Experimento 1

Processamento Digital de Sinais - II

Equipe:

- Arthur Munhoz (GRR20177243)
- Vinícius Parede (GRR20172137)

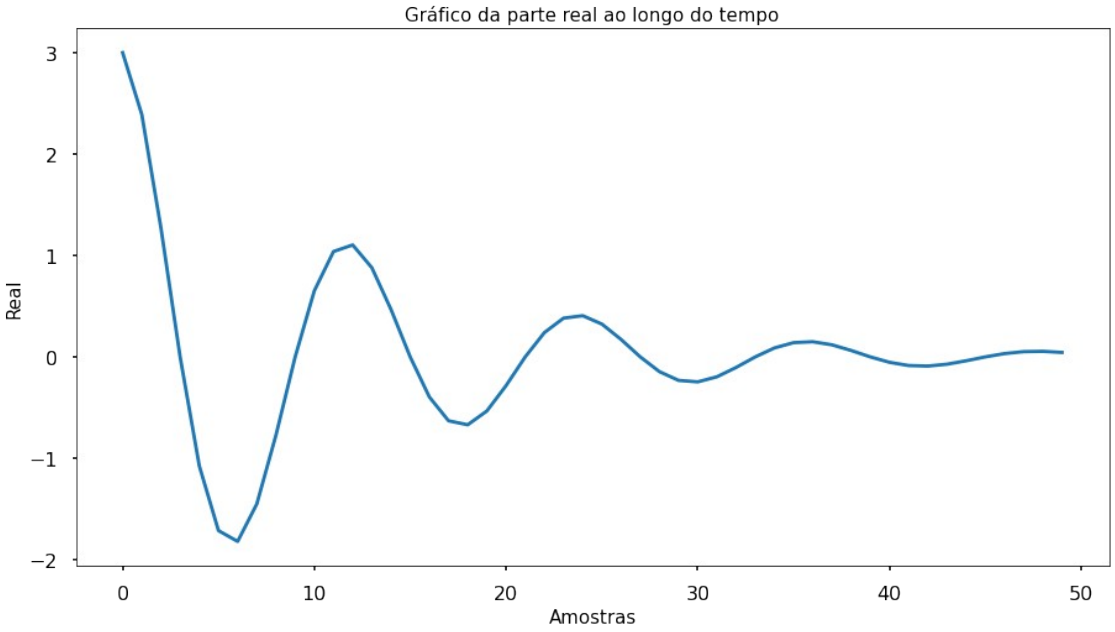
```
# Bibliotecas
import numpy as np
import matplotlib.pyplot as plt
import IPython
import math
from scipy.io import wavfile
plt.style.use('seaborn-poster')
```

1) Gere o sinal x de acordo com o código abaixo:

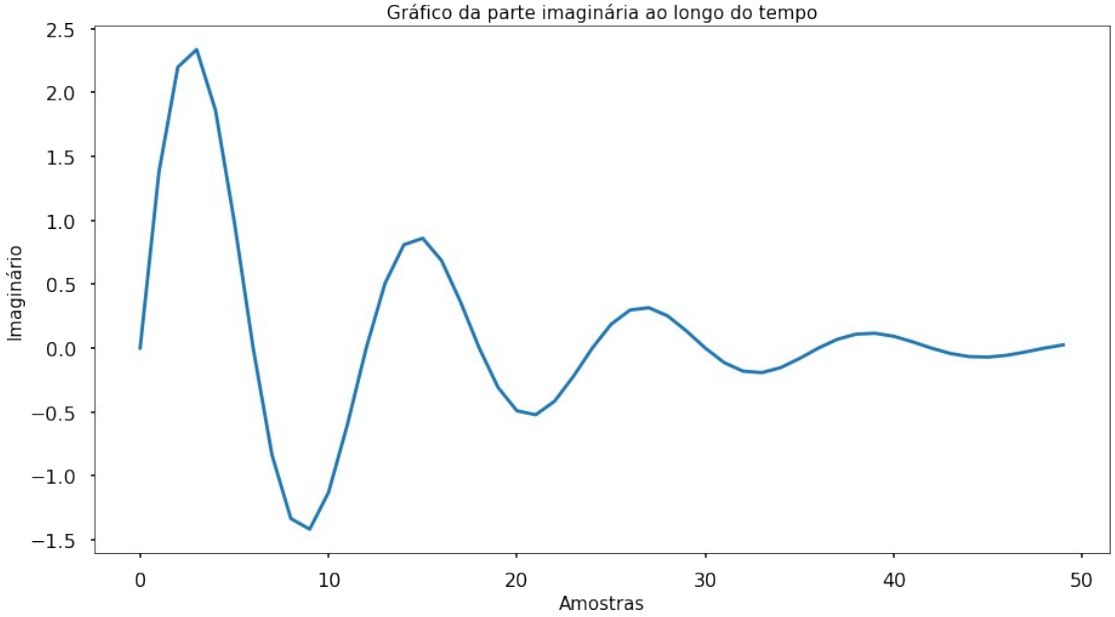
```
# Amplitude = 3
A = 3

# Decaimento
d = complex(-(1/12), (math.pi/6))

# Vetor tempo de 50 posições
n = np.linspace(0,49)

# Sinal
x = A * np.exp(d * n)

# Retirar parte real
real = [iterator.real for iterator in x]

# Retirar parte imaginária
imaginaria = [iterator.imag for iterator in x]
```

a. Mostre o gráfico da parte real e o gráfico da parte imaginária

```
# Gráfico parte real ao longo do tempo
plt.figure(figsize=(15, 8))
plt.plot(n, real)
plt.ylabel('Real', size=15)
plt.xlabel('Amostras', size=15)
plt.title('Gráfico da parte real ao longo do tempo', size=15)
plt.show()
```




```
# Gráfico parte imaginária ao longo do tempo
plt.figure(figsize=(15, 8))
plt.plot(n, imaginaria)
plt.ylabel('Imaginário', size=15)
plt.xlabel('Amostras', size=15)
plt.title('Gráfico da parte imaginária ao longo do tempo', size=15)
plt.show()
```



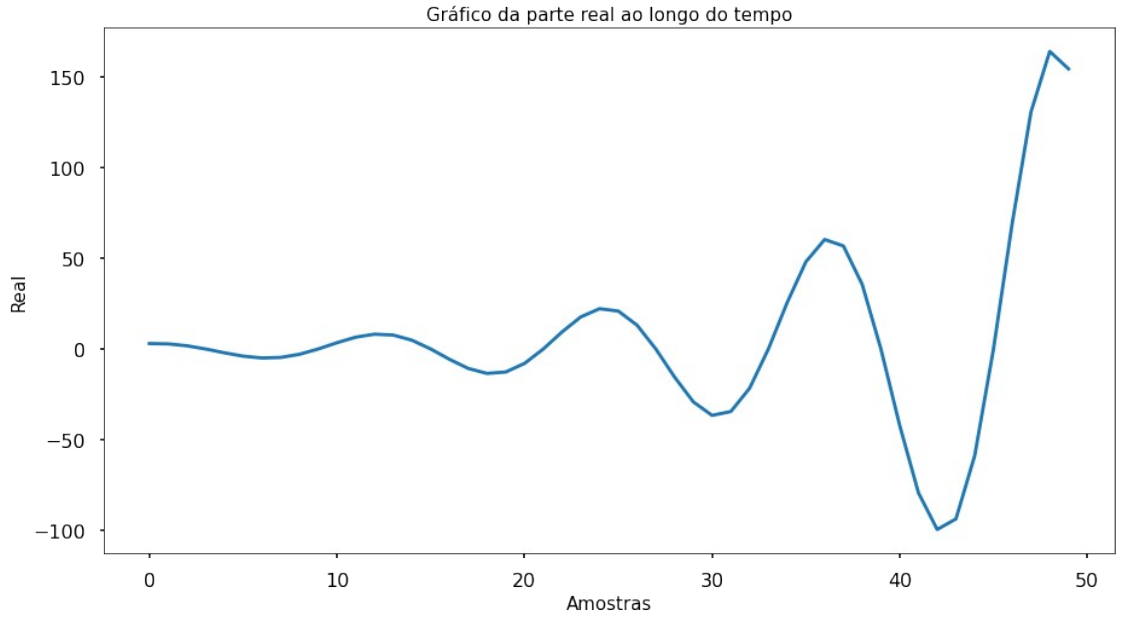

```
# Gráfico da parte real e imaginária em conjunto
plt.figure(figsize=(15, 8))
plt.plot(real, imaginaria)
plt.ylabel("Imaginário", size=15)
```

```
plt.xlabel("Real", size=15)
plt.title("Gráfico da parte real e imaginária em conjunto", size=15)
```

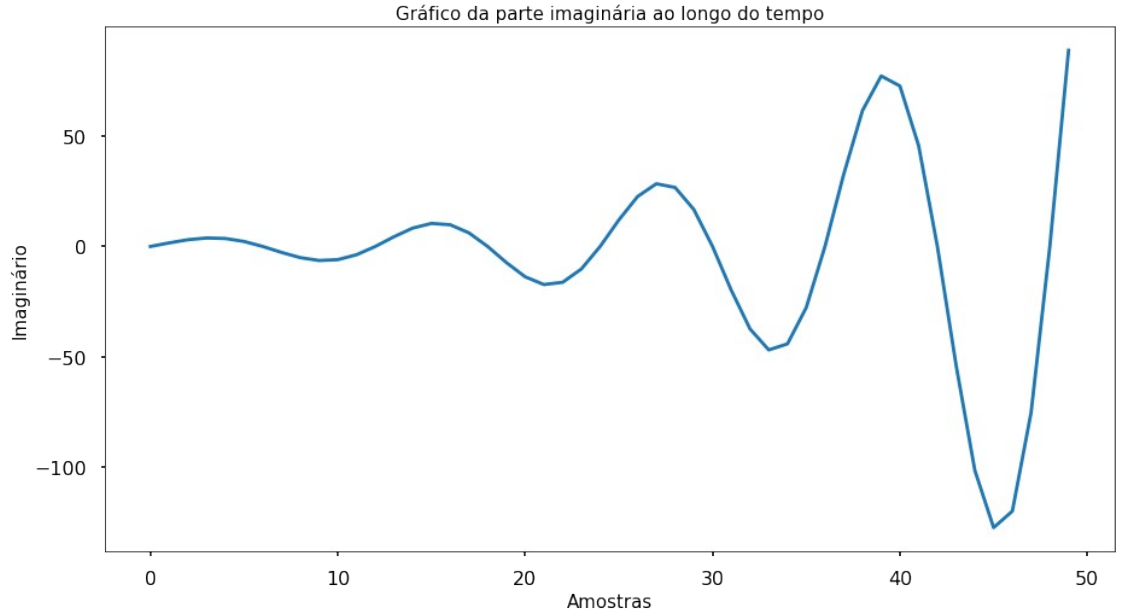
```
Text(0.5, 1.0, 'Gráfico da parte real e imaginária em conjunto')
```

b. Altere o sinal do parâmetro d para positivo e mostre o gráfico da parte real e da parte imaginária do sinal alterado

```
# Decaimento Positivo
d = complex((1/12), (math.pi/6))

# Sinal
x = A * np.exp(d * n)

# Retirar parte real
real = [iterator.real for iterator in x]

# Retirar parte imaginária
imaginaria = [iterator.imag for iterator in x]

# Gráfico parte real ao longo do tempo
plt.figure(figsize=(15, 8))
plt.plot(n, real)
plt.ylabel('Real', size=15)
plt.xlabel('Amostras', size=15)
plt.title('Gráfico da parte real ao longo do tempo', size=15)
plt.show()
```



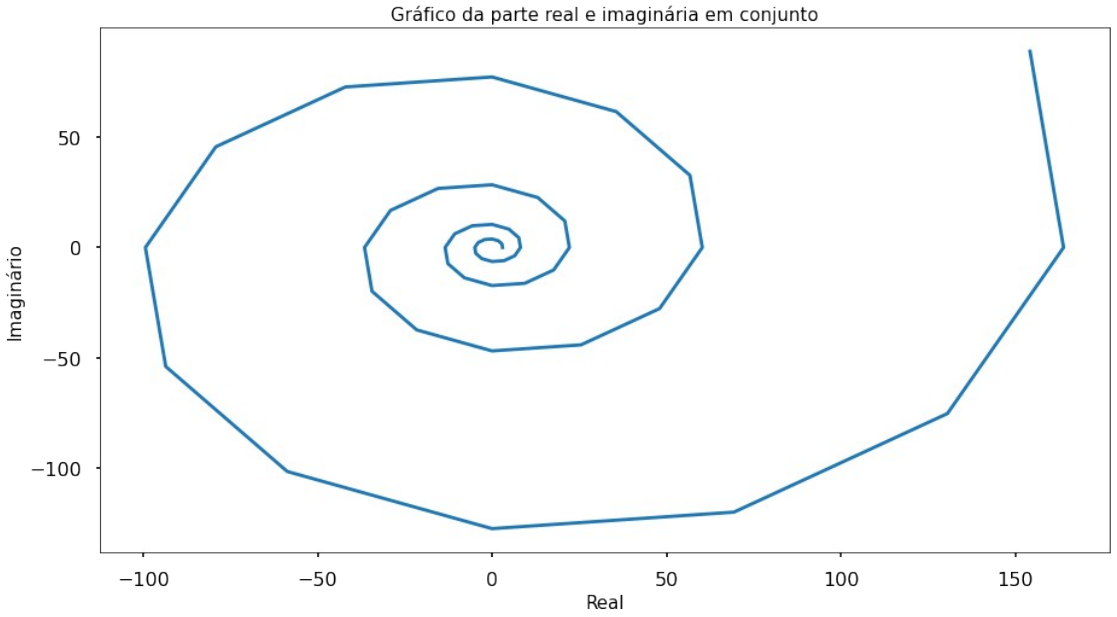

```
# Gráfico parte imaginária ao longo do tempo
plt.figure(figsize=(15, 8))
plt.plot(n, imaginaria)
plt.ylabel('Imaginário', size=15)
plt.xlabel('Amostras', size=15)
plt.title('Gráfico da parte imaginária ao longo do tempo', size=15)
plt.show()
```




```
# Gráfico da parte real e imaginária em conjunto
plt.figure(figsize=(15, 8))
plt.plot(real, imaginaria)
plt.ylabel("Imaginário", size=15)
```

```
plt.xlabel("Real", size=15)
plt.title("Gráfico da parte real e imaginária em conjunto", size=15)
```

```
Text(0.5, 1.0, 'Gráfico da parte real e imaginária em conjunto')
```



c. Compare e comente os gráficos gerados

R:

