UFRN

PROJETO FINAL

Processamento Digital de Sinais

Everton Andrade Leal Duarte
Fernando Lucas Sousa Silva
Pedro Leandro Batista Marques
Rychardson Ribeiro de Souza
Teóphilo Vitor de Carvalho Clemente

DCA

Sumário

- Introdução;
- (a) Plotagem do sinal no domínio do tempo;
- (b) Plotagem do espectro de frequências do sinal para as primeiras N amostras, utilizando a FFT;
- (c) Projeto de um filtro digital passa-baixas com resposta ao impulso finita (FIR) que corte metade do conteúdo espectral do arquivo de áudio;
- (d) Filtragem do sinal;
- (e) Interpolação do sinal por um fator L = 2, utilizando o filtro passa-baixas adequado;
- (f) Plotagem dos conteúdos temporais e espectrais após a expansão e após a interpolação.

O sinal de áudio a ser processado é: https://freesound.org/people/Hamface/sounds/98671/

DCA

Introdução

- Bibliotecas utilizadas para a realização do projeto completo:

```
# Fazer os imports das bibliotecas necessárias
import numpy as np
import soundfile as sf
import matplotlib.pyplot as plt
import matplotlib.style as stl
stl.use('default')
PI = np.pi
```

- Escolha do áudio e importando ao código:

```
# Importando o audio para o python
signal , sample_rate = sf.read('/content/sample_data/gato.wav') #sample_rate pega a frequencia do
sinal (no domínio do tempo) (ou taxa de amostragem)
period = 1/sample_rate #pois periodo é igual a 1/frequencia
```

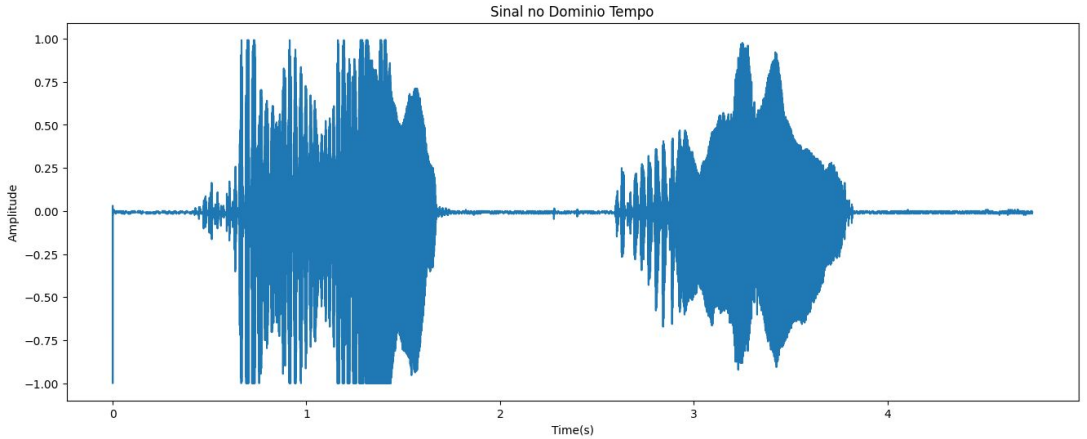
DCA

Plotagem do sinal no domínio do tempo

- Implementação:

```
#Obtendo o vetor tempo
time = np.arange(0,len(signal)*period,
period) #o vetor tempo começa em 0, e
vai até o tamanho do sinal, igualmente
espaçados pelo periodo

#Plotando sinal no dominio tempo
plt.figure(figsize=(16,6))
plt.title("Sinal no Dominio Tempo")
plt.xlabel("Tempo(s)")
plt.ylabel("Amplitude")
plt.plot(time,signal)
plt.show()
```

- Graficamente:



Link com os áudios:

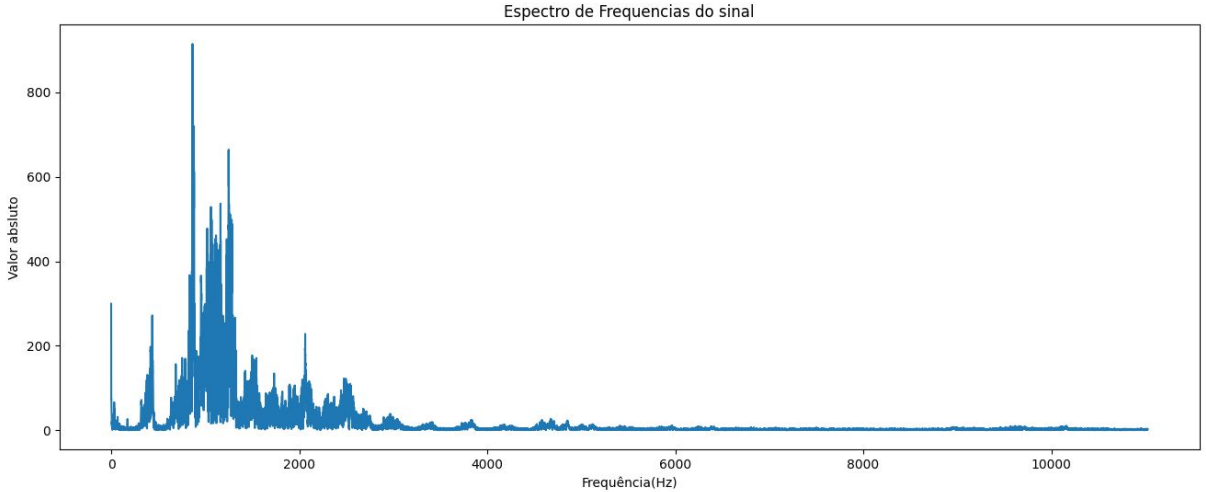
https://drive.google.com/drive/folders/1XNxDPplqk_T-rMRnoZilao8szty48TJX?usp=sharing

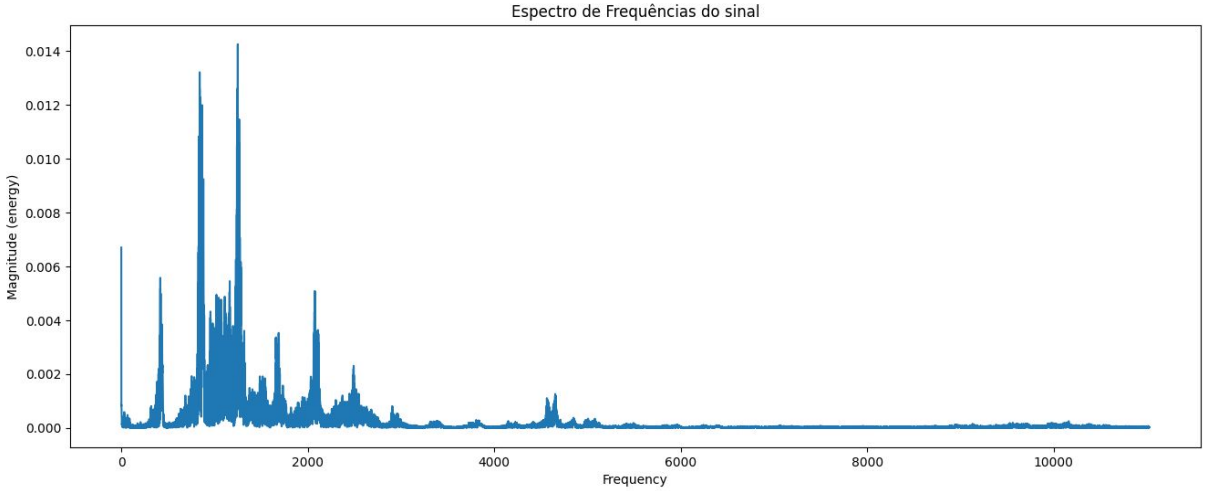
Plotagem do espectro de frequências do sinal para as primeiras N amostras, utilizando a FFT

- Implementação:

```
## pegando a fft do sinal
nfft = 50000
signal_fft = np.fft.fft(signal , nfft)
frequencias = np.fft.fftfreq(nfft,period)
mask = frequencias >= 0 #o mask vai de -pi até
+pi, então vc tá empurrando os sinal de 0 até 2pi

plt.figure(figsize=(16,6))
plt.title("Espectro de Frequencias do sinal" )
plt.xlabel("Frequência(Hz)")
plt.ylabel("Valor absluto")
plt.plot(frequencias[mask],np.abs(signal_fft[mask
]))
plt.show()
```

```
plt.figure(figsize=(16,6))
plt.title("Espectro de Frequências do sinal")
plt.magnitude_spectrum(signal,sample_rate)
plt.show()
```


Espectro de Frequencias do sinal
Valor absluto
0
200
400
600
800
0
2000
4000
6000
8000
10000
Frequência(Hz)


Espectro de Frequências do sinal
Magnitude (energy)
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
0
2000
4000
6000
8000
10000
Frequency

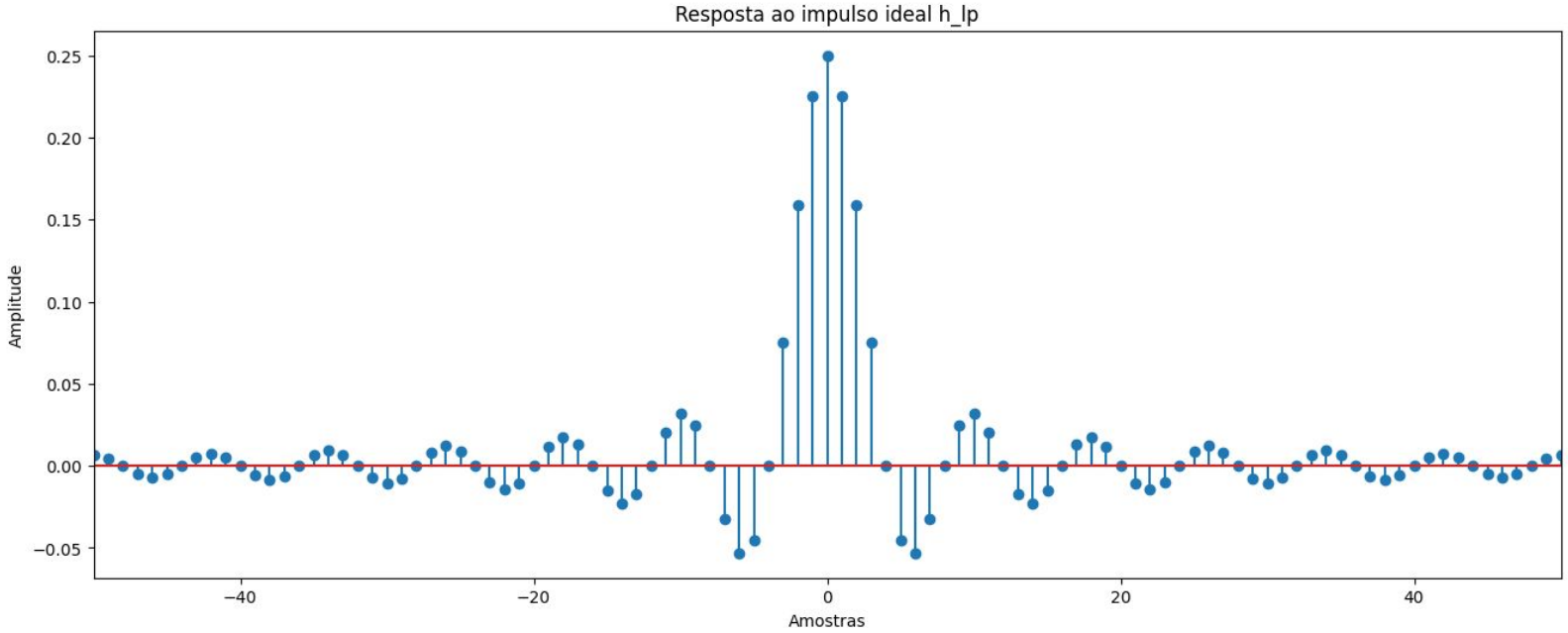
Projeto de um filtro digital passa-baixas com resposta ao impulso finita (FIR) que corte metade do conteúdo espectral do arquivo de áudio

- Implementação do filtro h_lp

```
fc = sample_rate/8 # frequência de corte (Hz)
wc = PI*fc/(sample_rate/2)
M = 101 #Número de amostras
M1 = -(M-1)/2
M2 = (M-1)/2
n1 = np.arange(M1,M2+1) # 101 amostras

#defininindo o sinc(wc*n1)
hlp = (wc/PI)*(np.sin(wc*n1))/(wc*n1)
hlp[n1 == 0] = wc/PI
```

```
#Janela retangular
plt.figure(figsize=(16,6))
plt.stem(n1,hlp)
plt.title("Resposta ao impulso ideal h_lp")
plt.xlabel("Amostras")
plt.ylabel("Amplitude")
plt.xlim((M1,M2))
plt.show()
```

