Universidade Federal do Paraná
Departamento de Engenharia Elétrica

Pedro Lucca Pereira da Veiga
Adriely Teixeira de Paula

# Processamento Digitais de Sinais II
# Laboratório 1 - Análise em frequência e filtros digitais

Curitiba
2025

# 1 Análise em frequência e filtros digitais

## 1.1 Gere o sinal x de acordo com o código abaixo:

A = 3;
d = $-(1/12)+(pi/6)*i$;
n = $0:49$;
x = $A*exp(d*n)$;

Código desenvolvido:

```
ampl = 3 #amplitude
dec  = (-1/12)+(np.pi/6)*1j #decaimento
n = np.arange(50) #vetor tempo com 50 instantes
x = ampl*np.exp(dec*n)

fig, ax = plt.subplots(2, 1)
ax[0].stem(x.real)
ax[1].stem(x.imag)

ax[0].set_xlabel('n')
ax[0].set_ylabel('Amplitude-real')

ax[1].set_xlabel('n')
ax[1].set_ylabel('Amplitude-imaginaria')

ax[0].grid()
ax[1].grid()

plt.show()
```

### 1.1.1 Mostre o gráfico da parte real e o gráfico da parte imaginária

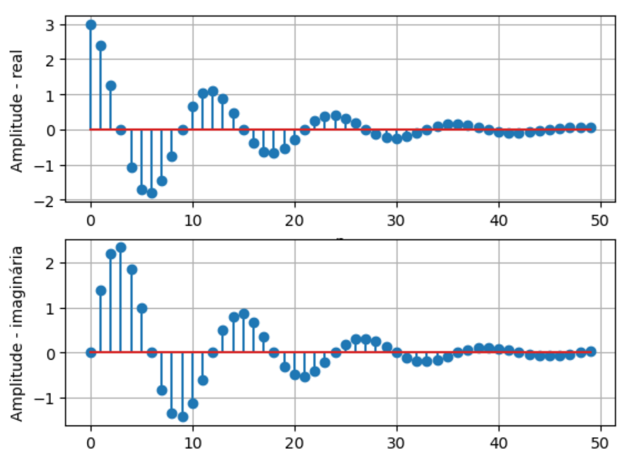


Figure 1: Amplitudes das partes real e imaginária

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

```
dec = -np.conjugate(dec)
x = ampl*np.exp(dec*n)

fig, ax = plt.subplots(2, 1)
ax[0].stem(x.real)
ax[1].stem(x.imag)

ax[0].set_xlabel('n')
ax[0].set_ylabel('Amplitude - real')

ax[1].set_xlabel('n')
ax[1].set_ylabel('Amplitude - imaginaria')

ax[0].grid()
ax[1].grid()

plt.show()
```

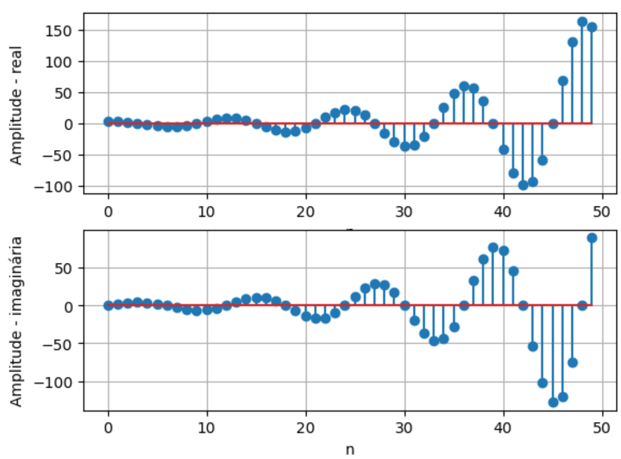


Figure 2: Amplitudes das partes real e imaginária

### 1.1.3 Compare e comente os gráficos gerados

Quando o decaimento $d$ é negativo, por conta do fator $exp(d*n)$ que faz com que o sinal decresça exponencialmente ao longo do tempo, a parte imaginária apresenta uma oscilação senoidal, enquanto a parte real apresenta oscilação cossenoidal, com ambas amplitudes moduladas por um decaimento exponencial.

Porém, quando o termo real de $d$ é positivo, tem-se um crescimento exponencial do sinal. As oscilações senoidal e cossenoidal nas partes imaginária e real permanecem, mas agora a amplitude da onda cresce com o tempo.

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

```
x = 0.5*np.exp(np.arange(30)*0.8)

plt.stem(x)
plt.xlabel('n')
```

```
plt.ylabel('Amplitude')
plt.grid()
plt.show()
```

### 1.2.1 Visualize o sinal gerado

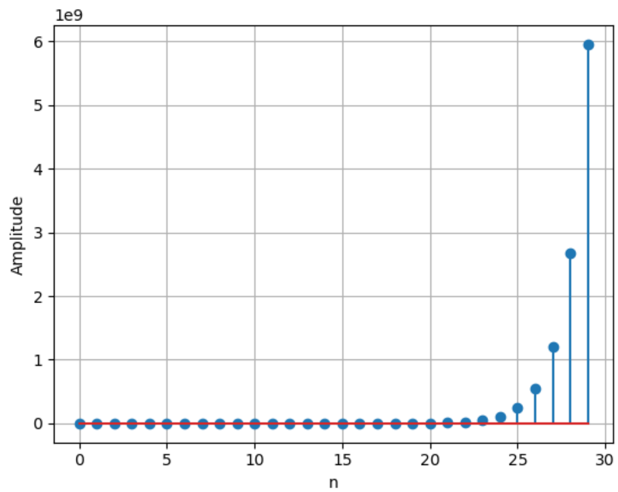


Figure 3: Amplitude do sinal com $d = 0.8$

### 1.2.2 Altere o parâmetro d para 0,2 e visualize o sinal

```
x = 0.5*np.exp(np.arange(30)*0.2)

plt.stem(x)
plt.xlabel('n')
plt.ylabel('Amplitude')
plt.grid()
plt.show()
```

