EE 242 Lab 2b – Frequency Domain Representation of Signals - Fourier Transform

Aaron McBride, Ben Eisenhart, Yehoshua Luna

This lab has 2 exercises to be completed as a team. Each should be given a separate code cell in your Notebook, followed by a markdown cell with report discussion. Your notebook should start with a markdown title and overview cell, which should be followed by an import cell that has the import statements for all assignments. For this assignment, you will need to import: numpy, the wavfile package from scipy.io, simpleaudio/librosa, and matplotlib.pyplot.

In [15]:
```
# We'll refer to this as the "import cell." Every module you import should be imported here.
import numpy as np
import matplotlib.pyplot as plt

# import whatever other modules you use in this lab -- there are more that you need than we've included
from IPython.display import display, Audio
from scipy.io import wavfile
```

Summary

In this lab, we will learn how to transform signals from the time domain to the frequency domain. The concepts we'll focus on include using a discrete implementation of the Fourier Transform (DFT) with a digitized signal and understanding the relationship between the discrete DFT index k and frequency ω for both the original continuous signal x(t). This is a one-week lab. The lab itself will be due the dayb before your next lab section (i.e. if your lab is on Monday, then the lab is due the following Sunday at 11:59 PM)

Lab 2b turn in checklist

• Lab 2b Jupyter notebook with code for the 2 exercises assignment in separate cells. Each assignment cell should contain markdown cells (same as lab overview cells) for the responses to lab report questions. Include your lab members' names at the top of the notebook.

Please submit the report as PDF

Assignment 3 -- Analyzing frequency content of a signal

For this assignment, you will use a discrete Fourier transform (specifically, the Python implementation of an FFT) to analyze the frequency content of the 100ms segment of the horn signal (from 200ms to 300ms) from assignment 2A. Because this is a periodic signal, the frequency content will have spikes, but because it is a discrete-time signal, they will have finite height. You will experiment with different FFT sizes and different plotting options. The description below assumes that you import numpy as np.

A. Use the np.fft.fft function to compute the FFT for the 100 ms horn signal, with an fft size of nfft=1024, which you can call **x_f**. Recall that the result of the FFT will be a vector that spans frequencies [0, f_s]. If this is a real-valued signal, then the first half of the FFT matters: [0,nfft/2]. **x_f** is not returned in the desired order so in order to get both the positive and negative frequencies, you need to use the np.fft.fftshift function to get **x_f2**. Create two different plots of the magnitude of result using **(np.abs(.))** in a 2x1 view: one with positive and negative frequencies and one with just positive frequencies. Be sure to scale the magnitude according to time signal window length f_s and signal duration. Label the frequency axis in terms of Hz by creating a vector **freq** that scales the FFT index by f_s/nfft.

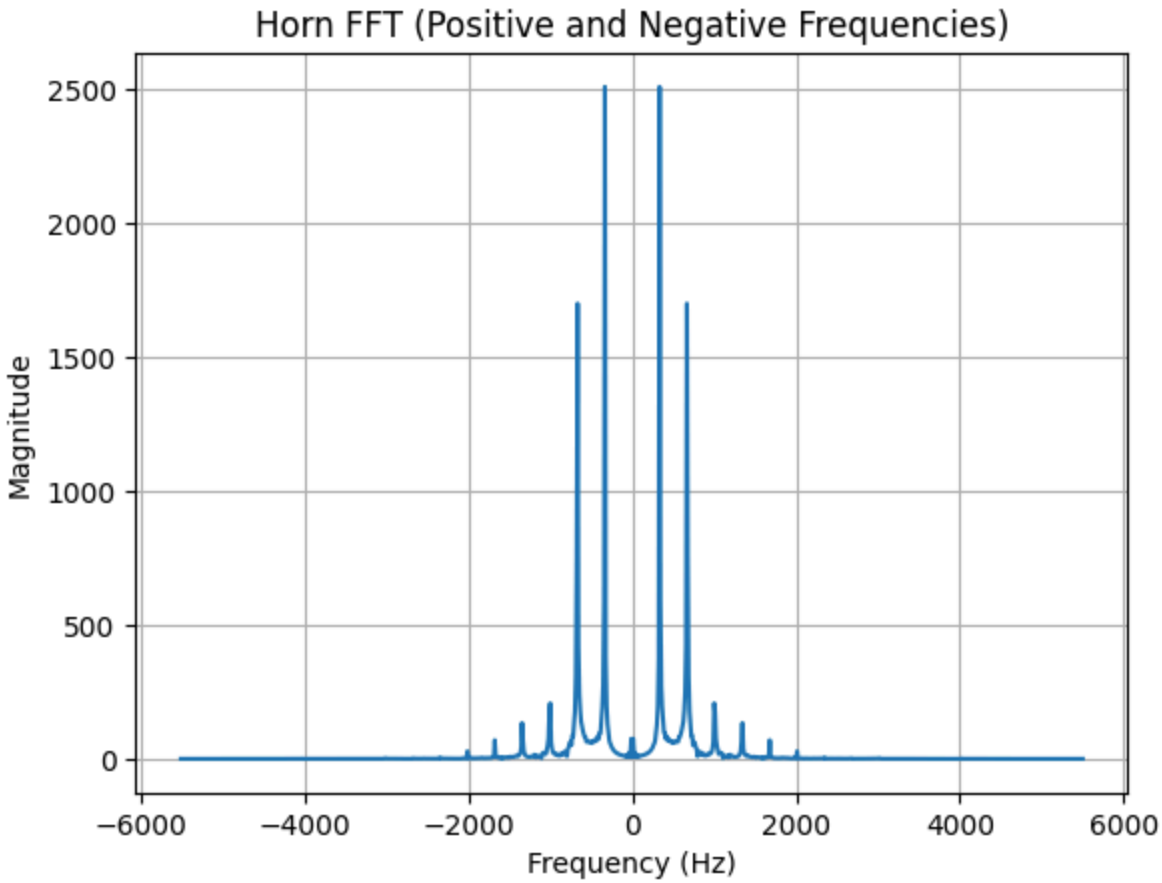
B. It is often the case that frequency content is plotted on a log scale. Plot the one-sided (positive frequency) magnitude using using a log scale.

