second pass used a window of 3.For clarity, only the data from 5 s to 8 s is shown in Figures 2.9 and 2.10.

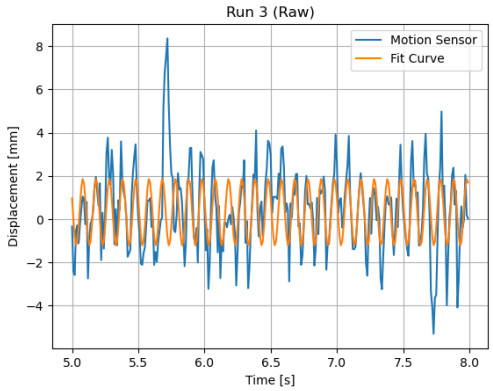

Figure 2.9: Raw data from Test 3, zoomed to only show 5 s to 8 s.

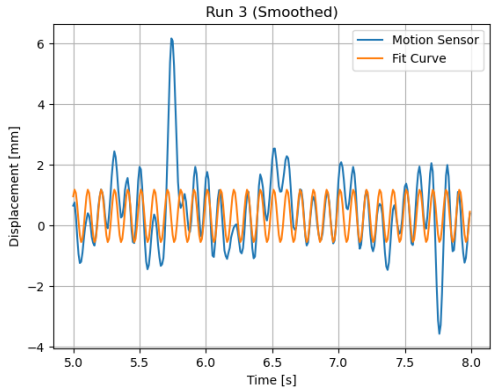

Figure 2.10: Smoothed data from Test 3, zoomed to only show 5 s to 8 s.

We observe from Figures 2.9 and 2.10 the same secondary oscillations we observed in Test 2 (Section 2.6.2. In addition to the secondary oscillations, we also note the large spikes at 5.75 s and 7.75 s. These spikes were due to the arbitrary impulses being applied to the beam by one of our group members.

The equations of the lines of best for Figures 2.9 and 2.10 are given in Equations 2.5 and 2.6 respectively.

$$f(t) = -1.547 \sin(62.82t - 0.3488) + 0.3259 \tag{2.5}$$

$$f(t) = 0.8700 \sin(62.82t + 0.9127) + 0.3228 \tag{2.6}$$

Chapter 3

Conclusion

3.1. Analysis

3.1.1. Test 1 and the Displacement Sensor

As described in Section 2.5.2, Test 1 was the only test to include the data from the displacement sensor. Figure 2.5 in Section 2.6.1 clearly shows why the displacement sensor was inadequate for our purposes in this lab. To reiterate the excerpt from Section 2.6.1:

> From these graphs, it is clear the displacement sensor's polling rate was too slow. In retrospect, the reason is clear. As shown from Figure 2.6 and as described in Section 2.5.1, the wave driver was forcing the beam to oscillate at a rate of approximately 10 Hz, but the polling rate of the displacement sensor was 5 Hz. In other words, the resolution of the displacement sensor was too low to accurately measure the true displacement of the beam. For this reason, the displacement sensor was omitted from the second and third tests.

The motion sensor, however, is a much more precise measurement tool. From the smoothed data in Figure 2.6, it is possible to count the number of cycles that occur in one second, and if done correctly, you will find there almost exactly 10 cycles each second. The line of best fit—which may not excel at matching the sinusoidal center at any given time—from the raw data in 2.5 correctly estimates the frequency of the beam's displacement to be 9.999 Hz (see Appendix B.2 for full script output). This quantitative result matches the qualitative

counting result, confirming that even despite the noise and variation in sinusoidal center, the beam was indeed oscillating at 10 Hz as expected, and the motion sensor was correctly recording the oscillations.

3.1.2. Data Processing

A detailed discussion of how we smoothed the data and generated the line of best fit can be found in Section 2.6. We used multi-pass rolling mean algorithms to smooth the data with assistance from the `pandas` Python library. When attempting to generate the line of best fit, we had troubles using the `scipy.optimize()` function. It seemed that the subtle variations in frequency due to the external forces being applied or even just stray vibrations in the apparatus were enough for the `scipy.optimize()` function to fail.

The solution—which is described in more detail in Section 2.6—was to use a Fast Fourier Transformation algorithm to predict the frequency of the data, and then to use the `scipy.optimize()` function to optimize the sine curve parameters. This method turned out to be quite successful, and despite the variations in the sinusoidal center, the line of best fit matches the peaks and troughs of the actual data well.

In Tests 2 and 3, the spring-weight mechanism (see Sections 2.4, 2.5.3, and 2.5.4) caused a secondary sinusoidal oscillation, proportional to the oscillation of the spring-weight system. This secondary oscillation caused the data to deviate from the line of best by changing the sinusoidal center of the data. If we had more time, we would have liked to modify the line of best function to include a second sine term in the optimized function.

Additionally, in Test 3, the random impulses applied by one of our group members showed up clearly in the data as discussed in Section 2.6.3. They significantly deviate from the line of best fit and despite the smoothing, it is still evident they were the result of an external impulse and not just erroneous measurements.

In Tests 1 and 2, the line of best fit for the raw data estimated the sinusoidal center to be less than zero. This was confusing initially, but upon further inspection we realized this was a reasonable result. In a vacuum with no gravitational influence, if a downward force were applied to the beam, and the beam reacted perfectly elastically to the deformation, the beam would oscillate exactly around the point of zero displacement. This experiment did not take place in a zero-gravity vacuum; so, we can assume the reason the sinusoidal center was below zero was due to gravity, air resistance, and energy loss due to vibrations in the apparatus.

Lastly, in Test 1, the graph of the full raw data shows large variations in the amplitude

and sinusoidal center of the displacement curve. We were unable to come up with a strong explanation; however, we hypothesized that the reason could have been due to the small downward force being applied by the displacement sensor "needle". The force of this needle was certainly slight, but the displacements being recorded were generally 5 mm or less, *i.e.*, even a small force is not negligible at this scale.

3.1.3. Sources of Error

In our experiments, we identified the following potential sources of error:

- The slight downward force of the displacement sensor "needle" could have absorbed some of the elastic energy generated by the wave driver displacing the beam. This would push the sinusoidal center of the displacement data lower than it should be.
- The PASCO tool kit sensors were zeroed before each test; however, its possible the instruments were bumped slightly while starting the test or jostled by the forces and vibrations of the wave driver. Additionally, after many uses, the measurement devices could be miscalibrated.
- The styrofoam reflector piece that was taped to the bottom of the beam was slightly loose, and if the reflector plate was rotating slightly or not staying flush to the beam, the result would be small errors in the displacement of the beam.
- The tautness of the string was checked qualitatively, and if the string were over or under-tightened, this would affect the amount of elastic energy being imparted on the beam.

