# (ECE251) SIGNALS & SYSTEMS

## PROJECT DOCUMENTATION


## Team members:

- Shehab Mahmoud Salah (2100320)
- Abdelrahman Hany Mohamed (2100627)
- Youssef Ahmed Mohamed (2101006)
- Omar Mamon Hamed (2100767)
- Seif Eldeen Ahmed Abdulaziz (2100339)


In the given project we are assigned to develop a code that applies filtering to an audio file on the computer's disk, the signal should be plotted in **time-domain**, and **frequency domain** as well for both input and output signals.

*The link to the python script is included in this documentation (Github Gist), the Jupyter Documentation is also available in the following pages*.

## OVERVIEW & DELIVERABLES

**The aim of this project is to develop a code that does the following:**

- Read an audio file from the hard disk of the computer
- Plot the audio signal in time domain
- Find the frequency domain representation of the signal and plot it
- Apply a filter (either a LPF or HPF) to change the original frequency components of the audio signal, then plot the filtered signal in frequency domain
- Find the corresponding signal in time domain for the filtered signal and plot it
- Save the filtered signal in time domain as an audio file on the hard disk (of the same format as the original one)

## DEVELOPED SOLUTION

The **Python script** can be found on the following URL:
https://gist.github.com/dizzydroid/821d050e5f3e13f273d5a0d626c6f92e

You can also find the **entire project**, with all source code on **the project's repo**:
https://github.com/dizzydroid/ASU_DigitalAudioFilteringPrjct

The **Jupyter Documentation** is also available in the *following pages* with all plots included.

A **shared video presentation** of the filtering in action is also available:
https://drive.google.com/file/d/1vQqxQfh-VZtb13e_OdOY2Y0vkM43Hh4M/view?usp=sharing

SciPy Documentation & `butter` function usage:
https://docs.scipy.org/
https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.butter.html

# (ECE251) Digital Audio Filtering

December 29, 2023

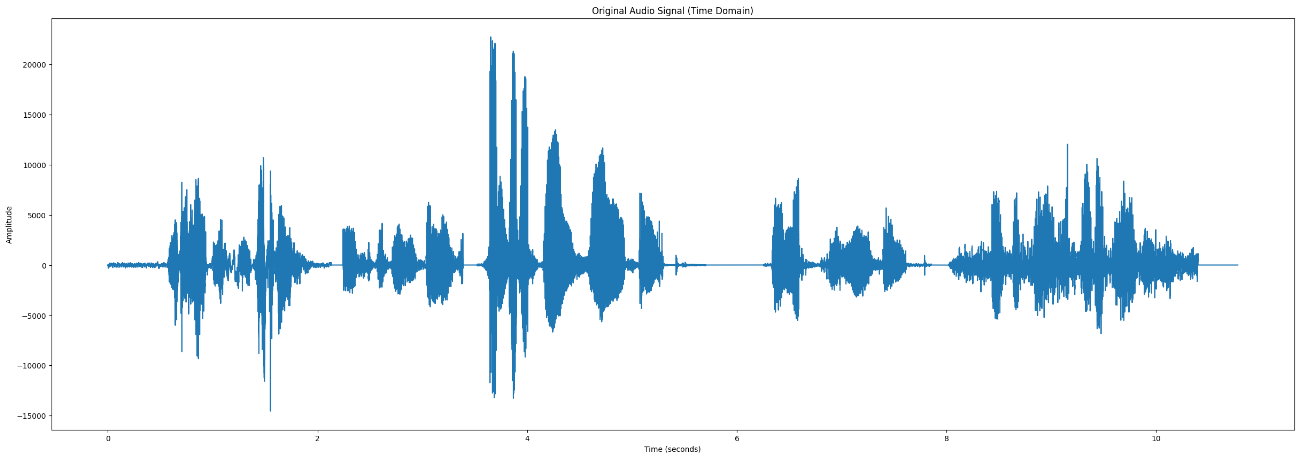
## 1 ECE251: Signals and Systems Project