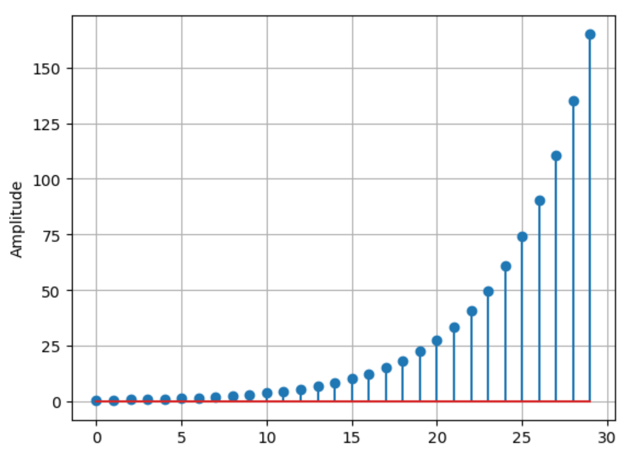


Figure 4: Amplitude do sinal com $d = 0.2$

### 1.2.3 Altere o parâmetro d para -0,2 e visualize o sinal

```
x = x**-1

plt.stem(x)
plt.xlabel('n')
plt.ylabel('Amplitude')
plt.grid()
plt.show()
```

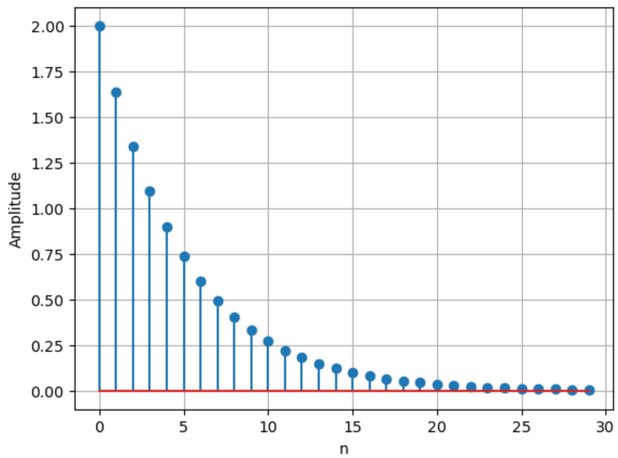


Figure 5: Amplitude do sinal com $d = -0.2$

### 1.2.4 Compare e comente os gráficos gerados

A função exponencial apresenta um crescimento maior de acordo com o valor do expoente. Ou seja, para maiores valores de (neste caso) $d$, a função irá apresentar um resultado maior no mesmo instante $n$. É possível interpretar este comportamento através das propriedades de sinais, como a compressão/dilatação e reversão no tempo, como ocorre no item c (1.2.3).

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

### 1.3.1 Visualize o sinal gerado com as funções stem, plot e stairs

```
x = 3*np.cos(2*np.pi*0.08*np.arange(50))

fig, ax = plt.subplots(3, 1)
ax[0].plot(x)
ax[1].stem(x)
ax[2].stairs(x)

for i in range(3):
    ax[i].set_xlabel('n')
    ax[i].set_ylabel('Amplitude')
    ax[i].grid()

plt.show()
```

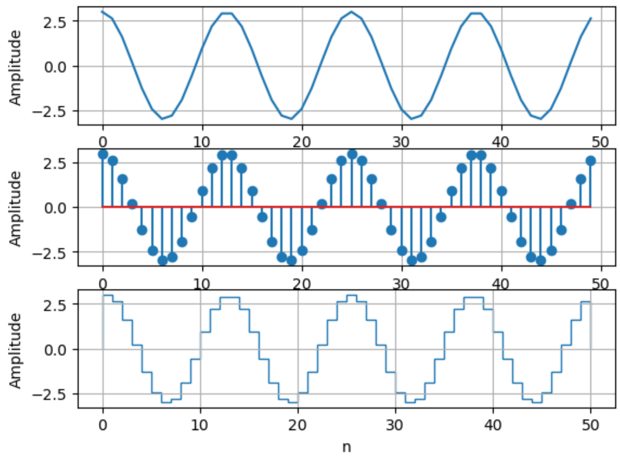


Figure 6: Diferentes formas de visualização de sinais

#### 1.3.2 Modifique o tamanho da sequência para 80, a frequência para 0,5 e a amplitude para 1,5

```
x = 1.5*np.cos(2*np.pi*0.5*np.arange(80))

fig, ax = plt.subplots(3, 1)
ax[0].plot(x)
ax[1].stem(x)
ax[2].stairs(x)

for i in range(3):
    ax[i].set_xlabel('n')
    ax[i].set_ylabel('Amplitude')
    ax[i].grid()

plt.show()
```

### 1.3.3 Visualize o sinal modificado com as funções stem, plot e stairs

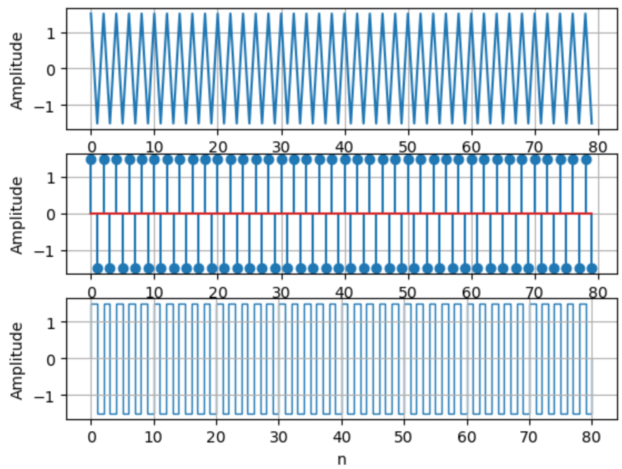


Figure 7: Diferentes formas de visualizar o novo sinal

### 1.3.4 Verifique e comente as diferenças entre as três funções de visualização

A função *stem* apresenta os sinais na visualização clássica de sinais discretos, com barras nos instantes de tempo. Ao utilizar a função *plot*, a visualização liga cada amostra do sinal com uma reta, sendo assim, uma espécie de interpolação linear na visualização (que pode ou não ser desejável para a análise do sinal).

A função *stairs* aplica uma espécie de *zero-order hold*, que mantém o valor da amostra nos instantes "vazios", facilitando a identificação dos valores de cada amostra. Contudo, esta forma de visualização, assim como o *plot*, "extende" o sinal para tempo contínuo (somente na visualização) e, por isso, pode ou não representar fielmente o conteúdo dos dados.