Plot da resposta ao impulso ideal h_lp



Plotagem da resposta em frequência de h_lp

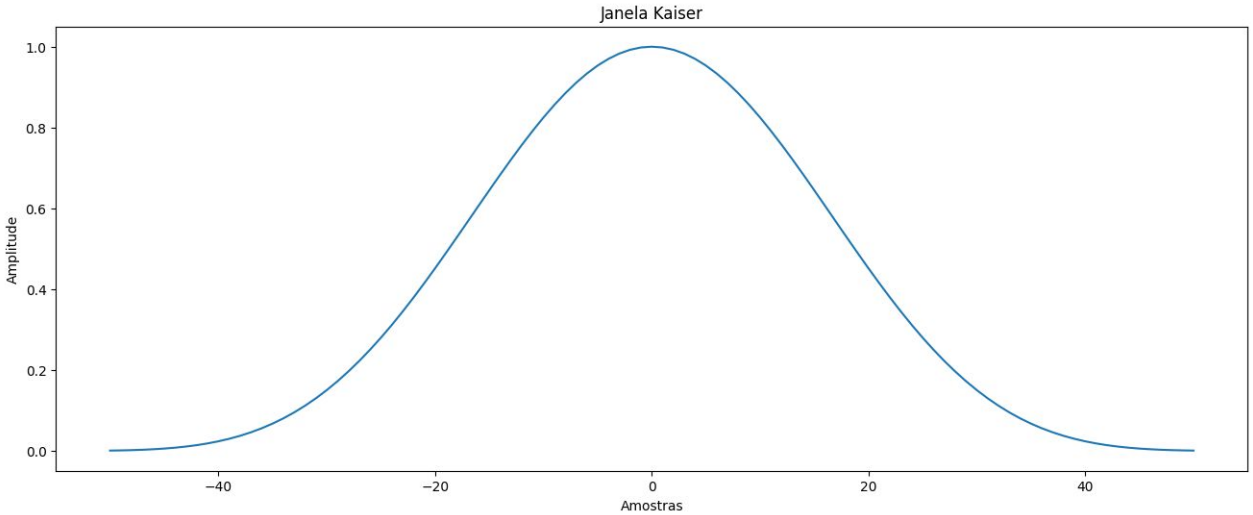
- Implementação:

```
#Janela retangular
plt.figure(figsize=(16,6))
plt.magnitude_spectrum(hlp,Fs=sample_rate)
plt.title("Resposta em frequência de H")
plt.show()
```

Janela Kaiser

- Janela de Kaiser:

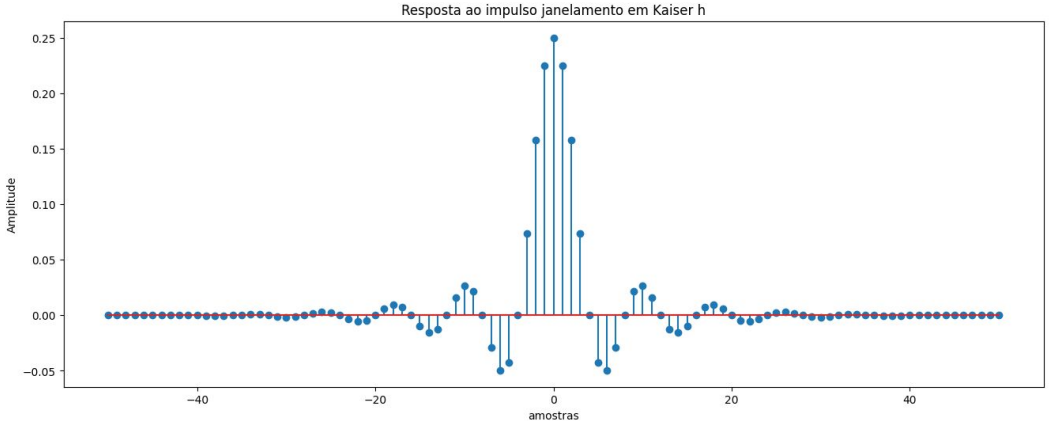
```
#Janela Kaiser
#parâmetros da janela Kaiser
r = 100
beta = 0.1102*(r - 8.7)
kaiser_window = np.kaiser(M,beta)

#Obtem o filtro hlp com a janela
Kaiser
hlp_kaiser = hlp * kaiser_window
```



Filtro com a janela Kaiser

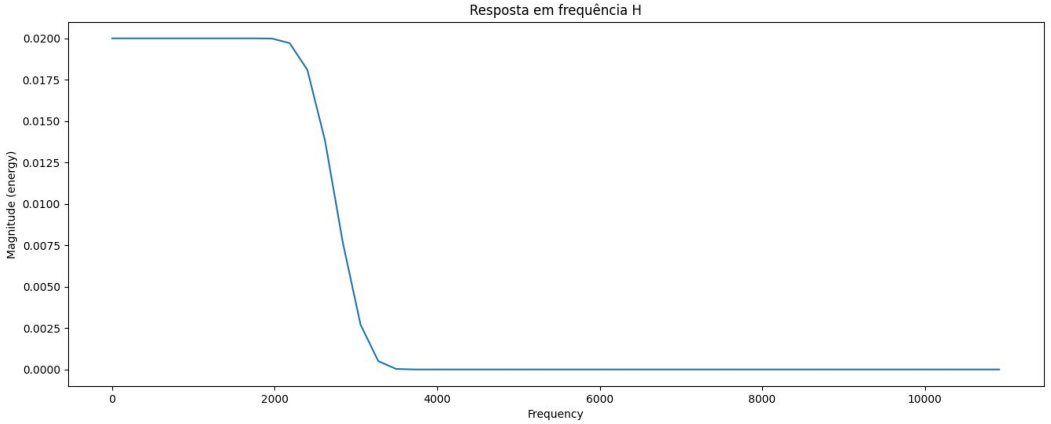
- Hlp_kaiser (Resposta ao impulso)

```
#Obtem o filtro hlp com a janela Kaiser
hlp_kaiser = hlp * kaiser_window
plt.figure(figsize=(16,6))
plt.stem(n1,hlp_kaiser)
```

