



Modelo matemático para amplificador

André Corso Pozzan
Atividade 2 - Iniciação Científica **GICS**

Orientador: Prof. Ph.D. Eduardo Gonçalves de Lima

20 de fevereiro de 2025

1 Atividade:

1.1 Enunciado

Exercício: Considere um conjunto de dados de entrada e saída de um amplificador.

$$\text{in} = \begin{bmatrix} \text{in}(0) \\ \text{in}(1) \\ \text{in}(2) \\ \vdots \\ \text{in}(99) \end{bmatrix} \quad \text{out} = \begin{bmatrix} \text{out}(0) \\ \text{out}(1) \\ \text{out}(2) \\ \vdots \\ \text{out}(99) \end{bmatrix}$$

1.2 Objetivo

Deseja-se criar um modelo matemático para o amplificador. O modelo escolhido é o polinômio com memória, um caso particular da série de Volterra.

$$d(n) = \text{out}(n) = \sum_{p=1}^P \sum_{m=0}^M h_p(m) [\text{in}(n-m)]^p = X(n)^T H(n)$$

Onde:

- P : é a ordem polinomial.
- M : é a duração da memória.

1.3 Exemplo:

Para $P = 5$ e $M = 1$ temos:

$$H(n) = \begin{bmatrix} h_1(0) \\ h_1(1) \\ h_2(0) \\ h_2(1) \\ h_3(0) \\ h_3(1) \\ h_4(0) \\ h_4(1) \\ h_5(0) \\ h_5(1) \end{bmatrix} \quad X(n) = \begin{bmatrix} \text{in}(n) \\ \text{in}(n-1) \\ \text{in}^2(n) \\ \text{in}^2(n-1) \\ \text{in}^3(n) \\ \text{in}^3(n-1) \\ \text{in}^4(n) \\ \text{in}^4(n-1) \\ \text{in}^5(n) \\ \text{in}^5(n-1) \end{bmatrix}$$

Escolher os coeficientes H que gerem o menor erro.

2 Resolução

Inicialmente, realizei a análise da equação por meio de cálculos manuais, desenvolvendo os primeiros passos do somatório. Além disso, como complemento, comparei os resultados com a atividade anterior sobre mínimos quadrados.

Essa abordagem permitiu uma melhor compreensão do problema e serviu de base para a implementação do código em Python.

2.1 Cálculos no papel

EQ. de Volterra pl amplificada:

(in) \rightsquigarrow (modelo) \rightsquigarrow (out)

Concentrar os coeficientes na saída

matriz dos coeficientes a ser encontrada

$\text{out}(m) = \sum_{p=1}^P \sum_{m=0}^M h_p(m) [\text{in}(m-m)]^p = X(m)^T H(m)$

para $P=2$ e $M=1$

$\text{out}(1) = h_1(0).[\text{in}(1)]^1 + h_1(1)[\text{in}(0)]^1 + h_2(0).[\text{in}(1)]^2 + h_2(1)[\text{in}(0)]^2$

$\text{out}(2) = h_1(0).[\text{in}(2)]^1 + h_1(1)[\text{in}(1)]^1 + h_2(0).[\text{in}(2)]^2 + h_2(1)[\text{in}(1)]^2$

Matriz de regressão

Matriz dos coeficientes

Minimos quadrados

$y = ax + b$

$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ x_4 & 1 \\ x_5 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix}$

Matriz de regressão

Matriz dos coeficientes

Saídas

Comparação entre os resultados

Matriz de regressão

Matriz dos coeficientes

Saídas

Também explorei algumas aplicações descritas nos vídeos (POSSANI, 2023) e (DRONGELEN, 2015), bem como na dissertação (BONFIM, 2016).

2.2 Código em Python

Para o código em Python, criei as funções `criaMatrizDeRegressao()`, responsável por criar a matriz de regressão no formato e organização necessária para ser aplicada na função `np.linalg.lstsq`. A função `gerarOndaDeEntrada()`, simula uma onda com três frequências para testar o modelo matemático posteriormente.