## 1.4 Gere o sinal x = 2*t* 0.9 ** t com t = 0:49

```
x = 2*np.arange(50)*0.9**np.arange(50)
```

### 1.4.1 Adicione ruído gaussiano no sinal (função randn)

```
x_ruido = x + np.random.randn(50)
```

### 1.4.2 Aplique o filtro de média móvel de 3 pontos no sinal ruidoso

```
x_filtrado = np.convolve(x_ruido, np.ones(3)/3, 'same')
```

### 1.4.3 Visualize o sinal original, o sinal ruidoso e o sinal filtrado

```
plt.plot(x, ':', label = 'sinal original')
plt.plot(x_ruido, label = 'sinal ruidoso')
plt.plot(x_filtrado, label = 'sinal filtrado')

plt.legend()
plt.grid()
plt.show()
```

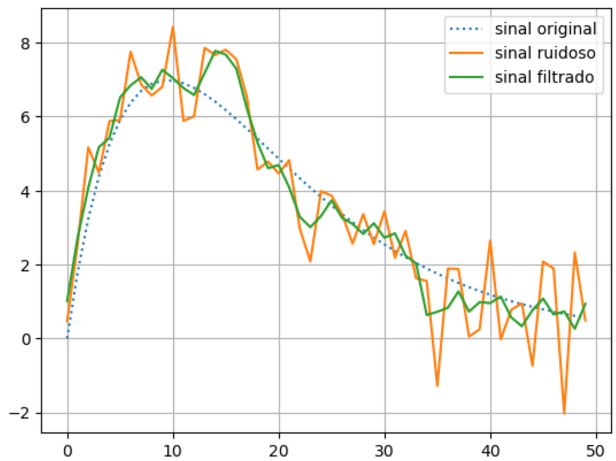


Figure 8: Sinais original, com ruído e filtrado

#### 1.4.4 O filtro aplicado é causal? Justifique sua resposta

Sim, porque não depende de valores futuros do sinal de entrada.

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

```
x = np.zeros(60); x[10:50] = 1
```

#### 1.5.1 Gere um sinal y que seja a convolução de x com ele mesmo

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

#### 1.5.2 Gere um sinal z que seja a convolução de x com y

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

#### 1.5.3 Visualize x, y e z

```
y = y/np.max(y)    #normalizacao para plotagem
z = z/np.max(z)    #normalizacao para plotagem

plt.plot(np.arange(60), x, label = 'x')
plt.plot(np.arange(-29, 90), y, label = 'y')
plt.plot(np.arange(-58, 120), z, label = 'z')

plt.xlabel('n')

plt.legend()
plt.grid()
plt.show()
```

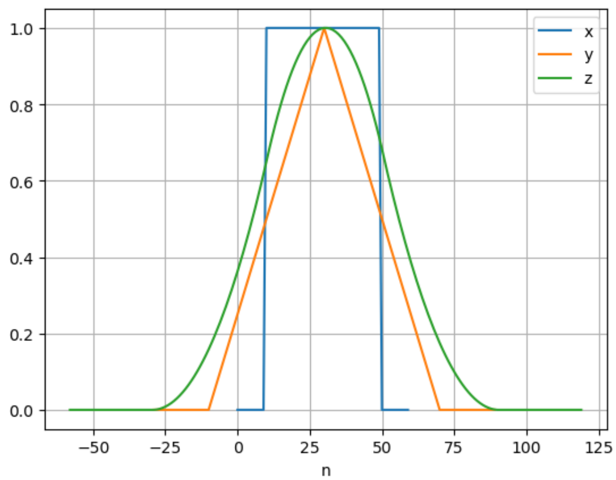


Figure 9: Janela retangular e convoluções consigo mesma

### 1.5.4 Comente os resultados de a e b (1.5.1 e 1.5.2)

O sinal y possui a forma de um pulso triangular. A transformada desse pulso triangular é igual a uma função do tipo $sinc^2(f)$, que confirma o resultado obtido por se tratar da multiplicação (decorrente da convolução no tempo) da transformada de um pulso retangular (função do tipo $sinc(f)$) consigo mesma.

Já o sinal z apresenta a forma parecida com um função gaussiana, resultado esperado, de certa forma, por conta do teorema do limite central, já que é o resultado da convolução de três pulsos retangulares (deve se aproximar da gaussiana com mais convoluções consigo mesmo). A transformada de Fourier deste pulso possui a forma da função $sinc^3(f)$.

# 2 Aplicações da transformada rápida de Fourier

A transformada rápida de Fourier (FFT) $X[k]$ de uma sequência finita $x[n]$ pode ser computada no Octave/Matlab usando 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.

### 2.0.1 Crie uma sequência x = cos(2*pi*0.4*n) de tamanho 32. Compute e visualize a 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.

```
x = np.cos(2*np.pi*0.4*np.arange(32))

fig, ax = plt.subplots(2, 1)
ax[0].stem(abs(np.fft.fft(x)))
ax[1].stem(np.angle(np.fft.fft(x)))

ax[0].set_xlabel('k (frequencia)')
ax[0].set_ylabel('Magnitude')

ax[1].set_xlabel('k (frequencia)')
ax[1].set_ylabel('Fase (rad)')

ax[0].grid()
ax[1].grid()
```

```python

plt.show()

fig, ax = plt.subplots(2, 1)
ax[0].stem(abs(np.fft.fft(x, 8)))
ax[1].stem(np.angle(np.fft.fft(x, 8)))

ax[0].set_xlabel('k (frequencia)')
ax[0].set_ylabel('Magnitude')

ax[1].set_xlabel('k (frequencia)')
ax[1].set_ylabel('Fase (rad)')

ax[0].grid()
ax[1].grid()

plt.show()
```