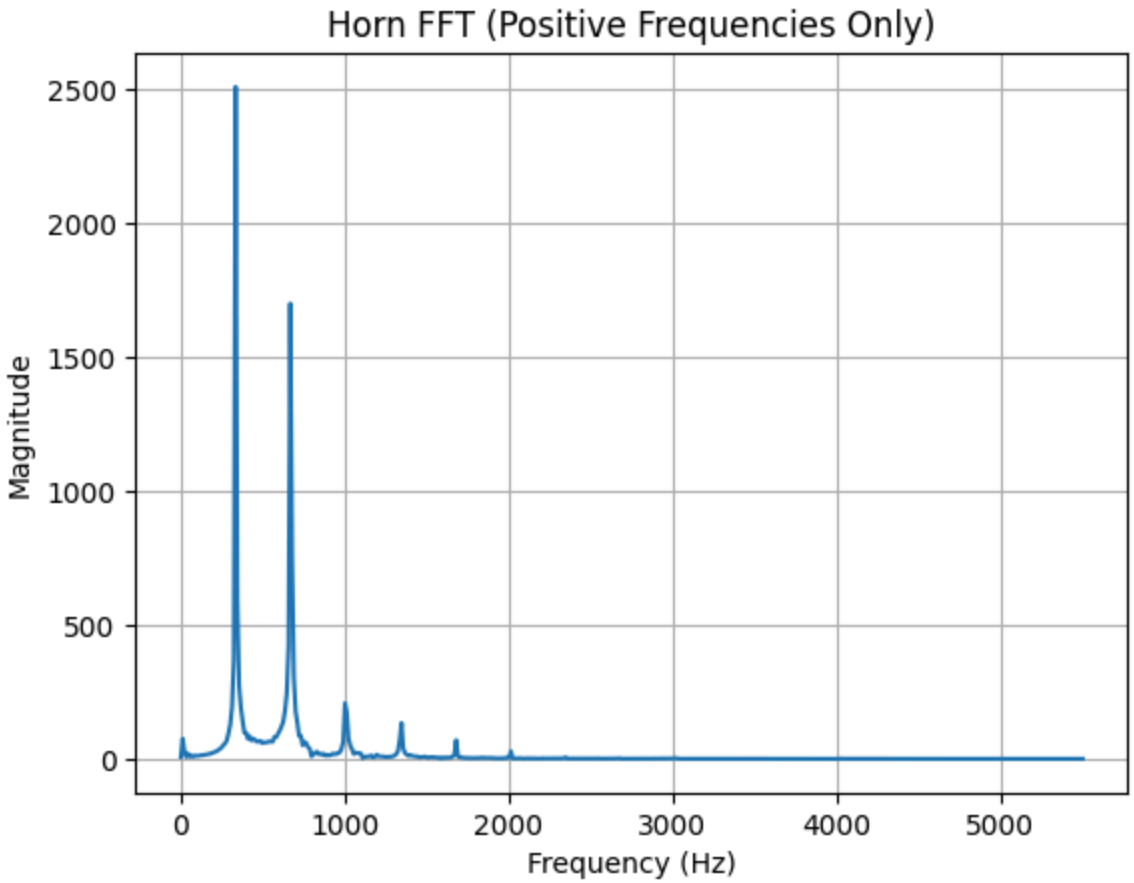
C. Changing the size of the FFT will change the frequency resolution, but it also changes the shape of the result a bit. Just as we saw with Gibbs phenomenon where increasing the number of Fourier series coefficients gave a high frequency ringing at sharp edges, increasing the FFT window will give a "ringing" effect for sharp peaks in frequency. To see this effect, compute the FFT using nfft=2048 and plot the log magnitude only using positive frequencies. Compare to your plot from Part B. The effect is easier to see on a log scale.

