Universidade Federal de Santa Catarina
Centro Tecnológico
Programa de Pós-Graduação em Engenharia Elétrica
Discentes: Carlos André Antunes, João Lucas Pereira dos Santos de Paula e Shirley Karolina da Silva Ferreira

Processamento de Sinais Biomédicos (T1)

Considerações Iniciais

Para o desenvolvimento deste trabalho, foi utilizada a linguagem de programação python e a ferramenta de desenvolvimento Colab.

Problema 1: Autocorrelação para detecção de sinais na presença de ruído.

Considerando uma senoide de 1 V_{pp} e 17 Hz, definida como $x_1(t)$; e um ruído aleatório de 5 V_{pp}, definido como $x_2(t)$, a Figura 1 ilustra cada sinal e a soma das componentes individuais, ou seja:

$$x(t) = x_1(t) + x_2(t) \tag{1}$$

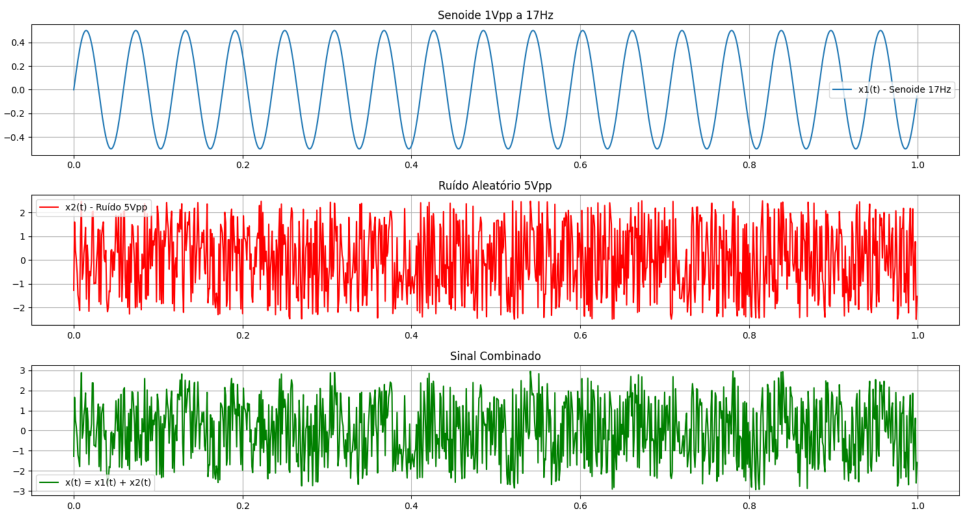

Figura 1 – Senoide (em azul), ruído aleatório (em vermelho) e sinal resultante da soma entre estes dois sinais (em verde).

Neste problema será utilizada a operação de autocorrelação no domínio discreto (Equação 2) para a detecção de sinais periódicos na presença de ruído, ou seja, a entrada da função de autocorrelação é o sinal ruidoso $x(t)$.

$$r_{xx}(j) = \frac{1}{N}\sum_{n=0}^{N-1} X(n) \times X(n+j) \tag{2}$$

A autocorrelação de $x(t)$ é mostrada na Figura 2.

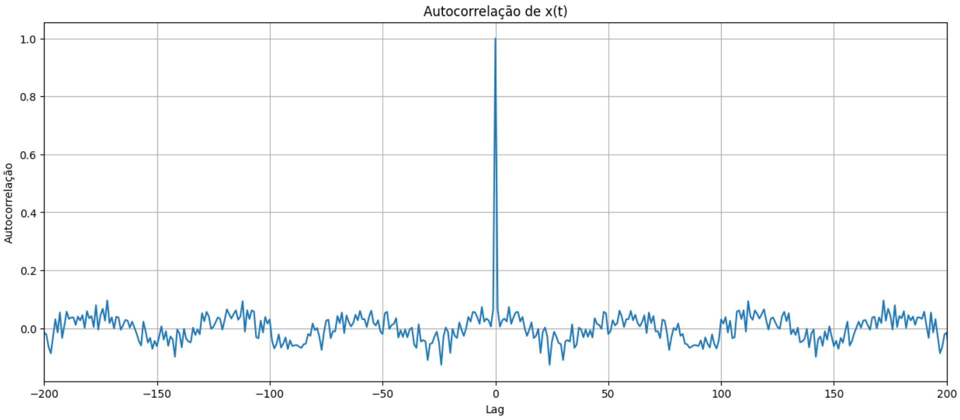

Figura 2 – Autocorrelação do sinal ruidoso $x(t)$.

Dessa forma, é possível verificar que a autocorrelação do sinal $x(t)$ consiste na soma da autocorrelação do sinal senoidal com a autocorrelação do sinal aleatório (ruído), ou seja:

$$r_{xx}(j) = r_{11}(j) + r_{22}(j) \tag{3}$$

A Figura 3 ilustra este resultado:

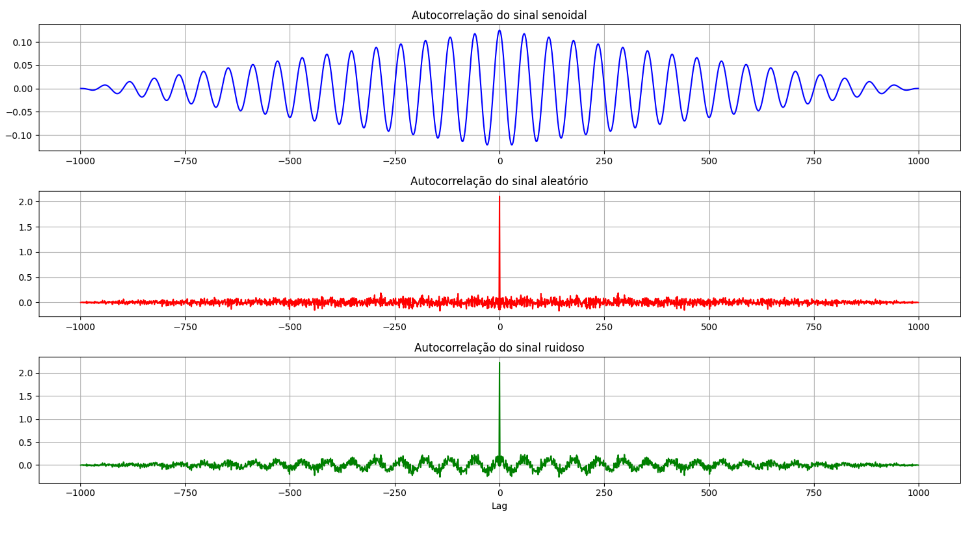

Figura 3 – A autocorrelação representando a soma de dois sinais não correlacionados, ou seja, sinal + ruído, consiste na soma das funções de autocorrelação das componentes individuais.