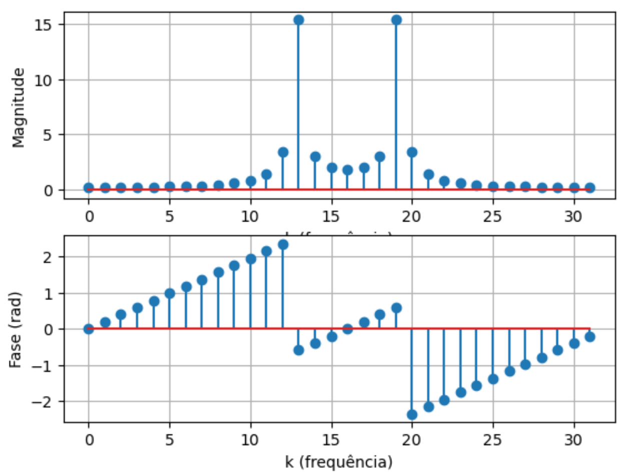


Figure 10: Magnitude e fase da FFT do sinal para $L = 32$ pontos

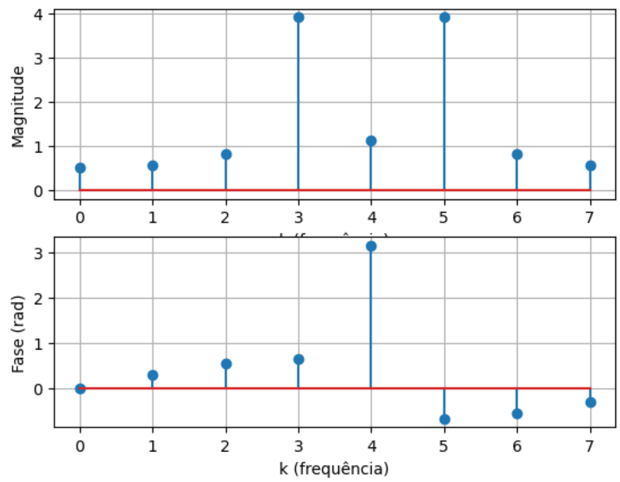


Figure 11: Magnitude e fase da FFT do sinal para $L = 8$ pontos

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

### 2.1.1 Mostre os gráficos do sinal no domínio do tempo, usando a função *plot* e da frequência, usando a função *stem*.

```
fs = 8000
t = np.arange(0,2,1/fs) #vetor de tempo de 2s

f1, f2 = 300, 3000
x = np.sin(2 * np.pi * f1 * t) + np.sin(2 * np.pi * f2 * t)

plt.figure(figsize=(12, 5))
plt.subplot(2, 1, 1)
plt.plot(t[:400], x[:400])  # Mostrar apenas 400 amostras para melhor visualizacao
plt.title("Sinal x no dominio do tempo")
plt.xlabel("Tempo (s)")
plt.ylabel("Amplitude")
plt.grid()

X = np.abs(np.fft.fft(x))
freqs = np.fft.fftfreq(len(x), d=1/fs)

plt.subplot(2, 1, 2)
plt.stem(freqs[:len(freqs)//2], X[:len(X)//2])
plt.title("Sinal x no dominio da frequencia")
plt.xlabel("Frequencia (Hz)")
plt.ylabel("Magnitude")
plt.grid()
plt.tight_layout()
plt.show()
```

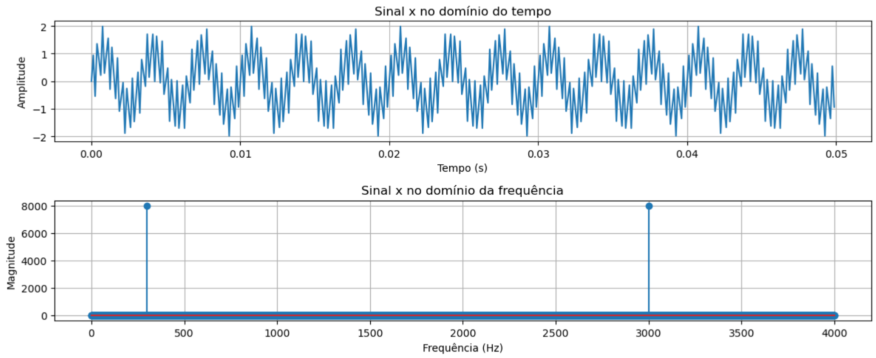


Figure 12: Sinal no tempo e na frequência

#### 2.1.2 Crie duas variáveis $H_1 = [1, 2.05, 2.05, 1]$ e $H_2 = [1, -2.05, 2.05, -1]$. Gere o sinal $y_1$ que seja a convolução de $x$ com $H_1$ e o sinal $y_2$ que seja a convolução de $x$ com $H_2$.

```python

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

y1 = signal.convolve(x, H1, mode='same')
y2 = signal.convolve(x, H2, mode='same')

def plot_signal(titulo, sinal, fs):
    plt.figure(figsize=(12, 5))

    plt.subplot(2, 1, 1)
    plt.plot(t[:400], sinal[:400])  # Exibir apenas as primeiras 400 amostras
    plt.title(f"{titulo} - Dominio do Tempo")
    plt.xlabel("Tempo (s)")
    plt.ylabel("Amplitude")
    plt.grid()

    Y = np.abs(np.fft.fft(sinal))
    freqs = np.fft.fftfreq(len(sinal), d=1/fs)

    plt.subplot(2, 1, 2)
    plt.stem(freqs[:len(freqs)//2], Y[:len(Y)//2])
    plt.title(f"{titulo} - Dominio da Frequencia")
    plt.xlabel("Frequencia (Hz)")
    plt.ylabel("Magnitude")
    plt.grid()

    plt.tight_layout()
    plt.show()

plot_signal("Sinal y1 (x * H1)", y1, fs)
plot_signal("Sinal y2 (x * H2)", y2, fs)
```