- O primeiro gráfico, correspondente ao gráfico da parte real, diz respeito a uma função que possui um fator de amortecimento ç e que possui um decaimento exponencial.
- O segundo gráfico, correspondente ao gráfico da parte imaginária, também diz respeito a uma funççao que possuiu um decaimento exponencial e um fator de amortecimento ç. A diferença para o primeiro grafico é apenas o deslocamento em fase.
- O terceiro gráfico, correspondente ao gráfico da parte real vs parte imaginária, diz respeito à espiral logaritimica, isto é, é a curva que forma com todas as retas, situadas no seu plano e passando por um ponto fixo desse plano, um angulo constante e corresponde a distancia à origem, de um ponto da curva em função de teta.
- Quando mudamos o sinal do parametro D para positivo, partimos da origigem com um sinal amortecido e conforme deslocamos no eixo X o sinal tende a oscilar

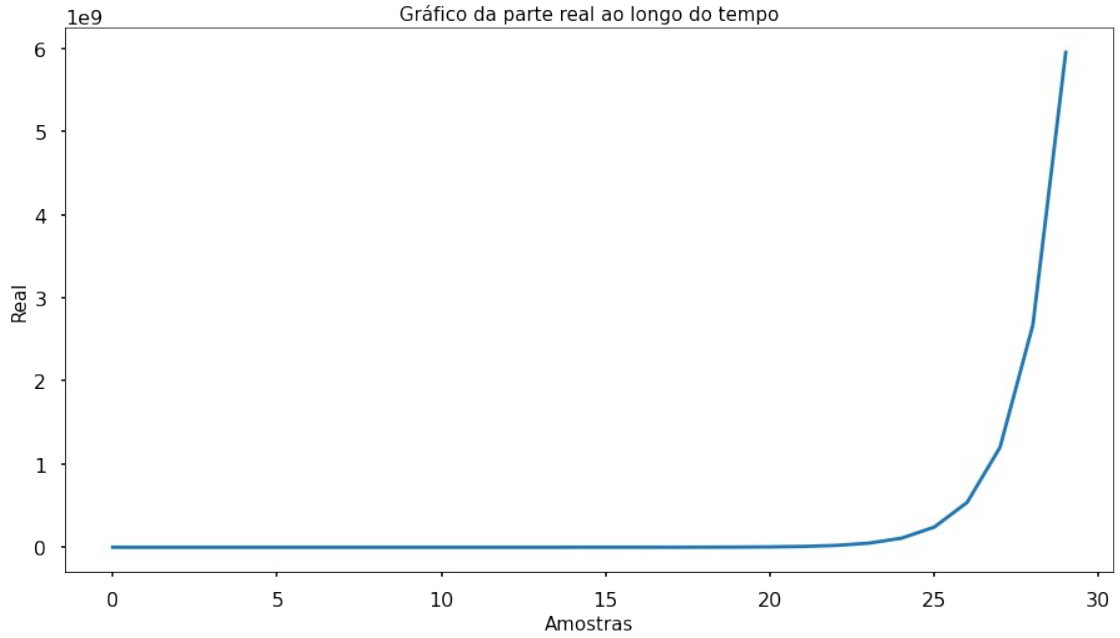
2) Gere o sinal x = A * exp (d * n), com n = 0:29, d = 0,8 e A = 0,5

```
# Amplitude = 0,5
A = 0.5

# Decaimento
d = 0.8
```

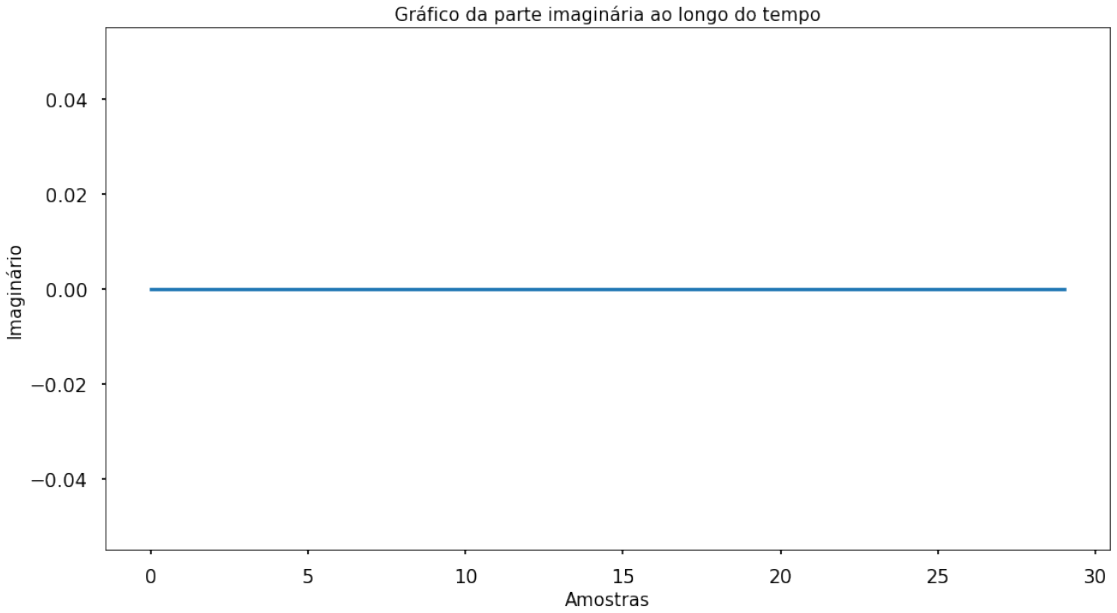
```
# Vetor tempo de 30 posições
n = np.linspace(0,29, 30)

# Sinal
x = A * np.exp(d * n)

# Retirar parte real
real = [iterator.real for iterator in x]

# Retirar parte imaginária
imaginaria = [iterator.imag for iterator in x]
```

a. Visualize o sinal gerado

```
# Gráfico parte real ao longo do tempo
plt.figure(figsize=(15, 8))
plt.plot(n, real)
plt.ylabel('Real', size=15)
plt.xlabel('Amostras', size=15)
plt.title('Gráfico da parte real ao longo do tempo', size=15)
plt.show()
```

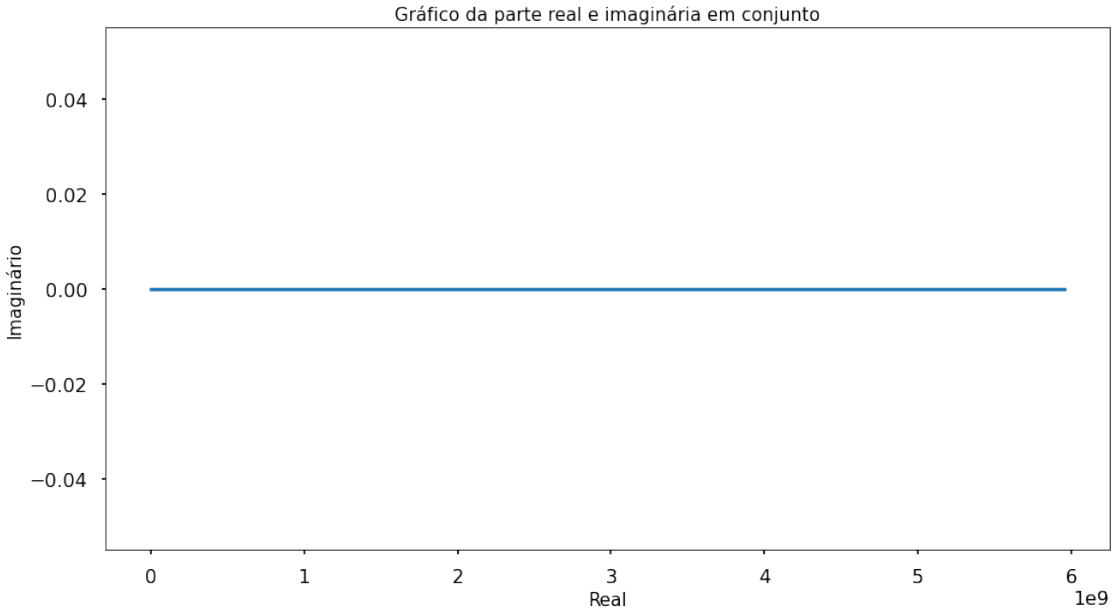



```
# Gráfico parte imaginária ao longo do tempo
plt.figure(figsize=(15, 8))
plt.plot(n, imaginaria)
plt.ylabel('Imaginário', size=15)
plt.xlabel('Amostras', size=15)
plt.title('Gráfico da parte imaginária ao longo do tempo', size=15)
plt.show()
```




```
# Gráfico da parte real e imaginária em conjunto
plt.figure(figsize=(15, 8))
plt.plot(real, imaginaria)
plt.ylabel("Imaginário", size=15)
plt.xlabel("Real", size=15)
plt.title("Gráfico da parte real e imaginária em conjunto", size=15)
```

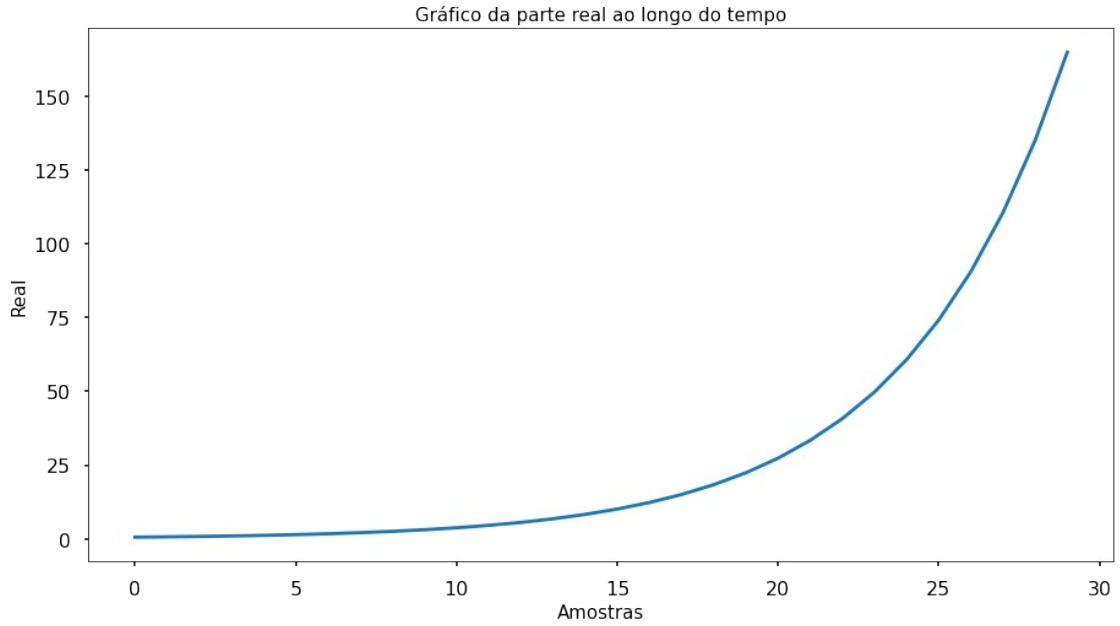
```
Text(0.5, 1.0, 'Gráfico da parte real e imaginária em conjunto')
```



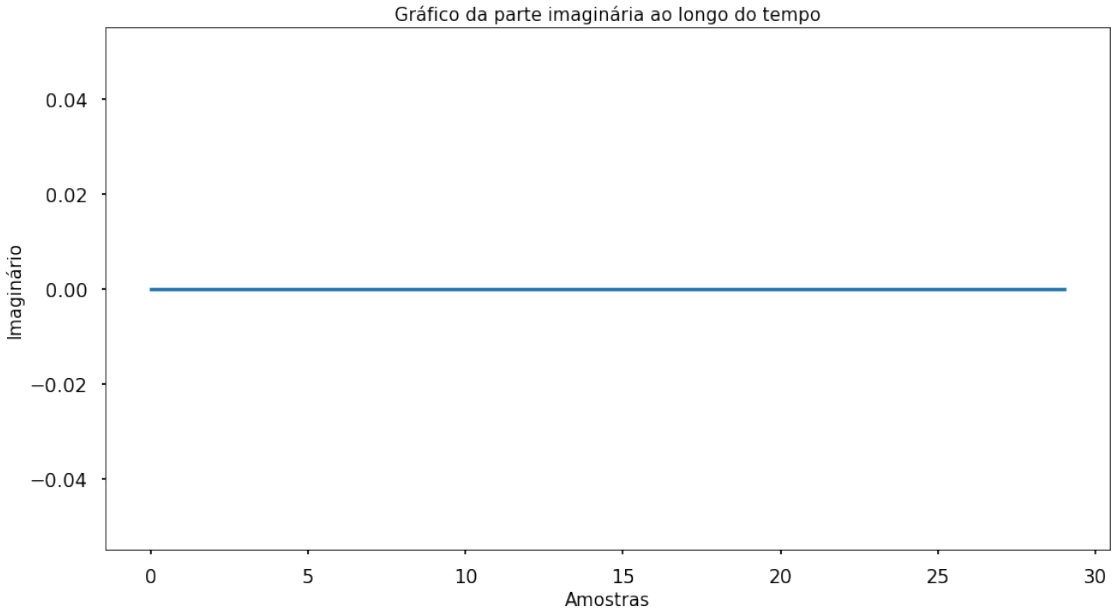
b. Altere o parâmetro d para 0,2 e visualize o sinal

```
# Decaimento Positivo
d = 0.2
```

```
# Sinal
x = A * np.exp(d * n)

# Retirar parte real
real = [iterator.real for iterator in x]

# Retirar parte imaginária
imaginaria = [iterator.imag for iterator in x]

# Gráfico parte real ao longo do tempo
plt.figure(figsize=(15, 8))
plt.plot(n, real)
plt.ylabel('Real', size=15)
plt.xlabel('Amostras', size=15)
plt.title('Gráfico da parte real ao longo do tempo', size=15)
plt.show()
```

