camila-perazzo-roteiro-aula12

August 17, 2023

1 Tópico I - Conceitos básicos de processamentos de Sinais: Transformada de Fourier

1.1 A.1) Produza um sinal senoidal com frequência funcamental (f0) igual a 10Hz. Amostre este sinal com uma frequência de amostragem (fs) de 200Hz. Reconstrua este sinal, empregando o filtro adequado, e utilize uma frequência de corte (cutoff_freq) de 5Hz para este filtro.

[1]:
```
# Importações
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, ifft
```

[2]:
```
# Sinal original
f0 = 10  # Frequência do sinal em Hz
t = np.linspace(0, 1, 10000)  # Tempo
signal = np.sin(2 * np.pi * f0 * t)  # Sinal senoidal

# Amostragem
fs = 200  # Frequência de amostragem em Hz
Ts = 1 / fs  # Período de amostragem
t_sampled = np.arange(0, t[-1], Ts)  # Tempo amostrado
signal_sampled = np.sin(2 * np.pi * f0 * t_sampled)  # Sinal amostrado

# Transformada de Fourier
# A transformada de Fourier é usada para converter um sinal do domínio do tempo␣
↪para o domínio da frequência.
spectrum = fft(signal_sampled)

# Filtro passa-baixa
cutoff_freq = 5  # Frequência de corte do filtro em Hz

freqs = np.fft.fftfreq(len(spectrum), Ts)
# np.fft.fftfreq() é usada para calcular as frequências correspondentes
# aos pontos na Transformada de Fourier do sinal.
spectrum_filtered = np.copy(spectrum)
spectrum_filtered[np.abs(freqs) > cutoff_freq] = 0
```

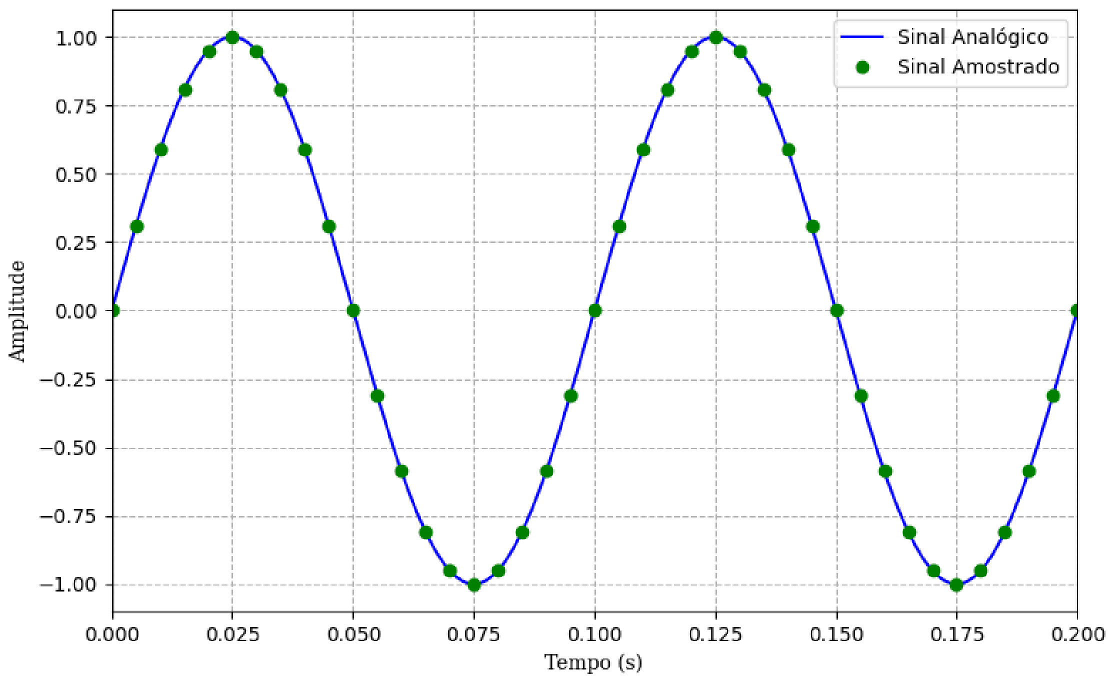
```
# Reconstrução do sinal
signal_reconstructed = ifft(spectrum_filtered)
```

