EE 242 Lab 3b – Frequency Domain Representation of Signals - Fourier Transform

Anna Petrbokova, Henry Adams, Grace Hwang, Leonard Paya

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 [24]:
```
# We'll refer to this as the "import cell." Every module you import should be imported here.
%matplotlib inline
import numpy as np
import matplotlib
import scipy.signal as sig
import matplotlib.pyplot as plt

import scipy.io.wavfile as wav
# import whatever other modules you use in this lab -- there are more that you need than we've ir
```

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.

Lab 3b turn in checklist

• Lab 3b 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

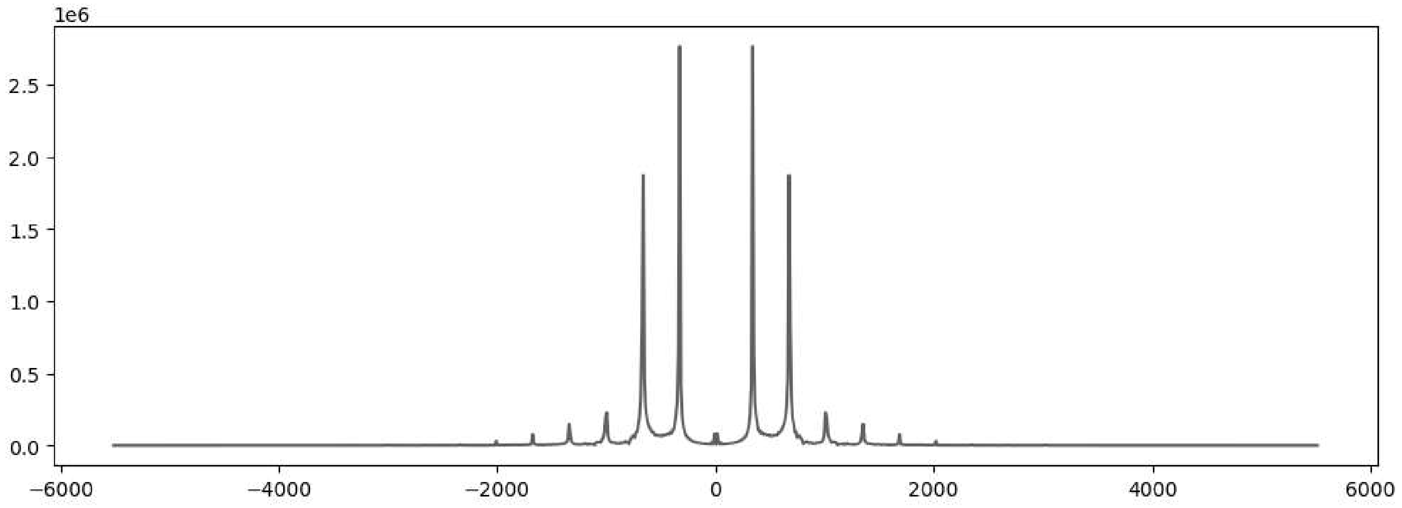
For this assignment, you will use a discrete Fourier transform (specifically, the Python implementation of an FFT) to analyze the frequency content of the 100ms segment of the horn signal (from 200ms to 300ms) from assignment 3A. Because this is a periodic signal, the frequency content will have spikes, 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.

