Final Project Report -- Arduino Based Guitar Tuner

Milad Mesbahi and Andrew Polonsky
Physics 4BL Lab 7
Dec 10, 2021

Abstract

The following project report on the design and assembly of a functioning guitar tuner demonstrates how physics principles of sound and fourier transforms can be coupled with computing software and circuitry to create a device that is used ubiquitously by most guitar players in the world. Our goal for the project is to create a circuit that takes in a sound signal using a microphone, processes the signal in an arduino microcomputer, and outputs the note to which the frequency of the signal corresponds to and whether that frequency is flat, sharp, or in tune with that corresponding note. The final product almost successfully identifies all the notes that the open strings of the guitar correspond to, with the exception of the low E note (82.4 Hz). Additionally, the tuner displays whether or not the frequency is in tune with what the open note of the guitar should be, but these results are slightly skewed to overestimating the actual frequency of the signal and the device tends to lead to a tuning that is flatter than the accepted values of the notes.

Introduction/Background

Our guitar tuner functions on the basic principles of mechanical waves and periodic function properties. When a sound wave is created, it is made up of a combination of simple sinusoidal functions of different amplitudes and frequencies that are multiples of each other, called harmonics. This combination can be written as a sum called the Fourier Series:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\frac{2\pi}{T}nx + \sum_{n=1}^{\infty} b_n \sin\frac{2\pi}{T}nx \quad (2.1)$$

$$a_n = \frac{2}{T}\int_{\tau} f(x)\cos\frac{2\pi}{T}nx\ dx,\ n = 0, 1, 2.. \quad (2.2)$$

$$b_n = \frac{2}{T}\int_{\tau} f(x)\sin\frac{2\pi}{T}nx\ dx,\ n = 1, 2, .. \quad (2.3)$$

This series gives us the signal as a function of a sum of simpler functions that depend on time. However, to identify which frequencies dominate the signal, we need to analyze it in the frequency domain.

The fourier transform is a mathematical operation that takes a signal of the time domain and converts it into a frequency dependent function.

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt$$

Plotting this function allows us to see which frequencies have the highest intensity and thus define the signal. These frequencies are then compared to the frequencies that correspond to the pitches produced by guitar strings and their difference allows us to identify if the input signal is flat or sharp.

The sound signal produced by a guitar string is called a standing wave, because it cannot propagate as it is confined by physical boundaries on each side. A standing wave is also composed of multiple frequencies and as it rings out on a guitar string, it causes air particles around it to oscillate at the same frequency. This air particle oscillation is the resulting sound wave.

Due to the nature of standing waves, they can only oscillate at certain frequencies, which are all multiples of a fundamental frequency. This frequency, which will be depicted as the first peak of a fourier transform, depends on the length of the string that it is on:

Standing Wave Frequencies:

$$f_n = \frac{nv}{2L} = nf_1$$

Fundamental Frequency:

$$f_1 = \frac{v}{2L} = \frac{1}{2L}\sqrt{\frac{F}{\mu}}$$

- *v the speed of the wave, F is the tension in the string, μ is the linear mass density of string.*

When a guitar is tuned, the tension in the string is changed, and therefore the pitch that the string produces also changes. The desired frequencies of for the notes produced by the open strings of the guitar are as follow:

String	Note (and harmonic)	Frequency (Hz)	Fundamental Frequency (Hz)
6th	E2	82.4	20.60
5th	A3	110	27.50
4th	D3	146.83	18.35

3rd	G3	196	24.50
2nd	B3	246.94	30.87
1st	E4	329.63	20.60

Table 1, frequencies of the notes played by a guitar in standard tuning

Knowing the desired frequencies from each string allows us to predict where the desired peaks of the fourier transform for each string should be.

For each string, we predict that the first peak will be at the fundamental frequency. This means that the 1st and 6th string will have their first peak at around the same frequency. However, the tallest peak for each string should be different. We hypothesize that each string's tallest peak will be *n* amounts of peaks away from the first peak, where *n* is the harmonic of the note. This frequency will be close to the frequency listed in the table, but if the guitar is out of tune, then it might be slightly smaller or larger than the desired frequency.

