Sétimo Relatório PCO119

João Vitor Yukio Bordin Yamashita

October 15, 2022

Todos os códigos usados podem ser encontrados no repositório[1] da disciplina.

1 Parte I – Projetos (por métodos algébricos) de filtros IIR baseados em Modelos Analógicos

1. Temos:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
import scipy
import copy
import control

# Exercicio 1
T = 0.02
fc = 1 # hz
wc = 2*np.pi*fc

a1 = T*wc - 1
b1 = T*wc

# y/x = b1*z**(-1) / (1 + a1*z**(-1)) = b1 / (z + a1)
FiltroPBZ = control.TransferFunction(b1, [1, a1], T)
FiltroPBZ

```

Listing 1: Imports e primeiro filtro

$$G_{PB}(z) = \frac{0.1257}{z - 0.8743} dt = 0.02$$

   (a) Temos:

$$a_1 = -0.8743362938564083$$
$$b_1 = 0.12566370614359174$$

   (b) Temos:

```
tsim = np.arange(0,2,0.02)
sinal = []
for t in tsim:
    sinal.append(1.5 + 0.5*np.sin(2*np.pi*10*t))
plt.plot(tsim, sinal)
plt.title("Sinal a ser filtrado")

```

Listing 2: Simulação do sinal

[1] https://github.com/JoaoYukio/PCO119/tree/main/Atividade%207

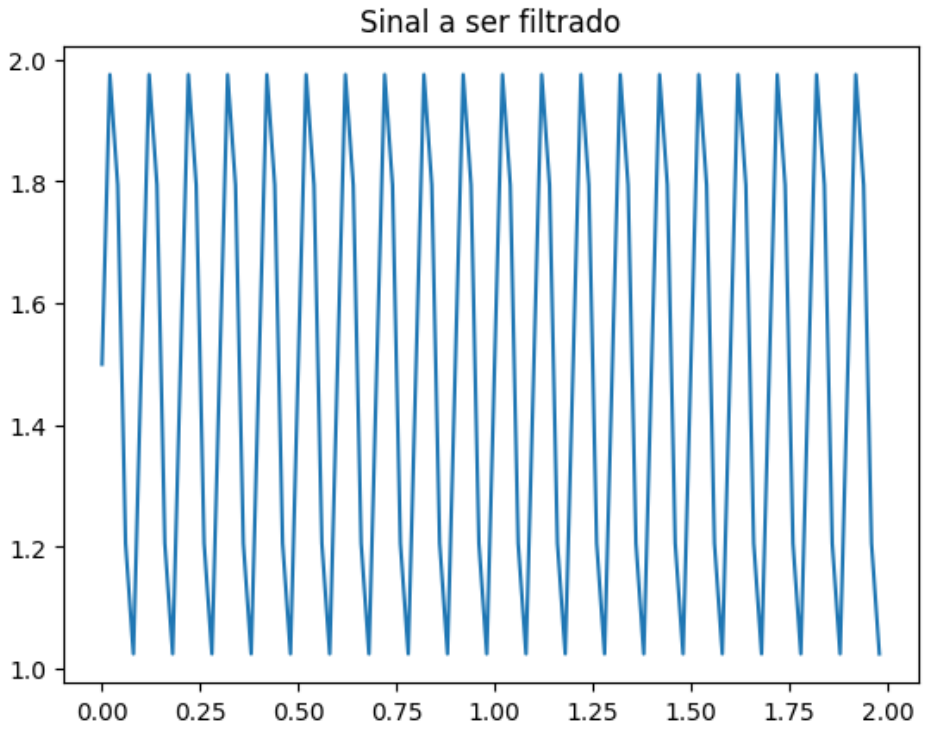

Figure 1: Sinal da simulação

(c) Temos a seguinte média do sinal de entrada:

```
mediaEnt = np.mean(sinal)
print(mediaEnt)

Output:
    1.4999999999999998

```

Listing 3: Média do sinal de entrada

(d) Filtrando o sinal:

```
# Filtrando o sinal usando os valores de a e b e a funcao lfilter
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.lfilter.
    html
# Precisa usar 1 para o a0, olhar documentacao
sinalFiltrado = signal.lfilter([b1], [1, a1], sinal)
plt.stairs(sinalFiltrado)

```

Listing 4: Filtrando o sinal de entrada

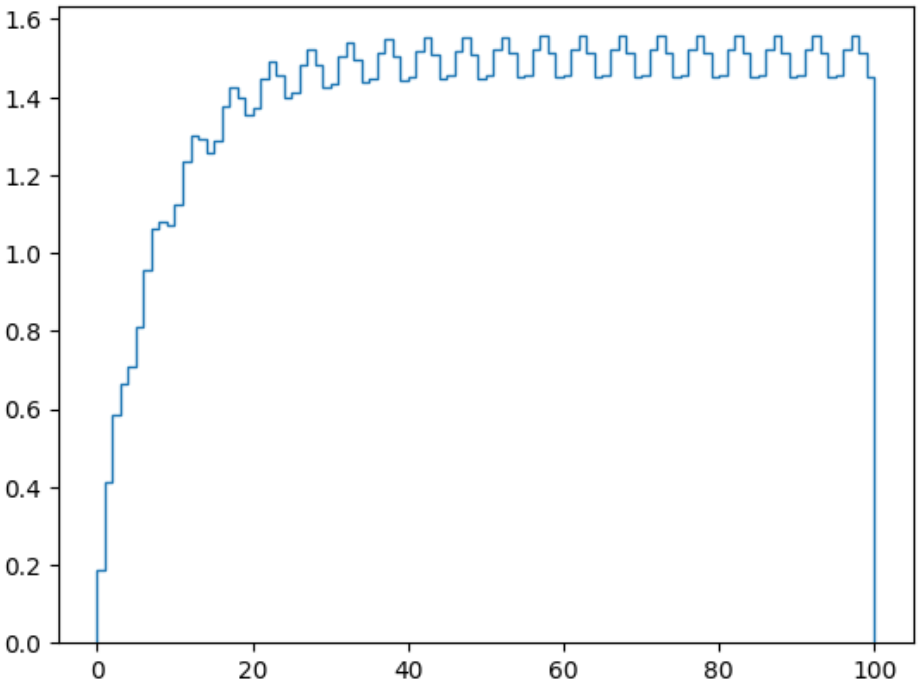

Figure 2: Sinal filtrado

(e) Temos a seguinte média do valor de saída:

```
# Calcular a media do sinalFiltrado
# Peguei os valores depois do transitorio
media = np.mean(sinalFiltrado[40:-1])
print("Media do sinal filtrado: ", media)

Output:
    Media do sinal filtrado:  1.5000324902989217

```

Listing 5: Média do sinal filtrado

(f) Considerando o filtro ideal, e que a frequência de corte é de 1Hz, na frequência de 10Hz, ou seja, uma década depois, teríamos um atenuamento de -20dB, ou seja, 1/10 da amplitude original.

(g) Não, pois ainda temos influencia do ruído na saída (aprox. 10%), deveríamos ter um filtro com frequência de corte menor ou ter um filtro com ordem maior.

2. Temos o seguinte filtro:

(a) Usando a função bilinear:

```
Wc2 = 2*np.pi*1
T = T
Fs = 1/T

num = [Wc2 ** 2]
den = [1, 2*Wc2, Wc2**2]

# Projeta um filtro usando a funcao bilinear
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.bilinear.html
# Retorna os coeficientes do filtro
b2, a2 = signal.bilinear(num, den, Fs)

```

Listing 6: Filtro 2ª ordem

(b) Temos os seguintes coeficientes:

```
#Printa os coeficientes b2 e a2
print("b2: ", b2)
print("a2: ", a2)

Output:
    b2:  [0.00349487 0.00698973 0.00349487]
    a2:  [ 1.         -1.76353041  0.77750988]

```

Listing 7: Coeficientes

(c) Temos o seguinte sinal filtrado:

```
sinalFiltrado2 = signal.lfilter(b2, a2, sinal)
plt.plot(sinalFiltrado2)

```

Listing 8: Sinal filtrado pelo filtro

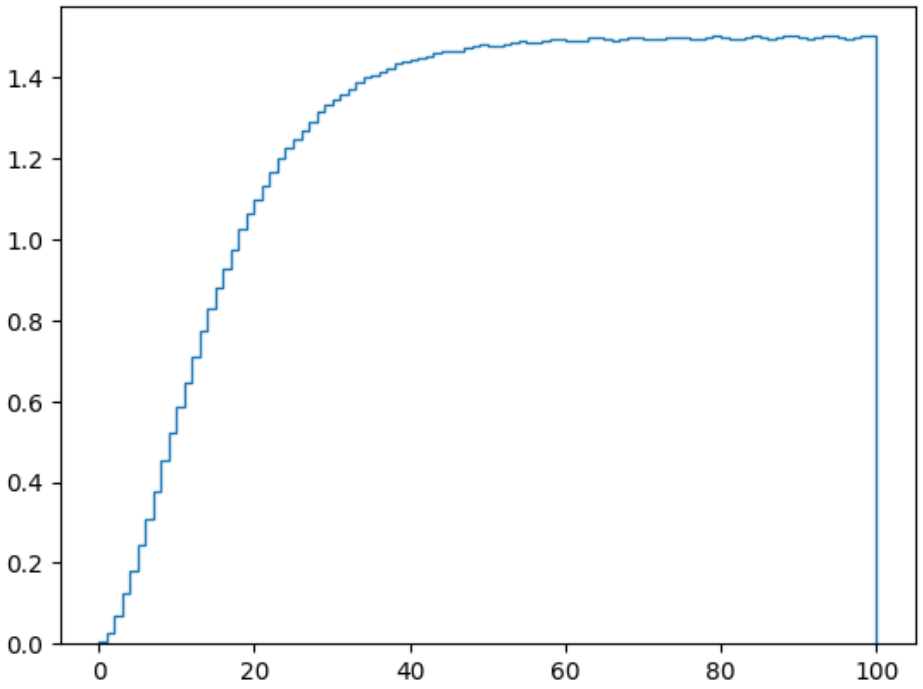

Figure 3: Sinal filtrado pelo segundo filtro

(d) Podemos comparar os filtros fazendo:

```
plt.plot(sinalFiltrado2, label="Filtro segunda ordem")
plt.plot(sinalFiltrado, label="Filtro primeira ordem")
plt.legend()

```

Listing 9: Comparação da resposta dos filtros

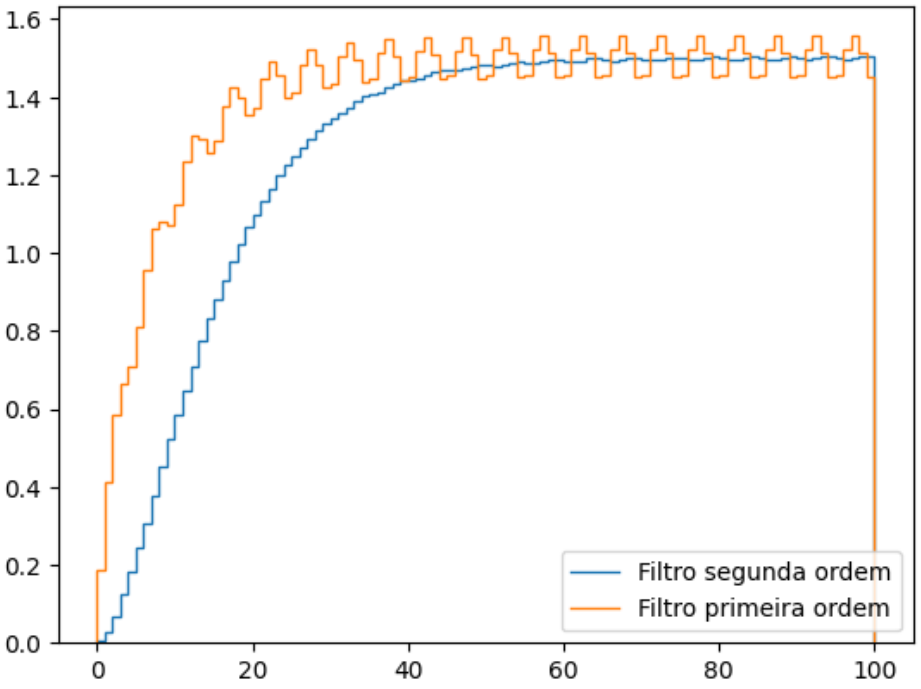

Figure 4: Comparação dos filtros

Podemos ver que temos bem menos oscilação no sinal, isso se deve ao fato da atenuação do segundo filtro ser de -40dB/dec, por isso temos uma atenuação de 0.01 no sinal, ou seja, 99% do sinal de 10Hz foi eliminado. Podemos ver isso observando a FFT do sinal:

```
# https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fft.html
N = len(sinalFiltrado)
T = T
yf = scipy.fft.fft(sinalFiltrado)
xf = np.linspace(0.0, 1.0/(2.0*T), N//2)

plt.plot(xf, 2.0/N * np.abs(yf[0:N//2]), color = 'blue')
plt.title("Sinal com filtro de primeira ordem")
plt.grid()
plt.show()

N = len(sinalFiltrado2)
T = T
yf = scipy.fft.fft(sinalFiltrado2)
xf = np.linspace(0.0, 1.0/(2.0*T), N//2)

plt.plot(xf, 2.0/N * np.abs(yf[0:N//2]), color = 'red')
plt.title("Sinal com filtro de segunda ordem")
plt.grid()
plt.show()

```

Listing 10: FFT dos sinais filtrados

Figure 5: FFT do primeiro filtro

Figure 6: FFT do segundo filtro

Podemos perceber na figura 6 que na frequência de 40Hz basicamente não temos um sinal, mas na 5 temos um sinal bem mais aparente na mesma frequência.

3. Temos o seguinte filtro:

```
Q = 8
W3 = 2*np.pi*100
T2 = 0.0001
Ns = [0,(W3/Q),0]
Ds = [1,(W3/Q), (W3**2)]

```

Listing 11: Parametros do filtro

(a) Calculando os coeficientes de forma manual, temos:

```
#Calculo manual
K = W3/Q
C0 = 1 + K*T2/2 + ((W3*T2)**2)/4
a12= (((W3*T2)**2)/2 -2)/C0
a22 = (1- (K*T2/2) + ((W3*T2)**2)/4)/C0
b00 = ((K*T2)/2)/C0
b10 = ((-K*T2)/2)/C0

#printar as variaveis acima
print("a12: ", a12)
print("a22: ", a22)
print("b00: ", b00)
print("b10: ", b10)
Output:
    a12:  -1.988255886606364
    a22:  0.9921844237268778
    b00:  0.0039077881365611215
    b10:  -0.0039077881365611215

```

Listing 12: Coeficientes do filtro

(b) Usamos a função bilinear, já que ela já utiliza o método de Tustin para conversão. Temos os seguintes coeficientes:

```
b3,a3 = signal.bilinear(Ns, Ds, 1/T2)

```

Listing 13: Coeficientes do filtro

(c) Temos:

```
print("b3: ", b3)
print("a3: ", a3)

Output:
    b3:  [ 0.00390779  0.         -0.00390779]
    a3:  [ 1.         -1.98825589  0.99218442]

```

Listing 14: Coeficientes do filtro

(d) São exatamente os mesmos. Isso se deve pelo fato que foi utilizada a mesma aproximação para discretizar o sistema.

(e) Para simulação, temos:

```
tsim2 = np.arange(0,0.1,T2)
sinal2 = []
for t in tsim2:
    sinal2.append(np.sin(2*np.pi*100*t) + 0.2*np.sin(2*np.pi*60*t) + 0.3*np.
    sin(2*np.pi*1000*t))

plt.plot(tsim2, sinal2)

```

Listing 15: Simulação do sinal

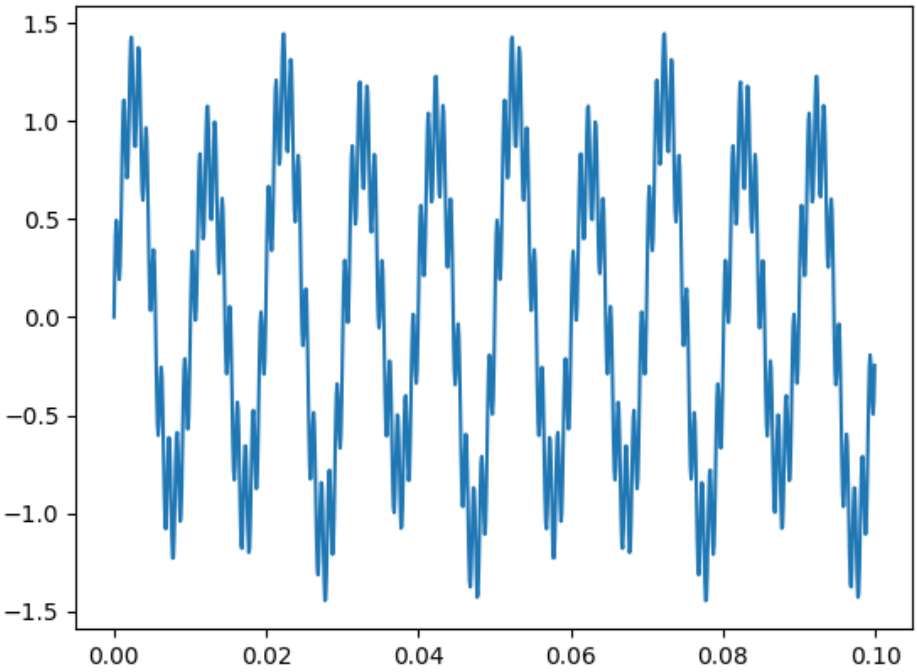

Figure 7: Segundo sinal

Para filtrar o sinal temos:

```
sinalFiltrado3 = signal.lfilter(b3, a3, sinal2)
plt.stairs(sinalFiltrado3)

```

Listing 16: Filtrando o sinal

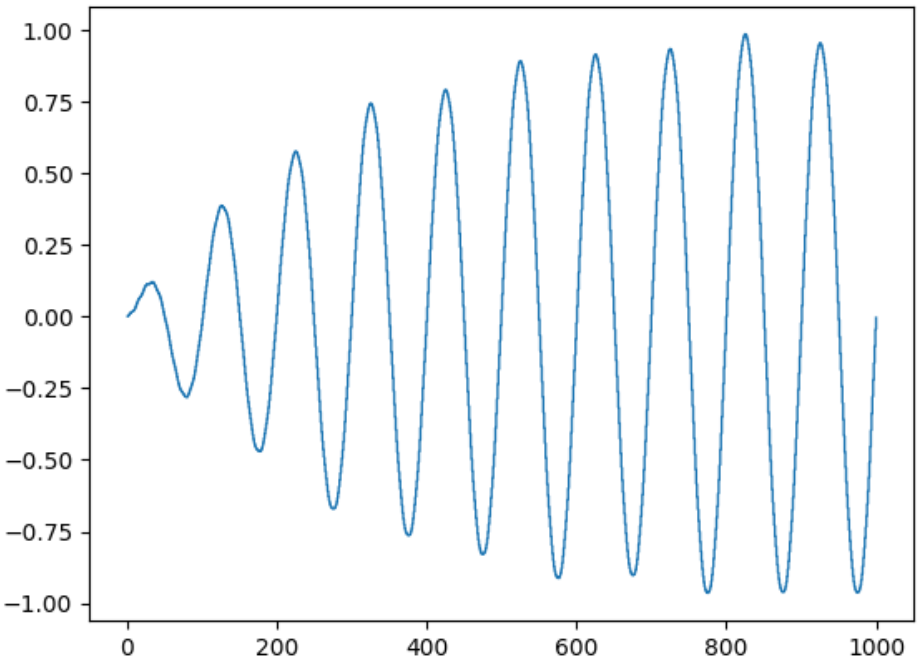

Figure 8: Segundo sinal filtrado

(f) Temos a seguinte FFT do sinal filtrado:

```
N = len(sinalFiltrado3[600:-1])
T = T2
yf = scipy.fft.fft(sinalFiltrado3[600:-1])
xf = np.linspace(0.0, 1.0/(2.0*T), N//2)

plt.plot(xf[0:42], 2.0/N * np.abs(yf[0:42]))
plt.grid()
plt.show()

```

Listing 17: FFT do sinal

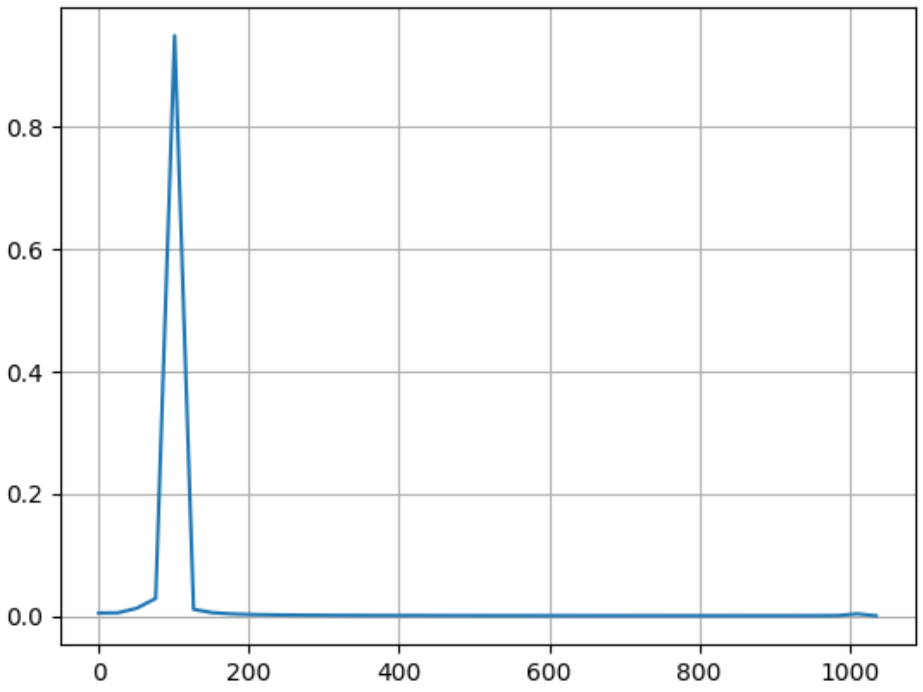

Figure 9: FFT do sinal filtrado

Podemos ver que o sinal de 100Hz é o mais predominante e tem um modulo de aproximadamente 1, mostrando que o sinal original não foi muito atenuado, além disso, conseguimos ver que na frequência de 1000Hz temos um modulo bem pequeno e na frequência de 60Hz temos um modulo ligeiramente maior, mas ainda assim bem menor do que o valor original.

2 Parte II – Projetos de filtros IIR utilizando comandos específicos do MATLAB

1. Temos:

```
Ts = 0.0001
frange = [80,120] # Banda passante

```

Listing 18: Parâmetros do problema

(a) Temos o seguinte filtro:

```
b4,a4 = scipy.signal.butter(2, frange, btype='bandpass', analog=False, fs =
    1/Ts)

```

Listing 19: Filtro

(b) Temos os seguintes parâmetros:

```
print('b: ', b4)
print('a: ', a4)

Output:
    b:  [ 0.00015515  0.         -0.0003103   0.          0.00015515]
    a:  [ 1.         -3.95695005  5.87912768 -3.8872447   0.96508117]

```

Listing 20: Parâmetros do Filtro

(c) Temos a seguinte resposta em frequência do filtro:

```
#Calcular a frequencia do sinal filtrado
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.freqz.
    html
w, h = signal.freqz(b4, a4)
plt.plot(w*(1/Ts)*0.1592, 20 * np.log10(abs(h)))
plt.xscale('log')
plt.title('Filtro passa faixa')
plt.xlabel('Frequencia [Hz]')
plt.ylabel('Amplitude [dB]')
plt.margins(0, 0.1)
plt.grid(which='both', axis='both')
plt.show()

```

Listing 21: Resposta em frequência do Filtro

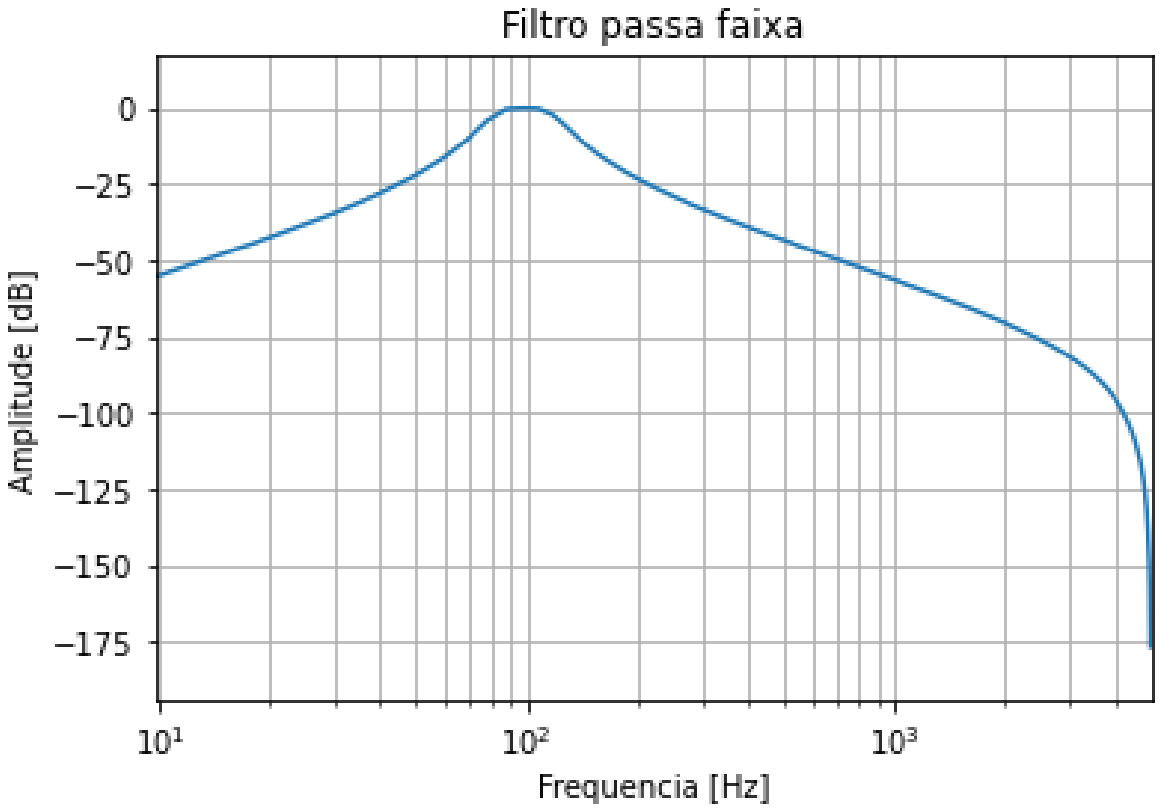

Figure 10: Resposta em frequência do filtro passa faixa

(d) Temos o seguinte sinal filtrado por esse filtro:

```
sinalFiltrado4 = signal.lfilter(b4, a4, sinal2)
plt.stairs(sinalFiltrado4)
plt.grid()

```

Listing 22: Sinal filtrado

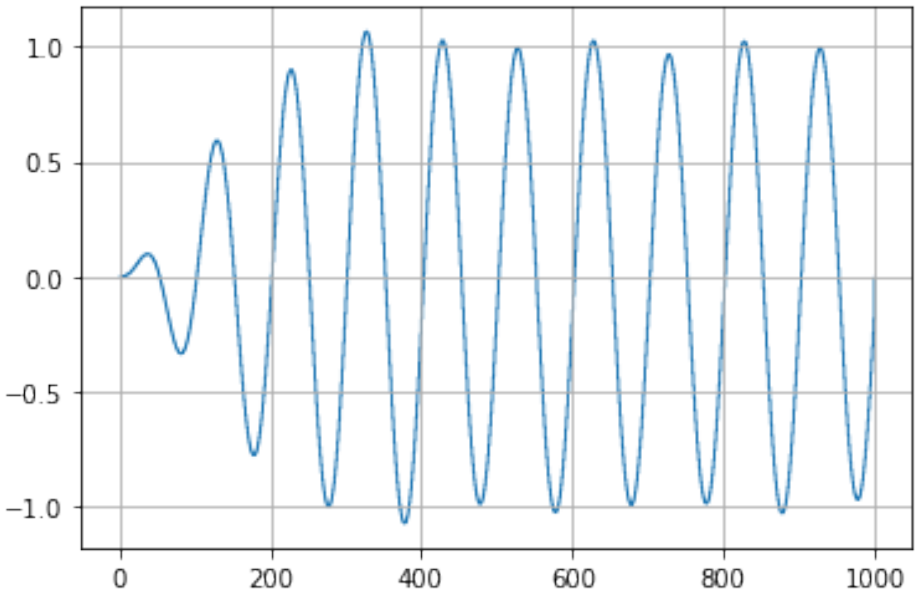

Figure 11: Sinal Filtrado

(e) Temos o seguinte espectro do sinal filtrado:

```
N = len(sinalFiltrado4[400:-1])
T = T2
yf = scipy.fft.fft(sinalFiltrado4[400:-1])
xf = np.linspace(0.0, 1.0/(2.0*T), N//2)

plt.plot(xf[0:65], 2.0/N * np.abs(yf[0:65]))
plt.grid()
plt.show()

```

Listing 23: Espectro do Sinal filtrado

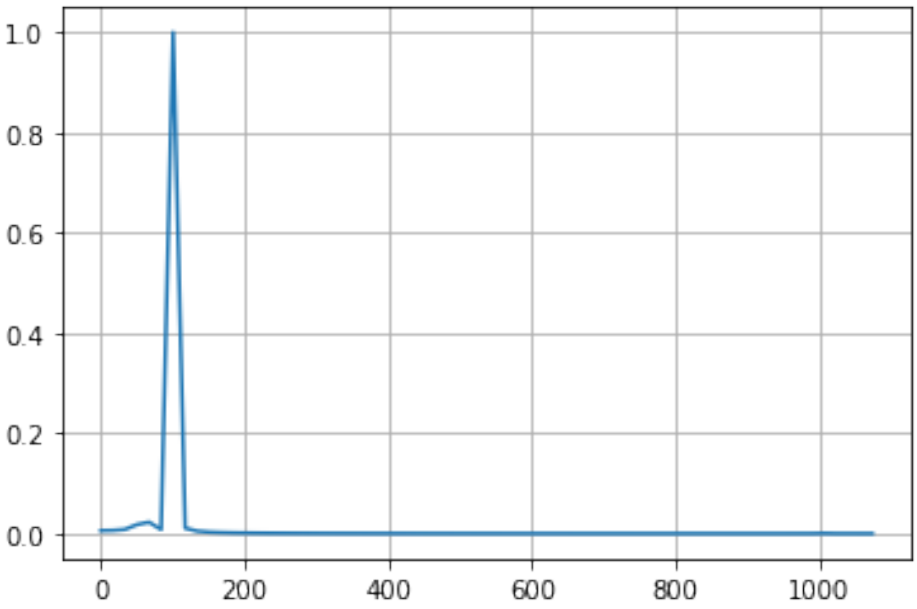

Figure 12: Espectro do sinal filtrado

Podemos ver que basicamente não temos perda do sinal desejado, visto que temos uma amplitude unitária para frequência de 100 Hz e não temos sinal em 1000 Hz, usando o zoom podemos ver na figura 13 também que temos um pequeno sinal na frequência de 60 Hz mas que foi bem atenuado.

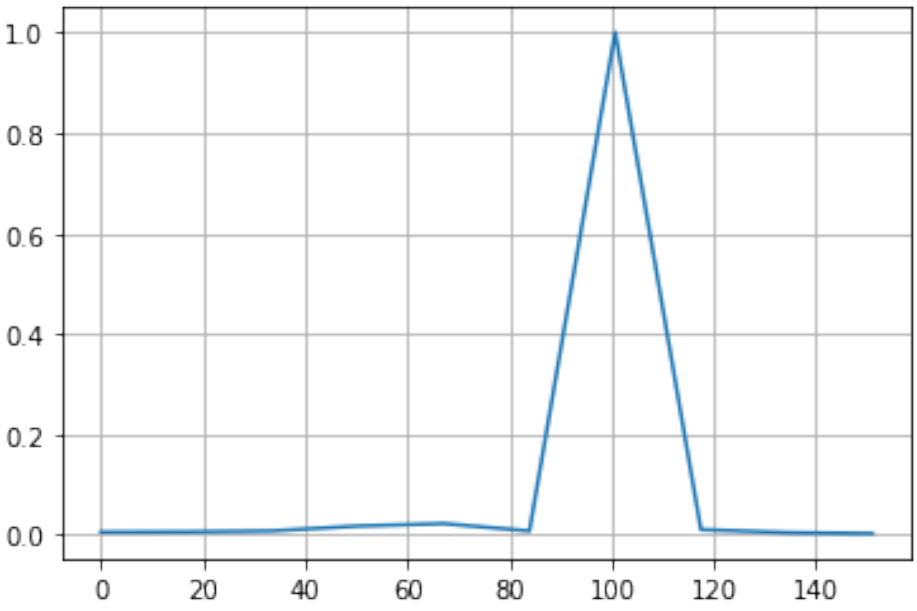

Figure 13: Espectro do sinal filtrado

2. Temos:

```
#https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.buttord.html
minOrd, Wn = scipy.signal.buttord([80/5000,120/5000], [60/5000,140/5000],1,10, fs
    = 1/Ts)

```

Listing 24: Cálculo da ordem do filtro

(a) Fazendo:

```
print(minOrd)

Output:
    4

```

Listing 25: Ordem mínima do filtro

(b) Temos os seguintes coeficientes para o filtro:

```
b5,a5 = scipy.signal.butter(minOrd, Wn, btype= 'bandpass')

```

Listing 26: Coeficientes do filtro

(c) Temos os seguintes coeficientes:

$$b_5 = array([0.00015515, 0., -0.0003103, 0., 0.00015515])$$

$$a_5 = array([1., -3.95695005, 5.87912768, -3.8872447, 0.96508117])$$

(d) Temos a seguinte resposta em frequência:

```
#Calcular a frequencia do sinal filtrado
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.freqz.
    html
w, h = signal.freqz(b5, a5)
plt.plot(w*(1/Ts)*0.1592, 20 * np.log10(abs(h)))
plt.xscale('log')
plt.title('Filtro passa faixa')
plt.xlabel('Frequencia [Hz]')
plt.ylabel('Amplitude [dB]')
plt.margins(0, 0.1)
plt.grid(which='both', axis='both')
plt.show()

```

Listing 27: Resposta em frequência do filtro

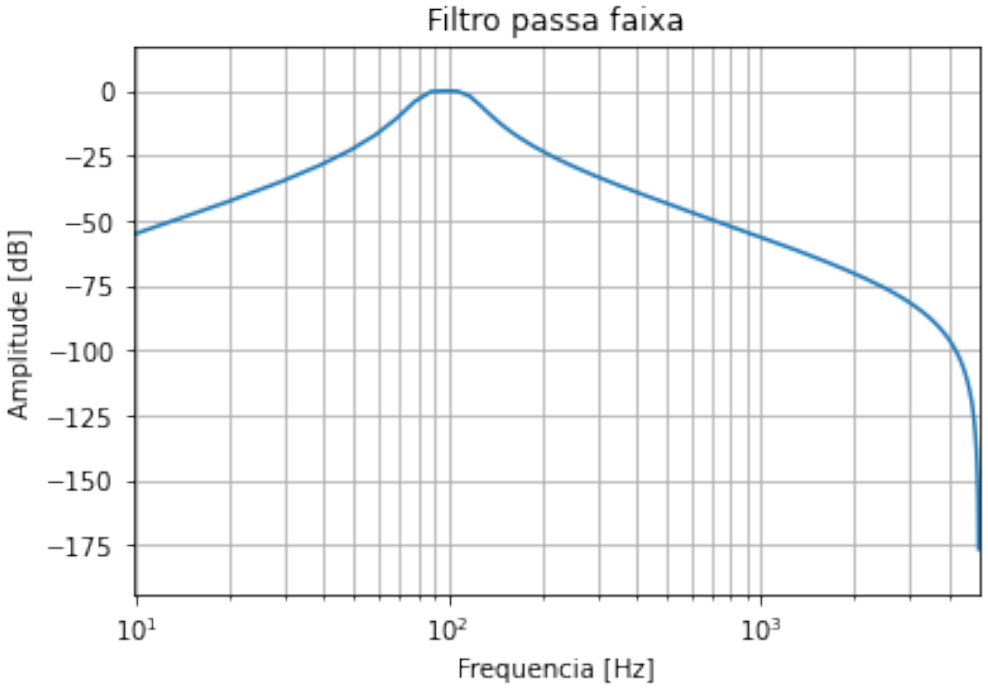

Figure 14: Resposta em frequência do filtro

(e) Temos o seguinte sinal filtrado:

```
sinalFiltrado5 = signal.lfilter(b5, a5, sinal2)

plt.stairs(sinalFiltrado5)
plt.grid()

```

Listing 28: Sinal Filtrado

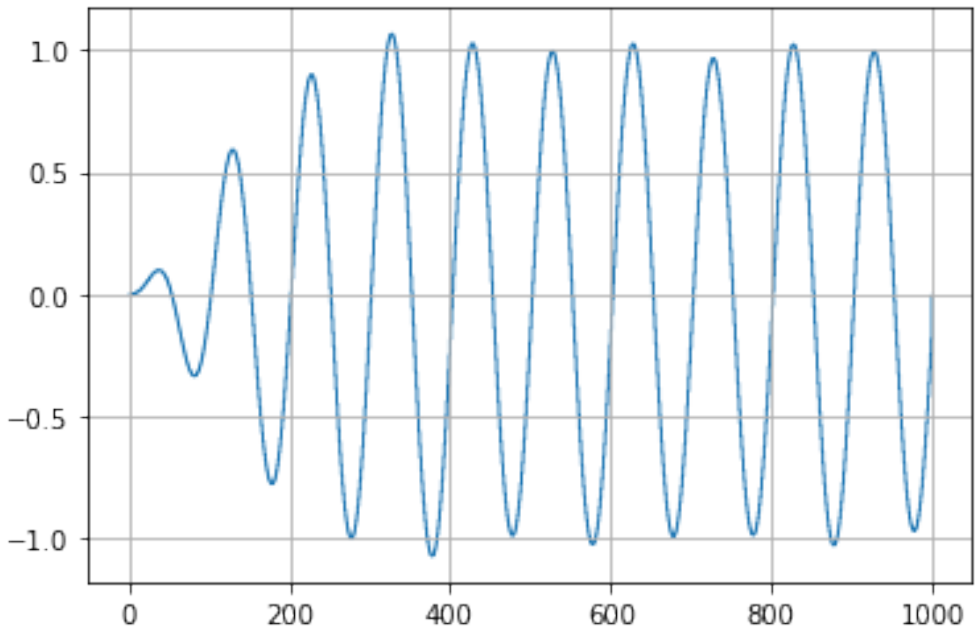

Figure 15: Saída do filtro

(f) Temos o seguinte espectro do sinal filtrado:

```
# Pegando o sinal filtrado depois do transitorio
N = len(sinalFiltrado5[400:-1])
T = T2
yf = scipy.fft.fft(sinalFiltrado5[400:-1])
xf = np.linspace(0.0, 1.0/(2.0*T), N//2)

plt.plot(xf[0:62], 2.0/N*np.abs(yf[0:62]))
plt.grid()
plt.show()

```

Listing 29: Espectro Sinal Filtrado

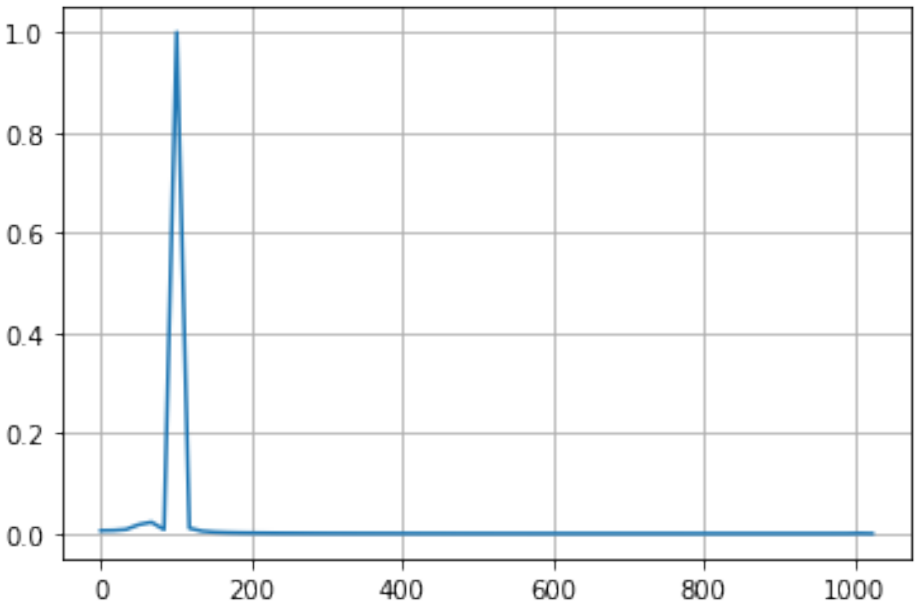

Figure 16: Espectro do sinal filtrado

De forma semelhante ao item anterior podemos ver que o sinal de interesse foi mantido e as frequências de ruído foram eliminadas.

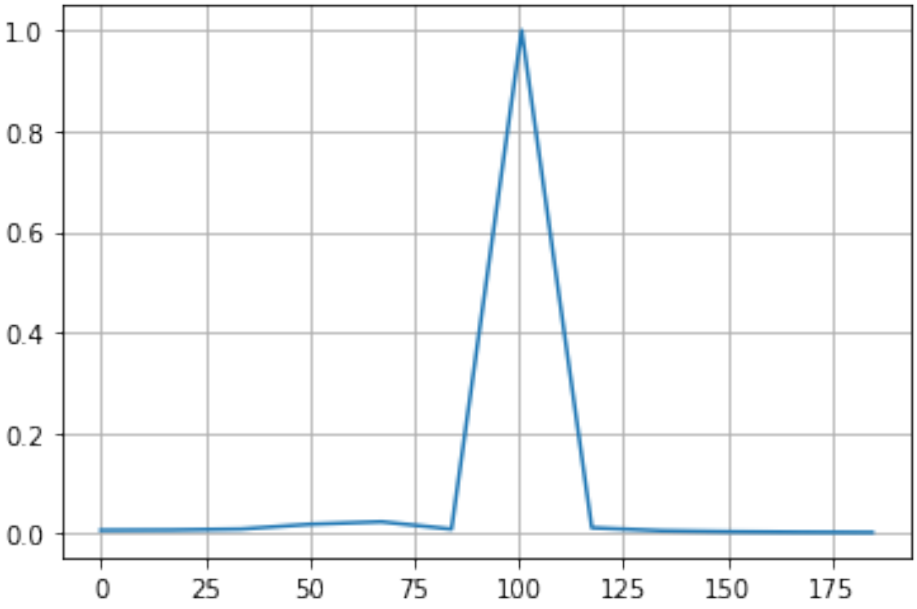

Figure 17: Espectro do sinal filtrado

3. Temos o seguinte filtro:

```
# Filtro eliptico
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.ellipord.html
minOrd, Wn = scipy.signal.ellipord([80/5000,120/5000], [60/5000,140/5000],1,10, fs
     = 1/Ts)

#https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.ellip.html
b6,a6 = scipy.signal.ellip(minOrd, 1, 10, Wn, btype= 'bandpass')
```

```

```

Listing 30: Filtro

(a) Temos a seguinte resposta em frequência para esse filtro:

```
#Calcular a frequencia do sinal filtrado
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.freqz.
    html
w, h = signal.freqz(b6, a6)
plt.plot(w*(1/Ts)*0.1592, 20 * np.log10(abs(h)), label ='Eliptico')
plt.xscale('log')
plt.title('Filtro passa faixa')
plt.xlabel('Frequencia [Hz]')
plt.ylabel('Amplitude [dB]')
plt.margins(0, 0.1)
plt.grid(which='both', axis='both')
plt.legend()
plt.show()

```

Listing 31: Resposta em frequência do Filtro

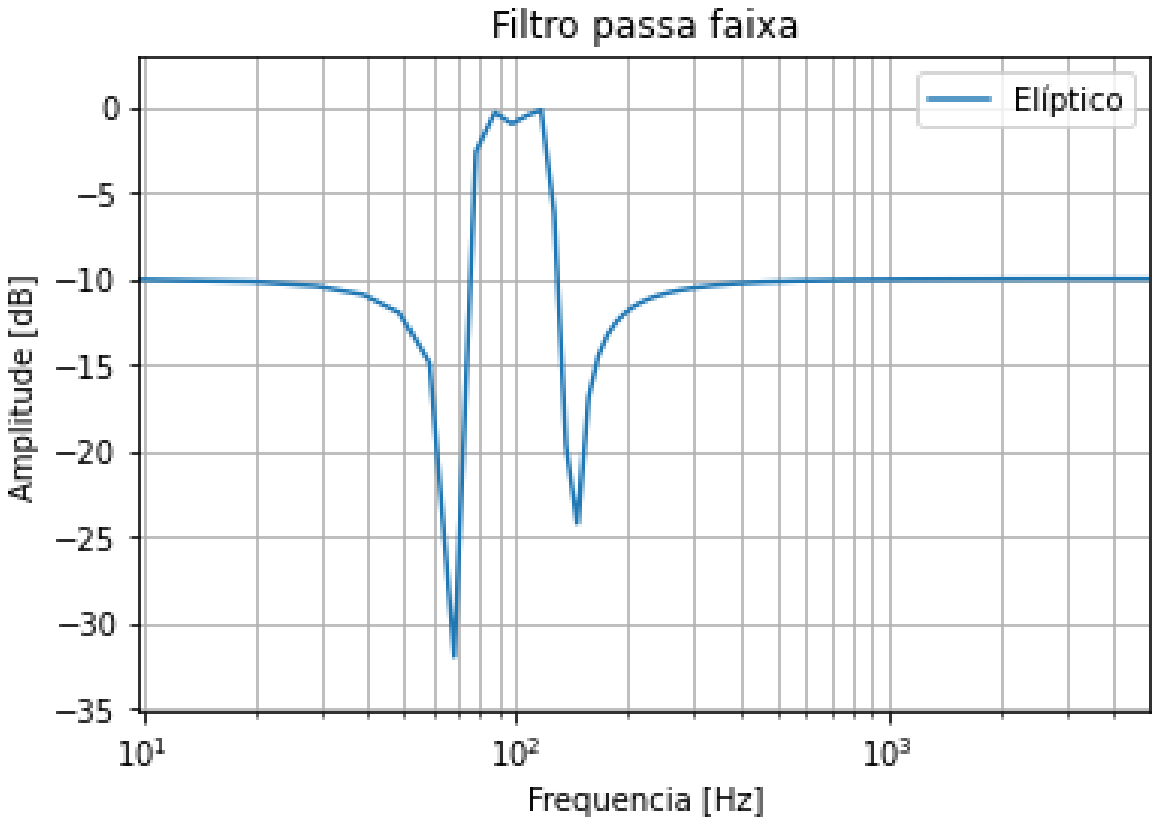

Figure 18: Resposta em frequência do filtro

(b) Podemos comparar a resposta em frequência dos dois filtros fazendo:

```
#Calcular a frequencia do sinal filtrado
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.freqz.
    html
w, h = signal.freqz(b6, a6)
plt.plot(w*(1/Ts)*0.1592, 20 * np.log10(abs(h)), label ='Elifptico')

```

```
w1, h1 = signal.freqz(b4, a4)
plt.plot(w1*(1/Ts)*0.1592, 20 * np.log10(abs(h1)), color = 'orange')

plt.xscale('log')
plt.title('Filtro passa faixa')
plt.xlabel('Frequencia [Hz]')
plt.ylabel('Amplitude [dB]')
plt.margins(0, 0.1)
plt.grid(which='both', axis='both')
plt.legend()
plt.show()


```

Listing 32: Comparação resposta dos filtros

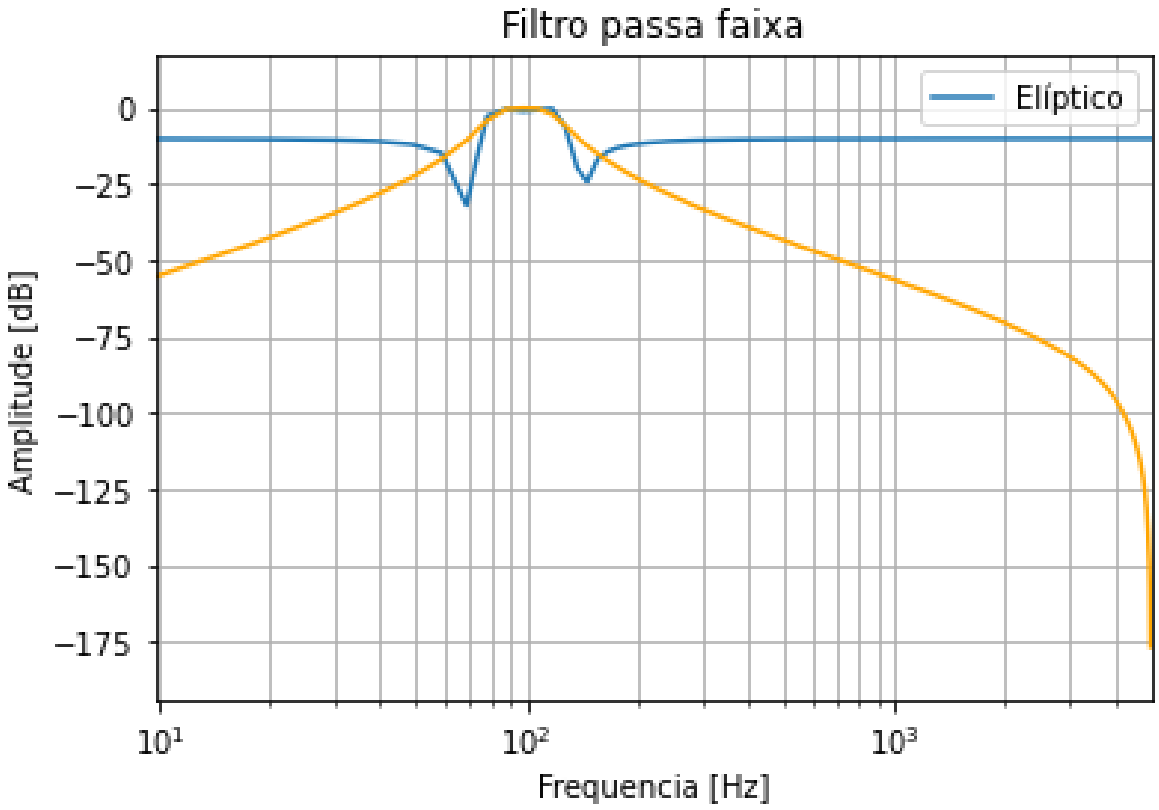

Figure 19: Comparação dos filtros

Podemos ver na figura 19 que o filtro elíptico apresenta uma atenuação melhor nas bandas perto da banda de passagem, o que pode ser útil em casos que o ruído esteja no limite da banda de passagem, desta forma ele é atenuado mesmo nesse caso, se não for nesse caso o filtro do item (b) do item anterior apresenta, no geral uma atenuação melhor.

3 Parte III – Projeto e simulação de filtros digitais FIR

1. Temos:

```
numCoef = 51
fsample = 200
srate = 1/fsample

filtkern = signal.firwin(numCoef, cutoff = 40, window = 'blackman', fs = fsample)

```

Listing 33: Projeto do filtro

(a) Temos:

```
        plt.plot((40/100)*np.sinc((40/100)*np.arange(-25,25,1)))

```

Listing 34: Sinc

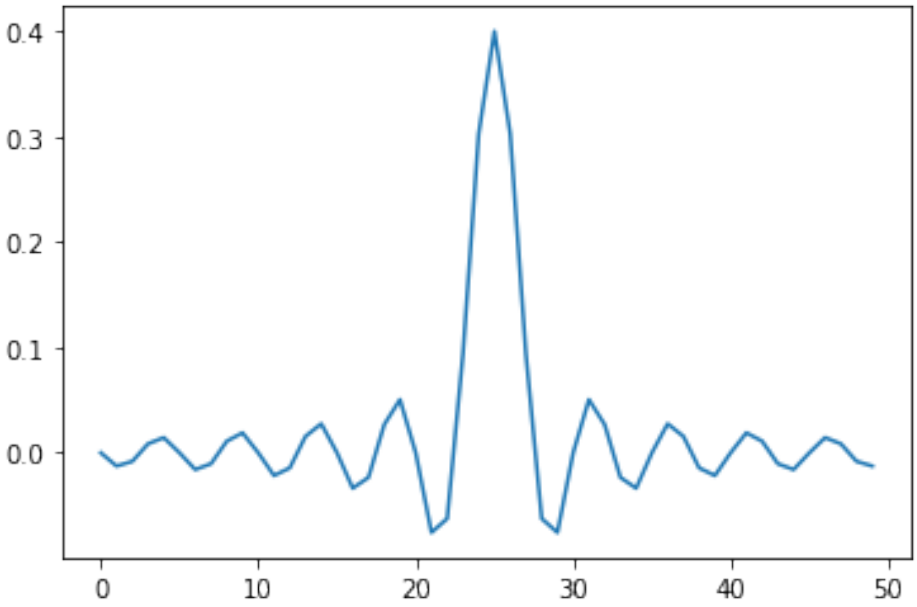

Figure 20: Sinc

(b) Temos:

```
        plt.plot(np.blackman(50))

```

Listing 35: Janela

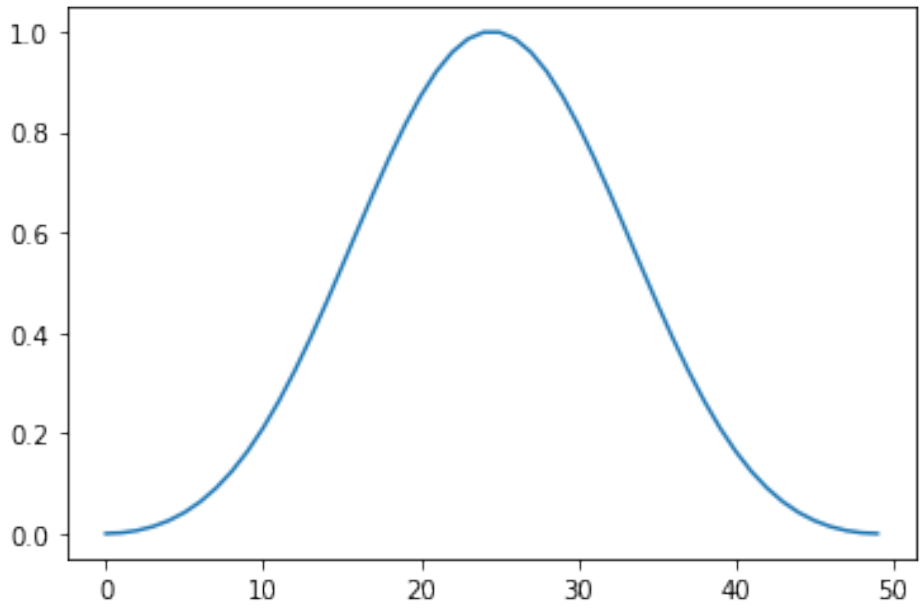

Figure 21: Janela Blackman

(c) Podemos também, obter o kernel do filtro de forma manual, fazendo:

```
kernel =((40/100)*np.sinc((40/100)*np.arange(-25,25,1)))*np.hamming(50)
plt.plot(kernel)

```

Listing 36: Kernel manual

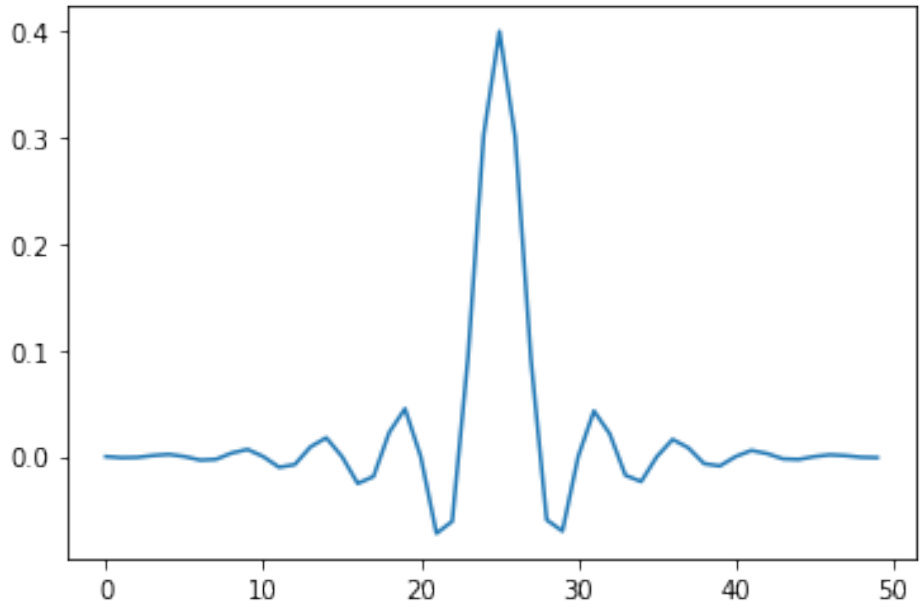

Figure 22: Kernel

Temos a seguinte resposta em frequência do filtro:

```
#Calcular a frequencia do sinal filtrado
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.freqz.
    html
```

```
w, h = signal.freqz(kernel)
plt.plot(w, 20 * np.log10(abs(h)))
plt.xscale('log')
plt.title('Filtro passa baixa')
plt.xlabel('Frequencia [radianos / segundo]')
plt.ylabel('Amplitude [dB]')
plt.margins(0, 0.1)
plt.grid(which='both', axis='both')
plt.show()

```

Listing 37: Resposta em frequência

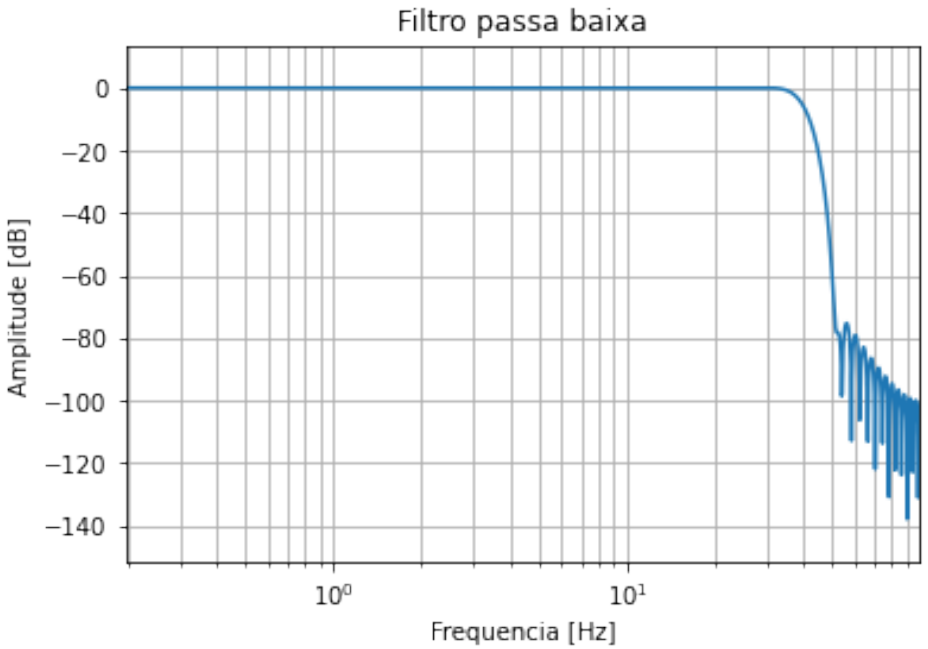

Figure 23: Resposta em frequência

(d) Podemos perceber que o filtro está de acordo com o especificado.

(e) Temos o seguinte sinal:

```
tsim3 = np.arange(0, 1, 1/fsample)
sinal = []
for t in tsim3:
    sinal.append(np.sin(2*np.pi*10*t) + 0.25*np.sin(2*np.pi*50*t))
plt.plot(tsim3, sinal)

sinalFiltrado5 = signal.lfilter(kernel, 1, sinal)
plt.stairs(sinalFiltrado5)

```

Listing 38: Sinal para ser filtrado

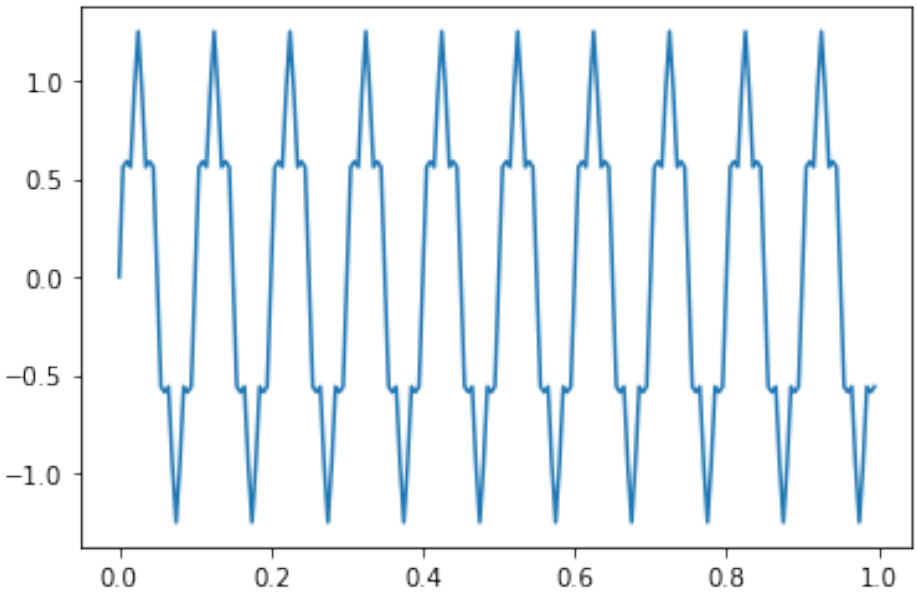

Figure 24: Sinal

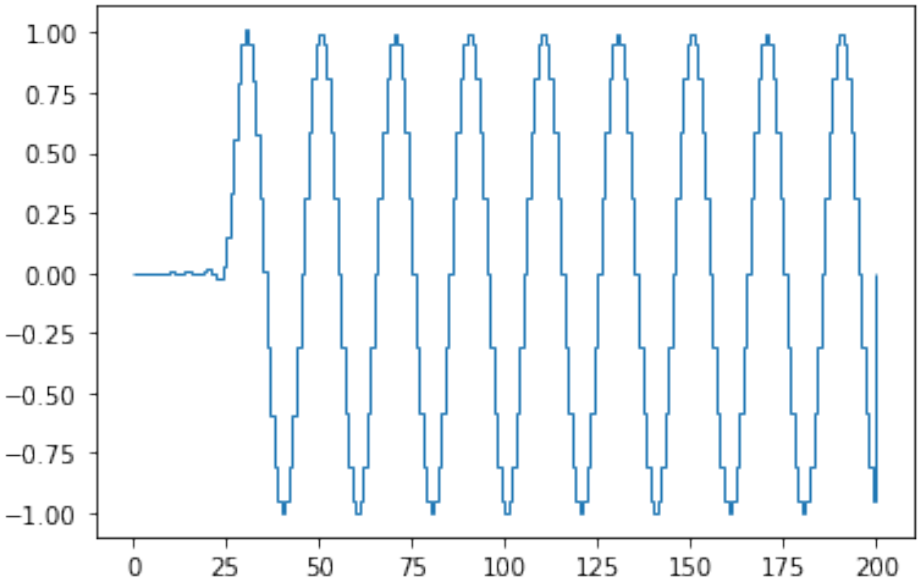

Figure 25: Sinal Filtrado

(f) Temos o seguinte espectro:

```
# Pegando o sinal filtrado depois do transitorio
N = len(sinalFiltrado5[75:-1])
T = srate
yf = scipy.fft.fft(sinalFiltrado5[75:-1])
xf = np.linspace(0.0, 1.0/(2.0*T), N//2)


plt.plot(xf[0:50], 2.0/N*np.abs(yf[0:50]))
plt.grid()
plt.show()

```

Listing 39: Espectro do sinal filtrado

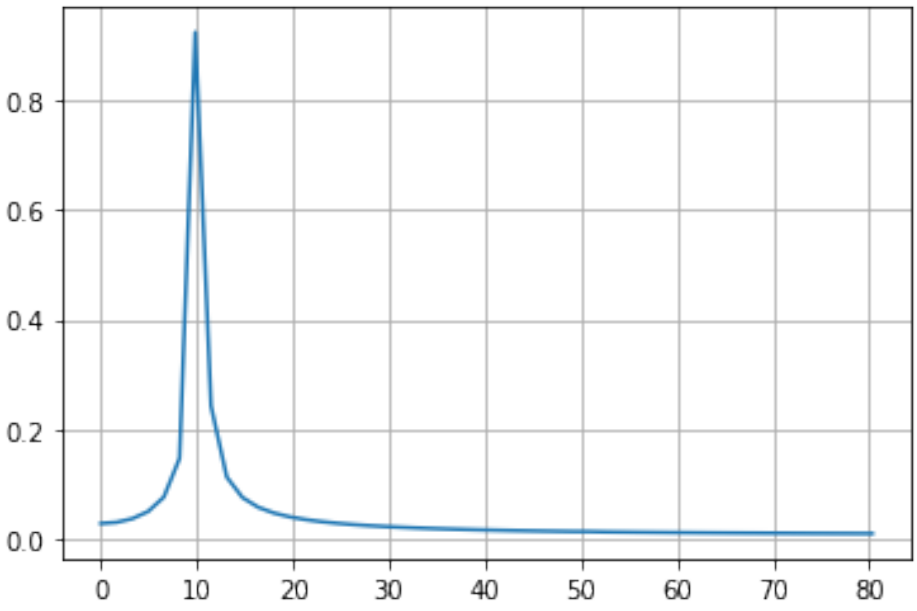

Figure 26: Espectro do Sinal Filtrado

Observando a figura 26 podemos perceber que o sinal teve alguma perda, mas no geral foi recuperado e a frequência do ruído eliminada.

2. Temos o seguinte filtro usando o comando firwin:

```
numCoef2 = 51
fsample2 = 200
srate2 = 1/fsample2

filtkern2 = signal.firwin(numCoef2, cutoff = [10, 30], window = 'hamming', fs =
    fsample2, pass_zero = False)

```

Listing 40: Calculo do filtro

(a) Calculando o filtro manualmente:

```
kernel2 = (((30/100)*np.sinc((30/100)*np.arange(-25,25,1)))-((10/100)*np.sinc
    ((10/100)*np.arange(-25,25,1))))
plt.plot(kernel2)

```

Listing 41: Calculo manual do kernel

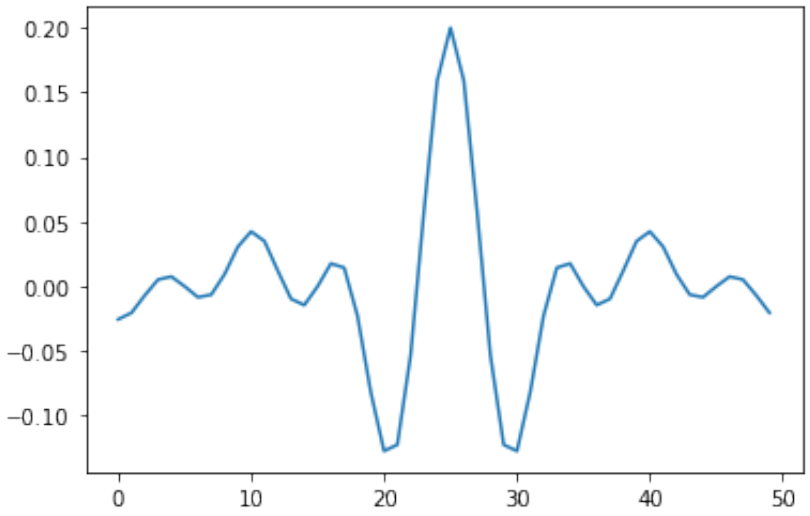

Figure 27: Kernel do filtro

Podemos perceber que o sinal foi normalizado de forma diferente em relação ao problema anterior, que foi normalizado entre 0 e 0.4, nesse caso o problema foi normalizado entre a diferença das frequências da banda de passagem (30-10)/fnyquist = 0.2.

(b) Temos a seguinte resposta em frequência do filtro:

```
#Calcular a frequencia do sinal filtrado
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.freqz.
    html
w, h = signal.freqz(filtkern)
plt.plot(w*200*0.1592, 20 * np.log10(abs(h)))
plt.xscale('log')
plt.title('Filtro passa baixa')
plt.xlabel('Frequencia [Hz]')
plt.ylabel('Amplitude [dB]')
plt.margins(0, 0.1)
plt.grid(which='both', axis='both')
plt.show()


```

Listing 42: Resposta em frequência do filtro

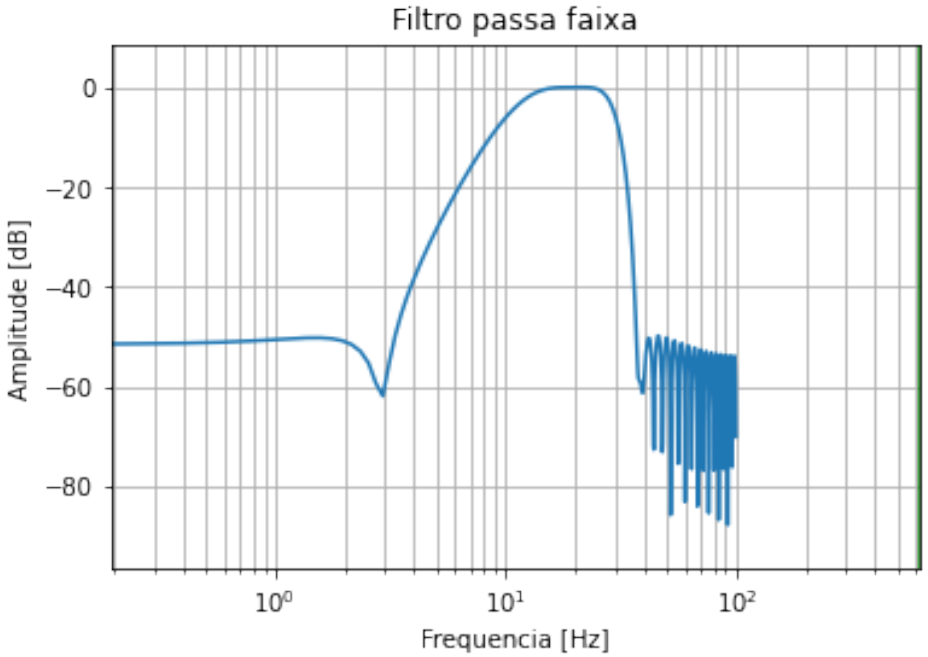

Figure 28: Resposta em frequência do filtro

(c) Temos um ganho aproximadamente unitário nas frequências entre 10Hz e 30H e em média -50dB nas outras frequências.

(d) Temos o seguinte sinal a ser simulado:

```
tsim4 = np.arange(0, 1, 1/fsample2)
sinal2 = []
for t in tsim4:
    sinal2.append(0.1*np.sin(2*np.pi*4*t)+ np.sin(2*np.pi*20*t) + 0.5*np.sin
    (2*np.pi*60*t))
plt.plot(tsim4, sinal2)

```

Listing 43: Sinal a ser filtrado

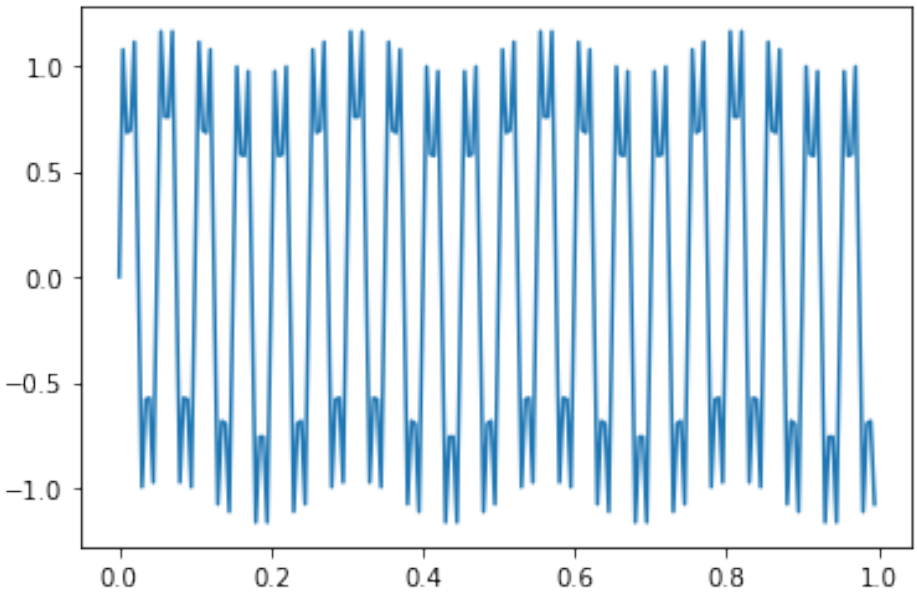

Figure 29: Sinal a ser filtrado

(e) Temos o seguinte sinal filtrado:

```
sinalFiltrado6 = signal.lfilter(kernel2, 1, sinal2)
plt.plot(sinalFiltrado6)

```

Listing 44: Sinal filtrado

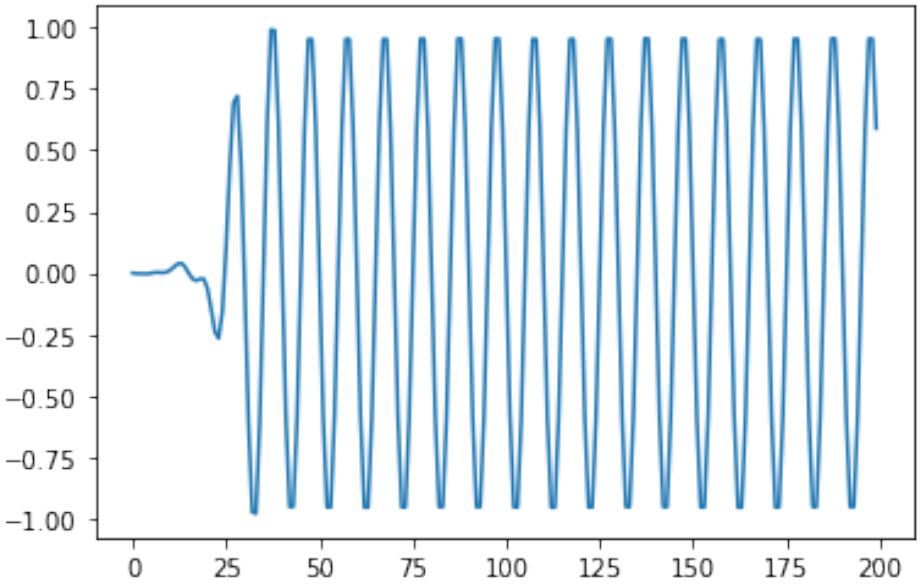

Figure 30: Sinal filtrado

(f) Temos o seguinte espectro para o sinal filtrado:

```
# Pegando o sinal filtrado depois do transitorio
N = len(sinalFiltrado6[75:-1])
T = srate2
yf = scipy.fft.fft(sinalFiltrado6[75:-1])
```

```
xf = np.linspace(0.0, 1.0/(2.0*T), N//2)


plt.plot(xf[0:50], 2.0/N*np.abs(yf[0:50]))
plt.grid()
plt.show()

```

Listing 45: Espectro do Sinal filtrado

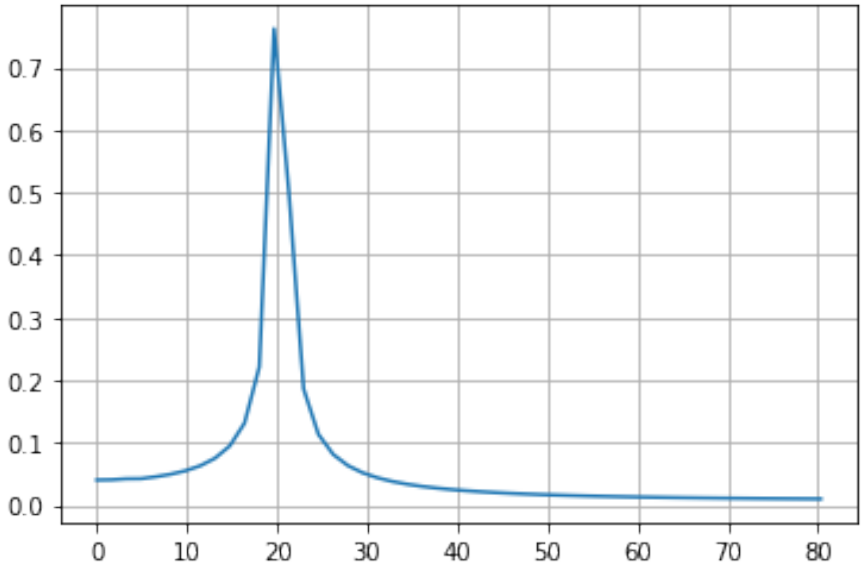

Figure 31: Espectro do Sinal filtrado

Podemos perceber que o sinal de interesse foi relativamente atenuado, mas os sinais de ruído, principalmente o de 60 Hz, que possuia uma amplitude grande, foi bem atenuado.

3. Temos o seguinte filtro:

```
N = 20
# N precisa ser impar
if N % 2 == 0:
    N = N + 1
freqs = [0,800,1000,4000]
m = [1,1,0,0]

b8 = signal.firwin2(N, freqs, m, fs=8000)
```

Listing 46: Filtro

(a) Temos a seguinte resposta do filtro:

```
#Calcular a frequencia do sinal filtrado
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.freqz.
    html
w, h = signal.freqz(b8,1)
plt.plot(w*(8000)*0.1592, 20 * np.log10(abs(h)))
plt.xscale('log')
plt.title('Filtro passa baixa')
plt.xlabel('Frequencia [Hz]')
plt.ylabel('Amplitude [dB]')
plt.margins(0, 0.1)
plt.grid(which='both', axis='both')
plt.show()

```

Listing 47: Resposta do Filtro

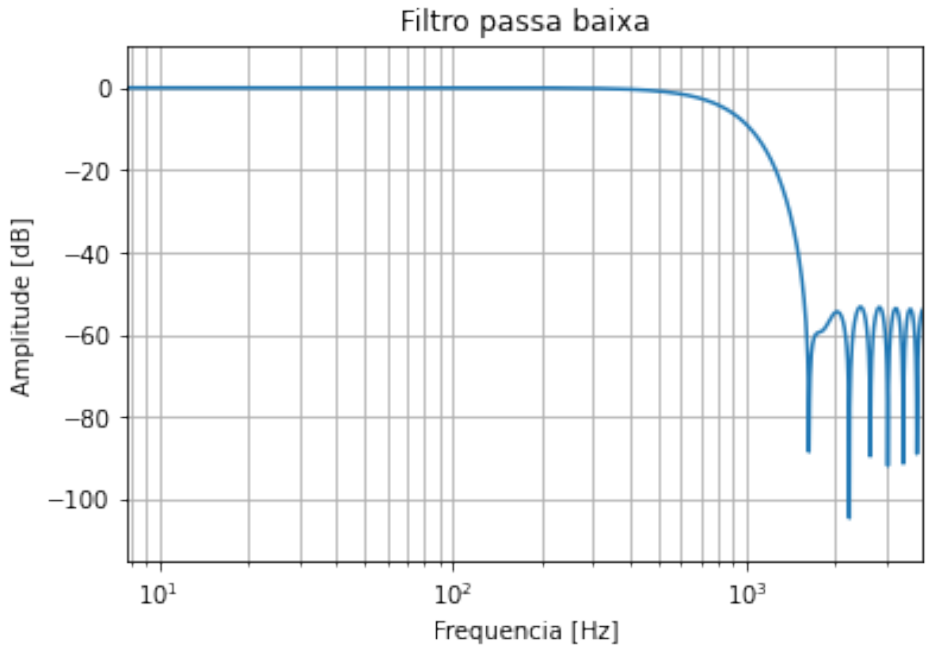

Figure 32: Resposta em frequência do filtro

(b) Temos o seguinte sinal a ser filtrado:

```
tsim5 = np.arange(0, 0.0125, 1/8000)
sinal3 = []
for t in tsim5:
    sinal3.append(1 + 0.25*np.sin(2*np.pi*3500*t))
plt.plot(tsim5, sinal3)

```

Listing 48: Sinal

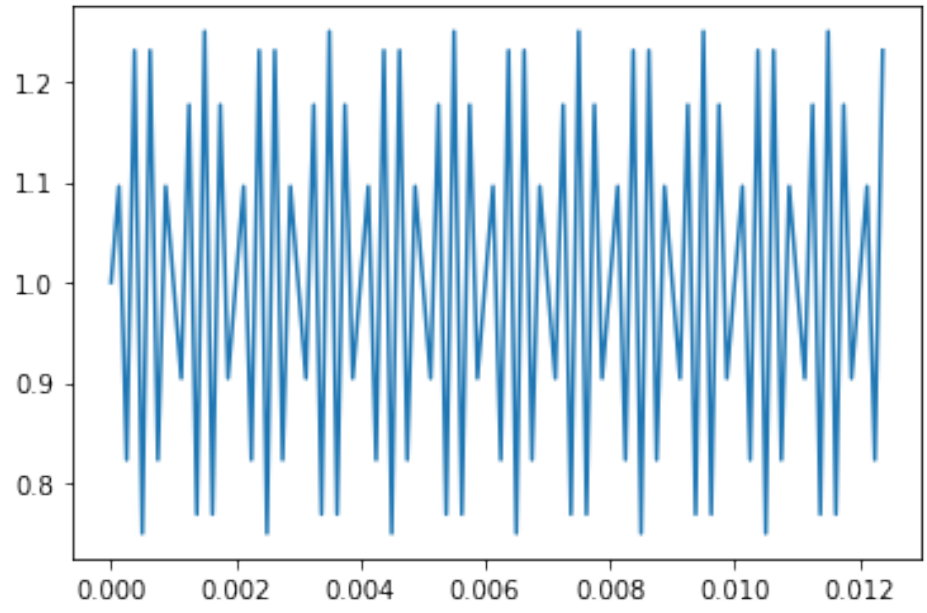

Figure 33: Sinal 3

(c) Temos o seguinte sinal filtrado:

```
sinalFiltrado7 = signal.lfilter(b8, 1, sinal3)
plt.plot(sinalFiltrado7)

```

Listing 49: Sinal filtrado

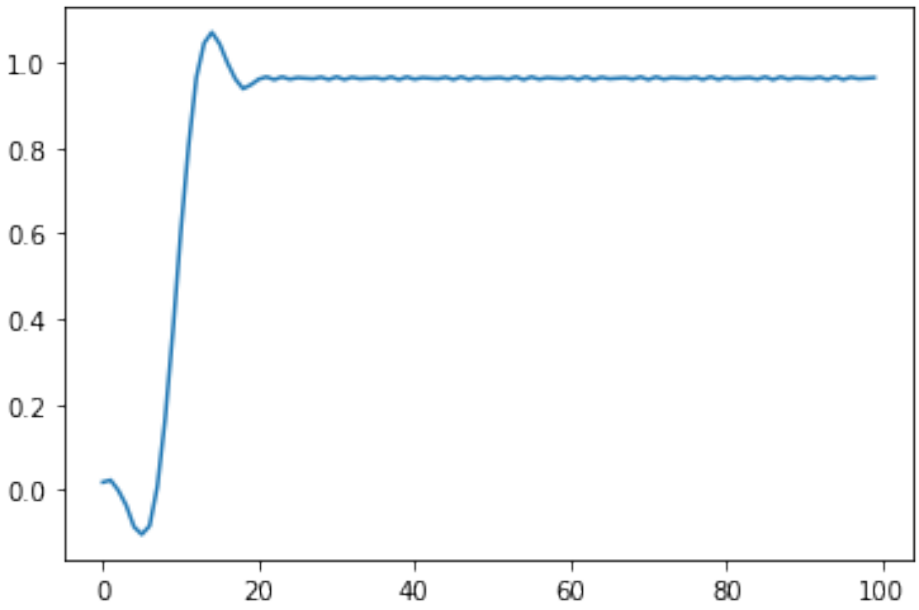

Figure 34: Sinal filtrado

(d) Temos o seguinte espectro do sinal:

```
# Pegando o sinal filtrado depois do transitorio
N = len(sinalFiltrado7[20:])
T = 1/8000
yf = scipy.fft.fft(sinalFiltrado7[20:])
xf = np.linspace(0.0, 1.0/(2.0*T), N//2)


plt.plot(xf, 1.0/N*np.abs(yf[0:N//2]))
plt.grid()
plt.show()

```

Listing 50: Sinal filtrado

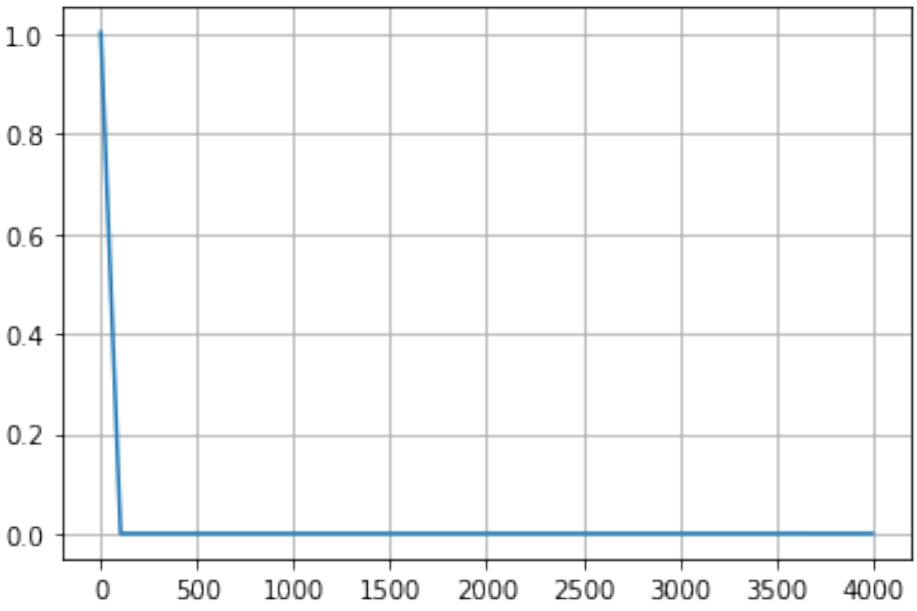

Figure 35: Espectro do Sinal filtrado

Podemos ver que o ruído na frequência de 3500 Hz foi eliminado e o sinal não apresentou perdas (após um transitório), o que mostra que o sinal foi recuperado.