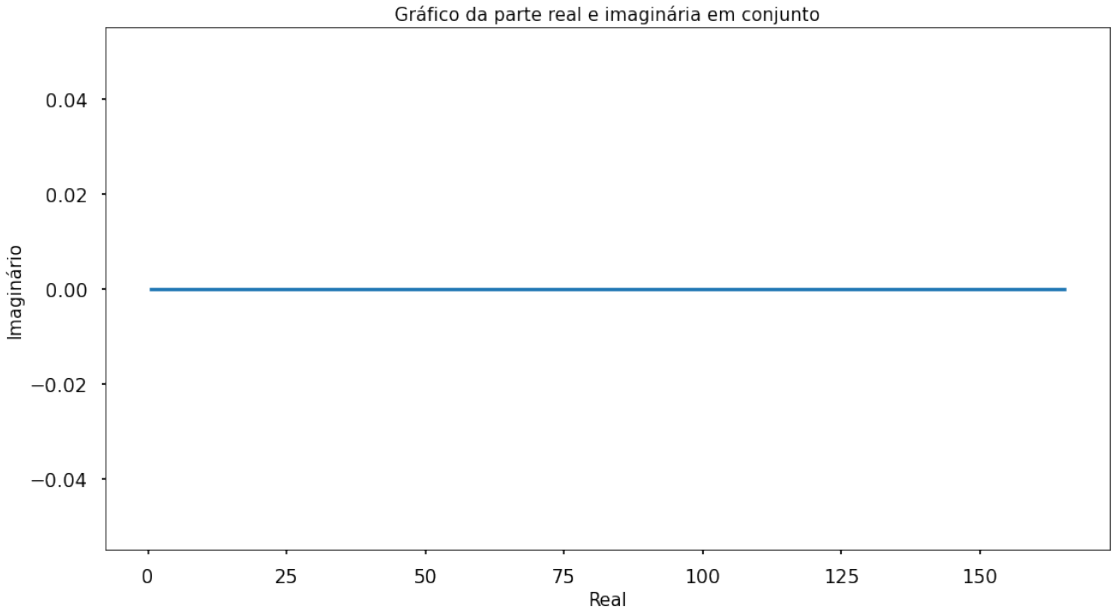



```
# Gráfico parte imaginária ao longo do tempo
plt.figure(figsize=(15, 8))
plt.plot(n, imaginaria)
plt.ylabel('Imaginário', size=15)
plt.xlabel('Amostras', size=15)
plt.title('Gráfico da parte imaginária ao longo do tempo', size=15)
plt.show()
```




```
# Gráfico da parte real e imaginária em conjunto
plt.figure(figsize=(15, 8))
plt.plot(real, imaginaria)
plt.ylabel("Imaginário", size=15)
plt.xlabel("Real", size=15)
plt.title("Gráfico da parte real e imaginária em conjunto", size=15)
```

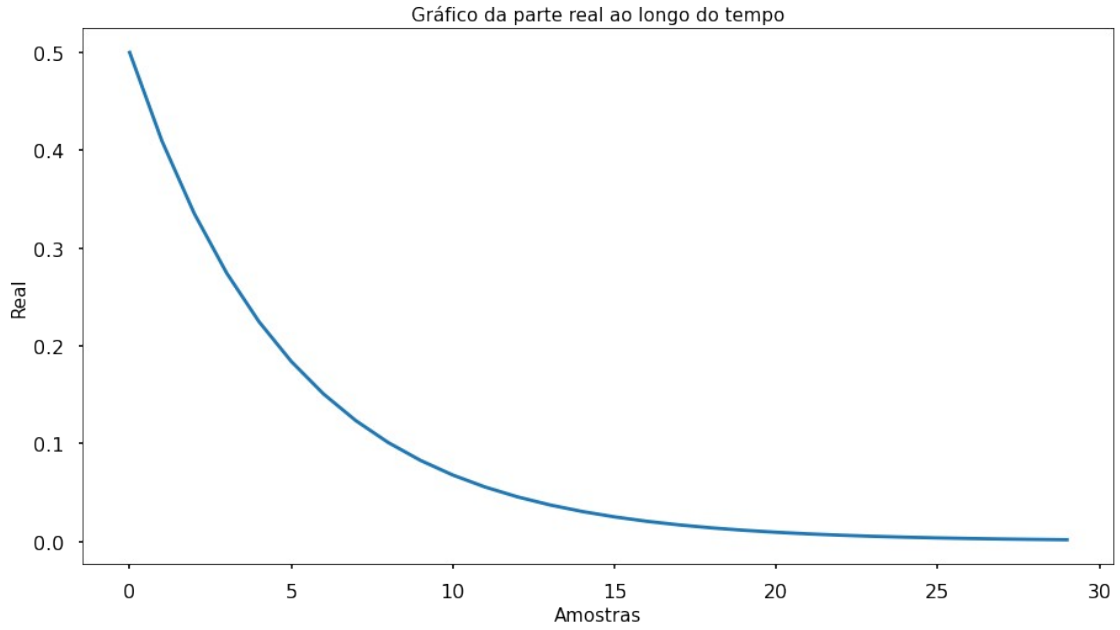
```
Text(0.5, 1.0, 'Gráfico da parte real e imaginária em conjunto')
```



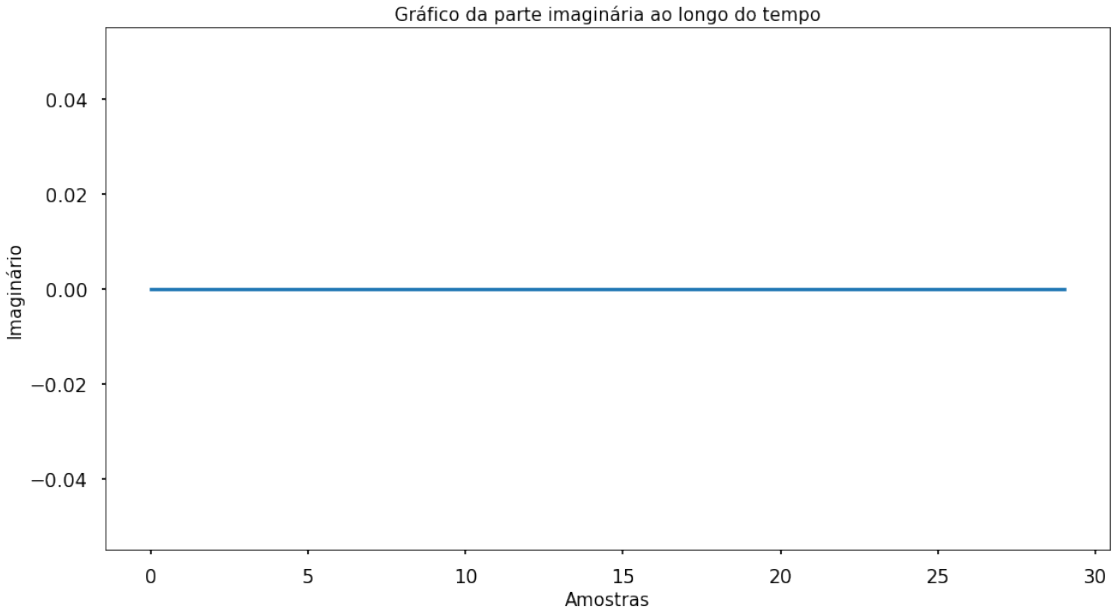
c. Altere o parâmetro d para -0,2 e visualize o sinal

```
# Decaimento Negativo
d = -0.2
```

```
# Sinal
x = A * np.exp(d * n)

# Retirar parte real
real = [iterator.real for iterator in x]

# Retirar parte imaginária
imaginaria = [iterator.imag for iterator in x]

# Gráfico parte real ao longo do tempo
plt.figure(figsize=(15, 8))
plt.plot(n, real)
plt.ylabel('Real', size=15)
plt.xlabel('Amostras', size=15)
plt.title('Gráfico da parte real ao longo do tempo', size=15)
plt.show()
```

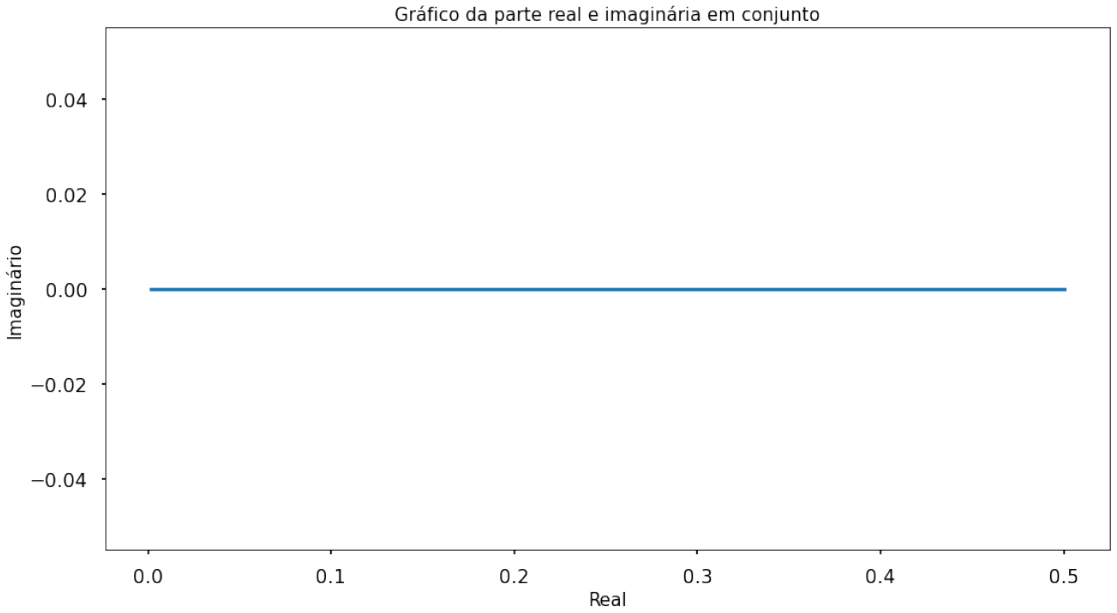



```
# Gráfico parte imaginária ao longo do tempo
plt.figure(figsize=(15, 8))
plt.plot(n, imaginaria)
plt.ylabel('Imaginário', size=15)
plt.xlabel('Amostras', size=15)
plt.title('Gráfico da parte imaginária ao longo do tempo', size=15)
plt.show()
```




```
# Gráfico da parte real e imaginária em conjunto
plt.figure(figsize=(15, 8))
plt.plot(real, imaginaria)
plt.ylabel("Imaginário", size=15)
plt.xlabel("Real", size=15)
plt.title("Gráfico da parte real e imaginária em conjunto", size=15)
```

```
Text(0.5, 1.0, 'Gráfico da parte real e imaginária em conjunto')
```



d. Compare e comente os fráficos gerados

R:

- No gráfico 1, onde d=0.8, tem-se um decaimento mais abrupto.
- No gráfico 2, onde d=0.2, tem-se um decaimento menos abdupto em relação ao gráfico 1.
- No gráfico 3, onde d=-0.2, tem-se um decaimento igual ao gráfico dois, mas um decaimento negativo.

Quando mais proximo de 1 for o fator d, mais ortogonal é o decaimento em relação ao eixo x. E quanto menor o d, o angulo entre eixo X e eixo Y é menor.

Por via de regra:

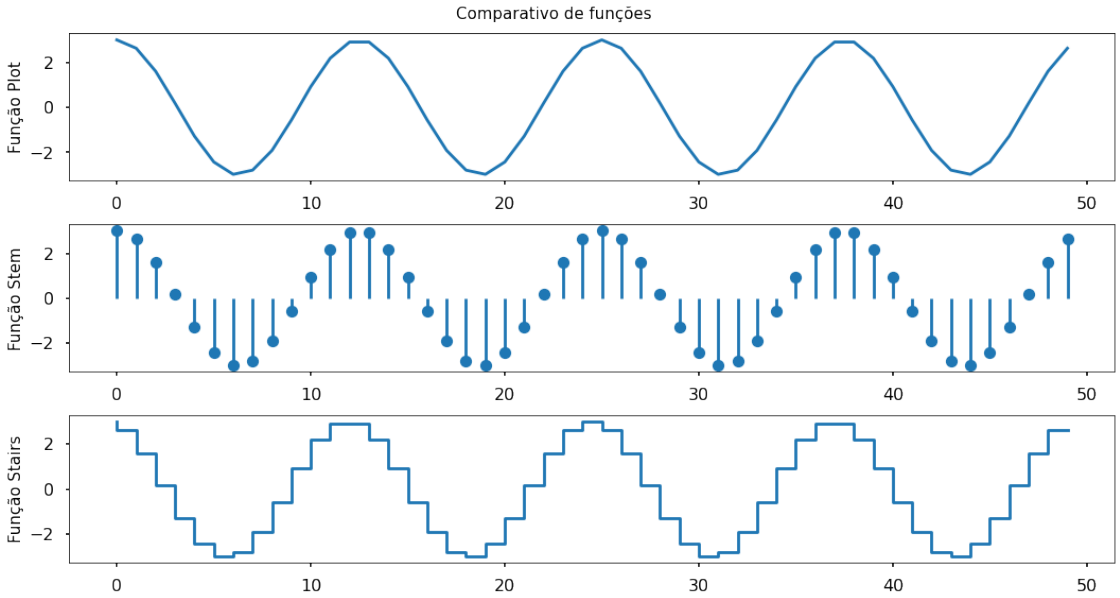
1. Maior d, mais abrupto é o decaimento. Angulo entre o eixo X e o eixo Y é maior.
2. Menor d, menos abrupto é o decaimento. Angulo entre eixo X e eixo Y é menor.
3. Quando d é positivo, o eixo Y tende a infinito e o eixo X tende a uma constante.
4. Quando d é negativo, o eixo X tente a infinito e o eixo Y tente a uma constante que não necessariamente é 0.

