



DEPARTAMENTO DE ENGENHARIA ELÉTRICA
UNIVERSIDADE FEDERAL DO PARANÁ

Modelo Matemático com Otimização Não Linear

André Corso Pozzan

Atividade 4 – Iniciação Científica **GICS**

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

Conteúdo

1	Atividade	2
1.1	Etapa A: Construção de uma Função	2
1.2	Etapa B: Otimização Não Linear	2
1.2.1	Dicas para Implementação	2
2	Resolução	3
2.1	Código em python	3
2.1.1	Definições iniciais	3
2.1.2	Matriz de regressão	4
2.1.3	Função de resíduos	4
2.1.4	Calculo dos coeficientes	5
2.1.5	Calculo do vetor de erro	5
2.1.6	Exibição do vetor de erro	6
2.1.7	Saída gráfica com os resultados	7
2.1.8	Código completo	7
3	Resultados	8
3.1	Estimativa inicial com zero, $P = 2$ e $M = 1$	8
3.2	Estimativa inicial com zero, $P = 5$ e $M = 2$	10
3.3	Estimativa inicial com um, $P = 5$ e $M = 2$	11
3.4	Estimativa inicial com números aleatórios, $P = 5$ e $M = 2$	12
3.5	Estimativa inicial com um, $P = 10$, $M = 5$	13
4	Extra	17
4.1	P até 5 e M até 3	17
4.2	P até 15 e M até 5	18
4.3	Código final	20

1 Atividade

Resolver novamente o exercício da Atividade 2, porém, ao invés de identificar os coeficientes usando otimização linear, usar otimização não linear. A Atividade 3 será dividida em duas etapas:

1.1 Etapa A: Construção de uma Função

A função desenvolvida deve ter as seguintes características:

- **Argumentos (incógnitas) da função:** os valores de todos os coeficientes do polinômio de memória (MP).
- **Processamentos internos dentro da função:**
 - Carregar os sinais de treinamento (entrada e saída);
 - Processar o sinal de entrada ao longo do MP (observe que dentro dessa função assume-se que os coeficientes são conhecidos, uma vez que esses valores são passados como argumentos);
 - Obter a saída estimada pelo MP e calcular o vetor de erro entre a saída desejada e a saída estimada pela rede.
- **Retorno da função:** vetor de erro.

1.2 Etapa B: Otimização Não Linear

Para encontrar simultaneamente os coeficientes do polinômio de memória (MP), utilizaremos otimização não linear. No MATLAB, essa tarefa pode ser realizada com o comando `lsqnonlin`, disponível no *Optimization Toolbox*. No Python, existe um equivalente que pode ser explorado.

1.2.1 Dicas para Implementação

- Consulte um exemplo no `help lsqnonlin`;
- Utilize a função desenvolvida na Etapa A dentro do comando `lsqnonlin`;
- Defina uma estimativa inicial para os coeficientes do MP, testando diferentes condições:
 - Todos os coeficientes iniciam como zero;
 - Todos os coeficientes iniciam como um;
 - Valores iniciais aleatórios.
- Durante a execução do `lsqnonlin`, os coeficientes são atualizados iterativamente;
- Para visualizar a evolução do erro quadrático médio (MSE) a cada iteração, utilize `Display "iter"` dentro de `optimoptions`;
- Para mais opções de configuração do otimizador, consulte `help optimoptions`.

2 Resolução

Inicialmente, pesquisei na biblioteca `Sci.py` a um equivalente do `lsqnonlin` do `matlab` e encontrei o `scipy.optimize.least_squares` que permite o cálculo de problemas não lineares, a seguir consta o formato da função da documentação e detalhes de alguns parâmetros que serão utilizados:

```
least_squares(fun, x0, jac='2-point', bounds=(-inf, inf), method='trf', ftol=1e-08,  
↳ xtol=1e-08, gtol=1e-08, x_scale=1.0, loss='linear', f_scale=1.0, diff_step=None,  
↳ tr_solver=None, tr_options=None, jac_sparsity=None, max_nfev=None, verbose=0,  
↳ args=(), kwargs=None)
```

(SCIPY, 2025)

- **fun**: recebe um função que calcula o erro estimado, ajuda o polinômio a convergir mais rápido, nesse caso a estimativa será dada pela equação de volterra.
- **x0**: valores iniciais estimados para os coeficientes que conforme a instrução serão testados em 0, 1 e valores aleatórios.
- **verbose=0**: nível de detalhamento da saída no console sobre o processo, quando em 2 exibe cada passo do método.
- **args=(x, y, P, M)**: recebe os parâmetros de entradas, saídas, ordem e memória do polinômio que serão repassados para a função de resíduos.

2.1 Código em python

O código foi feito a partir das instruções e utilizando os dados contidos em `IN_OUT_PA.mat` para a análise.

2.1.1 Definições iniciais

No cabeçalho, apenas a importação das bibliotecas, carregamento dos dados do arquivo e definição das variáveis iniciais.

```
from scipy.io import loadmat  
import numpy as np  
import matplotlib.pyplot as plt  
from scipy.optimize import least_squares  
  
loaded_data = loadmat('IN_OUT_PA.mat')  
in_training = loaded_data['in']  
out_training = loaded_data['out']  
  
# Definições dos parâmetros do modelo  
# P - Ordem do polinômio  
# M - Profundidade de memória  
P, M = 2, 1
```

```
# Estimativas iniciais para os coeficientes:

# initial_estimated = np.zeros(P*(M+1))
# initial_estimated = np.ones (P*(M+1))
initial_estimated = np.random.randn(P*(M+1))

print(f"P: {P}, M: {M} \ninitial_estimated: ", np.array(initial_estimated), "\n")
```

2.1.2 Matriz de regressão

A seguir a função `createRegressionMatrix` é a mesma utilizada na atividade 2 para gerar a matriz de regressão que será utilizada para multiplicar os coeficientes calculados e obter a estimativa.

```
def createRegressionMatrix(in_data, P, M):
    regression_matrix = []

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

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

    return np.array(regression_matrix);
```

2.1.3 Função de resíduos

Essa é a função que o método em Python recebe como parâmetro para calcular os coeficientes, ela é uma aplicação da equação de Volterra presente na atividade 2, o objetivo é retornar a diferença entre o calculado pelo método e o valor real.

```
def mp_residuals(coef, x, y, P, M):
    """
    coef: vetor de coeficientes de tamanho P*(M+1)
    x: sinal de entrada (array 1D)
    y: sinal de saída medido (array 1D)
    P: ordem polinomial
    M: profundidade de memória
    retorna: vetor de resíduos y_est - y
    """
```

```

# Reconstruir  $h_p(m)$  em matriz  $P \times (M+1)$ 
coef_matrix = coef.reshape(P, M + 1)
y_est = []

for n in range(len(x)):
    estimated_value = 0.0

    for p in range(1, P + 1): # Incluir termos até a ordem P
        for m in range(M + 1): # Considerar atrasos até M
            dataIndex = n - m
            if dataIndex >= 0: # Verificar se o índice é válido
                estimated_value += coef_matrix[p - 1, m] * (x[dataIndex] ** p) #
                ↪ Equação de Volterra

    y_est.append(estimated_value)

# Assim a função de resíduo ajuda o modelo a convergir com base na equação de
↪ Volterra.
return np.array(y_est) - y

