21113-lab05

March 10, 2024

0.0.1 BL.EN.U4AIE21113

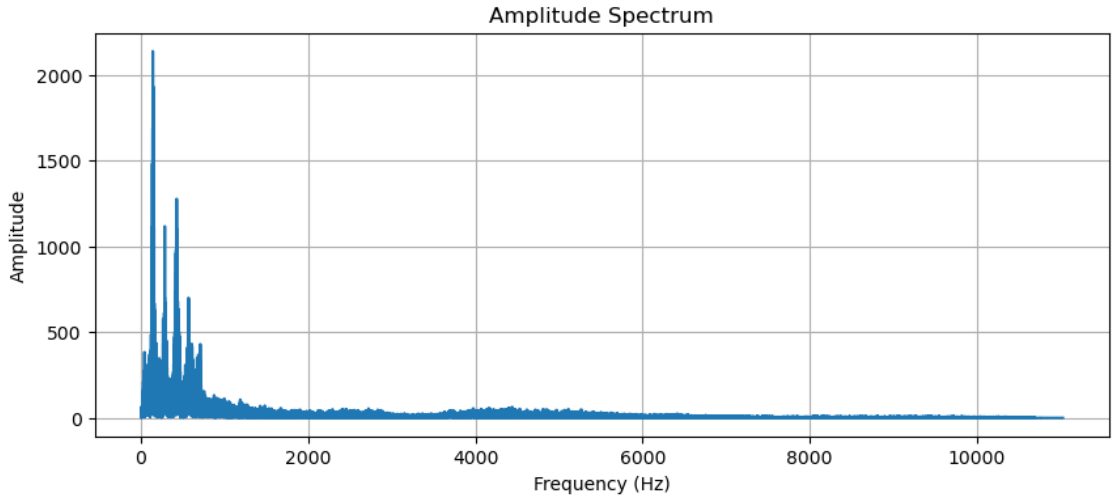
0.0.2 Sai Subhash M

0.0.3 Lab05

0.0.4 A1.

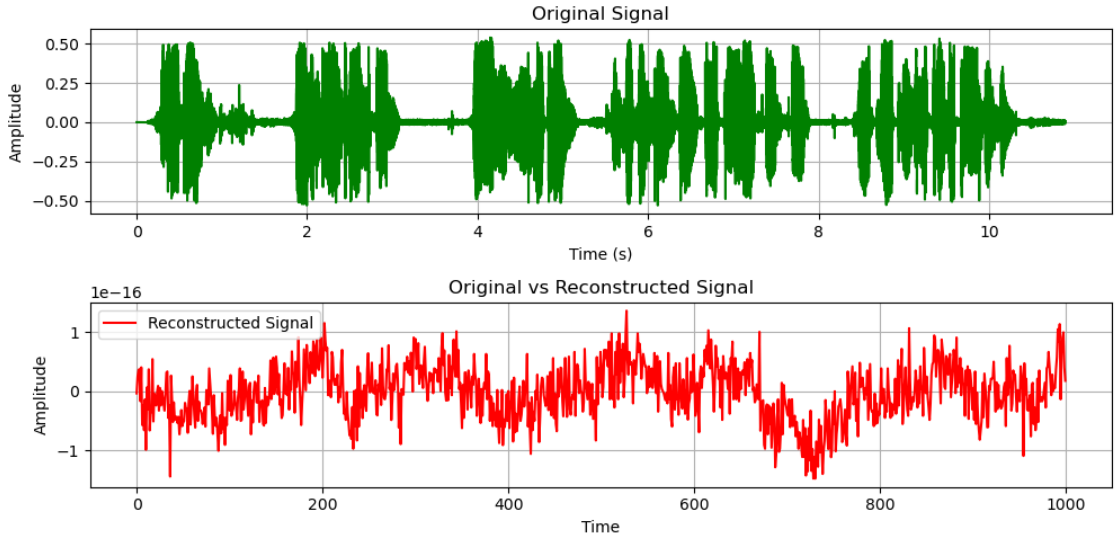
Use numpy.fft.fft() to transform the speech signal to its spectral domain. Please plot the amplitude part of the spectral components and observe it. Use numpy.fft.ifft() to inverse transform the frequency spectrum to time domain signal

[44]:
```
import numpy as np
import matplotlib.pyplot as plt
import librosa

# Load the audio file
audio_file_lab5 = "Subbu.mp3"
y_new, sr_new = librosa.load(audio_file_lab5)

y_new = np.fft.fft(y_new)

plt.figure(figsize=(10, 4))
freqs_new = np.fft.fftfreq(len(y_new), 1/sr_new)
plt.plot(freqs_new[:len(freqs_new)//2], np.abs(y_new)[:len(freqs_new)//2])
plt.title('Amplitude Spectrum')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.grid(True)
plt.show()
```