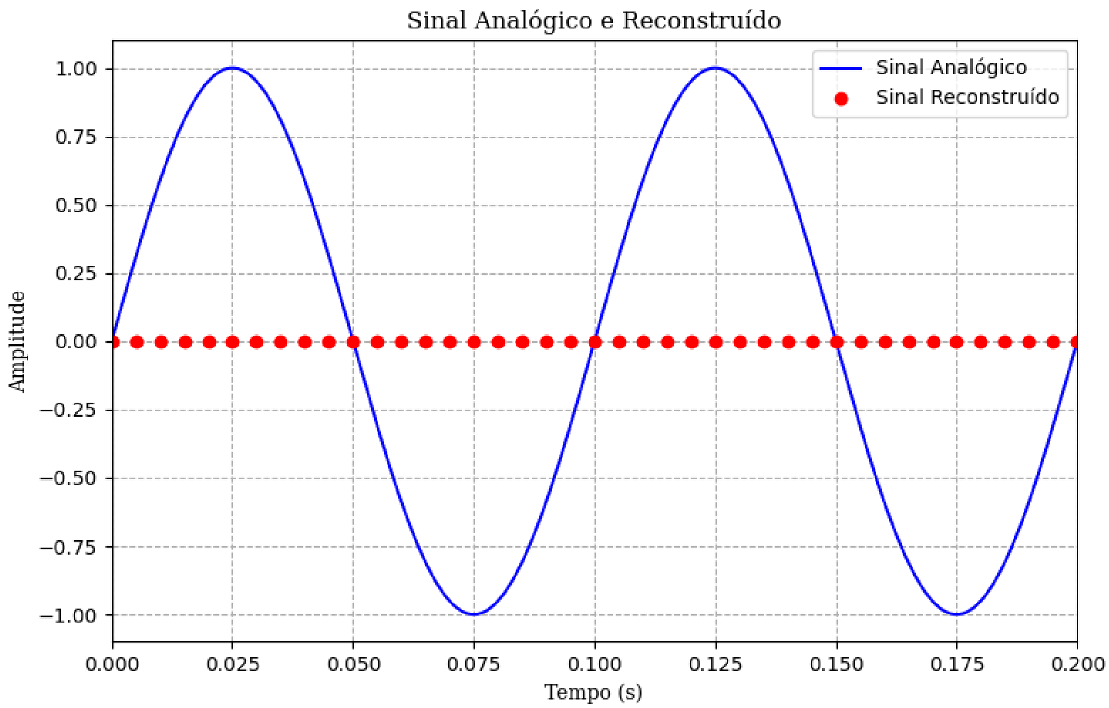
1.2 A.2) Gere dois gráficos:

- No primeiro gráfico, plote a representação do sinal analógico (em azul) e o sinal amostrado (em verde);
- No segundo gráfico, plote a representação do sinal analógico (em azul) e o sinal recosntruído (em vermelho).

[3]:
```
# Plotagem dos resultados
fig, axs = plt.subplots(2, 1, figsize=(8, 10))

# Primeiro gráfico: Sinal Analógico e Amostrado
axs[0].plot(t, signal, 'blue', label='Sinal Analógico')
axs[0].plot(t_sampled, signal_sampled, 'go', label='Sinal Amostrado')
axs[0].set_xlabel('Tempo (s)', fontname='serif')
axs[0].set_ylabel('Amplitude', fontname='serif')
axs[0].grid(True, linestyle='dashed')
axs[0].legend(fontsize=10)
axs[0].set_xlim(0, 0.2)

# Segundo gráfico: Sinal Analógico e Reconstruído
axs[1].plot(t, signal, 'blue', label='Sinal Analógico')
axs[1].plot(t_sampled, signal_reconstructed, 'ro', markersize=6, label='Sinal␣
↪Reconstruído')
axs[1].set_title('Sinal Analógico e Reconstruído', fontname='serif')
axs[1].set_xlabel('Tempo (s)', fontname='serif')
axs[1].set_ylabel('Amplitude', fontname='serif')
axs[1].grid(True, linestyle='dashed')
axs[1].legend(fontsize=10)
axs[1].set_xlim(0, 0.2)

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