Dessa forma, foram definidas as constantes `P_VOLTERRA` e `M_VOLTERRA` que representam a ordem polinomial e a memória a ser aplicada na Equação de Volterra. Por fim, o cálculo dos coeficientes foi realizado pela função `np.linalg.lstsq` que também havia sido utilizada na atividade anterior

```

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

data = loadmat('IN_OUT_PA.mat')

in_data = data['in']
out_data = data['out']

P_VOLTERRA = 10
M_VOLTERRA = 2

in_data = np.array(in_data)

def criaMatrizDeRegressao(in_data, P_VOLTERRA, M_VOLTERRA):
    matrizDeRegressao = []

    for n in range(len(in_data)):
        linhaDaMatrizDeRegressao = []

        for p in range(1, P_VOLTERRA + 1): # Incluir termos até a ordem P_VOLTERRA
            for m in range(M_VOLTERRA + 1): # Considerar atrasos até M_VOLTERRA
                dataIndex = n - m
                if dataIndex >= 0: # Verificar se o índice é válido
                    linhaDaMatrizDeRegressao.append(in_data[dataIndex, 0] ** p)
                else:
                    linhaDaMatrizDeRegressao.append(0) # Adicionar zero se o índice for inválido
            matrizDeRegressao.append(linhaDaMatrizDeRegressao)

    return np.array(matrizDeRegressao);

matrizDeRegressao = criaMatrizDeRegressao(in_data, P_VOLTERRA, M_VOLTERRA)

resultado = np.linalg.lstsq(matrizDeRegressao, out_data)

coeficientes = resultado[0]

print("Coeficientes:", coeficientes)

def gerarOndaDeEntrada(fs, time, freq1, freq2, freq3):
    t = np.linspace(0, time, fs)

    # Onda composta por múltiplas frequências
    ondaDeEntrada = 0.7 * np.sin(2 * np.pi * freq1 * t) + 0.5 * np.sin(2 * np.pi * freq2 * t) + 0.3 * np.sin(2
    ↵ * np.pi * freq3 * t)

```

```

# Normalizando o sinal entre -1 e 1
ondaDeEntrada = ondaDeEntrada / np.max(np.abs(ondaDeEntrada))
ondaDeEntrada = ondaDeEntrada.reshape(-1, 1)

return ondaDeEntrada


#Algumas ondas de exemplo
ondaDeEntrada = gerarOndaDeEntrada(100,0.065,5,10,200)
#ondaDeEntrada = gerarOndaDeEntrada(100,0.082,5,20,500)
#ondaDeEntrada = gerarOndaDeEntrada(100, 0.04, 10, 15, 100)
#ondaDeEntrada = gerarOndaDeEntrada(100,0.04,10,80,150)
#ondaDeEntrada = gerarOndaDeEntrada(100, 0.053, 7, 30, 250)

matrizDeRegressaoTeste = criaMatrizDeRegressao(ondaDeEntrada, P_VOLTERRA, M_VOLTERRA);

saídaEstimadaDaOnda = np.dot(matrizDeRegressaoTeste, coeficientes);

saídaEstimadaVolterra = np.dot(matrizDeRegressao, coeficientes)

fig, axs = plt.subplots(2, 1, figsize=(10, 8))

# Primeiro gráfico
axs[0].plot(in_data, label='Entrada (in_data)')
axs[0].plot(out_data, label='Saída (out_data)')
axs[0].plot(saídaEstimadaVolterra, label='Estimado (saídaEstimadaVolterra)', linestyle='--')
axs[0].set_xlabel('Amostras')
axs[0].set_ylabel('Amplitude')
axs[0].set_title(f"Comparação dos Dados de Entrada, Saída e Estimado para P = {P_VOLTERRA} M = {M_VOLTERRA}")
axs[0].legend()
axs[0].grid(True)

# Segundo gráfico
axs[1].plot(ondaDeEntrada, label='Entrada (ondaDeEntrada)')
axs[1].plot(saídaEstimadaDaOnda, label='Estimado Teste (saídaEstimadaDaOnda)', linestyle='--')
axs[1].set_xlabel('Amostras')
axs[1].set_ylabel('Amplitude')
axs[1].set_title(f"Comparação da onda simulada para a entrada e o comportamento de saída, para P = {P_VOLTERRA} M = {M_VOLTERRA}")
axs[1].legend()
axs[1].grid(True)

plt.tight_layout()
plt.show()