```

2.1.4 Cálculo dos coeficientes

Aqui os coeficiente propriamente ditos são calculados, com base no método equivalente do Python `least_squares` com os parâmetros aplicados como: `verbose=2` para melhorar a visualização do cálculo, e `in_training` com `out_training`

```

def calcCoefByLeastSquares():
    result = least_squares(mp_residuals, initial_estimated, args=(in_training.ravel(),
        ↪ out_training.ravel(), P, M), verbose=2)
    print("\nResultado da otimização:", result)
    coef = result.x
    print("\nCoeficientes resultantes: a =", coef)

    return coef

```

2.1.5 Cálculo do vetor de erro

O vetor de erro é calculado com base na saída estimada e na saída medida proveniente dos dados fornecidos, com isso é gerado o erro de cada cálculo e organizado em um vetor para análises futuras.

```

def calcVectorError(out_estimated):
    vector_error = []
    for i in range(len(in_training)):
        error = out_estimated[i] - out_training[i]
        vector_error.append(error)

    return np.array(vector_error)

```

2.1.6 Exibição do vetor de erro

Com o vetor de erro calculado é possível realizar algumas análises a fim de visualizar melhor a precisão do modelo, dessa forma, os valores de erro máximo, médio e mínimo são apresentados. Para o erro máximo e mínimo, uma visualização especial é atribuída da forma em que é possível visualizar a diferença entre o valor real e estimado do gráfico, a representação é feita com um ponto, uma cruz e uma reta entre os valores.

```
def checkError(vector_error):
    print("\nErro médio:", np.mean(vector_error), "\nErro máximo em módulo:",
        ↳ np.max(np.abs(vector_error)), "\nErro mínimo em módulo:",
        ↳ np.min(np.abs(vector_error)))
    max_error_index = np.argmax(np.abs(vector_error))
    min_error_index = np.argmin(np.abs(vector_error))

    # Pontos de maior e menor erro, para análise visual do modelo
    max_error_point_original = (in_training[max_error_index],
        ↳ out_training[max_error_index])
    max_error_point_estimated = (in_training[max_error_index],
        ↳ out_estimated_volterra[max_error_index])
    min_error_point_original = (in_training[min_error_index],
        ↳ out_training[min_error_index])
    min_error_point_estimated = (in_training[min_error_index],
        ↳ out_estimated_volterra[min_error_index])

    return max_error_point_original, max_error_point_estimated,
        ↳ min_error_point_original, min_error_point_estimated

def plotErrorPair(p1, p2, color, label_base, marker1="x", marker2="o", zorder=5):
    x1, y1 = p1[0].item(), p1[1].item()
    x2, y2 = p2[0].item(), p2[1].item()

    plt.scatter(x1, y1, color=color, label=f'{label_base} Original', marker=marker1,
        ↳ s=150, zorder=zorder)
    plt.scatter(x2, y2, color=color, label=f'{label_base} Estimado', marker=marker2,
        ↳ s=80, zorder=zorder)

    plt.plot([x1, x2], [y1, y2], color=color, linestyle='--', linewidth=1.5,
        ↳ label=f'Linha de {label_base}', zorder=zorder-1)
```

2.1.7 Saída gráfica com os resultados

Por fim, são definidas as configurações de visualização, como cores, posicionamento e parâmetros gerais do gráfico.

```
coef = calcCoefByLeastSquares()
regression_matrix = createRegressionMatrix(in_training, P, M)
out_estimated_volterra = np.dot(regression_matrix, coef)

vector_error = calcVectorError(out_estimated_volterra)
max_error_point_original, max_error_point_estimated, min_error_point_original,
↪ min_error_point_estimated = checkError(vector_error)

plotErrorPair(max_error_point_original, max_error_point_estimated, color='green',
↪ label_base='Erro Máximo')
plotErrorPair(min_error_point_original, min_error_point_estimated, color='red',
↪ label_base='Erro Mínimo')

plt.scatter(in_training, out_training, label='Dados Originais', color='blue')
plt.xlabel('in_training')
plt.ylabel('out_training')
plt.title('Dados Originais e Estimados com Erros')
plt.scatter(in_training, out_estimated_volterra, color='orange', label='Ajuste')
plt.legend()
plt.grid()
plt.show()
```

2.1.8 Código completo

O código completo está disponível no seguinte endereço do GitHub:

[script.py — Repositório no GitHub](#)

Também criei um repositório no GitHub com todos os códigos feitos na iniciação científica caso queira consultar:

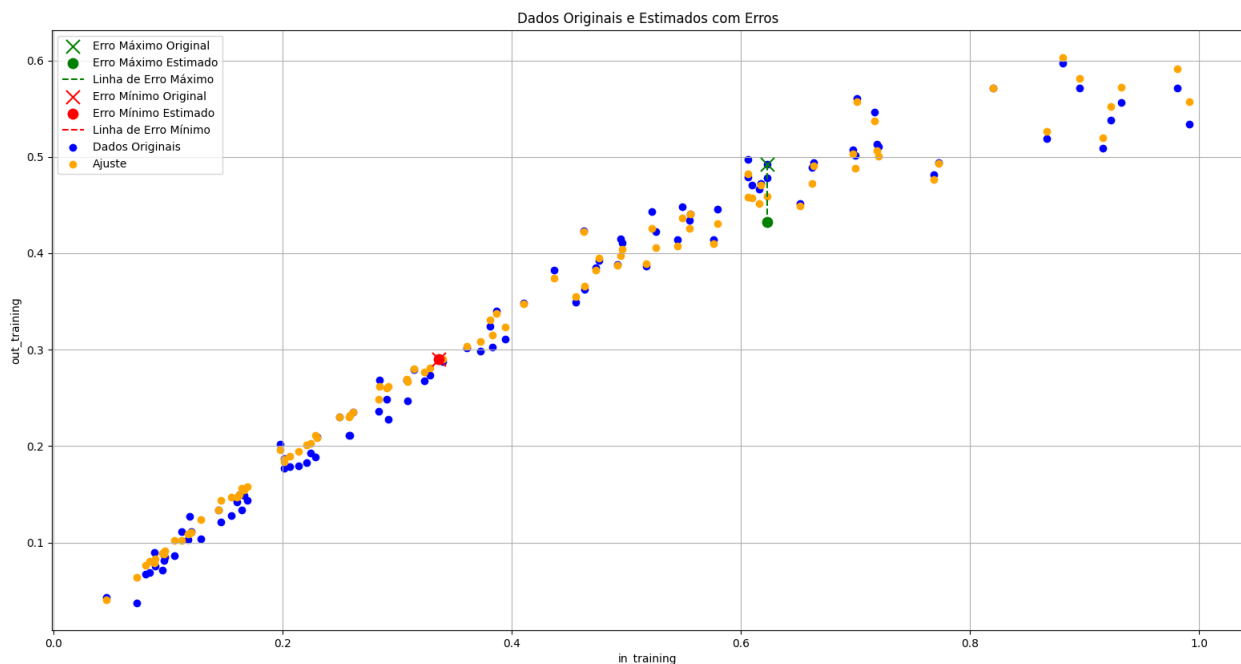
github.com/andrepozzan/ic-gics

3 Resultados

A partir do código construído, é possível obter os gráficos para análise e também modificar seus parâmetros para observar seu comportamento.

Observe a saída `initial_estimated` no console de cada execução a qual mostra o vetor inicial estimado para os coeficientes.