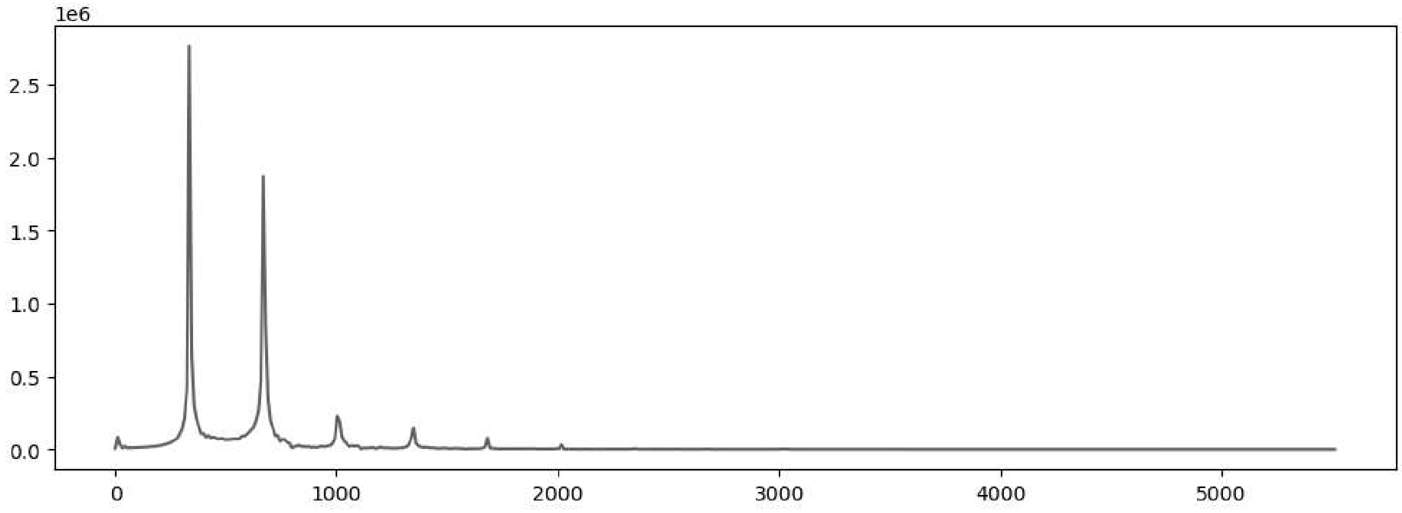
A. Use the np.fft.fft function to compute the FFT for the 100 ms horn signal, with an fft size of nfft=1024, which you can call **x_f**. Recall that the result of the FFT will be a vector that spans frequencies [0,f_s]. If this is a real-valued signal, then the first half of the FFT matters: [0,nfft/2]. **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.

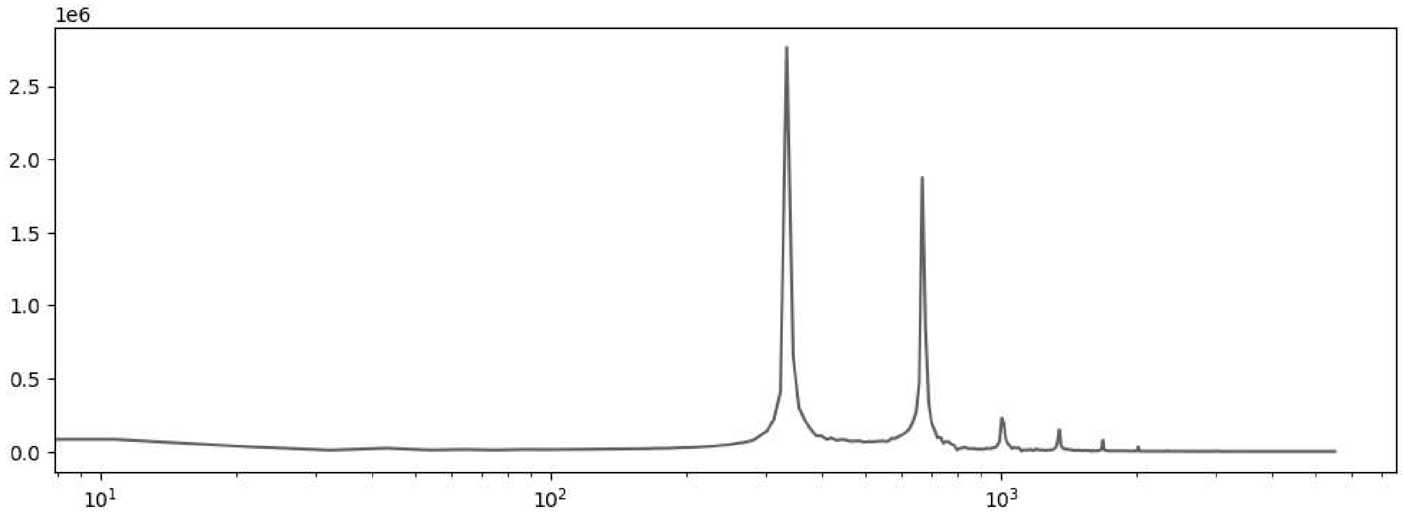
B. It is often the case that frequency content is plotted on a log scale. Plot the one-sided (positive frequency) magnitude using using a log scale.

