Projeto Final

Grupo 3

- Henrique Hideaki Koga
- Luiz Henrique Araujo Dantas
- Marcelo Cavalcanti de Medeiros Sobrinho
- Vinícius de Azevedo Menezes

Etapas do projeto

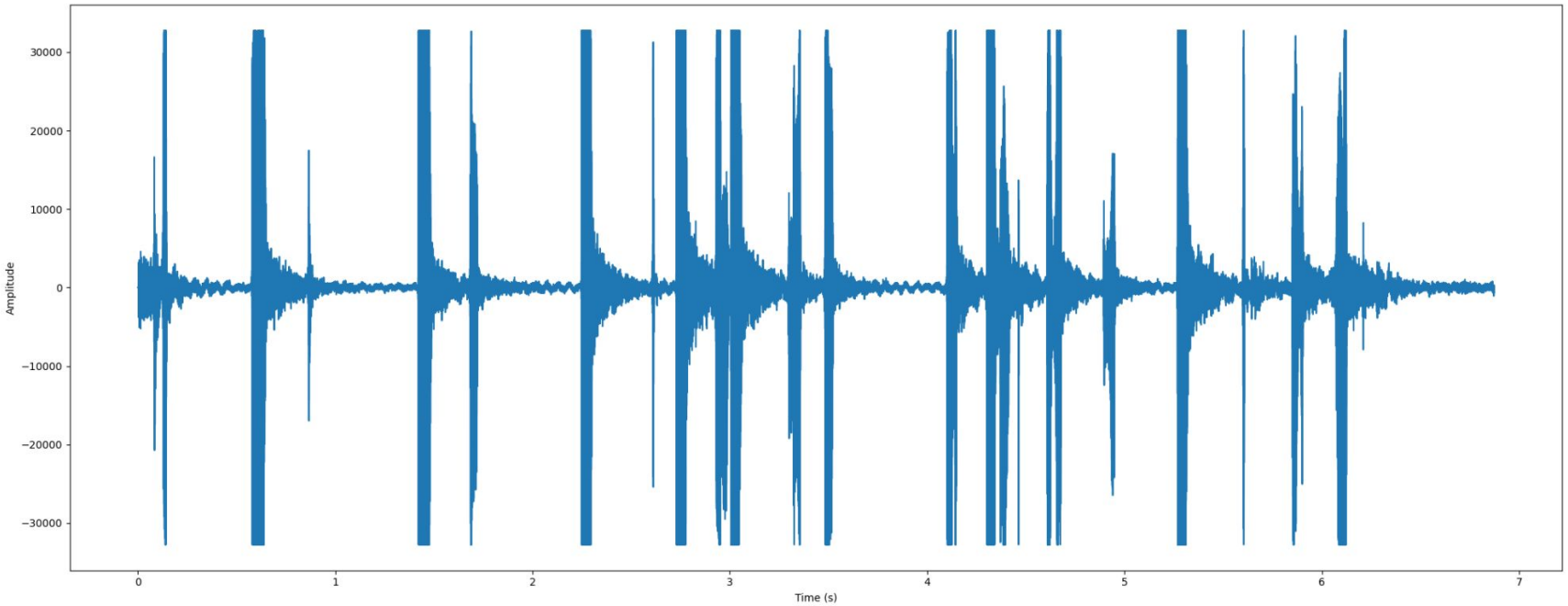
- (a) Plote o sinal no domínio do tempo
- (b) Usando a transformada rápida de Fourier (FFT), de alguma biblioteca de Python, plote o espectro de frequências do sinal para as primeiras N amostras. Use um valor de N adequado. Escalone o eixo de frequências das N amostras da FFT de forma adequada.
- (c) Fazendo uso da biblioteca pyFDA, projete um filtro digital passa-baixas com resposta ao impulso finita (FIR) que corte metade do conteúdo espectral do arquivo de áudio;
- (d) Com os coeficientes do filtro projetado, filtre o sinal.
  (d.1) Implemente a filtragem com a operação de convolução no domínio da frequência, fazendo uso de funções FFT e IFFT (utilize o método de sobreposição e soma ou sobreposição e armazenamento);
- (e) Utilizando os conceitos de mudança de taxa de amostragem vistos em aula, dizime o sinal por um fator M = 2, fazendo uso de um filtro passa-baixas adequado para evitar aliasing;
- (f) Plote os conteúdos temporais e espectrais após a filtragem e após a dizimação.