3) Gere o sinal x = 3 * cos(2 * pi * 0.08 * 0:49)

```
# Amplitude = 3
A = 3

# Vetor tempo de 50 posições
n = np.linspace(0,49)

# Sinal
x = A * np.cos(2 * np.pi * 0.08 * n)
```

a. Visualize o sinal gerado com as funções stem, plot e stairs

```
fig, axes = plt.subplots(nrows=3, figsize=(15,8))

axes[0].plot(n, x)
axes[1].stem(n, x, basefmt=" ")
axes[2].step(n, x)

axes[0].set_ylabel("Função Plot", size=15)
axes[1].set_ylabel("Função Stem", size=15)
axes[2].set_ylabel("Função Stairs", size=15)

fig.suptitle("Comparativo de funções", size=15)
fig.tight_layout()
```

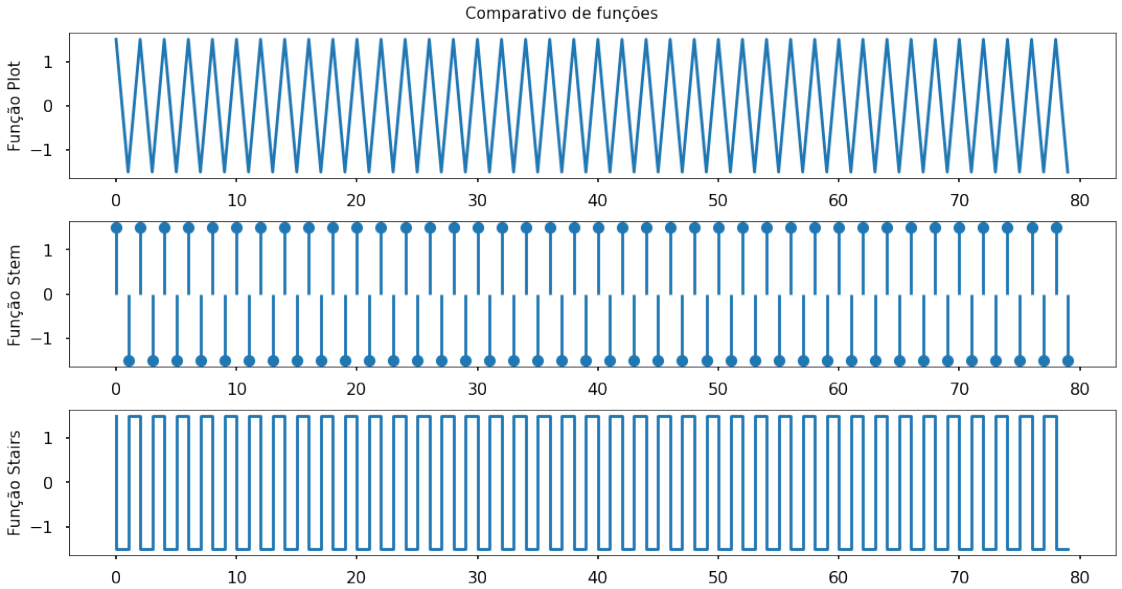


b. Modifique o tamanho da sequência para 80, a frequência para 0,5 e a amplitude para 1,5

```
# Amplitude = 1,5
A = 1.5

# Vetor tempo de 80 posições
n = np.linspace(0,79, 80)

# Sinal
x = A * np.cos(2 * np.pi * 0.5 * n)
```

c. Visualize o sinal modificado com as funções stem, plot e stairs

```
fig, axes = plt.subplots(nrows=3, figsize=(15,8))

axes[0].plot(n, x)
axes[1].stem(n, x, basefmt=" ")
axes[2].step(n, x)

axes[0].set_ylabel("Função Plot", size=15)
axes[1].set_ylabel("Função Stem", size=15)
axes[2].set_ylabel("Função Stairs", size=15)

fig.suptitle("Comparativo de funções", size=15)
fig.tight_layout()
```



d. Vizualize e comente as diferenças entre as três funcções de visualização

R:

- Quando aplicamos um uma sequencia de 50 posições cuja amplitude é 3 e a frequencia em 0.08, a função plot une todos os pontos do eixo N a fim de tornar o sinal `continuo`. Já a função stem mostra os pontos discretizados no tempo. Por último a função stairs tenta mostrar em formado de escada enfatizando a diferença entre os pontos do eixo n. Nesse caso, como a sequencia é menor que o grafico do item b, a amplitude maior e a frequencia menor, os gráficos não sao distorcidos.

- Quando diminuimos a amplitude e a usamos a frequencia de 0.5 e uma sequencia de 80 posições, temos uma amplitude menor para mostrar nos gráficos e uma sequencia maior, ou seja, a variação de amplitude entre os picos é menor, varia de -1.5 a 1.5 e isso faz com que as funções unam os pontos incorretamente.

- Quando temos a frequencia de 0.08, temos um periodo `1/0.08 = 12,5` que, neste caso, representa o numero de pontos por ciclo. Quando aumentamos a frequencia para 0.5 temos um periodo de `1/0.5 = 2` que, neste caso, também representa o número de pontos por ciclo. Portanto, no item a temos mais pontos por ciclo do cosseno e, portanto, o gráfico é mais detalhado. Já no item b, temos 2 pontos por ciclo e, portanto, o gráfico possuiu dois pontos apenas para expressar uma cossenóide.

- A frequencia de amostragem deve ser 2x maior que a frequencia do sistema

4) Gere o sinal x = 2 * t * 0.9^t com t = 0:49

```
# Vetor tempo de 50 posições
t = np.linspace(0, 49, 50)
```

```
# Sinal
sinal = 2 * t * pow(0.9, t)
```

a. Adicione ruído gaussiano no sinal (função randn)

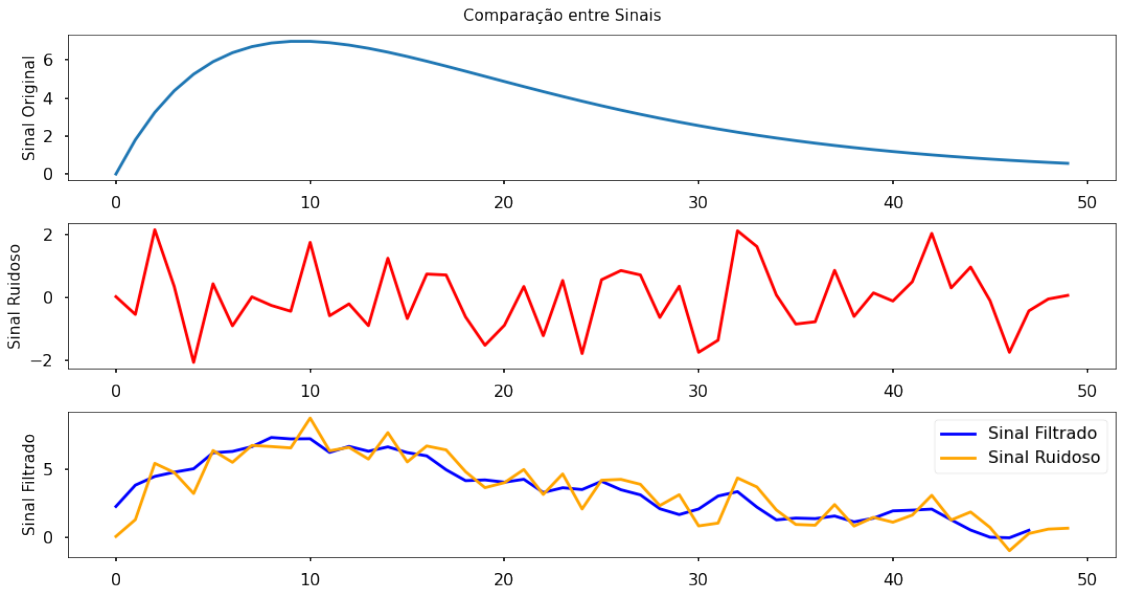
```
# Ruído Gaussiano
ruido = np.random.randn(1, 50).T
ruido_list = list()
for _, element in enumerate(ruido):
    ruido_list.append(element[0])
ruido = np.array(ruido_list)

# Sinal ruidoso
z = ruido + sinal
```

b. Aplique o filtro de médio móvel de 3 pontos do sinal ruidoso

```
def moving_average(a, n) :
    ret = np.cumsum(a, dtype=float)
    ret[n:] = ret[n:] - ret[:-n]
    return ret[n - 1:] / n

# Sinal Filtrado - Média Móvel de 3 períodos
filtro = moving_average(z, 3)
```

c. Vizualize o sinal original, o sinal ruidoso e o sinal filtrado

```
fig, ax = plt.subplots(nrows=3, ncols=1, figsize=(15, 8))

ax[0].plot(t, sinal)
ax[0].set_ylabel('Sinal Original', size=15)

ax[1].plot(t, ruido, color='red')
ax[1].set_ylabel('Sinal Ruidoso', size=15)

ax[2].plot(filtro, color='blue', label='Sinal Filtrado')
ax[2].plot(t, z, color='orange', label='Sinal Ruidoso')
ax[2].set_ylabel('Sinal Filtrado', size=15)
ax[2].legend()

fig.suptitle("Comparação entre Sinais", size=15)
fig.tight_layout()
```



d. O filtro aplicado é causal? Justifique sua resposta

R:

É causal pois a média movél é um somatório que apenas leva em conta valores atuais e do passado

5) Gere o sinal x = [zeros(1,10) ones(1,40) zeros(1,10)]

```
x = [np.zeros(10), np.ones(40), np.zeros(10)]
x = np.concatenate(x)
```

a. Gere um sinal y que seja a convulução de x com eles mesmo

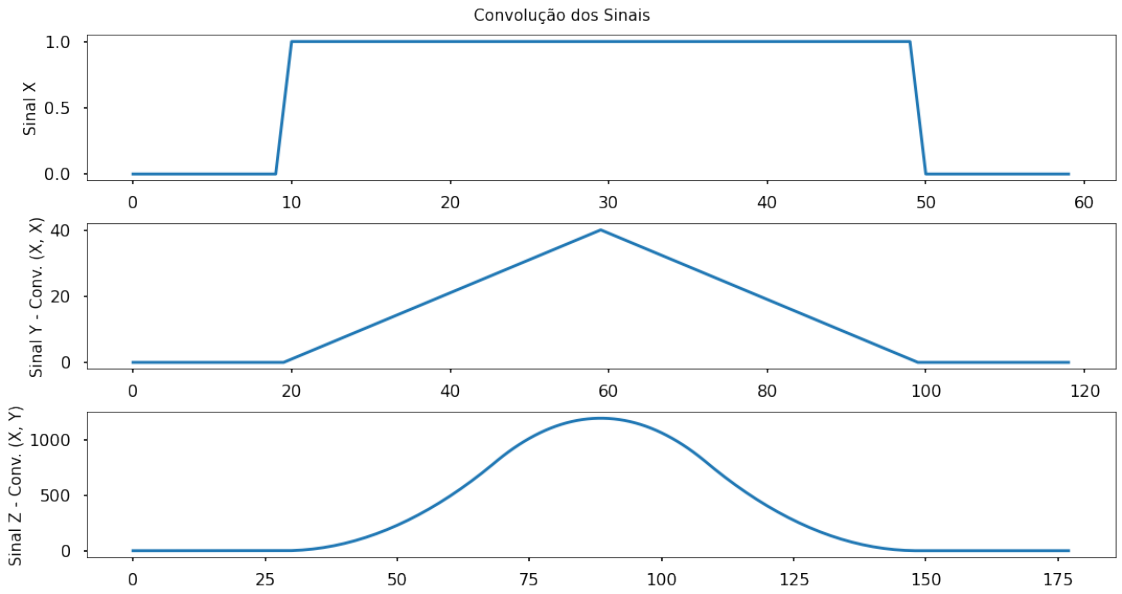
```
y = np.convolve(x, x)
```

b. Gere um sinal z que seja a convolução de x com y

```
z = np.convolve(x, y)
```

c. Visualize x, y, z

```
fig, axes = plt.subplots(nrows=3, figsize=(15,8))

axes[0].plot(x)
axes[1].plot(y)
axes[2].plot(z)

axes[0].set_ylabel("Sinal X", size=15)
axes[1].set_ylabel("Sinal Y - Conv. (X, X)", size=15)
axes[2].set_ylabel("Sinal Z - Conv. (X, Y)", size=15)

fig.suptitle("Convolução dos Sinais", size=15)
fig.tight_layout()
```



d. Comente os resultados de a e b

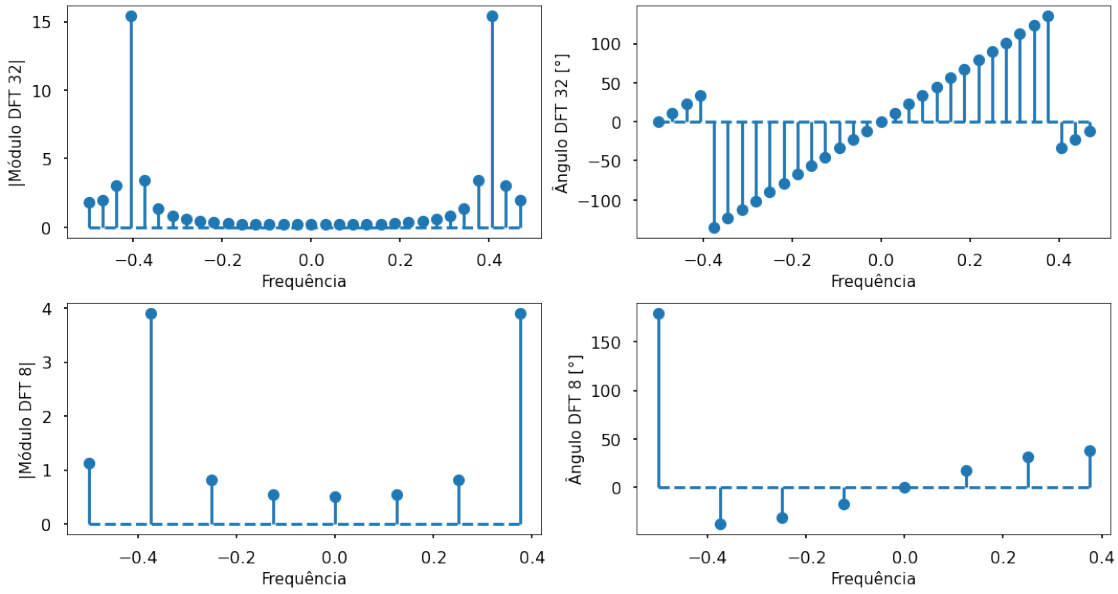
R:

- Convolução é um jeito matematico de combinar dois sinais para formar um terceiro. Sistemas sao descritos como resposta ao impulso. Para a convolução de dois sinais iguais, quando os sinais de encontram no deslocamento do tempo um sinal é linear é gerado tem seu ângulo calculado baseado no dobro da largura do sinal x. Quando os sinais são perfeitamente sobrepostos ocorre o pico do sinal da convolução, conforme o sinal que é deslocado no tempo passa da metade do sinal fixo, o sinal convoluido tem seu sinal trocado e começa a ser uma funçao linear com decaimento negativo. Isso é demonstrado no grafico da convolução de dois sinais step iguais.
- Quando convoluimos um sinal triangular com o sinal step, retangular, a saída é um grafico normalizado, conforme demonstadrado.