### 2.1.3 Mostre os gráficos de $y_1$ e $y_2$ no domínio do tempo e da frequência.

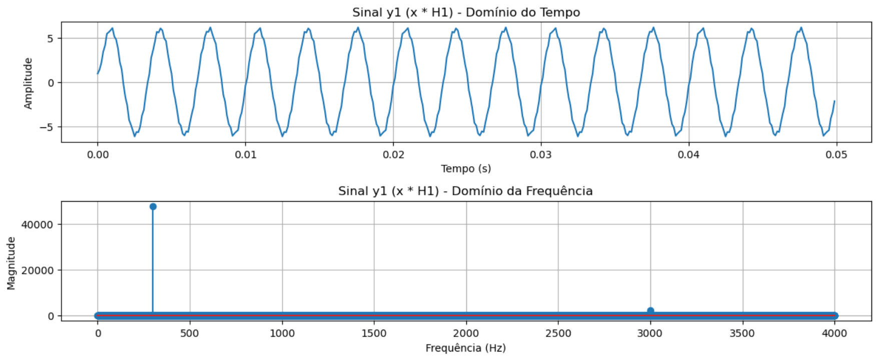


Figure 13: Sinal $y_1$ no tempo e na frequência

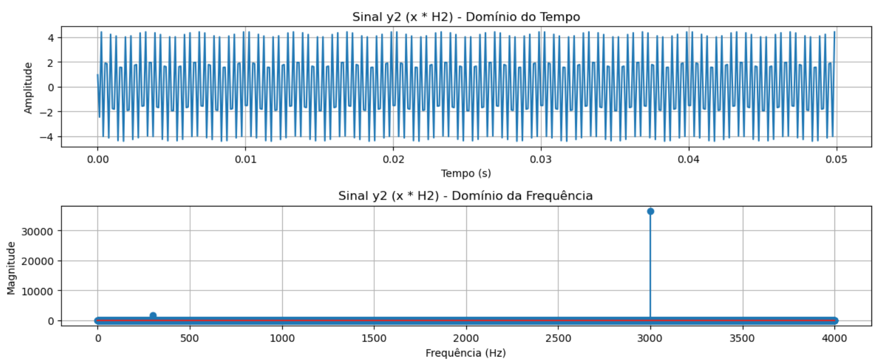


Figure 14: Sinal $y_2$ no tempo e na frequência

### 2.1.4 Comente os resultados

## 2.2 DTMF (*Dial Tone Multi Frequency*)

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 de especificação dos tons DTMF.

### 2.2.1 De acordo com a especificação DTMF, desenvolva uma rotina no Matlab que gere cada tom, plotando-os em um gráfico nos domínios do tempo e da frequência. Posteriormente, reproduza-os utilizando o áudio do PC. Para isto, utilize frequência de amostragem de 8kHz e tons com 100ms de duração.

```
freqs_x = [697, 770, 852, 941]
freqs_y = [1209, 1336, 1477]

fs = 8000 #taxa de amostragem 8kHz
duracao = 0.1
t= np.linspace(0, duracao, int(fs * duracao), endpoint=False)

x_freq = np.asarray([])
y_freq = np.asarray([])

