Inteligência Artificial: Ajuste de curvas polinomiais

Victor Calebe Cavalcante de Macedo
João Victor Rocha Muhamad
Instituto Federal de Goiás, IFG
Goiânia-GO, Brazil
calebe.m@academico.ifg.edu.br, joaovictormuhamad@hotmail.com

Resumo - Com o intuito de adquirir conhecimento na matéria de Inteligência artificial, é proposto o presente trabalho, onde, através de um conjunto de pontos "corrompidos", busca-se voltar para função original denominada "h(x)" escolhida pelo professor nos requisitos do trabalho.

I. Introdução

O ajuste polinomial é uma técnica que forma um polinômio que se assemelha à curva original. Com o mapeamento de variáveis, separando-as em variáveis de treinamento e de teste, os coeficientes do polinômio são calculados. A partir disso, calcula-se o erro de cada um deles, que indicará o quão distante a resposta calculada está da resposta medida. A inteção do trabalho é formar um polinômio que irá generalizar as respostas para novos valores de x emcontrados no ***Data Set*** de teste.

Para execução do treinamento e posteriormente o teste das curvas obtidas deste treino, foi utilizado a linguagem de programação ***Python*** junto a biblioteca ***NumPy***, possuindo diversas funções uteis para a realização das tarefas propostas. Além disso, foi utilizado a plataforma ***Google Colab*** junto a tecnologia dos ***jupyter Notebooks***.

II. Desenvolvimento

Para o desenvolvimento do trabalho, foi seguido as etapas propostas pelo livro Neural Networks for Patterns Recognition de Christopher M. Bishop onde foi escolhido um conjunto de valores para "x" e substituídos na função "$h(x)$", escolhida pelo professor, dada por :

$$h(x) = -0.85e^{0.9x} + 0.36 + 0.12sin(5\pi x) \tag{1}$$

Inicialmente, foram escolhidos valores de 0 a 1 divididos em 21 intervalos, os 21 valores são guardados em um vetor "x", ou seja, "x =[0.0 , 0.05, 0.10, 0.15, ..., 1.00]" . Assim como proposto pelo livro, maior parte dos dados serão destinados ao treinamento (2/3 dos exemplares) e a parte restante é para teste (1/3 dos exemplares), assim, o vetor "x" é dividido em dois sub vetores, "$xtre$" para os valores de treinamento e "xte" para valores de teste. Para essa tarefa, o vetor foi dividido de forma alternada, ou seja, a cada 3 valores do vetor "x" 2 ão para treinamento e 1 fica para teste.

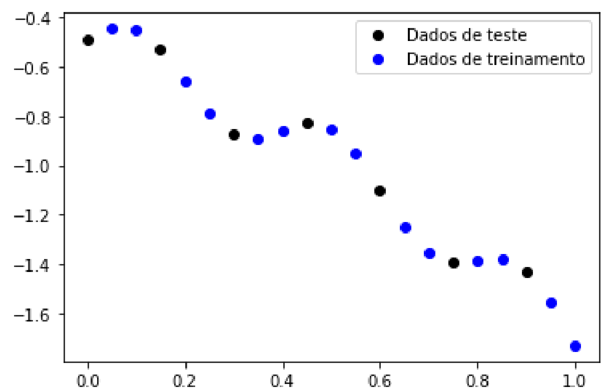


Figure 1: Pontos de teste em preto e pontos de treino em azul

O próximo passo do trabalho é "corromper" os dados do vetor "$xtre$"com um ruído Gaussiano, ou seja, adicionar valores aleatórios de ***Distribuição Gaussiana*** ao vetor. Utilizando da função random.normal(σ,μ), onde os parâmetros são dados pela função :

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{2}$$

Foi escolhido $\sigma = 0.05$ e $\mu = 0$, como proposto pelo professor. Os valores aleatórios foram somados um a um ao vetor de treinamento e colocados em outro vetor denominado "$xtrp$" com os dados "corrompidos", assim como mostrado na **Figura 2**:

$$xtrp = xtr + p(x) \tag{3}$$

O objtetivo agora é, através dos valores corrompidos, realizar um treinamento e encotrar um certo número de curvas (polinômios) até que se ajustem bem a função original. No entanto, para realizar esta etapa, será necessário encontrar os valores dos coeficientes dos respectivos polinômios.