O espectro da autocorrelação resultante indica a maior atividade na frequência de, aproximadamente, $17Hz$, como mostrado na Figura 4 a seguir.

Ao se utilizar a autocorrelação para a detecção de sinais periódicos escondidos em ruídos, não é possível recuperar o sinal original completamente, devido à perda de informação de fase, porém, é possível determinar a amplitude e a frequência do sinal original.

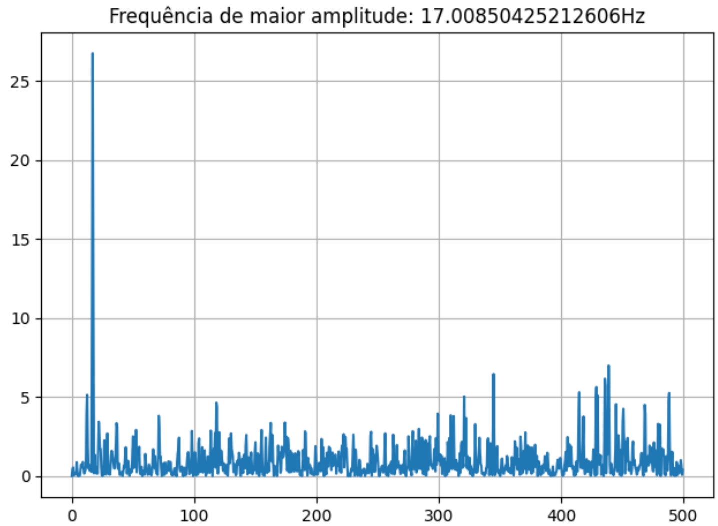

Figura 4 – Espectro do sinal de autocorrelação de $x(t)$.

Código Desenvolvido para Resolução do Problema 1:

```
# Definindo a sequência x(t)
t_values = np.arange(0, 33, 1)  # valores de t de 0 a 32 com T=1
x_values = np.exp(-0.5 * t_values)

# Resposta ao impulso h(t)
h_values = np.array([0, 1.5, 0.75, 0.375, 0.1875, 0.09375, 0.046875, 0.0234375])

# Calculando a convolução de x(t) e h(t)
y_values = np.convolve(x_values, h_values, mode='full')

# Plotando os sinais
plt.figure(figsize=(15, 8))
plt.subplot(3, 1, 1)
plt.plot(t, x1, label='x1(t) - Senoide 17Hz')
plt.title('Senoide 1Vpp a 17Hz')
plt.legend()
plt.grid(True)

plt.subplot(3, 1, 2)
plt.plot(t, x2, label='x2(t) - Ruído 5Vpp', color='red')
plt.title('Ruído Aleatório 5Vpp')
plt.legend()
plt.grid(True)

plt.subplot(3, 1, 3)
plt.plot(t, x, label='x(t) = x1(t) + x2(t)', color='green')
plt.title('Sinal Combinado')
```

```python
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

def autocorrelation(signal):
    n = len(signal)
    result = np.zeros(2*n - 1)
    # p = int(0.5*n)
    # signal_zeros = np.concatenate((signal[0:p], np.zeros(n-p)))
    for tau in range(-n + 1, n):
        if tau < 0:
            result[tau + n - 1] = np.sum(signal[:tau+n] * signal[-tau:])
        else:
            result[tau + n - 1] = np.sum(signal[tau:] * signal[:n-tau])
    return result*(1/n)

# Calculando a autocorrelação usando nossa função
correlation_custom = autocorrelation(x)
correlation_custom /= np.max(correlation_custom)  # Normalizando para [-1,1]

x_f = fft(correlation_custom)
Nx = len(correlation_custom) # número de amostras (pontos)


# Eixo do tempo para a autocorrelação
lags = np.arange(-len(x) + 1, len(x))

# Plotando a autocorrelação
plt.figure(figsize=(15, 6))
plt.plot(lags, correlation_custom)
plt.title('Autocorrelação de x(t)')
plt.xlabel('Lag')
plt.ylabel('Autocorrelação')
plt.grid(True)
plt.xlim([-200, 200])  # limitando a visualização para -200 a 200 lags para melhor visualização
plt.show()


# Propriedade: soma das autocorrelações
r11 = autocorrelation(x1)
r22 = autocorrelation(x2)
rxx = r11+r22

```

```python
plt.figure(figsize=(15, 8))
plt.subplot(3, 1, 1)
plt.plot(lags, r11, color='blue')
plt.title('Autocorrelação do sinal senoidal')
plt.grid(True)

plt.subplot(3, 1, 2)
plt.plot(lags, r22, color='red')
plt.title('Autocorrelação do sinal aleatório')
plt.grid(True)

plt.subplot(3, 1, 3)
plt.plot(lags, rxx, color='green')
plt.title('Autocorrelação do sinal ruidoso')
plt.grid(True)
plt.xlabel('Lag')

plt.tight_layout()
plt.show()

# Espectro da autocorrelação
xs = fft(correlation_custom)
Nx = len(correlation_custom) # número de amostras (pontos)
x_f = fftfreq(Nx, d=1/fs)[:Nx//2]
amp_max = np.argmax(np.abs(xs[0:Nx//2]))
# PLotando
plt.plot(x_f, np.abs(xs[0:Nx//2]))
plt.title(f'Frequência de maior amplitude: {x_f[amp_max]}Hz')
plt.grid(True)
plt.tight_layout()
plt.show()
```

Problema 2: Convolução digital de sinal x(t) com a resposta ao impulso do sistema h(t).