```

```
for i in range(4):
    for j in range(3):
        sinal = np.sin(2 * np.pi * freqs_x[i] * t) + np.sin(2 * np.pi * freqs_y[j] * t)
        plt.subplot(2, 1, 1)
        plt.plot(t[:200], sinal[:200])
        plt.title(f'Sinal DTMF: Tecla {(i*3) + j + 1}')
        plt.xlabel('Tempo (s)')
        plt.ylabel('Amplitude')
        plt.grid()

        Y = np.abs(scipy.fftpack.fft(sinal))[:len(sinal)//2]
        f = np.fft.fftfreq(len(sinal), 1/fs)[:len(sinal)//2]

        plt.subplot(2, 1, 2)
        plt.plot(f, Y)
        plt.title('Espectro de Frequencia')
        plt.xlabel('Frequencia (Hz)')
        plt.ylabel('Magnitude')
        plt.grid()

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

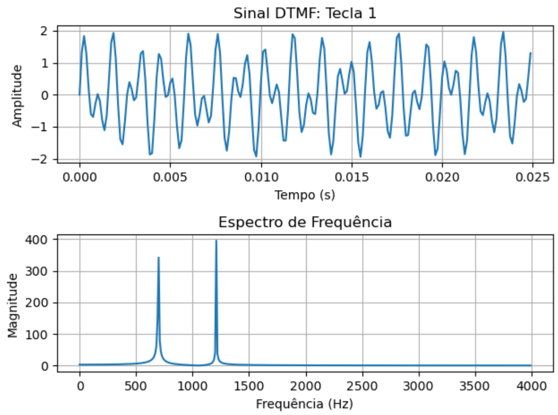


Figure 15: Sinal da tecla 1 no tempo e na frequência

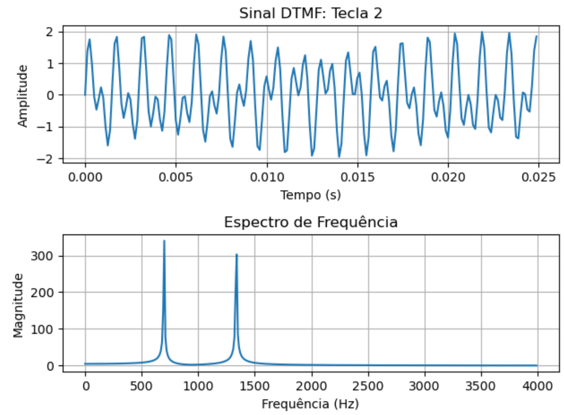


Figure 16: Sinal da tecla 2 no tempo e na frequência

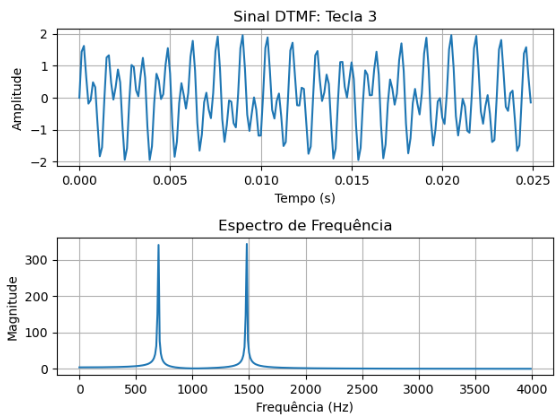


Figure 17: Sinal da tecla 3 no tempo e na frequência

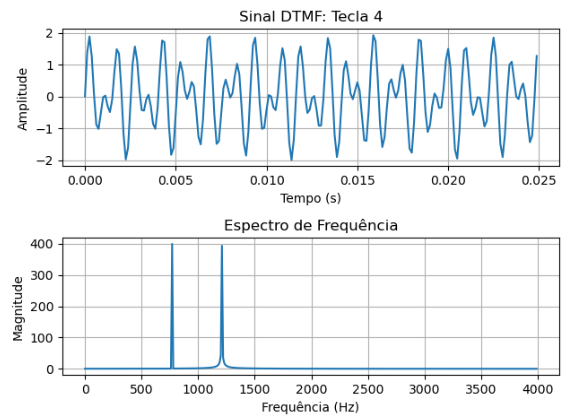


Figure 18: Sinal da tecla 4 no tempo e na frequência

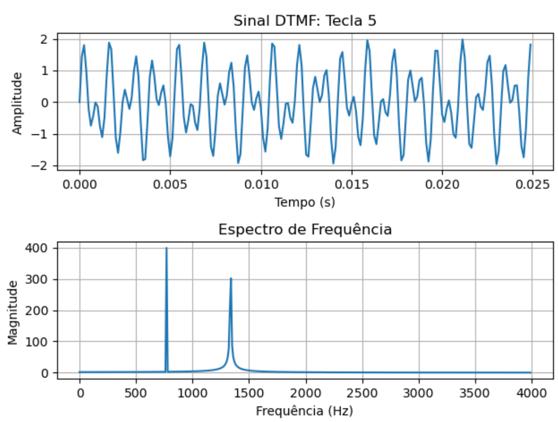


Figure 19: Sinal da tecla 5 no tempo e na frequência

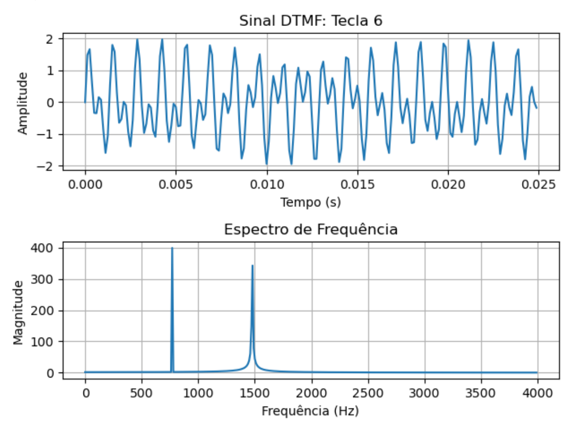


Figure 20: Sinal da tecla 6 no tempo e na frequência

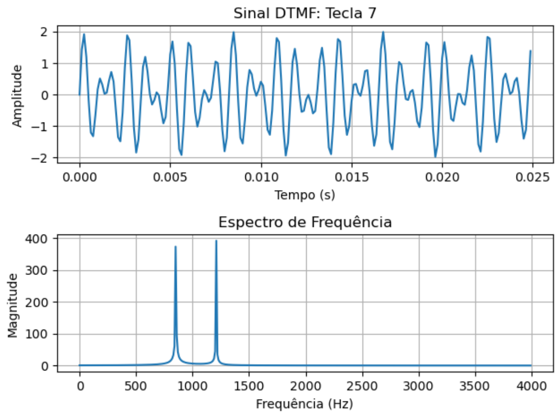


Figure 21: Sinal da tecla 7 no tempo e na frequência

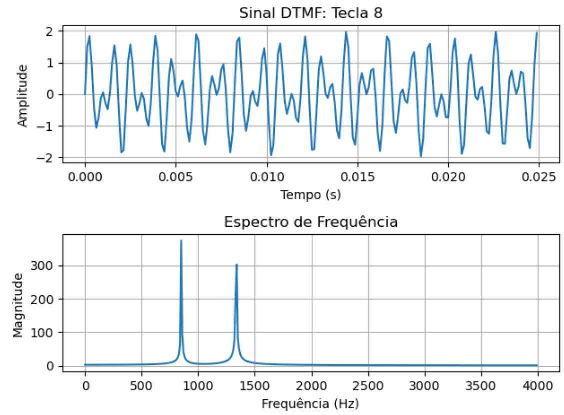


Figure 22: Sinal da tecla 8 no tempo e na frequência

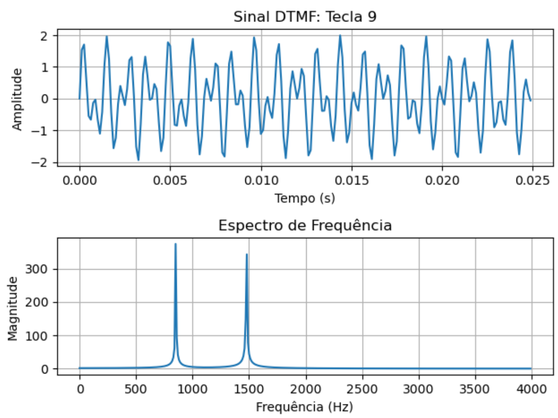


Figure 23: Sinal da tecla 9 no tempo e na frequência

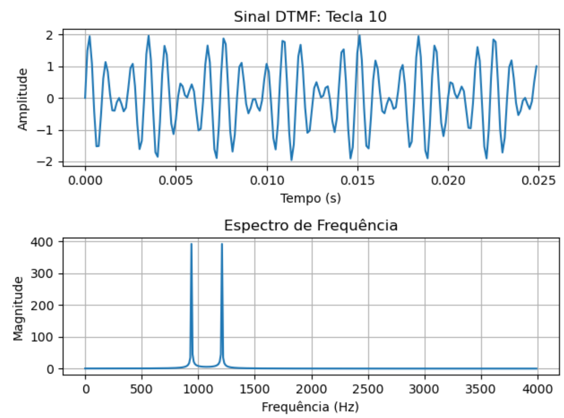


Figure 24: Sinal da tecla 10 no tempo e na frequência

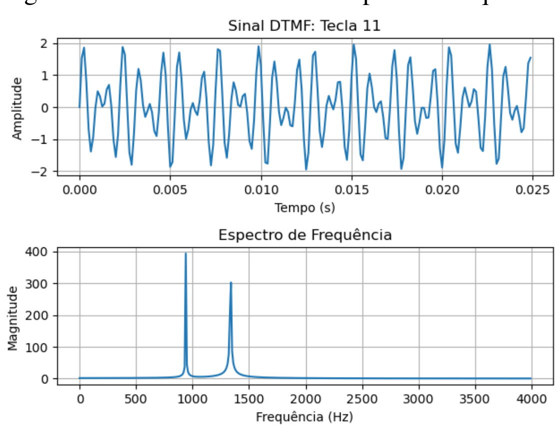


Figure 25: Sinal da tecla 11 no tempo e na frequência

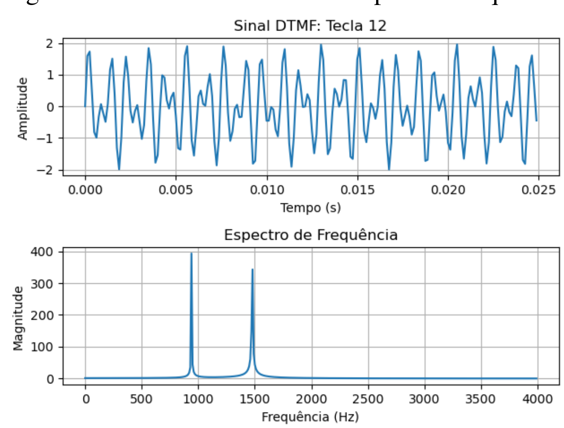


Figure 26: Sinal da tecla 12 no tempo e na frequência

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

### 2.3.1 Identifique a sequência de teclas.

É possível obter as teclas pressionadas ao analisar o espectrograma do sinal e relacionar as frequências presentes com o padrão DTMF. Abaixo é possível observar o espectrograma obtido do arquivo.

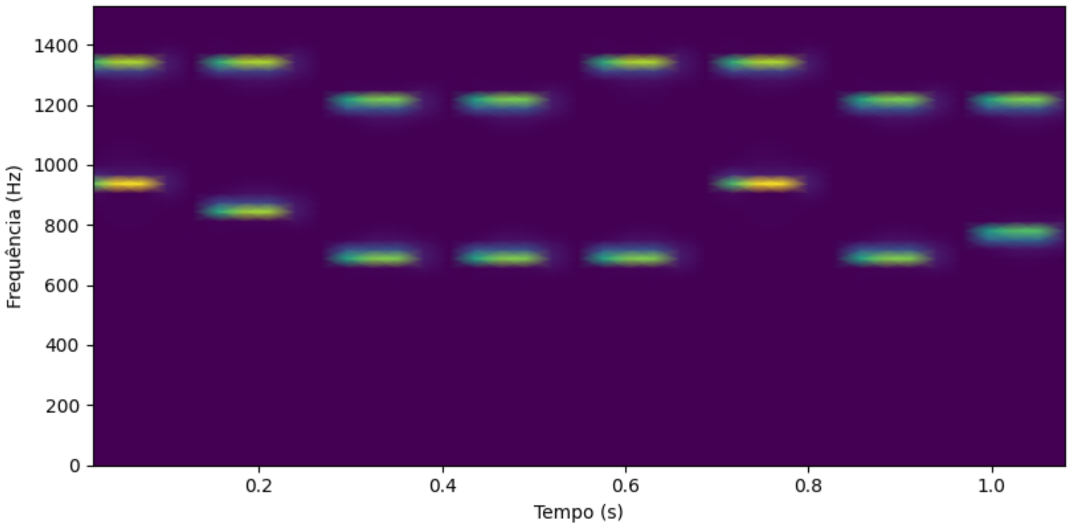


Figure 27: Espectrograma do arquivo $DTMF1.wav$

Conclui-se que as teclas pressionadas foram: $0, 8, 1, 1, 2, 0, 1, 4$.

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