Para achar os coeficientes (pesos) e portanto, as curvas , testou-se valores de "m" (grau do polinômio), foram testados $m = 1$, $m = 3$, $m = 5$, $m = 7$, $m = 9$, $m = 11$ e $m = 13$. Para isso utiliza-se a função 'polyfit' que recebe os valores de $xtrp$, htr e os respectivos pesos. Posteriormente, utilizou-se a função "$polyval$" para substituir os valores de "$xtrp$" no

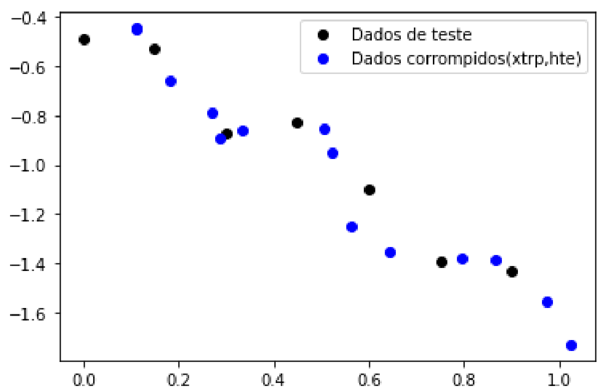


Figure 2: Pontos de teste em preto e pontos aleatórios(corrompidos) em azul

polinômio. Por fim, fez-se a plotagem do gráfico, utilizando a função "$plot$", passando como parâmetros os valores de $xtrp$ e os valores ajustados (curvas aproximadas). Para finalizar, configurou-se o gráfico para melhor apresentar as curvas.

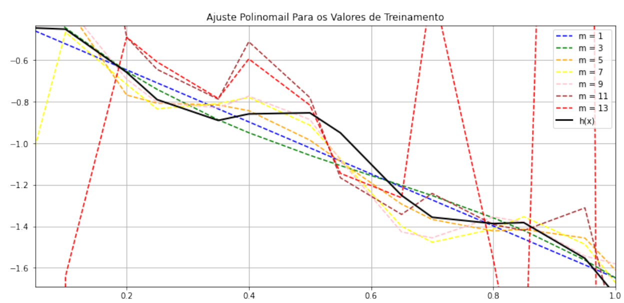


Figure 3: Curva original (em preto) e curvas ajustadas de treinamento com os respectivos valores de "m"

Observando somente pelo gráfico da **Figura 3** não é possível verificar qual grau "m" do polinômio se ajusta melhor a curva original(mostrada em preto), posteriomente, neste trabalho, será abordado como calcular matematicamente o valor do erro de cada curva.

Posteriormente, na etapa de validação dos polinômios, os valores de xte são substituidos nos ajustes obtidos na etapa anterior e postos junto a função original para melhor avaliação visual como mostrado na **Figura 4**.

Como mostrado no gráfico anterior, durante o teste, as curvas não estão tão bem ajustadas quanto no treinamento. No intuito de verificar qual das curvas acima se ajusta melhor à curva original é proposto o calculo do E^{RMS} (***Root-mean-square***) cuja formula é apresentada a seguir:

$$E^{RMS} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\{y(x^n; W^*) - t^n\}^2} \tag{4}$$

A partir dos valores de E^{RMS} e os valores de m, pode-se construir um gráfico de mXE^{RMS}. Os valores dos erros obtidos das curvas de treinamento com seus respectivos "m" (grau do polinômio) podem sem vistas a seguir na **Figura 5**. Pode-se perceber que a partir de polinômios com grau maior que 9 os erro se torna cada vez maior, porém esses valores são apenas do treinamento. No intuito de validar os

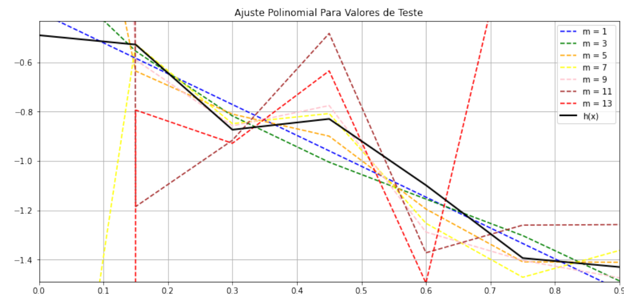


Figure 4: Curvas já treinadas com os valores de teste

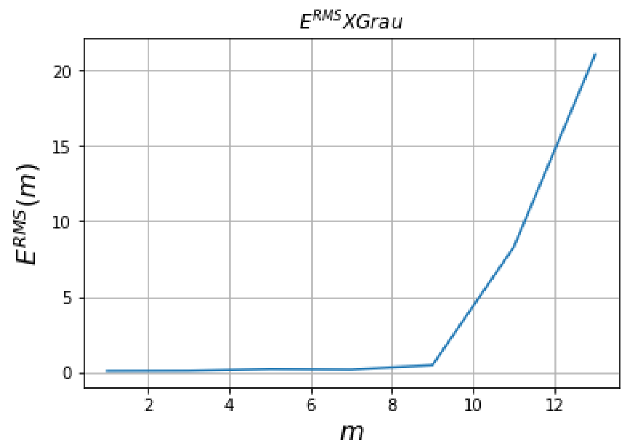


Figure 5: E^{RMS} das curvas encontradas durante o treinamento com os respectivos valores de m