Arquivo de áudio utilizado

- “Dog Squeeking Toy”: https://freesound.org/people/vdimitro6497/sounds/620362/
- Nomeado em nosso projeto: dog.wav
- Samplerate: 44100.0 Hz

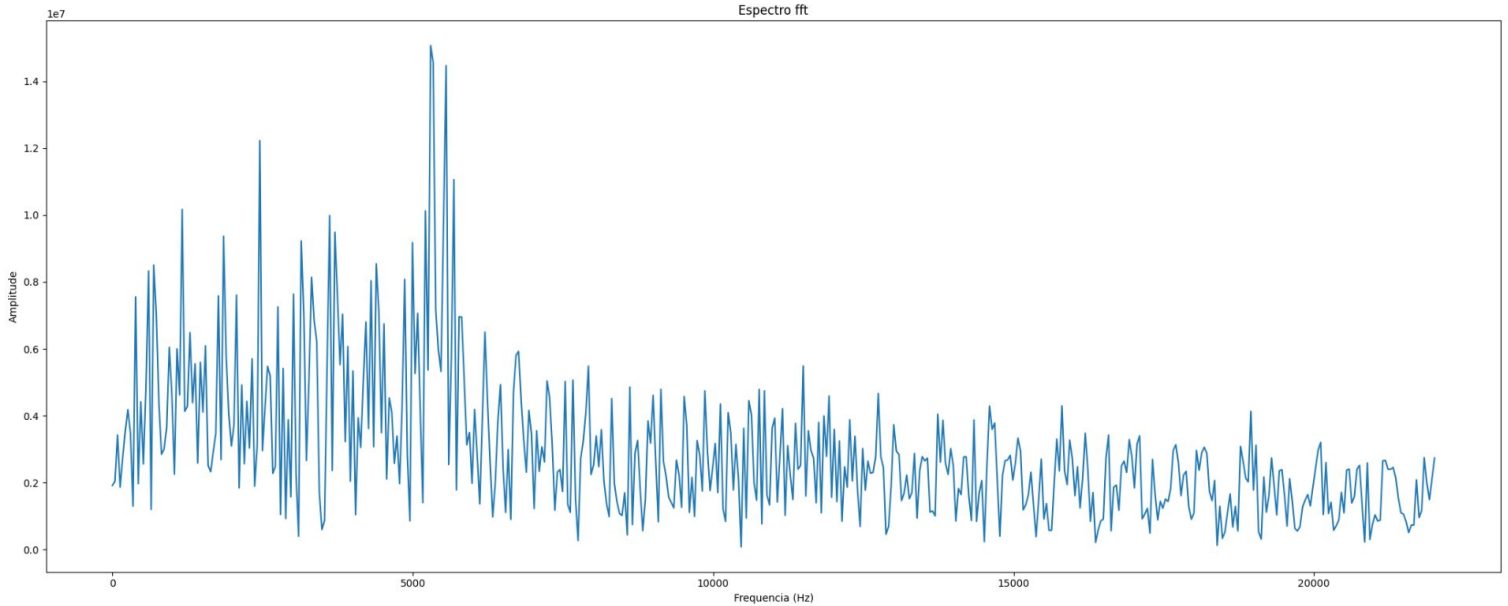
Etapa a: Plote o sinal no domínio do tempo.

```
#QUESTÃO A

#Length -> Tempo total = numero de amostras / frequencia de amostragem
frequencia_amostragem, data = wavfile.read('dog.wav')
length = data.shape[0] / frequencia_amostragem
time = np.linspace(0. , length, data.shape[0])

plt.plot(time, data)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.show()

print('\n\n')
print(f'Tempo de duracao do audio: {length} s')
print(f'Numero de amostras :       {data.shape[0]} ')
print(f'Frequencia de amostragem:   {frequencia_amostragem}Hz')
```


30000
20000
10000
0
−10000
−20000
−30000
Amplitude
0
1
2
3
4
5
6
7
Time (s)

Etapa b: Espectro de frequências do sinal para N amostras.

```
N = 1024
FFT = abs(fftpack.fft(data))
freqs = fftpack.fftfreq(N, (1.0/frequencia_amostragem))

# Plota o espectro fft
plt.plot(freqs[range(N//2)], FFT[range(N//2)])
plt.title("Espectro fft")
plt.xlabel('Frequencia (Hz)')
plt.ylabel('Amplitude')
plt.show()

hamming = np.genfromtxt("firPyFda.csv", delimiter=",")
```