3.1 Estimativa inicial com zero, $P = 2$ e $M = 1$



Saída no console:

Observando que no total foram necessárias 3 iterações para o método realizar a otimização.

```
P: 2, M: 1
initial_estimated: [0. 0. 0. 0.]

Iteration    Total nfev    Cost    Cost reduction    Step norm    Optimality
0            1           6.2259e+00           6.21e+00           1.00e+00           1.69e+01
1            2           1.5176e-02           6.21e-03           4.70e-01           3.68e-02
2            3           8.9624e-03           6.21e-03           4.70e-01           4.94e-09

`gtol` termination condition is satisfied.
Function evaluations 3, initial cost 6.2259e+00, final cost 8.9624e-03, first-order
↪ optimality 4.94e-09.

Resultado da otimização:      message: `gtol` termination condition is satisfied.
      success: True
      status: 1
```

```

fun: [-5.977e-02 -2.062e-02 ... -8.311e-04 -9.170e-03]
x: [ 1.125e+00 -1.305e-01 -6.905e-01  2.870e-01]
cost: 0.008962369480212881
jac: [[ 6.230e-01 -0.000e+00  3.882e-01  0.000e+00]
      [ 6.060e-01  6.230e-01  3.673e-01  3.882e-01]
      ...
      [ 2.299e-01  3.080e-01  5.286e-02  9.489e-02]
      [ 1.118e-01  2.299e-01  1.249e-02  5.286e-02]]
grad: [-4.754e-09 -4.942e-09 -2.873e-09 -2.752e-09]
optimality: 4.941966598968894e-09
active_mask: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00]
nfev: 3
njev: 3

```

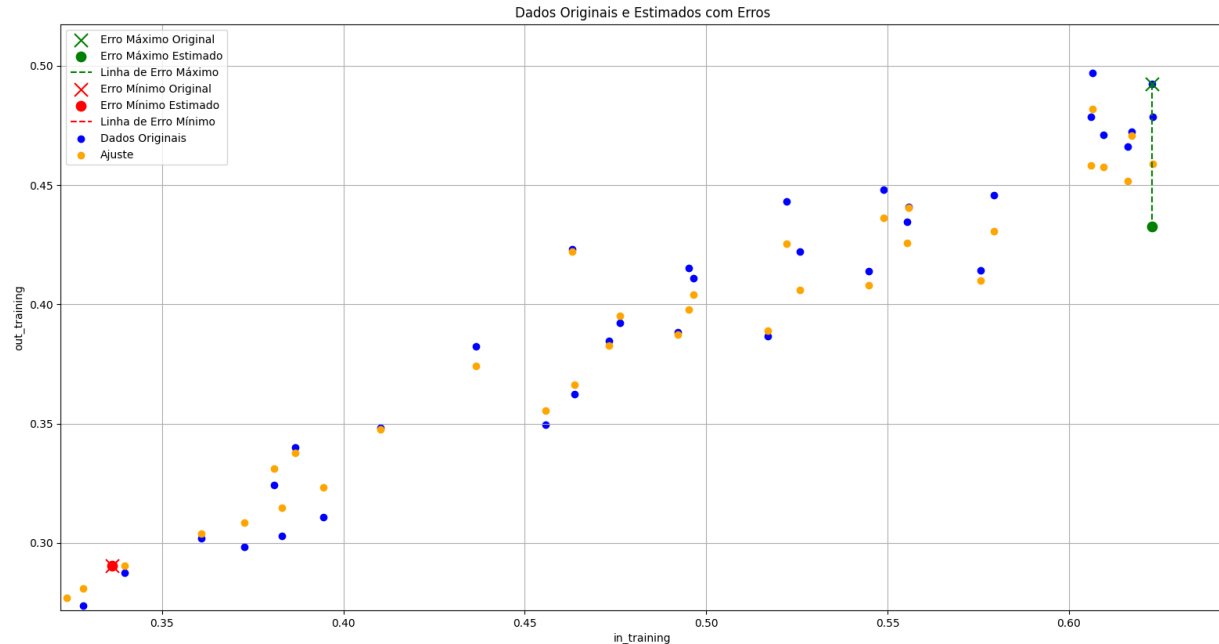
Coefficientes resultantes: $a = [1.12463317 -0.13050207 -0.69045674 0.28702601]$

Erro médio: 0.002151728708705714

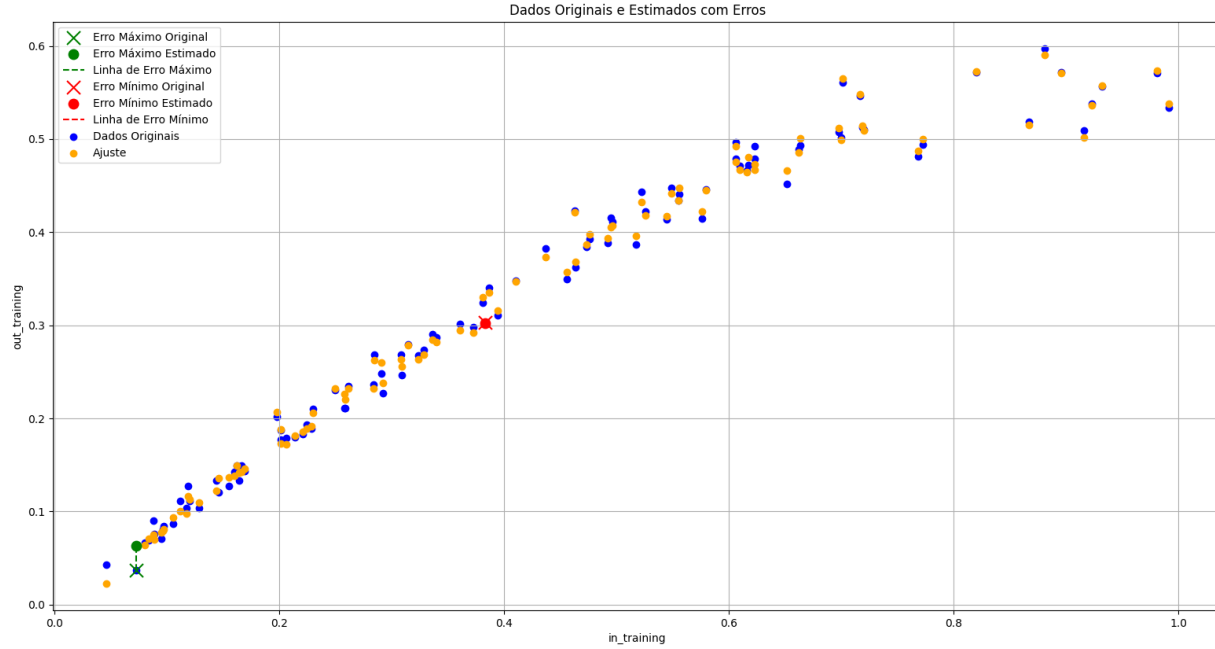
Erro máximo em módulo: 0.05976860426473757

Erro mínimo em módulo: 1.816783502101016e-05

É possível observar os pontos de maior e menor erro no gráfico com a reta indicando a distância entre eles.



3.2 Estimativa inicial com zero, $P = 5$ e $M = 2$



O erro máximo observado diminuiu consideravelmente ao aumentar a ordem e a memória do polinômio, mas, em contrapartida, foram necessárias mais iterações para o método encontrar os valores:

```
P: 5, M: 2
initial_estimated: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Iteration	Total nfev	Cost	Cost reduction	Step norm	Optimality
0	1	6.2259e+00			1.69e+01
1	2	4.0704e-03	6.22e+00	1.00e+00	2.98e-03
2	3	3.6238e-03	4.47e-04	2.00e+00	7.23e-05
3	4	3.2067e-03	4.17e-04	4.00e+00	4.85e-05
4	5	2.7994e-03	4.07e-04	8.00e+00	1.14e-05
5	6	2.7663e-03	3.31e-05	3.77e+00	7.52e-08
6	7	2.7663e-03	1.75e-14	6.48e-05	4.72e-10

```
`gtol` termination condition is satisfied.
Function evaluations 7, initial cost 6.2259e+00, final cost 2.7663e-03, first-order
↪ optimality 4.72e-10.