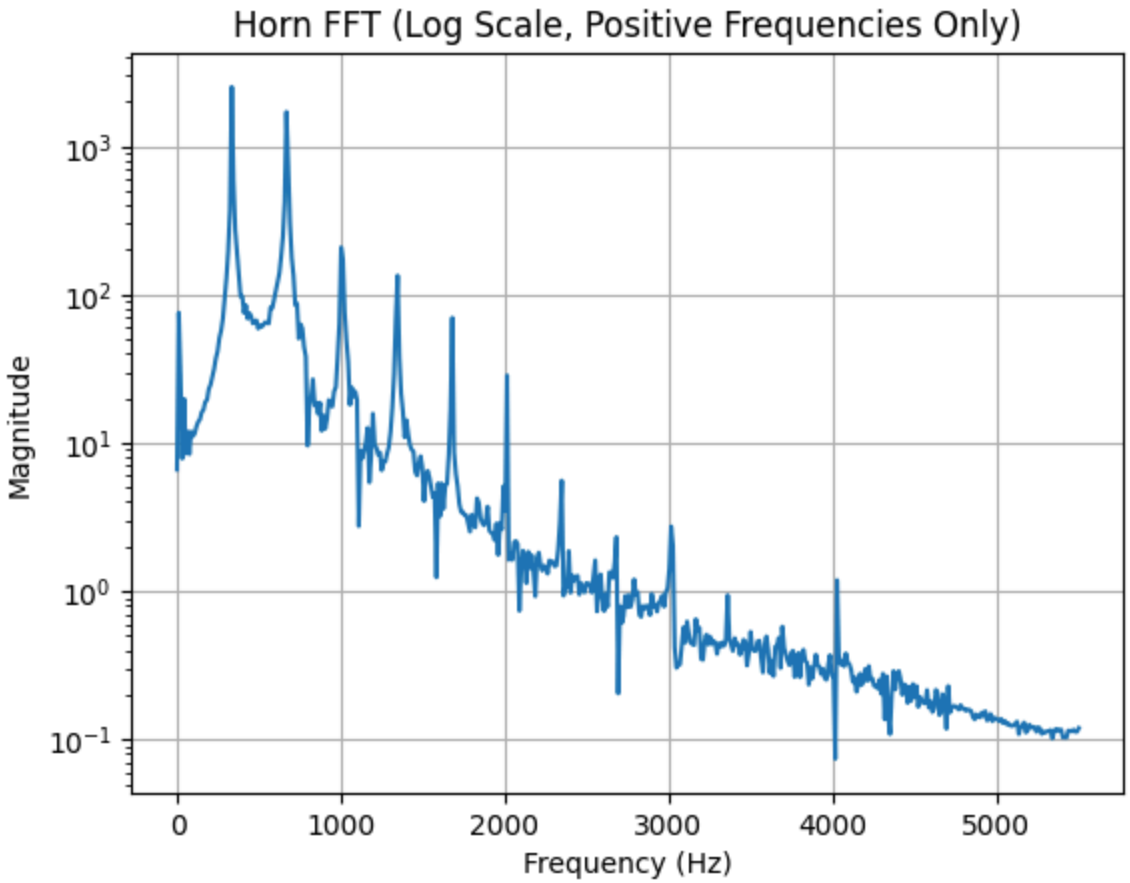
In [14]:
```
################################################################################
# Part A
################################################################################

# Load the horn signal and extract a 100ms segment from 200ms to 300ms
fs, horn_sig = wavfile.read('horn11short.wav')
start_idx = int(0.2 * fs)
end_idx = int(0.3 * fs)
horn_segment = horn_sig[start_idx:end_idx]
```