```

2.3 Resultado do código e possível utilização

Depois de obter os coeficientes com a equação de Volterra gerei algumas simulações de ondas para aplicá-las ao modelo e verificar qual seria o comportamento da saída. Dessa forma, caso sejam obtidos os dados de um amplificador seria possível estipular seu comportamento em outras situações apenas com o modelo matemático.

Ao mudar os valores de P e M no somatório podemos observar diferenças na precisão do modelo, conforme imagens abaixo:

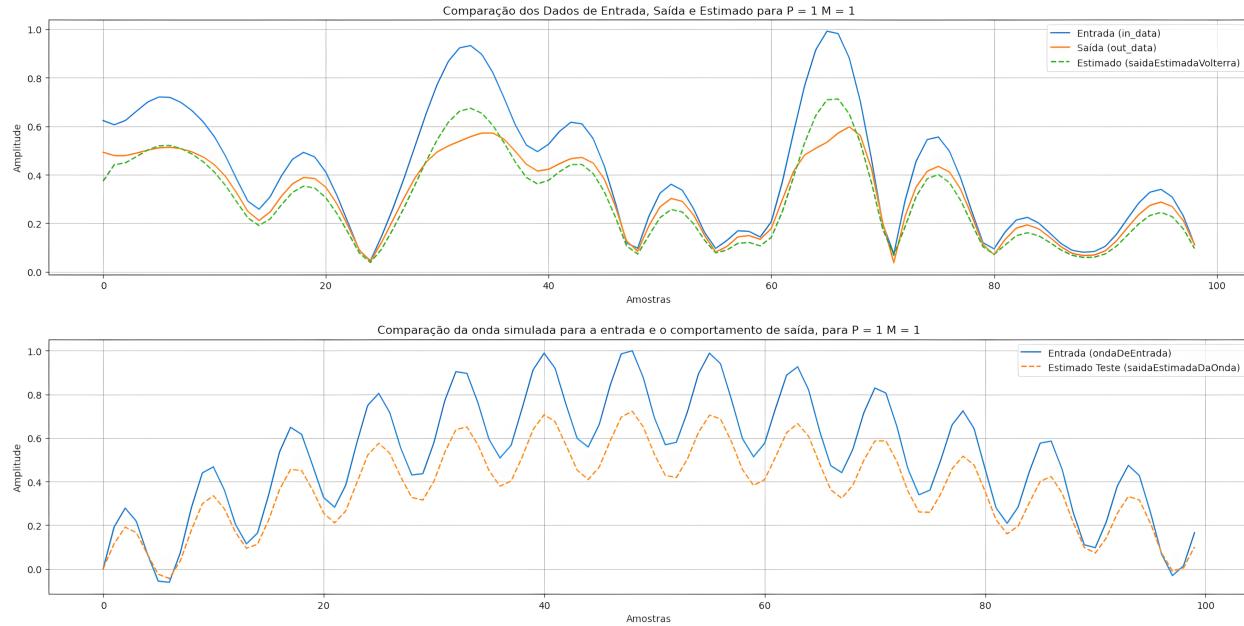