```
/usr/local/lib/python3.10/dist-packages/matplotlib/cbook/__init__.py:1335:
ComplexWarning: Casting complex values to real discards the imaginary part
  return np.asarray(x, float)
```





RSPOSTA ESPERADA:

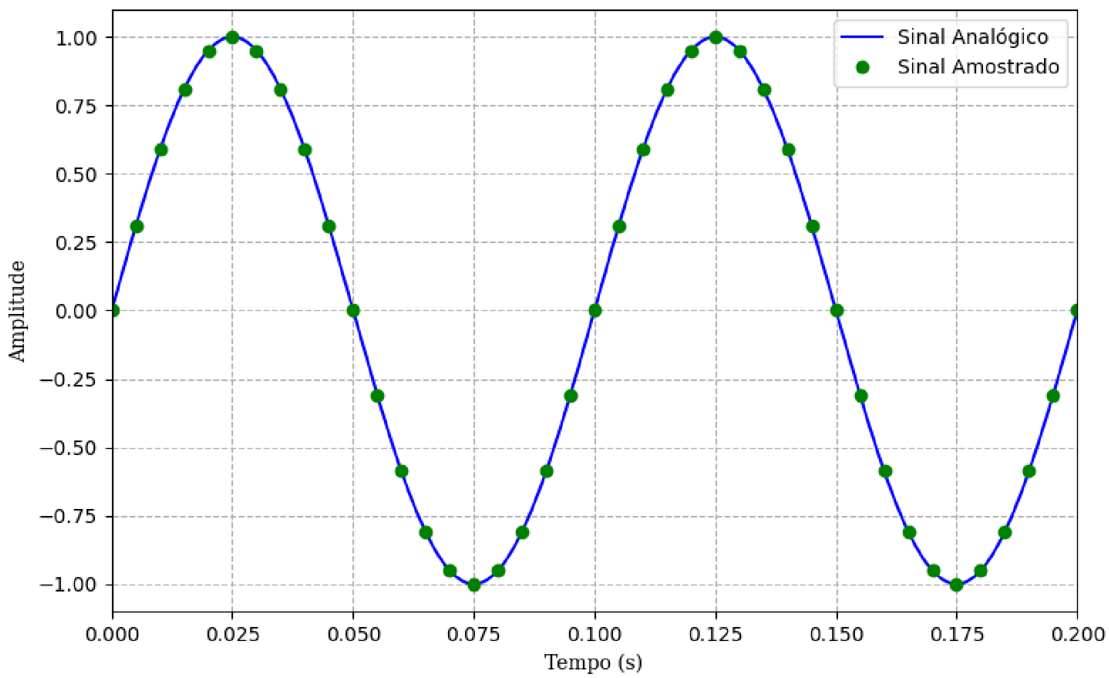
1.3 A.3) Repita o item A.1. Porém, empregue uma freqência de corte de 15Hz para o filtro em seu projeto e gere dois gráficos:

- No primeiro gráfico, plote a representação do sinal analógico (em azul) e o sinal amostrado (em verde);
- No segundo gráfico, plote a representação do sinal analógico (em azul) e o sinal recosntruído (em vermelho).

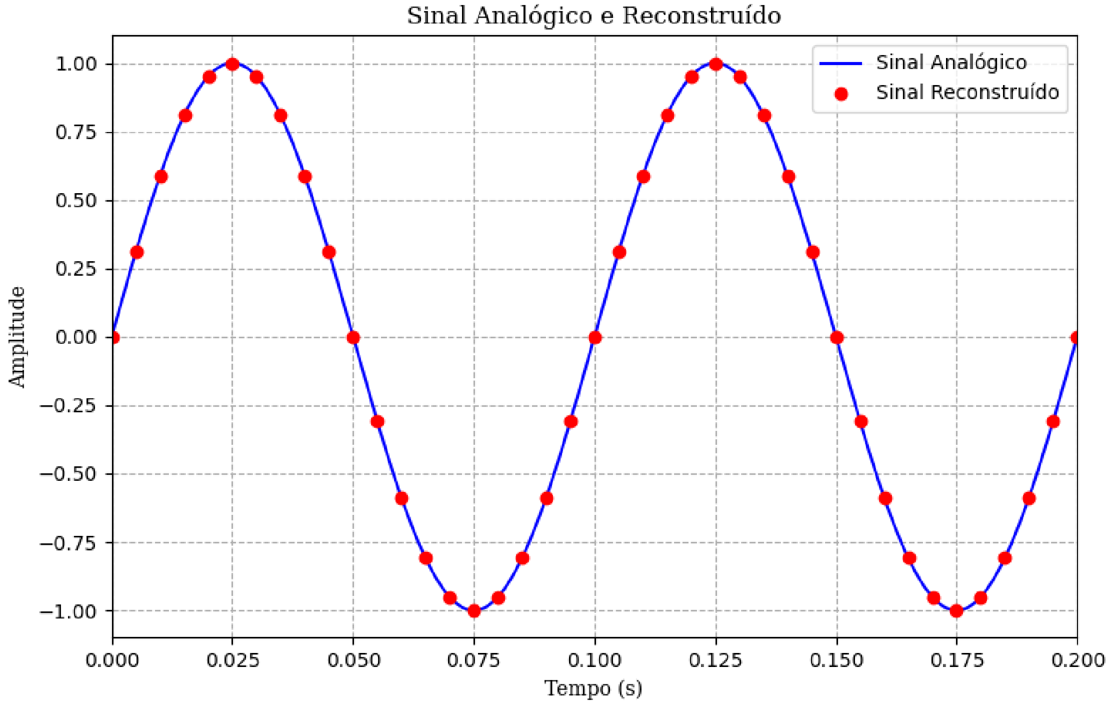
[4]:
```
# Sinal original
f0 = 10  # Frequência do sinal em Hz
t = np.linspace(0, 1, 10000)  # Tempo
signal = np.sin(2 * np.pi * f0 * t)  # Sinal senoidal