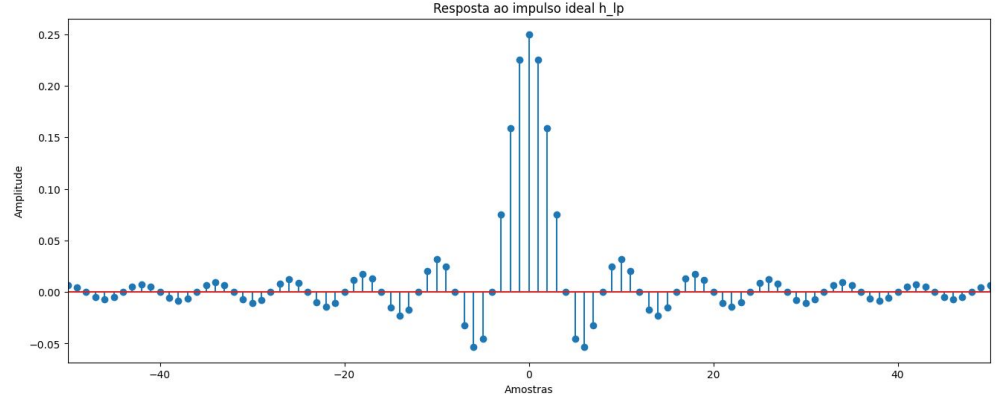


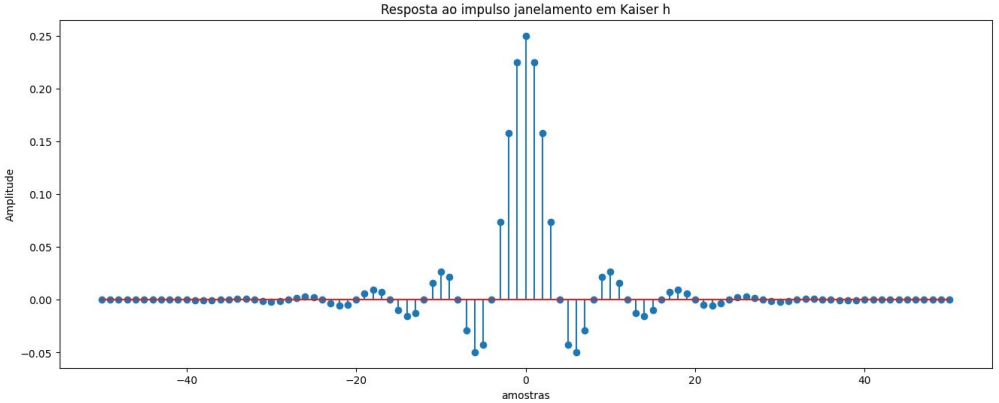
- Resposta em frequência de hlp_kaiser

```
plt.figure(figsize=(16,6))
plt.magnitude_spectrum(hlp_kaiser,Fs=sample_rate)
```

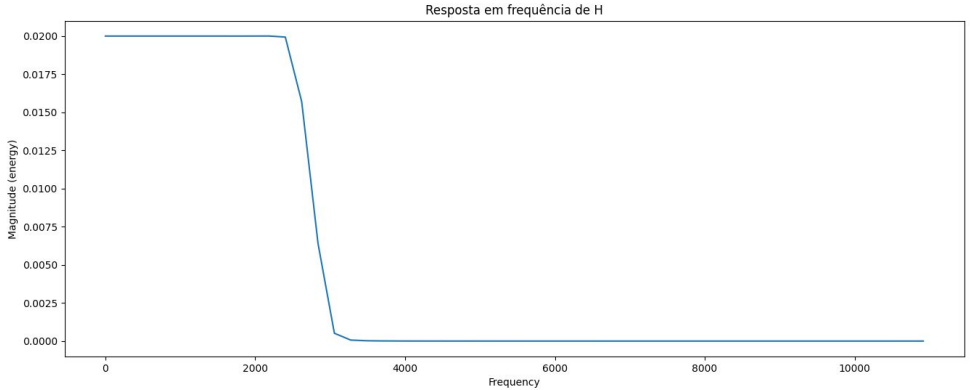


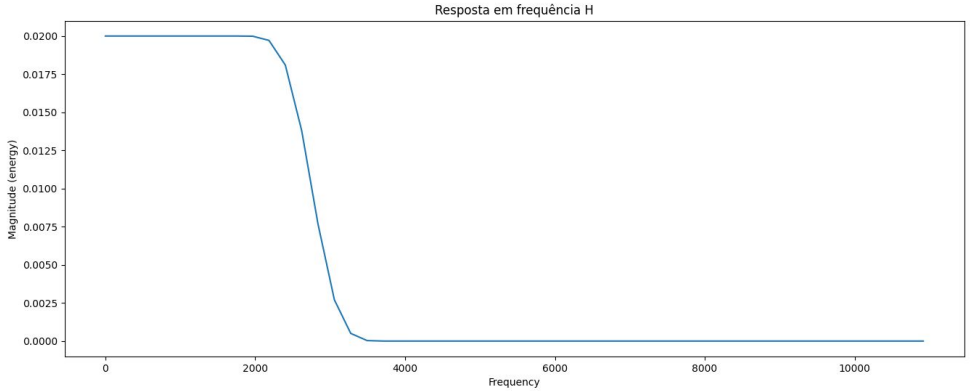
Comparativo: respostas ao impulso





Comparativo: respostas em frequência





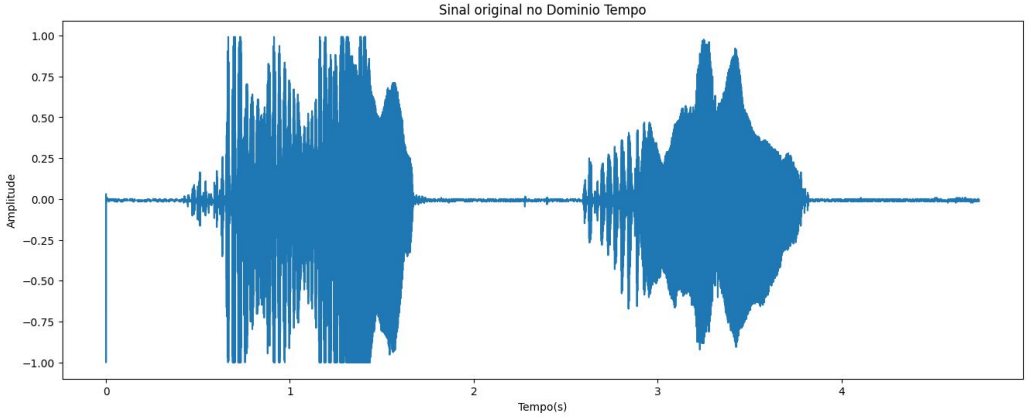
Filtragem do sinal

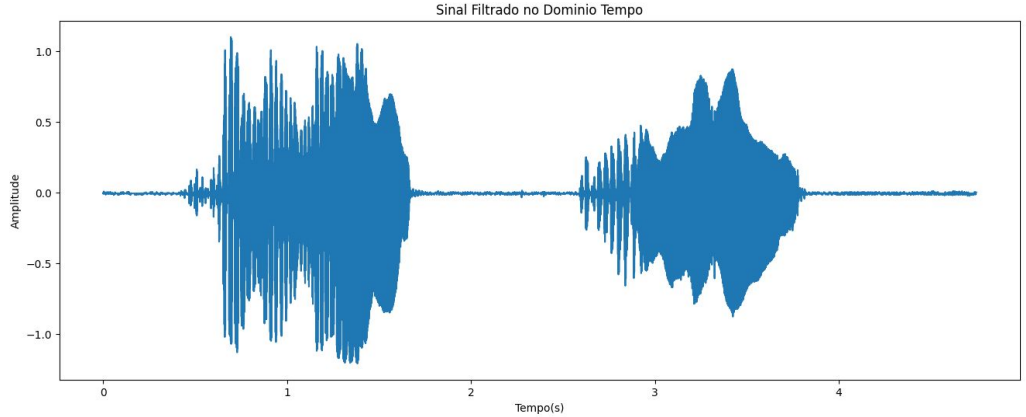
- Implementação:

```
signal_filtered = np.convolve(signal,hlp_kaiser)

#Plotando sinal no dominio tempo
plt.figure(figsize=(16,6))
plt.title("Sinal original no Dominio Tempo")
plt.xlabel("Tempo(s)")
plt.ylabel("Amplitude")
plt.plot(time,signal)
plt.show()
plt.figure(figsize=(16,6))

plt.title("Sinal Filtrado no Dominio Tempo")
plt.xlabel("Tempo(s)")
plt.ylabel("Amplitude")
plt.plot(time,signal_filtered[M-1:])
plt.show()
plt.figure(figsize=(16,6))
```

