Exercício Computacional - Processos Estocásticos

4,0

Igor Costa D'Oliveira - 11391446

1) Filtragem de um sinal ruidoso

a) Considere o sinal x0(t) = (sin^3(Ωt))e^(− t/τ) , t ≥ 0; 0, caso contrário, com Ω = 2π × 500 rad/s e τ = 0.5 s.

Amostre x0(t) com uma frequência de amostragem fa = 40 kHz por 2 segundos para obter o sinal discreto x0[n]. Qual é o comprimento de x0[n]? Ouça o sinal x0[n].

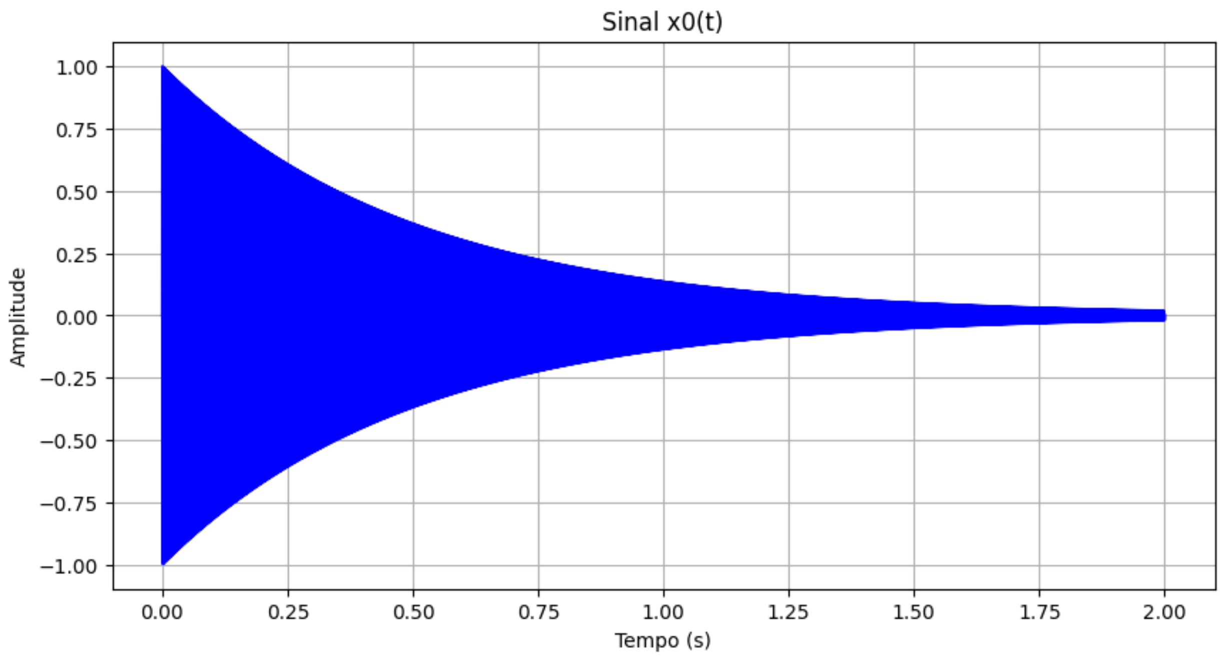
Igor, para gerar um .pdf a partir do .ipynb, escolha File → Print Preview, e imprima a página resultante em pdf.

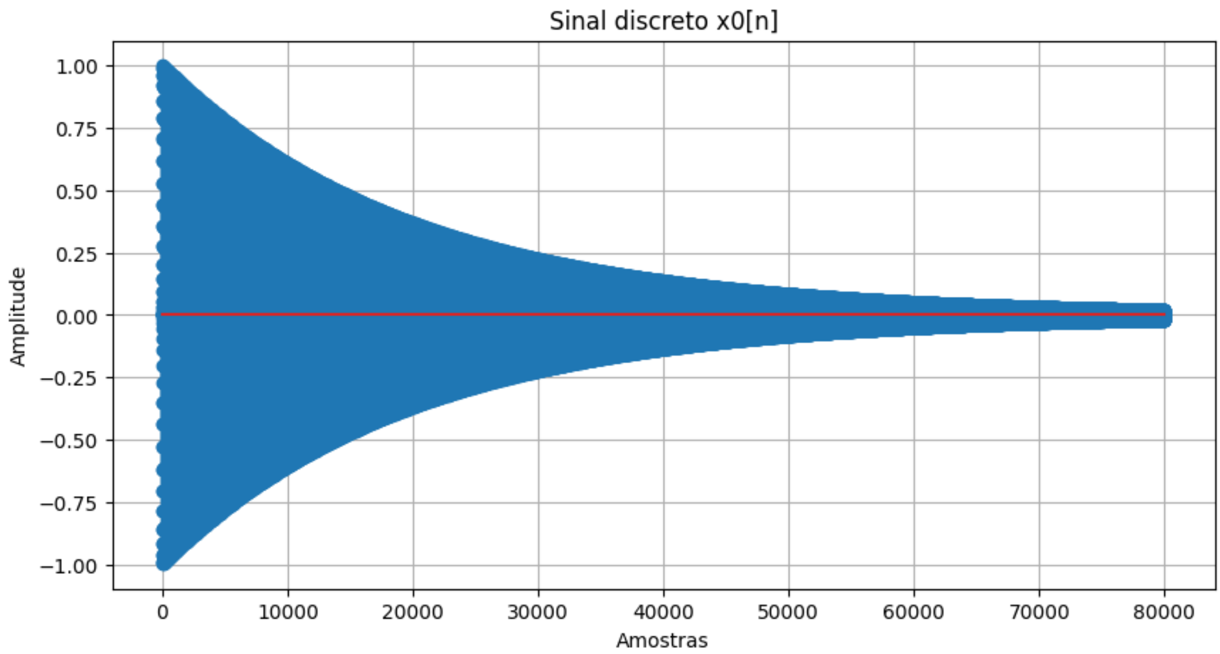
In [179…

```
import numpy as np
import matplotlib.pyplot as plt
#import sounddevice as sd

# Definição dos parâmetros do sinal x0(t)
omega = 2 * np.pi * 500  # Frequência angular
tau = 0.5  # Constante de tempo

# Definição da função x0(t)
def x0(t):
  return np.power(np.sin(omega * t), 3) * np.exp(-t/tau)

# Frequência de amostragem e período de amostragem
fa = 40000
T = 1/fa

# Número de amostras
N = 2 * fa

# Amostragem do sinal x0(t)
t = np.arange(N) * T
x0n = x0(t)

# Plotagem do sinal x0(t)
plt.figure(figsize=(10, 5))
plt.plot(t, x0(t), 'b')
plt.title('Sinal x0(t)')
plt.xlabel('Tempo (s)')
plt.ylabel('Amplitude')
plt.grid()
plt.show()

# Plotagem do sinal discreto x0[n]
plt.figure(figsize=(10, 5))
plt.stem(x0n)
plt.title('Sinal discreto x0[n]')
plt.xlabel('Amostras')
plt.ylabel('Amplitude')
plt.grid()
plt.show()

# Comprimento do sinal discreto x0[n]
print('\n\nComprimento de x0[n]:', len(x0n))

# Reproduzir o som do sinal
#sd.play(x0n, fa)
#sd.wait()
```

Está certo, mas veja que o último ponto não é $t=2$.

```
(80000,)
```



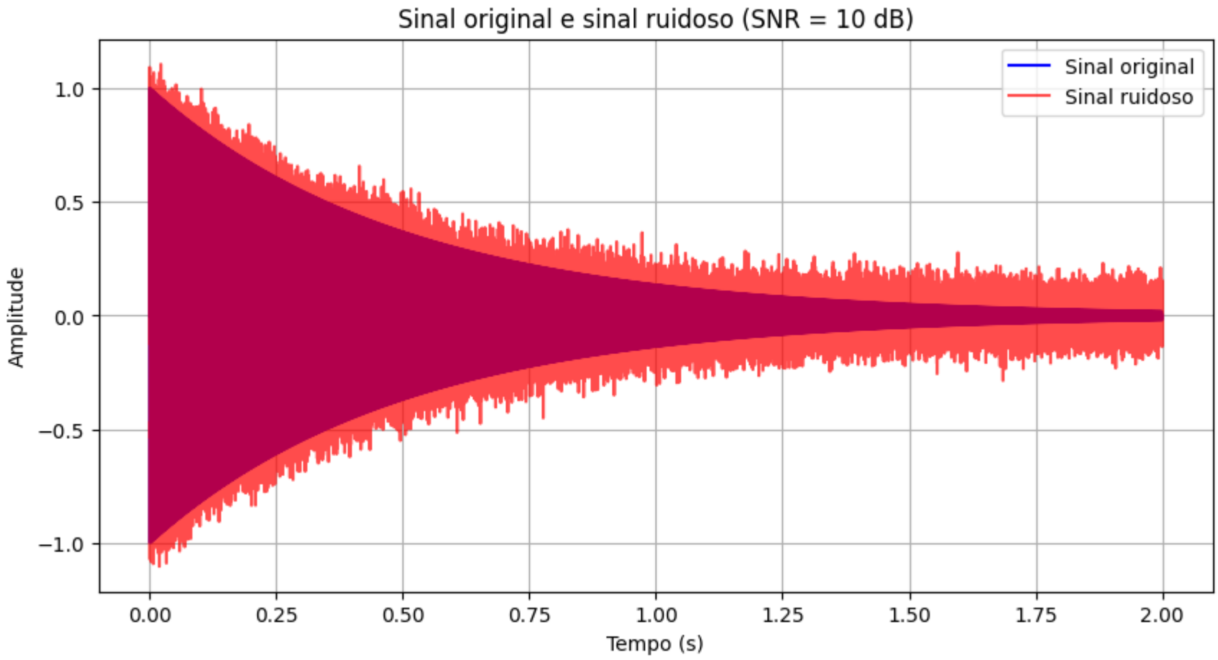



```
Comprimento de x0[n]: 80000
```

b) Adicione um ruído branco gaussiano a x0[n] para obter o sinal ruidoso x[n], de forma que x[n] tenha SNR igual a 10 dB. Mostre no mesmo gráfico o sinal ruidoso e o sinal sem ruído. Ouça o sinal x[n]

In [172…

```python
# Definição do SNR e cálculo do desvio padrão do ruído
SNR = 10  # dB
Psinal = np.mean(x0n**2)
desvio_padrao = np.sqrt(Psinal / (10**(SNR/10)))

# Adição do ruído branco gaussiano ao sinal x0[n]
xn = x0n + desvio_padrao * np.random.randn(len(x0n))

# Plotagem do sinal ruidoso e do sinal original
plt.figure(figsize=(10, 5))
plt.plot(t, x0(t), 'b', label='Sinal original')
plt.plot(t, xn, 'r', alpha=0.7, label='Sinal ruidoso')
plt.title('Sinal original e sinal ruidoso (SNR = 10 dB)')
plt.xlabel('Tempo (s)')
plt.ylabel('Amplitude')
plt.grid()
plt.legend()
plt.show()

# Reproduzir o som do sinal
#sd.play(xn, fa)
#sd.wait()
```



c) Considere o filtro com resposta ao impulso $h[n] = 0,1 \operatorname{sinc}(0,1(n-50))$, para $0 \leq n \leq 100$, e zero caso contrário. Plote a resposta em frequência deste filtro

```
In [173…
# Definição da resposta ao impulso h[n]
n = np.arange(0, 101)
h = 0.1 * np.sinc(0.1 * (n - 50))

# Cálculo da resposta em frequência H(e^(jw))
w = np.linspace(-np.pi, np.pi, 1000)
H = 0.1 * np.sum(h * np.exp(-1j * np.outer(w, n)), axis=1)

# Plotagem da resposta em frequência
plt.figure(figsize=(8, 5))
plt.plot(w, np.abs(H), 'b')
plt.title('Resposta em frequência do filtro')
plt.xlabel('Frequência (rad/s)')
plt.ylabel('Magnitude')
plt.grid()
plt.show()
```

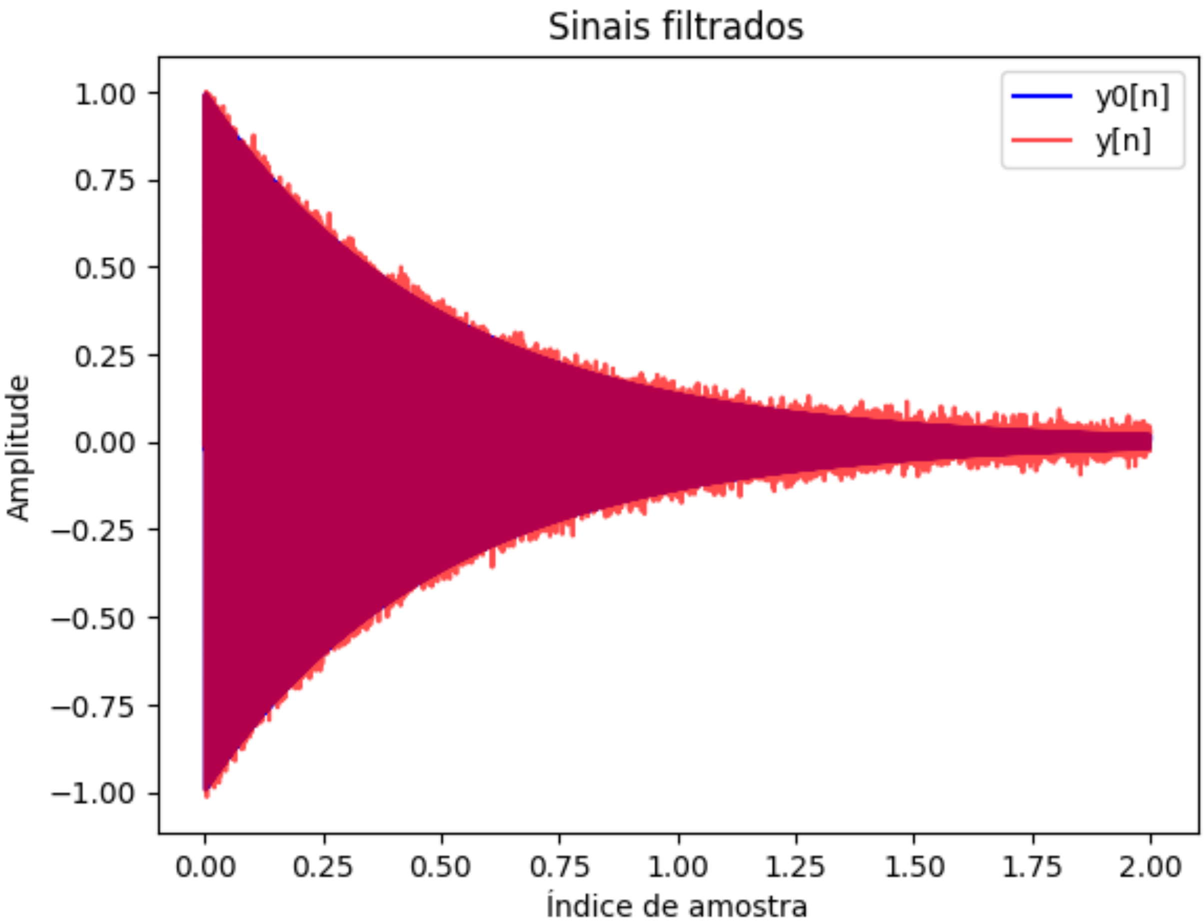
Podemos observar que este filtro é um filtro passa-baixa, com corte em torno de 0,1 rad/s (correspondente a 16 Hz, considerando uma frequência de amostragem de 40 kHz)

$$\omega = \frac{2\pi f}{f_a} \Rightarrow f = \frac{\omega f_a}{2\pi} = \frac{0{,}1\pi f_a}{2\pi} = \frac{0{,}1 f_a}{2} = \underline{\underline{2kHz}}$$

d) Passe os sinais x0[n] e x[n] pelo filtro H(z), obtendo os sinais y0[n] e y[n]. Plote no mesmo gráfico y0[n] e y[n]. Ouça os sinais y0[n] e y[n].

In [182…

```
y0_n = np.convolve(x0(t), h)[:len(x0(t))]  # Sinal filtrado sem ruído
y_n = np.convolve(xn, h)[:len(xn)]  # Sinal filtrado com ruído

# Plotagem dos sinais y0[n] e y[n]
plt.plot(t, y0_n, 'b', label='y0[n]')
plt.plot(t, y_n, 'r', alpha=0.7, label='y[n]')
plt.legend()
plt.xlabel('Índice de amostra')
plt.ylabel('Amplitude')
plt.title('Sinais filtrados')
plt.show()
```



Podemos observar que o sinal ruidoso x[n] foi atenuado pelo filtro, enquanto o sinal sem ruído x0[n] foi preservado. Isso ocorre porque o filtro possui uma resposta em frequência passa-baixa, que atenua as frequências mais altas presentes no sinal ruidoso.

e) Explique porque x[n] e y[n] não são processos estacionários em nenhum sentido, mas os ruídos vx[n] = x[n]−x0[n] e vy[n] = y[n]−y0[n] são estacionários no sentido amplo (no caso de vy[n], desprezando-se o transitório do filtro).

Os sinais x[n] e y[n] não são processos estacionários em nenhum sentido porque suas estatísticas, como média e variância, mudam ao longo do tempo. Isso ocorre porque eles contêm componentes determinísticos que variam ao longo do tempo (o sinal senoide e a exponencial decrescente).

Por outro lado, os ruídos vx[n] e vy[n] são estacionários no sentido amplo porque suas estatísticas não mudam ao longo do tempo, uma vez que são compostos apenas de componentes estocásticos, ou seja, não contêm informações que variam ao longo do tempo. Nesse caso, a média e a variância dos ruídos permanecem constantes ao longo do tempo, o que caracteriza a estacionariedade no sentido amplo.

f) Calcule a expressão teórica da densidade espectral de potência do ruído na entrada e na saída, bem como a potência do ruído na entrada e na saída.

→ Um sinal pode ser assim e ter características que variam ao longo do tempo.

Para o sinal ser WSS, a função valor esperado deve ser constante e a função de autocorrelação deve depender apenas do intervalo entre os instantes considerados — não é só a variância ser constante.

Considerando que o ruído adicionado em x0[n] é gaussiano branco, sua densidade espectral de potência (PSD) é constante em toda a faixa de frequência, dada por:

$Sx(f) = \sigma^2$, onde σ^2 é a variância do ruído gaussiano branco adicionado.

A potência do ruído na entrada é obtida integrando a PSD de Sx(f) ao longo de toda a faixa de frequência, ou seja:

$Px = \int Sx(f)df = \sigma^2 \int df = \sigma^2 f \ |f=-fs/2$ até $f=fs/2 = \sigma^2 fs$, onde fs é a frequência de amostragem.

Já para a PSD do ruído na saída, devemos levar em conta a resposta em frequência do filtro H(z), que é dada por:

$H(f) = 0{,}1e\text{\^{}}(-j2\pi f50)\sin(\pi f/5)/(\pi f)$ → Esta é a transformada de um pulso retangular de tempo contínuo. A transformada de um pulso de tempo discreto tem sen$\left(\frac{\omega}{2}\right)$ no denominador (veja o caderno).

Assim, a PSD do ruído na saída é dada por:

$Sy(f) = |H(f)|^2 \, Sx(f)$

Substituindo Sx(f) e H(f), temos:

$Sy(f) = |H(f)|^2 \sigma^2$

Portanto, a PSD do ruído na saída é a PSD do ruído de entrada multiplicada pelo quadrado da magnitude da resposta em frequência do filtro.

A potência do ruído na saída é obtida integrando a PSD de Sy(f) ao longo de toda a faixa de frequência, ou seja:

$Py = \int Sy(f)df = \sigma^2 \int |H(f)|^2 df = \sigma^2 \int |H(f)|^2 df \ |f=-fs/2$ até $f=fs/2$

A potência do ruído na saída pode ser calculada numericamente a partir da PSD de Sy(f) ou analiticamente a partir da resposta em frequência do filtro H(z).

→ Vocês parecem estar misturando as expressões para tempo contínuo e para tempo discreto. O nosso sinal foi amostrado, portanto é necessário usar fórmulas para tempo discreto. Em particular, reparem que (devido ao rebatimento), se

$p_T(t) = \begin{cases} 1, & 0 \leq t \leq T \\ 0, & t < 0 \text{ ou } t > T \end{cases}$ e $P_T(j\Omega)$ é a sua transf. de Fourier, e se $p_T[n] = p_T(nTa)$,

não vale $P_T(e^{j\omega}) = \sum_{n=-\infty}^{\infty} p_T[n] e^{-j\omega n} \neq P_T(j\Omega)\big|_{\Omega = \omega \cdot fa}$.

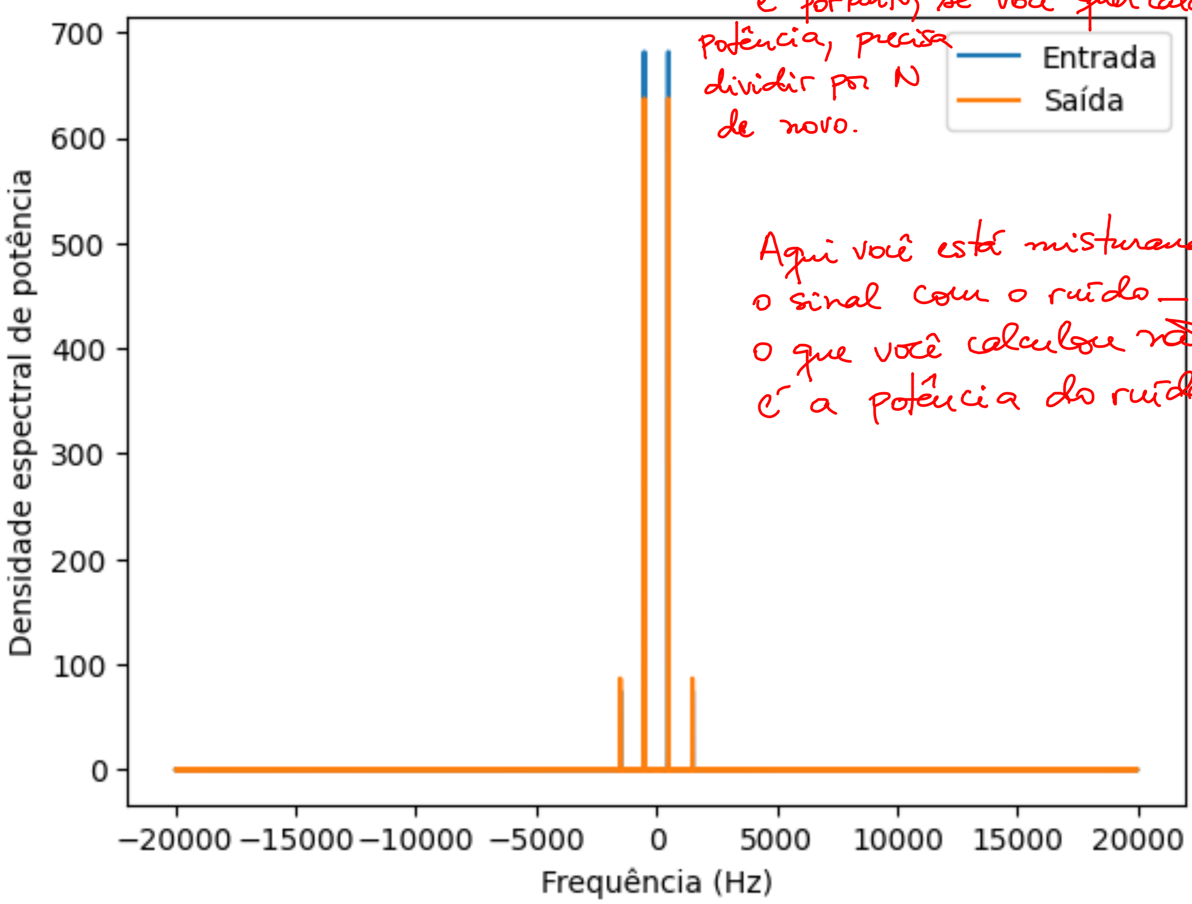
In [200…

```python
# Transformada de Fourier dos sinais
X = np.fft.fft(xn)
Y = np.fft.fft(y_n)

# A densidade espectral de potência (PSD) de um sinal x(t) é definida com
Pn = (1/N)*np.abs(X)**2
Pn_out = (1/N)*np.abs(Y)**2
freq = np.fft.fftfreq(N, T)

plt.plot(freq, Pn, label='Entrada')
plt.plot(freq, Pn_out, label='Saída')
plt.xlabel('Frequência (Hz)')
plt.ylabel('Densidade espectral de potência')
plt.legend()
plt.show()

# Calculando a potência do ruído na entrada e na saída
Pn = (1/N)*np.sum(np.abs(X)**2)
Pn_out = (1/N)*np.sum(np.abs(Y)**2)

print('\n\nPotência do ruído na entrada:', Pn)
print('\nPotência do ruído na saída:', Pn_out)
print('\nA potência do ruído diminuiu na saída por causa do filtro que el
```

Devido às propriedades da FFT, isto é N·Potência. Da maneira como definimos FFT,

$$\sum x^2[n] = \frac{1}{N}\sum |X[n]|^2,$$

e portanto, se você quer calcular potência, precisa dividir por N de novo.