Experimental Setup

Materials Used:

- Arduino Uno starter kit
    - 10k Potentiometer, 2 x 10k and 2k resistors, diode
- Max 9814 Microphone
- Laptop computer
- LCD
- IPhone
- Guitar
- Jumper wires
- Breadboard

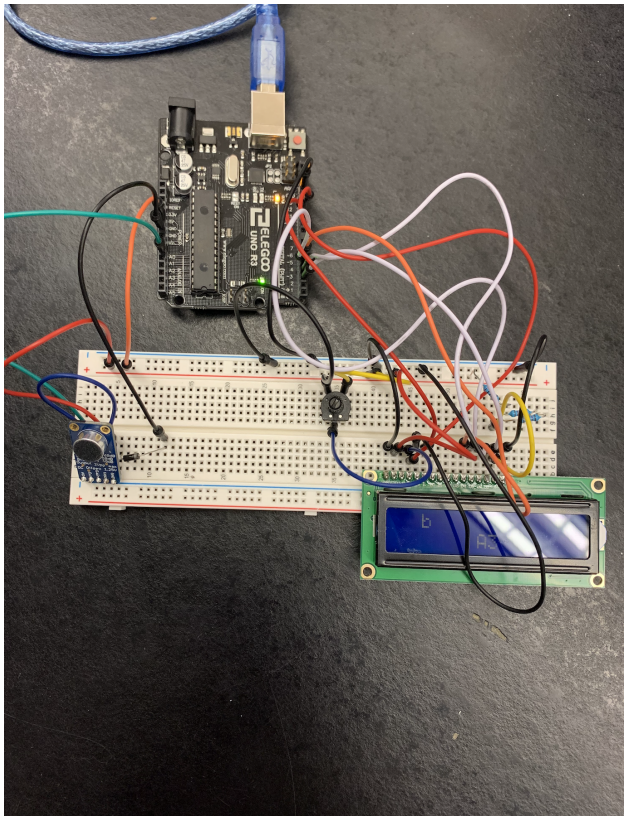

Fig. 1, Guitar Tuner

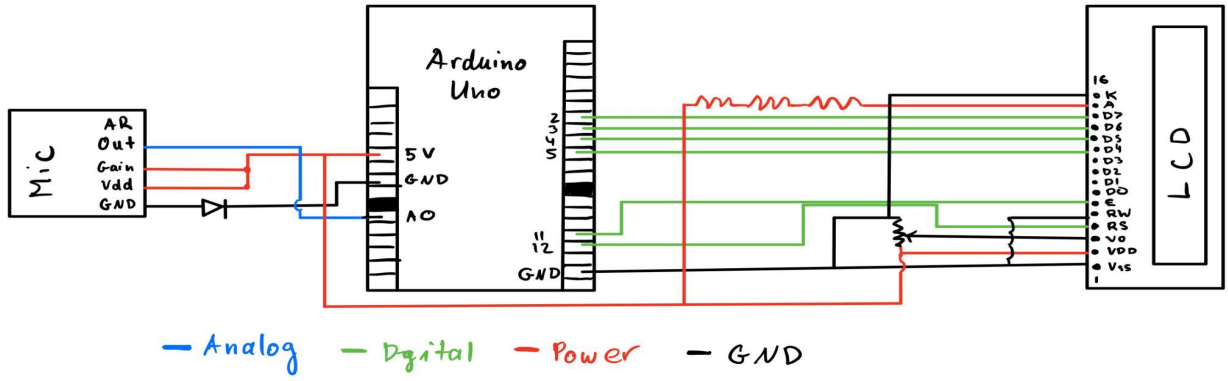

Fig. 2, Circuit diagram

Two circuits were assembled using the arduino, as shown in *Fig 1 and 2*. The first circuit took an analog data input from the microphone and sent it into the arduino. The second circuit received digital signals from the arduino and operated the Liquid Crystal Display.

Using the microphone required a diode pointing to ground to make sure that the sound recording device is not damaged. The LCD required a 22k resistor to make sure it does not receive too much current. Twenty-two thousand ohms of resistance was achieved by placing 2 10k and 1 2k

ohm resistors in series. Additionally, a 10k Ohm potentiometer was used in the LCD circuit to vary the display's brightness.

Methods / Code Explanation

As mentioned previously, we used a Fast Fourier Transform function from an arduino library to obtain the analog sound signal as a function of frequency. After that function was acquired, the code looped through it to identify the frequency of the highest intensity. That frequency was then compared to frequency ranges that correspond to each note (the frequency range increases with each harmonic):

Note	Frequency Range (Hz)
E2	75 - 90
A3	100 - 120
D3	130 - 150
G3	180 - 210
B3	230 - 260
E4	310 - 350

Table 2, frequency ranges that were used to identify a given frequency with a note from an open string.