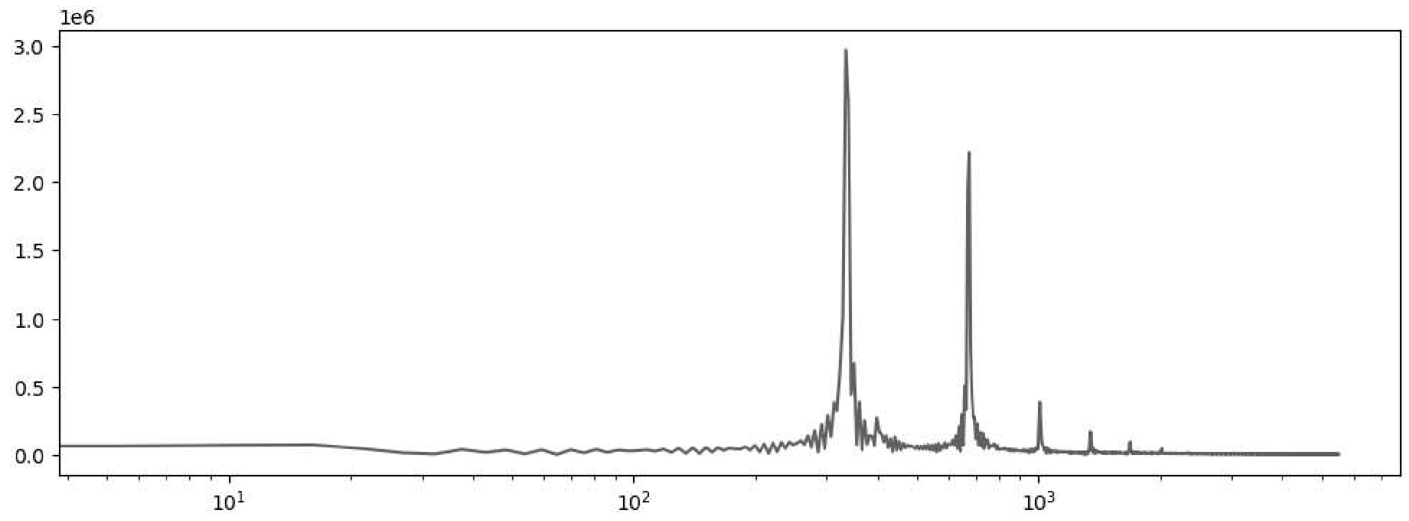
C. Changing the size of the FFT will change the frequency resolution, but it also changes the shape of the result a bit. Just as we saw with Gibbs phenomenon where increasing the number of Fourier series coefficients gave a high frequency ringing at sharp edges, increasing the FFT window will give a “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 [25]:
```
# Assignment 3 - Analyzing frequency content of a signal

fs, data = wav.read("horn11short.wav")
data=data[int((0.2)*fs):int((0.3)*fs)]
nfft=1024
x_f=np.fft.fft(data, nfft)
x_f2=np.fft.fftshift(x_f)
positive_x_f2=x_f2[int(len(x_f2)/2):]
freq=np.linspace(-fs/2,fs/2,num=nfft)
freq1=np.linspace(0,fs/2,num=int(nfft/2))
plt.figure(figsize=(12,4))
plt.plot(freq,np.abs(x_f2))
plt.figure(figsize=(12,4))
plt.plot(freq1,np.abs(positive_x_f2))


# Part A - Using python FFT function
# Use the np.fft.ftt function to find the fourier domain representation of the horn signal
# Use the np.fft.fftshift to find the shifted fft
# Plot and see the positive and negative frequencies (remember to scale the FFT index by fs/nfft
    # Seperate out the positive part into another vector
    # Plot the seperated signal

# Part B - Plot the FFT across frequency
# Repeat the process for plotting the positive frequencies
# Use the following to change your x-axis to logscale and plot in parallel ----- matplotlib.pypl
plt.figure(figsize=(12,4))
plt.plot(freq1,np.abs(positive_x_f2))
plt.xscale('log')

# Part C - Horn Signal Ringing effect.
# Do the same things as Part A but with nfft = 2048
nfft=2048
x_f=np.fft.fft(data, nfft)
x_f2=np.fft.fftshift(x_f)
positive_x_f2=x_f2[int(len(x_f2)/2):]
freq2=np.linspace(0,fs/2,num=int(nfft/2))
plt.figure(figsize=(12,4))
plt.plot(freq2,np.abs(positive_x_f2))
plt.xscale('log')
plt.show()
# Plot the the signal (positive frequencies only) and observe the additional ringing -- compare w
```









Discussion

In assignment 3, 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.

