CCO50- Digital Speech Processing

Short Test 3

Description: Design a FIR filter $h[n]$ of order $M = 5$ to allow for frequencies up to 1500 Hz to pass through, assuming that the input signal to be filtered $x[n]$ was sampled at 12000 samples per second. Considering, as an example, that $x[n] = (1, 1, 2, 2, 2)$, filter it by using $h[n]$, obtaining the output signal.

$$\begin{gathered} h[n] = \frac{\sin\left(\omega_c(n - \frac{M}{2})\right)}{\pi(n - \frac{M}{2})} \\ M = 5 \\ 0 \leq n \leq 5 \\ \pi = 6kHz \\ \omega_c = 1,5kHz = \frac{\pi}{4} \end{gathered}$$

Calculando cada index de h:

$$\begin{gathered} h[0] = \frac{\sin\left(\frac{\pi}{4}(0 - \frac{5}{2})\right)}{\pi(0 - \frac{5}{2})} = 0,11763192 \\ h[1] = \frac{\sin\left(\frac{\pi}{4}(1 - \frac{5}{2})\right)}{\pi(1 - \frac{5}{2})} = 0,19605321 \\ h[2] = \frac{\sin\left(\frac{\pi}{4}(2 - \frac{5}{2})\right)}{\pi(2 - \frac{5}{2})} = 0,243623839 \\ h[3] = \frac{\sin\left(\frac{\pi}{4}(3 - \frac{5}{2})\right)}{\pi(3 - \frac{5}{2})} = 0,243623839 \\ h[4] = \frac{\sin\left(\frac{\pi}{4}(4 - \frac{5}{2})\right)}{\pi(4 - \frac{5}{2})} = 0,196053321 \\ h[5] = \frac{\sin\left(\frac{\pi}{4}(5 - \frac{5}{2})\right)}{\pi(5 - \frac{5}{2})} = 0,11763192 \end{gathered}$$

$$ h = (0.117, 0.196, 0.243, 0.243, 0.196, 0.117)

com o h obtido, aplicar o filtro a $x = (1, 1, 2, 2, 2)$

					1	1	2	2	2	x
				0.117	0.196	0.243	0.243	0.196	0.117	h
					0.117	0.117	0.234	0.234	0.234	
				0.196	0.196	0.392	0.392	0.392	+	
			0.243	0.243	0.486	0.486	0.486	+		
		0.243	0.243	0.486	0.486	0.486	+			
	0.196	0.196	0.392	0.392	0.392	+				
0.117	0.117	0.234	0.234	0.234	+					
(0.117	0.313	0.673	1.112	1.551	1.677	1.481	1.112	0.626	0.234)	$= x * h$

$$x * h = (0.117, 0.313, 0.673, 1.112, 1.551, 1.677, 1.481, 1.112, 0.626, 0.234)$$

In [1]:
```
import numpy as np
```

Plus: implementing the filter in python

In [2]:
```
def low_pass_filter(cutoff, sample_rate, M=5):
    nyq = sample_rate / 2
```

```
    omega = cutoff / nyq
    h = np.zeros(M+1)
    for n in range(M+1):
        h[n] = (np.sin(np.pi * omega * (n - (M/2)))/(np.pi * (n - (M/2))))
    return h
```

In [3]:
```
h = low_pass_filter(1500, 12000, M=5)
print(h)
```

```
[0.117632   0.19605333 0.24362384 0.24362384 0.19605333 0.117632  ]
```

using the `discrete_time_convolution` method defined in ST2

In [4]:
```
def discrete_time_convolution(k, x):
    n = len(k)
    m = len(x)
    y = [0] * (m + n - 1)

    for i in range(m):
        for j in range(n):
            y[i + j] += x[i] * k[j]

    return y
```

In [5]:
```
x = [1, 1, 2, 2, 2]
print(discrete_time_convolution(h, x))
```

```
[0.11763199553648057, 0.3136853214306149, 0.6749411565682036, 1.1146183220634462, 1.5542954875586887, 1.680287331
623316, 1.4842340057291818, 1.1146183220634462, 0.6273706428612298, 0.23526399107296114]
```

Extra

Generating waves and filtering them

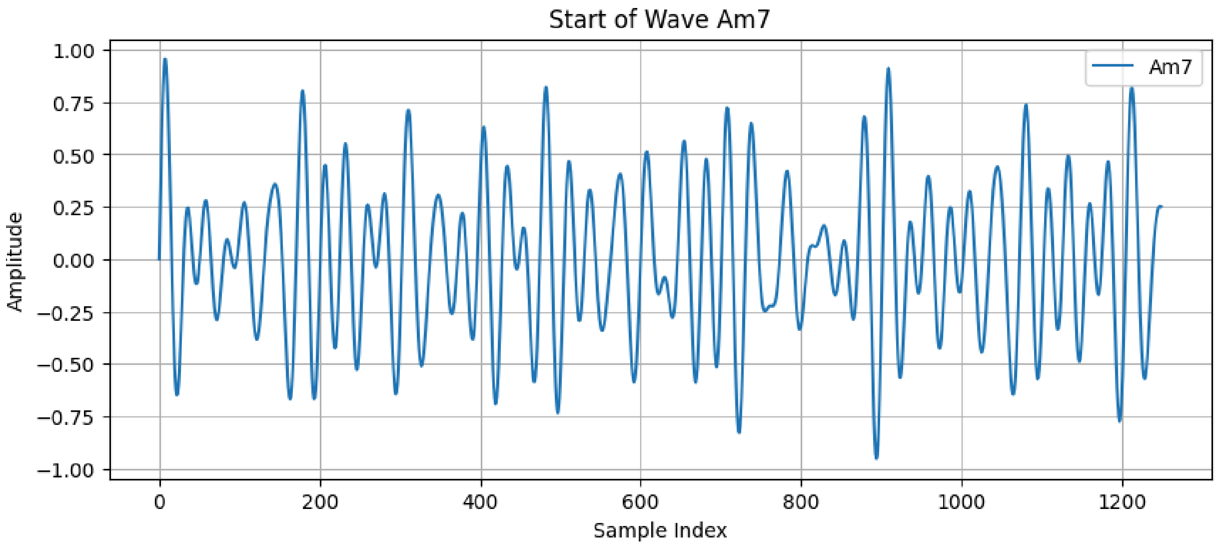
In [6]:
```
import numpy as np
import pyaudio
import matplotlib.pyplot as plt
```

In [7]:
```
# defining a function to play the signal
def play_signal(signal, sample_rate=44100):
    p = pyaudio.PyAudio()

    stream = p.open(format=pyaudio.paFloat32,
                    channels=1,
                    rate=sample_rate,
                    output=True)

    stream.write(signal.tobytes())

    stream.stop_stream()
    stream.close()

    p.terminate()
```