Once the highest intensity frequency was placed into a range, the tuner was able to identify which note the user was trying to tune to. Next, the code determined if the input frequency was above, below, or within 1 Hz of the desired frequency (considering this as "in-tune") and sent out the result to the LCD.

Python Analysis

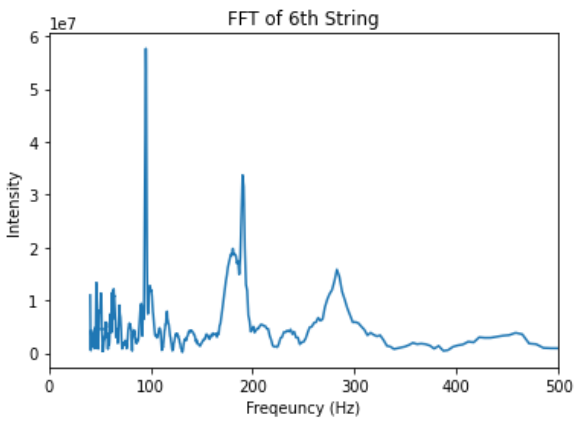

Fig. 3: FFT of 6th string of untuned guitar

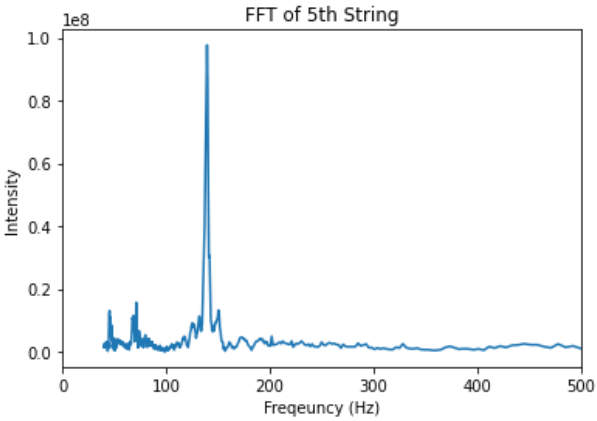

Fig. 4: FFT of 5th string of untuned guitar

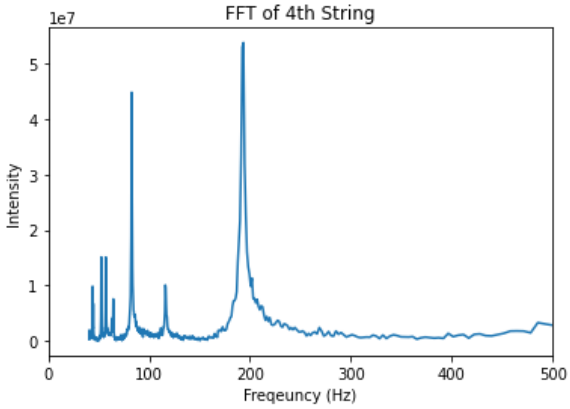

Fig. 5: FFT of 4th string of untuned guitar

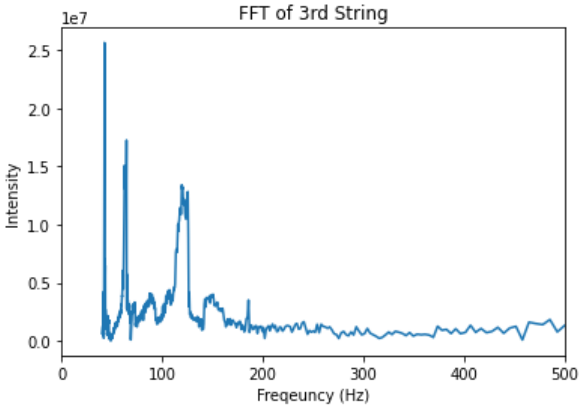

Fig. 6: FFT of 3rd string of untuned guitar

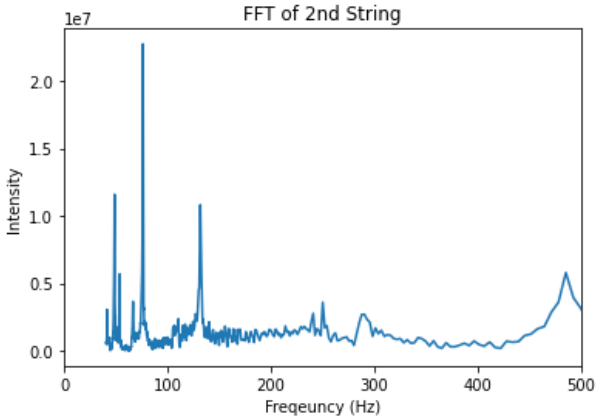

Fig. 7: FFT of 2nd string of untuned guitar

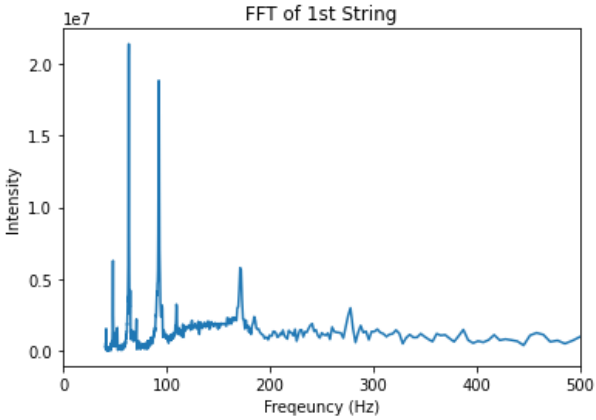

Fig. 8: FFT of 1st string of untuned guitar

In *fig 3.* the FFT measures the 6th string to be at 95 Hz (instead of 82.4 Hz), which is the location of the tallest peak. This peak is also the third peak because it represents the second harmonic, which is preceded by the first harmonic and the fundamental frequency.

A similar trend is seen in *fig. 4 and fig. 5*, in which the fourth peaks are the highest and also have a slightly larger frequency than the note played by that string should have. These peaks are at 140 Hz for the fifth string (A note) and 193 Hz for the fourth string (D note) instead of being at 110 Hz and 146.83 Hz, respectively. The overshoots in frequencies might be caused by either the guitar being out of tune or the microphone being unable to accurately receive and transmit a sound signal.

For *fig. 6 - 8*, we see that the peaks that should represent the note played by the strings are not the most intense ones. Instead, the 4th peaks in *fig. 6 and 7* and the 5th peak in *fig. 8* are relatively short, which could possibly be explained by noise in the signal. However, these peaks are the