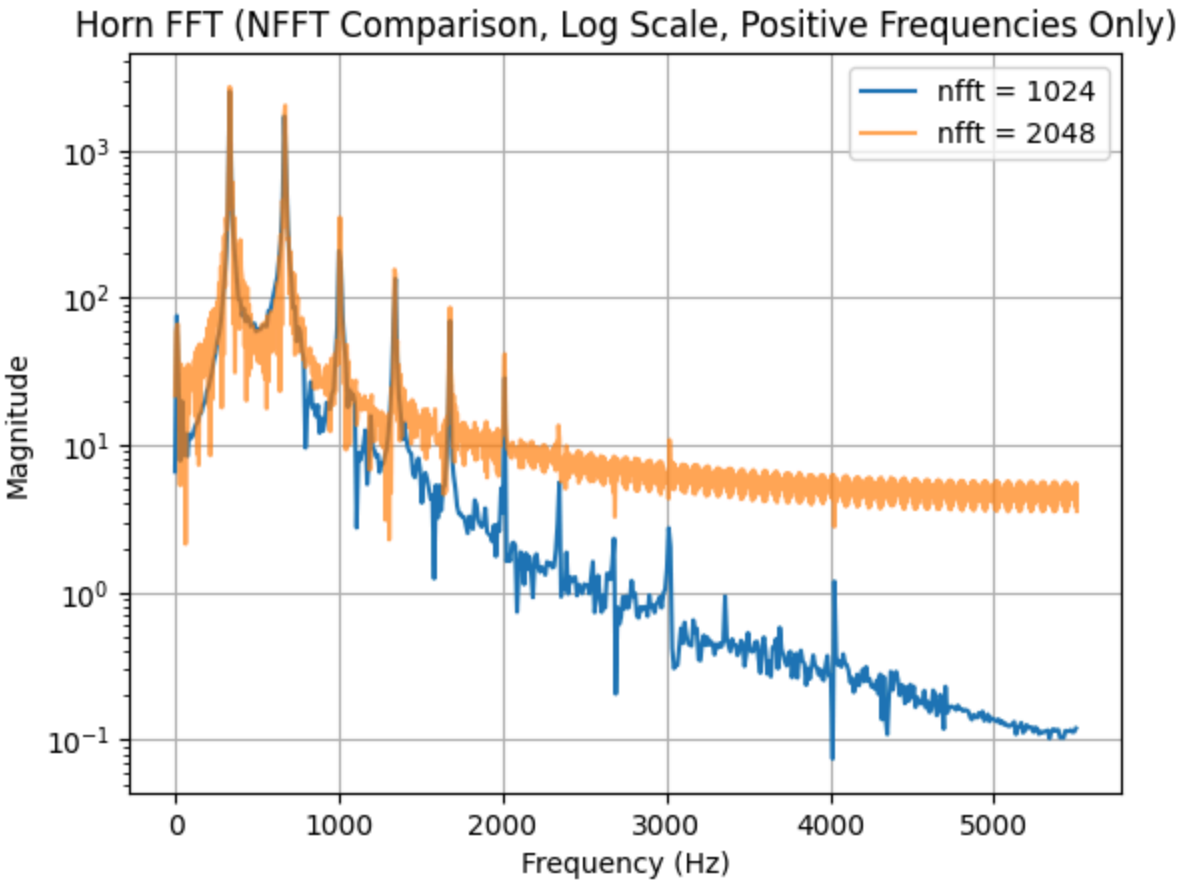
```
signal_duration = 0.1

# Compute the FFT and FFT shift
nfft = 1024
x_f = np.fft.fft(horn_segment, n=nfft)
x_f2 = np.fft.fftshift(x_f)

# Create frequency axes
freq_full = np.fft.fftshift(np.fft.fftfreq(nfft, d=1/fs))
freq_positive = np.fft.fftfreq(nfft, d=1/fs)[:nfft // 2]

# Scale magnitude based on signal duration and sample rate
scaling_factor = 1 / (fs * signal_duration)
x_f2_mag = np.abs(x_f2) * scaling_factor
x_f_mag_positive = np.abs(x_f)[:nfft // 2] * scaling_factor

# Plot positive and negative frequencies
plt.figure()
plt.plot(freq_full, x_f2_mag)
plt.title("Horn FFT (Positive and Negative Frequencies)")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Magnitude")
plt.grid(True)

# Plot only positive frequencies
plt.figure()
plt.plot(freq_positive, x_f_mag_positive)
plt.title("Horn FFT (Positive Frequencies Only)")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Magnitude")
plt.grid(True)

##############################################################################
# Part B
##############################################################################

# Plot FFT on log x-axis (positive frequencies only)
plt.figure()
plt.plot(freq_positive, x_f_mag_positive)
plt.title("Horn FFT (Log Scale, Positive Frequencies Only)")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Magnitude")
```

```
plt.grid(True)
plt.yscale('log')

##############################################################################
# Part C
##############################################################################

# Compute larger FFT with nfft = 2048
nfft_large = 2048
x_f_large = np.fft.fft(horn_segment, n=nfft_large)
freq_positive_large = np.fft.fftfreq(nfft_large, d=1/fs)[:nfft_large // 2]
x_f_mag_positive_large = np.abs(x_f_large)[:nfft_large // 2] * scaling_factor

# Plot log-scale FFT with nfft = 1024 & 2048
plt.figure()
plt.plot(freq_positive, x_f_mag_positive, label='nfft = 1024')
plt.plot(freq_positive_large, x_f_mag_positive_large, label='nfft = 2048', alpha=0.7)
plt.title("Horn FFT (NFFT Comparison, Log Scale, Positive Frequencies Only)")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Magnitude")
plt.grid(True)
plt.yscale('log')
plt.legend()
plt.show()

# Show all plots
plt.show()
```


