Signals and Systems Fundamentals

[ECE251s]

Course Project

Analyzing an Audio Signal Using MATLAB

Name: Waseem Sameh Makram Faheem

ID: 2001194

BN/SN/MJ: 145/3/ECE

Theoretical Background and Introduction:

Digital Signal Processing is a Brach of Science and Engineering Which Specializes in Analyzing Data Provided to Processors from Other Sources: like Sensors, User IOs, or Preexisting Files.

Signal Processing is Present in Every Field of Our Daily Life Either in Telecommunications, Multimedia Processing, or Medical Equipment.

The Study of Signal Processing is Corelated with the Study of the Fourier Analysis, which is a Set of Mathematical Operations Meant to Decompose Complex Signals into Simple Harmonics and Wavelets.

One of the Most Notable Applications of the Fourier Analysis is the MRI Scanner, which Receives Frequency–Amplitude Signals Through its Sensors and Concert it into a Space–Amplitude Image of the Scanned Specimen.

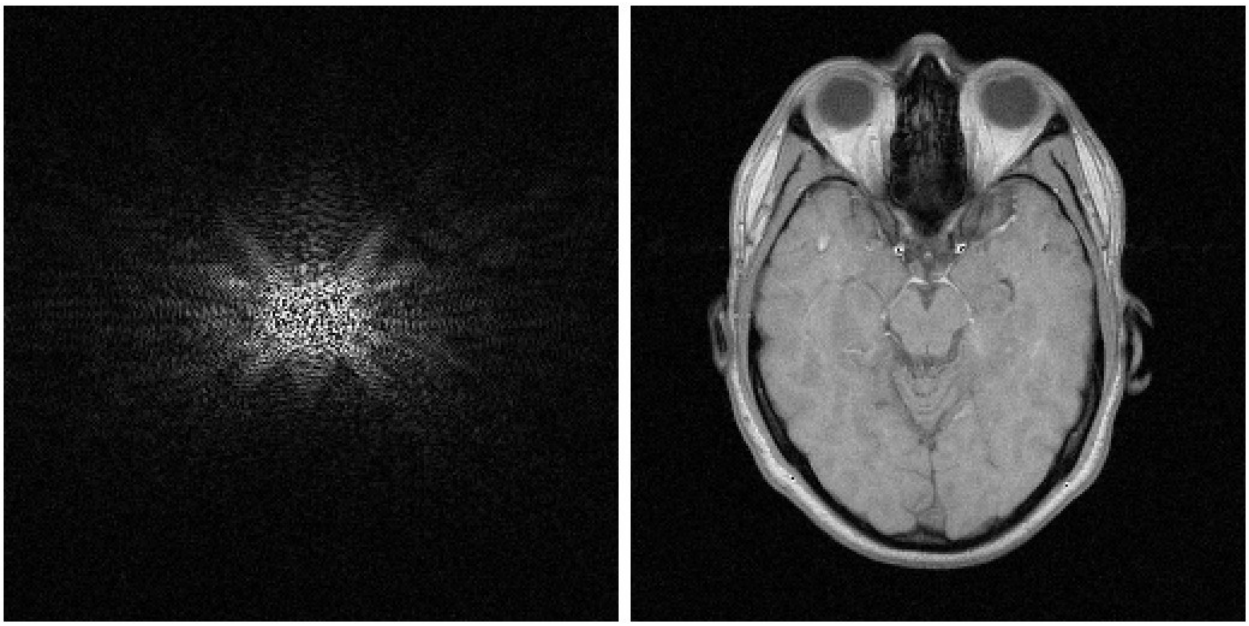


Figure 1 - The Signals Received by the MRI Scanner (Left), and the Image Displayed after Applying an IFT (Right)