#### 1.0.1 Team members:

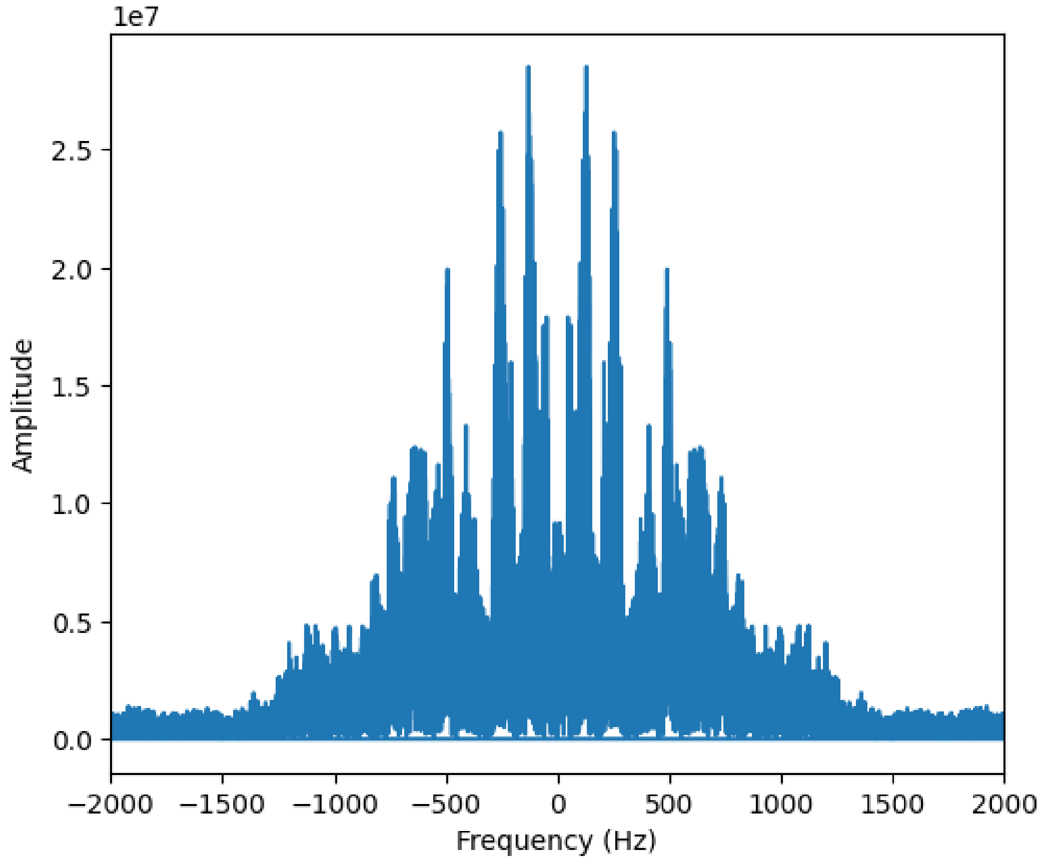

- **Shehab Mahmoud Salah** *2100320*
- **Abdelrahman Hany Mohamed** *2100627*
- **Youssef Ahmed Mohamed** *2101006*
- **Omar Mamon Hamed** *2100767*
- **Seif Eldeen Ahmed Abdulaziz** *2100339*


[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import butter, lfilter
from scipy.io import wavfile
import scipy.fft as fft

# Define file path and filter parameters
input_file='input.wav'
fs, data = wavfile.read(input_file) # sampling rate == f_s & data == x
data = np.mean(data, axis=1) # convert stereo to mono
cutoff = 400 # Hz (adjust for desired filter effect)
order = 5 # filter order -> to be used in butter()
filter_type='lowpass' # can be "lowpass" or "highpass"
```

[ ]:
```
# Create a time array (in seconds)
t = np.linspace(0, len(data) / fs, num=len(data))

# Plot time domain signal
plt.figure(figsize=(30, 10))
plt.plot(t, data)
plt.xlabel("Time (seconds)")
plt.ylabel("Amplitude")
plt.title("Original Audio Signal (Time Domain)")
plt.show()
```



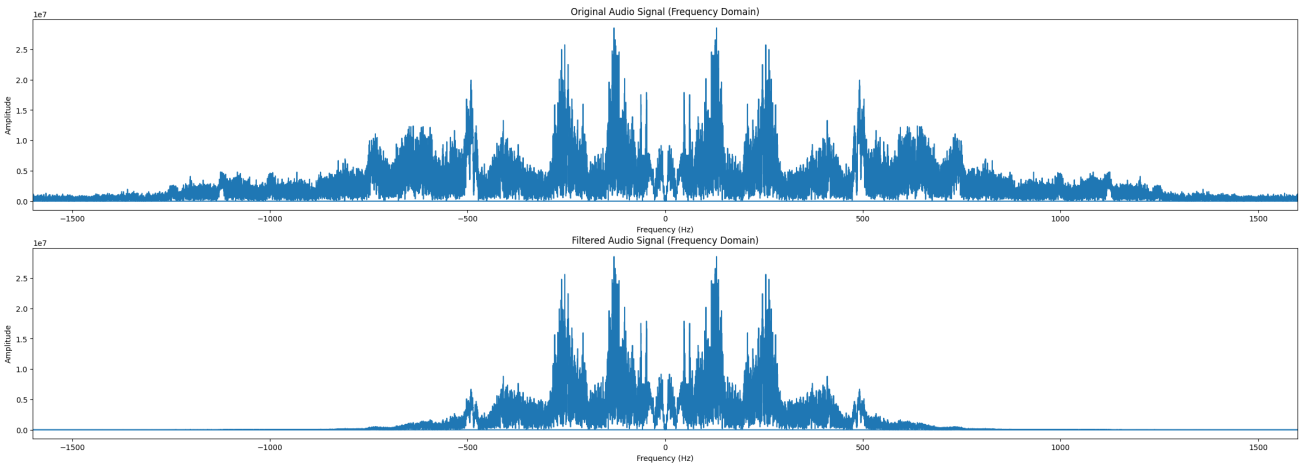

[ ]:
```python
# Calculate the FFT value of frequency spectrum
transformed_signal = fft.fft(data) # get fft of data

