Processamento Digital de Sinais - AB1

Aluno: Guilherme de Oliveira Costa

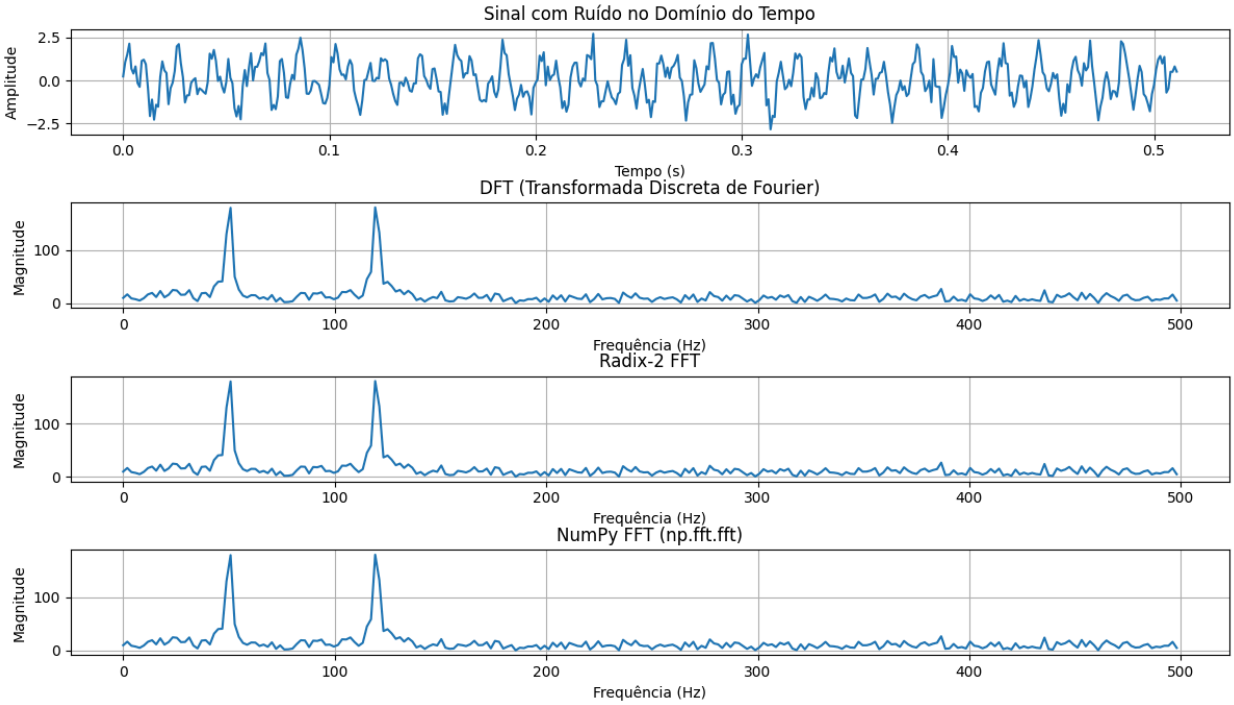
```python
import numpy as np
import matplotlib.pyplot as plt

def DFT(x):
    N = len(x)
    X = np.zeros(N, dtype=complex)
    for k in range(N):
        for n in range(N):
            X[k] += x[n] * np.exp(-2j * np.pi * k * n / N)
    return X

def FFT_radix2(x):
    N = len(x)
    if N <= 1:
        return x
    elif N % 2 > 0:
        raise ValueError("N deve ser uma potência de 2")  # Verifica se N é
potência de 2
    even = FFT_radix2(x[0::2])
    odd = FFT_radix2(x[1::2])
    factor = np.exp(-2j * np.pi * np.arange(N) / N)
    return np.concatenate([even + factor[:N//2] * odd,
                           even + factor[N//2:] * odd])

def FFT_radix3(x):
    N = len(x)
    if N == 1:
        return x
    elif N % 3 > 0:
        raise ValueError("N deve ser uma potência de 3")  # Verifica se N é
potência de 3
    x0 = FFT_radix3(x[0::3])
    x1 = FFT_radix3(x[1::3])
```

```
    x2 = FFT_radix3(x[2::3])
    factor = np.exp(-2j * np.pi * np.arange(N) / N)
    return np.concatenate([x0 + factor[:N//3] * x1 + factor[:N//3]**2 * x2,
                           x0 + factor[N//3:2*N//3] * x1 +
factor[N//3:2*N//3]**2 * x2,
                           x0 + factor[2*N//3:] * x1 + factor[2*N//3:]**2 *
x2])

# Escolher um N compatível com radix-2 e radix-3
N = 512  # Número de pontos # N = 512 e N = 243
fs = 1000  # Frequência de amostragem
f1, f2 = 50, 120  # Frequências do sinal
t = np.arange(N) / fs
signal = np.sin(2 * np.pi * f1 * t) + np.sin(2 * np.pi * f2 * t)

# Adicionando ruído ao sinal
noise_level = 0.5  # Ajuste o nível de ruído (desvio padrão)
noise = np.random.normal(0, noise_level, N)  # Ruído gaussiano
signal_with_noise = signal + noise  # Sinal com ruído

# Aplicação das transformadas
X_dft = DFT(signal_with_noise)

# Verifica se N é potência de 2
def is_power_of_2(n):
    return (n & (n - 1)) == 0 and n > 0

if is_power_of_2(N):
    X_fft2 = FFT_radix2(signal_with_noise)
else:
    X_fft2 = None

X_fft_np = np.fft.fft(signal_with_noise)

# Apenas executa FFT_radix3 se N for potência de 3
def is_power_of_3(n):
    while n % 3 == 0 and n > 1:
        n //= 3
```