In a real airplane, there are myriad forces battering the airplane at any point in time. The regular turbulence of the air in higher altitudes, strong gusts of wind or storms, or pockets of variable air pressure could all explain the spikes shown in Test 3. The secondary oscillation we observed in Test 2 and 3 could also be explained by a combination of vibrations: aerodynamically related vibrations coupled with strong internal vibrations such as that of the jet engines or motors. Even loose control surfaces or objects obstructing airflow over the skin of the plane could contribute to the type of oscillations we observed in our representative tests.

3.2. Conclusion

After analyzing all the data, our hypotheses in Section 1.3 generally held true, the most notable exception being our hypothesis of Test 1 (Section 1.3.1. We expected the data to look much cleaner, but even in the simplest apparatus configuration, there was considerable noise and variation in the recorded data.

This was an important lesson to learn: when measuring at such a precise scale, it is imperative to practice good data processing techniques. We also learned it is important to choose the correct measuring device. The displacement sensor was totally incapable of properly recording the displacement of the beam, but if we had not also used the motion sensor to record the beam deflection, it may have been hard to identify that the displacement sensor was the source of the "biased" error.

References

[1] Thomas Chiou. *Lab 1 Practice Experiment and Data Analysis*. Jan. 26, 2023, pp. 2–5.
[2] unsym. *How do I fit a sine curve to my data with pylab and numpy?* Feb. 19, 2017. URL: https://stackoverflow.com/a/42322656 (visited on 02/03/2023).

Appendix A

Graphs

A.1. Regular LOBF Graphs

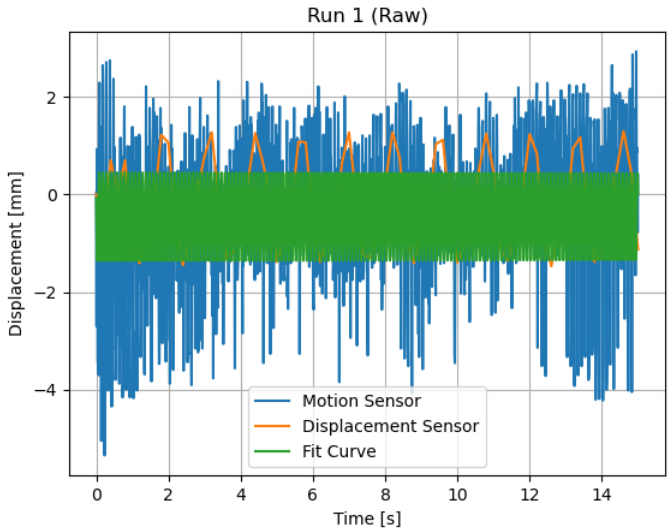

Figure A.1: Raw data from Test 1.

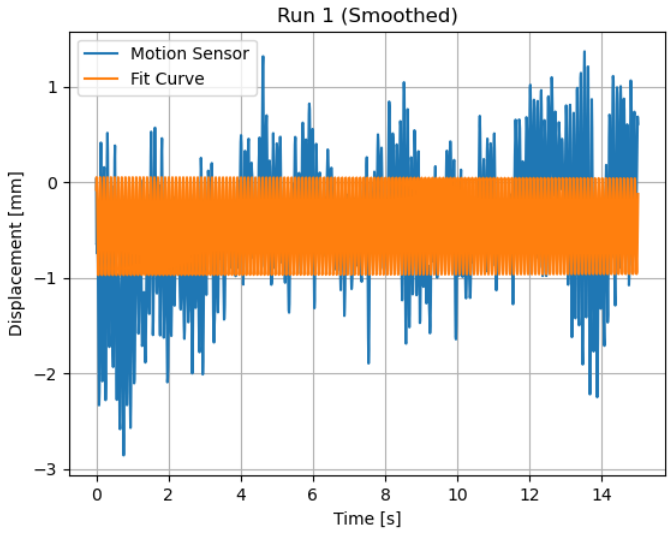

Figure A.2: Smoothed data from Test 1.

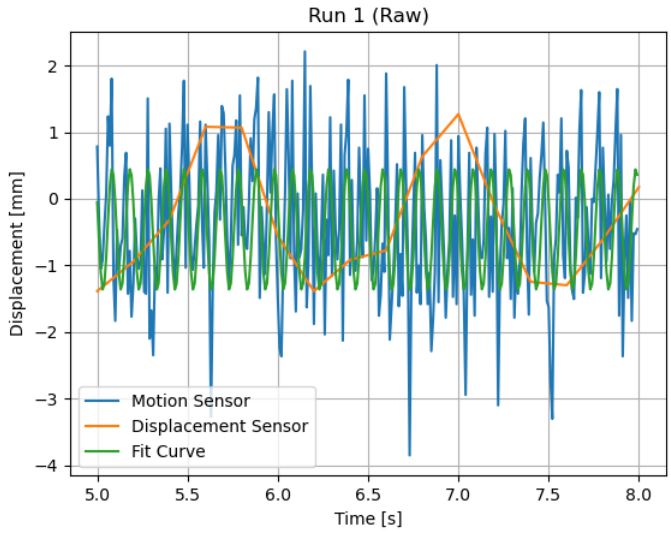

Figure A.3: Raw data from Test 1, zoomed to only show 5 s to 8 s.

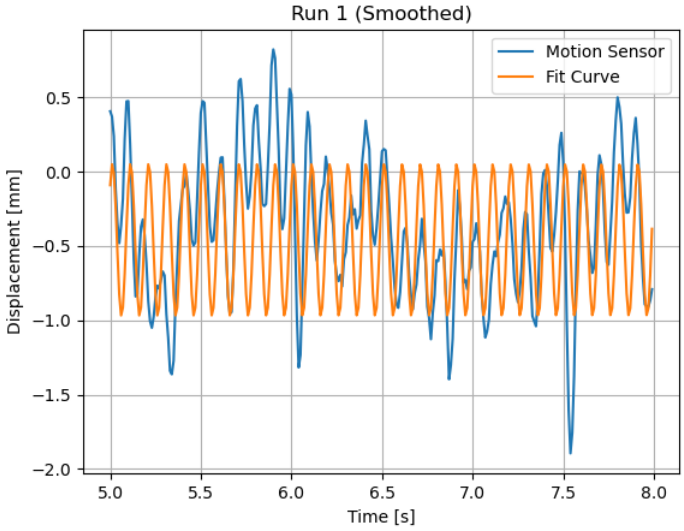

Figure A.4: Smoothed data from Test 1, zoomed to only show 5 s to 8 s.