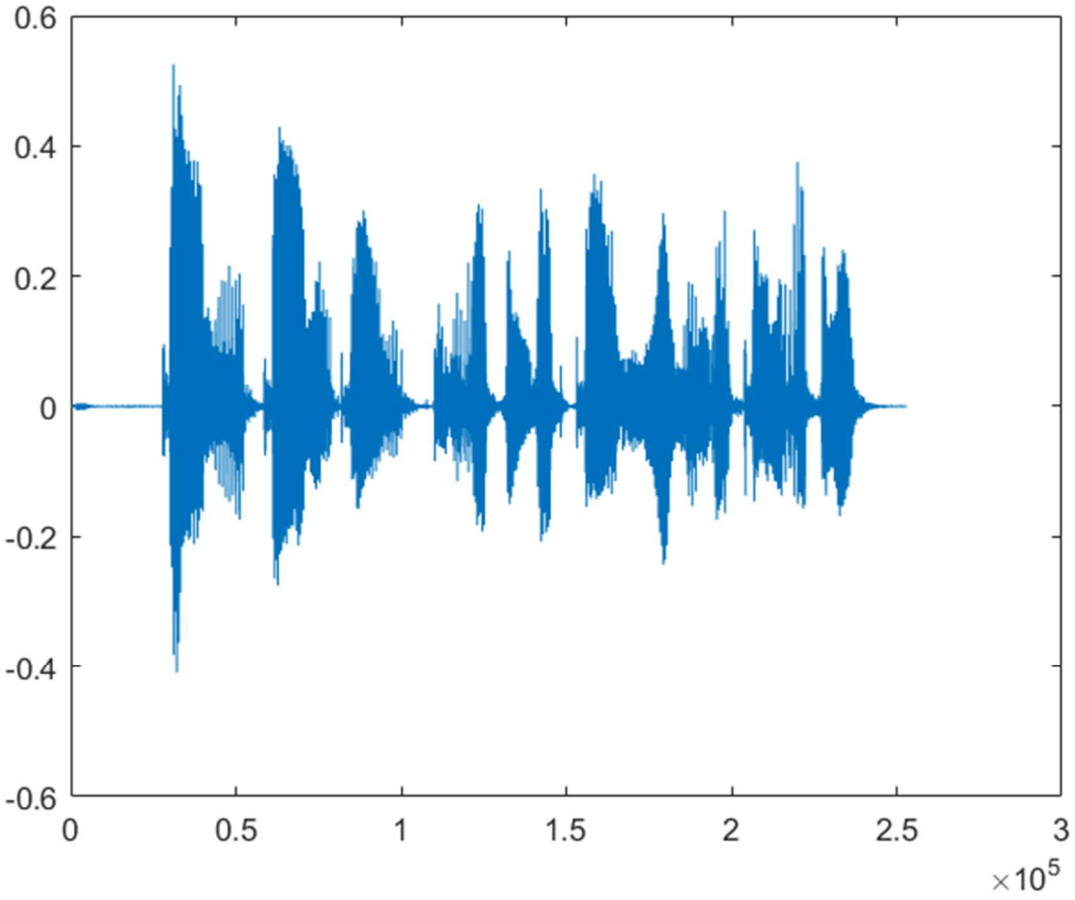
In this Project, we Will Explore the Applications of Digital Signal Processing, Analyzing and Audio File that we Provided.

Before Starting With the Project, Some Notes Should Be Said:

- The Project Was Coded on the MATLAB Live Script Tool
- The Audio File Was Recorded on a Mobile Phone on a Mono (Not Stereo) Mode and Converted from a .m4a to a .wav file to be compatible with MATLAB.

Code Blocks and Description:

```
[Sig, fs] = audioread("Final - Mono.wav");
plot(Sig)
```



To Load the Audio File in MATLAB, the **audioread()** Function was Used to Represent the Signal as a 2D Array.

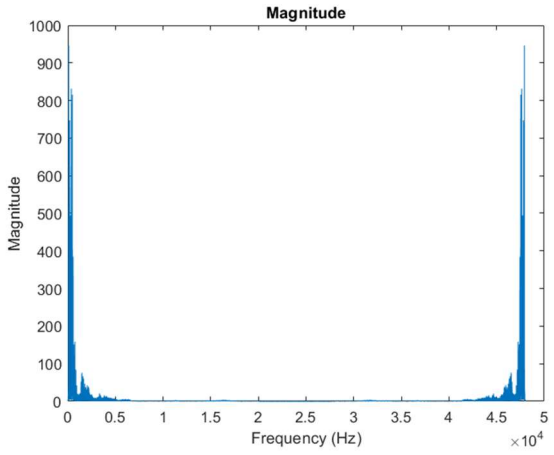
We Then Use the **plot()** Function to Plot the Values of Sig (Amplitude) Versus Time (Represented as Sample Times).