Considerando o sinal $x(t)$ definido como:

$$x(t) = \exp(-0.5t), \quad 0 \leq t \leq 32T \tag{4}$$

A resposta ao impulso do sistema ($h(t)$) é dada por:

$$h(t) = [0, 1.5, 0.75, 0.375, 0.1875, 0.09375, 0.046875, 0.0234375] \tag{5}$$

Dada a convolução:

$$y(t) = x(t) * h(t) \tag{6}$$

A Figura 5 detalha o comportamento do sinal $x(t)$, da resposta ao impulso $h(t)$ e da convolução digital entre os dois sinais $y(t)$.

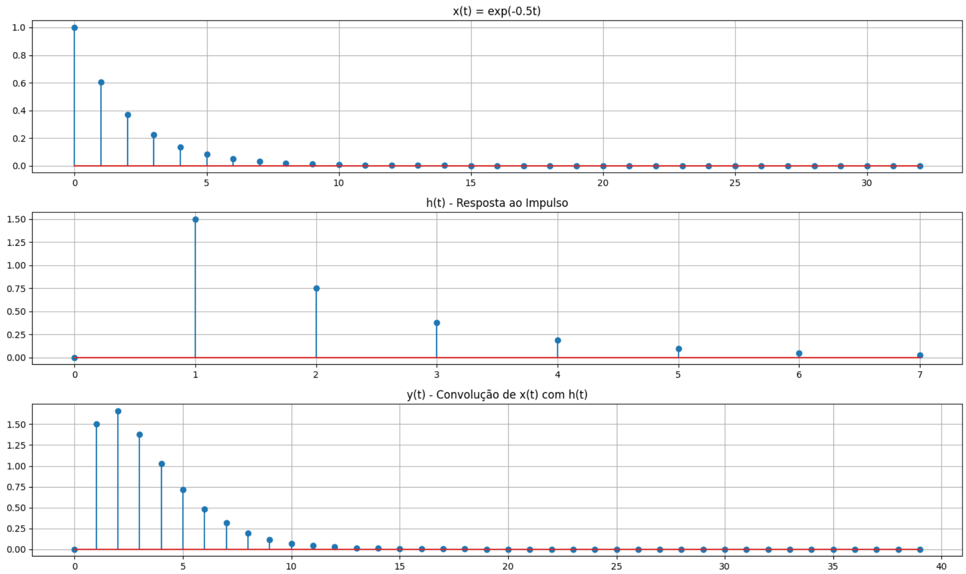

Figura 5 – Sinais e convolução digital.