Horn FFT (Positive and Negative Frequencies)
Magnitude
Frequency (Hz)
2500
2000
1500
1000
500
0
−6000
−4000
−2000
0
2000
4000
6000


Horn FFT (Positive Frequencies Only)
Magnitude
Frequency (Hz)
0
500
1000
1500
2000
2500
0
1000
2000
3000
4000
5000


Horn FFT (Log Scale, Positive Frequencies Only)
Magnitude
10^3
10^2
10^1
10^0
10^{-1}
0
1000
2000
3000
4000
5000
Frequency (Hz)



Discussion

In assignment 2, we used specific cosine frequencies to approximate the horn note, assuming the signal is periodic so the harmonics have non-zero energy. The FFT results show a different picture, and the synthesized version is easily distinguished from the original. Discuss reasons for these differences.

Answer

The reason why the synthesized horn looks different from the actual horn is due to the number of frequencies present in the sound. From the FFT graphs above, we can clearly see that the sound of the horn is composed of hundreds - if not thousands of individual frequencies. Although the fundamental frequency and its harmonics have the largest magnitude, other frequencies with smaller

amplitudes are clearly present. When synthesizing the sound of the horn, we only use cosine frequencies corresponding with the fundamental and its harmonics, thus leaving out all of these smaller sounds which makes it sound quite different from the original. Additionally, the original horn signal is not periodic in reality. This means that the sound does change slightly over time, and our synthesization fails to account for that.

Assignment 4 -- Comparing frequency content of a signal

Many interesting time signals have changing frequency content. Music is one example, since different notes have different fundamental frequency. Speech is another example: we distinguish different vowels and consonants based on their frequency content. In this assignment, you will use the FFT to compare the frequency content of two different speech sounds in a sentence. We'll use 30ms windows, where the frequency content is relatively stable.

A. Download the signal "bluenose3.wav", and read in the file. Plot the full waveform, using the sampling frequency to correctly label the time axis. Play the file.

B. Extract the samples corresponding to times [0.75,0.78]. (This corresponds to the "oo" sound in the word "grew.") Using a 2x1 plot, plot the time waveform (labeling the time axis with the specified time region) and the magnitude of the frequency response (positive frequencies only, labeling the frequency axis in Hz).

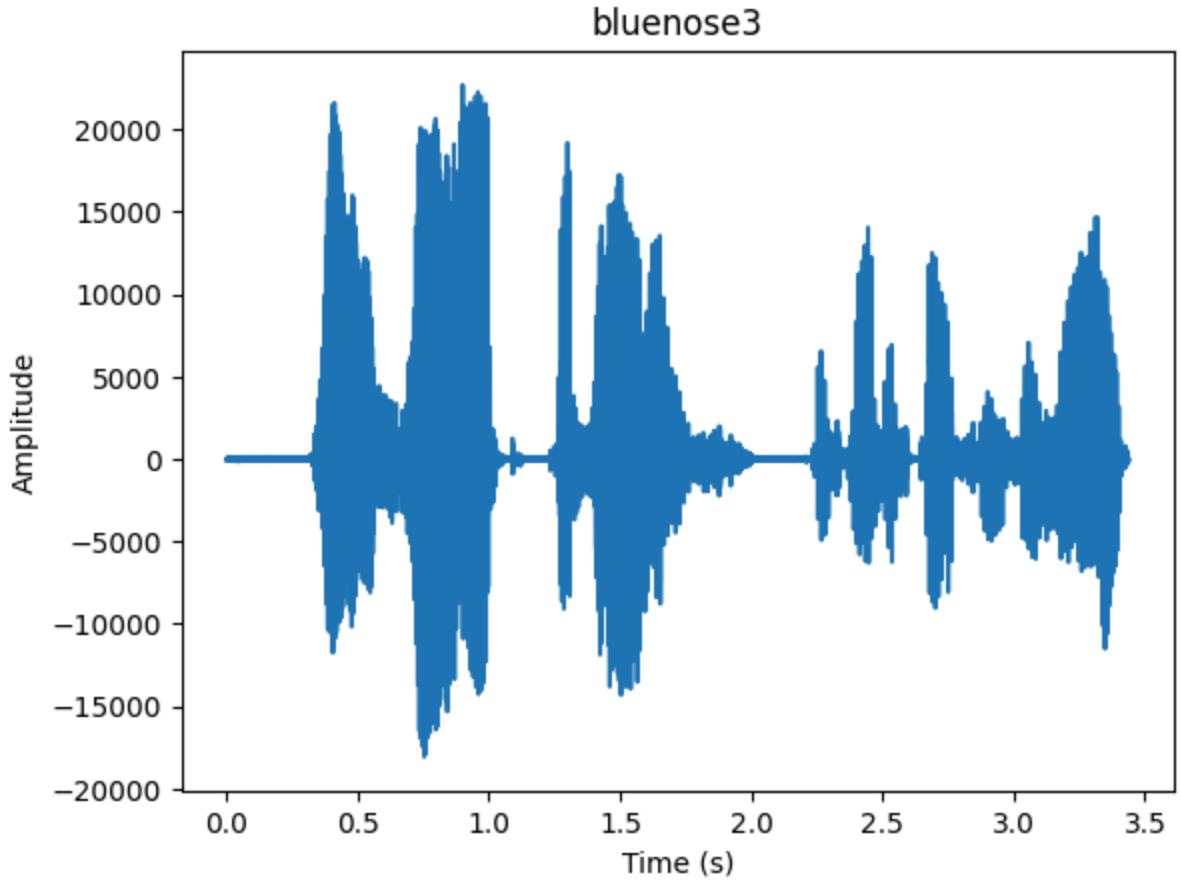
C. Repeat the exercise above using the samples corresponding to times [2.565,2.595]. (This corresponds to the "s" sound.)