closest to the frequency that the guitar strings should have so they are the ones of interest for us. In *fig. 6* the frequency of the fourth peak is 190 Hz, but the desired frequency for that string is 196 Hz (G note). In *fig. 7,* the frequency of the fourth peak is 250 Hz, but the desired frequency for that string is 246.94 Hz (B note). Finally, in *fig. 8*, the frequency of the 5th peak is 329.63 Hz, but the desired frequency for that string is 290 Hz (E note).

Results

Fig. 9: results from professional guitar tuner after using our tuner to tune the guitar

As seen above, the tuner generally tuned notes too flat, which limited its accuracy and caused the notes played to sound out of tune. Furthermore, the tuner could not pick up the low E note, which is why it is not shown above. This may require signal amplification for better results in the future.

Conclusion

In conclusion, our Arduino based guitar tuner was accurate in identifying the corresponding notes of the guitar's open strings expect for that of the low E note. However, the tuner struggled to accurately display when a note was in tune, as it repeatedly overestimated the actual frequency of a note played and led to a tuning that was flatter than the accepted value of a specific note. This was discovered by individually strumming the guitar's strings in the range of a legitimate guitar tuner after it had been tuned with our Arduino based tuner. As shown in the results section, nearly all the notes played were reported as being tuned to a value that was considered "too low" by the app, thus highlighting the limitations of our tuner and its inability to precisely indicate when a note was in tune. This could be explained by the limitations of our hardware, as the microphone has a certain range of frequencies and intensities that it can pick up. Furthermore, Arduino has limited computational power, so it fails to perform a very accurate FFT.