# Amostragem
fs = 200  # Frequência de amostragem em Hz
Ts = 1 / fs  # Período de amostragem
t_sampled = np.arange(0, t[-1], Ts)  # Tempo amostrado
signal_sampled = np.sin(2 * np.pi * f0 * t_sampled)  # Sinal amostrado

# Transformada de Fourier
# A transformada de Fourier é usada para converter um sinal do domínio do tempo␣
↪para o domínio da frequência.
spectrum = fft(signal_sampled)

# Filtro passa-baixa
cutoff_freq = 15  # Frequência de corte do filtro em Hz

freqs = np.fft.fftfreq(len(spectrum), Ts)
# np.fft.fftfreq() é usada para calcular as frequências correspondentes
# aos pontos na Transformada de Fourier do sinal.
spectrum_filtered = np.copy(spectrum)
spectrum_filtered[np.abs(freqs) > cutoff_freq] = 0

# Reconstrução do sinal
signal_reconstructed = ifft(spectrum_filtered)
```

[5]:
```
# Plotagem dos resultados
fig, axs = plt.subplots(2, 1, figsize=(8, 10))

# Primeiro gráfico: Sinal Analógico e Amostrado
axs[0].plot(t, signal, 'blue', label='Sinal Analógico')
axs[0].plot(t_sampled, signal_sampled, 'go', label='Sinal Amostrado')
axs[0].set_xlabel('Tempo (s)', fontname='serif')
axs[0].set_ylabel('Amplitude', fontname='serif')
axs[0].grid(True, linestyle='dashed')
axs[0].legend(fontsize=10)
axs[0].set_xlim(0, 0.2)
```

```python
# Segundo gráfico: Sinal Analógico e Reconstruído
axs[1].plot(t, signal, 'blue', label='Sinal Analógico')
axs[1].plot(t_sampled, signal_reconstructed, 'ro', markersize=6, label='Sinal Reconstruído')
axs[1].set_title('Sinal Analógico e Reconstruído', fontname='serif')
axs[1].set_xlabel('Tempo (s)', fontname='serif')
axs[1].set_ylabel('Amplitude', fontname='serif')
axs[1].grid(True, linestyle='dashed')
axs[1].legend(fontsize=10)
axs[1].set_xlim(0, 0.2)

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



Sinal Analógico e Reconstruído



RESPOSTA ESPERADA:

1.4 A.4) Responda:

Nesta aplicação, observando os resultados produzidos no item A.2 e A.3, em qual item o sinal foi reconstruído de forma adequada?

De acordo com os resultados que obtive, o sinal foi reconstruído de maneira mais apropriada foi A.3, no qual, foi utilizado um valor mais alto para a frequência de corte que foi 15Hz, com a frequência de corte igual a 15Hz foi possível preservar as frequências mais elevadas no sinal reconstruído, resultando em uma aproximação mais fiel ao sinal original.