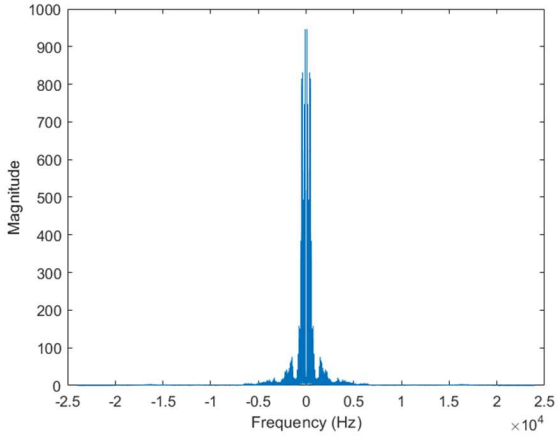
To Represent the Signal in the Frequency Domain, one of two approaches can be taken: Either to represent the Power–Frequency Curve of the Signal Using the **pspectrum()** Function, or, By Using the FFT() Function to Get the Magnitude–Frequency Curve of the Signal.

The **pspectrum()** approach is good at Identifying the Bandwidth of the Signal, However, it not a Very Accurate Representation as the Power of a Signal is Relative to the to the Square of the Amplitude: $P = |A|^2$.

```
SigFt = fft(Sig);
f = (0:length(SigFt)-1)*fs/length(SigFt);
plot(f,abs(SigFt))
xlabel('Frequency (Hz)')
ylabel('Magnitude')
title('Magnitude')
```

```
n = length(Sig);
fshift = (-n/2:n/2-1)*(fs/n);
yshift = fftshift(SigFt);
plot(fshift,abs(yshift))
xlabel('Frequency (Hz)')
ylabel('Magnitude')
```

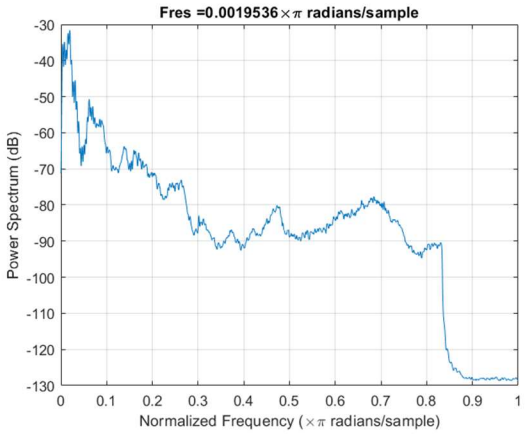




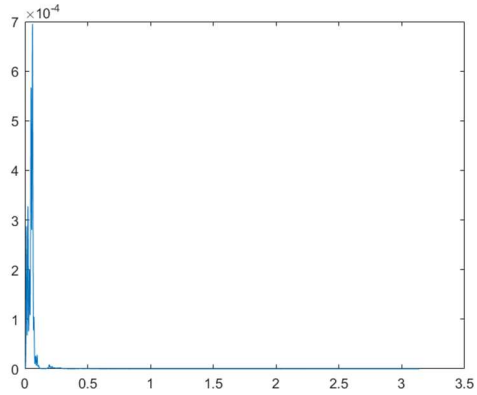
Using the **fft()** us a mirrored distribution of amplitude, which when shifted, provides the amplitude for positive and negative harmonics (+ and – harmonics represent phase of that harmonic).

By zooming in on the middle peak, we find that it nears zero between 600 and 700 Hz, so we can primarily say that out bandwidth is 650 Hz (It'll be accurately measured later).

```
pspectrum(Sig)
```




```
[p,f] = pspectrum(Sig);
plot(f, p)
```

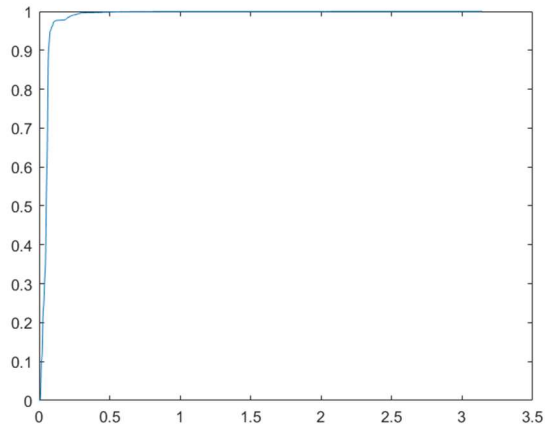


Using the pspectrum() function then plotting its values on a linear (Watt) instead of a logarithmic (dB) scale, gives a curve near that of the fft() function.