6) A Transformada Rápidade Fourier (FFT) X[k] de uma sequência finita x[n] pode ser computada no Octave / Matlabusando a função fft. É possível usar a função na forma fft(x), gerando um sinal do mesmo tamanho de x, ou na forma fft(x,L) computando a DFT em L pontos. Crie uma sequência x = cos(2*pi*0.4*n) de tamanho 32. Compute e visualizea DFT desta sequência. Repita a operação com L=8 para a DFT. Visualize usando stem a magnitude (abs) e a fase (angle) dos dois sinais obtidos.

```
# Definindo o tamanho
n = np.linspace(0, 31, 32)

# Sinal
x = np.cos(2 * np.pi * 0.4 * n)
```

```
# DFT (Discret Fourier Transform) -> Tamanho 32
fft_32 = np.fft.fft(x, n=32)
fft_freq_32 = np.fft.fftfreq(fft_32.size, d=1)

modulo_32 = np.absolute(fft_32)
fase_32 = np.angle(fft_32)

# DFT (Discret Fourier Transform) -> Tamanho 8
fft_8 = np.fft.fft(x, n=8)
fft_freq_8 = np.fft.fftfreq(fft_8.size, d=1)
modulo_8 = np.absolute(fft_8)
fase_8 =  np.angle(fft_8)

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(15, 8))

axes[0][0].stem(fft_freq_32, modulo_32, basefmt='--')
axes[0][1].stem(fft_freq_32, np.rad2deg(fase_32), basefmt='--')
axes[0][0].set_ylabel("|Módulo DFT 32|", size=15)
axes[0][0].set_xlabel("Frequência", size=15)
axes[0][1].set_ylabel("Ângulo DFT 32 [°]", size=15)
axes[0][1].set_xlabel("Frequência", size=15)


axes[1][0].stem(fft_freq_8, modulo_8, basefmt='--')
axes[1][1].stem(fft_freq_8, np.rad2deg(fase_8), basefmt='--')
axes[1][0].set_ylabel("|Módulo DFT 8|", size=15)
axes[1][0].set_xlabel("Frequência", size=15)
axes[1][1].set_ylabel("Ângulo DFT 8 [°]", size=15)
axes[1][1].set_xlabel("Frequência", size=15)

plt.tight_layout()
```



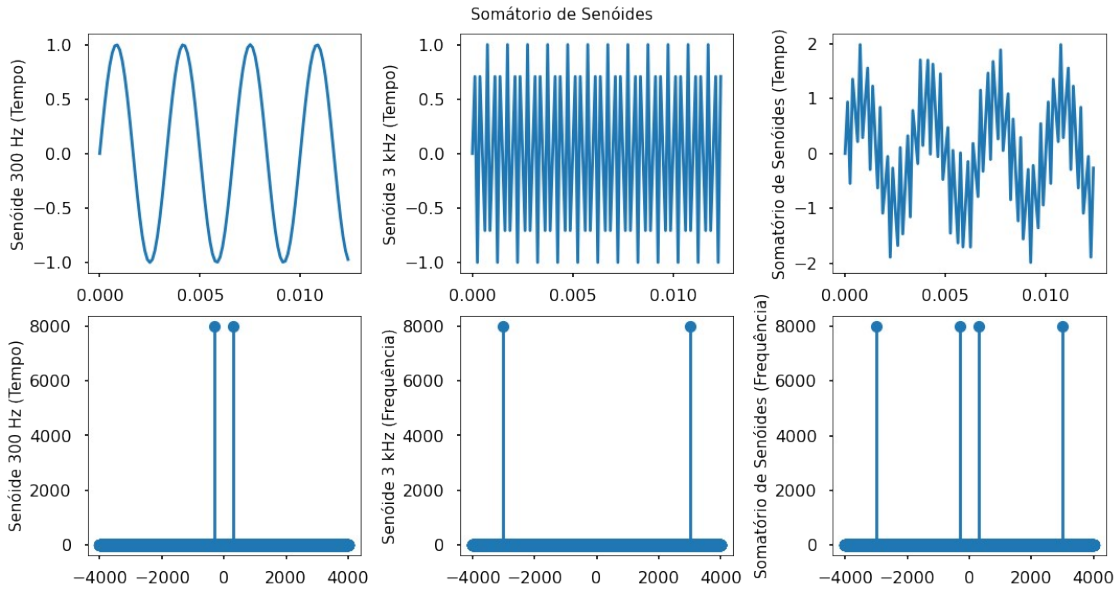
7) Gere um sinal x que seja o somatório de duas senóides de 300 e 3000 Hz, com duração de 2 segundos e frequência de amostragem de 8kHz

```
# Frequência de Amostragem = 8kHz
freq_amostragem = 8000

# Array de amostras (tempo)
tempo = np.arange(0, 2, 1/freq_amostragem)

# Sinal 1 -> Domínio do tempo
y1 = np.sin(2 * np.pi * 300 * tempo)

# Sinal 1 --> Domínio da frequência
y1_dft = np.fft.fft(y1, n=tempo.shape[0])
y1_dft_freq = np.fft.fftfreq(y1_dft.size, d=1/freq_amostragem)
modulo_y1_dft = np.absolute(y1_dft)

# Sinal 2 - > Domínio do tempo
y2 = np.sin(2 * np.pi * 3e3 * tempo)

# Sinal 2 -> Domínio da frequência
y2_dft = np.fft.fft(y2, n=tempo.shape[0])
y2_dft_freq = np.fft.fftfreq(y2_dft.size, d=1/freq_amostragem)
modulo_y2_dft = np.absolute(y2_dft)

# Sinal x -> Somátorio dos sinais 1 e 2 (Domínio do tempo)
x = y1 + y2

# Sinal x -> Domínio da frequência
x_fft = np.fft.fft(x, n=tempo.shape[0])
x_fft_freq = np.fft.fftfreq(x_fft.size, d=1/freq_amostragem)
modulo_x_dft = np.absolute(x_fft)
```

a. Mostre os gráficos do sinal no domínio do tempo, usando a função plot e a frequeência, usando a função stem

```
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(15, 8))


axes[0][0].plot(tempo[0:100], y1[0:100])
axes[0][1].plot(tempo[0:100], y2[0:100])
axes[0][2].plot(tempo[0:100], x[0:100])

axes[1][0].stem(y1_dft_freq, modulo_y1_dft, basefmt='--')
axes[1][1].stem(y2_dft_freq, modulo_y2_dft, basefmt='--')
axes[1][2].stem(x_fft_freq, modulo_x_dft, basefmt='--')

axes[0][0].set_ylabel("Senóide 300 Hz (Tempo)", size=15)
axes[0][1].set_ylabel("Senóide 3 kHz (Tempo)", size=15)
axes[0][2].set_ylabel("Somatório de Senóides (Tempo)", size=15)
```

```
axes[1][0].set_ylabel("Senóide 300 Hz (Tempo)", size=15)
axes[1][1].set_ylabel("Senóide 3 kHz (Frequência)", size=15)
axes[1][2].set_ylabel("Somatório de Senóides (Frequência)", size=15)

fig.suptitle("Somátorio de Senóides", size=15)

fig.tight_layout()
```



Crie duas variáveis H1 = [1, 2;05, 2.05, 1] e H2 = [1, -2.05, 2.05, -1]

```
H1 = np.array([1, 2.05, 2.05, 1])
H2 = np.array([1, -2.05, 2.05, -1])
```

b. Gere o sinal y1 que seja a convolução de x com H1

```
# Convolução de x com H1
y1 = np.convolve(x, H1)

# DFT de Y1
y1_dft = np.fft.fft(y1, n=y1.shape[0])
y1_dft_freq = np.fft.fftfreq(y1_dft.size, d=1/freq_amostragem)

# Módulo de Y1 DFT
modulo_y1_dft = np.absolute(y1_dft)
```

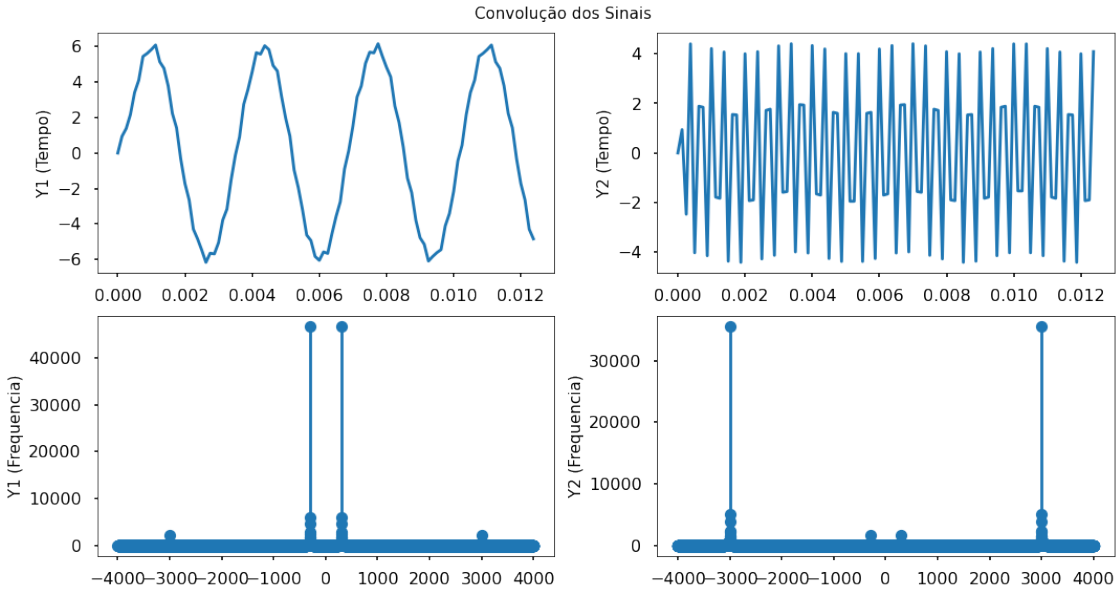
c. Gere o sinal y2 que seja a convolução de x com H2

```
# Convolução de x com H2
y2 = np.convolve(x, H2)

# DFT de Y2
y2_dft = np.fft.fft(y2, n=y2.shape[0])
y2_dft_freq = np.fft.fftfreq(y2_dft.size, d=1/freq_amostragem)
```

```
# Módulo de Y2 DFT
modulo_y2_dft = np.absolute(y2_dft)
```

d. Mostre os gráficos de y1 e y2 no domínio do tempo e da frequência.

```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(15, 8))

axes[0][0].plot(tempo[0:100], y1[0:100])
axes[0][1].plot(tempo[0:100], y2[0:100])
axes[0][0].set_ylabel("Y1 (Tempo)", size=15)
axes[0][1].set_ylabel("Y2 (Tempo)", size=15)

axes[1][0].stem(y1_dft_freq, modulo_y1_dft, basefmt='--')
axes[1][1].stem(y2_dft_freq, modulo_y2_dft, basefmt='--')
axes[1][0].set_ylabel("Y1 (Frequencia)", size=15)
axes[1][1].set_ylabel("Y2 (Frequencia)", size=15)

fig.suptitle("Convolução dos Sinais", size=15)

fig.tight_layout()
```



e. Comente os resultados

R:

- Na letra A, somamos dois gráficos no tempo e também somamos dois gráficos na frequência. Quando somamos gráficos no dominio do tempo, a somatória é feita ponto a ponto. Exemplo: `x[3] + y[3] = z[3]`. Quando somamos gráficos no dominio da frequencia, temos os picos 'adicionados' ao sinal somado final. Isso foi demonstrado na atividade A.
- Na letra b, convoluimos o sinal X com o vetor H1 e o resultado é uma senoide perfeita.

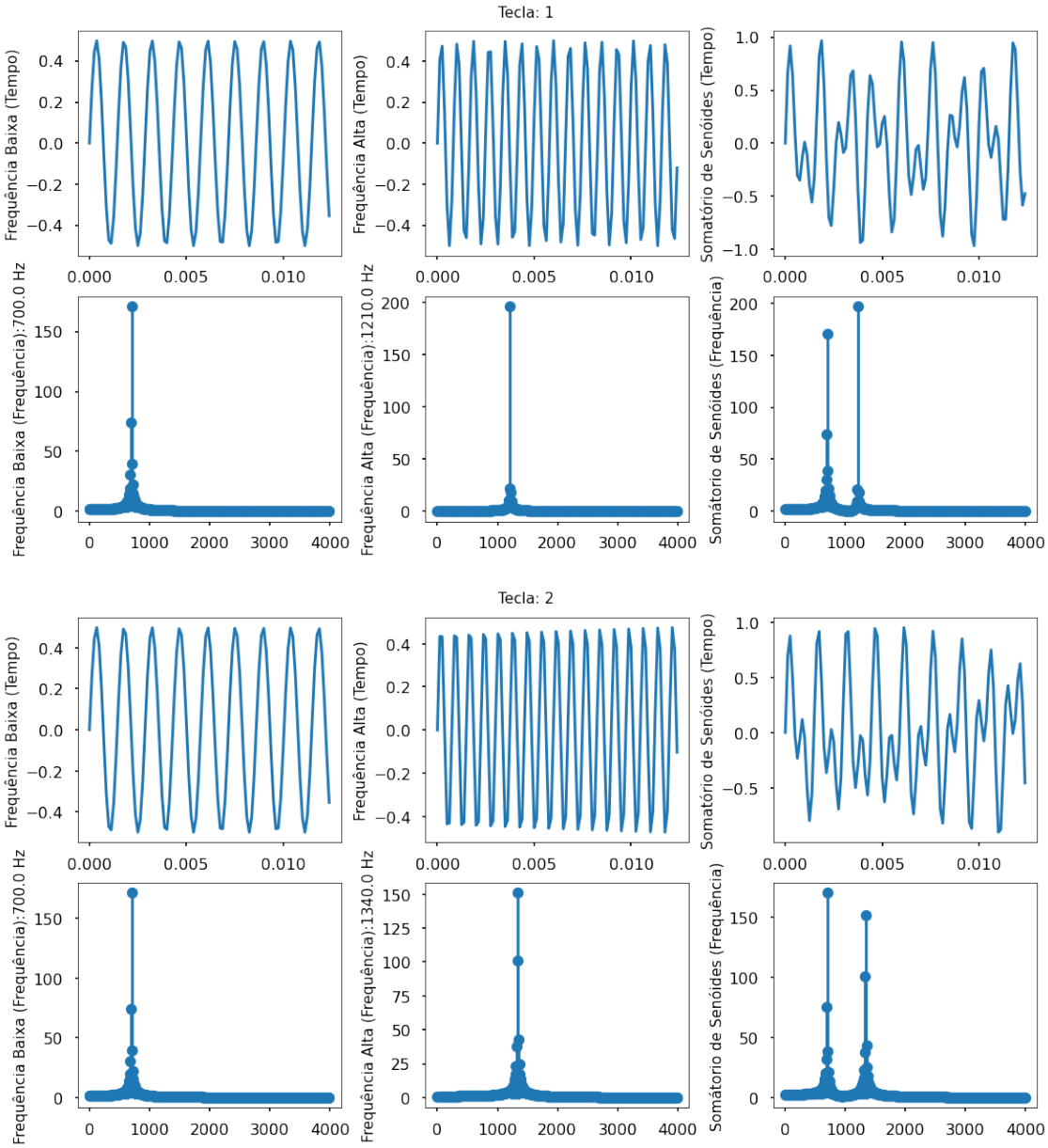
- Na letra c, convoluimos o sinal X com o vetor H2 e, devido a troca de sinal das posições 1 e 3 do vetor de H1 para H2, o sinal tem distorção.