Since we are assuming the signal is perfectly periodic- the Fourier series will not correctly represent any time varying components. This includes any additional harmonic components. Also phase relationships will no tbe correct since magnitude and phase will not be correctly represented by this assumption

Assignment 4 -- Comparing frequency content of a signal

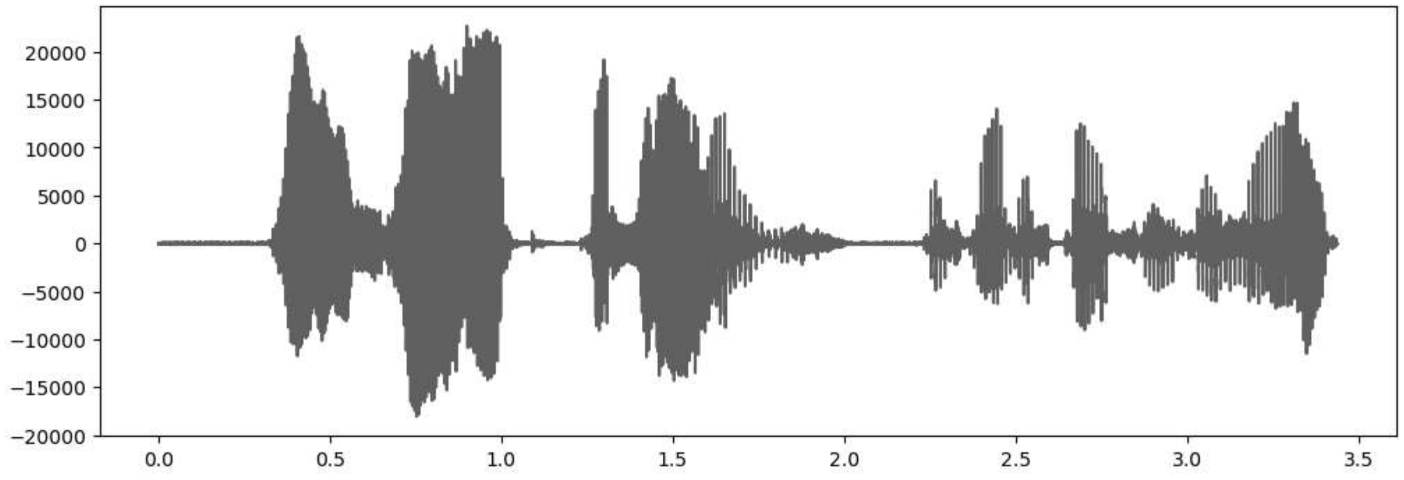
Many interesting time signals have changing frequency content. Music is one example, since different notes have different fundamental frequency. Speech is another example: we distinguish different vowels and consonants based on their frequency content. In this assignment, you will use the FFT to compare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

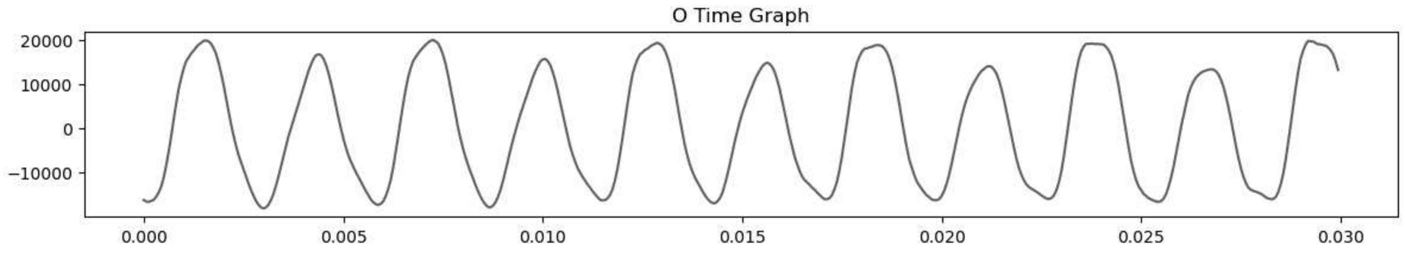
A. Download the signal "bluenose3.wav", and read in the file. Plot the full waveform, using the sampling frequency to correctly label the time axis. Play the file.