In [8]:
```
# defining a function to see the signal
def plot_waveform(wave, label, title):
    plt.figure(figsize=(10, 4))
    plt.plot(wave, label=label)
    plt.title(title)
    plt.xlabel("Sample Index")
    plt.ylabel("Amplitude")
    plt.legend()
    plt.grid()
    plt.show()
```

In [9]:
```
# generating a Am7 chord waves
sample_rate = 44100
seconds = 2
octave = 4 # 1 for central octave, to lower pitch use 1/octave
s_A = (np.sin(2 * np.pi * np.arange(sample_rate * seconds) * 440 * octave / sample_rate)).astype(np.float32)
s_C = (np.sin(2 * np.pi * np.arange(sample_rate * seconds) * 256.63 * octave / sample_rate)).astype(np.float32)
s_E = (np.sin(2 * np.pi * np.arange(sample_rate * seconds) * 329.63 * octave / sample_rate)).astype(np.float32)
s_G = (np.sin(2 * np.pi * np.arange(sample_rate * seconds) * 392 * octave / sample_rate)).astype(np.float32)
s_Am7 = s_A + s_C + s_E + s_G
s_Am7 /= np.max(np.abs(s_Am7))  # Normalize the signal
```

In [10]:
```
plot_waveform(s_Am7[:1250], "Am7", "Start of Wave Am7")
```



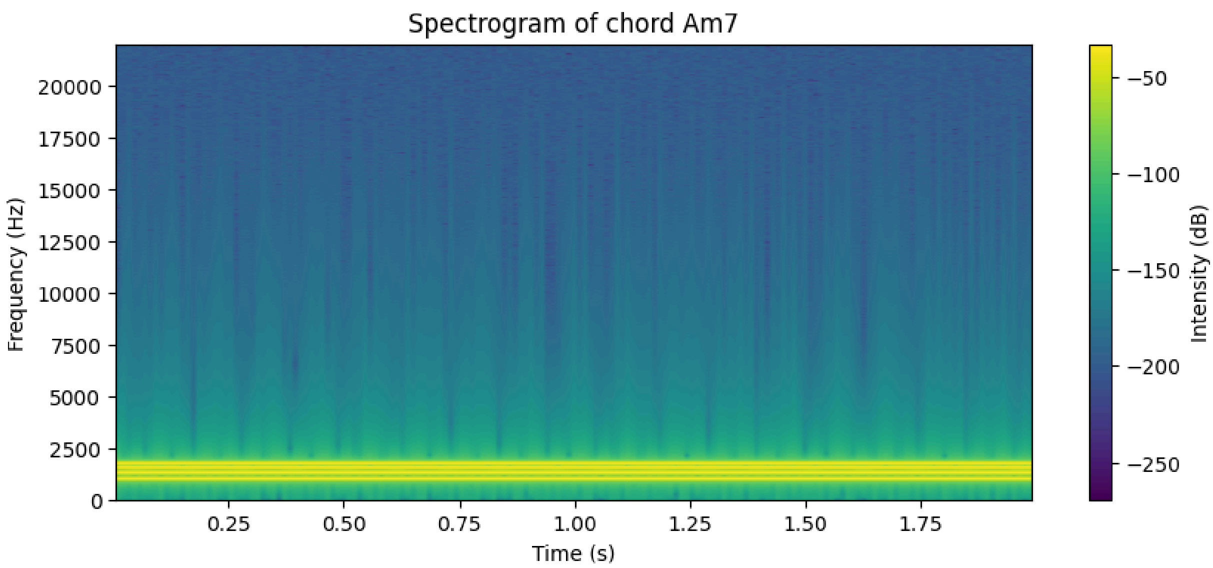
In [11]:
```
play_signal(s_Am7 * 0.5)
```

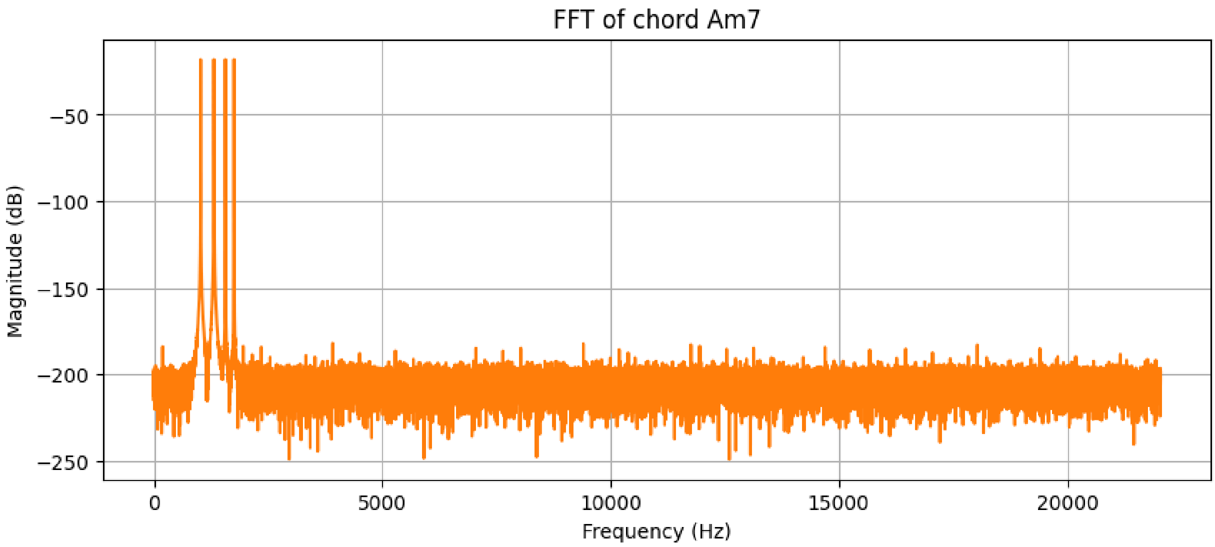
Analyzing the spectogram

In [12]:
```
# defining a function to see the spectrogram
def plot_spectrogram(signal, title):
    plt.figure(figsize=(10, 4))
    plt.specgram(signal, Fs=sample_rate, NFFT=1024, noverlap=512, cmap='viridis')
    plt.title(title)
    plt.xlabel("Time (s)")
    plt.ylabel("Frequency (Hz)")
    plt.colorbar(label='Intensity (dB)')
    plt.show()
```

In [13]:
```
def plot_fft(signal, title):
    plt.figure(figsize=(10, 4))
    plt.magnitude_spectrum(signal, Fs=sample_rate, scale='dB', color='C1')
    plt.title(title)
    plt.xlabel("Frequency (Hz)")
    plt.ylabel("Magnitude (dB)")
    plt.grid()
    plt.show()
```

In [14]:
```
plot_spectrogram(s_Am7, "Spectrogram of chord Am7")
plot_fft(s_Am7, "FFT of chord Am7")
```