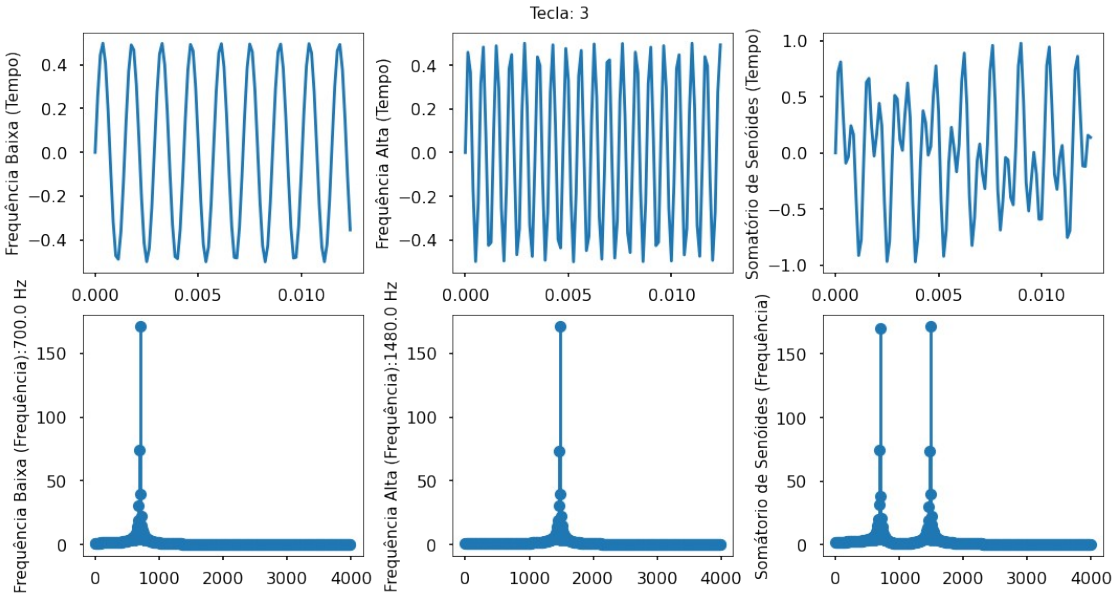
8) O padrão DTMF (Dial Tone Multi Frequency) é amplamente utilizado em sistemas de telefonia. O padrão industrial de especificação de frequências para todas as teclas segue o diagrama da Figura 1:

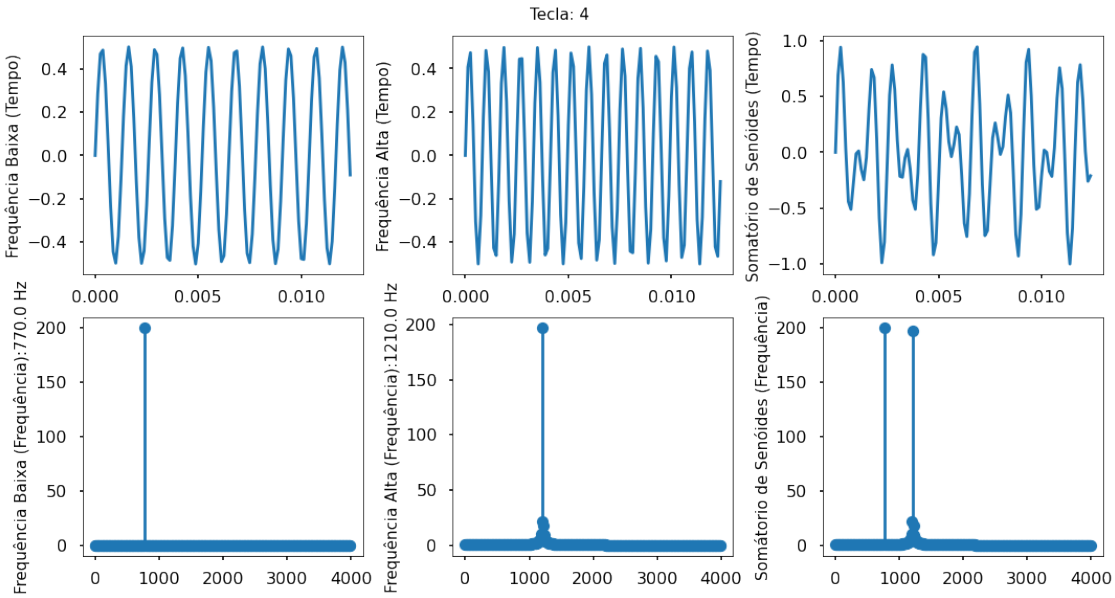
```
def matriz_de_frequencia (frequencias_altas: list, frequencias_baixas:
list) -> list:
    frequency_list = list()
    for _, frequencia_baixa in enumerate(frequencias_baixas):
        for _, frequencia_alta in enumerate(frequencias_altas):
            append_values = [frequencia_baixa, frequencia_alta]
            frequency_list.append(append_values)
    return frequency_list

def graph_plot(
    senoide_baixa: np.ndarray,
    senoide_alta: np.ndarray,
    audio: np.ndarray,
    tempo: np.ndarray,
    index: int) -> None:

    string_teclas = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '*',
'0', '#']

    # Transformadas de Fourier
    fft_baixa = np.absolute(np.fft.fft(senoide_baixa))
    fft_freq_baixa = np.fft.fftfreq(fft_baixa.size, d=1/8000)

    fft_alta = np.absolute(np.fft.fft(senoide_alta))
    fft_freq_alta = np.fft.fftfreq(fft_alta.size, d=1/8000)

    fft_audio = np.absolute(np.fft.fft(audio))
    fft_freq_audio = np.fft.fftfreq(fft_audio.size, d=1/8000)
    fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(15, 8))

    N = int(fft_freq_audio.size / 2)
    max_alta_index = np.where(fft_alta[:N] == np.amax(fft_alta[:N]))
[0][0]
    max_baixa_index = np.where(fft_baixa[:N] ==
np.amax(fft_baixa[:N]))[0][0]

    # Gráficos
    axes[0][0].plot(tempo[0:100], senoide_baixa[0:100])
    axes[0][0].set_ylabel("Frequência Baixa (Tempo)", size=15)
    axes[0][1].plot(tempo[0:100], senoide_alta[0:100])
    axes[0][1].set_ylabel("Frequência Alta (Tempo)", size=15)
    axes[0][2].plot(tempo[0:100], audio[0:100])
```

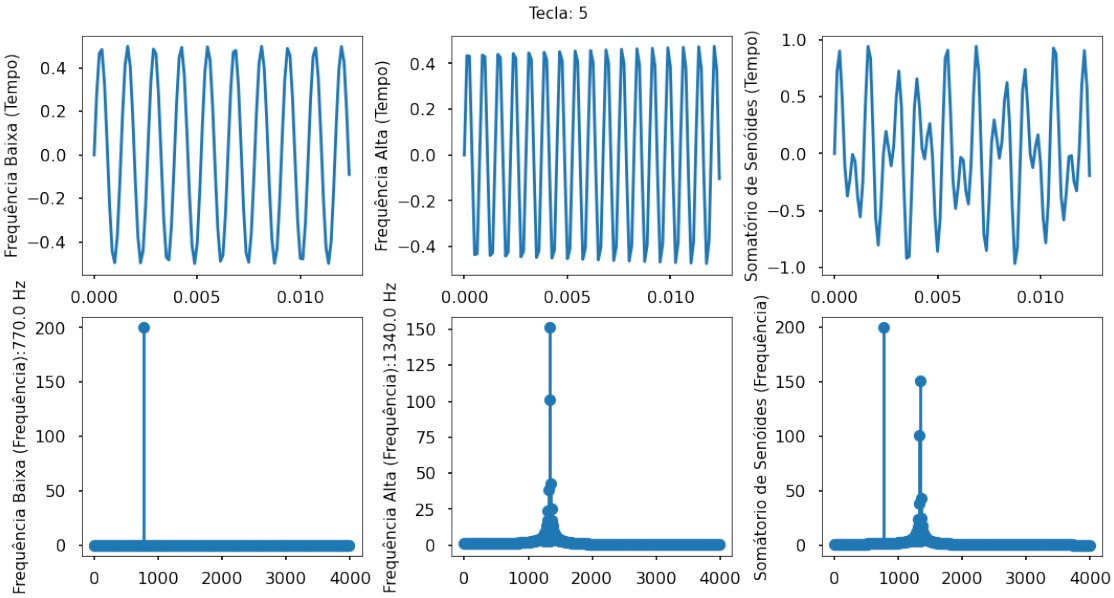
```
    axes[0][2].set_ylabel("Somatório de Senóides (Tempo)", size=15)

    axes[1][0].stem(fft_freq_baixa[:N], fft_baixa[:N], basefmt='--')
    axes[1][0].set_ylabel("Frequência Baixa (Frequência):{}
Hz".format(fft_freq_baixa[max_baixa_index]), size=15)
    axes[1][1].stem(fft_freq_alta[:N], fft_alta[:N], basefmt='--')
    axes[1][1].set_ylabel("Frequência Alta (Frequência):{}
Hz".format(fft_freq_alta[max_alta_index]), size=15)
    axes[1][2].stem(fft_freq_audio[:N], fft_audio[:N], basefmt='--')
    axes[1][2].set_ylabel("Somátorio de Senóides (Frequência)",
size=15)

    title = 'Tecla: ' + string_teclas[index]
    fig.suptitle(title, size=15)
    fig.tight_layout()
    pass

def to_signal (frequencies: list, tempo: np.ndarray) -> list:
    audio_list = [0] * 12
    for index, frequency in enumerate(frequencies):
        senoide_baixa = 0.5 * np.sin(2 * np.pi * (frequency[0]) *
tempo)
        senoide_alta = 0.5  * np.sin(2 * np.pi * frequency[1] * tempo)
        audio = senoide_baixa + senoide_alta
        graph_plot(senoide_baixa, senoide_alta, audio, tempo, index)
        audio_list[index] = audio
    return audio_list


def play_audio(audio_list, sample_rate: float):
    audio_complete = list()
    for _, audio_sound in enumerate(audio_list):
        for index, _ in enumerate(audio_sound):
            audio_complete.append(audio_sound[index])
    wavfile.write('audio_completed.wav', rate=int(sample_rate),
data=np.array(audio_complete).astype(np.float32))
    return audio_complete

freq_altas = [1209, 1336, 1477]
freq_baixas = [697, 770, 852, 941]

freq_amostragem = 8e3

frequencies =  matriz_de_frequencia(freq_altas, freq_baixas)
tempo = np.arange(0, 0.1, 1/freq_amostragem)
audio_list = to_signal(frequencies, tempo)

audio = play_audio(audio_list, freq_amostragem)
IPython.display.Audio("audio_completed.wav")
```