1.5 A.5) Responda:

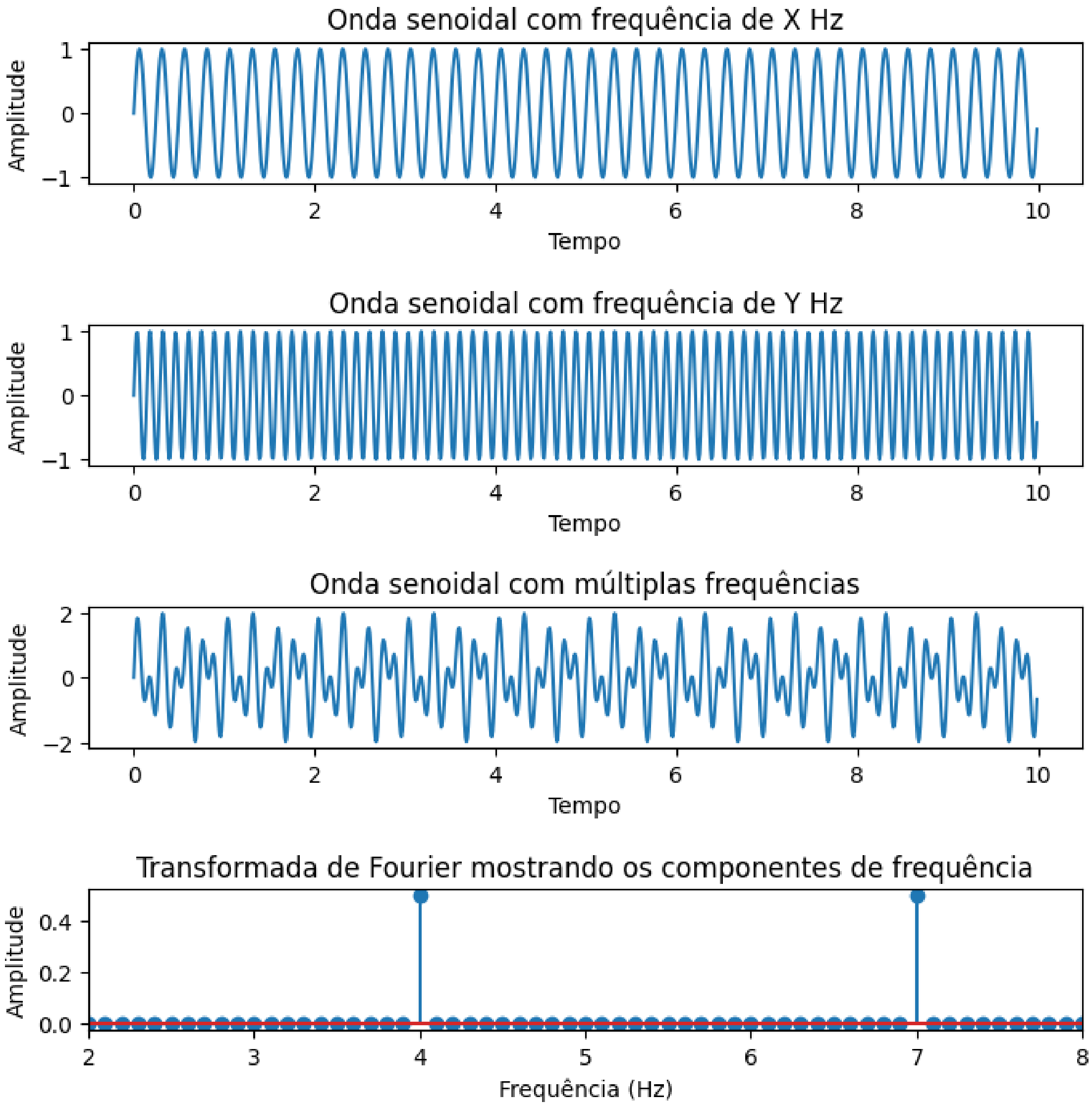
Nesta aplicação, qual foi o fator limitante, ou seja, o que prejudicou a reconstrução do sinal? Foi a frequência de amostragem ou a frequência de corte do filtro?

O fator que prejudica a reconstrução do sinal é a frequência de amostragem (fs), quando se comparado a frequência do sinal original (f0). Em casos da frequência de amostragem estiver significativamente abaixo da frequência do sinal original pode resultar em uma má reconstrução do sinal.

1.6 A.6) Rode o código abaixo:

[6]:
```python
# Importações
import numpy as np
import matplotlib.pyplot as plt

# Frequência de amostragem
Fs = 100

# Período de amostragem
Ts = 1 / Fs

# Início do período de tempo dos sinais
tempoInicial = 0

# Fim do período de tempo dos sinais
tempoFinal = 10

# Frequência dos sinais
frequenciaSinal1 = 4
frequenciaSinal2 = 7

# Pontos de tempo
tempo = np.arange(tempoInicial, tempoFinal, Ts)

# Criar duas ondas senoidais
Sinal1 = np.sin(2*np.pi*frequenciaSinal1*tempo)
Sinal2 = np.sin(2*np.pi*frequenciaSinal2*tempo)
```

```
# Criar subplot
figura, eixos = plt.subplots(4, 1, figsize=(8, 8))
plt.subplots_adjust(hspace=1)

# Representação no domínio do tempo para a onda senoidal 1
eixos[0].set_title('Onda senoidal com frequência de X Hz')
eixos[0].plot(tempo, Sinal1)
eixos[0].set_xlabel('Tempo')
eixos[0].set_ylabel('Amplitude')

# Representação no domínio do tempo para a onda senoidal 2
eixos[1].set_title('Onda senoidal com frequência de Y Hz')
eixos[1].plot(tempo, Sinal2)
eixos[1].set_xlabel('Tempo')
eixos[1].set_ylabel('Amplitude')

# Soma das ondas senoidais
Sinal3 = Sinal1 + Sinal2

# Representação no domínio do tempo da onda senoidal resultante
eixos[2].set_title('Onda senoidal com múltiplas frequências')
eixos[2].plot(tempo, Sinal3)
eixos[2].set_xlabel('Tempo')
eixos[2].set_ylabel('Amplitude')

# Representação no domínio da frequência
transformadaFourier = np.fft.fft(Sinal3)/len(Sinal3)
transformadaFourier = transformadaFourier[range(int(len(Sinal3)/2))]

contagemPontosTempo = len(Sinal3)
valores = np.arange(int(contagemPontosTempo/2))
periodoTempo = contagemPontosTempo/Fs
frequencias = valores/periodoTempo

# Representação no domínio da frequência
eixos[3].set_title('Transformada de Fourier mostrando os componentes de␣
↪frequência')
eixos[3].stem(frequencias, abs(transformadaFourier))
eixos[3].set_xlabel('Frequência (Hz)')
eixos[3].set_ylabel('Amplitude')
eixos[3].set_xlim(2, 8)

plt.show()
```