[30]:
```
import numpy as np
import matplotlib.pyplot as plt
import librosa

# Load the audio file
audio_file_lab5 = "Subbu.mp3"
y_new, sr_new = librosa.load(audio_file_lab5, sr=None, mono=True, dtype=np.
 ↪float32)
y = np.fft.fft(y_new)
plt.figure(figsize=(10, 5))

# Plot the waveform of the original signal
plt.subplot(2, 1, 1)
time = np.arange(0, len(y_new)) / sr_new
plt.plot(time, y_new, color='green')
plt.title('Original Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.grid(True)

# Compute inverse FFT
y_inv_new = np.fft.ifft(Y_new)

# Plot the reconstructed signal
plt.subplot(2, 1, 2)
plt.plot(np.real(y_inv_new[:1000]), label='Reconstructed Signal', color='red')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.title('Original vs Reconstructed Signal')
plt.legend()
```

```
plt.grid(True)

plt.tight_layout()
plt.show()
```

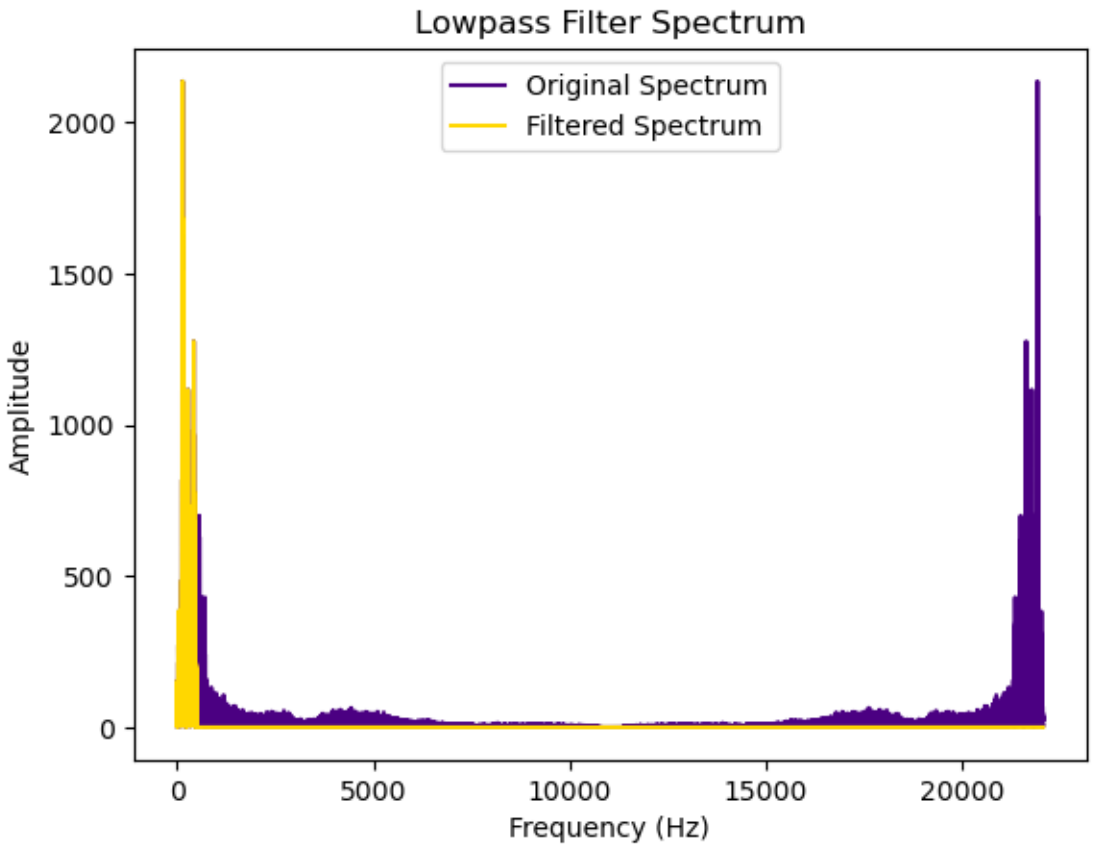


0.0.5 A2.

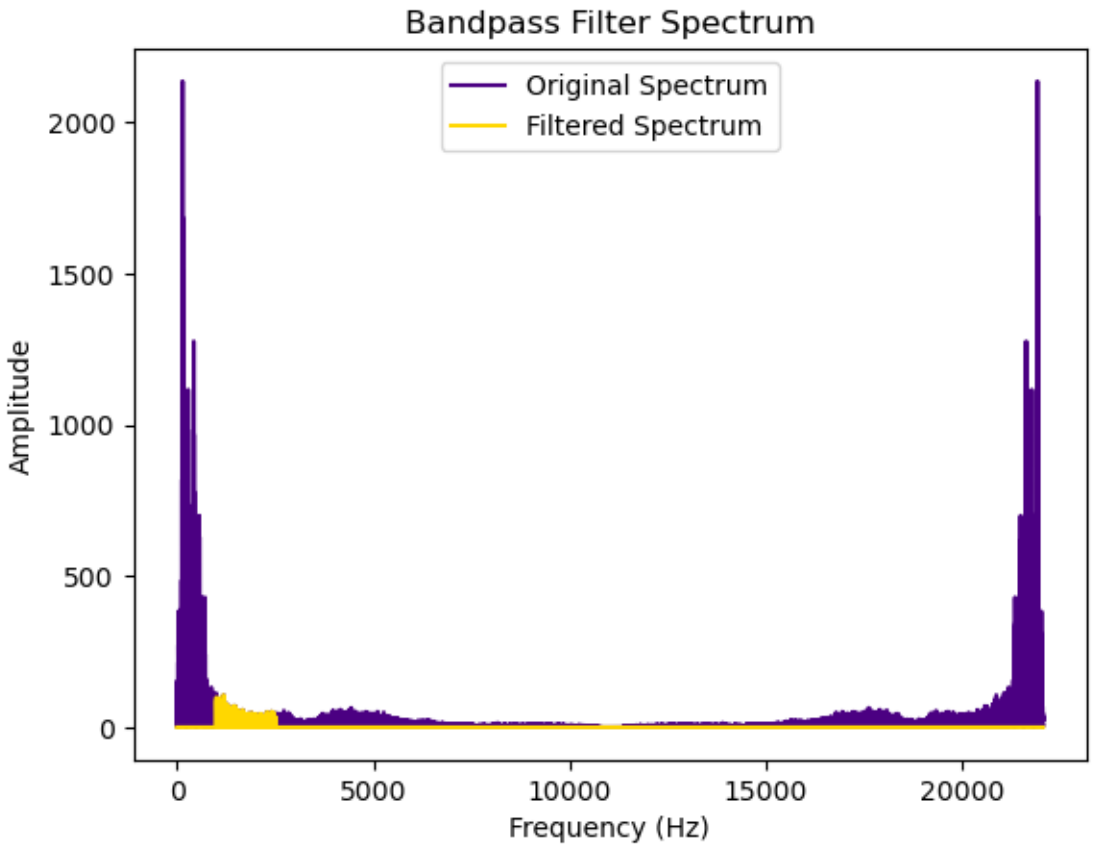
Use a rectangular window to select the low frequency components from your spectrum. Inverse transform the filtered spectrum and listen to this sound. Repeat the same for band pass and high pass frequencies of spectrum.

[42]:
```
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import Audio as IAudio

speech_custom, sr_custom = librosa.load("Subbu.mp3")

fft_custom = np.fft.fft(speech_custom)
length_custom = len(fft_custom)

def custom_filter_and_play(filter_type_custom, cutoff_custom=None):
    window_custom = np.ones(length_custom)
    if filter_type_custom == "lowpass":
        window_custom[int(cutoff_custom[0] * length_custom / sr_custom):] = 0
    elif filter_type_custom == "highpass":
        window_custom[:int(cutoff_custom[0] * length_custom / sr_custom)] = 0
    elif filter_type_custom == "bandpass":
        window_custom[:int(cutoff_custom[0] * length_custom / sr_custom)] = 0
        window_custom[int(cutoff_custom[1] * length_custom / sr_custom):] = 0
```