Espectro fft
1e7
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Amplitude
0
5000
10000
15000
20000
Frequencia (Hz)

Filtro FIR com janelamento de Hamming

Equação da janela de Hamming

Hamming

$$w[n] = \begin{cases} 0{,}54 - 0{,}46\cos(2\pi n/M), & 0 \le n \le M, \\ 0, & \text{caso contrário} \end{cases} \quad (7.60\text{d})$$

Fonte: Processamento em tempo discreto de sinais. Allan V. Oppenheim p. 317

Etapa c: Filtro FIR com a biblioteca pyFDA

```
hamming = np.genfromtxt("firPyFda.csv", delimiter=",")
```

pyFDA - Python Filter Design and Analysis

Specs | b,a | P/Z | Info | Fixpoint

LOAD FILTER | SAVE FILTER

Lowpass | FIR | Windowed FIR

FFT Hamming

Order: Minimum N = 64

DESIGN FILTER | Quit

Freq. Unit Hz | 0...½

f_s = 4.41e+04

Target Specifications

Frequency in Hz | Ripple in dB

f_{PB} 4.41e+03 | A_{PB} 0.347

f_{SB} 8.82e+03 | A_{SB} 60

Frequency Specs in Hz

f_C 6.62e+03

Enter desired filter order **N** and **-6 dB** pass band corner frequency(ies) f_C .