There are multiple additions and corrections we would have made to this experiment if given more time or if pursued again in the future. One of these things would be to perform the experiment in a controlled environment to improve the accuracy of the tuner. This would include attempting to use broadband filters to filter out noise and unwanted frequencies. This would yield better results and limit any random errors we faced. Another thing we would like to do is create a tuner that can differentiate sharps and flats of different notes. This would not only increase the accuracy of our tuner but would make it much more useful and multifunctional. Lastly, we would have liked to 3D print a case for the tuner to make it more aesthetically pleasing, damage protected and easy to carry.

Appendix / Code

Arduino:

```
//LCD Libraries
#include <Wire.h>
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
//FFT
#include "arduinoFFT.h"

#define SAMPLES 128
#define SAMPLING_FREQUENCY 1000

arduinoFFT FFT = arduinoFFT();

unsigned int sampling_period_us;
unsigned long microseconds;

double vReal[SAMPLES];
double vImag[SAMPLES];

//tuning variables
String tune = "IN TUNE";
void setup()
{
  Serial.begin(9600);
  lcd.begin(16,2);
  //lcd.backlight();
  lcd.setCursor(0,1);
  lcd.print(" Guitar Tuner");
  delay(3000);
  lcd.clear();
  lcd.setCursor(2,0);
  lcd.print("Play a note!");
  sampling_period_us = round(1000000*(1.0/SAMPLING_FREQUENCY));
}
```

```
void loop()
{
  //Sampling
  for(int i = 0; i<SAMPLES; i++)
  {
    microseconds = micros();

    vReal[i] = analogRead(0);
    vImag[i] = 0;

    while(micros() < (microseconds + sampling_period_us)){
    }
  }

  //FFT
  FFT.Windowing(vReal, SAMPLES, FFT_WIN_TYP_HAMMING, FFT_FORWARD);
    FFT.Compute(vReal, vImag, SAMPLES, FFT_FORWARD);
    FFT.ComplexToMagnitude(vReal, vImag, SAMPLES);
    double peak = FFT.MajorPeak(vReal, SAMPLES, SAMPLING_FREQUENCY);

    double frequency = peak;
    /*PRINT RESULTS*/
//   lcd.clear();
//   lcd.print(peak);     //Print out what frequency is the most dominant.
//   delay(1000);
//
//   displayToLCD(peak);
if(frequency == -1)
  {
    return;
  }
 //E2
 if(frequency >= 75 && frequency <= 90)
{
  lcd.clear();
  lcd.setCursor(7,1); //center it
  lcd.print("E2");
  if(frequency < 81)
  {
    lcd.setCursor(2,0);//to the left
```

```
   lcd.print("b");
  }
  if(frequency > 83.5)
  {
   lcd.setCursor(14,0); //to the right
   lcd.print("#");
  }
  if(frequency >= 81 && frequency <= 83.5)
  {
   lcd.clear();
   lcd.setCursor(7,0);
   lcd.print("E2");
   lcd.setCursor(4,1);
   lcd.print(tune);
  }
}
//A3
 else if(frequency <= 120.00 && frequency >= 100.00)
{
  lcd.clear();
  lcd.setCursor(7,1); //center it
  lcd.print("A3");
  if(frequency < 109.00)
  {
   lcd.setCursor(2,0);//to the left
   lcd.print("b");
  }
  if(frequency > 111.00)
  {
   lcd.setCursor(14,0); //to the right
   lcd.print("#");
  }
  if(frequency >= 109.00 && frequency <= 111.00)
  {
   lcd.clear();
   lcd.setCursor(7,0);
   lcd.print("A3");
   lcd.setCursor(4,1);
   lcd.print(tune);
  }
```

```
}
//D3
else if(frequency >= 130.00 && frequency <= 150.00)
{
  lcd.clear();
  lcd.setCursor(7,1); //center it
  lcd.print("D3");
  if(frequency < 146)
  {
    lcd.setCursor(2,0);//to the left
    lcd.print("b");
  }
  if(frequency > 148)
  {
    lcd.setCursor(14,0); //to the right
    lcd.print("#");
  }
  if(frequency >= 146 && frequency <= 148)
  {
    lcd.clear();
    lcd.setCursor(7,0);
    lcd.print("D3");
    lcd.setCursor(4,1);
    lcd.print(tune);
  }
}
//G3
else if(frequency >= 180.00 && frequency <= 210.00)
{
  lcd.clear();
  lcd.setCursor(7,1); //center it
  lcd.print("G3");
  if(frequency < 195.00)
  {
    lcd.setCursor(2,0);//to the left
    lcd.print("b");
  }
  if(frequency > 197.00)
  {
    lcd.setCursor(14,0); //to the right
```

```
    lcd.print("#");
  }
  if(frequency >= 195.00 && frequency <= 197.00)
  {
    lcd.clear();
    lcd.setCursor(7,0);
    lcd.print("G3");
    lcd.setCursor(4,1);
    lcd.print(tune);
  }
}
//B3
else if(frequency >= 230.00 && frequency <= 260.00)
{
  lcd.clear();
  lcd.setCursor(7,1); //center it
  lcd.print("B3");
  if(frequency < 245)
  {
    lcd.setCursor(2,0);//to the left
    lcd.print("b");
  }
  if(frequency > 248)
  {
    lcd.setCursor(14,0); //to the right
    lcd.print("#");
  }
  if(frequency >=245 && frequency <= 248)
  {
    lcd.clear();
    lcd.setCursor(7,0);
    lcd.print("B3");
    lcd.setCursor(4,1);
    lcd.print(tune);
  }
}
//E4
else if(frequency >= 310.00 && frequency <= 350.00)
{
  lcd.clear();
```

```
  lcd.setCursor(7,0); //center it
  lcd.print("E4");
  if(frequency < 328)
  {
    lcd.setCursor(2,0);//to the left
    lcd.print("b");
  }
  if(frequency > 330)
  {
    lcd.setCursor(14,0); //to the right
    lcd.print("#");
  }
  if(frequency >= 328 && frequency <= 330)
  {
    lcd.clear();
    lcd.setCursor(7,0);
    lcd.print("E4");
    lcd.setCursor(4,1);
    lcd.print(tune);
  }
}
}
```