Código Desenvolvido para Resolução do Problema 2:

```
# Definindo a sequência x(t)
t_values = np.arange(0, 33, 1)  # valores de t de 0 a 32 com T=1
x_values = np.exp(-0.5 * t_values)

# Resposta ao impulso h(t)
h_values = np.array([0, 1.5, 0.75, 0.375, 0.1875, 0.09375, 0.046875, 0.0234375])

# Calculando a convolução de x(t) e h(t)
y_values = np.convolve(x_values, h_values, mode='full')

# Plotando x(t), h(t) e y(t)
plt.figure(figsize=(15, 9))

# x(t)
plt.subplot(3, 1, 1)
plt.stem(t_values, x_values, use_line_collection=True)
plt.title('x(t) = exp(-0.5t)')
plt.grid(True)

# h(t)
plt.subplot(3, 1, 2)
plt.stem(h_values, use_line_collection=True)
plt.title('h(t) - Resposta ao Impulso')
plt.grid(True)
```

```python

# y(t)
plt.subplot(3, 1, 3)
plt.stem(y_values, use_line_collection=True)
plt.title('y(t) - Convolução de x(t) com h(t)')
plt.grid(True)

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

Problema 3: Filtro FIR e resposta em frequência.

Considerando o seguinte sinal $S(t)$:

$$S(t) = sen(2\pi f_1 t) + sen(2\pi f_2 t) + sen(2\pi f_3 t) \tag{7}$$

Sendo $f_1 = 750Hz$, $f_2 = 2500Hz$, $f_3 = 3250Hz$ e a frequência de amostragem igual a $8kHz$. Foi então projetado um filtro FIR passa baixa com as seguintes especificações: frequência de corte (f_c): $1,5kHz$; banda de transição (Δf): $0,5kHz$; atenuação da faixa de rejeição: $> 50dB$ e frequência de amostragem (f_s): $8kHz$

Dadas as especificações do filtro, constatou-se que a janela de Hamming satisfaz o problema, cuja matemática é definida como:

$$w(n) = 0.54 + 0.46 \times cos\left(\frac{2\pi n}{N}\right) \tag{8}$$

Sendo N o número de coeficientes do filtro. Considerando que a banda de transição em Hz, normalizada, para a janela de Hamming é calculada por:

$$\Delta f = \frac{3,3}{N} \tag{9}$$

É possível então calcular o número de coeficientes do filtro (N). Sendo Δf a fração da frequência de amostragem, ou seja:

$$\Delta f = \frac{0,5kHz}{8kHz} = 0,0625 \tag{10}$$

Logo:

$$N = \frac{3,3}{0,0625} = 52,8 \tag{11}$$

Tomamos, portanto, $N = 53$ coeficientes. O intervalo dos coeficientes do filtro, definido por:

$$|n| \leq \frac{(N-1)}{2} \tag{12}$$

Resulta na equação final, definida como:

$$\begin{gathered} h(n) = h_d(n) \times w(n) \\ -26 \leq n \leq 26 \end{gathered} \tag{13}$$

Em que o filtro propriamente dito é dado por:

$$h_d(n) = \begin{cases} 2f_c \times \frac{sen(nw_c)}{nw_c}, n \neq 0 \\ 2f_c, n = 0 \end{cases} \tag{14}$$

Que será aplicado em conjunto com a janela de Hamming 8. Porém, devido ao uso desta janela na resposta do filtro, a frequência de corte do filtro resultante será diferente da especificada $(1,5kHz)$. Para este caso, f_c é recalculada, sendo centrada na banda de transição. Assim:

$$f_c = f_c + \frac{\Delta f}{2} = 1,5 + \frac{0,5}{2} = 1,75kHz \tag{15}$$

Sendo a f_c normalizada dada por:

$$f_c = \frac{1,75kHz}{8kHz} = 0,21875 \tag{16}$$

Considerando que o filtro $h(n)$ é simétrico e após a implementação do algoritmo para cálculos dos seus coeficientes, o filtro resultante é ilustrado na Figura 6.

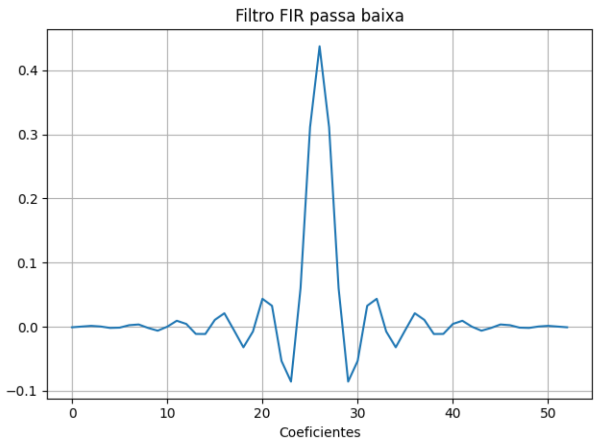

Figura 6 – Filtro FIR passa baixa com $N = 53$ coeficientes, utilizando a janela de Hamming.

A Figura 7 ilustra o sinal $S(t)$ para $6s$, o espectro de frequência deste sinal, destacando a atividade em f_1, f_2 e f_3, o sinal filtrado pelo filtro FIR passa baixa e o espectro de frequência do sinal filtrado, com atividade na componente frequencial de $f_1 = 750Hz$, uma vez que a frequência de corte do filtro é $1,5kHz$, ou seja, f_2 e f_3 são eliminadas pelo filtro.

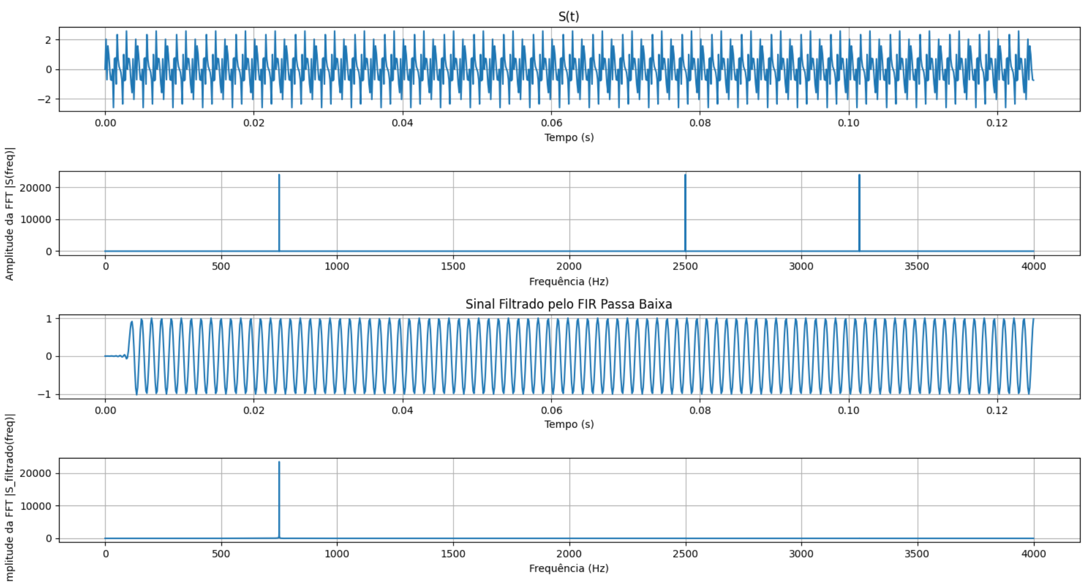

Figura 7 – Sinais original e filtrado com seus respectivos espectros de frequência.

Código Desenvolvido para Resolução do Problema 3:

```python
# Gerando o sinal S
f1 = 750
f2 = 2500
f3 = 3250
fs = 8000
T = 1/fs  # período de amostragem
t = np.arange(0, 6, T) # vetor de tempo
S = np.sin(2 * np.pi * f1 * t) + np.sin(2 * np.pi * f2 * t) + np.sin(2 * np.pi * f3 * t)

# Espectro de frequência de S
S_f = fft(S)
N = len(S) # número de amostras (pontos)
xf = fftfreq(N, d=T)[:N//2]

# calculo dos coeficientes do filtro FIR usando o metódo da janela com a função Hamming
f_cutoff = 1500
delta_f = 500
delta_f_normalised = delta_f/fs
print(f'Banda de Transição normalizada:{delta_f_normalised}')
N_coeff = int(np.ceil(3.3/delta_f_normalised))
print(f'Número de coeficientes do filtro FIR:{N_coeff}')
n = np.arange(-(N_coeff-1)/2,(N_coeff-1)/2+1)
print(f'Índices:{n}')

# Devido ao uso da janela, a frequência de corte resultante será diferente da especificada (f_cutoff). Normalizando:
f_cutoff = f_cutoff + delta_f/2
f_cutoff_normalised = f_cutoff/fs
print(f'Nova frequência de corte normalizada:{f_cutoff_normalised}')

hd = np.zeros(int((N_coeff-1)/2)+1)
w = np.zeros(int((N_coeff-1)/2)+1)
h = np.zeros(N_coeff)

for n in range(int((N_coeff-1)/2)+1): # 0 a 26

   if n == 0:
     hd[n] = 2 * f_cutoff_normalised
   else:
     hd[n] = 2 * f_cutoff_normalised * np.sin(n*2*np.pi*f_cutoff_normalised)/(n*2*np.pi*f_cutoff_normalised)

```

```
    w[n] = 0.54 + 0.46 * np.cos((2*np.pi*n)/N_coeff)

    h[26-n] = h[n+26] = hd[n] * w[n] # simetria

print(f'Coeficientes do filtro FIR:{h}')