To get the bandwidth, we need to know the value of the frequency band, below which, 90% of the power of the Signal is delivered.

To do this, we need to take a cumulative sum of the power using the cumsum() function (as Integration), normalize the resulting curve, then know the frequency at which the normalized curve equals 0.9.

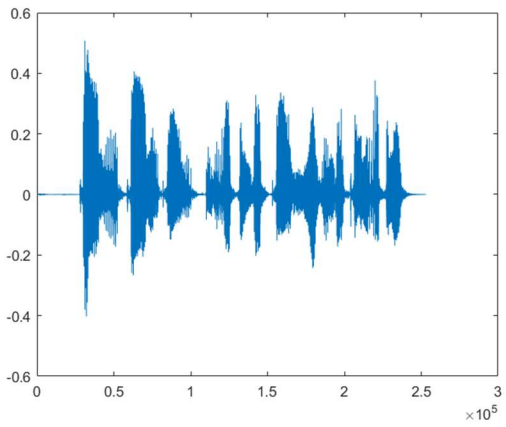
```
[p,f] = pspectrum(Sig);
cp = cumsum(p);
cpn = cp/cp(end);
plot(f, cpn)
```



```
fc = (0.066)*(10^4)
```

```
fc = 660
```

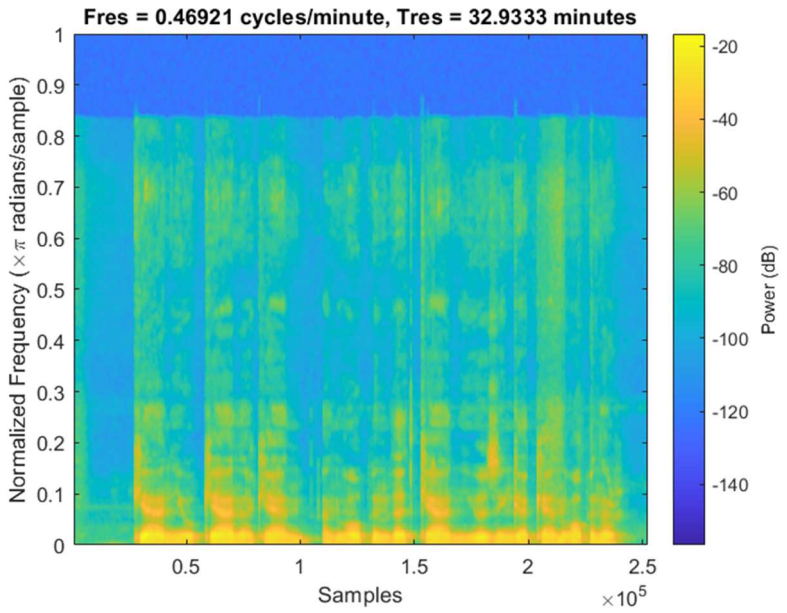
```
filtered = lowpass(Sig, 660, fs);
plot(filtered)
```



From Our Plots, we get that the band frequency is equal to 660 Hz (Between 600 and 700 as predicted). To Confirm this, the signal was almost the same after being filtered from frequencies above 660 Hz.

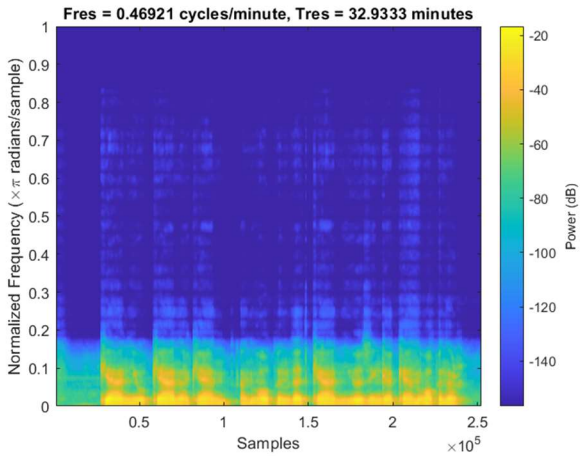
To plot the spectrogram, the pspectrum() is used again with an additional argument.

```
pspectrum(Sig, "spectrogram")
```