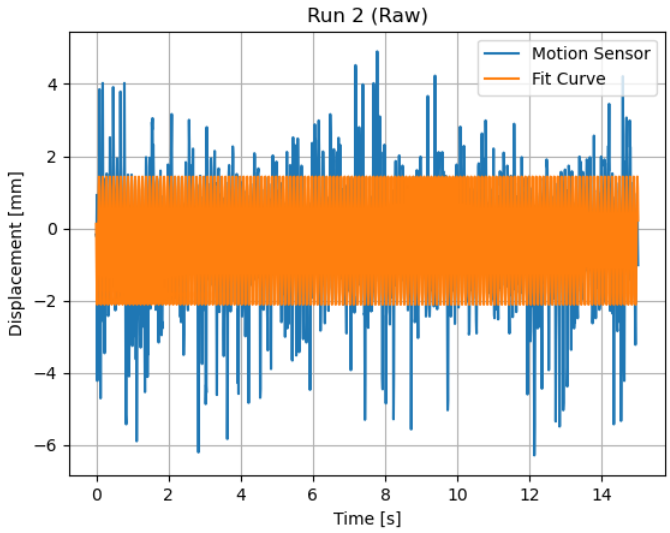

Figure A.5: Raw data from Test 2.

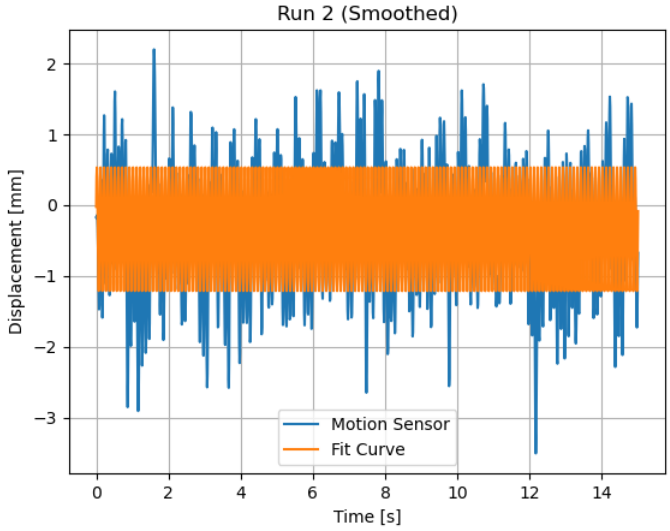

Figure A.6: Smoothed data from Test 2.

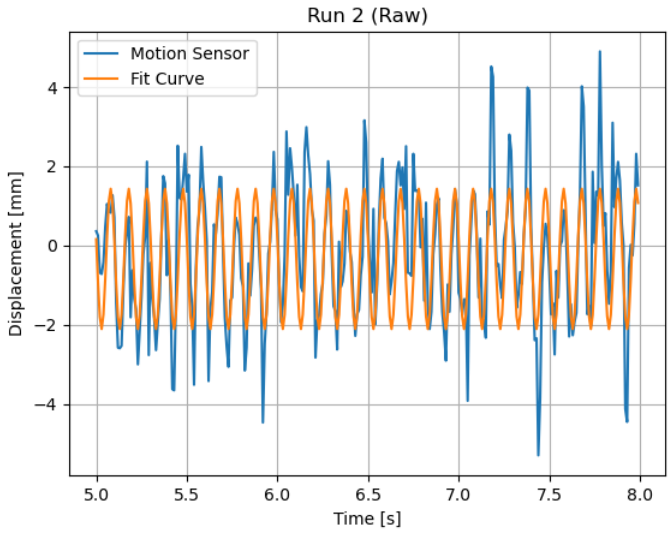

Figure A.7: Raw data from Test 2, zoomed to only show 5 s to 8 s.

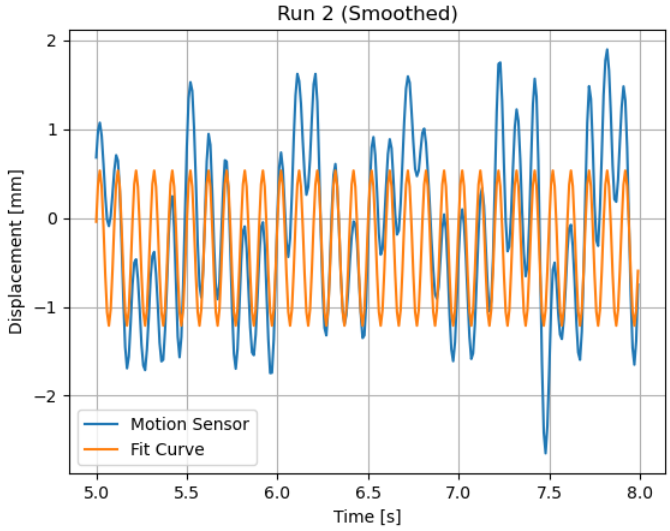

Figure A.8: Smoothed data from Test 2, zoomed to only show 5 s to 8 s.

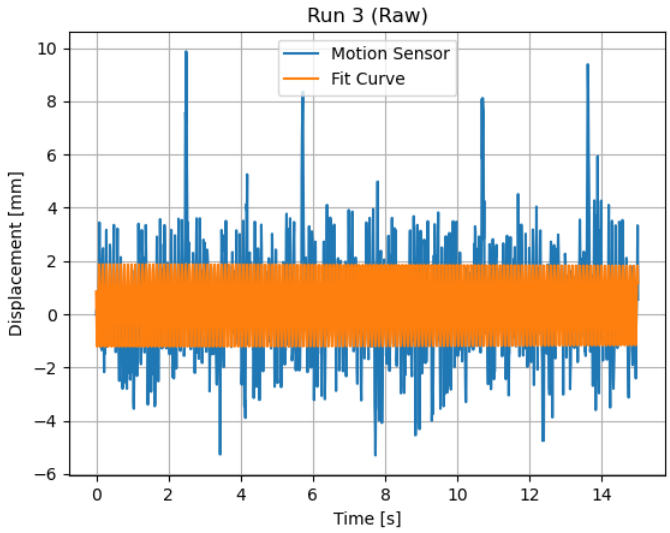

Figure A.9: Raw data from Test 3.