# Compute the frequencies
frequencies = np.fft.fftfreq(len(transformed_signal), 1/fs)

# Plot the absolute value of the FFT
plt.figure(figsize=(6, 5))
plt.plot(frequencies, np.abs(transformed_signal))
plt.xlim(-2000,2000)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.show()
```



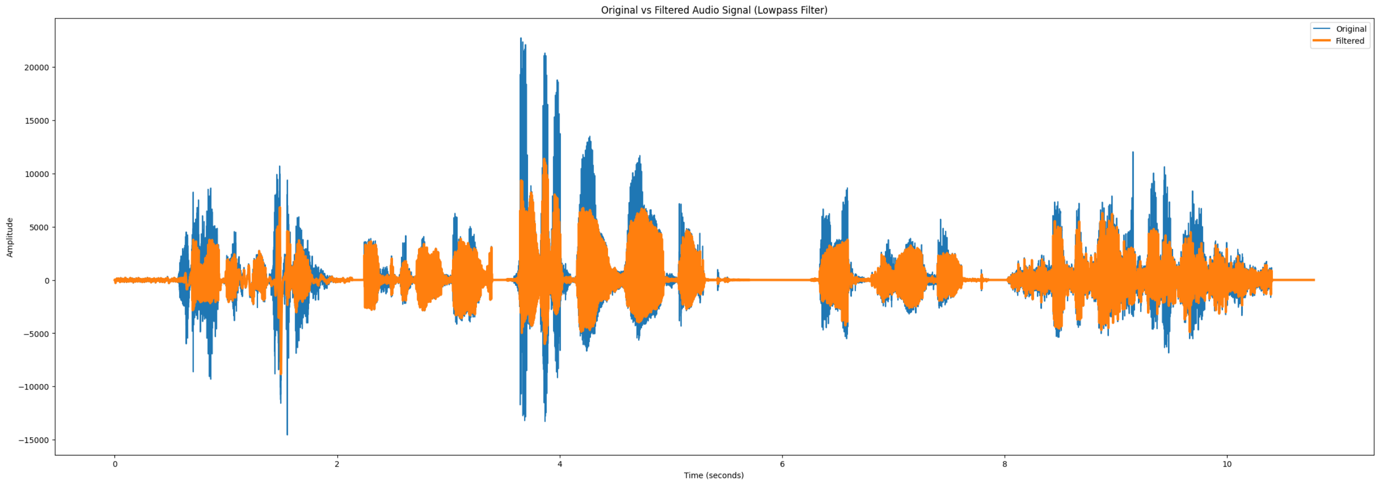

[ ]:

```
# Design and apply filter (can be either HPF or LPF)
b, a = butter(N=order,Wn= cutoff, btype=filter_type, analog=False, fs = fs) #␣
 ↪5th order filter , b == numerator, a == denominator
filtered_signal_time = lfilter(b, a, data) # apply filter to fft data (time␣
 ↪domain)
filtered_signal = fft.fft(filtered_signal_time) # get fft of filtered data␣
 ↪(frequency domain)

# Compute the frequencies
frequencies = np.fft.fftfreq(len(filtered_signal), 1/fs)

# show the original and filtered signals in the frequency domain:
plt.figure(figsize=(30, 10))
plt.subplot(2, 1, 1)
plt.plot(frequencies, np.abs(transformed_signal))
plt.xlim(-cutoff*4,cutoff*4)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.title("Original Audio Signal (Frequency Domain)")
```

```
plt.subplot(2, 1, 2)
plt.plot(frequencies, np.abs(filtered_signal))
plt.xlim(-cutoff*4,cutoff*4)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.title("Filtered Audio Signal (Frequency Domain)")
plt.show()
```




[ ]:
```
# Convert filtered FFT back to time domain and save
filtered_data = filtered_signal_time # filtered data in time domain

# Create a time array (in seconds)
t = np.linspace(0, len(filtered_data) / fs, num=len(filtered_data))

# show the original and filtered signals in time domain:
plt.figure(figsize=(30, 10))
plt.plot(t, data, label='Original')
plt.plot(t, filtered_data, linewidth=3, label='Filtered')
plt.xlabel("Time (seconds)")
plt.ylabel("Amplitude")
plt.title(f"Original vs Filtered Audio Signal ({filter_type.capitalize()} Filter)")
plt.legend()
plt.show()
```




[ ]:
```
# Normalize and save audio
wavfile.write('filtered_audio.wav', fs, filtered_data.astype(np.int16)) #␣
↪normalize and save audio
# can also normalize using: filtered_data = filtered_data / np.max(np.
↪abs(filtered_data))

print("Filtered audio saved as 'filtered_audio.wav'!")
```

```
Filtered audio saved as 'filtered_audio.wav'!
```