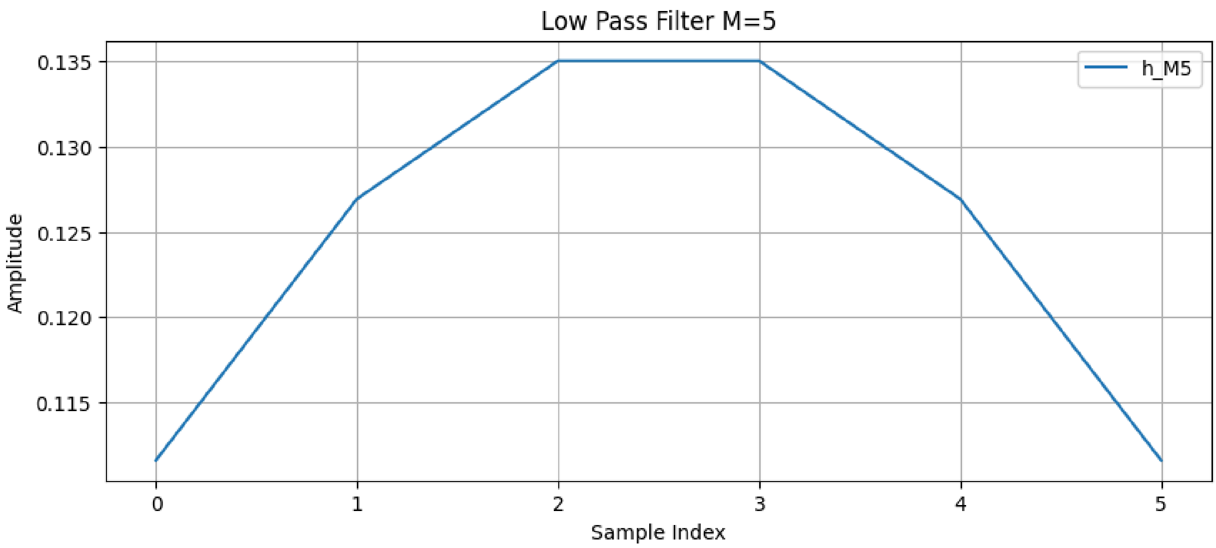
As seen in the Spectogram of the not filtered wave, there are peaks of energy for some lower frequencies, that are the harmonics in the chord, and a lot of variation in the rest.