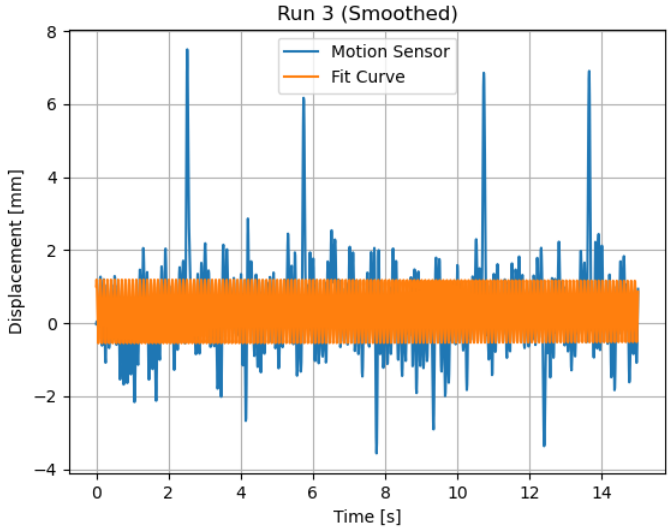

Figure A.10: Smoothed data from Test 3.

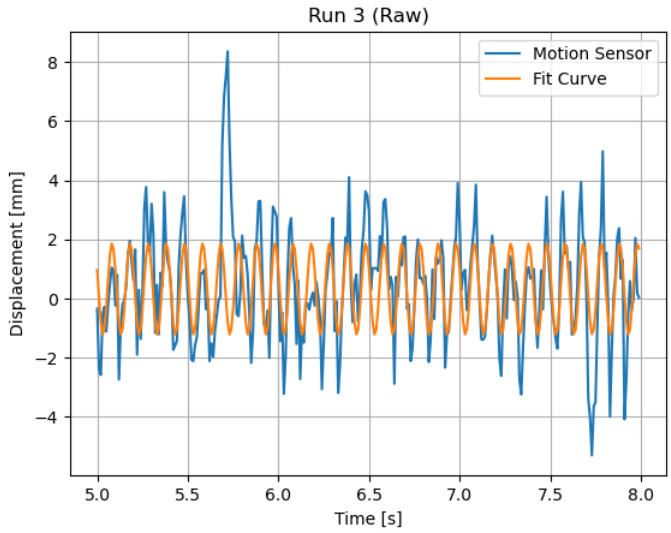

Figure A.11: Raw data from Test 3, zoomed to only show 5 s to 8 s.

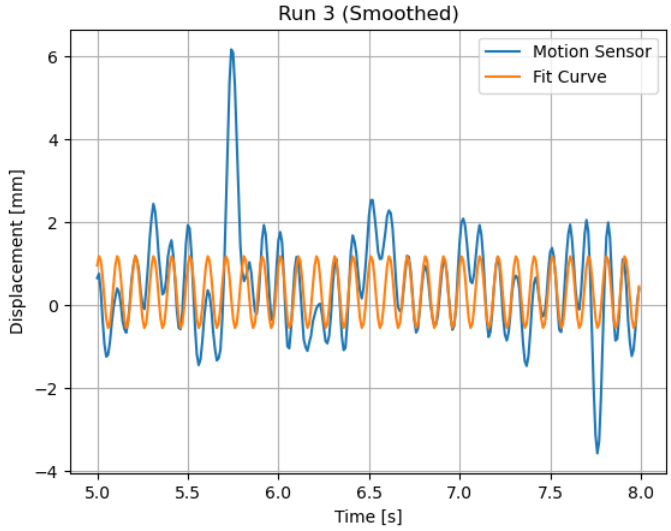

Figure A.12: Smoothed data from Test 3, zoomed to only show 5 s to 8 s.

A.2. Piece-Wise LOBF Graphs

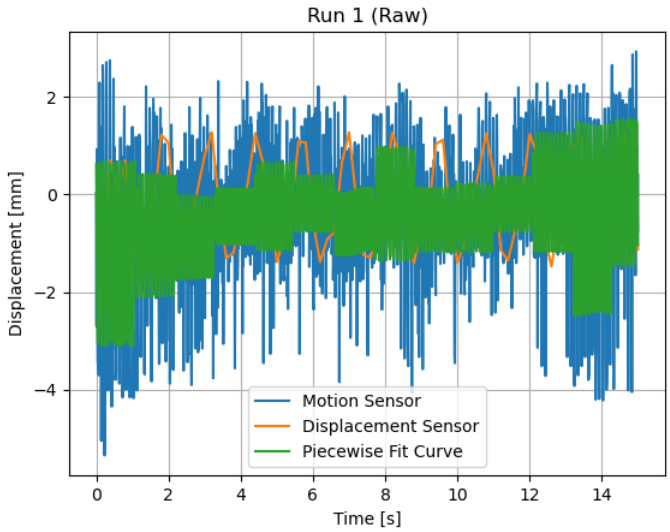

Figure A.13: Raw data from Test 1. Uses the piece-wise line of best fit.

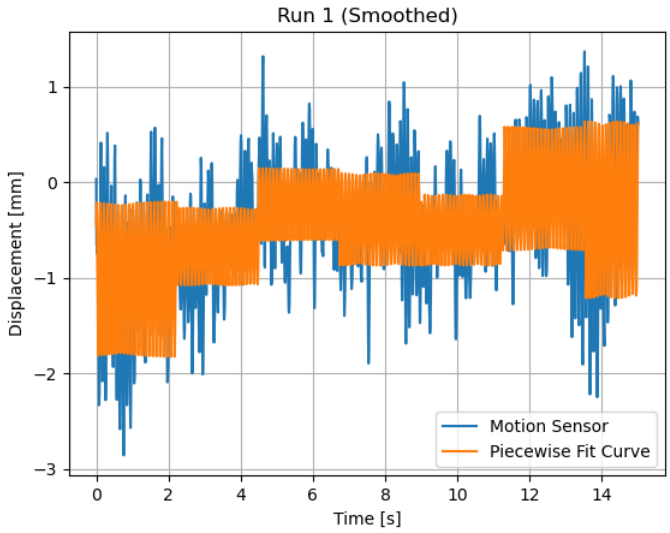

Figure A.14: Smoothed data from Test 1. Uses the piece-wise line of best fit.