|H(f)| | φ(f) | τ(f) | P / Z | y[n] | 3D

Magnitude Frequency Response

$|H(e^{j\Omega})|$ in dB →

0 −10 −20 −30 −40 −50 −60 −70 −80

0 2500 5000 7500 10000 12500 15000 17500 20000

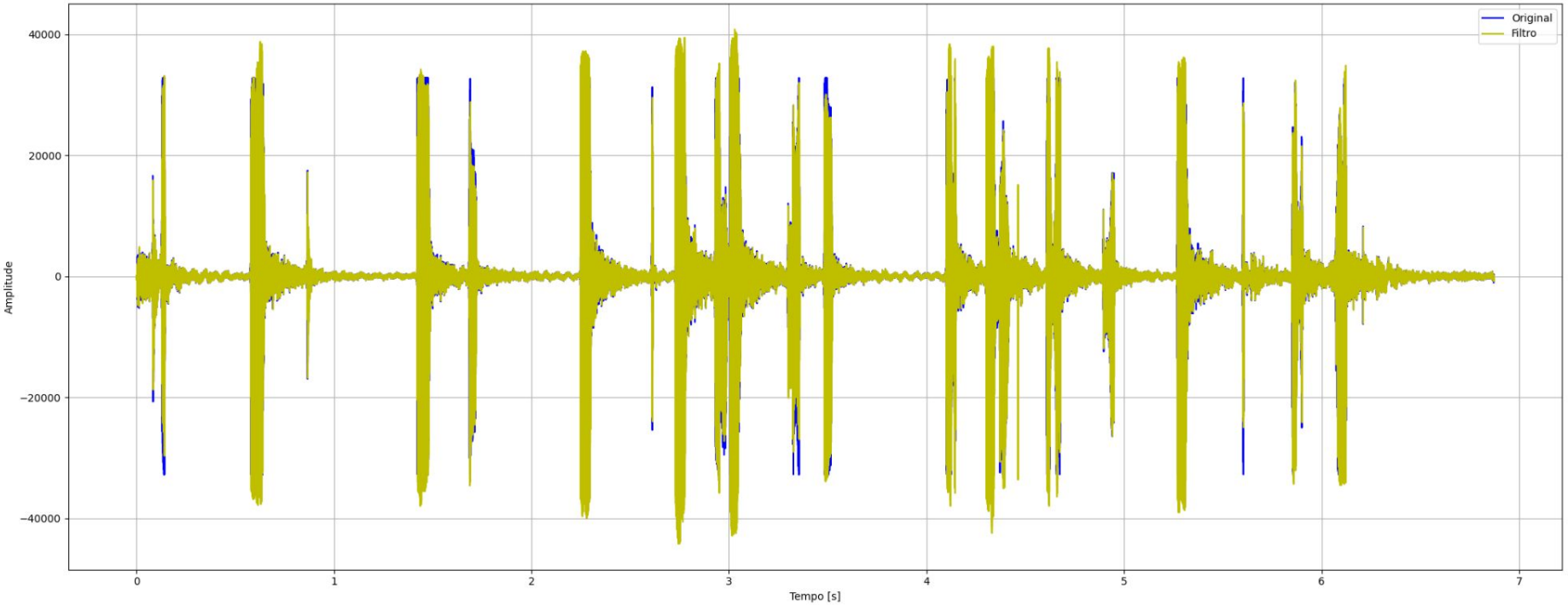
f in Hz →

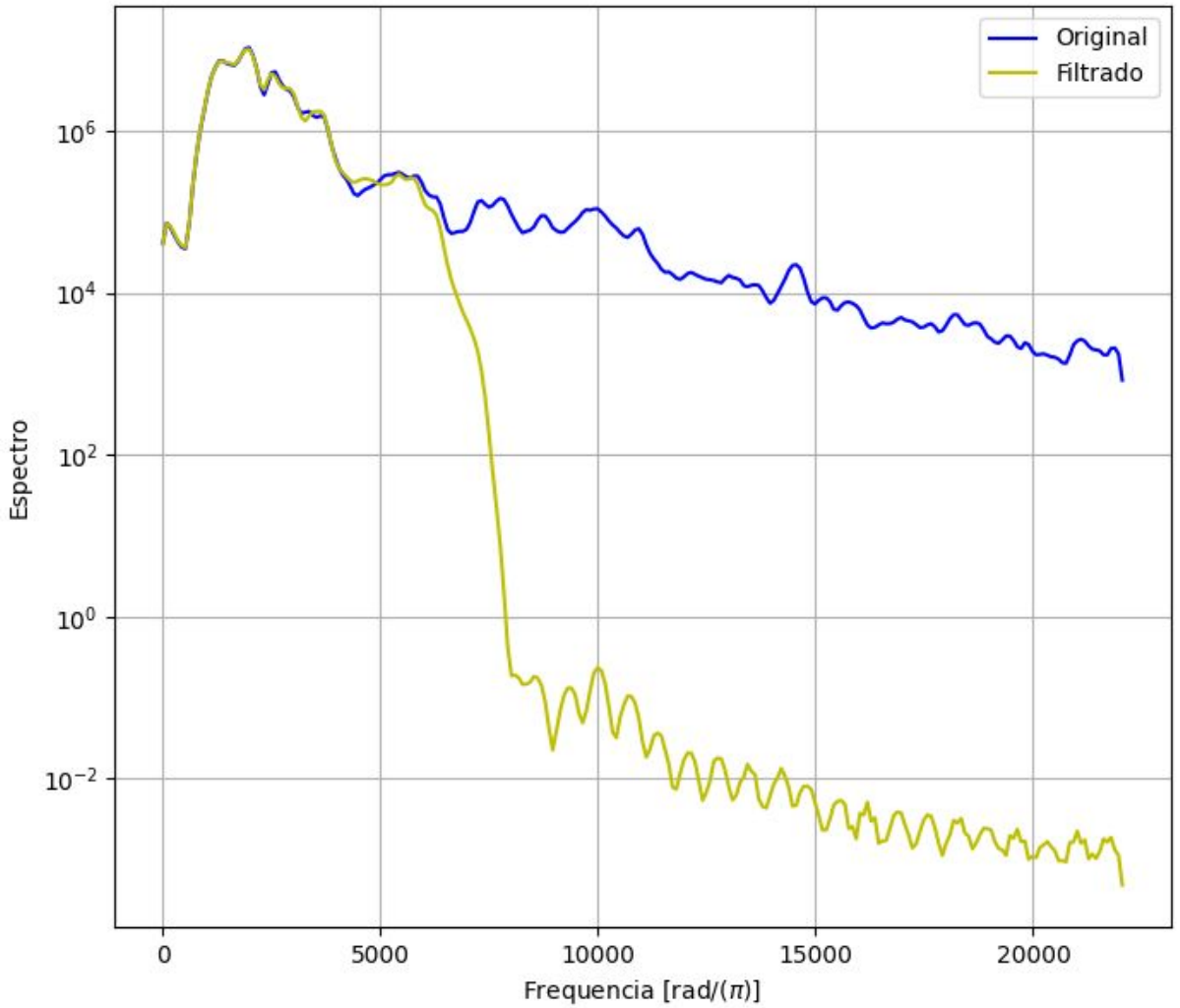
| H | Unit: Auto min = -80.0 dB Zero phase Inset off Specs Phase