Resultado da otimização:      message: `gtol` termination condition is satisfied.
success: True
status: 1
fun: [-2.504e-02 -3.375e-03 ... -3.822e-03 -1.084e-02]
x: [ 1.558e+00 -3.974e-01 ... -1.764e+00 -1.221e+00]
cost: 0.0027662790958105712
jac: [[ 6.230e-01 -0.000e+00 ... -0.000e+00 -0.000e+00]
      [ 6.060e-01 6.230e-01 ... 9.387e-02 -0.000e+00]
      ...]
```

```

      [ 2.299e-01  3.080e-01 ...  2.774e-03  4.524e-03]
      [ 1.118e-01  2.299e-01 ...  6.423e-04  2.774e-03]]
    grad: [ 3.633e-10  3.988e-10 ...  1.089e-10  1.923e-10]
    optimality: 4.72014045145748e-10
    active_mask: [ 0.000e+00  0.000e+00 ...  0.000e+00  0.000e+00]
    nfev: 7
    njev: 7

Coeficientes resultantes: a = [ 1.55752587 -0.39739039 -0.31094042 -3.91930797
↪  2.3182382  1.57725336
   8.82537828 -5.1016161 -3.3497784 -9.71997122  5.05170015  3.3135624
   3.71322425 -1.7635762 -1.22108716]

Erro médio: -0.00010535172824854574
Erro máximo em módulo: 0.026171578991292327
Erro mínimo em módulo: 5.248732737350714e-05

```

3.3 Estimativa inicial com um, $P = 5$ e $M = 2$

O gráfico manteve-se idêntico ao resultado anterior, porém foram necessárias 2 iterações a menos no método para encontrar o resultado, o que economiza poder computacional e torna a estimativa inicial com números 1 melhor para esse caso:

```

P: 5, M: 2
initial_estimated: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

  Iteration    Total nfev      Cost    Cost reduction    Step norm    Optimality
        0             1    8.1859e+02      8.19e+02      3.87e+00      1.83e+02
        1             2    4.1418e-03      1.08e-03      7.75e+00      8.70e-02
        2             3    3.0605e-03      2.94e-04      9.60e+00      3.82e-05
        3             4    2.7663e-03      2.86e-14      5.68e-05      5.55e-07
        4             5    2.7663e-03
`gtol` termination condition is satisfied.
Function evaluations 5, initial cost 8.1859e+02, final cost 2.7663e-03, first-order
↪ optimality 5.34e-10.

Resultado da otimização:      message: `gtol` termination condition is satisfied.
    success: True
    status: 1
    fun: [-2.504e-02 -3.375e-03 ... -3.822e-03 -1.084e-02]
    x: [ 1.558e+00 -3.974e-01 ... -1.764e+00 -1.221e+00]
    cost: 0.002766279095810758
    jac: [[ 6.230e-01 -0.000e+00 ... -0.000e+00 -0.000e+00]
          [ 6.060e-01  6.230e-01 ...  9.387e-02 -0.000e+00]
          ...
          [ 2.299e-01  3.080e-01 ...  2.774e-03  4.524e-03]
          [ 1.118e-01  2.299e-01 ...  6.423e-04  2.774e-03]]
    grad: [ 2.510e-10  1.746e-10 ... -4.481e-11 -8.940e-11]
    optimality: 5.339689380760848e-10

```

```

active_mask: [ 0.000e+00  0.000e+00 ...  0.000e+00  0.000e+00]
nfev: 5
njev: 5

Coeficientes resultantes: a = [ 1.55752557 -0.3973905 -0.31094019 -3.919306
↪ 2.31823898 1.57725231
8.82537324 -5.1016183 -3.34977631 -9.71996566 5.05170275 3.31356058
3.71322203 -1.76357725 -1.22108661]

Erro médio: -0.0001053522113393183
Erro máximo em módulo: 0.02617157521596135
Erro mínimo em módulo: 5.2481547906491066e-05

```

3.4 Estimativa inicial com números aleatórios, $P = 5$ e $M = 2$

Mesmo gráfico do passo anterior porém os valores aleatórios não reduziram a quantidade de iterações necessárias:

```

P: 5, M: 2
initial_estimated: [-1.17078009 -0.20586654 0.3186317 -0.21442648 0.88641537
↪ 1.13386784
1.55845472 -0.5760856 -0.10819173 0.40346181 -0.33170728 -0.32759056
2.82429461 0.01111982 -0.02282294]

Iteration    Total nfev    Cost          Cost reduction    Step norm    Optimality
0            1            2.8947e+01      2.89e+01          3.84e+00     1.63e+01
1            2            3.9185e-03      9.42e-04          7.68e+00     3.23e-02
2            3            2.9764e-03      2.10e-04          8.19e+00     3.06e-05
3            4            2.7663e-03      1.66e-14          3.79e-05     1.96e-07
4            5            2.7663e-03      1.66e-14          3.79e-05     1.64e-09

`gtol` termination condition is satisfied.
Function evaluations 5, initial cost 2.8947e+01, final cost 2.7663e-03, first-order
↪ optimality 1.64e-09.

Resultado da otimização:      message: `gtol` termination condition is satisfied.
success: True
status: 1
fun: [-2.504e-02 -3.375e-03 ... -3.822e-03 -1.084e-02]
x: [ 1.558e+00 -3.974e-01 ... -1.764e+00 -1.221e+00]
cost: 0.0027662790958127067
jac: [[ 6.230e-01 -0.000e+00 ... -0.000e+00 -0.000e+00]
      [ 6.060e-01 6.230e-01 ... 9.387e-02 -0.000e+00]
      ...
      [ 2.299e-01 3.080e-01 ... 2.774e-03 4.524e-03]
      [ 1.118e-01 2.299e-01 ... 6.423e-04 2.774e-03]]
grad: [ 5.119e-10 -1.476e-10 ... -8.599e-11 -4.819e-11]
optimality: 1.638959861924305e-09
active_mask: [ 0.000e+00  0.000e+00 ...  0.000e+00  0.000e+00]
nfev: 5

```

```
njev: 5

Coeficientes resultantes: a = [ 1.55752653 -0.39739127 -0.31094027 -3.91931199
↪ 2.31824377 1.5772525
 8.8253879 -5.10162987 -3.34977656 -9.71998128 5.05171505 3.31356063
 3.71322803 -1.76358197 -1.22108658]