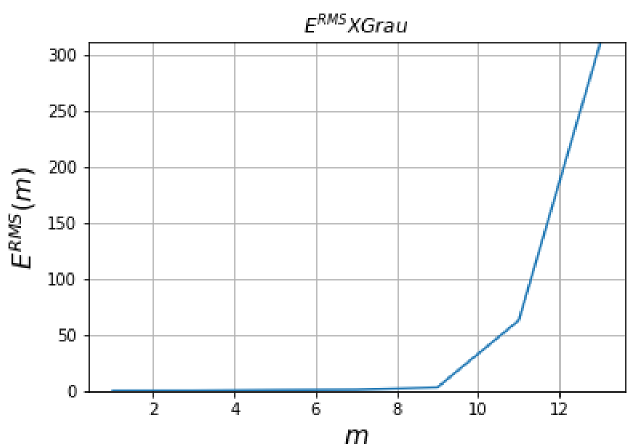


Figure 6: E^{RMS} das curvas encontradas durante o teste com os respectivos valores de m

polinômios encontrados, é necessário mostrar os erros das curvas com os dados de teste como mostrado na **Figura 6**.

Pode-se perceber , na **Figura 6**, que a partir de valores maiores que 9 o grau do polinômio começa a pirorar drasticamente a qualidade do ajuste. Os valores ótimos se encontram entre 1 e 8.

III. Aumentando o número de exemplares

O objetivo dessa seção é mostrar os efeitos no aprendizado e no teste ao se analisar um número maior de dados. Assim como proposto pelo professor, o número de exemplares foi aumentado de 14 para 200 exemplares, mantendo a mesma divisão anterior para o treinamento e para o teste. Da mesma forma como foi adicionado um valor aleatório no exemplo anterior, será também adicionado aqui: Novamente, através

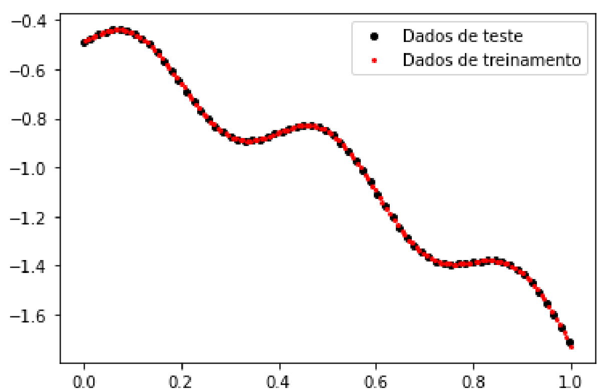


Figure 7: Valores de teste em escuro e de treinamento em vermelho

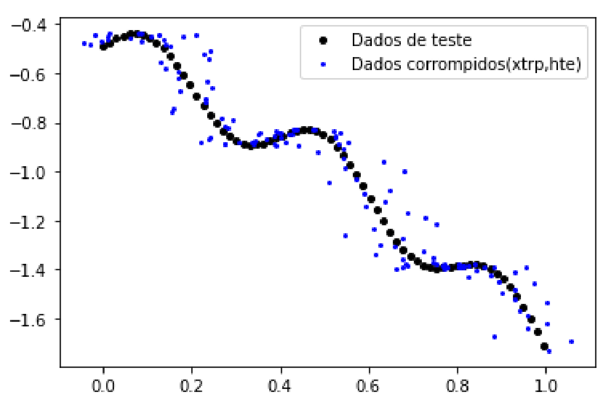


Figure 8: Valores de teste em escuro e valores aleatórios em azul

dos valores corrompidos(aleatórios), é feito o ajuste para os polinômios de grau $m = 1$, $m = 3$, $m = 5$, $m = 7$, $m = 9$, $m = 11$ e $m = 13$. Na **Figura 9** e **Figura 10**, pode-se ver os ajustes com os valores de treino e teste respectivamente.