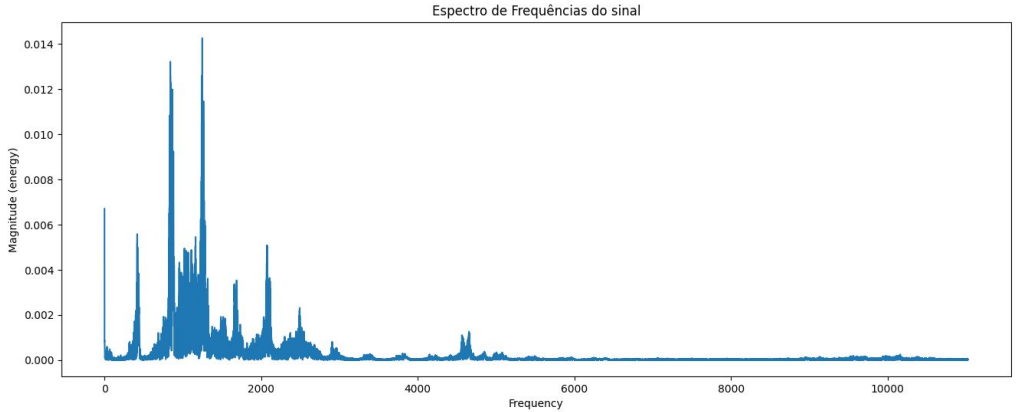
DCA

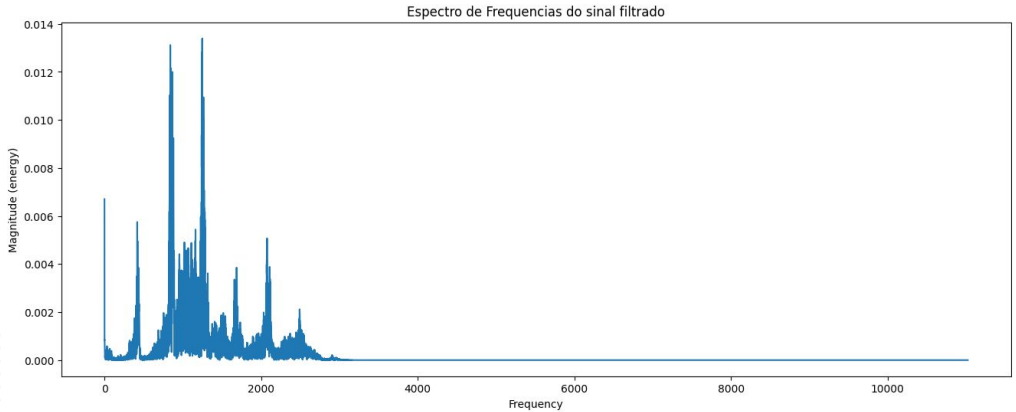
Comparativo: sinal original e filtrado (domínio tempo)





Comparativo: espectros dos sinais (original e filtrado)





Link com os áudios:

https://drive.google.com/drive/folders/1XNxDPplqk_T-rMRnoZilao8szty48TJX?usp=sharing

Interpolação do sinal por um fator L = 2, utilizando o filtro passa-baixas adequado

- Implementação:

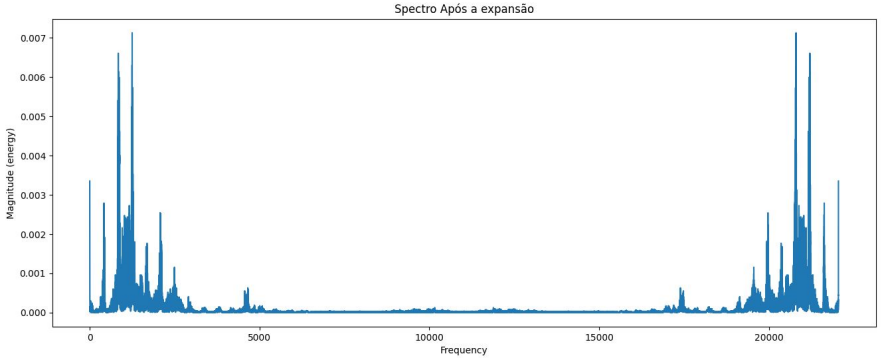
```
# Função para redefinir um FPB (reaproveita o que já fizemos no item c)
def filter_fir_kaiser_low_pass(signal,fs,fc,n_coeficientes,atenuacao):
  wc = PI*fc/(fs/2)

  M1 = -(n_coeficientes-1)/2
  M2 = (n_coeficientes-1)/2
  n1 = np.arange(M1,M2+1)

  hlp = (wc/PI)*(np.sin(wc*n1))/(wc*n1)
  hlp[n1 == 0] = wc/PI

  # Janela kaiser
  r = atenuacao
  beta = 0.1102*(r - 8.7)
  kaiser_window = np.kaiser(n_coeficientes,beta)

  #Obtem o filtro hlp com a janela Kaiser
  hlp_kaiser = hlp * kaiser_window
  signal_filtered = np.convolve(signal,hlp_kaiser) #Filtrando o sinal

  return signal_filtered[n_coeficientes-1:]
```