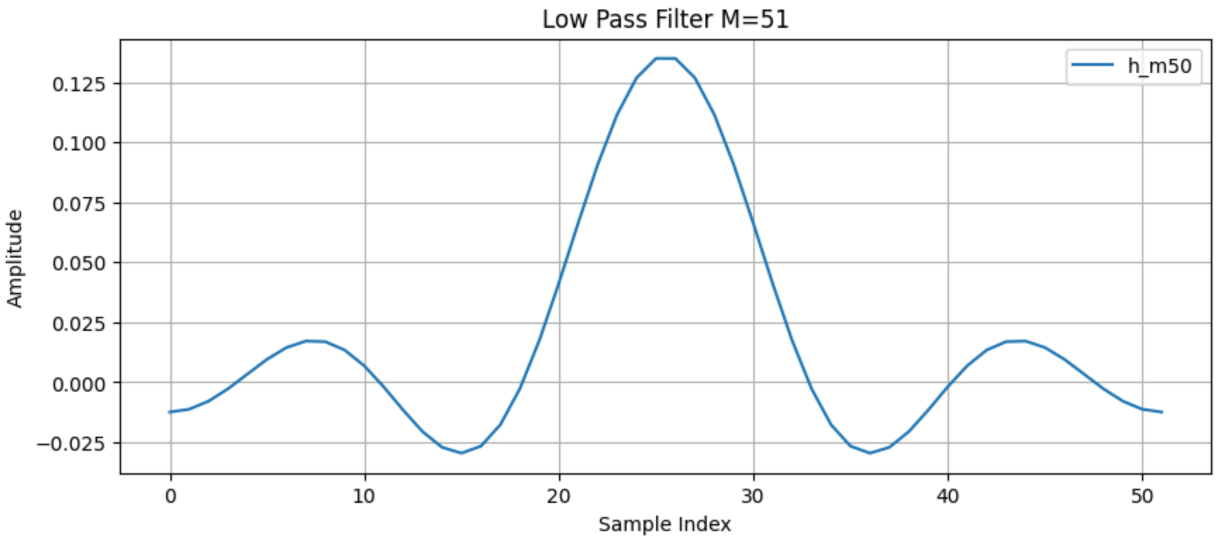
Creating a low pass filter

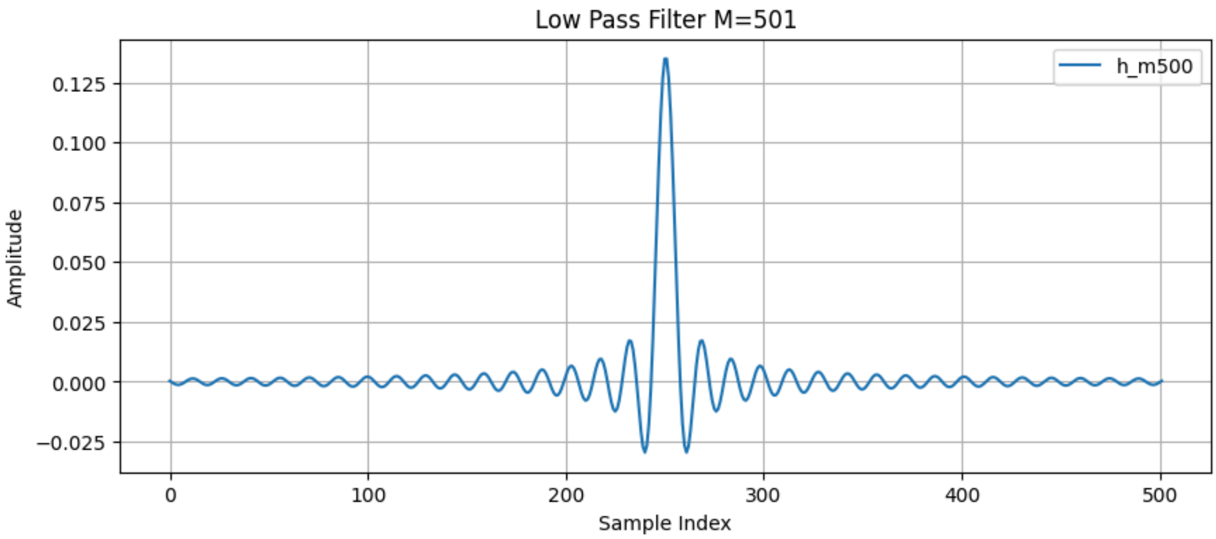
Lets filter up to 3kHz, changing the M

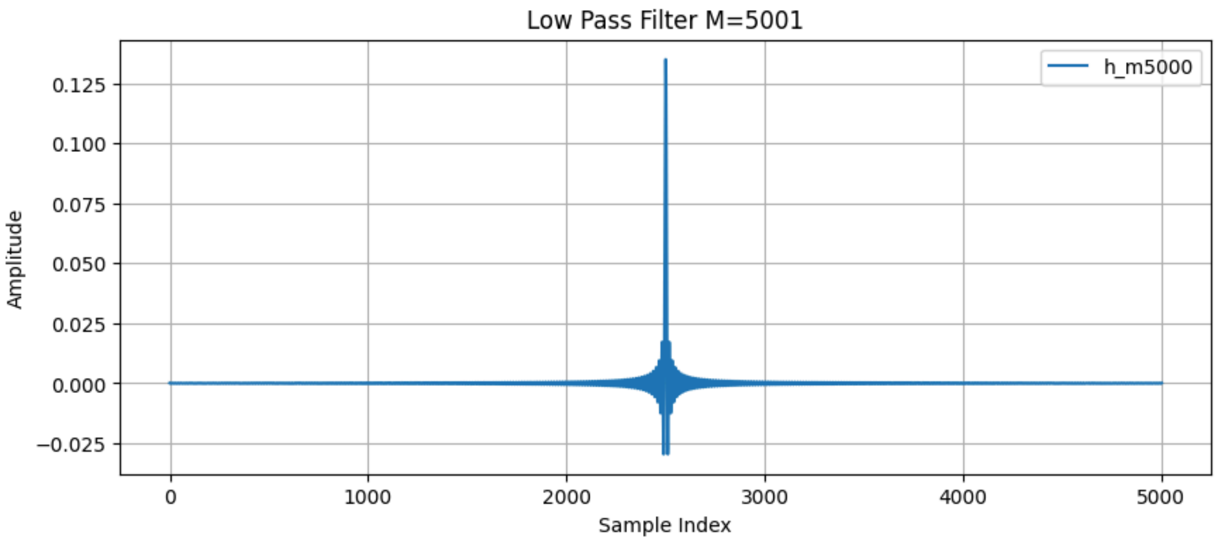
In [15]:
```
h_M5 = low_pass_filter(3000, 44100, M=5)
h_m50 = low_pass_filter(3000, 44100, M=51)
h_m500 = low_pass_filter(3000, 44100, M=501)
h_m5000 = low_pass_filter(3000, 44100, M=5001)
```

In [16]:
```
plot_waveform(h_M5, "h_M5", "Low Pass Filter M=5")
plot_waveform(h_m50, "h_m50", "Low Pass Filter M=51")
plot_waveform(h_m500, "h_m500", "Low Pass Filter M=501")
plot_waveform(h_m5000, "h_m5000", "Low Pass Filter M=5001")
```









In [17]:

```
s_Am7_filtered_M5 = np.convolve(h_M5, s_Am7)
s_Am7_filtered_m50 = np.convolve(h_m50, s_Am7)
s_Am7_filtered_m500 = np.convolve(h_m500, s_Am7)
s_Am7_filtered_m5000 = np.convolve(h_m5000, s_Am7)

#normalize the filtered signals
print("Max value before normalization:")
print("M5:", np.max(np.abs(s_Am7_filtered_M5)))
print("m50:", np.max(np.abs(s_Am7_filtered_m50)))
print("m500:", np.max(np.abs(s_Am7_filtered_m500)))
print("m5000:", np.max(np.abs(s_Am7_filtered_m5000)))
```

```
s_Am7_filtered_M5 = s_Am7_filtered_M5 / np.max(np.abs(s_Am7_filtered_M5))
s_Am7_filtered_m50 = s_Am7_filtered_m50 / np.max(np.abs(s_Am7_filtered_m50))
s_Am7_filtered_m500 = s_Am7_filtered_m500 / np.max(np.abs(s_Am7_filtered_m500))
s_Am7_filtered_m5000 = s_Am7_filtered_m5000 / np.max(np.abs(s_Am7_filtered_m5000))

print("Max value after normalization:")
print("M5:", np.max(np.abs(s_Am7_filtered_M5)))
print("m50:", np.max(np.abs(s_Am7_filtered_m50)))
print("m500:", np.max(np.abs(s_Am7_filtered_m500)))
print("m5000:", np.max(np.abs(s_Am7_filtered_m5000)))
```

```
Max value before normalization:
M5: 0.7044316531944156
m50: 0.9887638750681299
m500: 0.9969472157165598
m5000: 0.9993701749226062
Max value after normalization:
M5: 1.0
m50: 1.0
m500: 1.0
m5000: 1.0
```

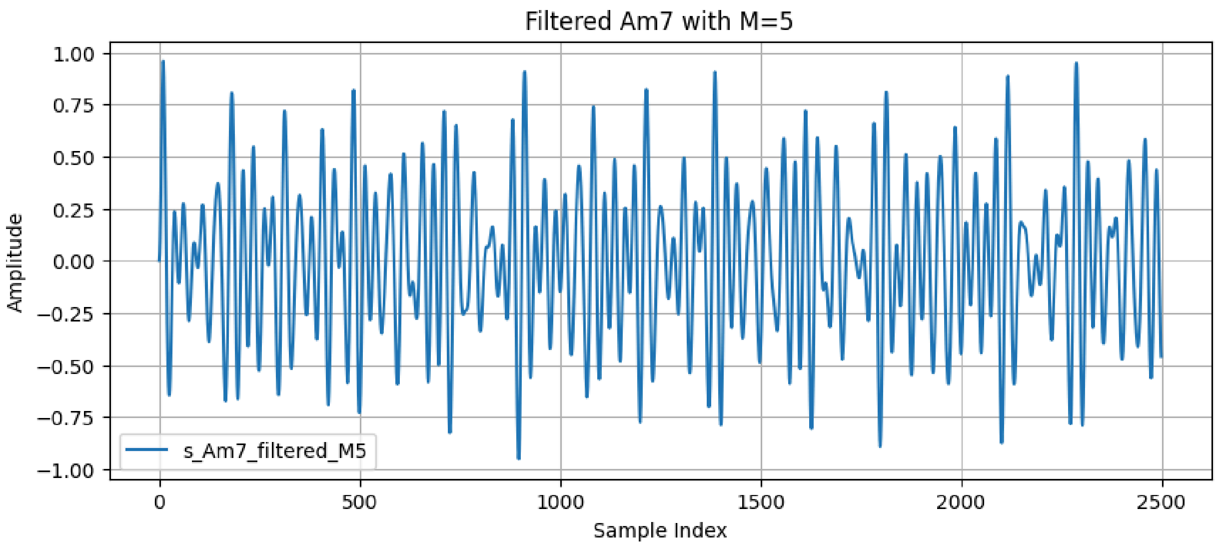
In [ ]:

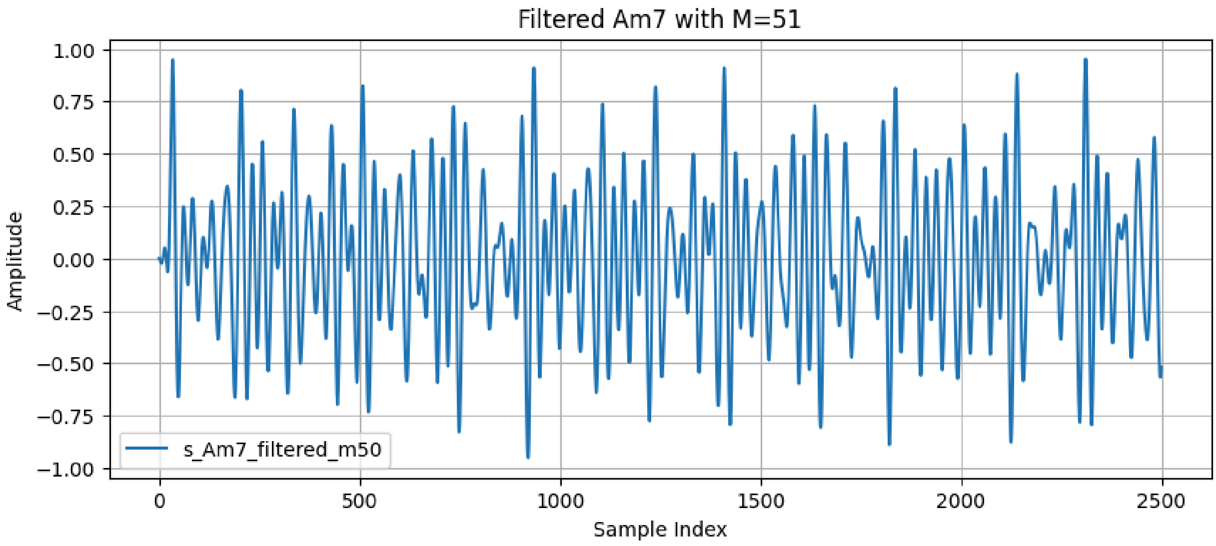
```
play_signal(s_Am7_filtered_M5, sample_rate=44100)
play_signal(s_Am7_filtered_m50, sample_rate=44100)
play_signal(s_Am7_filtered_m500, sample_rate=44100)
play_signal(s_Am7_filtered_m5000, sample_rate=44100)
```

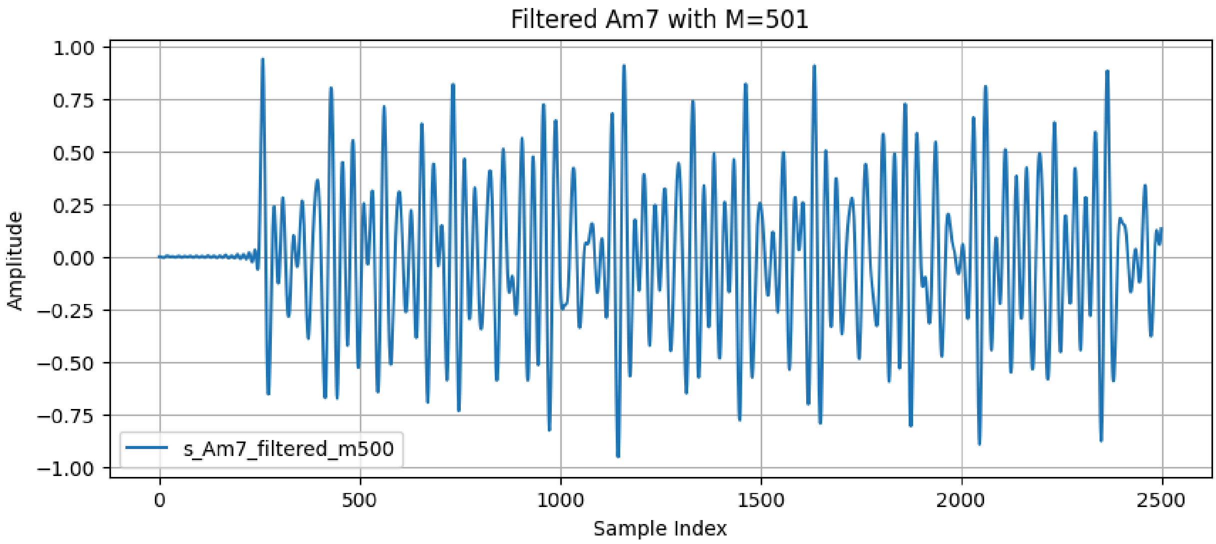
I really dont know what I did wrong, but ir didn`t work as I expected, so much noise and so loud! but the signal is normalized, even when multiplying by a very small factor to lower the amplitude, the signal still is loud.

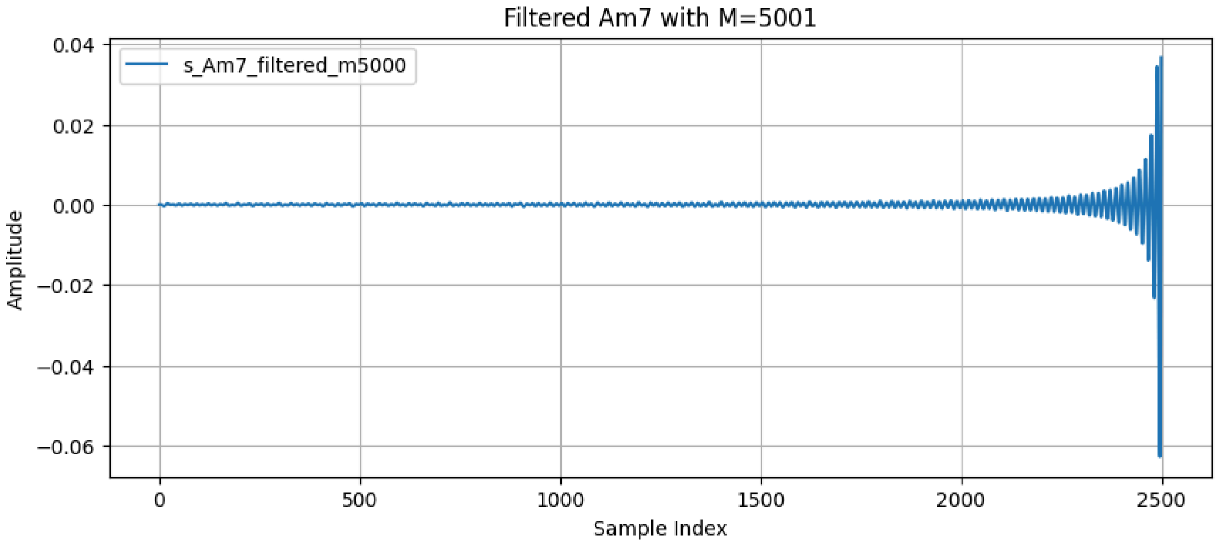
In [19]:

```
plot_waveform(s_Am7_filtered_M5[:2500], "s_Am7_filtered_M5", "Filtered Am7 with M=5")
plot_waveform(s_Am7_filtered_m50[:2500], "s_Am7_filtered_m50", "Filtered Am7 with M=51")
plot_waveform(s_Am7_filtered_m500[:2500], "s_Am7_filtered_m500", "Filtered Am7 with M=501")
plot_waveform(s_Am7_filtered_m5000[:2500], "s_Am7_filtered_m5000", "Filtered Am7 with M=5001")
plot_spectrogram(s_Am7_filtered_M5, "Spectrogram of filtered Am7 with M=5")
plot_spectrogram(s_Am7_filtered_m50, "Spectrogram of filtered Am7 with M=51")
plot_spectrogram(s_Am7_filtered_m500, "Spectrogram of filtered Am7 with M=501")
plot_spectrogram(s_Am7_filtered_m5000, "Spectrogram of filtered Am7 with M=5001")
plot_fft(s_Am7_filtered_M5, "FFT of filtered Am7 with M=5")
plot_fft(s_Am7_filtered_m50, "FFT of filtered Am7 with M=51")
plot_fft(s_Am7_filtered_m500, "FFT of filtered Am7 with M=501")
plot_fft(s_Am7_filtered_m5000, "FFT of filtered Am7 with M=5001")
```