Erro médio: -0.00010535185964552545
Erro máximo em módulo: 0.026171567791037076
Erro mínimo em módulo: 5.249206977842036e-05
```

E em alguns casos os números aleatórios custaram 2 iterações a mais que a inicialização apenas com números 1:

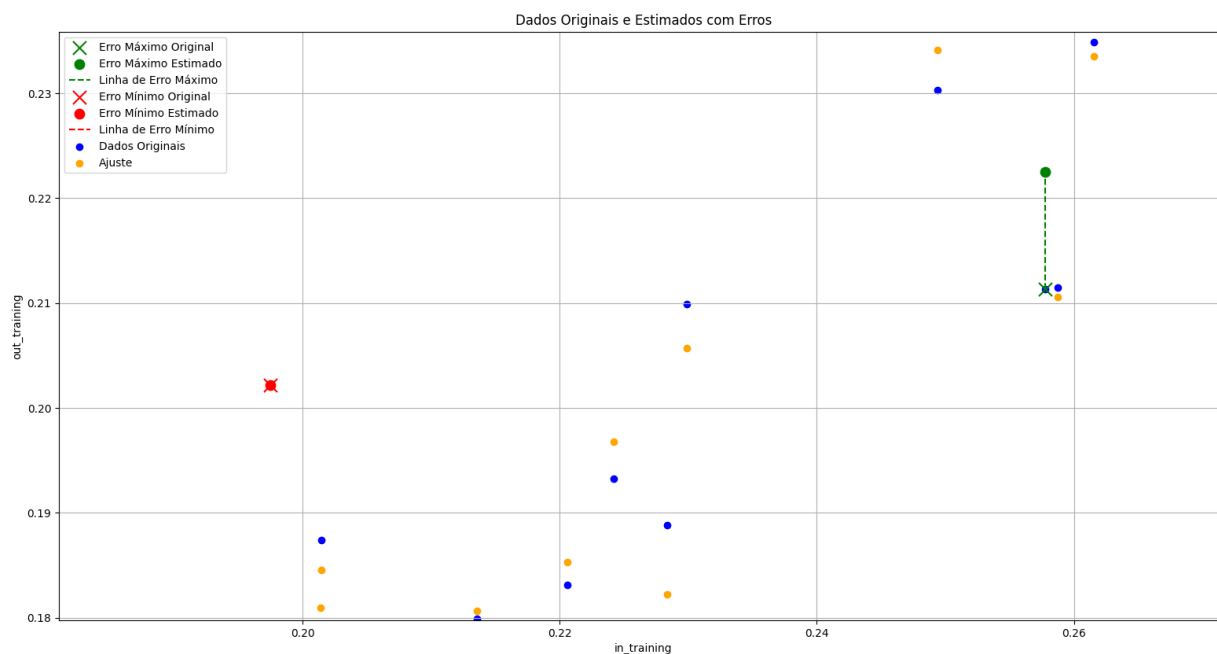
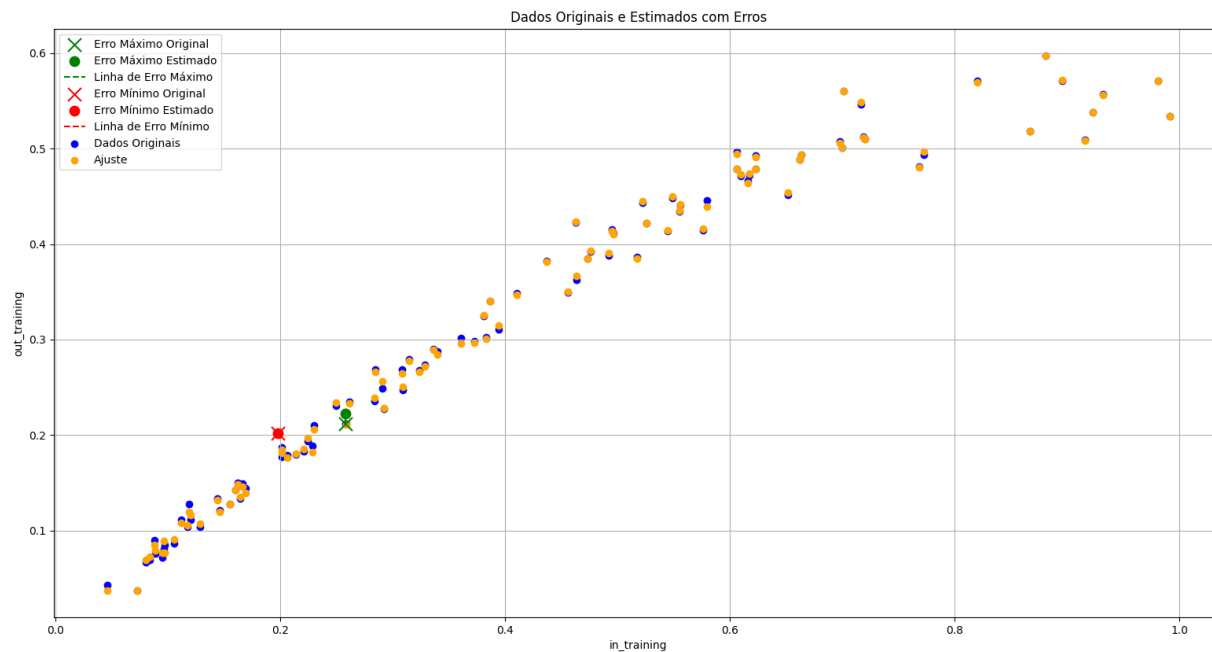
```
P: 5, M: 2
initial_estimated: [-0.63095277 0.13343712 0.01241339 0.13786534 -0.15895004
↪ 0.14439986
-0.58959949 0.15227403 0.31884679 -0.20060631 -0.02172653 0.65131776
-0.16213513 -0.25705356 0.06997731]

  Iteration    Total njev      Cost    Cost reduction    Step norm    Optimality
      0           1    1.7710e+01
      1           2    4.2676e-02    1.77e+01    1.23e+00    3.03e-01
      2           3    3.6586e-03    3.90e-02    2.46e+00    7.69e-05
      3           4    3.1518e-03    5.07e-04    4.92e+00    4.53e-05
      4           5    2.7692e-03    3.83e-04    9.85e+00    3.12e-06
      5           6    2.7663e-03    2.91e-06    1.18e+00    6.96e-08
      6           7    2.7663e-03    7.93e-16    2.34e-05    9.32e-10

`gtol` termination condition is satisfied.
Function evaluations 7, initial cost 1.7710e+01, final cost 2.7663e-03, first-order
↪ optimality 9.32e-10.
```

3.5 Estimativa inicial com um, $P = 10$, $M = 5$

Neste caso o erro foi bem menor que os anteriores porém o número de iterações aumentou expressivamente o que leva maior tempo para ser calculado:



P: 10, M: 5

```
initial_estimated: [-1.08633350e-01  2.12482633e-01 -1.60073722e-01 -2.82308328e-01
 2.68145046e+00 -2.75477011e-01  8.45065359e-01 -8.01266652e-01
 1.57240527e-01  3.27955436e-02  7.47700051e-01 -2.97375866e-01
-1.20124505e+00 -3.47440102e-01  7.44279132e-01  1.07019480e+00
 1.31725066e+00  4.60133091e-01  1.53706280e+00  5.41823525e-01
 3.24846051e-01  7.01426761e-02  1.61383201e+00  1.51402344e+00
-1.52200351e+00 -4.36167781e-01 -2.05594281e-01 -2.91840034e-01]
```