1.7 A.7) Após rodar o código fornecido em A.6, responda:

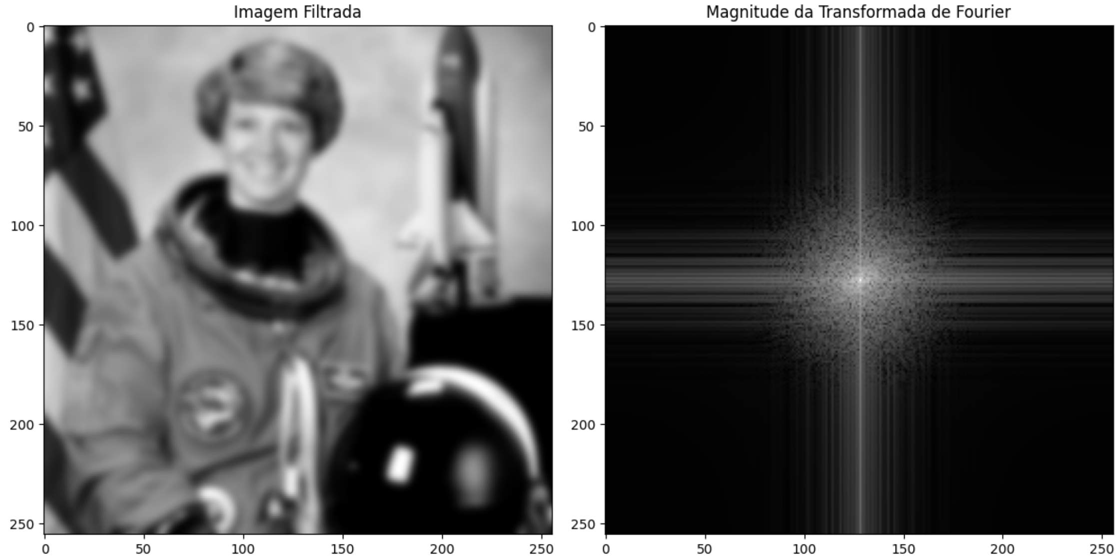
Segundo o gráfico gerado via Transformada de Fourier, quais frequências estão contidas no sinal senoidal com múltiplas frequências?

É possível visualizar no gráfico as frequências contidas no sinal senoidal, nas quais, são representadas por picos que correspondem às frequências originais, aproximadamente 4 Hz e 7 Hz. Os picos menores são artefatos da transformada e da discrepância entre as frequências reais e as amostras usadas.

