UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

***CAMPUS* CEUNES**

CURSO DE ENGENHARIA DE COMPUTAÇÃO

LUIZ FILLIPE OLIVEIRA MORAIS

AMOSTRAGEM

SÃO MATEUS-ES
2022

LUIZ FILLIPE OLIVEIRA MORAIS

AMOSTRAGEM

Trabalho realizado pelo aluno de Graduação em Engenharia de Computação apresentado à disciplina de Sinais e Sistemas.

Prof.ª Drª. Daniel José Custódio Coura

SÃO MATEUS-ES
2022

Sumário

INTRODUÇÃO

A amostragem, em termos de processamento de sinais, é a transformação de um sinal contínuo (mundo real) em um sinal discreto. Sua definição é o processo de medição instantânea de valores de um sinal analógico em intervalos regulares. O intervalor entre as amostras é determinado por um pulso de sincronismo e a sua frequência é chamada de taxa de amostragem.

A quantização e a codificação também fazem parte de amostragem, sendo a quantização a etapa responsável por mapear valores de entrada de um grande conjunto (geralmente um conjunto contínuo) para valores de saída em um conjunto menor (contável), geralmente com um número finito de elementos. E a codificação transforma a amostra quantificada em uma sequência binária correspondente.

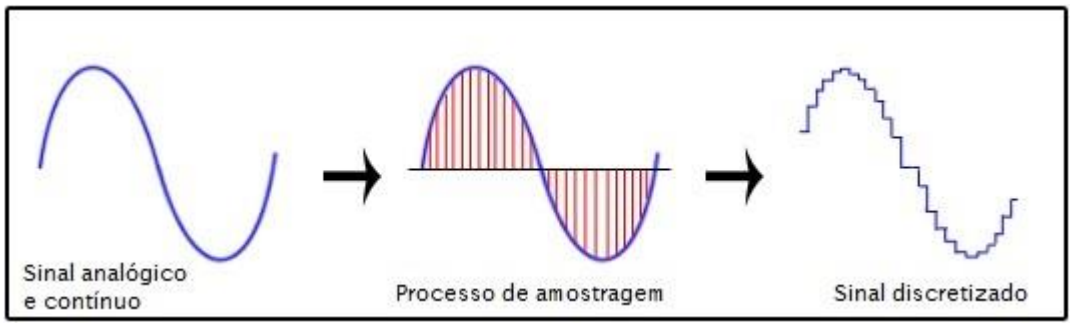

Figura 1 - Transformação do sinal analógico em discreto

OBJETIVO

O objetivo desta prática foi aprender como amostrar qualquer tipo de sinal analógico presente no mundo real em um sistema de computador utilizando a linguagem de programação Python, a fim de manipula-lo utilizando sistemas invariante no tempo e retransformados para analógicos. Sendo assim foi possível trazer o conhecimento teórico para a prática e comparar o funcionamento das transformadas que foi aprendido em sala de aula.

DESENVOLVIMENTO

O sinal escolhido

O sinal escolhido é um trecho de um diálogo entre a ovelha e o lobo dos contos dos Kindred do universo de League of Legends. Este sinal, é um áudio em formato **WAV** (Waveform Audio File format) desenvolvida pela Microsoft e IBM. É um formato a qual armazena o som sem perda de qualidade, reproduzindo com alta fidelidade tudo que foi gravado. Por não perder dado ele exige bastante espaço de armazenamento em relação a outros formatos.

Descrevendo o sinal

Usando a biblioteca *wave* pertencente a linguagem de programação Python, podemos importar o áudio e usar suas funções para obter informações sobre o áudio escolhido.

```
import wave as fw

arquivo = 'ovelhaMONO.wav';
arquivoWav = fw.open(arquivo, 'r');

print("Número canais: ", arquivoWav.getnchannels());
print("Número bytes: ", arquivoWav.getsampwidth());
print("Taxa de amostragem: ", arquivoWav.getframerate());
print("Número de frames: ", arquivoWav.getnframes());
print("Compactação: ", arquivoWav.getcompname());
print("Duração (s): ", arquivoWav.getnframes() / float(arquivoWav.getframerate()))
```

Figura 2 – [in] obtendo informações do sinal via Python

```
Número canais:  1
Número bytes:  2
Taxa de amostragem:  48000
Número de frames:  1307040
Compactação:  not compressed
Duração (s):  27.23
```

Figura 2 – [out] Saída do trecho de código

Como amostrado pelo código acima, o nosso sinal tem uma taxa de amostragem de 4,8 kHz, apenas um canal (mono), com 1.307.040 quadros e duração de 27,23 segundos.

Fazendo um simples código usando a biblioteca *soundfile*, podemos plotar o seu gráfico em relação ao tempo:

```
import soundfile as sf
sinal, amostragem = sf.read('ovelhaMONO.wav')
time = np.arange(0, len(sinal) * 1/amostragem, 1/amostragem)
```

Figura 3 - Trecho responsável por armazenar os dados do sinal

O que gera o seguindo gráfico:

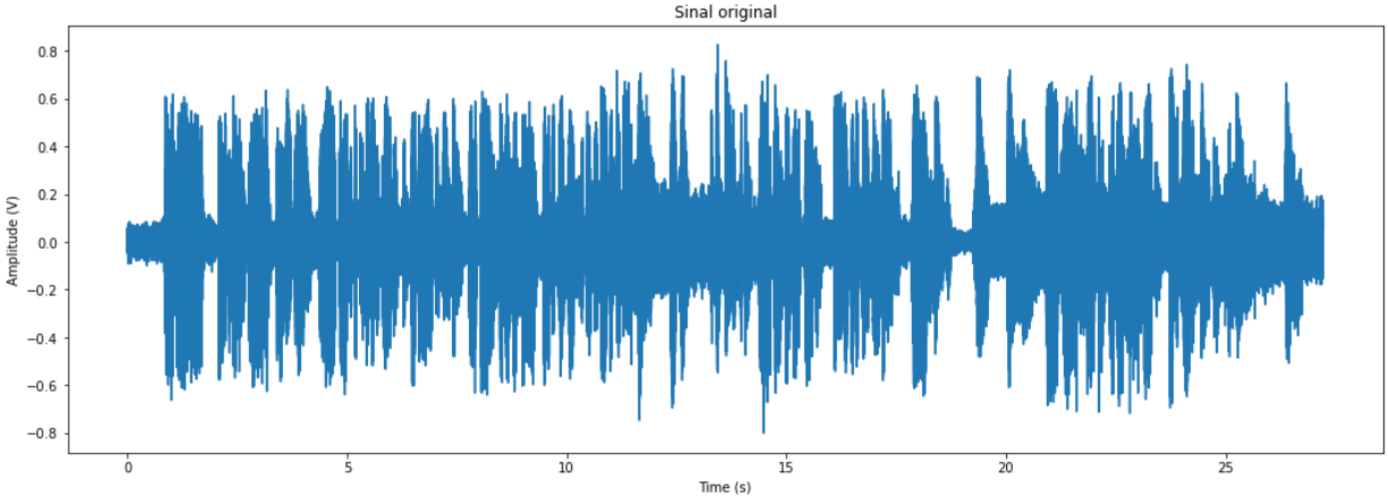

Figura 4 – Gráfico de amostragem por tempo (s) do sinal escolhido

Classificação do sinal

Contínuo: Esse sinal ele é contínuo (analógico), logo está definido para todo instante *t*.
Sinal escalar: É aqueles provenientes de uma única fonte (áudio mono).
Aperiódico: Não existe um T tal que satisfaça a seguinte condição: x(t)= x(t + T)
Sinal aleatório: Não pode ser representado por uma função analítica (não é possível determinar precisamente o valor do sinal em um dado instante de tempo).
Sinal causal: É definido apenas para $t > 0$.

Provando que o sinal escolhido é o a ser amostrado e as suas análises

Para tal, iremos analisar o sinal no tempo e na frequência e após isso, salvaremos o áudio manipulado através da FFT inversa plotando o seu gráfico (no tempo) e gerando o seu arquivo .wav comparando com o sinal original.
Iremos fazer as análises através do Python, e abaixo está o código usado para importar o áudio, fazer as manipulações necessárias e fazer a sua FFT para encontrarmos o seu espectro e fase.

```
#Importando o áudio
arquivo = 'ovelhaMONO.wav';
arquivoWav = fw.open(arquivo, 'r');

#Forçando o áudio a ser do tipo int16
tipos = np.int16;
Damp = 32760;
frames = arquivoWav.readframes(-1);
#Criando o np.array das amostragens do sinal
Amplitude = np.fromstring(frames, tipos)/Damp;

arquivoWav.close();

#Fazendo a FFT (Fast Fourier Transform)
N = 2**18
Tam = time[1]- time[0]
FFT = np.fft.fft(Amplitude,N) * Tam
FFT_2 = np.fft.fft(Amplitude)
w = np.fft.fftfreq(len(FFT), d=Tam)
wd = np.fft.fftshift(w)
Xd = np.fft.fftshift(FFT)

ModX = np.abs(Xd)
phaseD = np.angle(np.fft.fftshift(FFT))
```

Figura 5 - Trecho de código para a FFT

Após isso, vamos plotar os gráficos e fazer as análises. O primeiro gráfico refere-se ao sinal original demonstrado suas amplitudes no tempo em segundos, já o segundo é a sua transformada de Fourier (FFT) que no código é chamada de *ModX*.

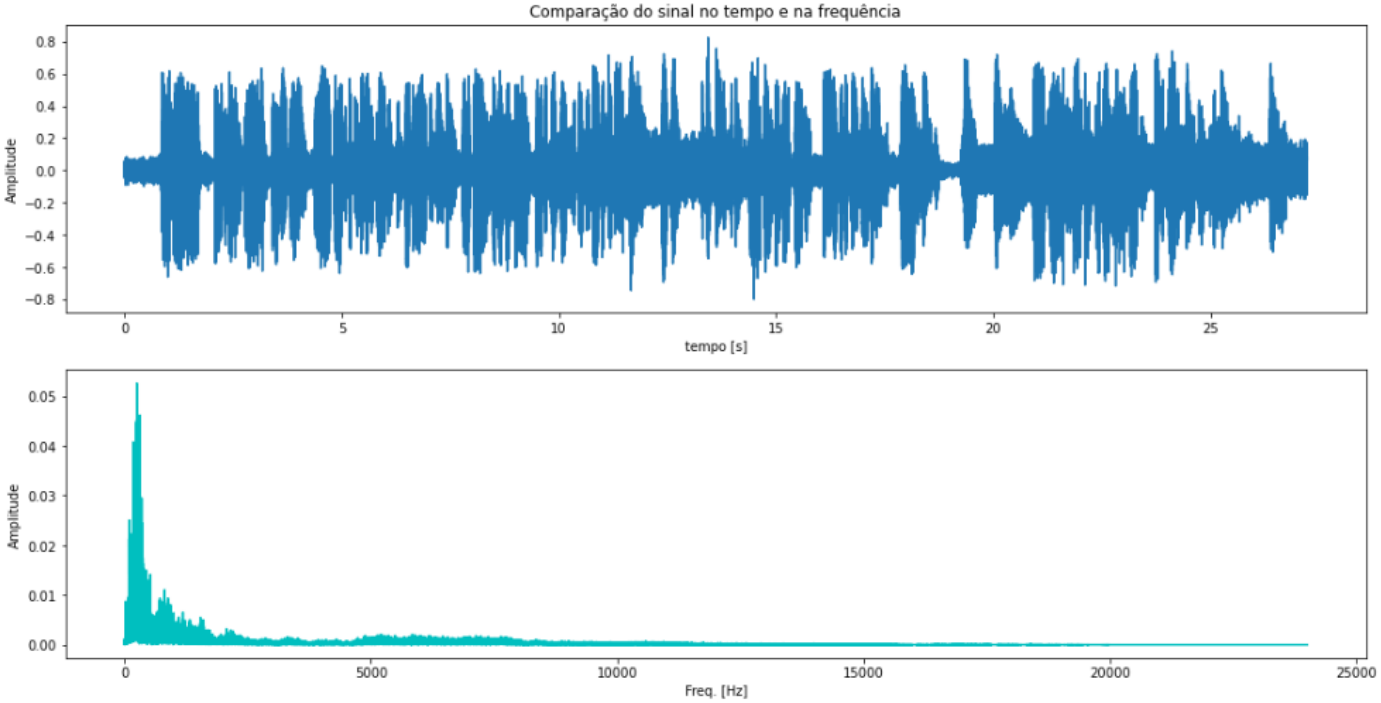

Figura 6 - Azul escuro: sinal em função do tempo; Ciano: Sinal na frequência

Para uma melhor visualização do sinal na frequência, iremos plotar até a frequência de 3400Hz pois como se trata de um diálogo podemos usar essa banda que é utilizada na telefonia sem perdas de qualidade.

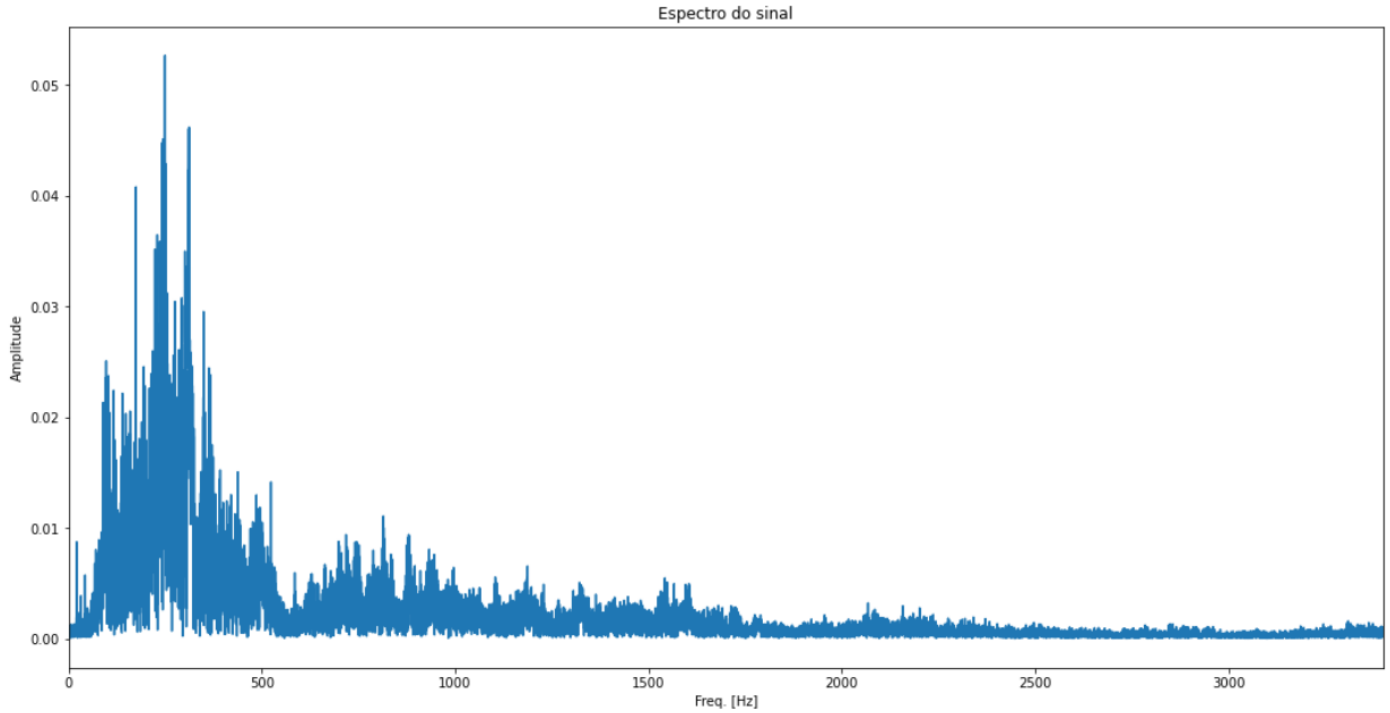

Figura 7 - Representação do sinal até a frequência de 3.4 KHz

Como sabemos, a magnitude do espectro é par, mas a sua fase é ímpar, para provar isso, plotaremos a sua fase a seguir:

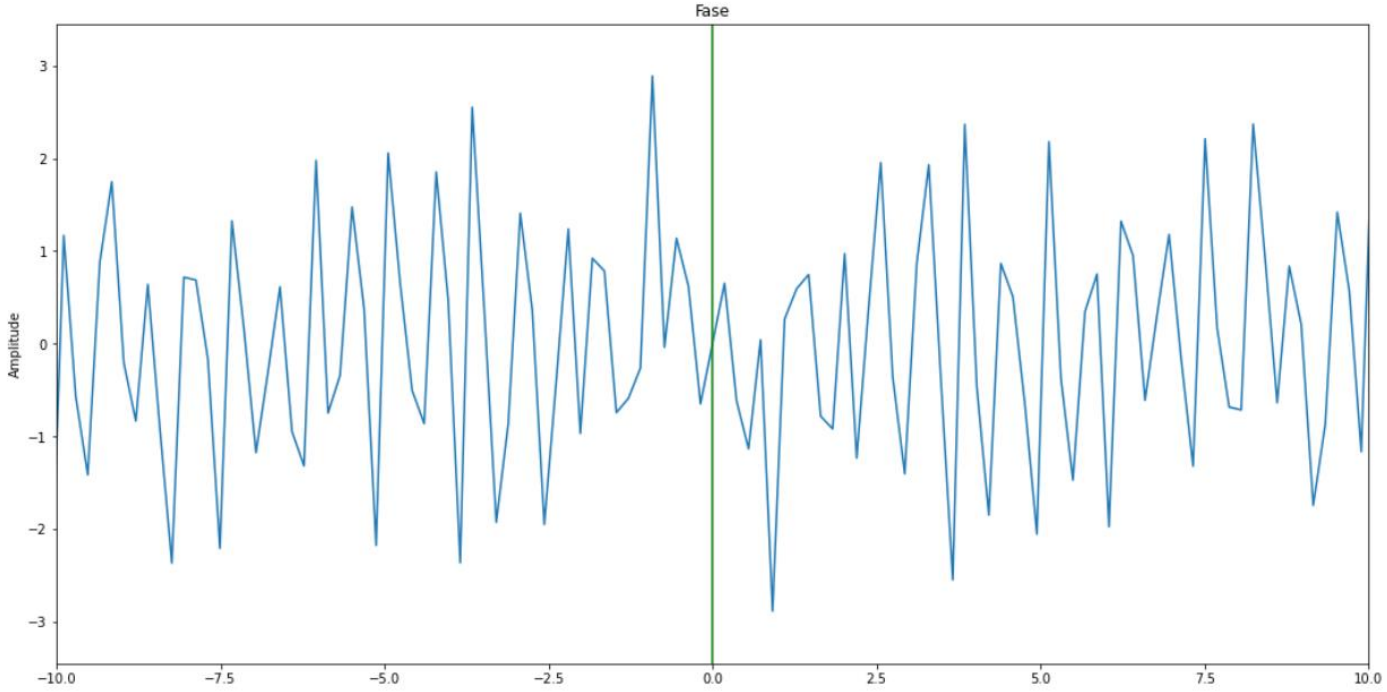

Figura 8 - Fase do sinal na frequência

Agora, para constatar que o nosso sinal manipulado através das transformadas, podemos calcular a sua inversa – da frequência para o tempo – e comparamos graficamente com o sinal original.

```
FFT_inverted = np.real(np.fft.ifft(FFT_2))

plt.figure(figsize=(18,9))
plt.plot(time,FFT_inverted)
plt.title('FFT inversa');
plt.xlabel("Tempo [s]")
plt.ylabel("Amplitude")
plt.show()
```

Figura 9 – [in] Trecho responsável pela inversa da FFT

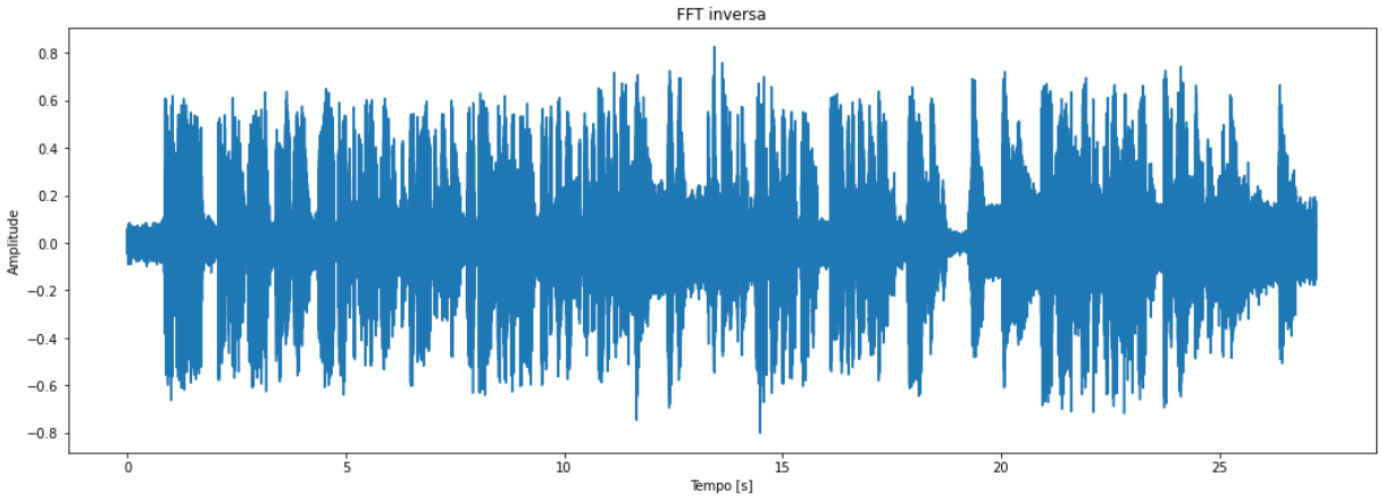

Figura 9 – [out] Sinal inverso

Como podemos notar, temos o mesmo gráfico gerado pela inversa em relação ao gráfico do sinal original mostrado na figura 4. Além disso, o programa gera a saída da inversa em formato de áudio (WAV) o qual é notória a relação entre o sinal original e o manipulado.

Sistema Linear Invariante no Tempo

O sistema escolhido para atuar no nosso sinal foi o efeito de eco. O eco é um tipo de efeito baseado no atraso de um sinal ao longo do tempo, esses atrasos são cópias separadas de um sinal original que voltam a ocorrer um intervalo de milissegundos uma das outras.

Para construir esse sistema em código, for utilizado um laço de repetição a qual preenche um *np.array* com a amostra atual do sinal e uma amostra atrasada multiplicada por uma constante.

Propriedades do sistema:

Com memória: nosso sistema é com memória pois a saída depende de valores passados do sinal de entrada.
Não causal: Para uma entrada -n, o nosso sistema dependerá de valores futuros
Estável: Toda entrada limitada resultará em uma saída limitada
Linear: O nosso sistema cumpre com os princípios de superposição: homogeneidade e aditividade.
Invariante no tempo: Um deslocamento no tempo no sinal de entrada resulta em um deslocamento idêntico no sinal de saída.

Invariância no tempo

$$H\{x[n]\} = y[n] = x[n] + 0.5\,x[n-i]$$

$$H\{x[n-n_0]\} = x[n-n_0] + 0.5\,x[n-n_0-i]$$
$$y[n-n_0] = x[n-n_0] + 0.5\,x[n-n_0-i]$$

Linearidade

Homogeneidade:

$$aH\{x[n]\} = H\{ax[n]\}$$

$$H\{x[n]\} = y[n] = x[n] + 0.5\,x[n-i] \quad i = \text{constante qualquer}$$

$$aH\{x[n]\} = a(x[n] + x[n-i]$$
$$=$$
$$H\{ax[n]\} = ax[n] + ax[n-i]$$

Aditividade:

$$H\{x_1[n] + x_2[n]\}$$
$$x_1[n] + x_2[n] + 0.5\,(x_2[n-i] + x_2[n-i])$$

$$H\{x_1[n]\} + H\{x_2[n]\}$$
$$x_1[n] + 0.5\,x_1[n-i] + x_2[n] + 0.5\,x_2[n-i] =$$
$$x_1[n] + x_2[n] + 0.5\,(x_2[n-i] + x_2[n-i])$$

Criação do sistema LTI

Para criar um efeito de eco em nosso sinal de áudio foi usado o seguinte algoritmo mostrado na figura 10.Nós temos a variável *echo* a qual armazena a cópia do *np.array* do *sinal* a qual tem as informações no tempo do áudio original. A variável *Amplitude_sinal* que também tem as informações das amplitudes do áudio original, e a variável *N* armazena a metade da taxa de amostragem do sinal.
No laço de repetição *-for-* é o local em que é aplicado o efeito de eco no sinal escolhido fazendo com que o *np.array echo* receba o sinal na posição *i* mais meio sinal na posição *i-N* até preencher todo o *np.array* com o tamanho *Amplitude* que é o *np.array* das amostragens do sinal.
Código responsável pelo efeito de eco do sistema:

```
#Criação do sistema LTI - efeito sonoro de eco
echo = sinal.copy()
Amplitude_sinal  = sinal
N = int(amostragem/2)

for i in range(len(Amplitude)):
    echo[i] = Amplitude_sinal[i] + 0.5 * Amplitude_sinal[i-N]
```

Figura 10 – Código responsável pelo efeito de eco do sistema

Após a criação do áudio com o efeito de eco, podemos analisa-ló de forma graficamente, a fim de que as mudanças quanto ao seu espectro no tempo e na frequência.