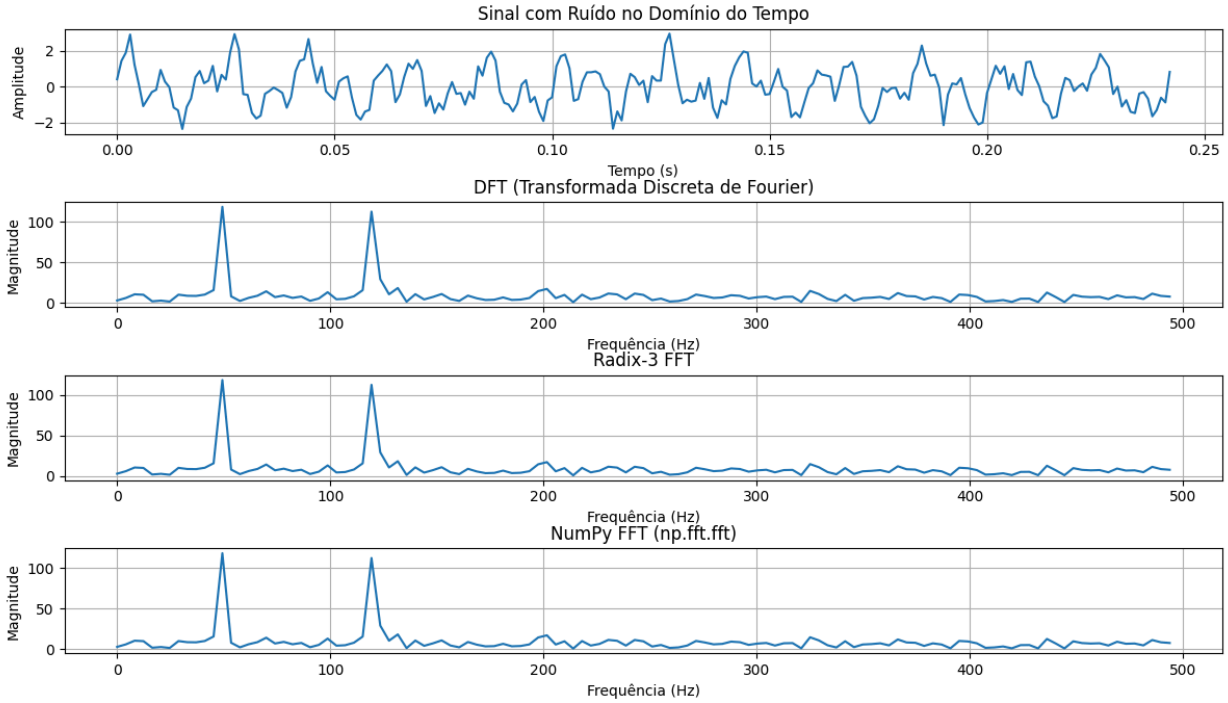
```
    return n == 1

if is_power_of_3(N):
    X_fft3 = FFT_radix3(signal_with_noise)
else:
    X_fft3 = None

# Frequências associadas
freqs = np.fft.fftfreq(N, d=1/fs)

# Plot dos resultados
plt.figure(figsize=(12, 10))

# Sinal no domínio do tempo
plt.subplot(5, 1, 1)
plt.plot(t, signal_with_noise)
plt.title("Sinal com Ruído no Domínio do Tempo")
plt.xlabel("Tempo (s)")
plt.ylabel("Amplitude")
plt.grid()

# Função auxiliar para plotagem do espectro
def plot_fft(ax, freqs, X, title):
    ax.plot(freqs[:N//2], np.abs(X[:N//2]))
    ax.set_title(title)
    ax.set_xlabel("Frequência (Hz)")
    ax.set_ylabel("Magnitude")
    ax.grid()

# Comparação dos resultados das FFTs
ax1 = plt.subplot(5, 1, 2)
plot_fft(ax1, freqs, X_dft, "DFT (Transformada Discreta de Fourier)")

if X_fft2 is not None:
    ax2 = plt.subplot(5, 1, 3)
    plot_fft(ax2, freqs, X_fft2, "Radix-2 FFT")

if X_fft3 is not None:
```

```
    ax3 = plt.subplot(5, 1, 3)
    plot_fft(ax3, freqs, X_fft3, "Radix-3 FFT")

ax4 = plt.subplot(5, 1, 4)
plot_fft(ax4, freqs, X_fft_np, "NumPy FFT (np.fft.fft)")

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

Para N = 512



Para N = 243


Sinal com Ruído no Domínio do Tempo
Amplitude
Tempo (s)
DFT (Transformada Discreta de Fourier)
Magnitude
Frequência (Hz)
Radix-3 FFT
Magnitude
Frequência (Hz)
NumPy FFT (np.fft.fft)
Magnitude
Frequência (Hz)