```
y1 = np.convolve(audio, h1, mode = 'same')
```

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

```
y2 = np.convolve(audio, h2, mode ='same')
```

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

```
def plot_signal(titulo, sinal, fs):
    plt.figure(figsize=(12, 5))

    plt.subplot(2, 1, 1)
    plt.plot(sinal)
    plt.title(f"{titulo} - Dominio do Tempo")
    plt.xlabel("Amostras")
    plt.ylabel("Amplitude")
    plt.grid()

    Y = np.abs(np.fft.fft(sinal))
    freqs = np.fft.fftfreq(len(sinal), d=1/fs)

    plt.subplot(2, 1, 2)
    plt.plot(freqs[:len(freqs)//2], Y[:len(Y)//2])
    plt.title(f"{titulo} - Dominio da Frequencia")
    plt.xlabel("Frequencia (Hz)")
    plt.ylabel("Magnitude")
    plt.grid()
```

```python

    plt.tight_layout()
    plt.show()

plot_signal("Filtro H1", h1, fs)
plot_signal("Filtro H2", h2, fs)
```

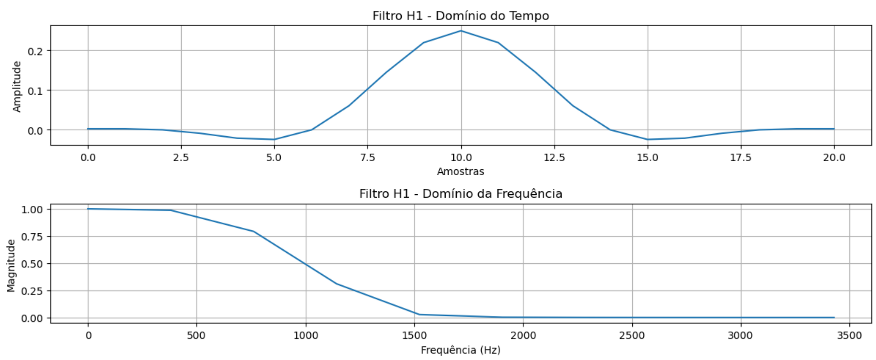


Figure 28: Resposta ao impulso e em frequência do filtro $H_1$

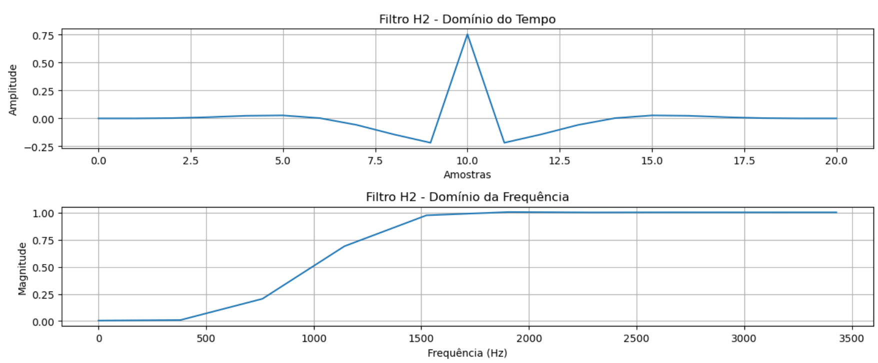


Figure 29: Resposta ao impulso e em frequência do filtro $H_2$

#### 2.3.5 Comente os resultados.

O filtro $H_1$ mostra uma resposta de frequência como um filtro passa-baixa. Enquanto $H_2$ atua como um filtro passa-alta.

Ao observar os gráficos, nota-se que os filtros têm diferentes distribuições de coeficientes, indicando que um deles suaviza o sinal enquanto o outro realça componentes de alta frequência.

Já no domínio da frequência, se vê como cada filtro afeta diferentes faixas do espectro do sinal de entrada.

#### 2.3.6 Mostre os gráficos de $y_1$ e $y_2$ no domínio da frequência.