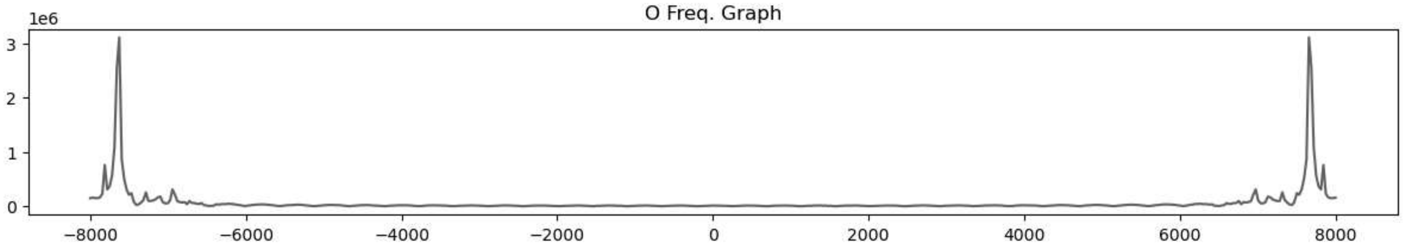
B. Extract the samples corresponding to times [0.75,0.78]. (This corresponds to the “oo” sound in the word “grew.”) Using a 2x1 plot, plot the time waveform (labeling the time axis with the specified time region) and the magnitude of the frequency response (positive frequencies only, labeling the frequency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

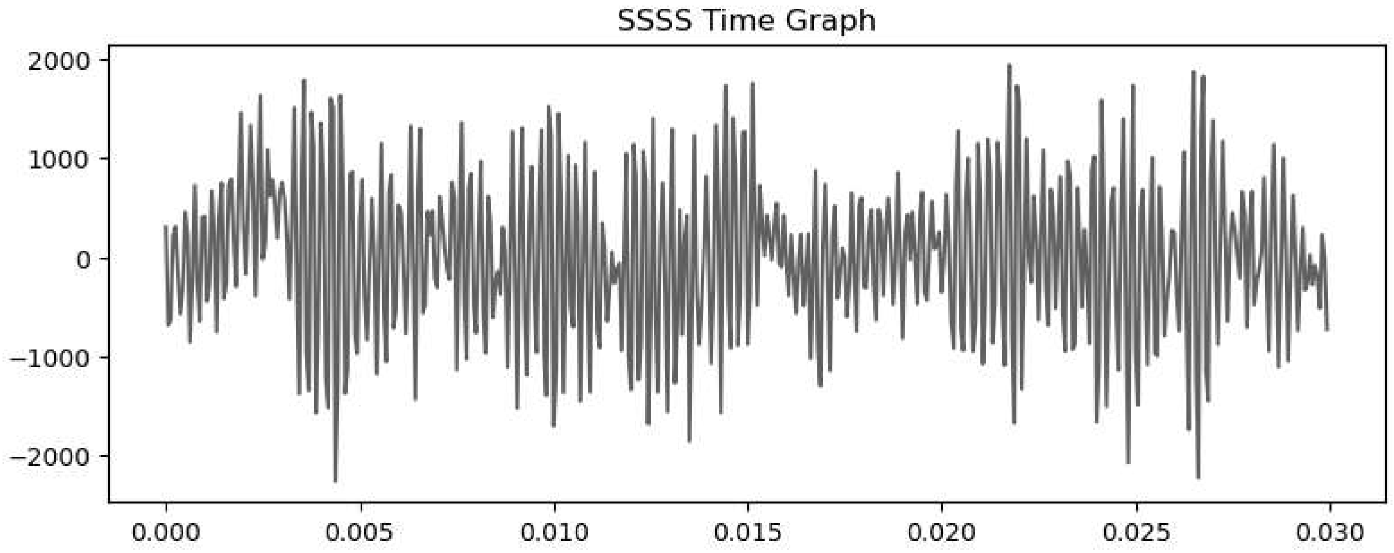
C. Repeat the exercise above using the samples corresponding to times [2.565,2.595]. (This corresponds to the “s” sound.)