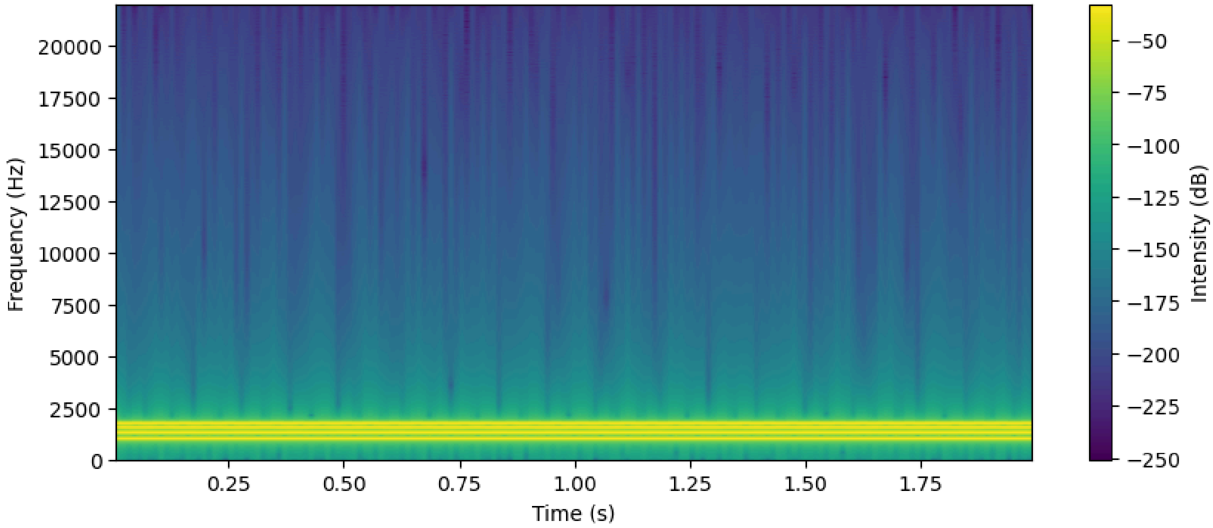



Filtered Am7 with M=51
Amplitude
Sample Index
s_Am7_filtered_m50
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
500
1000
1500
2000
2500

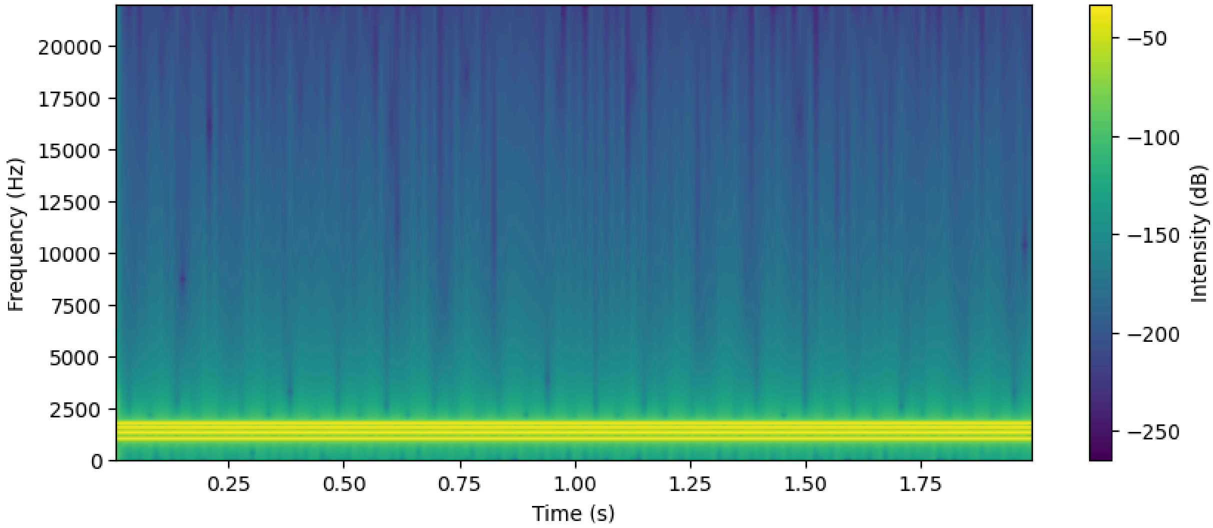

Filtered Am7 with M=501
Amplitude
Sample Index
s_Am7_filtered_m500
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
500
1000
1500
2000
2500


Filtered Am7 with M=5001
Amplitude
Sample Index
s_Am7_filtered_m5000
0.04
0.02
0.00
−0.02
−0.04
−0.06
0
500
1000
1500
2000
2500