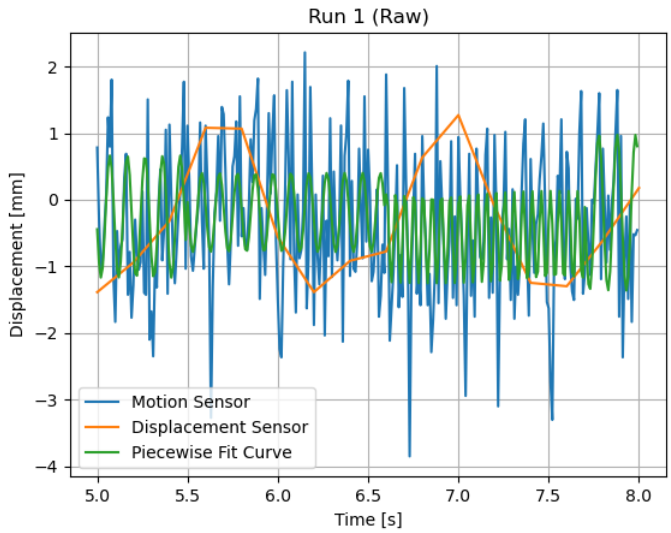

Figure A.15: Raw data from Test 1, zoomed to only show 5 s to 8 s. Uses the piece-wise line of best fit.

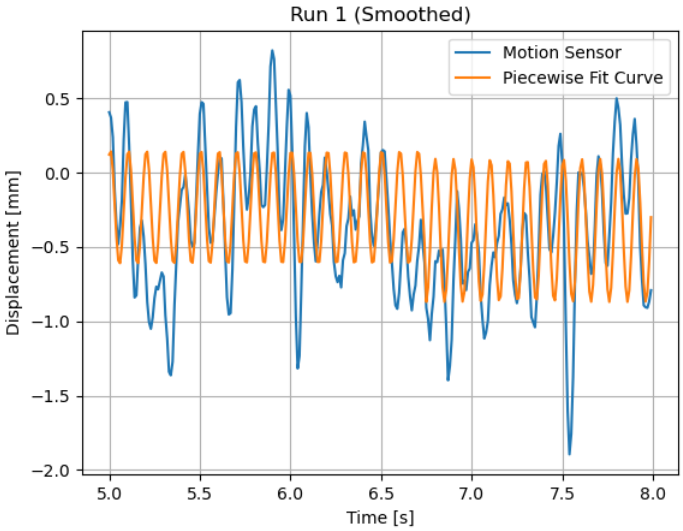

Figure A.16: Smoothed data from Test 1, zoomed to only show 5 s to 8 s. Uses the piece-wise line of best fit.

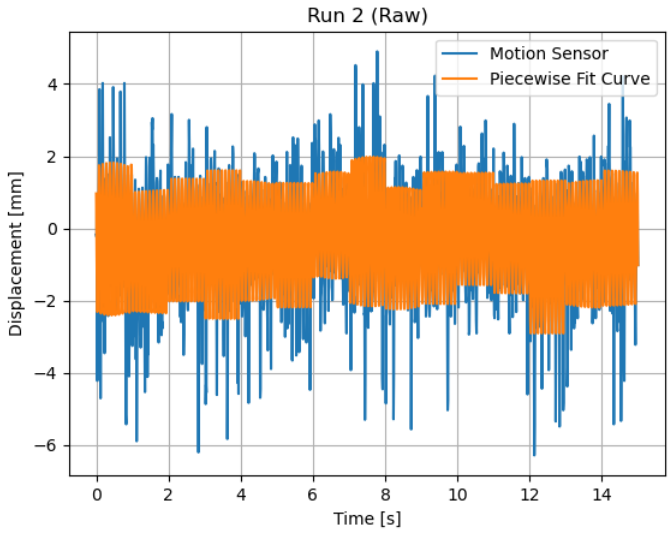

Figure A.17: Raw data from Test 2. Uses the piece-wise line of best fit.

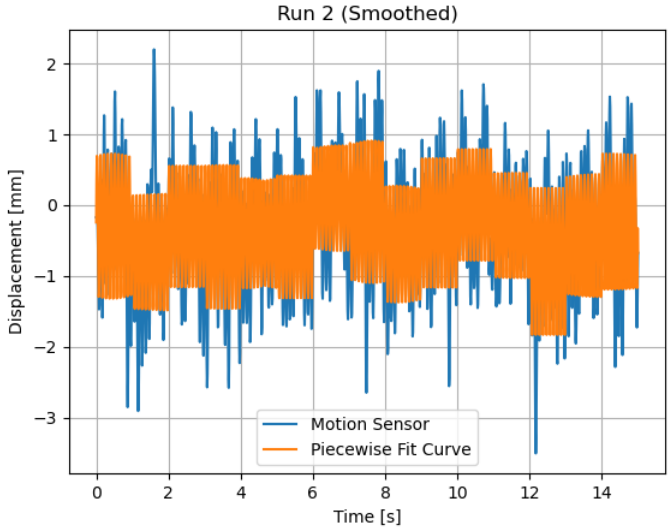

Figure A.18: Smoothed data from Test 2. Uses the piece-wise line of best fit.

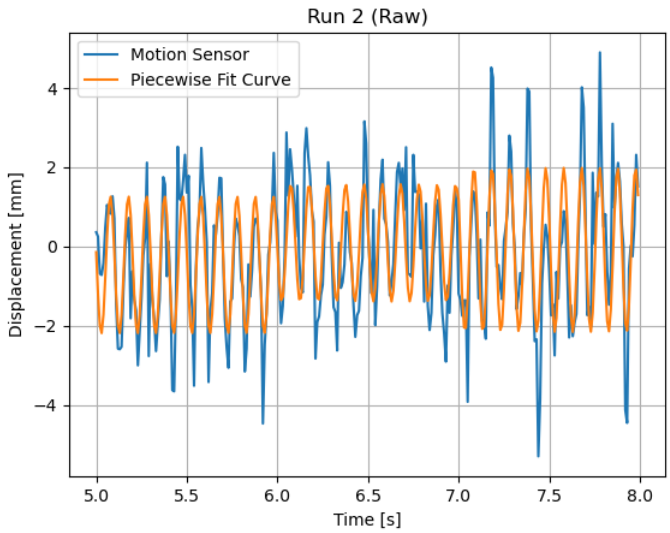

Figure A.19: Raw data from Test 2, zoomed to only show 5 s to 8 s. Uses the piece-wise line of best fit.