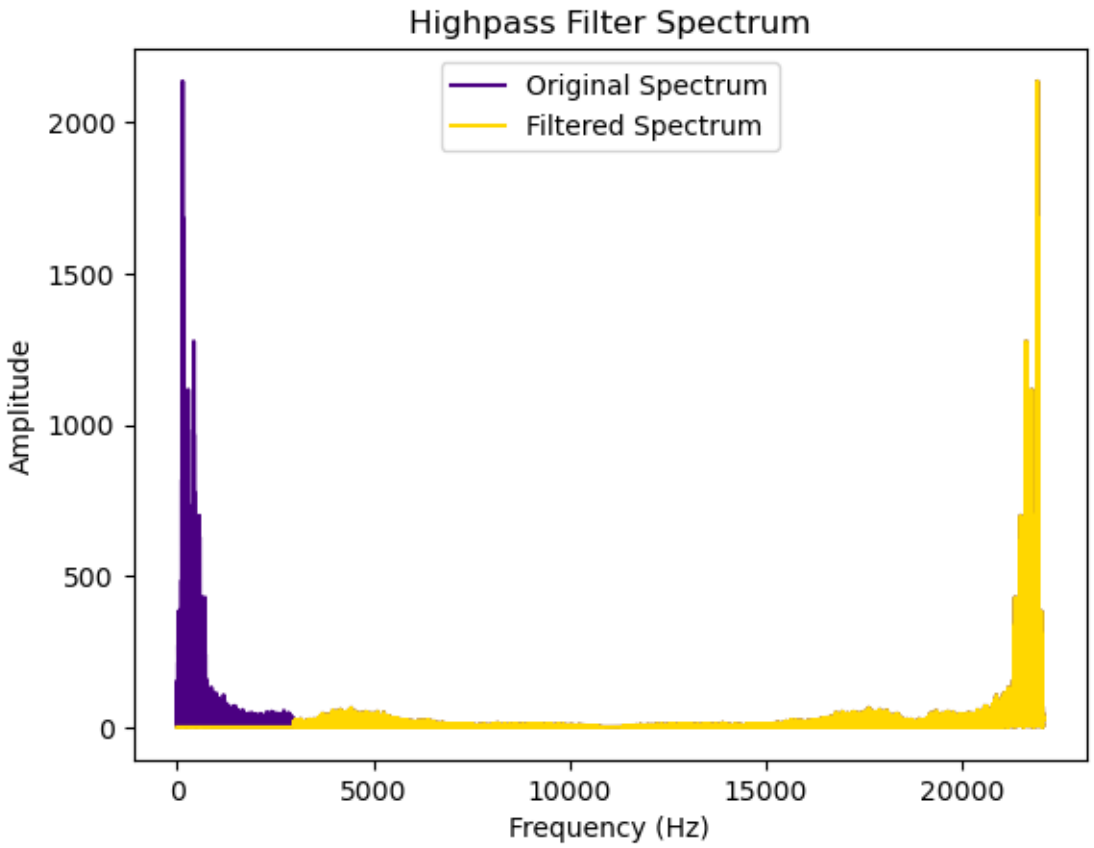
```
    else:
        print(f"Invalid filter type: {filter_type_custom}")
        return None

    filtered_custom = fft_custom * window_custom

    freqs_custom = np.arange(0, sr_custom, sr_custom / length_custom)
    plt.plot(freqs_custom, np.abs(fft_custom), label="Original Spectrum",␣
 ↪color='indigo')
    plt.plot(freqs_custom, np.abs(filtered_custom), label="Filtered Spectrum",␣
 ↪color='gold')
    plt.xlabel("Frequency (Hz)")
    plt.ylabel("Amplitude")
    plt.title(f"{filter_type_custom.capitalize()} Filter Spectrum")
    plt.legend()
    plt.show()

    filtered_audio_custom = np.real(np.fft.ifft(filtered_custom))

    display(IAudio(data=filtered_audio_custom, rate=sr_custom))

    return filtered_audio_custom


filtered_low_custom = custom_filter_and_play("lowpass", cutoff_custom=(500,))
filtered_bandpass_custom = custom_filter_and_play("bandpass",␣
 ↪cutoff_custom=(1000, 2500))
filtered_high_custom = custom_filter_and_play("highpass", cutoff_custom=(3000,))
```




```
<IPython.lib.display.Audio object>
```




```
<IPython.lib.display.Audio object>
```