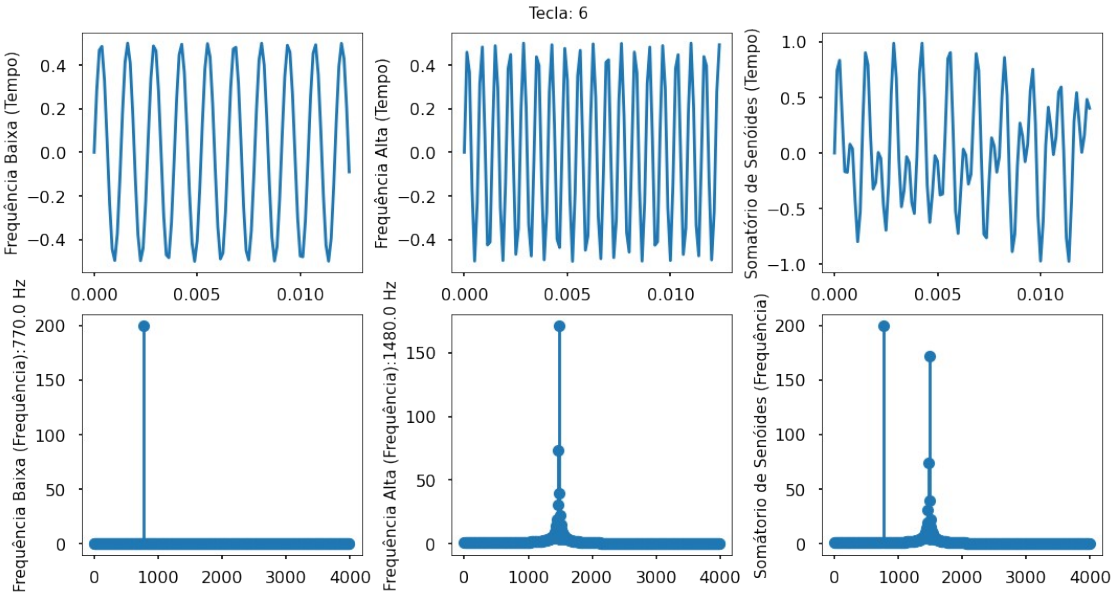
<IPython.lib.display.Audio object>

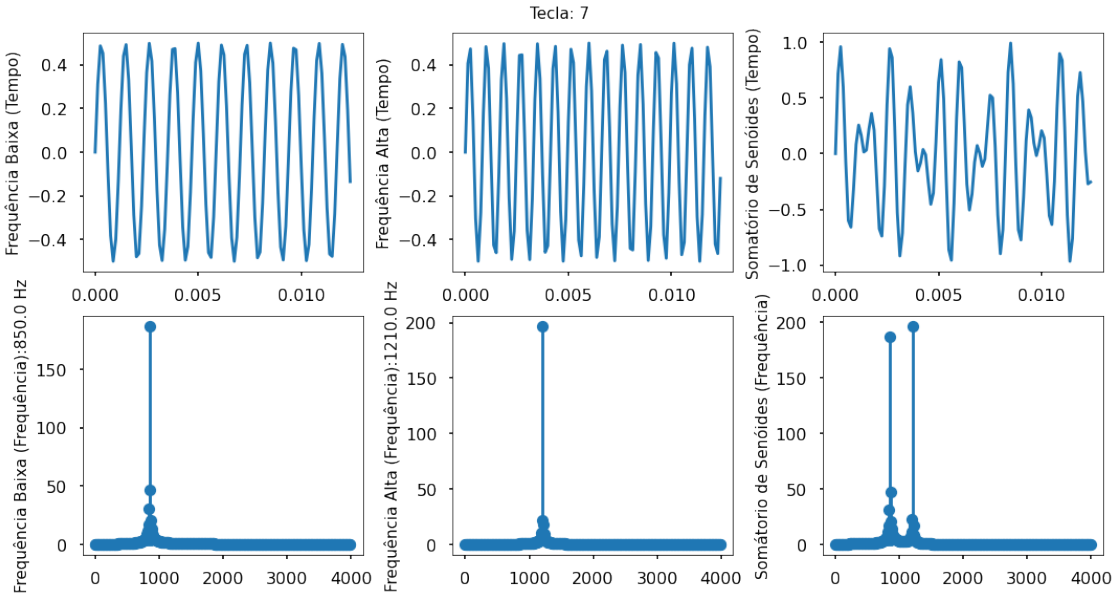


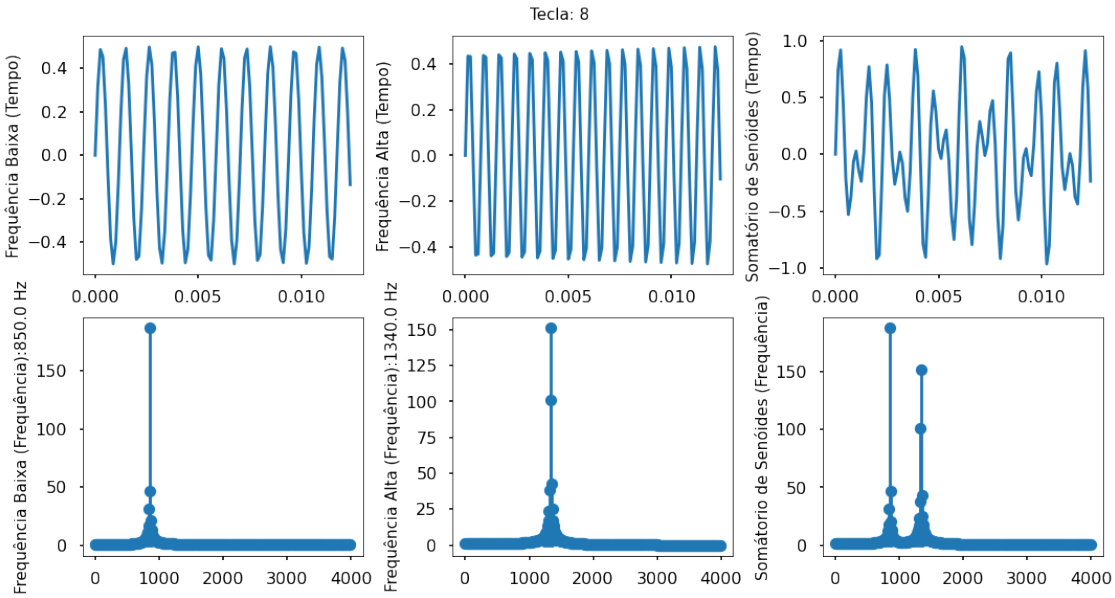

Tecla: 3
Frequência Baixa (Tempo)
Frequência Alta (Tempo)
Somatório de Senóides (Tempo)
Frequência Baixa (Frequência):700.0 Hz
Frequência Alta (Frequência):1480.0 Hz
Somatorio de Senóides (Frequência)

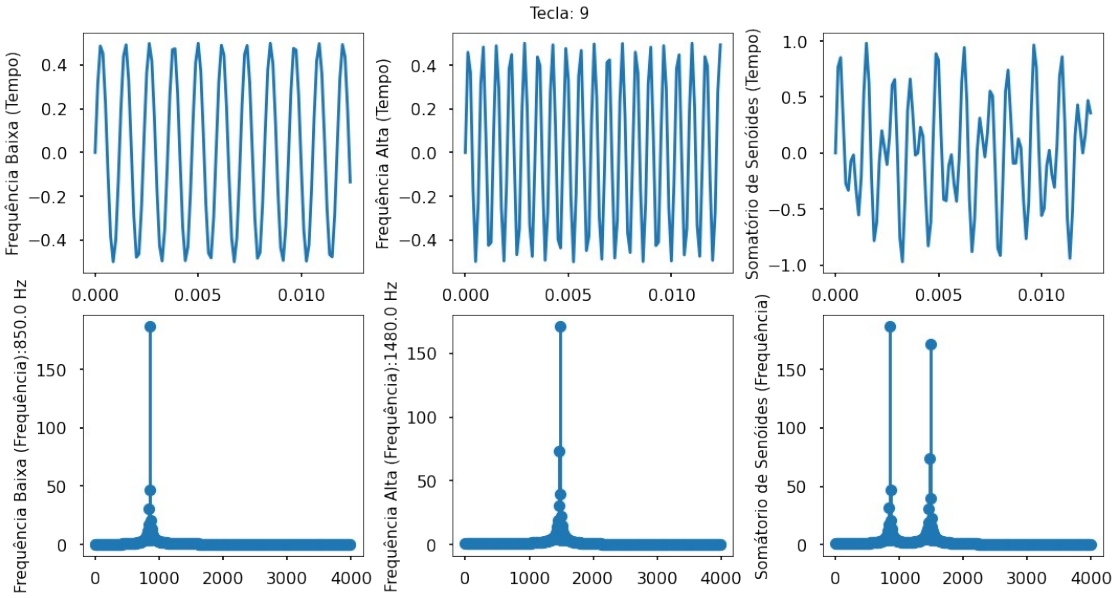

Tecla: 4
Frequência Baixa (Tempo)
Frequência Alta (Tempo)
Somatório de Senóides (Tempo)
Frequência Baixa (Frequência):770.0 Hz
Frequência Alta (Frequência):1210.0 Hz
Somatorio de Senóides (Frequência)

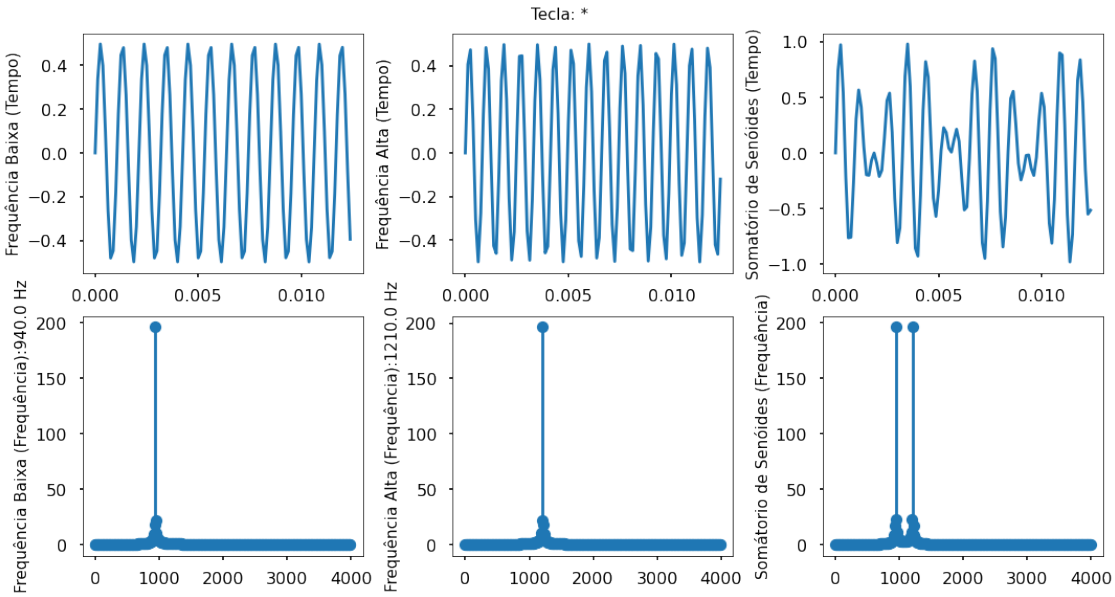

Tecla: 5
Frequência Baixa (Tempo)
Frequência Alta (Tempo)
Somatório de Senóides (Tempo)
Frequência Baixa (Frequência):770.0 Hz
Frequência Alta (Frequência):1340.0 Hz
Somatório de Senóides (Frequência)

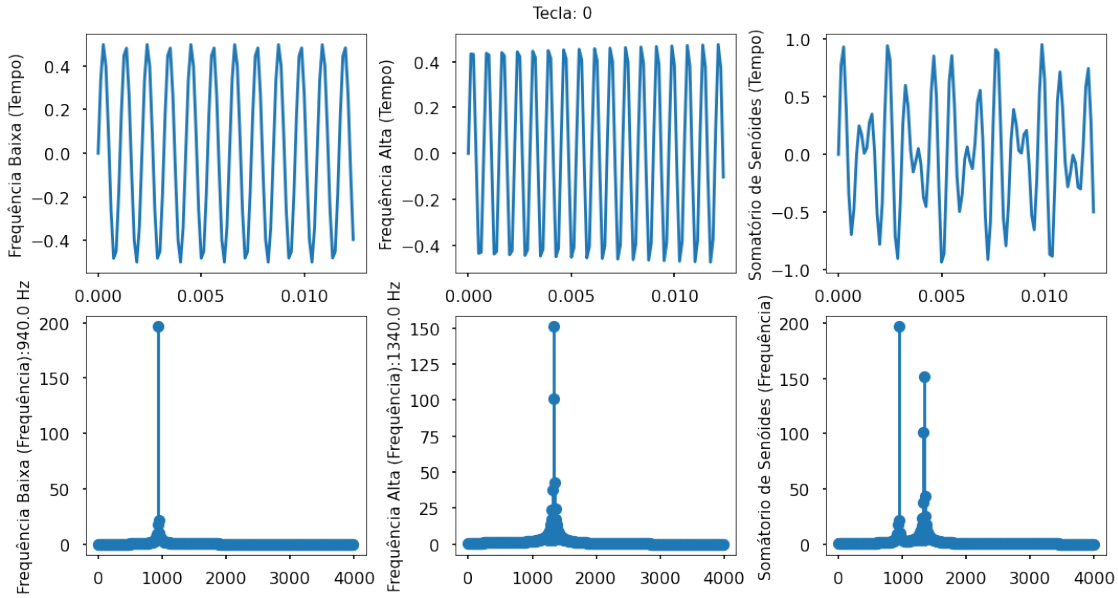

Tecla: 6
Frequência Baixa (Tempo)
Frequência Alta (Tempo)
Somatório de Senóides (Tempo)
Frequência Baixa (Frequência):770.0 Hz
Frequência Alta (Frequência):1480.0 Hz
Somatório de Senóides (Frequência)

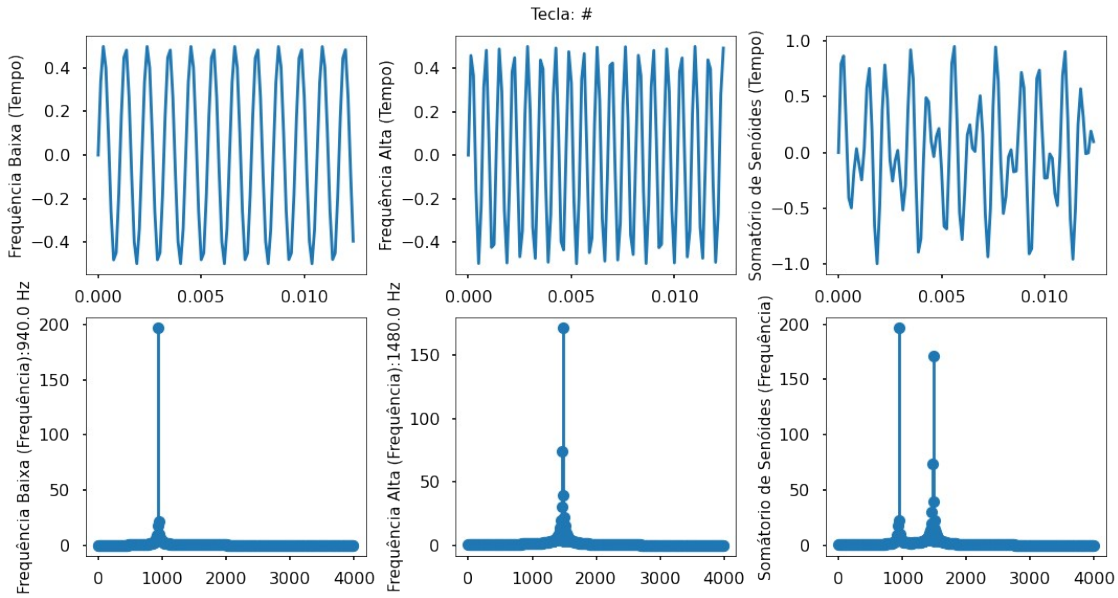

Tecla: 7
Frequência Baixa (Tempo)
Frequência Alta (Tempo)
Somatório de Senóides (Tempo)
Frequência Baixa (Frequência):850.0 Hz
Frequência Alta (Frequência):1210.0 Hz
Somatório de Senóides (Frequência)