# Filtrando o sinal
S_filtered = np.convolve(h,S)

# espectro do sinal filtrado
Sfiltered_f = fft(S_filtered)
N2 = len(Sfiltered_f) # número de amostras (pontos)
xf2 = fftfreq(N2, d=T)[:N2//2]

# Plot da janela
plt.plot(h)
plt.title('Filtro FIR passa baixa')
plt.xlabel('Coeficientes')
plt.grid(True)
plt.tight_layout()
plt.show()

# Plot do sinal e do seu espectro de frequência
plt.figure(figsize=(15, 8))
plt.subplot(4, 1, 1)
plt.plot(t[0:1000], S[0:1000], label='S(t)')
plt.title('S(t)')
plt.grid(True)
plt.xlabel('Tempo (s)')

plt.subplot(4,1,2)
# plt.plot(xf, 2.0/N * np.abs(S_f[0:N//2]), label='S(f)')
plt.plot(xf, np.abs(S_f[0:N//2]), label='S(f)')
plt.grid(True)
plt.xlabel('Frequência (Hz)')
plt.ylabel('Amplitude da FFT |S(freq)|')


plt.subplot(4, 1, 3)
plt.plot(t[0:1000], S_filtered[0:1000])
plt.title('Sinal Filtrado pelo FIR Passa Baixa')
plt.grid(True)
plt.xlabel('Tempo (s)')

plt.subplot(4,1,4)
# plt.plot(xf, 2.0/N * np.abs(S_f[0:N//2]), label='S(f)')
```

```
plt.plot(xf2, np.abs(Sfiltered_f[0:N2//2]))
plt.grid(True)
plt.xlabel('Frequência (Hz)')
plt.ylabel('Amplitude da FFT |S_filtrado(freq)|')