Etapas d e d.1

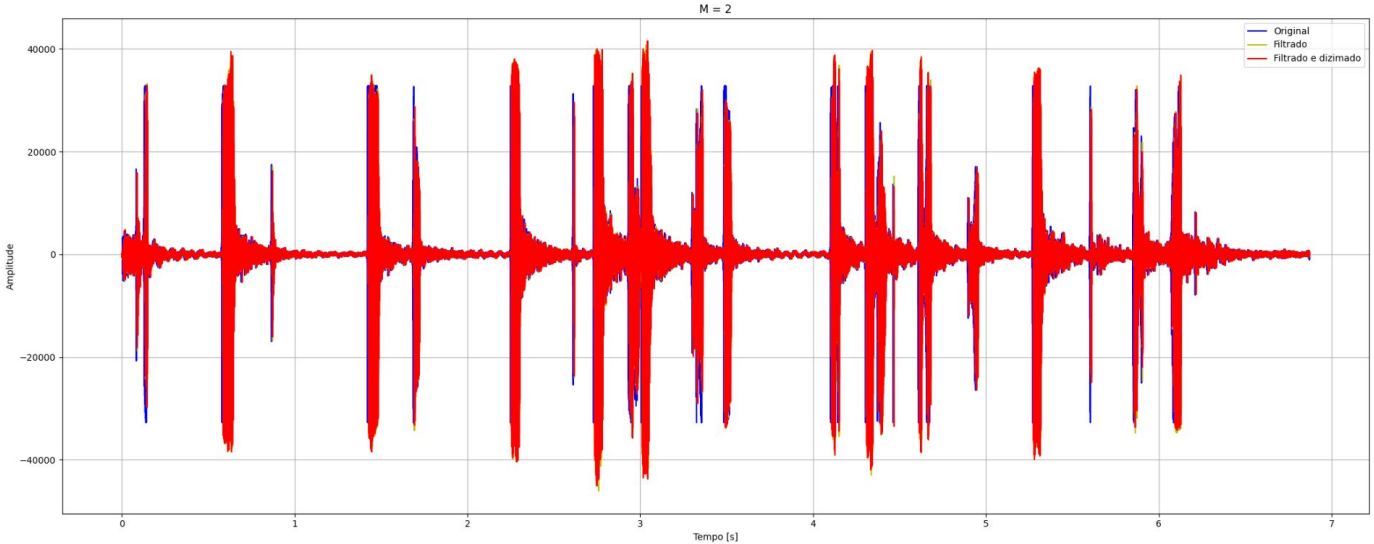
Método utilizado: sobreposição e soma

```python
#QUESTAO D

def overlapAndAdd(signal, fir):
    max_len = max(len(signal), len(fir))

    x = np.zeros(max_len)
    h = np.zeros(max_len)
    x[:len(signal)] = signal
    h[:len(fir)] = fir

    X = np.fft.fft(x)
    H = np.fft.fft(h)
    Y = X * H

    return np.real(np.fft.ifft(Y))

conv = overlapAndAdd(data, hamming)

# Plota no tempo os sinais original e filtrado
plt.plot(time, data, 'b-', label='Original')
plt.plot(time, conv, 'y-', label='Filtro')
plt.xlabel('Tempo [s]')
plt.ylabel("Amplitude")
plt.grid()
plt.legend()
plt.show()
```