```
-5.51962775e-02 -2.32713428e-04 7.77310148e-02 1.25922364e+00
6.88256254e-02 -3.82683290e-02 -5.07315073e-01 -5.09691483e-01
-1.77253351e-02 7.57981234e-01 -6.50275180e-03 -1.12421325e+00
-1.56111325e+00 2.18220242e-01 8.14729243e-01 4.26521891e-01
-1.21641574e-01 -4.81518002e-01 -6.25873367e-01 5.20910885e-01
-2.51561187e-01 1.03226719e+00 -1.82816877e+00 2.19451553e+00
3.59752640e-01 -2.36237132e-03 -5.56535598e-01 5.34127300e-01
1.00979471e+00 -2.69201907e-02 1.32686811e+00 1.19891653e+00]
```

Iteration	Total nfev	Cost	Cost reduction	Step norm	Optimality
0	1	3.2720e+02			1.13e+02
1	2	2.0008e-03	3.27e+02	6.88e+00	4.86e-03
2	3	1.5311e-03	4.70e-04	1.38e+01	5.62e-06
3	4	1.2486e-03	2.83e-04	2.75e+01	2.45e-06
4	5	1.1202e-03	1.28e-04	5.51e+01	3.77e-06
5	6	1.0247e-03	9.55e-05	1.10e+02	1.16e-05
6	7	9.3183e-04	9.28e-05	2.20e+02	3.12e-05
7	8	8.8136e-04	5.05e-05	4.41e+02	2.84e-05
8	9	8.0662e-04	7.47e-05	8.81e+02	3.63e-05
9	10	7.0160e-04	1.05e-04	1.76e+03	9.35e-05
10	11	6.3413e-04	6.75e-05	3.52e+03	1.14e-03
11	12	6.1004e-04	2.41e-05	7.05e+03	1.84e-03
12	13	5.7127e-04	3.88e-05	1.41e+04	4.96e-04
13	17	5.7122e-04	4.93e-08	4.41e+02	2.83e-03
14	18	5.6929e-04	1.93e-06	1.10e+02	2.48e-04
15	19	5.6866e-04	6.28e-07	2.20e+02	5.54e-05
16	20	5.6755e-04	1.11e-06	4.41e+02	2.76e-04
17	21	5.6688e-04	6.70e-07	8.81e+02	1.62e-03
18	22	5.6595e-04	9.31e-07	2.20e+02	6.38e-05
19	23	5.6461e-04	1.34e-06	4.41e+02	9.54e-05
20	24	5.6251e-04	2.10e-06	8.81e+02	8.14e-05
21	25	5.5844e-04	4.07e-06	1.76e+03	2.09e-04
22	26	5.5056e-04	7.88e-06	3.52e+03	6.08e-04
23	27	5.3709e-04	1.35e-05	7.05e+03	8.83e-04
24	28	5.1621e-04	2.09e-05	1.41e+04	3.03e-03
25	29	4.9326e-04	2.29e-05	2.82e+04	2.14e-03
26	31	4.9056e-04	2.70e-06	5.57e+03	1.83e-03
27	34	4.9039e-04	1.69e-07	3.48e+02	7.64e-04
28	35	4.8902e-04	1.37e-06	8.70e+01	1.01e-04
29	36	4.8854e-04	4.84e-07	8.70e+01	8.09e-05
30	37	4.8820e-04	3.35e-07	8.70e+01	7.92e-05
31	38	4.8788e-04	3.25e-07	1.74e+02	2.75e-04
32	42	4.8784e-04	3.18e-08	5.44e+00	1.58e-04
33	43	4.8784e-04	9.19e-09	1.36e+00	7.35e-06
34	44	4.8783e-04	6.41e-09	1.36e+00	7.45e-05
35	45	4.8782e-04	4.60e-09	3.40e-01	9.87e-06
36	46	4.8782e-04	5.42e-09	3.40e-01	4.37e-06
37	47	4.8781e-04	4.08e-09	3.40e-01	1.89e-05
38	51	4.8781e-04	1.29e-10	5.31e-03	9.49e-07
39	52	4.8781e-04	1.99e-11	1.33e-03	5.98e-07
40	53	4.8781e-04	4.88e-12	3.32e-04	1.62e-06

`xtol` termination condition is satisfied.

Function evaluations 53, initial cost 3.2720e+02, final cost 4.8781e-04, first-order
 ↪ optimality 1.62e-06.

```

Resultado da otimização:      message: `xtol` termination condition is satisfied.
      success: True
      status: 3
      fun: [-9.798e-04 -1.157e-04 ... -4.253e-03 -3.136e-03]
      x: [ 4.835e+00 -1.704e+00 ...  2.133e+03  2.067e+03]
      cost: 0.00048781465236729656
      jac: [[ 6.230e-01 -0.000e+00 ...  0.000e+00  0.000e+00]
            [ 6.060e-01  6.230e-01 ...  0.000e+00  0.000e+00]
            ...
            [ 2.299e-01  3.080e-01 ...  3.413e-06  2.727e-07]
            [ 1.118e-01  2.299e-01 ...  1.448e-05  3.413e-06]]
      grad: [-3.602e-08  3.600e-07 ...  1.784e-08 -7.870e-10]
      optimality: 1.6209953246550807e-06
      active_mask: [ 0.000e+00  0.000e+00 ...  0.000e+00  0.000e+00]
      nfev: 53
      njev: 41

```

```

Coeficientes resultantes: a = [ 4.83535573e+00 -1.70368874e+00 -3.21767843e+00
↪  8.61252447e-01

```

```

-7.25502172e-01 -1.17468218e+00 -6.97557645e+01  2.56029328e+01
 5.75995600e+01 -7.47024143e+00  2.37356515e+01  3.07469691e+01
 6.16039931e+02 -1.93594320e+02 -5.20219567e+02  4.49032339e+00
-2.90941846e+02 -3.39051491e+02 -3.18057040e+03  8.90925460e+02
 2.73002938e+03  2.85432193e+02  1.85856259e+03  2.01816171e+03
 1.02253872e+04 -2.67620027e+03 -8.88521361e+03 -1.94129631e+03
-6.96593088e+03 -7.21453280e+03 -2.10787075e+04  5.36418258e+03
 1.85008084e+04  6.17368684e+03  1.61441242e+04  1.62178670e+04
 2.78583175e+04 -7.10251755e+03 -2.46645961e+04 -1.10560263e+04
-2.34185723e+04 -2.30905083e+04 -2.28145934e+04  5.94696726e+03
 2.03514248e+04  1.13919989e+04  2.06901706e+04  2.01869304e+04
 1.05348839e+04 -2.84251306e+03 -9.46009313e+03 -6.29755898e+03
-1.01735804e+04 -9.87500688e+03 -2.09543415e+03  5.89149902e+02
 1.89313361e+03  1.44635442e+03  2.13287690e+03  2.06666204e+03]

```

```

Erro médio: -1.025542252342756e-05
Erro máximo em módulo: 0.011150810721393845
Erro mínimo em módulo: 6.765678819575793e-06