plt.tight_layout()
plt.show()

```

Problema 4: Método da colocação de pólos e zeros no plano Z usando sinal de ECG.

Considerando as seguintes especificações do filtro Notch:

1. Notch frequency 60 Hz
2. 3dB width of notch ± 5 Hz
3. Sampling frequency 1200 Hz ou alguma frequência múltipla de 60 Hz.

E tendo como sequência de dados, um sinal de ECG de 15 segundos amostrado a uma taxa de $500Hz$. A Figura 8 detalha um trecho do sinal ECG bruto.

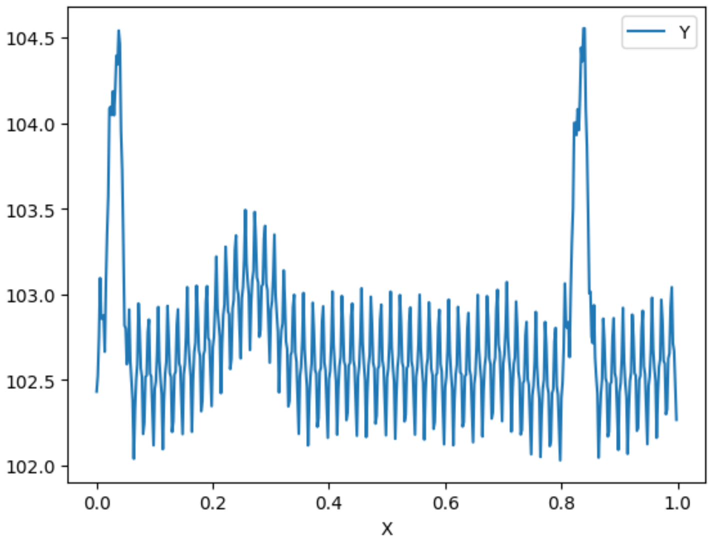

Figura 8 – Trecho do sinal original. Constata-se a forte presença da interferência de $60Hz$.

Foi realizada a reamostragem do sinal para uma frequência de amostragem de 1200Hz. Além disso, aplicou-se a interpolação para estimar os novos valores de amplitude nos pontos de tempo reamostrados. Em seguida, um filtro FIR passa-faixa de $0,04 - 40Hz$ foi aplicado através de um estágio passa alta e um estágio passa baixa, com ordens 3 e 4, respectivamente.

A Figura 9 ilustra o sinal após a aplicação do filtro passa-faixa.

Após a aplicação do filtro passa-faixa, foi então aplicado o filtro Notch de 60Hz, calculado através da alocação de pólos e zeros. A Figura 10 ilustra o sinal ECG após a aplicação do Notch.

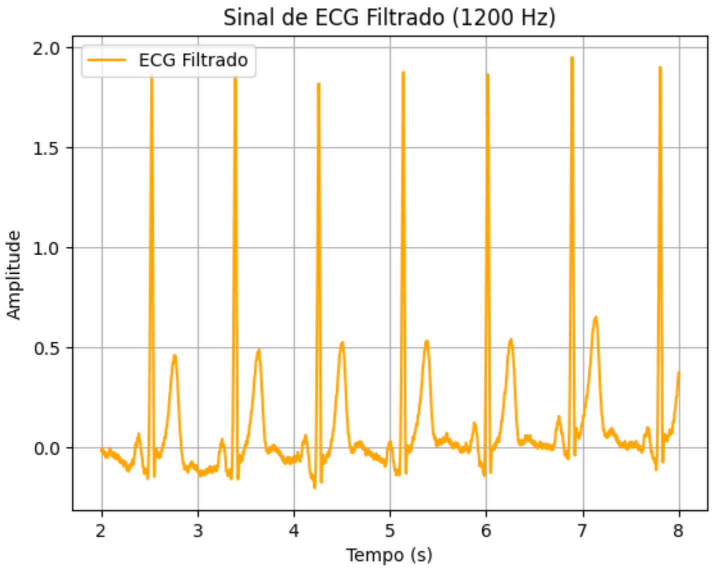

Figura 9 – Sinal ECG reamostrado, após aplicação do filtro passa-faixa.

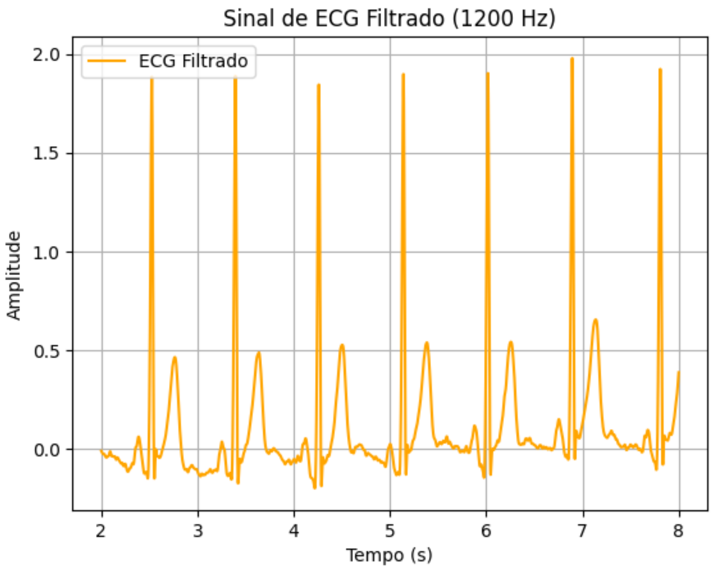

Figura 10 – Sinal após a aplicação do filtro Notch.

A Figura 11 mostra o Diagrama de Bode do filtro Notch em $60Hz$. Veja a faixa de rejeição em $60Hz$ na curva de magnitude.

A Figura 12 ilustra um trecho do sinal bruto camparado com o mesmo trecho do sinal filtrado resultante.

Já a Figura 13 detalha o espectro de frequência do sinal antes e após a filtragem, destacando a atividade em baixas frequências.

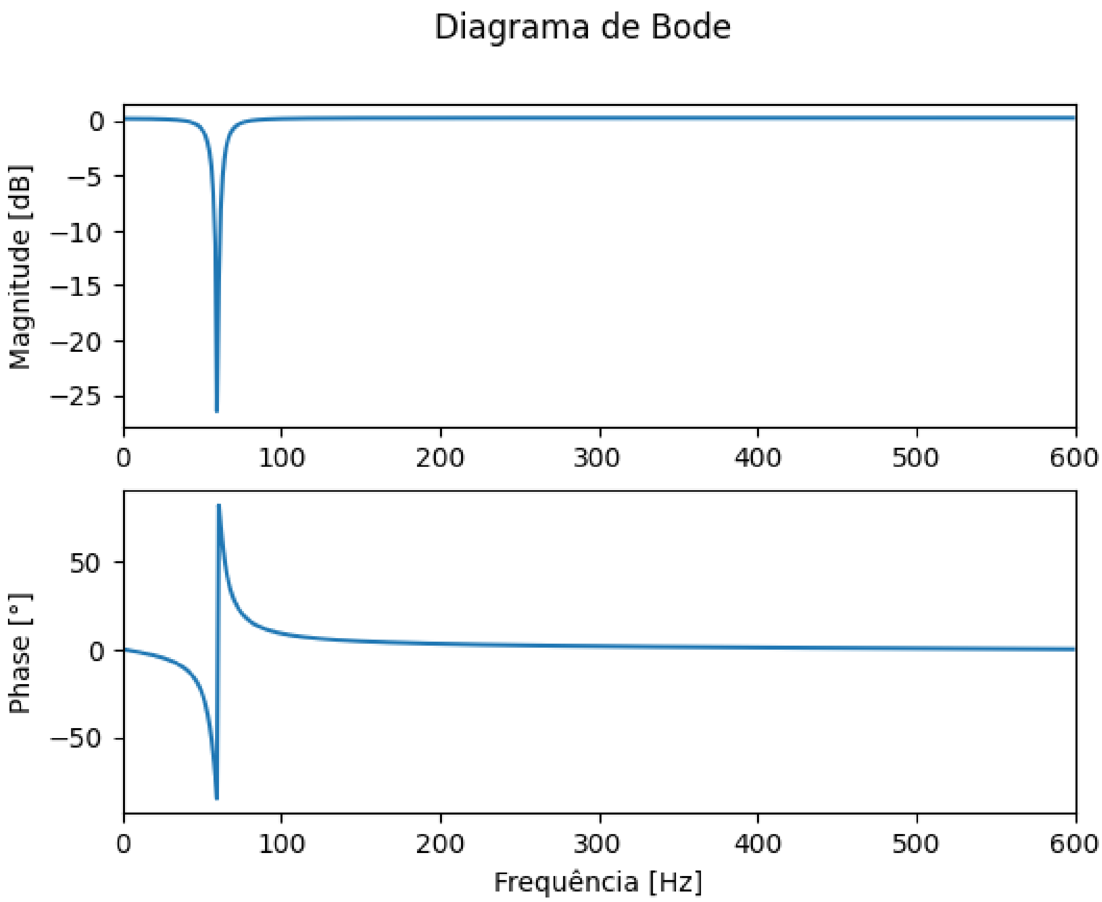

Figura 11 – Diagrama de Bode do filtro Notch.

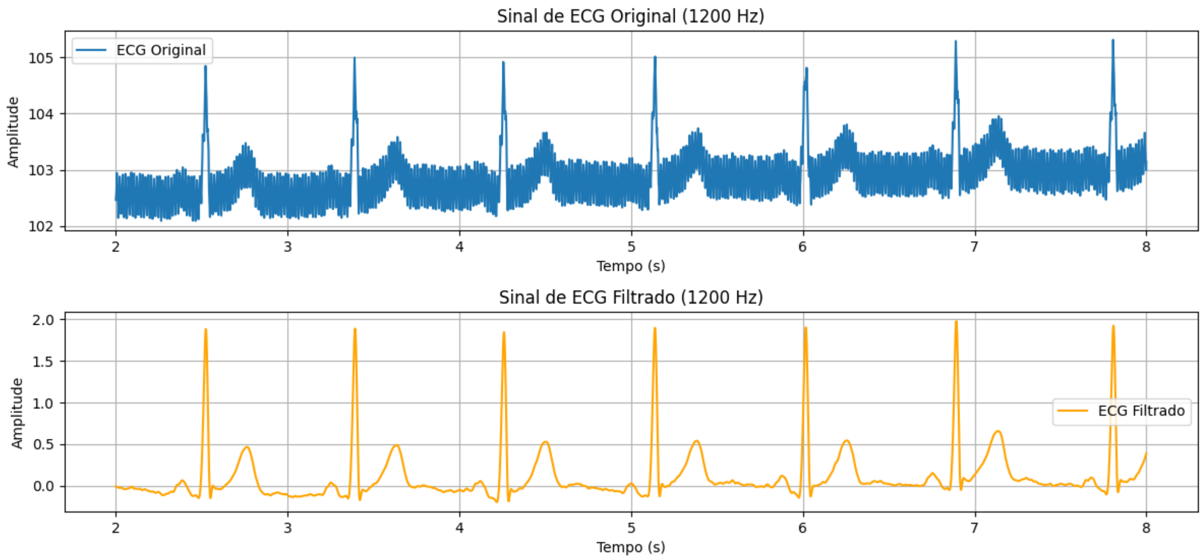

Figura 12 – Caption

Código Desenvolvido para Resolução do Problema 1:

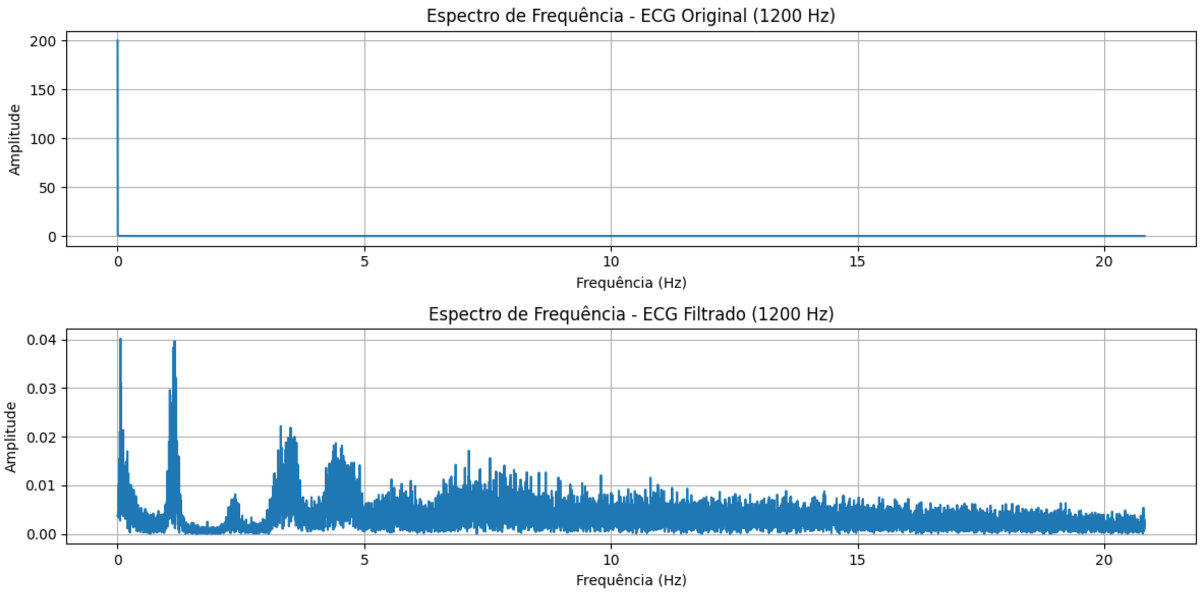

Figura 13 – Espectro de Frequência do sinal bruto e filtrado.