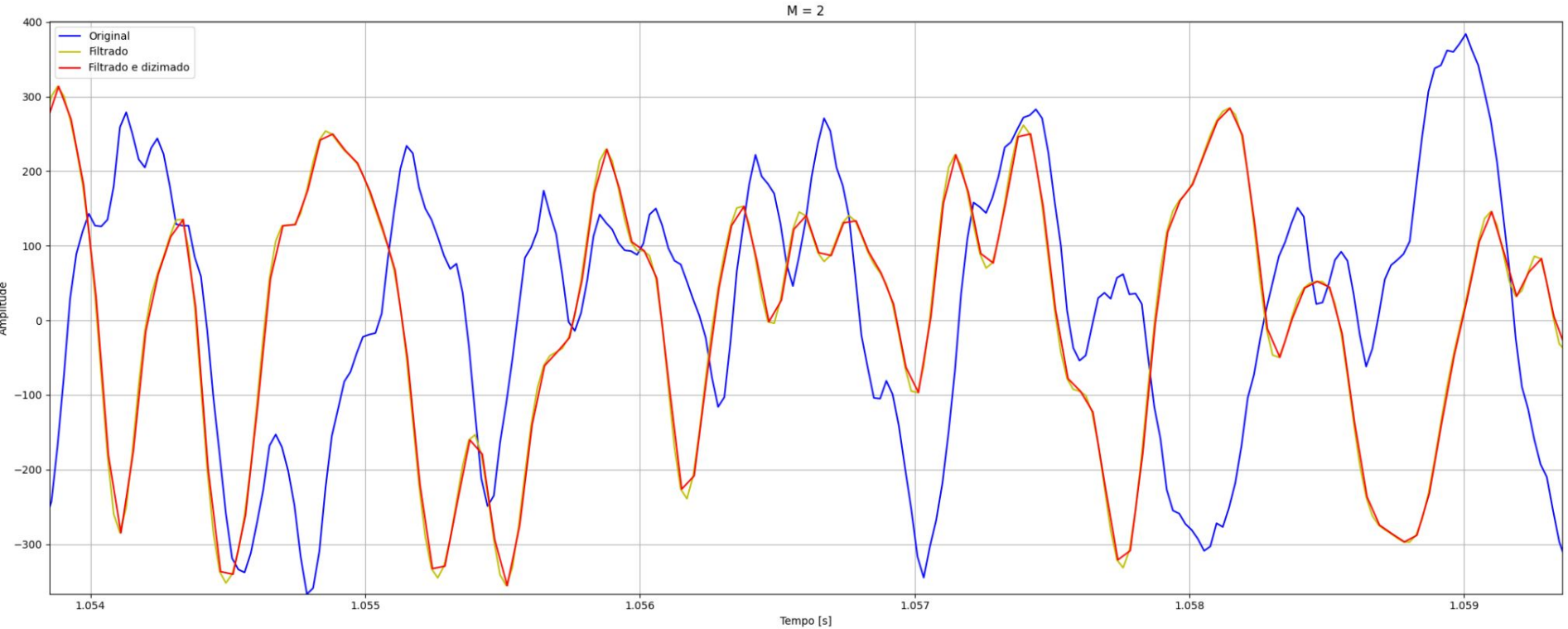

Original
Filtro
40000
20000
0
−20000
−40000
Amplitude
0
1
2
3
4
5
6
7
Tempo [s]


Original
Filtrado
10^6
10^4
10^2
10^0
10^{-2}
Espectro
0
5000
10000
15000
20000
Frequencia [rad/(π)]