```python
plot_signal("Sinal y1 (audio * H1)", y1, fs)
plot_signal("Sinal y2 (audio * H2)", y2, fs)
```

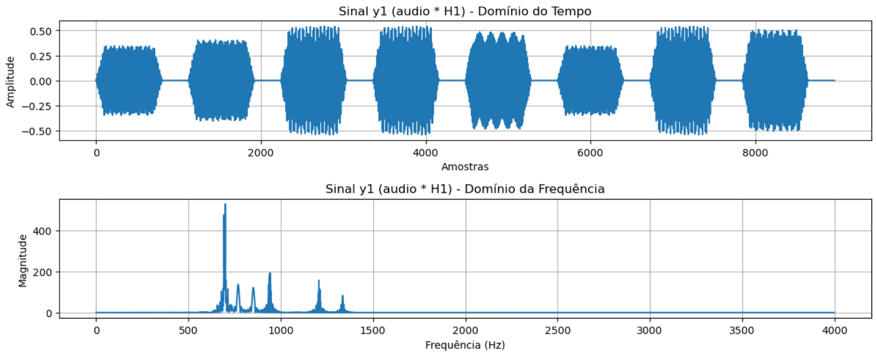


Figure 30: Convolução entre $x$ e $H_1$

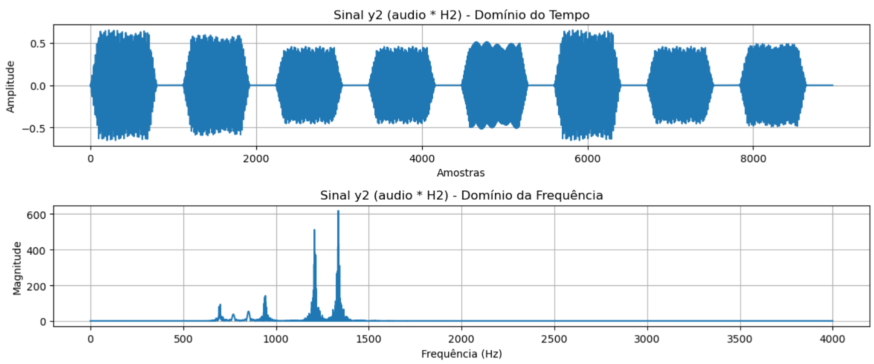


Figure 31: Convolução entre $x$ e $H_2$

#### 2.3.7 Comente os resultados da convolução de $x$ com $H_1$ ($y_1$) e de $x$ com $H_2$ ($y_2$).

Após a convolução do sinal de entrada do audio com o filtro $H_1$ e $H_2$, observa-se que o sinal $y_1$ mantém características semelhantes ao sinal original, mas com certas frequências suavizadas ou realçadas. Como $H_1$ é um passa-baixa, tem-se uma atenuação das frequências mais altas.

O sinal $y_2$ mostra um comportamento diferente, como $H_2$ é um filtro passa-alta, o sinal resultante tem menos componentes de baixa frequência, evidenciando detalhes e mudanças abruptas no sinal original.

Tais filtros podem ser utilizados para separar as componentes de baixa e alta frequência do sinal da tecla, essencialmente separando a informação das linhas (frequências menores) e das colunas (frequências maiores).

# 3 Cálculo da transformada discreta de Fourier (DFT)

## 3.1 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 frequência de amostragem de 1000Hz. Apresente graficamente (utilizando a função *stem*) as frequências componentes deste sinal, resultado da DFT.

```
#GRR202127676
Fs = 1e3
x = np.sin(2*np.pi*27*np.arange(0.4*Fs)/Fs)

npt = len(x)

```

```
w = np.zeros([npt, npt], dtype='complex')

for i in range(npt):
    w[i] = np.exp(-2j*np.pi*np.arange(npt)/npt)**i

w /= npt

fft_x = np.transpose(np.matmul(w, np.transpose(x)))

fig, ax = plt.subplots(2, 1)

ax[0].plot(abs(fft_x))
ax[1].plot(np.angle(fft_x))

ax[0].set_xlabel('k')
ax[1].set_xlabel('k')

ax[0].set_ylabel('Magnitude')
ax[1].set_ylabel('Fase (rad)')

ax[0].grid()
ax[1].grid()
plt.show()
```

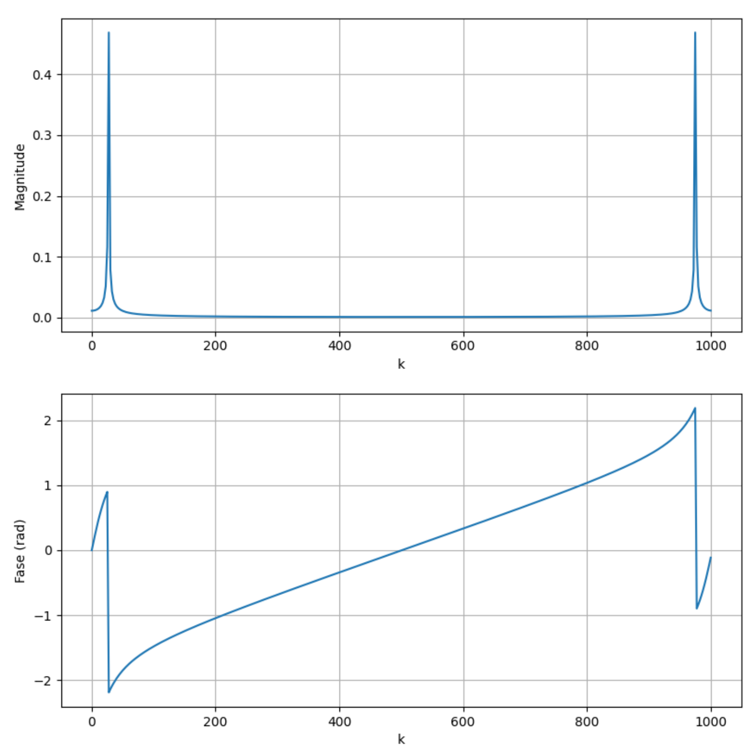


Figure 32: Transformada discreta de Fourier do sinal $x$

Para a obtenção do resultado não foi utilizada a função *stem* para a visualização, visto que a mesma impediria a compreensão clara do resultado da DFT pela quantidade de pontos utilizada.