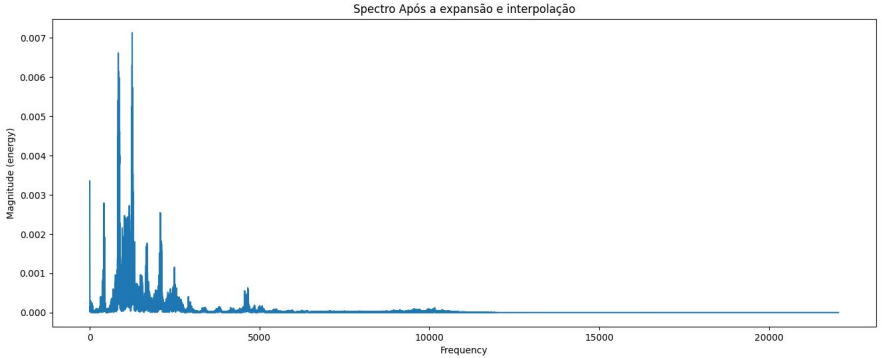
Continuação: implementação

```
L = 2
new_fs = sample_rate*2
new_period = 1/new_fs
interpolated_signal = np.zeros(len(signal)*L)
cont = 0
for i in range (0,len(interpolated_signal),L):
  if(i%L ==0):
    interpolated_signal[i] = signal[cont]
    cont+=1

interpolated_signal_filtered = filter_fir_kaiser_low_pass(interpolated_signal,new_fs,new_fs/4,101,100)
#aplicando o FBP no sinal interpolado
```

DCA

Plotagem dos conteúdos temporais e espectrais após a expansão e após a interpolação



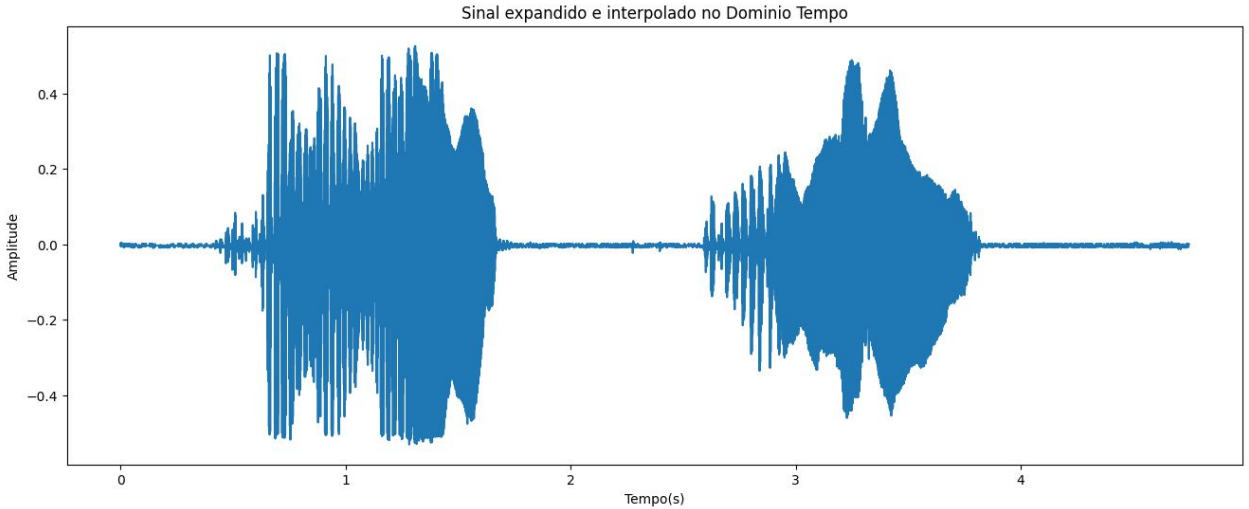


Sinal expandido e interpolado

- Implementação:

```
new_time = np.arange(0,len(interpolated_signal_filtered)*new_period , new_period)
```

- Graficamente:



Link com os áudios: https://drive.google.com/drive/folders/1XNxDPplqk_T-rMRnoZilao8szty48TJX?usp=sharing

OBRIGADO PELA ATENÇÃO!