2 Tópico II - Conceitos básicos de processamentos de Sinais: Transformada de Fourier em Imagens

2.1 B.1) Rode o código fornecido:

[7]:
```python
# Importações
import numpy as np
import matplotlib.pyplot as plt
from skimage import data, color, io, filters
from skimage.transform import resize
from skimage.filters import gaussian
from scipy.fftpack import fftshift, fft2

# Carrega uma imagem de exemplo (neste caso, uma imagem da astronauta)
image = data.astronaut()

# Converte a imagem para escala de cinza
gray_image = color.rgb2gray(image)

# Redimensiona a imagem para um tamanho menor
resized_image = resize(gray_image, (256, 256), mode='reflect',␣
 ↪anti_aliasing=True)

# Aplica um filtro gaussiano para suavizar a imagem
smoothed_image = gaussian(resized_image, sigma=2)

# Calcula a Transformada de Fourier 2D da imagem suavizada
fourier_transform = fft2(smoothed_image)

# Centraliza a transformada de Fourier
centered_fourier_transform = fftshift(fourier_transform)

# Configura inicial dos plots
plt.subplots(1, 2, figsize=(12, 12))

# Plota a imagem suavizada
plt.subplot(1, 2, 1)
plt.imshow(smoothed_image, cmap='gray')
plt.title('Imagem Filtrada')

# Plota a magnitude da transformada de Fourier
plt.subplot(1, 2, 2)
plt.imshow(np.log1p(np.abs(centered_fourier_transform)), cmap='gray')
plt.title('Magnitude da Transformada de Fourier')

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

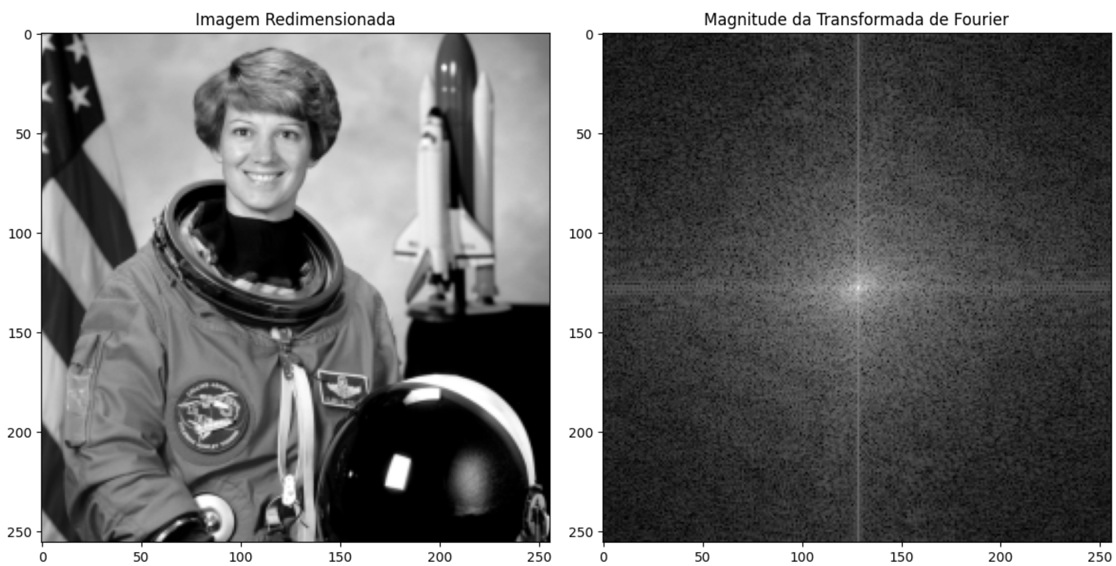


2.2 B.2) Após rodar o código fornecido em B.1, replique o código fornecido. Porém, não filtre a imagem. Apenas empregue resized_image como objeto de análise.

[8]:

```python
import numpy as np
import matplotlib.pyplot as plt
from skimage import data, color, io, filters
from skimage.transform import resize
from skimage.filters import gaussian
from scipy.fftpack import fftshift, fft2

# Carrega uma imagem de exemplo (neste caso, uma imagem da astronauta)
image = data.astronaut()

# Converte a imagem para escala de cinza
gray_image = color.rgb2gray(image)

# Redimensiona a imagem para um tamanho menor
resized_image = resize(gray_image, (256, 256), mode='reflect',
    anti_aliasing=True)

# Configura inicial dos plots
plt.subplots(1, 2, figsize=(12, 12))

# Plota a imagem redimensionada
plt.subplot(1, 2, 1)
plt.imshow(resized_image, cmap='gray')
plt.title('Imagem Redimensionada')
```

```python
# Calcula a Transformada de Fourier 2D da imagem redimensionada
fourier_transform = fft2(resized_image)

# Centraliza a transformada de Fourier
centered_fourier_transform = fftshift(fourier_transform)

# Plota a magnitude da transformada de Fourier
plt.subplot(1, 2, 2)
plt.imshow(np.log1p(np.abs(centered_fourier_transform)), cmap='gray')
plt.title('Magnitude da Transformada de Fourier')

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



RESPOSTA ESPERADA:

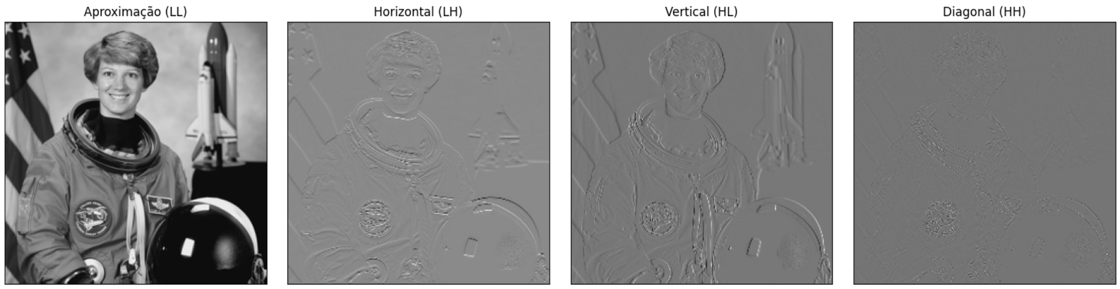
3 Tópico III - Conceitos básicos de processamentos de Sinais: Transformada Wavelet