In [ ]:
```
################################################################################
# Part A
################################################################################

# Loads the bluenose3 signal and creates a time axis for it.
blue_fs, bluenose3 = wavfile.read('bluenose3.wav')
blue_time = np.linspace(0, len(bluenose3)/blue_fs, num = len(bluenose3))
print("Sampling frequency:", blue_fs)

# Plots the bluenose3 signal in the time domain.
plt.plot(blue_time, bluenose3)
plt.title('bluenose3')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.show()
```

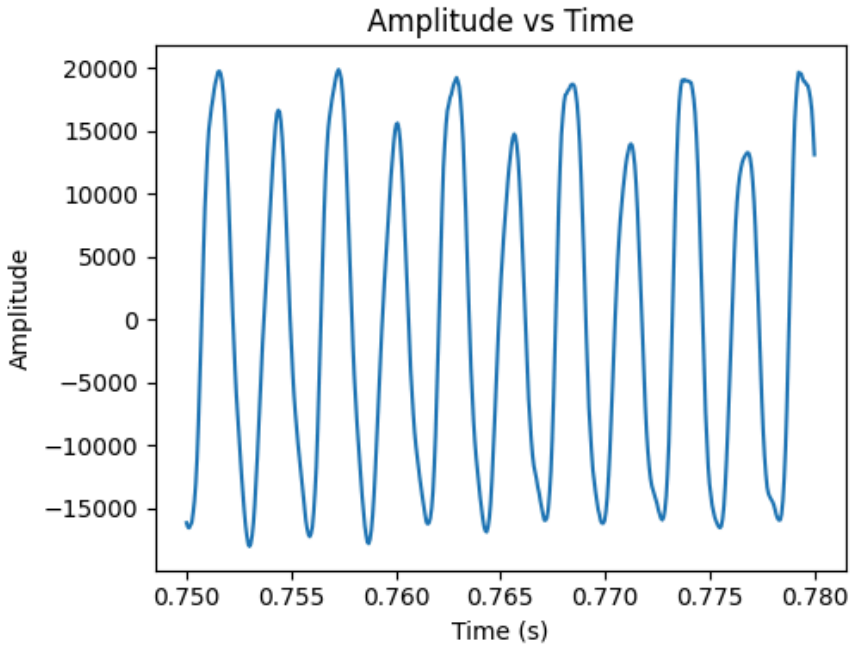
```
display(Audio(bluenose3, rate=blue_fs)) # Plays the sound

################################################################################
# Part B
################################################################################

# Extracts a 30ms segment of the 'ooo' sound and creates another time axis.
ooo_part = bluenose3[int(0.75 * blue_fs) : int(0.78 * blue_fs)]
ooo_time = np.linspace(0.75, 0.78, num = len(ooo_part))

# Makes a 2x1 plot of the 'ooo' sound and its FFT.
ooo_fig, ooo_axis = plt.subplots(1, 2, figsize=(10, 4))

# Plots the 'ooo' sound in the time domain.
ooo_axis[0].plot(ooo_time, ooo_part)
ooo_axis[0].set_title('Amplitude vs Time')
ooo_axis[0].set_xlabel('Time (s)')
ooo_axis[0].set_ylabel('Amplitude')

# Computes the FFT of the 'ooo' sound with nfft = 240.
ooo_nfft = 240
ooo_freq = np.fft.fft(ooo_part, n = ooo_nfft)
ooo_mag_positive = np.abs(ooo_freq)[:ooo_nfft//2]

# Shifts the FFT to center the zero frequency component.
ooo_shifted = np.fft.fftshift(ooo_freq)
ooo_freq_time = np.fft.fftfreq(ooo_nfft, d=1/blue_fs)[:ooo_nfft//2]

# Plots the FFT of the 'ooo' sound in the frequency domain.
ooo_axis[1].plot(ooo_freq_time, ooo_mag_positive)
ooo_axis[1].set_title('Amplitude vs Frequency')
ooo_axis[1].set_xlabel('Frequency (Hz)')
ooo_axis[1].set_ylabel('Amplitude')

plt.tight_layout()
plt.show()

################################################################################
# Part C
################################################################################
```