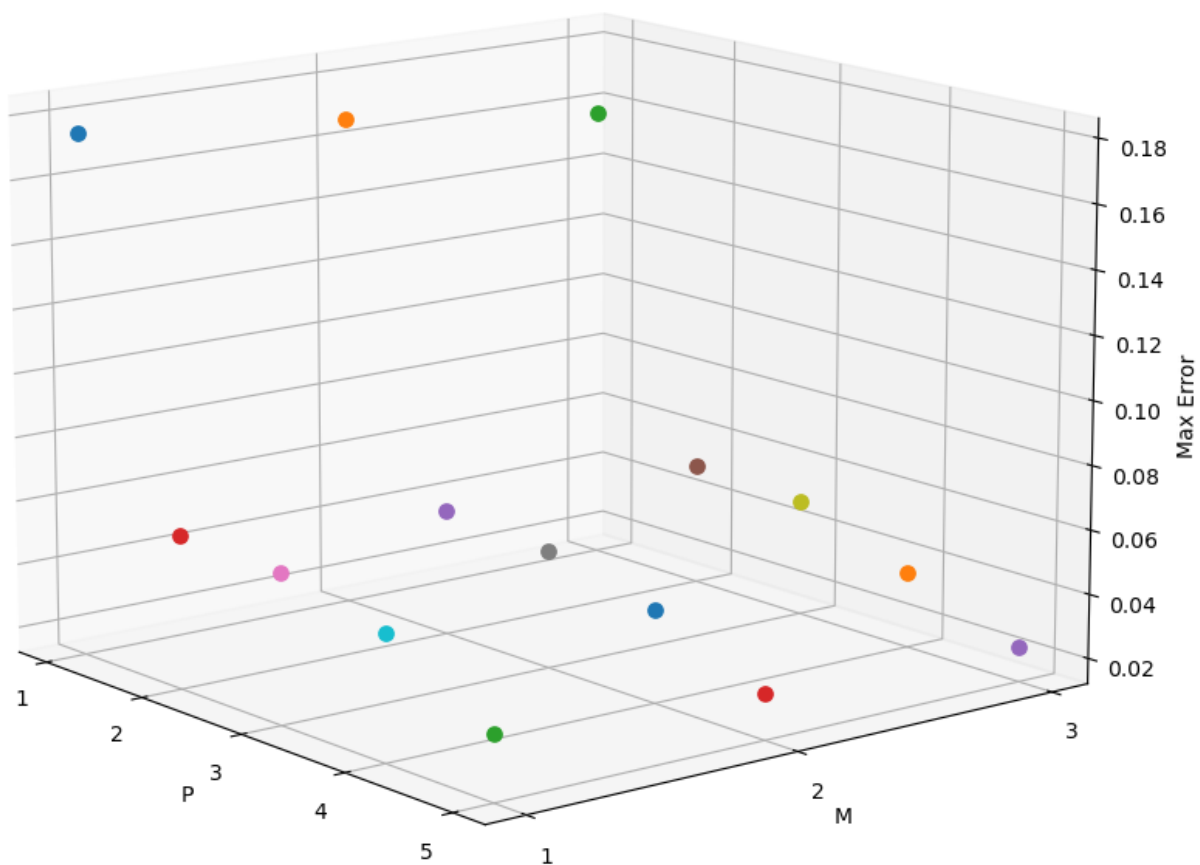
```

4 Extra

Observei que ao alterar os valores de P e M no método o erro máximo era alterado, então tive a dúvida de verificar quais seriam as combinações de P e M que gerariam melhor resultado, assim, automatizei o código para executar várias vezes o método e salvar o erro obtido para cada P e M .

No final, fiz o código exibir um gráfico no \mathbb{R}^3 para comparação:

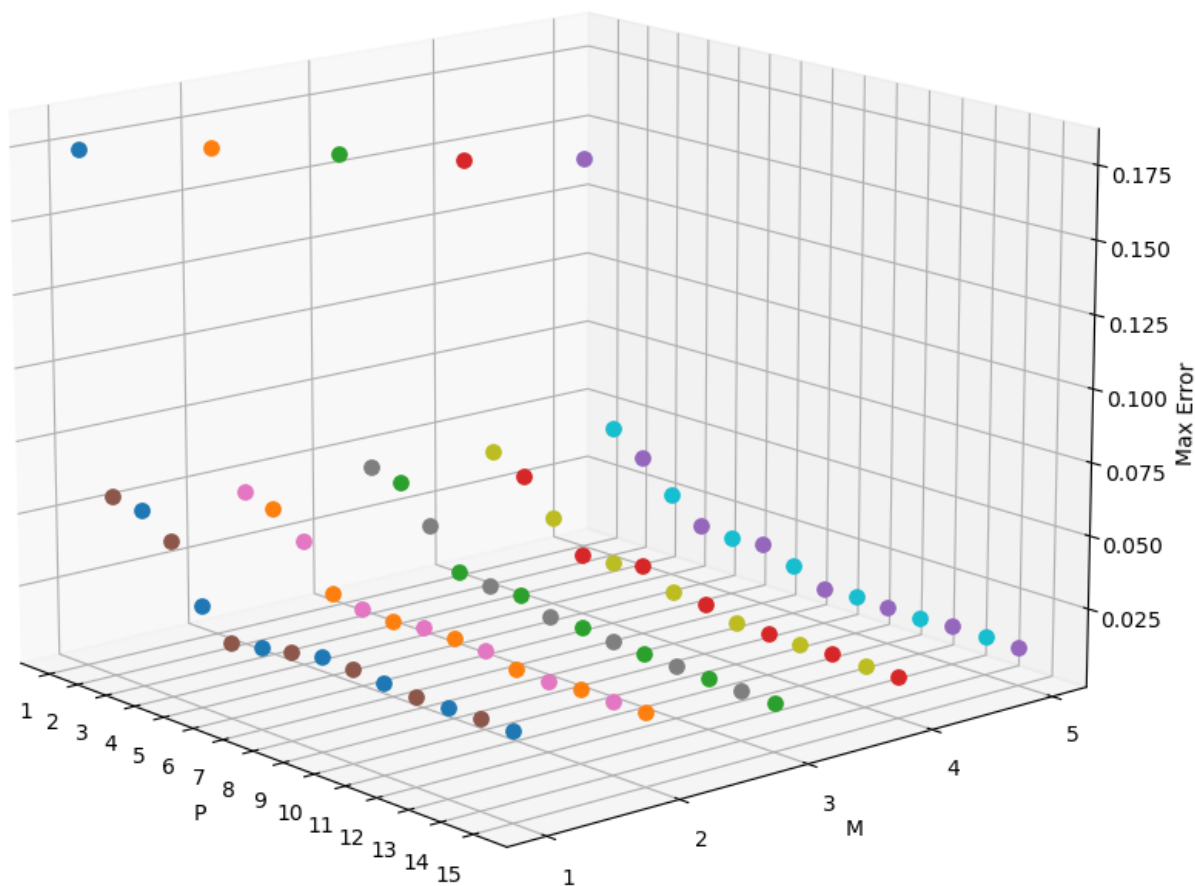
4.1 P até 5 e M até 3

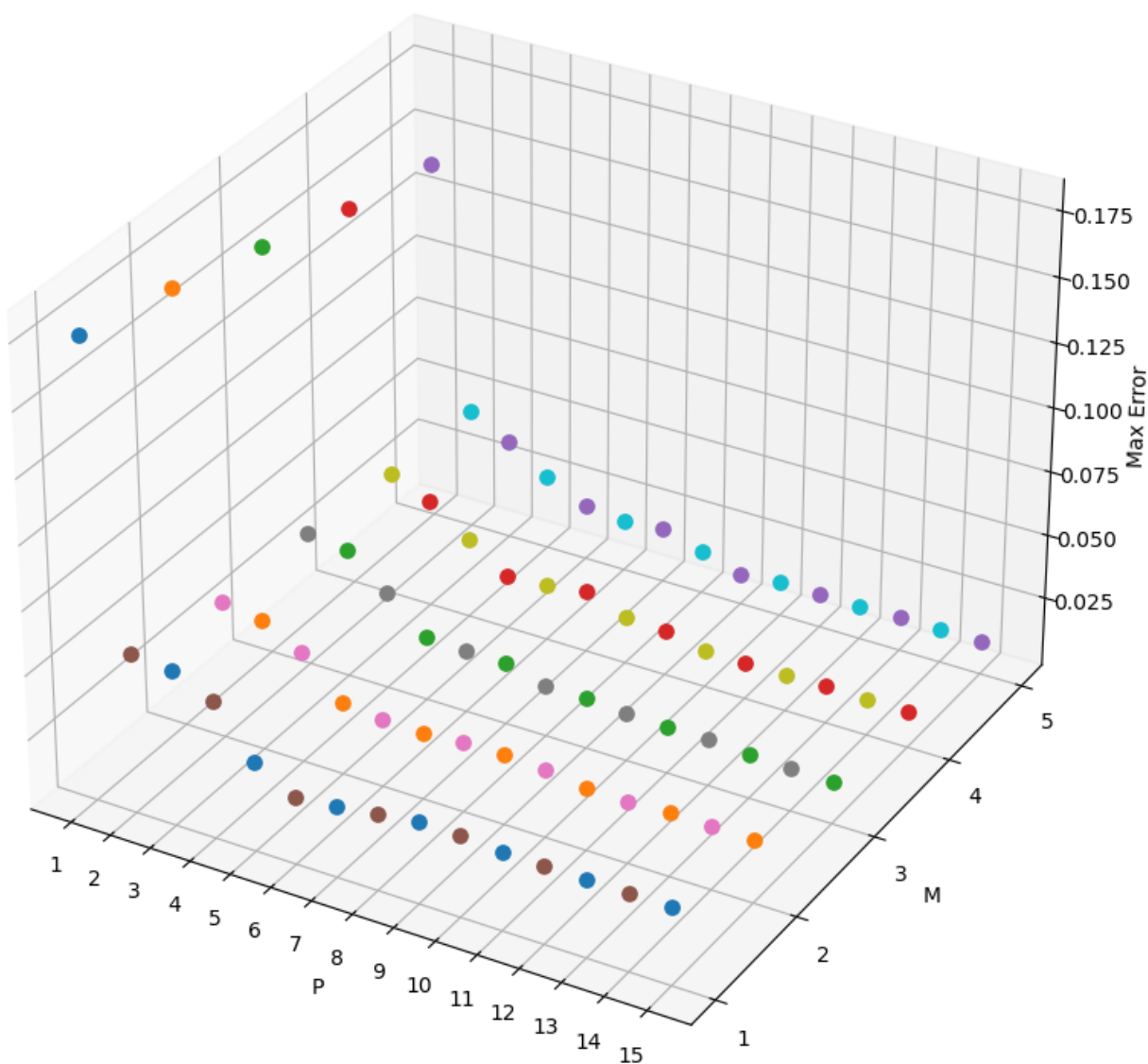


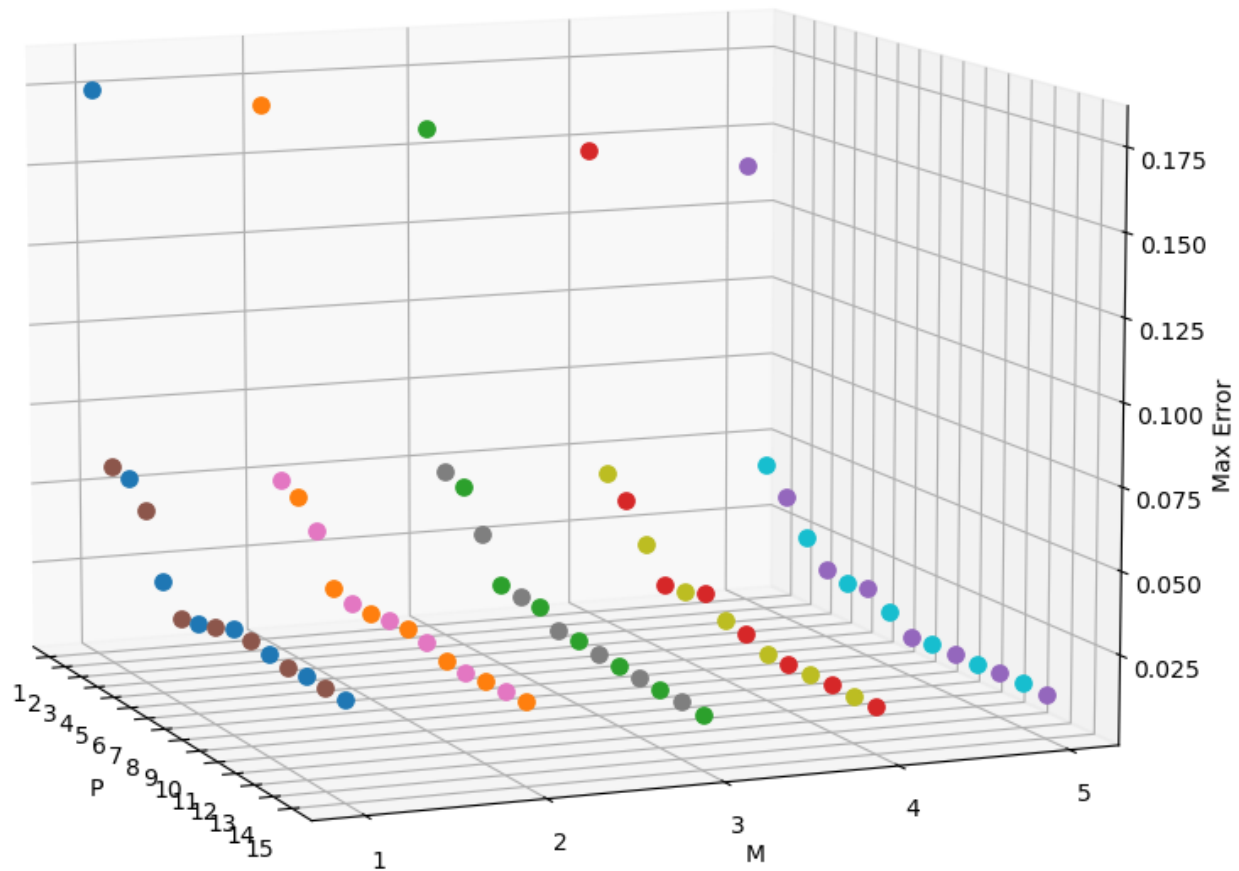
observe que há um salto grande de precisão de $P = 1$ para $P > 1$ e também que a maior parte da precisão é determinada pelo grau do polinômio e a memória contribui um pouco menos para grandes saltos de precisão.

4.2 P até 15 e M até 5

Neste cenário o gráfico demorou alguns minutos para ser calculado por conta do grande volume de operações, a observação foi interessante, o método ganhou bastante precisão até $P = 6$, depois foi questão de aperfeiçoamento impulsionado pela memória. Agora com uma visão mais ampla a memória se provou ser efetiva causando grandes alterações quando se compara $M = 1$ e $M = 5$.







4.3 Código final

O código dessa seção extra pode ser encontrado em:

[andrepozzan - 3D-model.py](#)

Caso queira testar recomento alterar os valores de `P_MAX` e `M_MAX` para não levar muito tempo de execução.

Referências

LIMA, Eduardo Gonçalves de. **Behavioral modeling and digital base-band predistortion of RF power amplifiers**. Jan. 2009. Tese (Doutorado) – POLITECNICO DI TORINO.

MATHWORKS. **lsqnonlin: Solve nonlinear least-squares (nonlinear data-fitting) problems**. [S.l.: s.n.], 2025. Acessado em: 1 abr. 2025. Disponível em: <https://www.mathworks.com/help/optim/ug/lsqnonlin.html>.

SCIPY. **scipy.optimize.least_squares: Solve a nonlinear least-squares problem with bounds on the variables**. [S.l.: s.n.], 2025. Acessado em: 20 abr. 2025. Disponível em: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html.