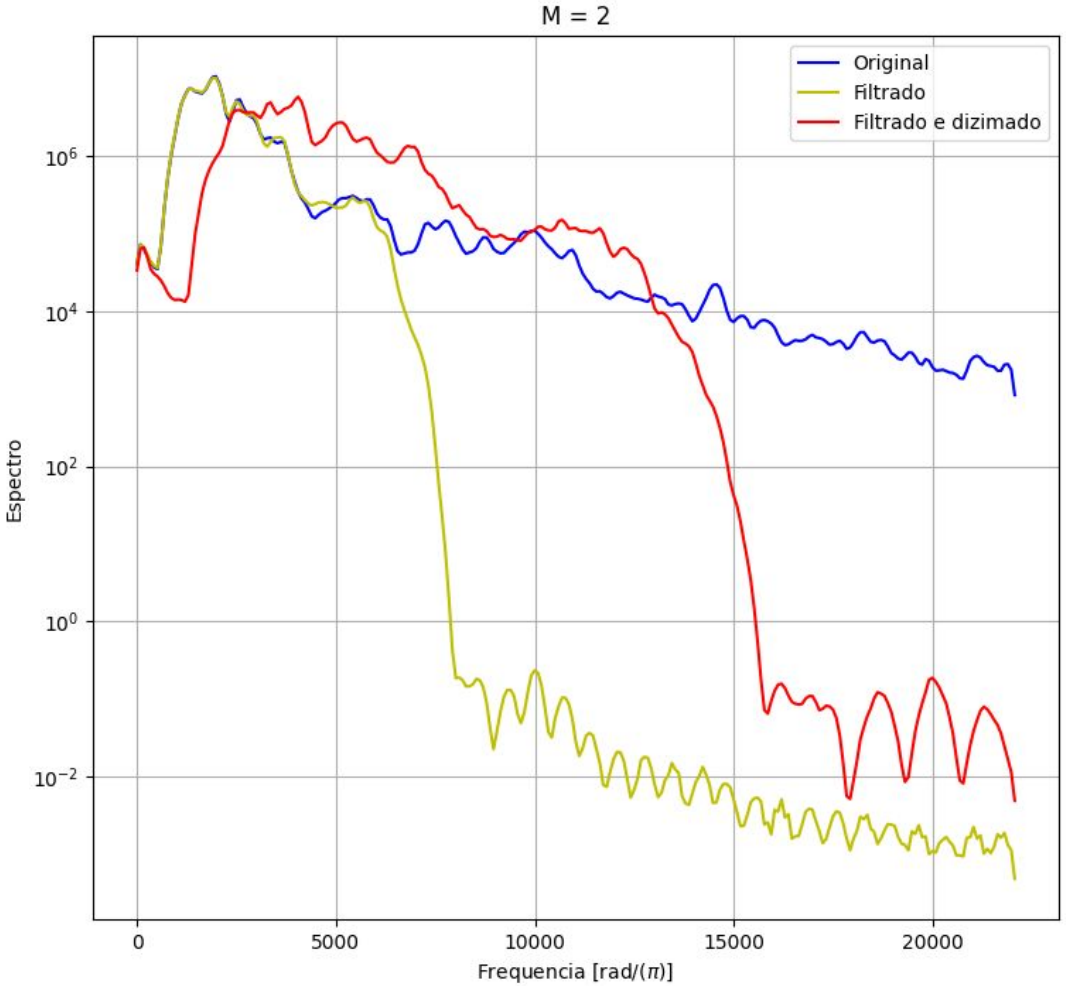
Etapa e: Dizimação utilizando *M* = 2



```
#QUESTAO E

M = 2
x = conv[0:-1:M]  # dizimação

time2 = np.linspace(0., length, x.shape[0])


plt.plot(time, data, 'b-', label='Original')
plt.plot(time, conv, 'y-', label='Filtrado')
plt.plot(time2, x, 'r-', label='Filtrado e dizimado')
plt.title(f'M = {M}')
plt.xlabel('Tempo [s]')
plt.ylabel("Amplitude")
plt.grid()
plt.legend()
plt.show()
```


M = 2
Original
Filtrado
Filtrado e dizimado
400
300
200
100
0
−100
−200
−300
Amplitude
1.054
1.055
1.056
1.057
1.058
1.059
Tempo [s]

Etapa f: Plote os conteúdos temporais e espectrais após a filtragem e após a dizimação


M = 2
Original
Filtrado
Filtrado e dizimado
Espectro
10^6
10^4
10^2
10^0
10^{-2}
0
5000
10000
15000
20000
Frequencia [rad/(π)]

```python
#QUESTAO F

f, Pxx_spec = welch(data, frequencia_amostragem, 'flattop', 512, scaling='spectrum')
f_filtered, Pxx_spec_filtered = welch(
    conv, frequencia_amostragem, 'flattop', 512, scaling='spectrum')
f_decimated, Pxx_spec_decimated = welch(
    x, frequencia_amostragem, 'flattop', 512, scaling='spectrum')

# Plota o espectro do sinal para frequencias normalizadas entre 0 1 pi
plt.semilogy(f, Pxx_spec, 'b-', label='Original')
plt.semilogy(f_filtered, Pxx_spec_filtered, 'y-', label='Filtrado')
plt.semilogy(f_decimated, Pxx_spec_decimated,
             'r-', label='Filtrado e dizimado')
plt.title(f'M = {M}')
plt.xlabel('Frequencia [rad/($\pi$)]')
plt.ylabel('Espectro')
plt.grid()
plt.legend()
plt.show()
```

Obrigado pela atenção!

Dúvidas?

Referências

- **OPPENHEIM**, A. V.; WILLSKY, A. S.; NAWAB, S. H. Processamento em tempo discreto de sinais
- Numpy
- Scipy
- pyFDA