Atividade 4

Leonardo Santos - GRR20196154

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

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat


# carregar arquivo .mat
mat = loadmat('in_out_SBRT2_direto.mat')

in_data_ext = mat['in_extraction']
out_data_ext = mat['out_extraction']
in_data_val = mat['in_validation']
out_data_val = mat['out_validation']

max_value = np.concatenate((max(abs(in_data_ext)), max(abs(out_data_ext)),max(abs(in_data_val)),max(abs(out_data_val)))).max()


in_ext = in_data_ext/max_value
out_ext = out_data_ext/max_value
in_val = in_data_val/max_value
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 osvalores se encotravam entre os valores de -1 e 1. O resultado esta ilustrado pela Figura 3 a seguir:

```python
# plotar sinais de entrada e saída
x_ext = range(len(in_data_ext))
x_val = range(len(in_data_val))

fig, axs = plt.subplots(2, sharex=True, figsize=(8, 6))
fig.suptitle('Sinais de entrada e saída')
axs[0].plot(x_ext, np.real(in_ext), label='Entrada')
axs[0].plot(x_ext, np.real(out_ext), label='Saída')
axs[0].legend()
axs[1].plot(x_val, np.real(in_val), label='Entrada')
axs[1].plot(x_val, np.real(out_val), label='Saída')
axs[1].legend()
plt.show()
```

Figura 2: Código pra plotar gráficos de entrada e saída

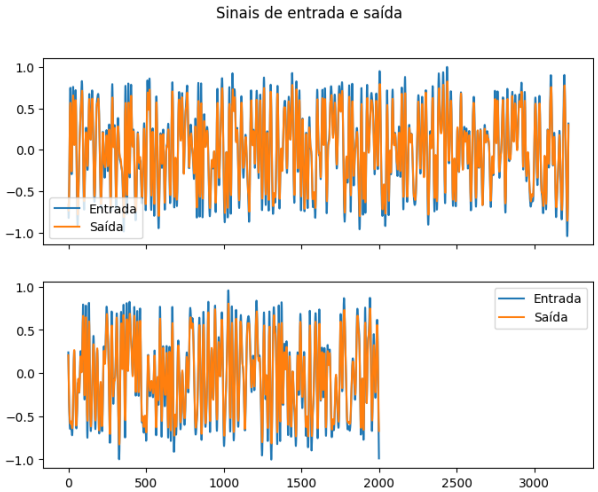


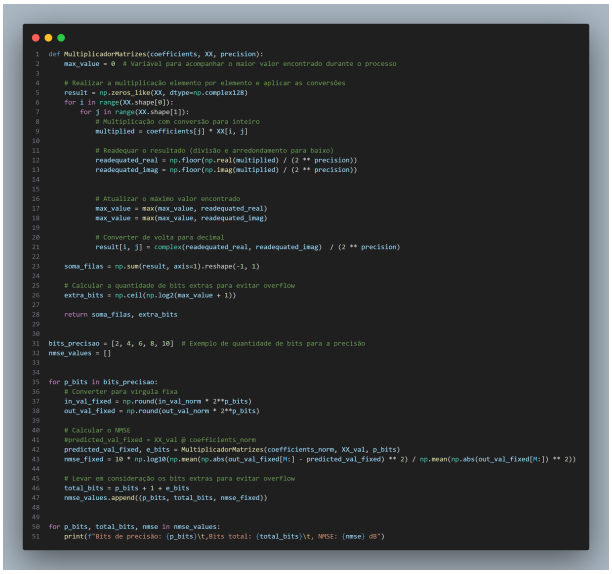
Figura 3: Gráficos entrada e saída

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

```python
import numpy as np

def mp(P, M, xn):
    L = xn.shape
    XX = np.zeros((L[0] - M, P * (M+1)), dtype=np.complex128)
    for l in range(M+1, L[0]):
        for p in range(1, P+1):
            for m in range(0, M+1):
                part_real = np.floor(np.abs(xn[l-m]**(2*p-2))*np.real(xn[l-m]))
                part_imag = np.floor(np.abs(xn[l-m]**(2*p-2))*np.imag(xn[l-m]))
                XX[l-M-1, ((p-1)*(M+1))+m] = complex (part_real, part_imag)
    return XX

P = 5
M = 1

XX_ext_norm = mp(P, M, in_ext_norm)
coefficients_norm, _, _, _ = np.linalg.lstsq(XX_ext_norm, out_ext_norm[M:], rcond=None)

XX_val = mp(P, M, in_val_norm)
```

Figura 4: Calculo das matrizes de Coeficientes

Em seguida foi feito o calculo da adequeação e da multiplicação de matrizes utilizando esses trechos de código:



```python
def MultiplicadorMatrizes(coefficients, XX, precision):
    max_value = 0  # Variável para acompanhar o maior valor encontrado durante o processo

    # Realizar a multiplicação elemento por elemento e aplicar as conversões
    result = np.zeros_like(XX, dtype=np.complex128)
    for i in range(XX.shape[0]):
        for j in range(XX.shape[1]):
            # Multiplicação com conversão para inteiro
            multiplied = coefficients[j] * XX[i, j]

            # Readequar o resultado (divisão e arredondamento para baixo)
            readequated_real = np.floor(np.real(multiplied) / (2 ** precision))
            readequated_imag = np.floor(np.imag(multiplied) / (2 ** precision))


            # Atualizar o máximo valor encontrado
            max_value = max(max_value, readequated_real)
            max_value = max(max_value, readequated_imag)

            # Converter de volta para decimal
            result[i, j] = complex(readequated_real, readequated_imag)  / (2 ** precision)

    soma_filas = np.sum(result, axis=1).reshape(-1, 1)

    # Calcular a quantidade de bits extras para evitar overflow
    extra_bits = np.ceil(np.log2(max_value + 1))

    return soma_filas, extra_bits


bits_precisao = [2, 4, 6, 8, 10]  # Exemplo de quantidade de bits para a precisão
nmse_values = []


for p_bits in bits_precisao:
    # Converter para vírgula fixa
    in_val_fixed = np.round(in_val_norm * 2**p_bits)
    out_val_fixed = np.round(out_val_norm * 2**p_bits)

    # Calcular o NMSE
    #predicted_val_fixed = XX_val @ coefficients_norm
    predicted_val_fixed, e_bits = MultiplicadorMatrizes(coefficients_norm, XX_val, p_bits)
    nmse_fixed = 10 * np.log10(np.mean(np.abs(out_val_fixed[M:] - predicted_val_fixed) ** 2) / np.mean(np.abs(out_val_fixed[M:]) ** 2))

    # Levar em consideração os bits extras para evitar overflow
    total_bits = p_bits + 1 + e_bits
    nmse_values.append((p_bits, total_bits, nmse_fixed))


for p_bits, total_bits, nmse in nmse_values:
    print(f"Bits de precisão: {p_bits}\t,Bits total: {total_bits}\t, NMSE: {nmse} dB")
```

Figura 5: Calculo da precisão em bits

Porém o resultado obtido não foi o esperado!