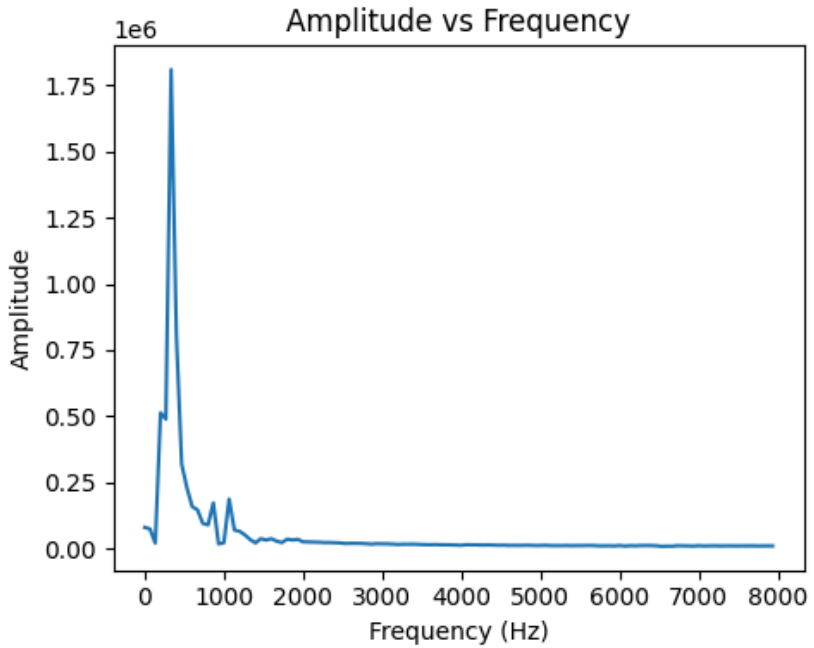
```
# Extracts a 30ms segment of the 'sss' sound and creates another time axis.
sss_part = bluenose3[int(2.565 * blue_fs):int(2.595 * blue_fs)]
sss_time = np.linspace(2.565, 2.595, num=len(sss_part))

# Makes a 2x1 plot of the 'sss' sound and its FFT.
sss_fig, sss_axis = plt.subplots(1, 2, figsize=(10, 4))

# Plots the 'sss' sound in the time domain.
sss_axis[0].plot(sss_time, sss_part)
sss_axis[0].set_title('Amplitude vs Time')
sss_axis[0].set_xlabel('Time (s)')
sss_axis[0].set_ylabel('Amplitude')

# Computes the FFT of the 'sss' sound with nfft = 480.
sss_nfft = 480
sss_freq = np.fft.fft(sss_part, n = sss_nfft)
sss_mag_positive = np.abs(sss_freq)[:sss_nfft//2]

# Shifts the FFT to center the zero frequency component.
sss_shifted = np.fft.fftshift(sss_freq)
sss_freq_time = np.fft.fftfreq(sss_nfft, d=1/blue_fs)[:sss_nfft//2]

# Plots the FFT of the 'sss' sound in the frequency domain.
sss_axis[1].plot(sss_freq_time, sss_mag_positive)
sss_axis[1].set_title('Amplitude vs Frequency')
sss_axis[1].set_xlabel('Frequency (Hz)')
sss_axis[1].set_ylabel('Amplitude')

plt.tight_layout()
plt.show()

##############################################################################
# Part D
##############################################################################

"""
Because the 'sss' sound is more complex than the 'ooo' sound, with higher
frequencies in its FFT, it requires a larger value for the nfft (and thus window
size) to provide a good and accurate frequency resolution. The 'ooo' sound is
much more periodic and has a simple frequency structure, so its nfft value can
be lower while maintaining a good frequency resolution.
"""
```