Python:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft
from scipy.io import wavfile
from google.colab import drive
from scipy.signal import find_peaks
drive.mount('drive')

sound_data = np.loadtxt('/content/drive/MyDrive/CSV Data Files/Physics
4BL/Copy of sound_file_1_0.csv', delimiter = ',')

index = np.linspace(0, len(sound_data), len(sound_data))

#normalized_data = sound_data - min(sound_data)
```

```
#normalized_data = normalized_data / max(normalized_data)


time = np.linspace(0, (len(sound_data)-1) / 32500., len(sound_data))
plt.plot(time, sound_data)

frequency = 1.0 /time
fourier_transform = np.fft.fft(sound_data)
transformed_data = np.abs(fourier_transform)
#plt.plot(frequency, transformed_data)
#plt.xlim(0,800)
trimmed_frequency = frequency[1:800]
trimmed_data = transformed_data[1:800]
plt.plot(trimmed_frequency, trimmed_data)
plt.xlim(0,800)

print(max(trimmed_data))

sound_data = np.loadtxt('/content/drive/MyDrive/CSV Data Files/Physics
4BL/Low_E_0.csv', delimiter = ',')

index = np.linspace(0, len(sound_data), len(sound_data))

#normalized_data = sound_data - min(sound_data)
#normalized_data = normalized_data / max(normalized_data)


time = np.linspace(0, (len(sound_data)-1) / 32500., len(sound_data))
plt.plot(time, sound_data)

frequency = 1.0 /time
fourier_transform = np.fft.fft(sound_data)
transformed_data = np.abs(fourier_transform)
#plt.plot(frequency, transformed_data)
#plt.xlim(0,800)
trimmed_frequency = frequency[1:800]
trimmed_data = transformed_data[1:800]
plt.plot(trimmed_frequency, trimmed_data)
plt.xlim(0,500)
plt.title("FFT of 6th String")
```

```
plt.xlabel("Freqeuncy (Hz)")
plt.ylabel("Intensity")
print(max(trimmed_data))

sound_data = np.loadtxt('/content/drive/MyDrive/CSV Data Files/Physics
4BL/A_note_1.csv', delimiter = ',')

index = np.linspace(0, len(sound_data), len(sound_data))