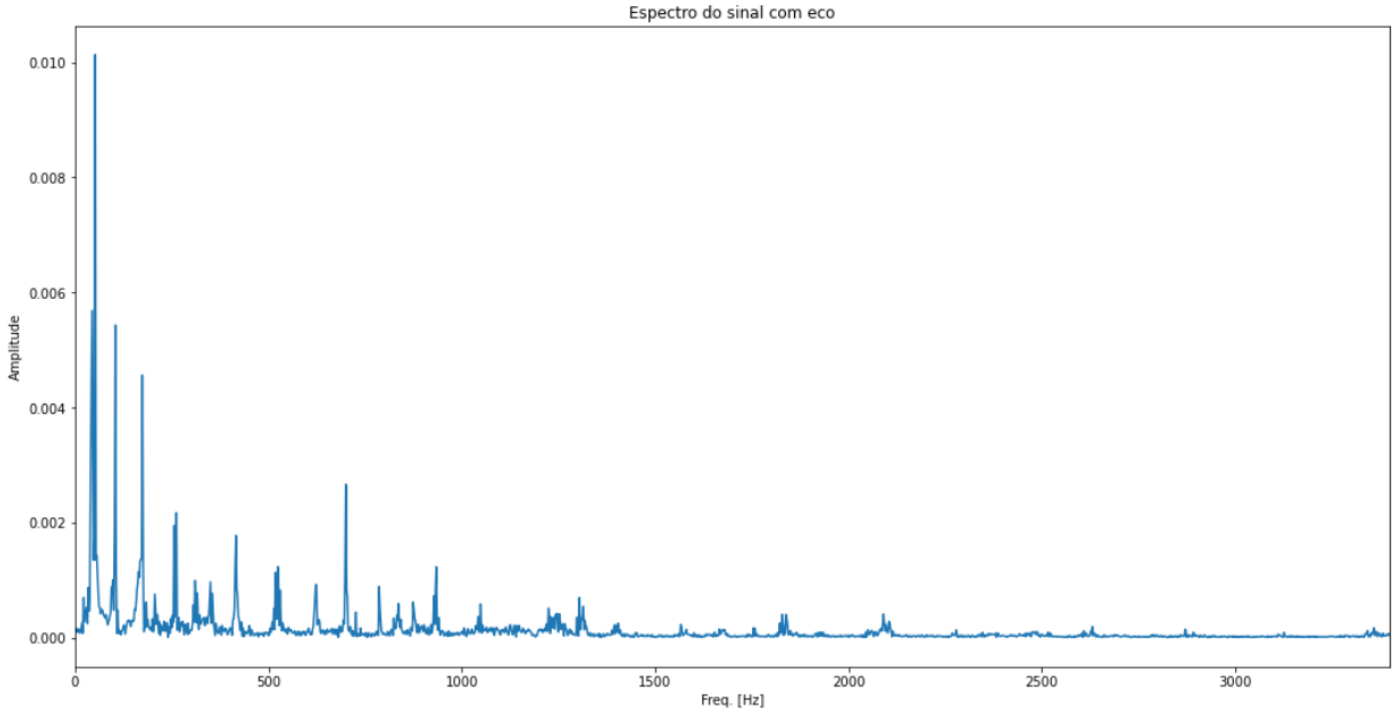

Figura 11 – Espectro do sinal após a passagem pelo sistema LTI

Podemos notar uma semelhança graficamente com a figura **7**, apesar de ter uma queda expressivamente na amplitude. Na figura 12 abaixo, temos a comparação entre o espectro na frequência do sinal original com o gerado a partir do efeito eco.

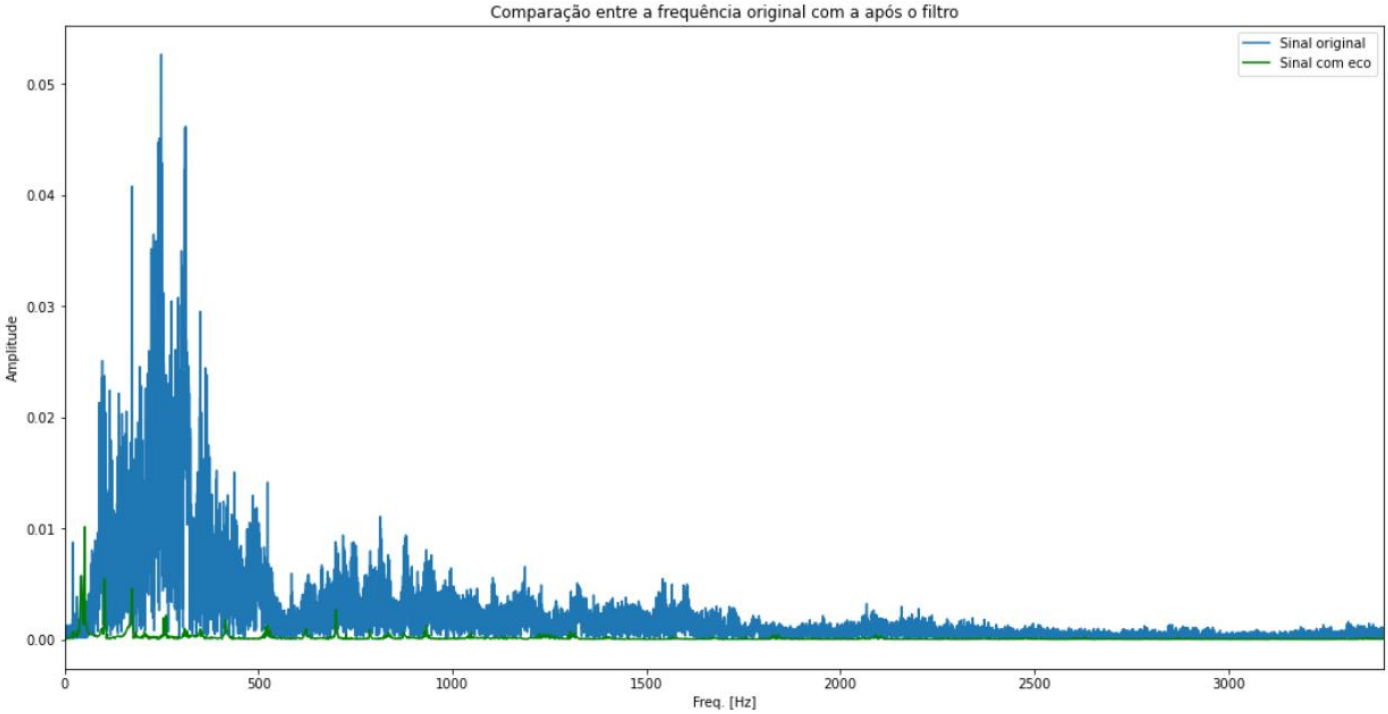

Figura 12 – Comparação do espectro do sinal original e o manipulado pelo sistema LTI

Na fase, temos o seguinte:

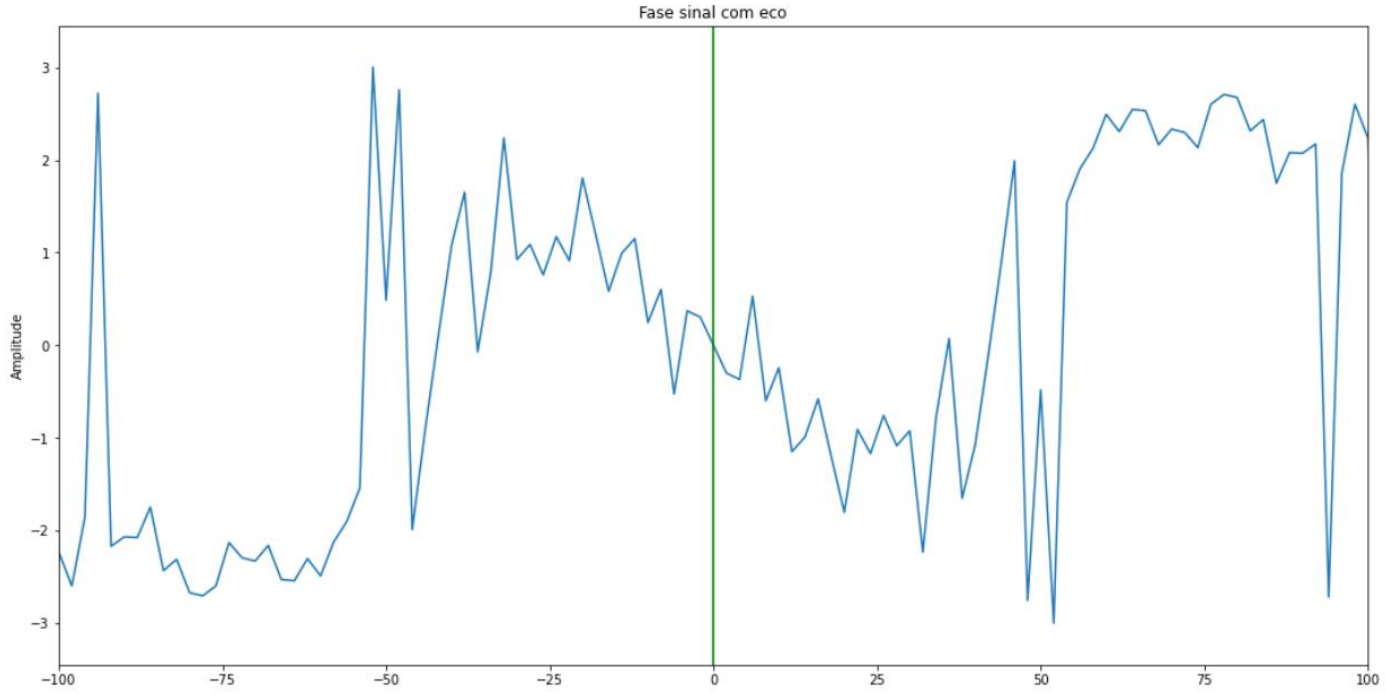

Figura 13 – Fase do sinal após o sistema

A comparação entre o sinal original e o com efeito no tempo é mostrada na figura 14, no qual podemos notar um leve atraso na forma da ontem e uma amplitude um pouco maior.

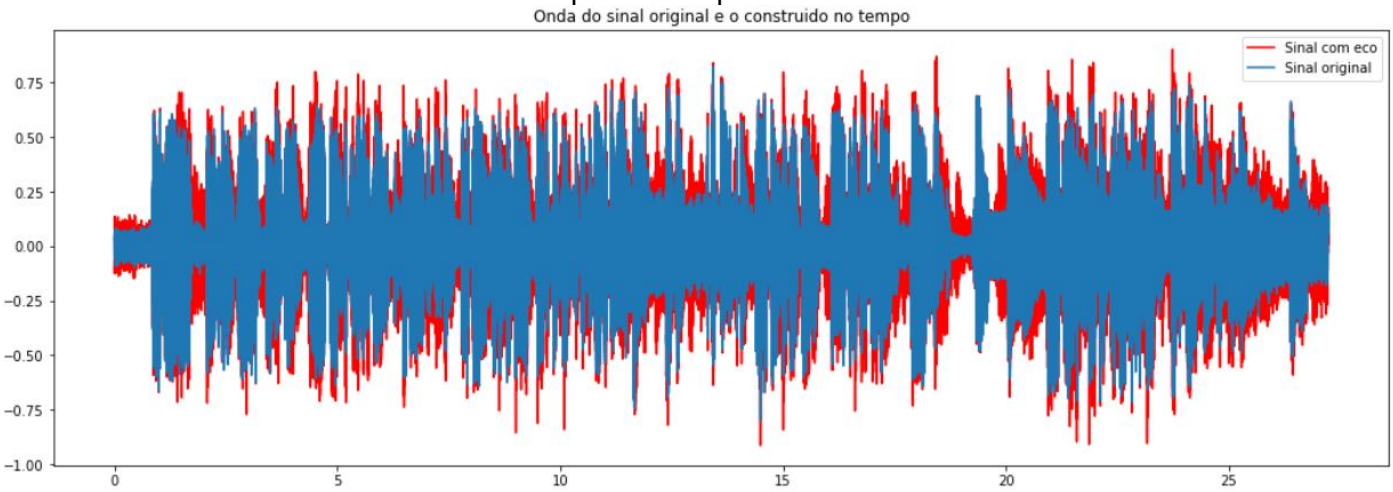

Figura 14 – Comparação da onda do sinal original e o manipulado pelo sistema LTI

Resposta ao impulso do sistema LTI

Primeiramente iremos criar o sinal impulso no tempo com a mesma duração do sinal de áudio a qual criamos o efeito.

```
#Criando o np.array do impulso
n = np.arange(0, len(time))
l = np.size(n)
imp = np.zeros(l)
ind = np.where(n==0)
imp[ind] = 1
```

Figura 15 – Trecho responsável pela criação do impulso unitário

A qual graficamente é expressa como:

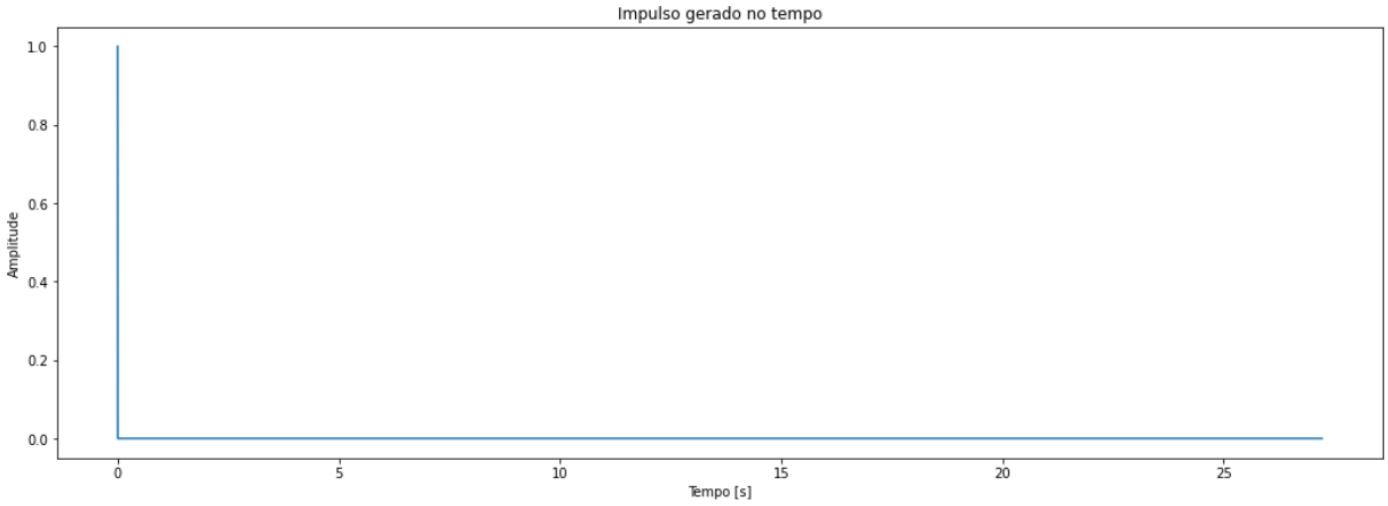

Figura 16 – Impulso representado graficamente no tempo [s]

Temos que inserir esse sinal impulso em nosso sistema para encontrarmos a resposta ao impulso dele. O código está demostrado na figura abaixo.

```
#Passando o impulso gerado no sistema LTI
N = int(amostragem/2)
data_impulse = imp.copy()
Amplitude_impulse  = imp

for i in range(len(imp)):
    data_impulse[i] = Amplitude_impulse[i] + 0.5 * Amplitude_impulse[i-N]
```

Figura 17 – Utilizando o sistema LIT para encontrar a resposta ao impulso do sistema

A saída graficamente é dois impulsos no tempo, o qual é verdadeiro pois o sistema é uma amostra somada um com metade do valor no passado armazenado em um outro *np.array*, e como só temos um pulso em 0 (segundos) de amplitude 1, obteremos dois impulsos como resposta.

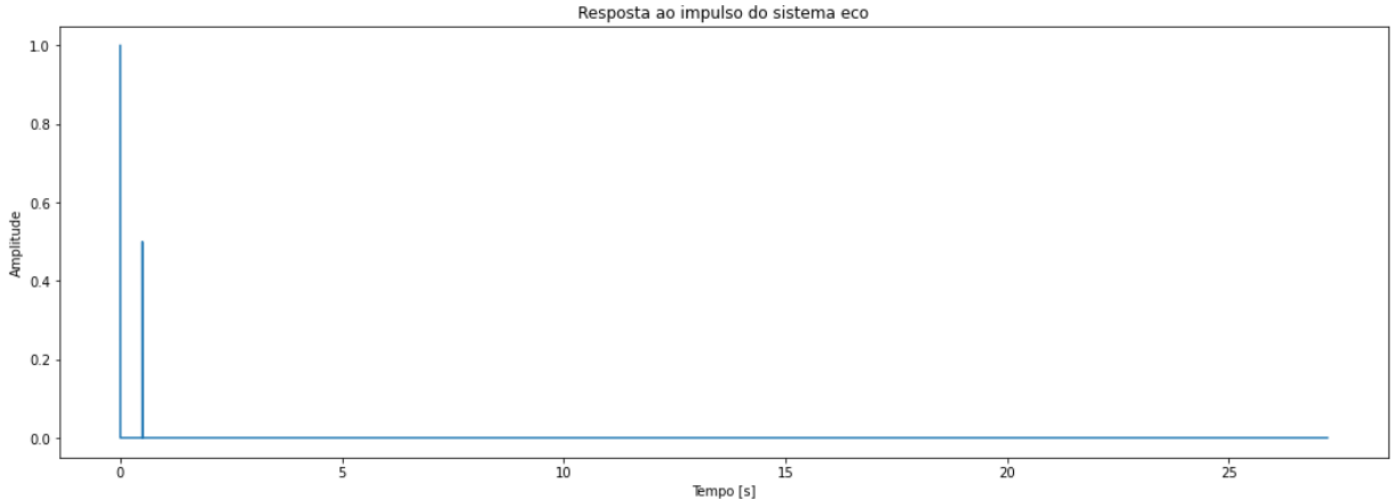

Figura 18 – Resposta ao impulso do sistema LTI

Criando a resposta ao impulso do sistema no domínio da frequência teremos o seguinte trecho de código:

```
#Criando a FFT da resposta ao impulso gerado anteriormente
N = 2**18
Tam = time[1]- time[0]
FFT_abs_impulse = abs(np.fft.fft(data_impulse));
FFT_impulse = np.fft.fft(data_impulse,N) * Tam

w_impulse = np.fft.fftfreq(len(FFT_impulse), d=Tam)
wd_impulse = np.fft.fftshift(w_impulse)
Xd_impulse = np.fft.fftshift(FFT_impulse)
ModX_impulse = np.abs(Xd_impulse)
```

Figura 19 – Trecho responsável por encontrar a resposta do Impulso no domínio da frequência

Pela definição da Transformada de Fourier, o impulso na origem contém todas as frequências com igual amplitude e com fase nula. Como nós temos dois impulsos teremos uma combinação linear dessas duas constantes como consta na figura abaixo.