Aqui você está misturando o sinal com o ruído — o que você calculou não é a potência do ruído.

Potência do ruído na entrada: 3443.0203108610654

Potência do ruído na saída: 3029.925825871206

Estes valores estão muito grandes, correspondem a 80.000 × (Pot. do sinal + Pot. do ruído)

A potência do ruído diminuiu na saída por causa do filtro que eliminou al
gumas de suas componentes

g) Meça experimentalmente a potência média do ruído em x[n] e em y[n] por duas maneiras:

(a) Calculando rvx[0] e rvy [0] pela média de diversas realizações dos processos. Por exemplo no caso de rvx[0] use a fórmula abaixo para L = 1000:

Calcule a expressão para todos os valores de n, e veja se eles são aproximadamente constantes ou não.

In [213…

```
# Calculando a potência do ruído na entrada e na saída
Pn = (1/1000)*np.sum(np.abs(X[:1000])**2)
Pn_out = (1/1000)*np.sum(np.abs(Y[:1000])**2)

Pnm = (1/1000)*np.mean(np.abs(X[:1000])**2)
Pn_outm = (1/1000)*np.mean(np.abs(Y[:1000])**2)

print('\n\nPotência do ruído na entrada:', Pn)
print('\nPotência do ruído na saída:', Pn_out)

print('\n\nMÉDIA do ruído na entrada:', Pnm)
print('\nMÉDIA do ruído na saída:', Pn_outm)

print('Sim, eles são aproximadamente constantes')
```

```
Potência do ruído na entrada: 29551.20846275611

Potência do ruído na saída: 27697.06355265655


MÉDIA do ruído na entrada: 29.551208462756108

MÉDIA do ruído na saída: 27.69706355265655
Sim, eles são aproximadamente constantes
```

Eu ainda não expliquei como calcular pot. média usando a FFT – há alguns detalhes que é necessário considerar. Os valores que você obteve estão muito grandes. E, como você está fazendo a conta no domínio transformado, estes resultados correspondem a uma única realização

b) Usando o fato dos ruídos serem ergódicos, ou seja, usando a expressão rvx[0] ≈ 1NNX−1n=0 em que agora l corresponde a uma realização qualquer do ruído, e N é o comprimento do sinal.

Compare os resultados obtidos com as duas expressões entre si e com o valor teórico. Os resultados ficam mais próximos quando os valores de N e L são aumentados para 10.000? Para 100.000

In [214…

```
# Calculando a potência do ruído na entrada e na saída
Pn = (1/80000)*np.sum(np.abs(X[:80000])**2)
Pn_out = (1/80000)*np.sum(np.abs(Y[:80000])**2)

Pnm = (1/80000)*np.mean(np.abs(X[:80000])**2)
Pn_outm = (1/80000)*np.mean(np.abs(Y[:80000])**2)

print('\n\nPotência do ruído na entrada:', Pn)
print('\nPotência do ruído na saída:', Pn_out)

print('\n\nMÉDIA do ruído na entrada:', Pnm)
print('\nMÉDIAa do ruído na saída:', Pn_outm)

print('Sim, quando N e L são aumentados eles diminuem e ficam mais próxim
```

```
Potência do ruído na entrada: 3443.0203108610654

Potência do ruído na saída: 3029.925825871206


MÉDIA do ruído na entrada: 0.04303775388576332

MÉDIAa do ruído na saída: 0.03787407282339008
Sim, quando N e L são aumentados eles diminuem e ficam mais próximos
```

Compare com os valores calculados no tempo.

Foi boa a ideia de usar a FFT, mas infelizmente não deu certo — a FFT tem alguns detalhes que nós não vimos ainda, e que causaram problemas quando você tentou calcular as potências médias.

Independente da FFT, para calcular a potência do ruído você precisava subtrair o sinal, e isto fez com que seus valores dessem errado. Também houve uma confusão na hora de calcular $r_{vx}[0]$ pela definição e considerando que o sinal é ergódico.