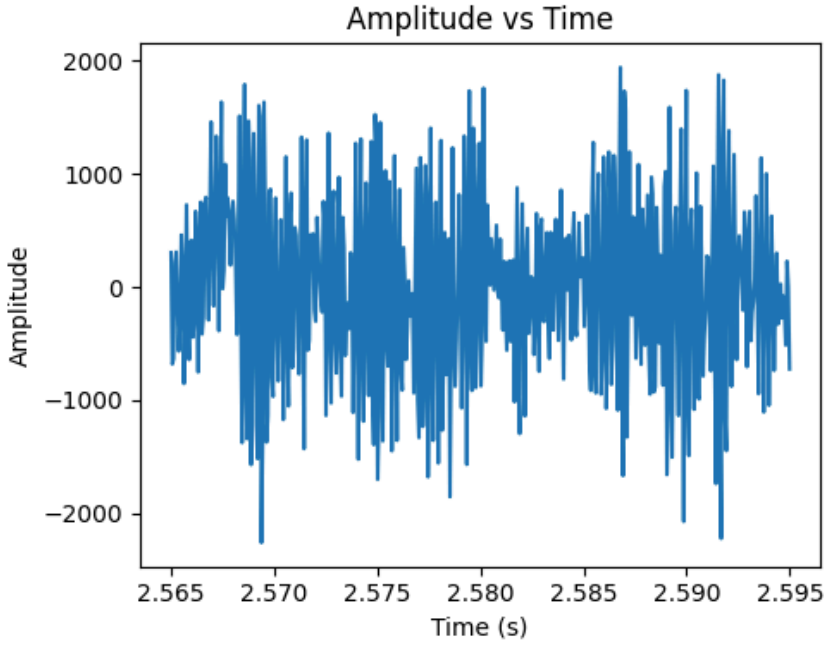
```
Sampling frequency: 16000
```

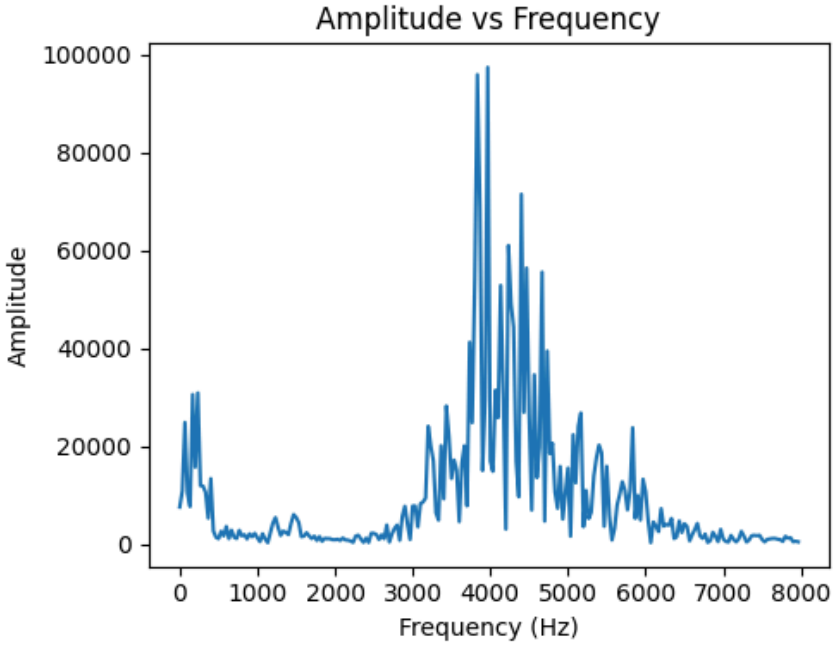


0:00 / 0:03


Amplitude vs Time
Amplitude
Time (s)
20000
15000
10000
5000
0
−5000
−10000
−15000
0.750
0.755
0.760
0.765
0.770
0.775
0.780


Amplitude vs Frequency
1e6
Amplitude
Frequency (Hz)
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
1000
2000
3000
4000
5000
6000
7000
8000


Amplitude vs Time
Amplitude
Time (s)
2000
1000
0
−1000
−2000
2.565
2.570
2.575
2.580
2.585
2.590
2.595


Amplitude vs Frequency
Amplitude
Frequency (Hz)
100000
80000
60000
40000
20000
0
0
1000
2000
3000
4000
5000
6000
7000
8000

Discussion

State what size FFT you used and explain your choice. Comment on the differences between the time and frequency plots for the two segments and the auditory differences.

Answer

For the "ooo" sound, we used a shorter window size of 240 samples because the amplitude vs time graph looks more simple and periodic. From the amplitude vs frequency graph, we can also see that the range of frequencies present in the sound is pretty small. For the "sss" sound, we used a window size of 480 samples, the full length of the 30ms audio clip, because the amplitude vs time graph looks very complex. The sound itself also covers a much broader spectrum of frequencies, as the amplitude vs frequency graph shows. Some experimentation also backs these findings. There is very little difference between an FFT with window size 240 and 480 when transforming the "ooo" sound. However, we lose much more information when doing this to the "sss" sound. Because the 30ms audio sample is already so short, we generally tried to use a window size that was as long as the audio sample, since there is very little variation in frequency composition over these short time frames. Even from an auditory perspective, it is easy to hear that the "sss" sound contains higher frequencies than the "ooo" sound, and the FFT again supports this.