Figura 1: Resultado para $P = 1$ e $M = 1$

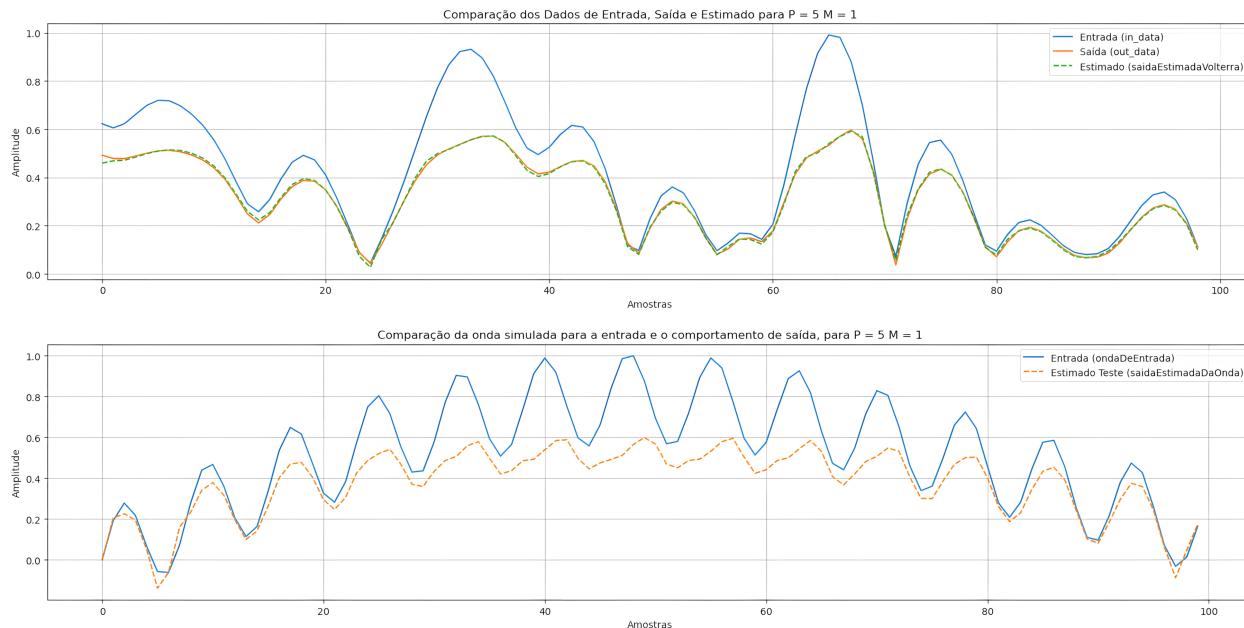


Figura 2: Resultado para $P = 5$ e $M = 1$

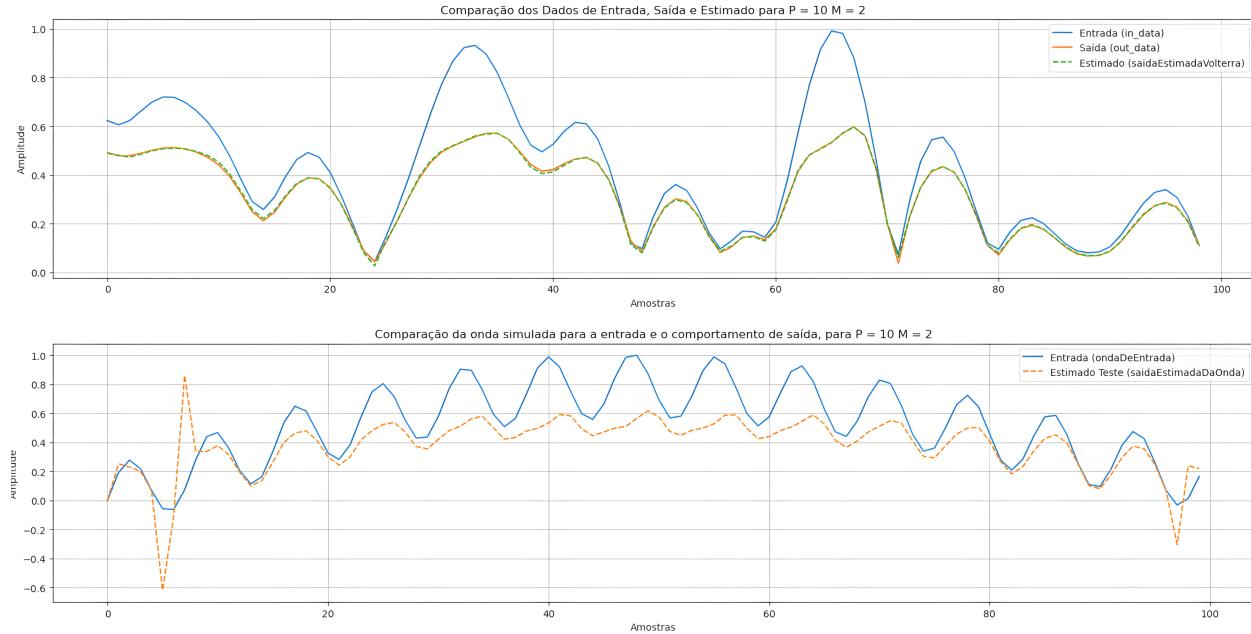
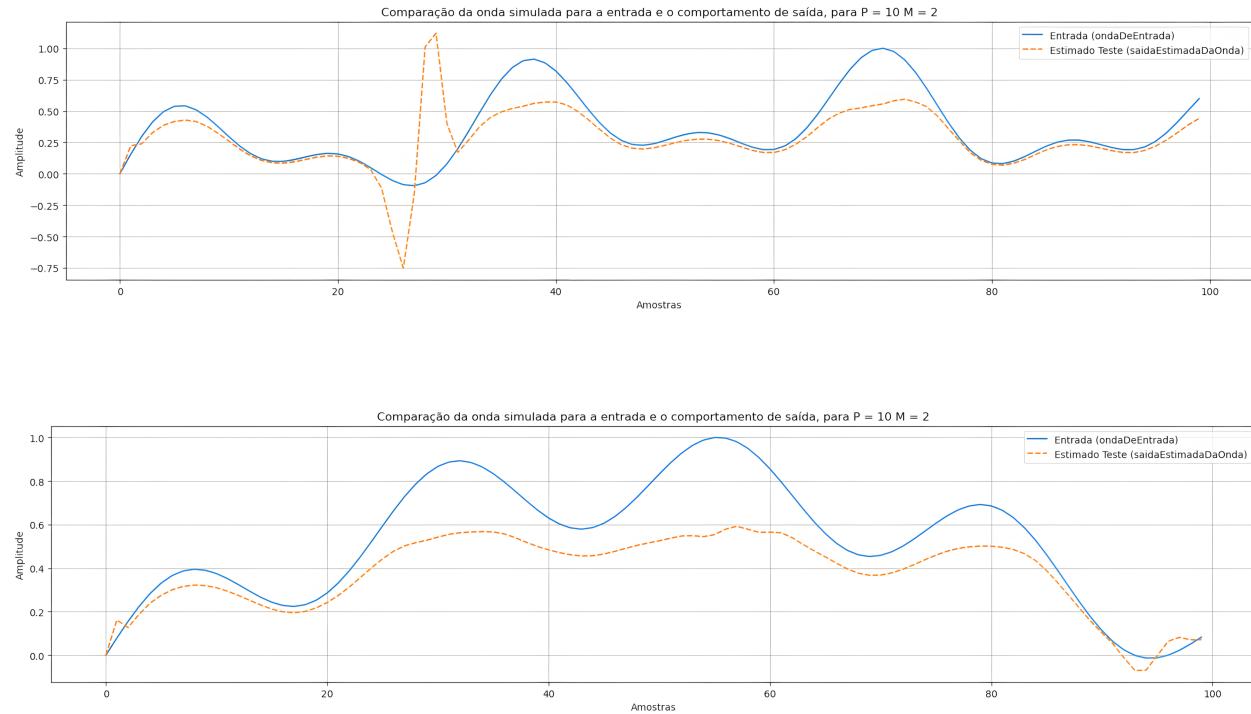
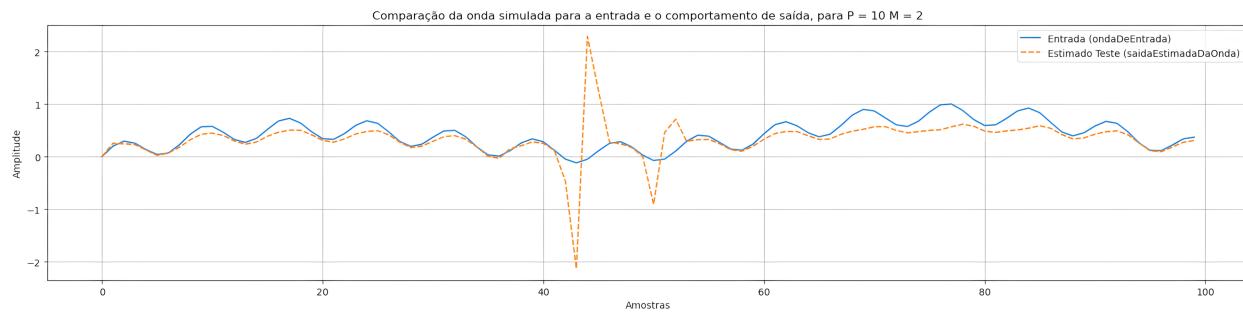
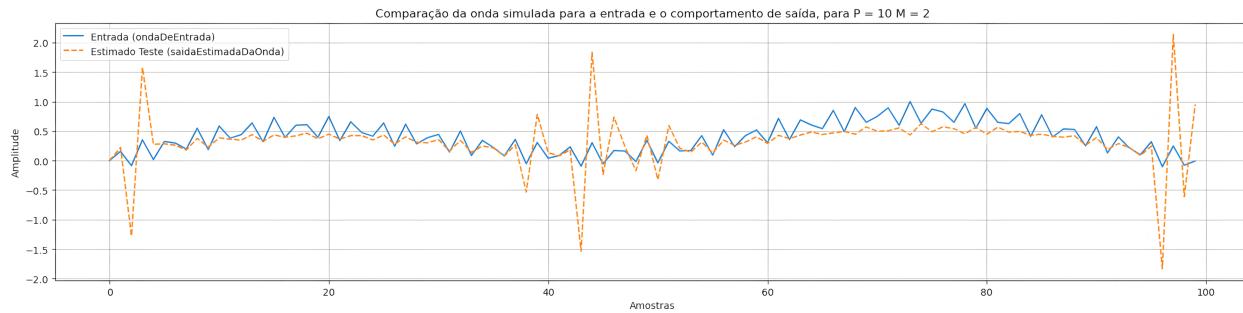