Tecla: 8
Frequência Baixa (Tempo)
Frequência Alta (Tempo)
Somatório de Senóides (Tempo)
Frequência Baixa (Frequência):850.0 Hz
Frequência Alta (Frequência):1340.0 Hz
Somatório de Senóides (Frequência)


Tecla: 9
Frequência Baixa (Tempo)
Frequência Alta (Tempo)
Somatório de Senóides (Tempo)
Frequência Baixa (Frequência):850.0 Hz
Frequência Alta (Frequência):1480.0 Hz
Somatorio de Senóides (Frequência)


Tecla: *
Frequência Baixa (Tempo)
Frequência Alta (Tempo)
Somatório de Senóides (Tempo)
Frequência Baixa (Frequência):940.0 Hz
Frequência Alta (Frequência):1210.0 Hz
Somatorio de Senóides (Frequência)





9) Crie uma rotina no Octave/Matlab para abrir o arquivo de áudio DTMF1.wav. Analise o sinal e, através do padrão de frequências DTMF:

a. Identifique a sequência de teclas

```
frequencia_amostragem, audio = wavfile.read('DTMF1.wav')

precision = 0.04
duration = audio.size / frequencia_amostragem
step = int(audio.size // (duration // precision))

c=''

dtmf = {
```

```
    (697, 1209): "1",
    (697, 1336): "2",
    (697, 1477): "3",
    (770, 1209): "4",
    (770, 1336): "5",
    (770, 1477): "6",
    (852, 1209): "7",
    (852, 1336): "8",
    (852, 1477): "9",
    (941, 1209): "*",
    (941, 1336): "0",
    (941, 1477): "#",
    (697, 1633): "A",
    (770, 1633): "B",
    (852, 1633): "C",
    (941, 1633): "D"
}
print("Número interpretado ")
for index in range (0, audio.size - step, step):
    signal = audio[index:index+step]
    amplitudes = np.fft.fft(signal)
    frequencies = np.fft.fftfreq(signal.size,
d=1/frequencia_amostragem)

    # ----------------Low Frequency----------------------------
    i_min = np.where(frequencies > 0)[0][0]
    i_max = np.where(frequencies > 1050)[0][0]

    freq = frequencies[i_min:i_max]
    amp = abs(amplitudes.real[i_min:i_max])

    lf = freq[np.where(amp == max(amp))[0][0]]

    delta = 20
    best = 0

    for f in [697, 770, 852, 941]:
        if abs(lf - f) < delta:
            delta = abs(lf - f)
            best = f
    lf = best
    # ---------------------------------------------------------

    # ----------------High Frequency----------------------------
    i_min = np.where(frequencies > 1100)[0][0]
    i_max = np.where(frequencies > 2000)[0][0]

    freq = frequencies[i_min:i_max]
    amp = abs(amplitudes.real[i_min:i_max])
```

```
    hf = freq[np.where(amp == max(amp))[0][0]]

    delta = 20
    best = 0

    for f in [1209, 1336, 1477, 1633]:
        if abs(hf - f) < delta:
            delta = abs(hf - f)
            best = f
    hf = best

    if lf == 0 or hf == 0:
        c=''

    elif dtmf[(lf, hf)] != c:
        c = dtmf[(lf, hf)]
        print(c, end='', flush=True)
```

```
Número interpretado
08112014
```

```
H1 = np.array([0.00253628299721552, 0.00255348555988337, 0, -
0.00863465184119861,
-0.0210222026271213, -0.0242112092355797, 0, 0.0605565098224690,
0.144591562515657, 0.219130722316847, 0.248999000983656,
0.219130722316847,
0.144591562515657, 0.0605565098224690, 0, -0.0242112092355797,
-0.0210222026271213, -0.00863465184119861, 0, 0.00255348555988337,
0.00253628299721552])

H1_dft = np.fft.fft(H1)
H1_dft_freq = np.fft.fftfreq(H1_dft.size, d=1/frequencia_amostragem)


H2 = np.array([-0.00255268288602145, -0.00256999668237861, 0,
0.00869048446327945,
0.0211581345345303, 0.0243677616154823, 0, -0.0609480749706223,
-0.145526507689415, -0.220547646013415, 0.751827168906504,
-0.220547646013415, -0.145526507689415, -0.0609480749706223, 0,
0.0243677616154823, 0.0211581345345303, 0.00869048446327945, 0,
-0.00256999668237861, -0.00255268288602145])

H2_dft = np.fft.fft(H2)
H2_dft_freq = np.fft.fftfreq(H2_dft.size, d=1/frequencia_amostragem)

x = audio / 75e3
x_dft = np.fft.fft(x)
x_dft_freq = np.fft.fftfreq(x_dft.size, d=1/frequencia_amostragem)
```

b. Gere o sinal y1 que seja a convolução de x com H1

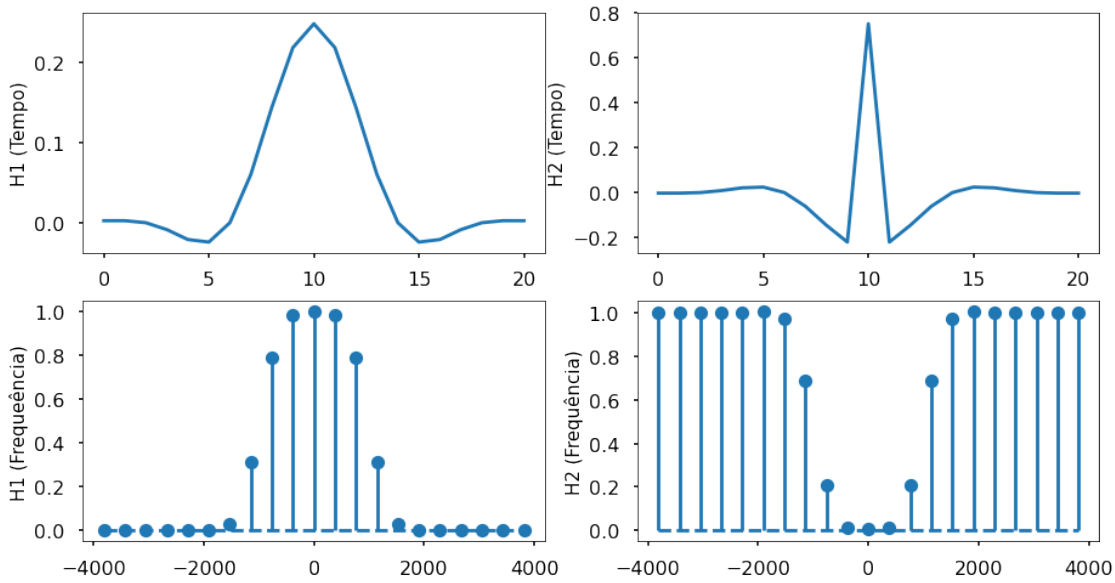
```
# Sinal y1 - Convolução x com H1
y1 = np.convolve(x, H1)

# DFT Sinal Y1
y1_dft = np.fft.fft(y1)
y1_dft_freq = np.fft.fftfreq(y1_dft.size, d=1/frequencia_amostragem)
```

c. Gere o sinal y2 que seja a convolução de x H2

```
# Sinal y2 - Convolução x com H2
y2 = np.convolve(x, H2)

# DFT Sinal Y2
y2_dtf = np.fft.fft(y2)
y2_dft_freq = np.fft.fftfreq(y2_dft.size, d=1/frequencia_amostragem)
```

d. Mostre os gráficos de H1 e H2 nos domínios do tempo e da frequência

```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(15, 8))

axes[0][0].plot(H1)
axes[0][0].set_ylabel("H1 (Tempo)", size=15)
axes[0][1].plot(H2)
axes[0][1].set_ylabel("H2 (Tempo)", size=15)

axes[1][0].stem(H1_dft_freq, np.absolute(H1_dft), basefmt='--')
axes[1][0].set_ylabel("H1 (Frequeência)", size=15)
axes[1][1].stem(H2_dft_freq, np.absolute(H2_dft), basefmt='--')
axes[1][1].set_ylabel("H2 (Frequência)", size=15)

Text(0, 0.5, 'H2 (Frequência)')
```



e. Comente os Resultados

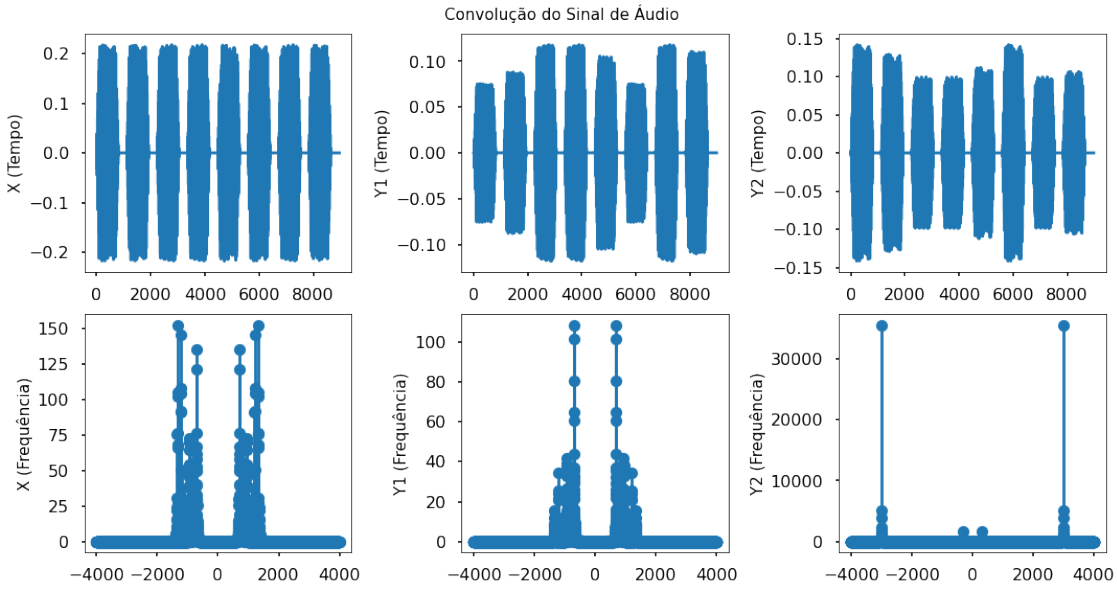
R:

Os seguintes passos foram tomados:

1. Incialmente separou-se cada parte do sinal que corresponde aos números discados
2. Aplicou-se a FFT
3. Os picos na frequencia foram selecionados e esses picos foram comparados com a tabela padrao.
4. O número correspondente foi gerado

f. Mostre os gráficos de y1 e y2 no domínio da frequência

```
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(15, 8))

axes[0][0].plot(x)
axes[0][0].set_ylabel("X (Tempo)", size=15)
axes[0][1].plot(y1)
axes[0][1].set_ylabel("Y1 (Tempo)", size=15)
axes[0][2].plot(y2)
axes[0][2].set_ylabel("Y2 (Tempo)", size=15)

axes[1][0].stem(x_dft_freq, np.absolute(x_dft), basefmt='--')
axes[1][0].set_ylabel('X (Frequência)', size=15)
axes[1][1].stem(y1_dft_freq, np.absolute(y1_dft), basefmt='--')
axes[1][1].set_ylabel('Y1 (Frequência)', size=15)
axes[1][2].stem(y2_dft_freq, np.absolute(y2_dft), basefmt='--')
axes[1][2].set_ylabel('Y2 (Frequência)', size=15)

fig.suptitle("Convolução do Sinal de Áudio", size=15)
fig.tight_layout()
```



g. Comente os resultados da convolução de x com H1 (y1) e de x com H2 (y2)

R:

- Olhando para os gráficos Y1 e Y2 temos:
  - Y1 é o resultado aplicando um filtro de frequencias altas.
  - Y2 é o resultado aplicando um filtro de frequencias baixas.

10) Construa uma rotina que calcule a DFT de um sinal utilizando a definição. Gere um sinal senoidal com amplitude 1 e frequência igual a soma dos números do seu GRR e sequência de amostragem de 1000Hz. Apresente graficamente (utilizando a função STEM) as frequências componentes deste sinal, resultado da DFT.

*É proibido utilizar a função pronta FFT para computar a resposta do exercício, mas pode se utilizar a mesma para validar esta resposta

```
def DFT(x):
    N = len(x)
    n = np.arange(N)
    k = n.reshape((N, 1))
    e = np.exp(-2j * np.pi * k * n / N)
    X = np.dot(e, x)
    return X

amplitude = 1
arthur_grr = 2 + 0 + 1 + 7 + 7 + 2 + 4 + 3
vinicius_grr = 2 + 0 + 1 + 7 + 2 + 1 + 3 + 7

# Soma dos GRR
grr_sum = arthur_grr + vinicius_grr

# Frequência de amostragem
freq_amostragem = 1e3

# Vetor de tempo
t  = np.linspace(0, 250, 100)

# Sinal
sinal = amplitude * np.sin(2 * np.pi * grr_sum * t)

# Utilizar DFT (Função criada)
sinal_dft_criada = DFT(sinal)

# DFT - Numpy (Biblioteca)
sinal_dft_biblioteca = np.fft.fft(sinal)

# Gráfico de Comparação
```

```
fig, axes = plt.subplots(nrows=2, figsize=(15, 8))

axes[0].stem(np.fft.fftfreq(sinal_dft_criada.size, d=1/1000),
np.absolute(sinal_dft_criada), basefmt='--')
axes[0].set_ylabel("DFT - USUÁRIO", size=15)
axes[1].stem(np.fft.fftfreq(sinal_dft_biblioteca.size, d=1/1000),
np.absolute(sinal_dft_biblioteca), basefmt='--')
axes[1].set_ylabel("DFT - BIBLIOTECA", size=15)
axes[1].set_xlabel("Frequência", size=15)


fig.suptitle("Comparação de Rotinas DFT", size=15)
fig.tight_layout()
```