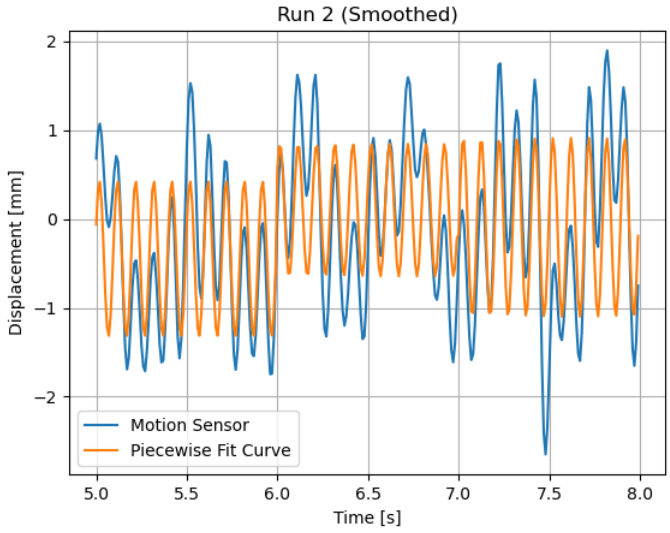

Figure A.20: Smoothed data from Test 2, zoomed to only show 5 s to 8 s. Uses the piece-wise line of best fit.

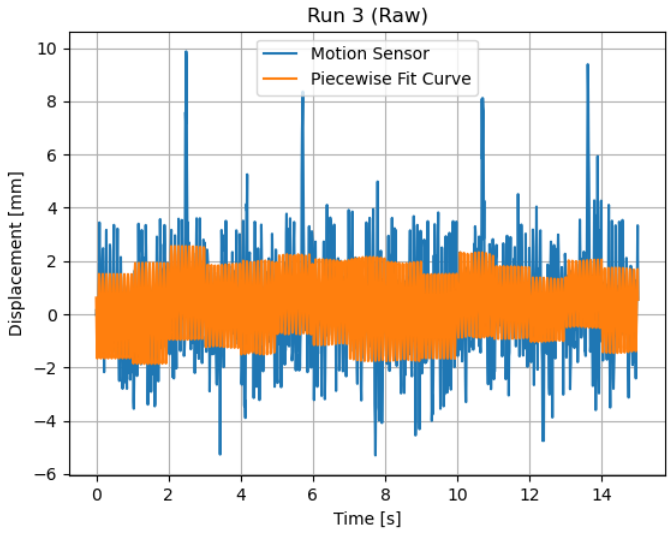

Figure A.21: Raw data from Test 3. Uses the piece-wise line of best fit.

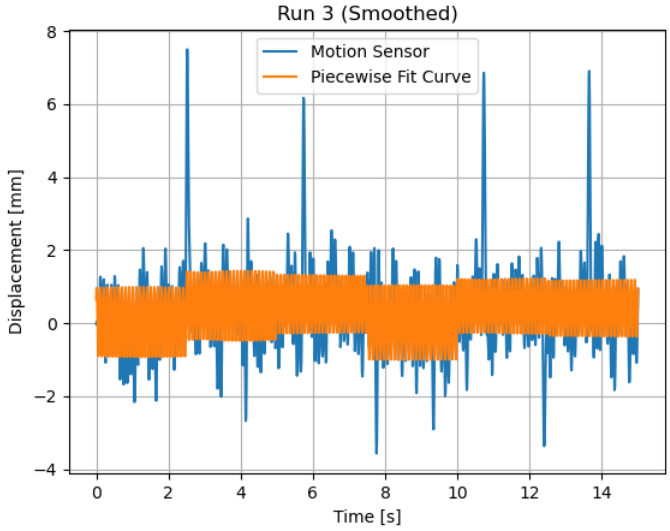

Figure A.22: Smoothed data from Test 3. Uses the piece-wise line of best fit.

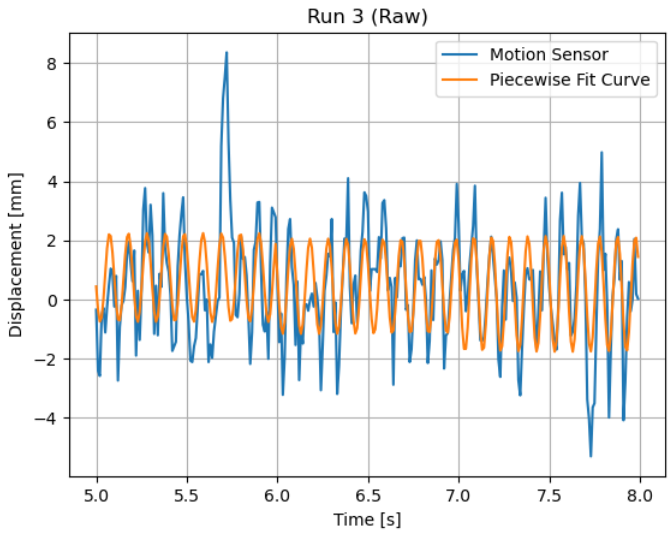

Figure A.23: Raw data from Test 3, zoomed to only show 5 s to 8 s. Uses the piece-wise line of best fit.

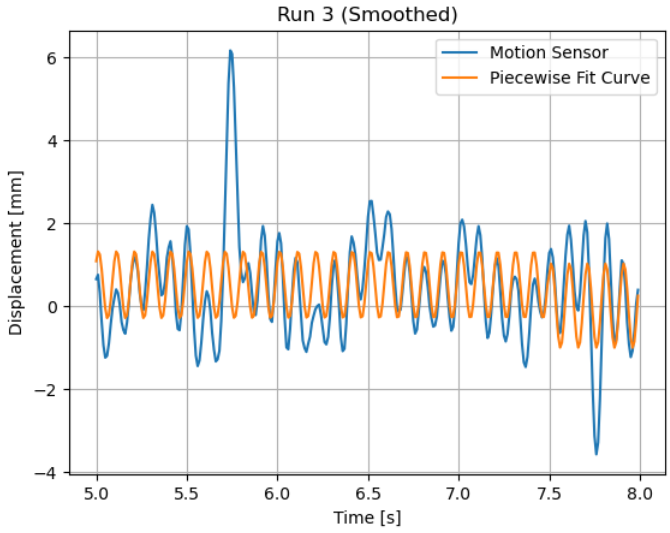

Figure A.24: Smoothed data from Test 3, zoomed to only show 5 s to 8 s. Uses the piece-wise line of best fit.

Appendix B

Code

B.1. Lab-01-Analysis.py

```
"""
AER E 322 Lab 01
Spring 2023
Matthew Mehrtens, Peter Mikolitis, and Natsuki Oda

This Python script imports our experimental data from the .csv file and
    then runs a number of post-
processing algorithms on the data.
"""
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit


DATA_FILE = "Lab 01 Data.csv"
USE_PIECEWISE_LOBF = False


# Starting point of the script
def main():
    # Import the data
    data = import_data(DATA_FILE)
```

```
23
24      # Parse the data into variables
25      time = data["Time (s)"] # DataFrame
26
27      # Parse the displacement sensor data
28      displacement_time = time[0::20]
29      displacement = data["Displacement (mm)"][0::20]
30
31      # Parse the motion sensor data from three runs
32      position = {
33          "run1": data["Position (m)"] * 1000,
34          "run2": data["Position (m).1"] * 1000,
35          "run3": data["Position (m).2"] * 1000
36      }
37
38      # Smooth the motion sensor data from the three runs
39      position_smoothed = {
40          "run1": smooth_df(position["run1"], [5, 3]),
41          "run2": smooth_df(position["run2"], [5, 3, 3]),
42          "run3": smooth_df(position["run3"], [5, 3])
43      }
44
45      # Generate the Line of Best Fit (LOBF) for both the raw data and
    smoothed data from the three runs
46      lobf = {
47          "run1": sin_lobf(position["run1"], time, True)["fitfunc"](time),
48          "run1-smoothed": sin_lobf(position_smoothed["run1"], time, True)["
    fitfunc"](time),
49          "run2": sin_lobf(position["run2"], time, True)["fitfunc"](time),
50          "run2-smoothed": sin_lobf(position_smoothed["run2"], time, True)["
    fitfunc"](time),
51          "run3": sin_lobf(position["run3"], time, True)["fitfunc"](time),
52          "run3-smoothed": sin_lobf(position_smoothed["run3"], time, True)["
    fitfunc"](time),
53      }
54
55      # Generated LOBFs based on smaller chunks of data
56      lobf_piecewise = {
57          "run1": sin_lobf_piecewise(position["run1"], time, 110),
58          "run1-smoothed": sin_lobf_piecewise(position_smoothed["run1"],
    time, 225),
59          "run2": sin_lobf_piecewise(position["run2"], time, 100),
60          "run2-smoothed": sin_lobf_piecewise(position_smoothed["run2"],
```

```
    time, 100),
61          "run3": sin_lobf_piecewise(position["run3"], time, 100),
62          "run3-smoothed": sin_lobf_piecewise(position_smoothed["run3"],
    time, 250),
63      }
64
65      # Display Graph of Run 1 (Raw)
66      plt.figure()
67      plt.plot(time, position["run1"], label="Motion Sensor")
68      plt.plot(displacement_time, displacement, label="Displacement Sensor")
69      if USE_PIECEWISE_LOBF:
70          plt.plot(time, lobf_piecewise["run1"], label="Piecewise Fit Curve"
    )
71      else:
72          plt.plot(time, lobf["run1"], label="Fit Curve")
73      plt.title("Run 1 (Raw)")
74      plt.xlabel("Time [s]")
75      plt.ylabel("Displacement [mm]")
76      plt.legend(loc="best")
77      plt.grid()
78
79      # Display Graph of Run 1 (Smoothed)
80      plt.figure()
81      plt.plot(time, position_smoothed["run1"], label="Motion Sensor")
82      if USE_PIECEWISE_LOBF:
83          plt.plot(time, lobf_piecewise["run1-smoothed"], label="Piecewise
    Fit Curve")
84      else:
85          plt.plot(time, lobf["run1-smoothed"], label="Fit Curve")
86      plt.title("Run 1 (Smoothed)")
87      plt.xlabel("Time [s]")
88      plt.ylabel("Displacement [mm]")
89      plt.legend(loc="best")
90      plt.grid()
91
92      # Display Graph of Run 2 (Raw)
93      plt.figure()
94      plt.plot(time, position["run2"], label="Motion Sensor")
95      if USE_PIECEWISE_LOBF:
96          plt.plot(time, lobf_piecewise["run2"], label="Piecewise Fit Curve"
    )
97      else:
98          plt.plot(time, lobf["run2"], label="Fit Curve")
```

```
    plt.title("Run 2 (Raw)")
    plt.xlabel("Time [s]")
    plt.ylabel("Displacement [mm]")
    plt.legend(loc="best")
    plt.grid()

    # Display Graph of Run 2 (Smoothed)
    plt.figure()
    plt.plot(time, position_smoothed["run2"], label="Motion Sensor")
    if USE_PIECEWISE_LOBF:
        plt.plot(time, lobf_piecewise["run2-smoothed"], label="Piecewise Fit Curve")
    else:
        plt.plot(time, lobf["run2-smoothed"], label="Fit Curve")
    plt.title("Run 2 (Smoothed)")
    plt.xlabel("Time [s]")
    plt.ylabel("Displacement [mm]")
    plt.legend(loc="best")
    plt.grid()

    # Display Graph of Run 3 (Raw)
    plt.figure()
    plt.plot(time, position["run3"], label="Motion Sensor")
    if USE_PIECEWISE_LOBF:
        plt.plot(time, lobf_piecewise["run3"], label="Piecewise Fit Curve")
    else:
        plt.plot(time, lobf["run3"], label="Fit Curve")
    plt.title("Run 3 (Raw)")
    plt.xlabel("Time [s]")
    plt.ylabel("Displacement [mm]")
    plt.legend(loc="best")
    plt.grid()

    # Display Graph of Run 3 (Smoothed)
    plt.figure()
    plt.plot(time, position_smoothed["run3"], label="Motion Sensor")
    if USE_PIECEWISE_LOBF:
        plt.plot(time, lobf_piecewise["run3-smoothed"], label="Piecewise Fit Curve")
    else:
        plt.plot(time, lobf["run3-smoothed"], label="Fit Curve")
    plt.title("Run 3 (Smoothed)")
```

```
139     plt.xlabel("Time [s]")
140     plt.ylabel("Displacement [mm]")
141     plt.legend(loc="best")
142     plt.grid()
143 
144     # Print some statistics
145     print("Max Displacement of Run 1: %g mm or %g mm" % (position["run1"].
        min(), position["run1"].max()))
146     print("Max Displacement of Run 2: %g mm or %g mm" % (position["run2"].
        min(), position["run2"].max()))
147     print("Max Displacement of Run 3: %g mm or %g mm" % (position["run3"].
        min(), position["run3"].max()))
148 
149     # Show all the graphs
150     plt.show()
151 
152 
153 # Imports the .csv file into a pandas DataFrame
154 def import_data(filename):
155     return pd.read_csv(filename,
156                        skiprows=1,
157                        usecols=[0,1,2,7,12])
158 
159 
160 # Smooths a DataFrame
161 # Input Args:
162 #   df: 1 col DataFrame
163 #   smooth: list of smoothing windows, e.g., [5, 3, 3]
164 def smooth_df(df, smooth):
165     smoothed_data = df
166     for win in smooth:
167         # min_periods=1 stops the NaNs
168         smoothed_data = smoothed_data.rolling(win, min_periods=1).mean()
169 
170     return smoothed_data
171 
172 
173 # Modified from https://stackoverflow.com/questions/16716302/how-do-i-fit-
        a-sine-curve-to-my-data-with-pylab-and-numpy
174 # Uses a Fast Fourier Transform to generate an initial guess then uses a
        non-linear least squares algorithm to
175 # generate a matching sine curve
176 def sin_lobf(data, t, stats=False):
```

```
177     # Fit sin to the input time sequence, and return fitting parameters "
        amp", "omega", "phase", "offset", "freq", "period" and "fitfunc"
178     t = np.array(t)
179     data = np.array(data)
180     ff = np.fft.fftfreq(len(t), (t[1]-t[0]))   # assume uniform spacing
181     Fyy = abs(np.fft.fft(data))
182     guess_freq = abs(ff[np.argmax(Fyy[1:])+1])   # excluding the zero
        frequency "peak", which is related to offset
183     guess_amp = np.std(data) * 2.**0.5
184     guess_offset = np.mean(data)
185     guess = np.array([guess_amp, 2.*np.pi*guess_freq, 0., guess_offset])
186
187     def sinfunc(t, A, w, p, c):  return A * np.sin(w*t + p) + c
188     popt, pcov = curve_fit(sinfunc, t, data, p0=guess) # this is the least
         squares bit
189     A, w, p, c = popt
190     f = w/(2.*np.pi)
191     fitfunc = lambda t: A * np.sin(w*t + p) + c
192     dict = {"amp": A, "omega": w, "phase": p, "offset": c, "freq": f, "
        period": 1./f, "fitfunc": fitfunc, "maxcov": np.max(pcov), "rawres": (
        guess,popt,pcov)}
193
194     # Print some stats
195     if stats:
196         print("Amplitude: %g mm\nFrequency: %g Hz\nLOBF: f(t) = %gsin(%g *
         t + %g) + %g\n
        ----------------------------------------------------------" % (dict['
        amp'], dict['freq'], dict['amp'], dict['omega'], dict['phase'], dict['
        offset']))
197     return dict
198
199
200 # This function used the same non-linear least squares algorithm, but it
     struggled
201 # because the initial frequency guess was so finnicky... The above
     function works
202 # much better for resolving the appropriate frequency
203
204 # Generate line of best fit for a sine curve
205 # def sin_lobf(data, t):
206 #     guess_mean = np.mean(data)
207 #     guess_phase = 0
208 #     guess_freq = 2 * np.pi * 10
```

```
#     guess_amp = -1

#     # Define the function to optimize, in this case, we want to minimize the difference
#     # between the actual data and our "guessed" parameters
#     optimize_func = lambda x: x[0] * np.sin(x[1] * (t - x[2])) + x[3] - data
#     opt_amp, opt_freq, opt_phase, opt_mean = leastsq(optimize_func, [guess_amp, guess_freq, guess_phase, guess_mean])[0]

#     # recreate the fitted curve using the optimized parameters
#     return opt_amp * np.sin(opt_freq * (t - opt_phase)) + opt_mean


# Generates a line of best for pieces of the data
def sin_lobf_piecewise(data, t, n):
    pw_lobf = np.zeros(len(data))
    i = 0
    j = n

    # Run the LOBF fit algorithm on consecutive n chunks of data
    while j <= len(data):
        f = sin_lobf(data[i:j], t[i:j])["fitfunc"]
        pw_lobf[i:j] = f(t[i:j])
        i += n
        j += n

    # After the bulk of the data is done, how much is left?
    if j != len(data) and len(data) - i > 1:
        # If more than 1 element is left, run the algorithm per usual
        f = sin_lobf(data[i:len(data)], t[i:len(data)])["fitfunc"]
        pw_lobf[i:len(data)] = f(t[i:len(data)])
    else:
        # If only 1 element is left, just assign the variable :P
        pw_lobf[len(data) - 1] = data[len(data) - 1]
    return pw_lobf


# Calls the main function
if __name__ == "__main__":
```

```
246    main()
```

Listing B.1: Our data analysis script, `Lab-01-Analysis.py`.

B.2. Output

```
Amplitude: -0.911702 mm
Frequency: 9.99878 Hz
LOBF: f(t) = -0.911702sin(62.8242 * t + -0.423851) + -0.458014
-----------------------------------------------------------
Amplitude: 0.513407 mm
Frequency: 9.99862 Hz
LOBF: f(t) = 0.513407sin(62.8232 * t + 0.842215) + -0.459104
-----------------------------------------------------------
Amplitude: -1.77111 mm
Frequency: 9.99939 Hz
LOBF: f(t) = -1.77111sin(62.828 * t + -0.26507) + -0.339128
-----------------------------------------------------------
Amplitude: 0.873077 mm
Frequency: 9.99919 Hz
LOBF: f(t) = 0.873077sin(62.8267 * t + 0.372368) + -0.338719
-----------------------------------------------------------
Amplitude: -1.54725 mm
Frequency: 9.99753 Hz
LOBF: f(t) = -1.54725sin(62.8163 * t + -0.348804) + 0.325918
-----------------------------------------------------------
Amplitude: 0.870026 mm
Frequency: 9.99747 Hz
LOBF: f(t) = 0.870026sin(62.816 * t + 0.912736) + 0.322807
-----------------------------------------------------------
Max Displacement of Run 1: -5.3458 mm or 2.9268 mm
Max Displacement of Run 2: -6.2847 mm or 4.9003 mm
Max Displacement of Run 3: -5.3 mm or 9.8693 mm
```