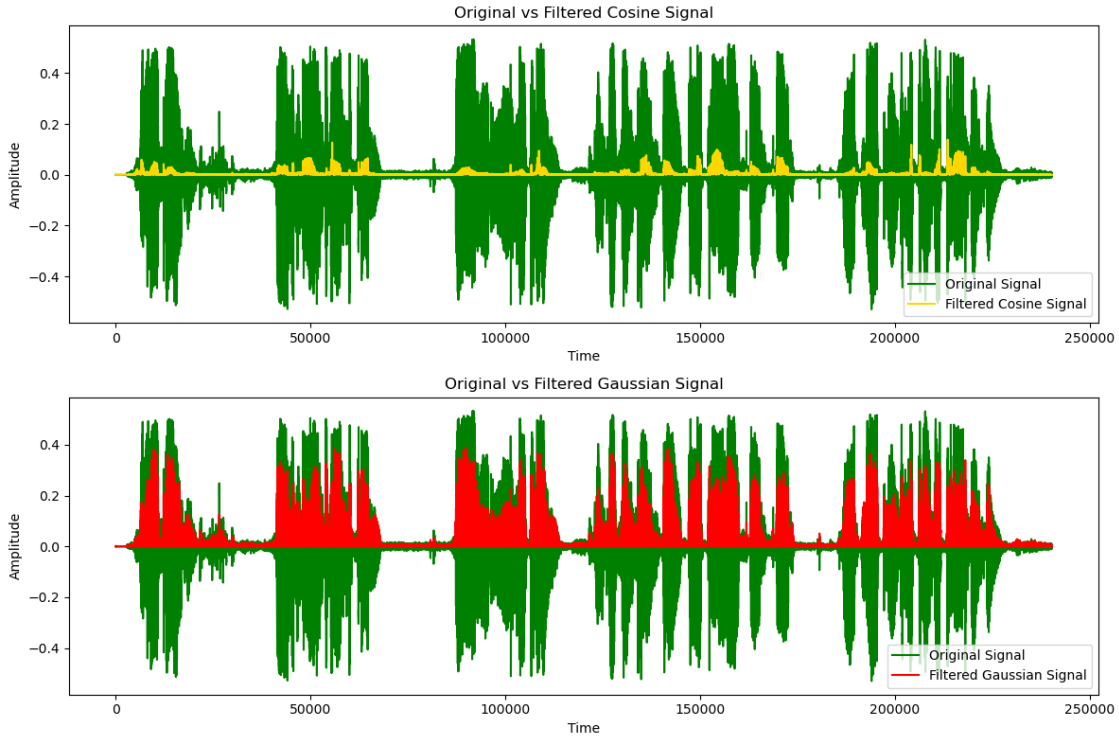

```
<IPython.lib.display.Audio object>
```

0.0.6 A2.

Repeat A2 with other filter types such as Cosine / Gausian filters.

[61]:
```
import numpy as np
import librosa
import matplotlib.pyplot as plt

signal, sampling_rate = librosa.load('Subbu.mp3')
fourier_transform = np.fft.fft(signal)
frequency_axis = np.fft.fftfreq(len(fourier_transform), 1/sampling_rate)


def custom_cosine_filter(freqs, center, band):
    return np.exp(-0.5 * ((freqs - center) / band) ** 2)

def custom_gaussian_filter(freqs, center, std):
    return np.cos(np.pi * (freqs - center) / (2 * std)) ** 2
```

```
center_frequency = 1000
bandwidth = 200
standard_deviation = 100

cosine_mask = custom_filter(frequency_axis, center_frequency, bandwidth)
gaussian_mask = custom_gaussian(frequency_axis, center_frequency,␣
 ↪standard_deviation)

filtered_cosine = fourier_transform * custom_mask
filtered_gaussian = fourier_transform * gaussian_mask

cosine_signal = np.fft.ifft(filtered_custom)
gaussian_signal = np.fft.ifft(filtered_gaussian)


plt.figure(figsize=(12, 8))

plt.subplot(2, 1, 1)
plt.plot(signal, label='Original Signal', color='green')
plt.plot(np.abs(custom_signal), label='Filtered Cosine Signal', color='gold')
plt.title('Original vs Filtered Cosine Signal')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(signal, label='Original Signal', color='green')
plt.plot(np.abs(gaussian_signal), label='Filtered Gaussian Signal', color='red')
plt.title('Original vs Filtered Gaussian Signal')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.legend()

plt.tight_layout()
plt.show()
```



For the Finite Impulse Response Filtered (FIR)

[64]:
```
def fir_filter(signal, taps):

    window = np.hamming(taps)
    b = window / np.sum(window)

    # Apply the filter
    filtered_signal = np.convolve(signal, b, mode='same')
    return filtered_signal

center_frequency = 1000
bandwidth = 200
standard_deviation = 100
num_taps = 31

plt.subplot(2, 1, 2)
plt.plot(signal, label='Original Signal', color='green')
plt.plot(filtered_signal_fir, label='Filtered FIR Signal', color='magenta')
plt.title('Original vs Filtered FIR Signal')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.legend()
```

```
plt.tight_layout()
plt.show()
```