#normalized_data = sound_data - min(sound_data)
#normalized_data = normalized_data / max(normalized_data)


time = np.linspace(0, (len(sound_data)-1) / 32500., len(sound_data))
plt.plot(time, sound_data)

frequency = 1.0 /time
fourier_transform = np.fft.fft(sound_data)
transformed_data = np.abs(fourier_transform)
#plt.plot(frequency, transformed_data)
#plt.xlim(0,800)
trimmed_frequency = frequency[1:800]
trimmed_data = transformed_data[1:800]
plt.plot(trimmed_frequency, trimmed_data)
plt.xlim(0,500)
plt.title("FFT of 5th String")
plt.xlabel("Freqeuncy (Hz)")
plt.ylabel("Intensity")
print(max(trimmed_data))

sound_data = np.loadtxt('/content/drive/MyDrive/CSV Data Files/Physics
4BL/D_note_2.csv', delimiter = ',')

index = np.linspace(0, len(sound_data), len(sound_data))

#normalized_data = sound_data - min(sound_data)
#normalized_data = normalized_data / max(normalized_data)


time = np.linspace(0, (len(sound_data)-1) / 32500., len(sound_data))
plt.plot(time, sound_data)
```

```
frequency = 1.0 /time
fourier_transform = np.fft.fft(sound_data)
transformed_data = np.abs(fourier_transform)
#plt.plot(frequency, transformed_data)
#plt.xlim(0,800)
trimmed_frequency = frequency[1:800]
trimmed_data = transformed_data[1:800]
plt.plot(trimmed_frequency, trimmed_data)
plt.xlim(0,500)
plt.title("FFT of 4th String")
plt.xlabel("Freqeuncy (Hz)")
plt.ylabel("Intensity")
print(max(trimmed_data))

sound_data = np.loadtxt('/content/drive/MyDrive/CSV Data Files/Physics
4BL/G_note_3.csv', delimiter = ',')

index = np.linspace(0, len(sound_data), len(sound_data))

#normalized_data = sound_data - min(sound_data)
#normalized_data = normalized_data / max(normalized_data)


time = np.linspace(0, (len(sound_data)-1) / 32500., len(sound_data))
plt.plot(time, sound_data)

frequency = 1.0 /time
fourier_transform = np.fft.fft(sound_data)
transformed_data = np.abs(fourier_transform)
#plt.plot(frequency, transformed_data)
#plt.xlim(0,800)
trimmed_frequency = frequency[1:800]
trimmed_data = transformed_data[1:800]
plt.plot(trimmed_frequency, trimmed_data)
plt.xlim(0,500)
plt.title("FFT of 3rd String")
plt.xlabel("Freqeuncy (Hz)")
plt.ylabel("Intensity")
print(max(trimmed_data))
```

```
sound_data = np.loadtxt('/content/drive/MyDrive/CSV Data Files/Physics
4BL/B_note_4.csv', delimiter = ',')

index = np.linspace(0, len(sound_data), len(sound_data))

#normalized_data = sound_data - min(sound_data)
#normalized_data = normalized_data / max(normalized_data)


time = np.linspace(0, (len(sound_data)-1) / 32500., len(sound_data))
plt.plot(time, sound_data)

frequency = 1.0 /time
fourier_transform = np.fft.fft(sound_data)
transformed_data = np.abs(fourier_transform)
#plt.plot(frequency, transformed_data)
#plt.xlim(0,800)
trimmed_frequency = frequency[1:800]
trimmed_data = transformed_data[1:800]
plt.plot(trimmed_frequency, trimmed_data)
plt.xlim(0,500)
plt.title("FFT of 2nd String")
plt.xlabel("Freqeuncy (Hz)")
plt.ylabel("Intensity")
print(max(trimmed_data))

sound_data = np.loadtxt('/content/drive/MyDrive/CSV Data Files/Physics
4BL/High_E_1.csv', delimiter = ',')

index = np.linspace(0, len(sound_data), len(sound_data))

#normalized_data = sound_data - min(sound_data)
#normalized_data = normalized_data / max(normalized_data)


time = np.linspace(0, (len(sound_data)-1) / 32500., len(sound_data))
plt.plot(time, sound_data)

frequency = 1.0 /time
fourier_transform = np.fft.fft(sound_data)
```

```
transformed_data = np.abs(fourier_transform)
#plt.plot(frequency, transformed_data)
#plt.xlim(0,800)
trimmed_frequency = frequency[1:800]
trimmed_data = transformed_data[1:800]
plt.plot(trimmed_frequency, trimmed_data)
plt.xlim(0,500)
plt.title("FFT of 1st String")
plt.xlabel("Freqeuncy (Hz)")
plt.ylabel("Intensity")
print(max(trimmed_data))
```