```
ECGtest03 = "/content/drive/MyDrive/PDSB/ECG and PPG
↪  Signals-20231021/ECGtest03.txt"

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
from scipy.signal import lfilter
from scipy.fft import fft

ecg_df = pd.read_csv(ECGtest03,sep="\t", names=["X","Y"])
ecg_df

ecg_df.iloc[:500].plot(x='X', y='Y')

largura_banda_3dB = 10
frequencia_notch = 60
# Calcular a frequência de amostragem
tempo = ecg_df["X"].to_numpy()
frequencia_amostragem = 1 / (tempo[1] - tempo[0])
frequencia_amostragem

amplitude = ecg_df["Y"].to_numpy()

# Criar uma função de interpolação
interp_funcao = interp1d(tempo, amplitude, kind='linear')

```

```python
nova_frequencia_amostragem_1200Hz = 1200 # 1200 Hz

# Gerar novos pontos de tempo para a nova frequência de amostragem
tempo_reamostrado_1200Hz = np.arange(tempo[0], tempo[-1],
↪  1/nova_frequencia_amostragem_1200Hz)

# Interpolar para obter os novos valores de amplitude
amplitude_reamostrada_1200Hz = interp_funcao(tempo_reamostrado_1200Hz)

# Mostrar algumas informações básicas sobre os dados reamostrados
pd.DataFrame({
    "Tempo (s)": tempo_reamostrado_1200Hz[:5],
    "Amplitude": amplitude_reamostrada_1200Hz[:5]
})
plt.plot(tempo_reamostrado_1200Hz, amplitude_reamostrada_1200Hz, label="ECG
↪  Original")

# Calcular o raio dos polos (r) para a nova frequência de amostragem
r_1200Hz = 1 - np.pi * largura_banda_3dB / nova_frequencia_amostragem_1200Hz

# Calcular os coeficientes do filtro Notch para a nova frequência de amostragem
b0_1200Hz = 1
b1_1200Hz = -2 * np.cos(2 * np.pi * frequencia_notch /
↪  nova_frequencia_amostragem_1200Hz)
b2_1200Hz = 1
a0_1200Hz = 1
a1_1200Hz = -2 * r_1200Hz * np.cos(2 * np.pi * frequencia_notch /
↪  nova_frequencia_amostragem_1200Hz)
a2_1200Hz = r_1200Hz**2

# Coeficientes do filtro para a nova frequência de amostragem
b_1200Hz = [b0_1200Hz, b1_1200Hz, b2_1200Hz]
a_1200Hz = [a0_1200Hz, a1_1200Hz, a2_1200Hz]


TF_notch = control.tf(b_1200Hz,a_1200Hz)