3.1 Aplicação da Transformada Wavelet em imagens através da *dwt2*.

3.1.1 Carregue a imagem astronaut e, no espaço de cores de tons de cinza, aplique a Transformada Wavelet via *dwt2*. Empreque a família *'bior1.3'*. Plote LL, LH, HL e HH, conforme discutido em aula.

[9]:
```
import numpy as np
import matplotlib.pyplot as plt
import pywt
from skimage import data, color

# Carregando a imagem já no tom de cinza
img = data.astronaut()
gray_img = color.rgb2gray(img)

# Título para cada uma das subplots
titles = ['Aproximação (LL)', 'Horizontal (LH)', 'Vertical (HL)', 'Diagonal␣
  ↪(HH)']

# Realiza a Transformada Wavelet 2D utilizando a família 'bior1.3'
coeffs2 = pywt.dwt2(gray_img, 'bior1.3')

# Extrai os coeficientes da transformada em subbandas
LL, (LH, HL, HH) = coeffs2

# Cria uma figura para os plots
fig, axes = plt.subplots(1, 4, figsize=(16, 4))

# Lista das subbandas
subbands = [LL, LH, HL, HH]

# Loop para plotar cada subbanda
for i, ax in enumerate(axes):
    # Plota a subbanda atual
    ax.imshow(subbands[i], cmap='gray')

    # Define o título da subplot
    ax.set_title(titles[i], fontsize=12)

    # Remove os ticks dos eixos x e y
    ax.set_xticks([])
    ax.set_yticks([])

# Ajusta o layout da figura para melhor visualização
fig.tight_layout()
```

```
# Mostra a figura com os plots
plt.show()
```



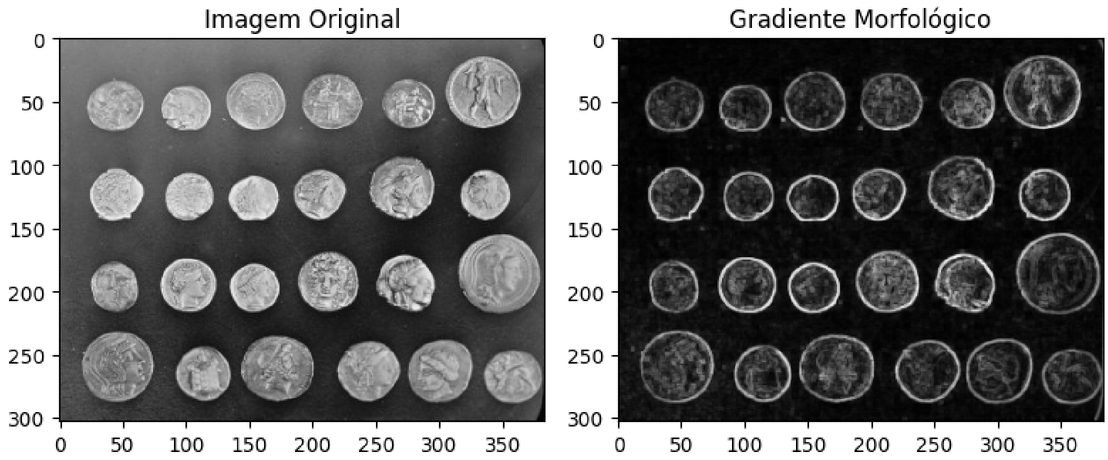
RESPOSTA ESPERADA:

4 Tópico IV - Operações Morfológicas em Imagens

4.1 Complete o código abaixo de forma que seja possível aplicar o gradiente morfológico na imagem composta por moedas:

[15]:
```
# RESPOSTA:

import numpy as np
import matplotlib.pyplot as plt
import cv2
from skimage import data, color

image = data.coins()

# Elemento estruturante para o gradiente morfológico
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))

# Aplicar o gradiente morfológico
gradient_image = cv2.morphologyEx(image, cv2.MORPH_GRADIENT, kernel)

# Plotar as imagens
plt.figure(figsize=(8, 4))

# Imagem original
plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Imagem Original')
```

```
# Imagem após o gradiente morfológico
plt.subplot(1, 2, 2)
plt.imshow(gradient_image, cmap='gray')
plt.title('Gradiente Morfológico')

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



RESULTADO ESPERADO: