

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_SBR2_direto.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 mod_in_data_ext = max(abs(in_data_ext))
15 mod_out_data_ext = max(abs(out_data_ext))
16 mod_in_data_val = max(abs(in_data_val))
17 mod_out_data_val = max(abs(out_data_val))
18
19 in_ext = in_data_ext/mod_in_data_ext
20 out_ext = out_data_ext/mod_out_data_ext
21 in_val = in_data_val/mod_in_data_val
22 out_val = out_data_val/mod_out_data_val
23
```

Figura 1: Código de normalização dos dados

Em seguida foi avaliado se eles se encontravam entre os valores de -1 e 1. o resultado esta ilustrado pela Figura 2 a seguir:

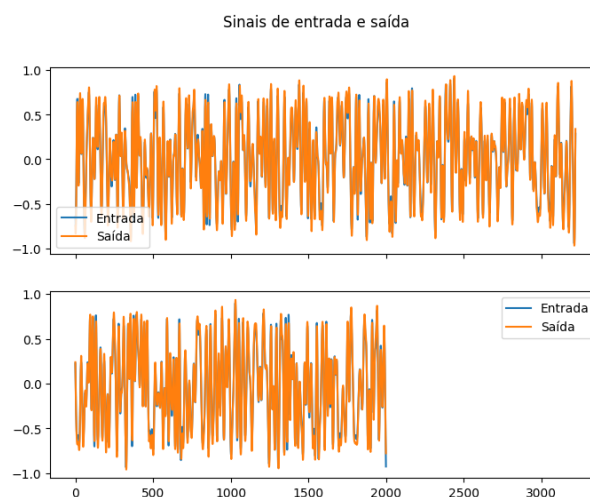


Figura 2: Gráficos entrada e saída

Com os dados normalizados foi feito a matriz de coeficientes, utilizando o seguinte trecho de código ilustrado pela Figura 3 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)

```

Figura 3: Calculo das matrizes de Coeficientes

Em seguida foi realizados os passos do item 3 porem os valores de maximo me mino deram menores que 1. O codigo utilizado esta ilustrado pela Figura 4 a seguir:

```

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 4: Calculo da precisão em bits