# b_1200Hz, a_1200Hz
TF_notch

w, h = freqz(b_1200Hz, a_1200Hz) # resposta em frequencia
w *= nova_frequencia_amostragem_1200Hz / (2 * np.pi) # convertendo rad/sample
↪  para Hz
# Plotando o diagrama de bode:
plt.subplot(2, 1, 1)
plt.suptitle('Diagrama de Bode')
```

```
plt.plot(w, 20 * np.log10(abs(h))) # Convert to dB
plt.ylabel('Magnitude [dB]')
plt.xlim(0, nova_frequencia_amostragem_1200Hz / 2)

plt.subplot(2, 1, 2) # Plot the phase response
plt.plot(w, 180 * np.angle(h) / np.pi) # Convert argument to degrees
plt.xlabel('Frequência [Hz]')
plt.ylabel('Phase [°]')
plt.xlim(0, nova_frequencia_amostragem_1200Hz / 2)

# Filtro passa faixa
# Ordem do filtro
order_low = 4
order_high = 3

# faixa de frequencia do filtro
f_low = 40
f_high = 0.04

# calculando os filtros que compoem do passa-faixa
b_low, a_low = butter(order_low, f_low, fs=nova_frequencia_amostragem_1200Hz,
↪ btype='lowpass')
b_high, a_high =  butter(order_high, f_high,
↪ fs=nova_frequencia_amostragem_1200Hz, btype='highpass')

# Filtrando o sinal pelo passa-alta
ecg_filtrado_1200Hz = filtfilt(b_high, a_high, amplitude_reamostrada_1200Hz)

# Filtrando o sinal pelo passa-baixa
ecg_filtrado_1200Hz = filtfilt(b_low, a_low,ecg_filtrado_1200Hz)

ecg_filtrado_1200Hz

# pós filtro passa-faixa
plt.plot(tempo_reamostrado_1200Hz[2400:9600], ecg_filtrado_1200Hz[2400:9600],
↪ label="ECG Filtrado", color='orange')
plt.xlabel("Tempo (s)")
plt.ylabel("Amplitude")
plt.title("Sinal de ECG Filtrado (1200 Hz)")
plt.grid(True)
plt.legend()

# Aplicar o filtro Notch ao sinal de ECG reamostrado para 1200 Hz
ecg_filtrado_1200Hz = lfilter(b_1200Hz, a_1200Hz, ecg_filtrado_1200Hz)

# Mostrar algumas amostras do sinal filtrado
```

```python
pd.DataFrame({
    "Tempo (s)": tempo_reamostrado_1200Hz[:5],
    "ECG Original": amplitude_reamostrada_1200Hz[:5],
    "ECG Filtrado": ecg_filtrado_1200Hz[:5]
})

# plot pós notch
plt.plot(tempo_reamostrado_1200Hz[2400:9600], ecg_filtrado_1200Hz[2400:9600],
↪  label="ECG Filtrado", color='orange')
plt.xlabel("Tempo (s)")
plt.ylabel("Amplitude")
plt.title("Sinal de ECG Filtrado (1200 Hz)")
plt.grid(True)
plt.legend()

# Configurar o tamanho da figura
plt.figure(figsize=(12, 8))

# Plotar o sinal de ECG original para 1200 Hz
plt.subplot(3, 1, 1)
plt.plot(tempo_reamostrado_1200Hz[2400:9600],
↪  amplitude_reamostrada_1200Hz[2400:9600], label="ECG Original")
plt.xlabel("Tempo (s)")
plt.ylabel("Amplitude")
plt.title("Sinal de ECG Original (1200 Hz)")
plt.grid(True)
plt.legend()

# Plotar o sinal de ECG filtrado para 1200 Hz
plt.subplot(3, 1, 2)
plt.plot(tempo_reamostrado_1200Hz[2400:9600], ecg_filtrado_1200Hz[2400:9600],
↪  label="ECG Filtrado", color='orange')
plt.xlabel("Tempo (s)")
plt.ylabel("Amplitude")
plt.title("Sinal de ECG Filtrado (1200 Hz)")
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()

# Configurar o tamanho da figura
plt.figure(figsize=(12, 8))

# Plotar o sinal de ECG original para 1200 Hz
plt.subplot(3, 1, 1)
```

```
plt.plot(tempo_reamostrado_1200Hz, amplitude_reamostrada_1200Hz, label="ECG Original")
plt.xlabel("Tempo (s)")
plt.ylabel("Amplitude")
plt.title("Sinal de ECG Original (1200 Hz)")
plt.grid(True)
plt.legend()

# Plotar o sinal de ECG filtrado para 1200 Hz
plt.subplot(3, 1, 2)
plt.plot(tempo_reamostrado_1200Hz, ecg_filtrado_1200Hz, label="ECG Filtrado", color='orange')
plt.xlabel("Tempo (s)")
plt.ylabel("Amplitude")
plt.title("Sinal de ECG Filtrado (1200 Hz)")
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()

# Função para calcular e plotar o espectro de frequência
def plotar_espectro_de_frequencia(sinal, frequencia_amostragem, titulo):
    n = len(sinal)
    T = 1 / frequencia_amostragem
    yf = fft(sinal)
    xf = fftfreq(n, T)[:n//2]
    plt.plot(xf[0:10000], 2.0/n *np.abs(yf[0:n//2])[0:10000])
    plt.grid()
    plt.title(titulo)
    plt.xlabel('Frequência (Hz)')
    plt.ylabel('Amplitude')

# Configurar o tamanho da figura
plt.figure(figsize=(12, 6))

# Espectro de frequência do sinal de ECG original para 1200 Hz
plt.subplot(2, 1, 1)
plotar_espectro_de_frequencia(amplitude_reamostrada_1200Hz, nova_frequencia_amostragem_1200Hz, "Espectro de Frequência - ECG Original (1200 Hz)")

# Espectro de frequência do sinal de ECG filtrado para 1200 Hz
plt.subplot(2, 1, 2)
```

```
plotar_espectro_de_frequencia(ecg_filtrado_1200Hz, nova_frequencia_amostragem_1200Hz, "Espectro de Frequência - ECG Filtrado (1200 Hz)")

plt.tight_layout()
plt.show()

```