Figura 3: Resultado para $P = 10$ e $M = 2$

2.3.1 Simulações de ondas de entrada

Estes são os gráficos das simulações de algumas ondas diferentes que apliquei ao modelo para observar a saída:





Referências

BONFIM, Elton John. **Modelagem Comportamental de Amplificadores de Potência de Radiofrequência Usando Termos Unidimensionais e Bidimensionais de Séries de Volterra.** 2016. Diss. (Mestrado) – Universidade Federal do Paraná, Curitiba. Disponível em:
https://bdtd.ibict.br/vufind/Record/UFPR_54e3f81abd9cae4e13508c3e91186f38.

DRONGELEN, Wim van. **Lecture 14: Volterra Series, Dr. Wim van Drongelen, Modeling and Signal Analysis for Neuroscientists.** 2015. Disponível em:
<https://www.youtube.com/watch?v=sTN9FGFJ0Yw>.

POSSANI. **Equações de Volterra-Lotka (Aula 05).** 2023. Disponível em:
<https://www.youtube.com/watch?v=6Lyl4mvre1k>.

SIMPLIFICADA, Matemática. **Equações Integrais de Volterra: Como Resolver Usando Transformada de Laplace e a Convolução?** 2022. Disponível em:
<https://www.youtube.com/watch?v=srDd1fy6RC8>.