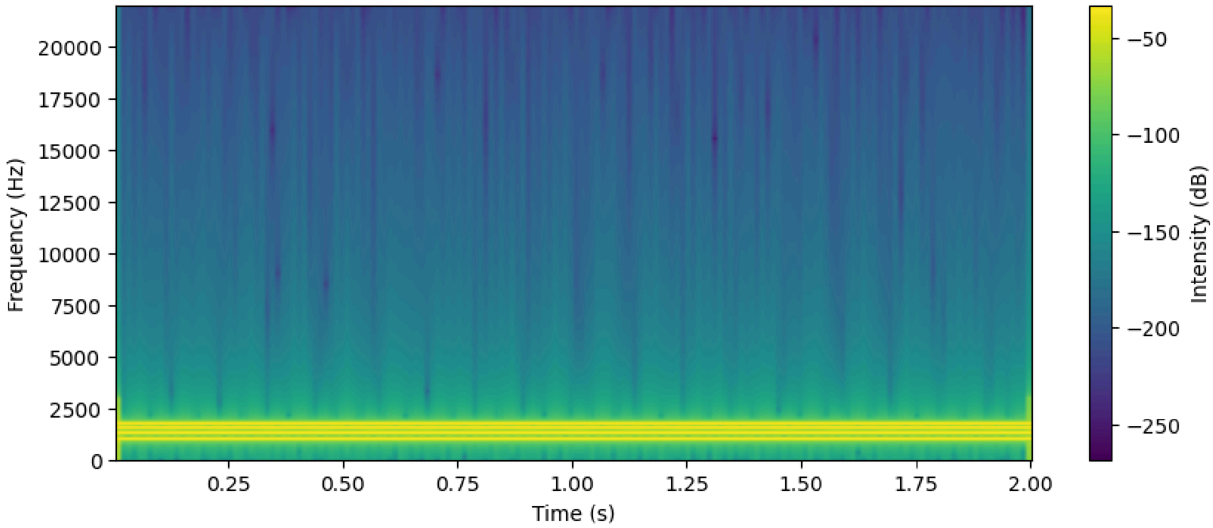
Spectrogram of filtered Am7 with M=5

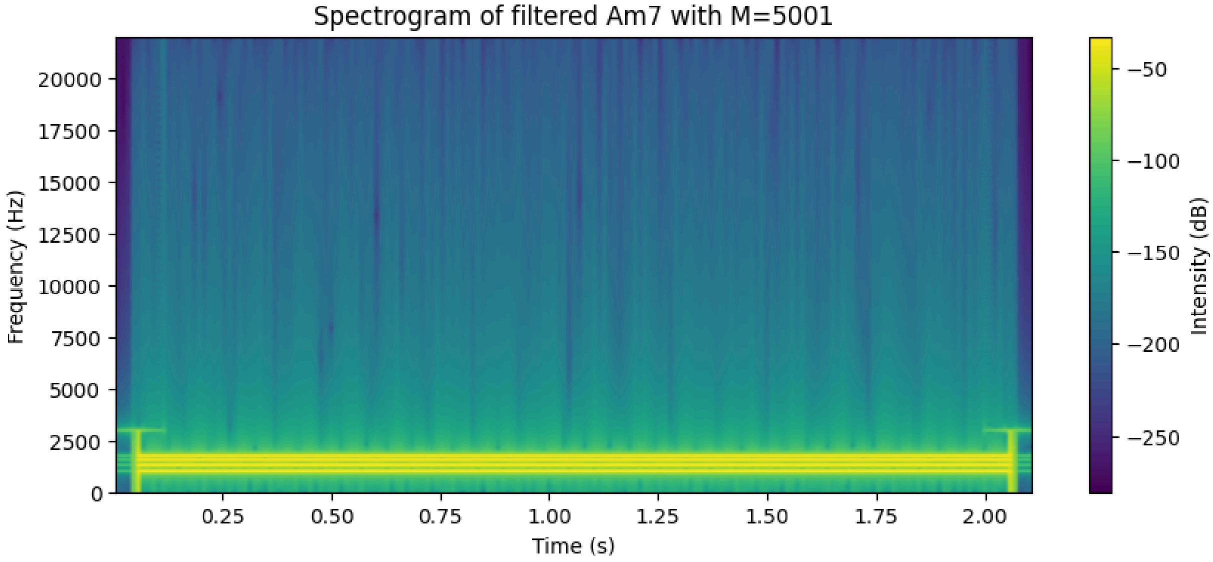


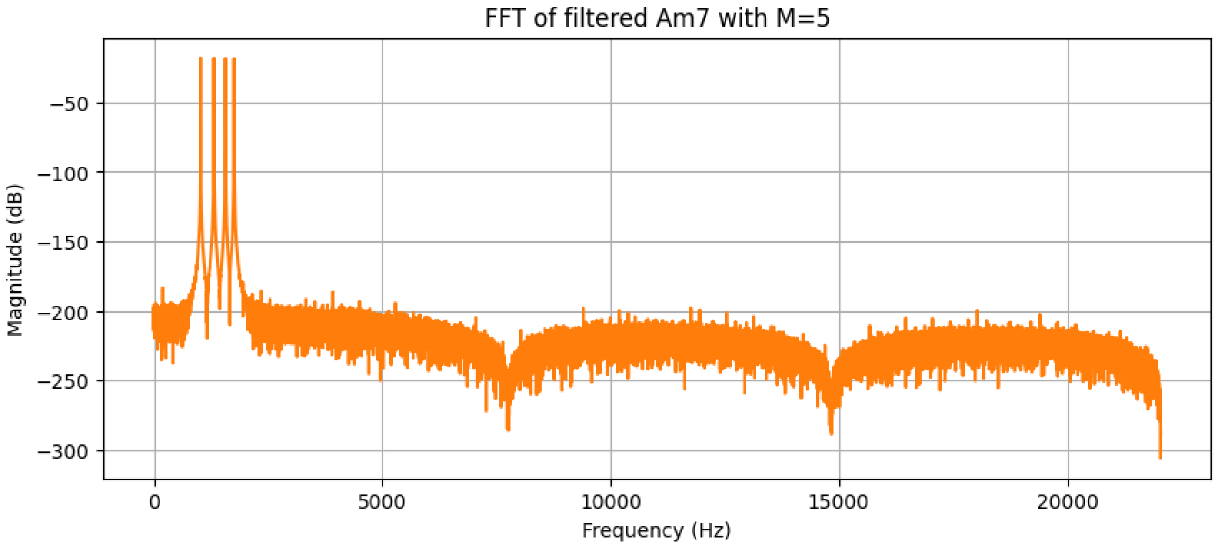
Spectrogram of filtered Am7 with M=51

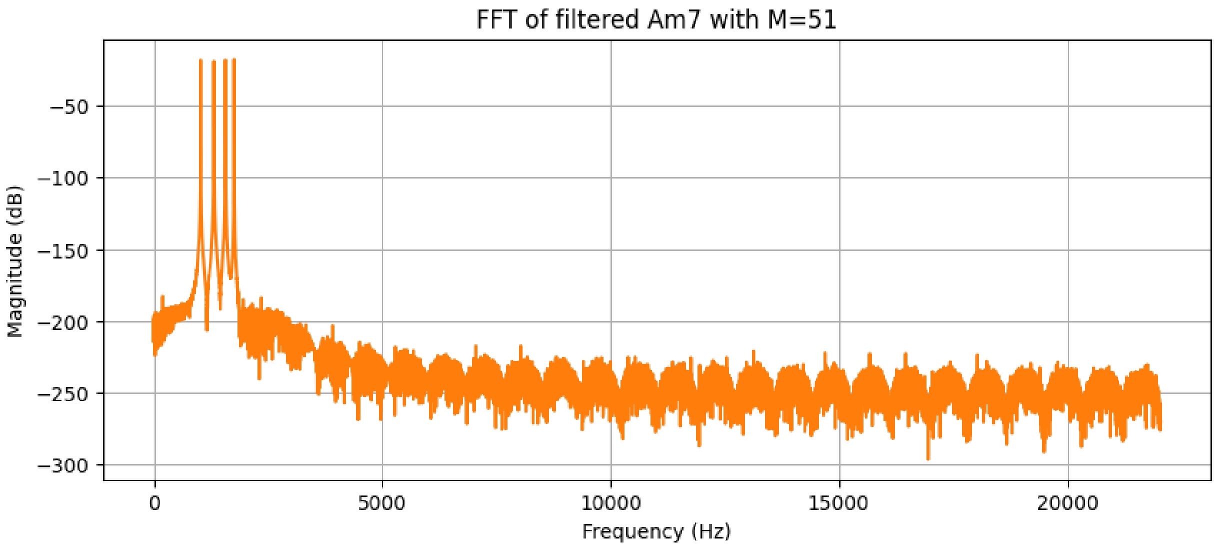


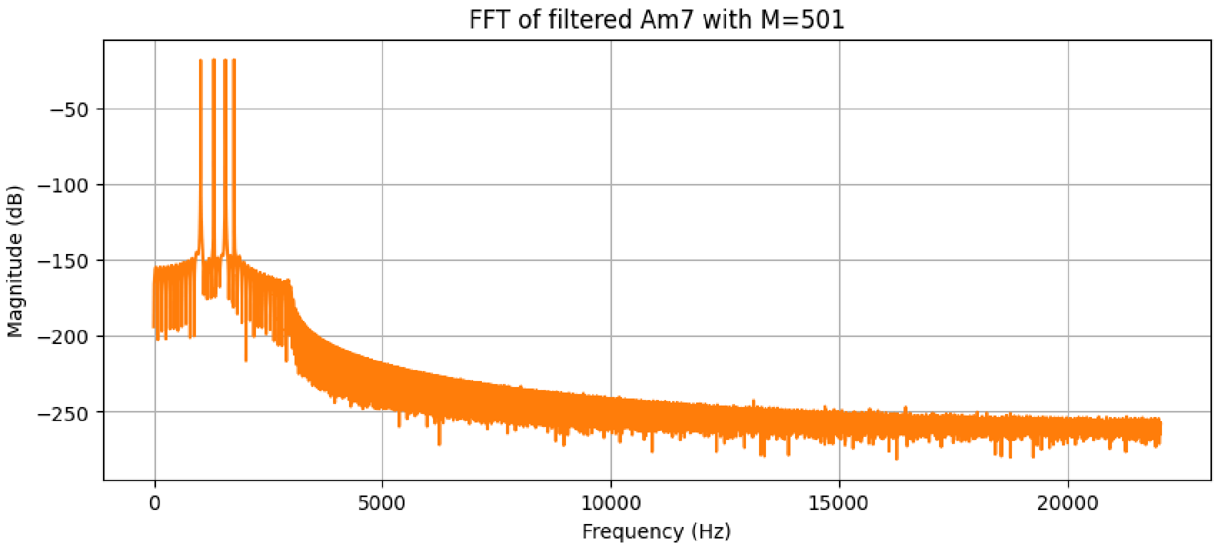
Spectrogram of filtered Am7 with M=501

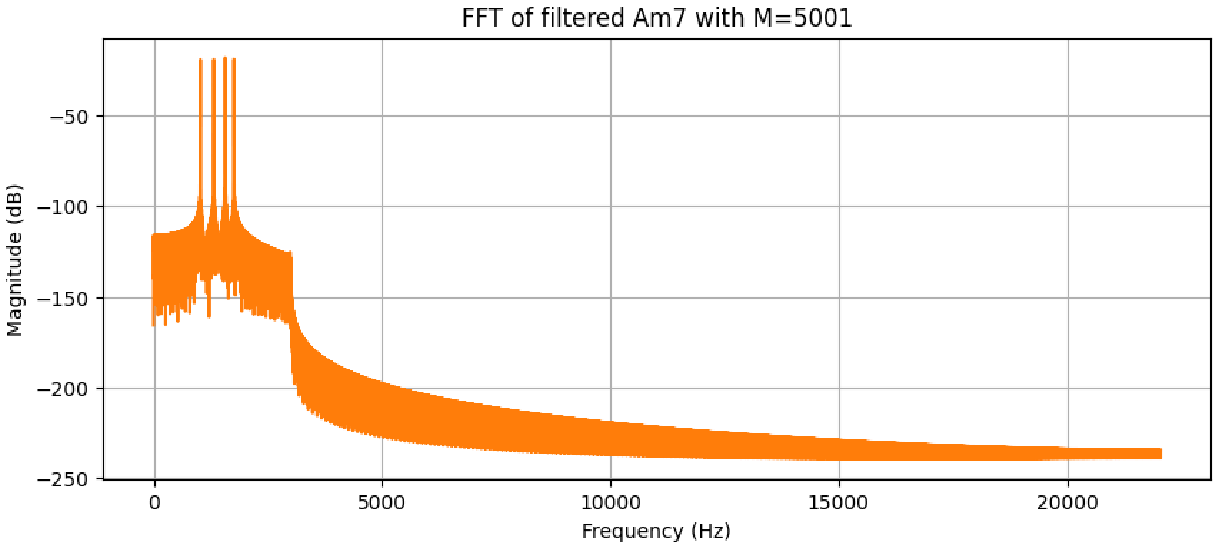



Spectrogram of filtered Am7 with M=5001
Frequency (Hz)
Time (s)
Intensity (dB)
20000
17500
15000
12500
10000
7500
5000
2500
0
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
−50
−100
−150
−200
−250


FFT of filtered Am7 with M=5
Magnitude (dB)
Frequency (Hz)
−50
−100
−150
−200
−250
−300
0
5000
10000
15000
20000


FFT of filtered Am7 with M=51
Magnitude (dB)
Frequency (Hz)
−50
−100
−150
−200
−250
−300
0
5000
10000
15000
20000





Well, seing the graphs, the filter indeed cuted off the higher frequencies, but why is the audio playing so noisy? It seems that the higher M, the better is the cutoff, but the higher it is, a small glitch is added at the beginning of the audio form, as seem in the waveplot and the spectogram. But the FFTs are very nice.

Author: Matheus Sinto Novaes
E-mail: matheus.sinto@unesp.br

Course: Digital Speech Processing
Professor: Dr. Eng. Rodrigo Capobianco Guido
Date: March 17 2025