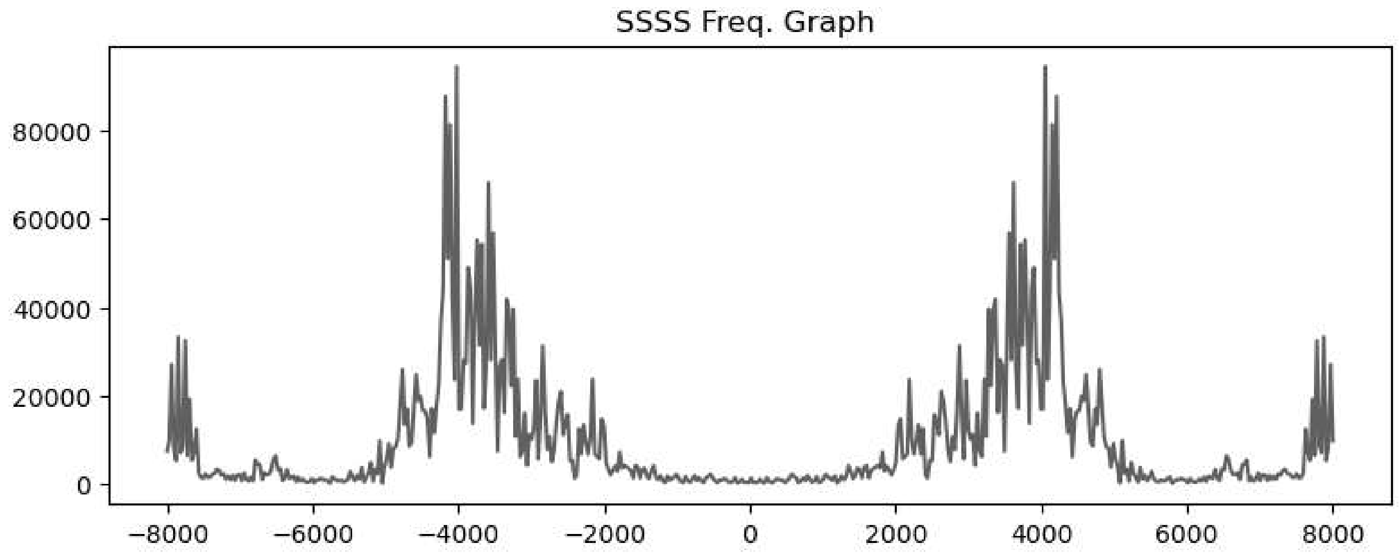
In [26]:
```
# Assignment 4 - Comparing frequency content of a signal

# Part A - Reading, plotting and playing the bluenose3.wav file
fs,data=wav.read("bluenose3.wav")
time=np.arange(len(data))/fs
plt.figure(figsize=(12,4))
plt.plot(time,data)
# Part B - Extracting ooo part of the signal
# Extract the ooo part of the signal from 0.75 seconds to 0.78 seconds
# Plot the signal in time and frequency domain
extracted_data=data[int(0.75*fs):int(0.78*fs)]
extracted_time=np.arange(len(extracted_data))/fs
plt.figure(figsize=(12,4))
plt.subplot(2,1,1)
plt.title('O Time Graph')
plt.plot(extracted_time,extracted_data)


plt.tight_layout()
plt.show()
nfft=516
x_f=np.fft.fft(extracted_data, nfft)
freq=np.linspace(-fs/2,fs/2,num=nfft)
plt.figure(figsize=(12,4))
plt.subplot(2,1,2)
plt.title('O Freq. Graph')
plt.plot(freq,np.abs(x_f))
plt.tight_layout()
plt.show()
# Part C - Extracting sss part of the signal
# Extract the sss part of the signal from 2.565 seconds to 2.595 seconds
# Plot the signal in time and frequency domain
extracted_data2=data[int(2.565*fs):int(2.595*fs)]
extracted_time2=np.arange(len(extracted_data2))/fs
plt.figure(figsize=(8,6))
plt.subplot(2,1,1)

plt.plot(extracted_time2,extracted_data2)
plt.title('SSSS Time Graph')
plt.tight_layout()
plt.show()

nfft=516
x_f=np.fft.fft(extracted_data2, nfft)
freq=np.linspace(-fs/2,fs/2,num=nfft)
plt.figure(figsize=(8,6))
plt.subplot(2,1,2)
plt.title('SSSS Freq. Graph ')
plt.plot(freq,np.abs(x_f))
plt.tight_layout()
plt.show()
# Part D - Justify the difference and choice of nfft used.
'''i just wanted a less wiggly(?) graph'''
```











```
Out[26]: 'i just wanted a less wiggly(?) graph'
```

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.

We used a size of 516 for our FFT as any higher as when we went higher there was no great improvement in quality and any lower we started to see great changes in the frequency qaulity. For these segements, the O time domain graph has a sinusoid with a lower freqency, while the S time domain has a considerably higher frequency with alot more changes in magnitude. This matched their frequency plots in which the O frequency graph has spikes near 8000 / -8000, while the ssss freq. graph has frequency spikes from 2k - 4k, and a spike at around 8k.