

Atividade 4

Leonardo Santos - GRR20196154

Primeiramente foi feito a normalização dos dados, utilizando esse trecho de código ilustrado pela Figura 1 a seguir:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.io import loadmat
4
5
6 # Carregar arquivo .mat
7 mat = loadmat('in_out_SMR2_direta.mat')
8
9 in_data_ext = mat['in_extraction']
10 out_data_ext = mat['out_extraction']
11 in_data_val = mat['in_validation']
12 out_data_val = mat['out_validation']
13
14 max_value = np.concatenate((max(abs(in_data_ext)), max(abs(out_data_ext)), max(abs(in_data_val)), max(abs(out_data_val))))
15
16 in_ext = in_data_ext/max_value
17 out_ext = out_data_ext/max_value
18 in_val = in_data_val/max_value
19 out_val = out_data_val/max_value
```

Figura 1: Código de normalização dos dados

Em seguida foi utilizado o trecho de código ilustrado pela Figura 2, para avaliar se os valores se encontravam entre os valores de -1 e 1. O resultado está ilustrado pela Figura 3 a seguir:

```
1 # plotar sinais de entrada e saída
2 x_ext = range(len(in_data_ext))
3 x_val = range(len(in_data_val))
4
5 fig, axs = plt.subplots(2, sharex=True, figsize=(8, 6))
6 fig.suptitle('Sinais de entrada e saída')
7 axs[0].plot(x_ext, np.real(in_ext), label='Entrada')
8 axs[0].plot(x_ext, np.real(out_ext), label='Saída')
9 axs[0].legend()
10 axs[1].plot(x_val, np.real(in_val), label='Entrada')
11 axs[1].plot(x_val, np.real(out_val), label='Saída')
12 axs[1].legend()
13 plt.show()
```

Figura 2: Código pra plotar gráficos de entrada e saída



Figura 3: Gráficos entrada e saída

Com os dados normalizados foi feita a matriz de coeficientes, utilizando o seguinte trecho de código ilustrado pela Figura 4 a seguir:

```

1 def MatrixCoeficientes(in_data, M, P):
2     n = len(in_data)
3     XX = np.zeros((n - M, 2*P), dtype=np.complex128)
4     for i in range(M, n):
5         XX[i-M, 0] = in_data[i]
6         for j in range(1, P):
7             XX[i-M, 2*j-1] = in_data[i-j].real ** j
8             XX[i-M, 2*j] = in_data[i-j].imag ** j
9     return XX
10
11 P = 5
12 M = 1
13
14 XX_ext = MatrixCoeficientes(in_ext, M, P)
15 XX_val = MatrixCoeficientes(in_val, M, P)
16
17 coefficients, _, _ = np.linalg.lstsq(XX_ext, out_ext[M:], rcond=None)

```

Figura 4: Calculo das matrizes de Coeficientes

Em seguida foram feitos os cálculos utilizando os valores do extraído e normalizados que correspondem a -21dB.

Em seguida foram realizados os passos do item 3, na etapa do item a) observou-se que o maior valor, depois da conversão para um número inteiro foi de 15, enquanto para a próxima etapa do item b) na etapa de readequação o maior valor foi arredondado para 0. A figura Figura 5 ilustra o código utilizado:

```

1 bits = 4
2 multiplicador = 2**bits
3 precisão = 1/multiplicador
4 in_val_fix = np.round(in_val * multiplicador)
5 out_val_fix = np.round(out_val * multiplicador)
6 in_val_fix_imag = np.floor(in_val_fix.imag/multiplicador)
7 in_val_fix_real = np.floor(in_val_fix.real/multiplicador)
8 out_val_fix_imag = np.floor(out_val_fix.imag/multiplicador)
9 out_val_fix_real = np.floor(out_val_fix.real/multiplicador)
10
11
12 in_val_real_float = in_val_fix_real / multiplicador
13 in_val_imag_float = in_val_fix_imag / multiplicador
14
15
16 out_val_real_float = out_val_fix_real / multiplicador
17 out_val_imag_float = out_val_fix_imag / multiplicador
18
19 out_val = out_val_real_float + 1j*out_val_imag_float
20
21 in_val = in_val_real_float + 1j*in_val_imag_float

```

Figura 5: Calculo da precisão em bits

Estou com dificuldade nessa etapa!