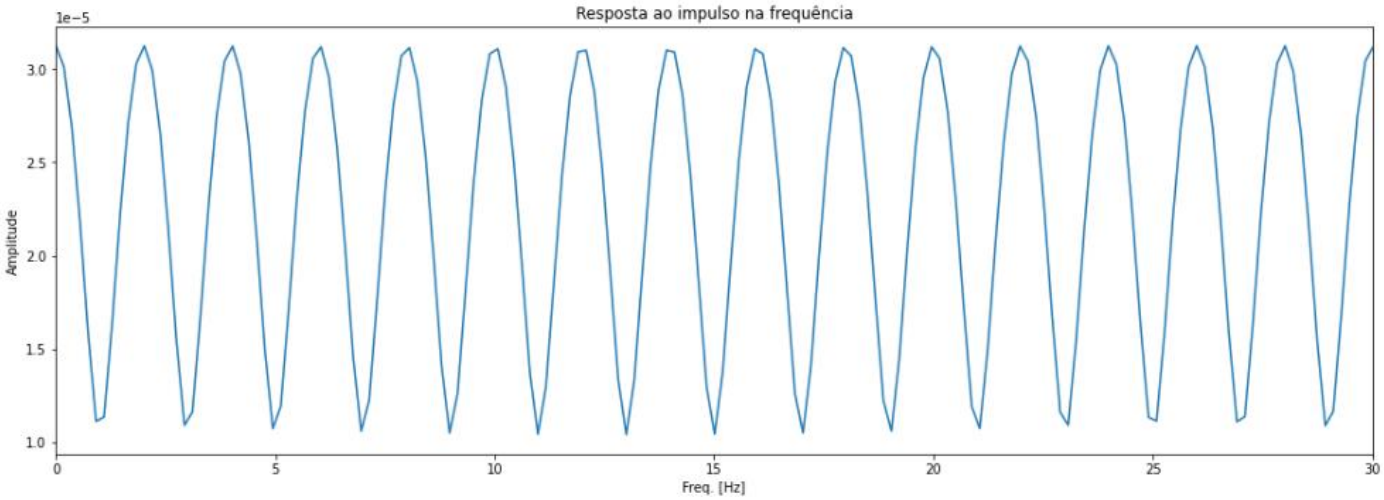

Figura 20 – Resposta ao impulso na frequência representado graficamente

Resposta ao degrau do sistema LTI

Criaremos o sinal degrau da seguinte forma:

```
#Criando o np.array do degrau

n1 = np.arange(0,len(time))
l1 = np.size(n)
degree = np.zeros(l)
d = np.where(n>0)
degree[d] = 1
```

Figura 21 – Trecho responsável pela criação degrau unitário

No qual irá gerar graficamente o seguinte:

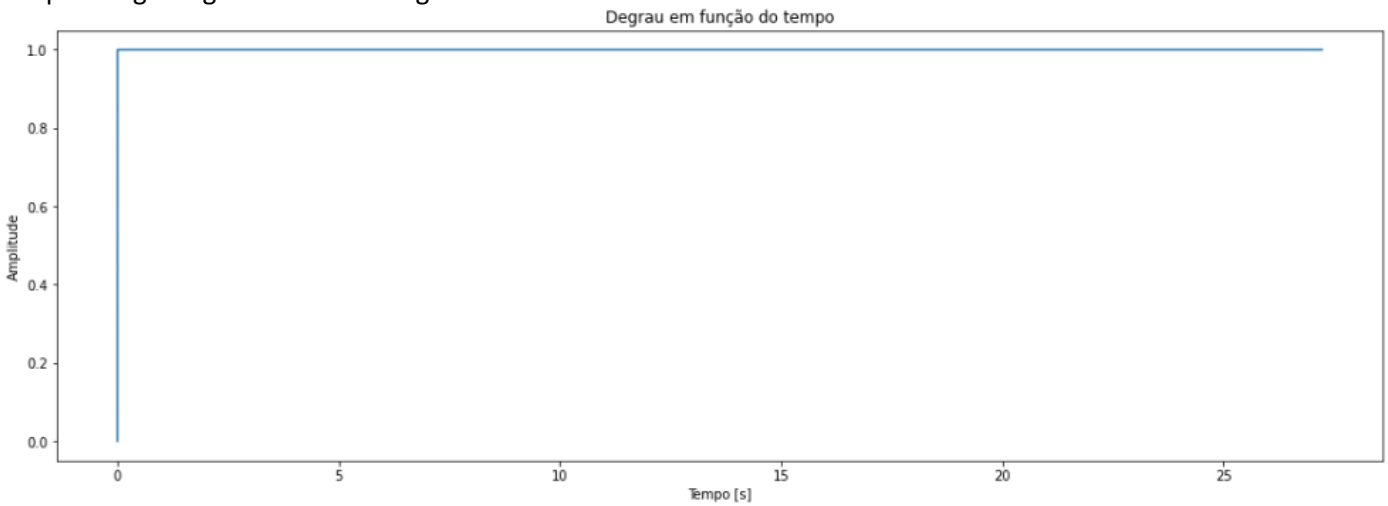

Figura 22 – Degrau representado graficamente no tempo [s]

Após passarmos o degrau unitário em nosso sistema LTI, encontramos a sua resposta ao degrau que é representeada nas duas figuras a seguir.

```
#Passando o degrau gerado no sistema LTI
N = int(amostragem/2)
data_degrau = degree.copy()
Amplitude_degrau  = degree

for i in range(len(degree)):
    data_degrau[i] = Amplitude_degrau[i] + 0.5 * Amplitude_degrau[i-N]
```

Figura 23 – Trecho responsável por encontrar a resposta do Impulso no domínio da frequência

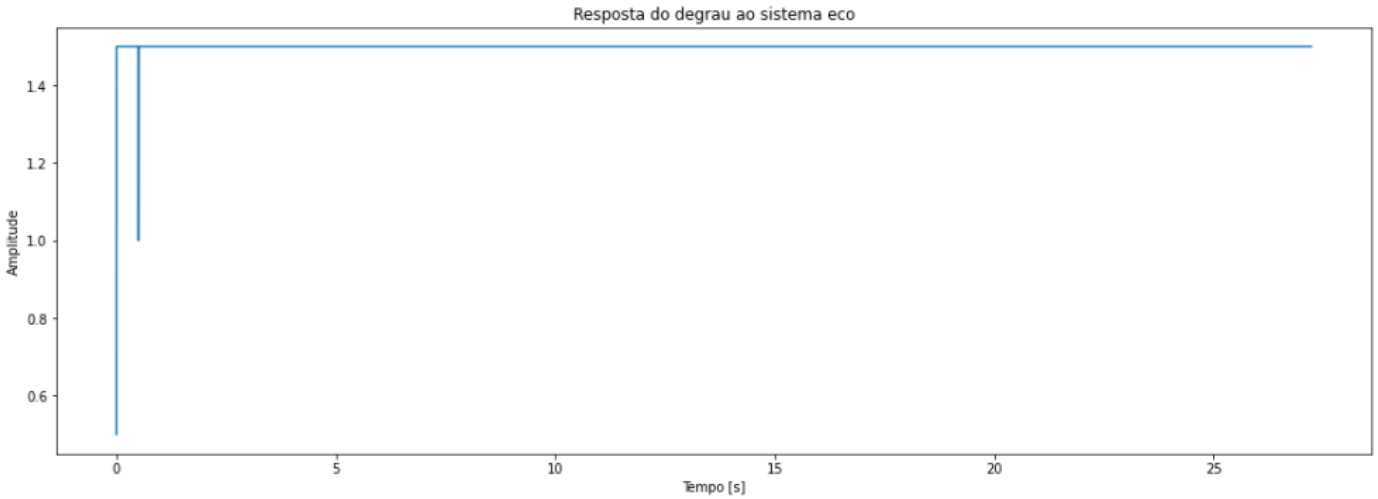

Figura 24 – Resposta ao degrau do sistema LTI

CONCLUSÃO

A amostragem, quantificação e codificação está presente em qualquer sistema eletrônico nos dias atuais logo, foi possível através da linguagem de programação Python ter uma experiência em como esses sinais analógicos são lidos e interpretados por esses dispositivos eletrônicos, além de poder manipula-los.