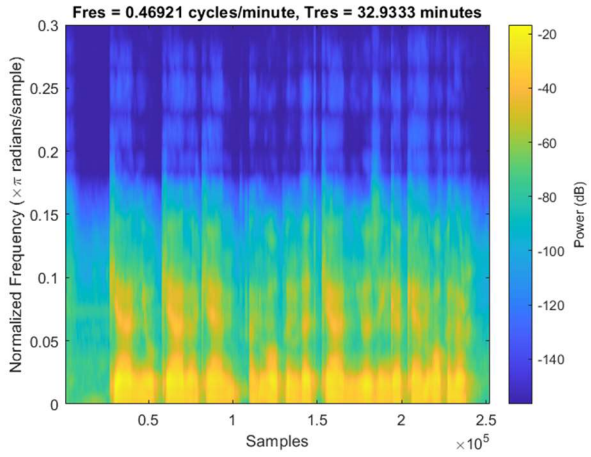


The Signal Can be Represented Cleanly after Conditioning and Filtering.

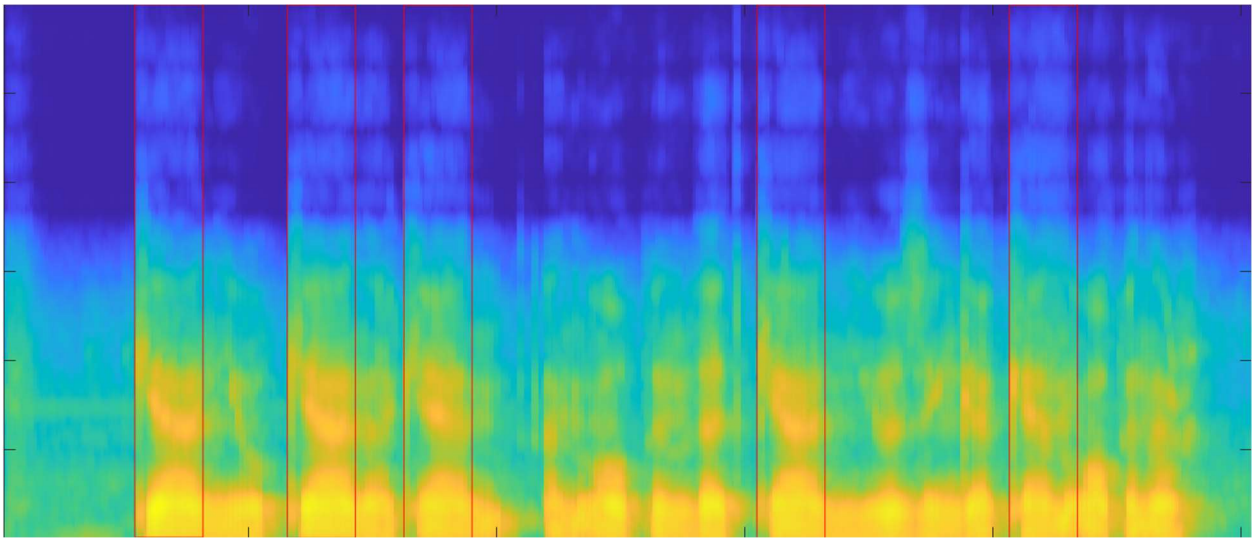
```
pspectrum(filtered, "spectrogram")
```




```
pspectrum(filtered, "spectrogram", "FrequencyLimits", [0 0.3*pi])
```



We Can now use the spectrogram to find "Can" Words.



Notice that since the first word of the sentence is “Can”, we can use it spectrogram representation to find other iterations of “Can”.

Notice that The Shape of the Spectrogram takes the shape of a sand watch with a notch and a certain width, a distinctive feature for this signal, making it easy to be detected (Marked by red Rectangles).

To implement an algorithm to detect “Can” words we can use the mathematical operation of dot products, the dot product is responsible to say how much two matrices corelate to each other.

So if we consider the “Can” Spectrogram sample we just extracted as a 2D matrix, and measure the correlation between it and each region of the spectrogram matrix, we should be able to find words with high correlation factors, i.e. “Can” s.