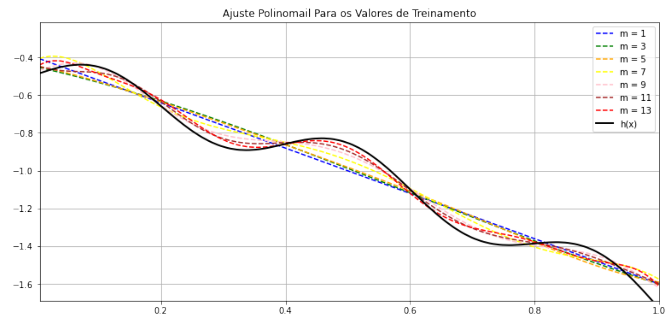


Figure 9: Curva original(em preto) e curvas ajustadas do treinamento

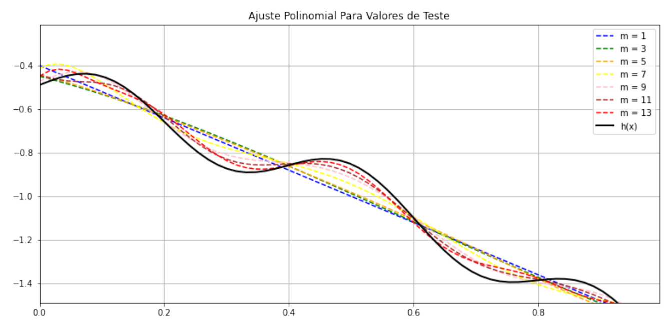


Figure 10: Curva original(em preto) e curvas ajustadas de teste

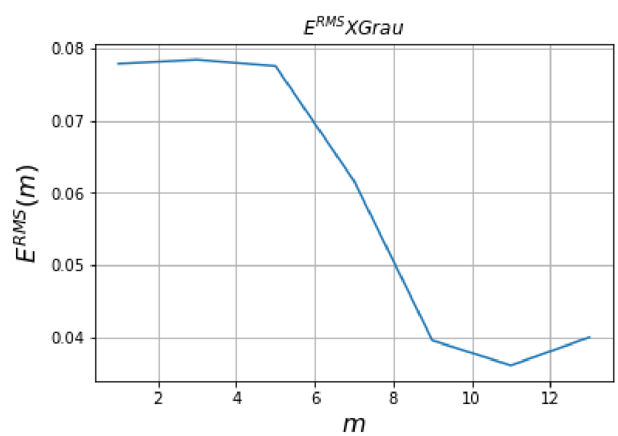


Figure 11: Curva original(em preto) e curvas ajustadas de teste

Posteriomente, foi calculado o valor do erro das cruvas durante o treinamento e durante o teste, como mostrado nas **Figura 11** e **Figura 12** respectivamente. Como pode ser visto, as curvas da **Figura 9** e **Figura 10** são ,visualmente, identicas ou muito parecidas. Isso se deve ao grande número de pontos de treinamento relativamente próximos aos pontos de teste. Além disso, com o acréscimo de dados, as melhores curvas de ajuste, tanto durante o treinamento quanto no

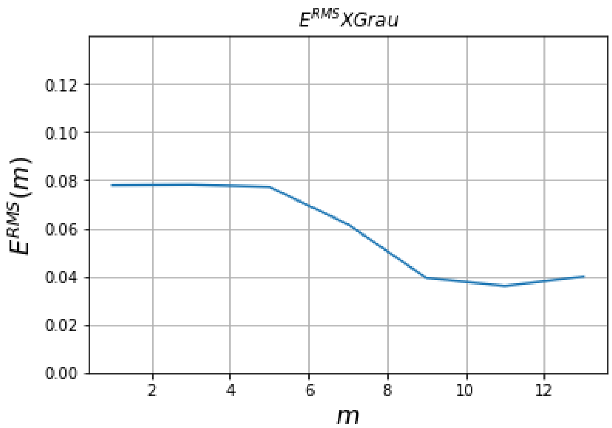


Figure 12: Curva original(em preto) e curvas ajustadas de teste

treino, possuiram um grau maior do que as do exemplo anterior com 21 exemplares.

IV. Conclusão

Após o término do trabalho, pode-se concluir que a técnica de ajuste de curvas polinomiais é essencial para o entendimento das aplicações de inteligência artificial, sendo um ótimo primeiro contato com a área. Este trabalho foi importante para construção de conhecimento dos envolvidos, pois aprimorou a capacidade de utilizar técnicas estatísticas a favor da construção de polinômios, separar variáveis para treino e teste, cálculo de erro e plotagem de gráficos. Além disso, foi possível uma aprendizagem mais prática das linguagens de programação em situações mais diversas.

V. Referências

https://numpy.org/, https://matplotlib.org,https://numpy.org/
Livro: Neural Networks for Pattern Recognition

VI. Apêndice

Todos os códigos e gráficos foram colocados em forma de apêndice no final deste trabalho. Além disso, o documento, de autoria do professor Pedro Abrão, que mostra os passos a serem seguidos pelos alunos para a realização da tarefa também foram colocados no final do documento. Os códigos utilizados podem ser encontrados no repositório do GitHub no seguinte link: github.com/VictorCalebeCavalcante/Ajuste_Polinomial_De_Curva_IA

Ajuste Ponimial de Curva (21 exemplares)

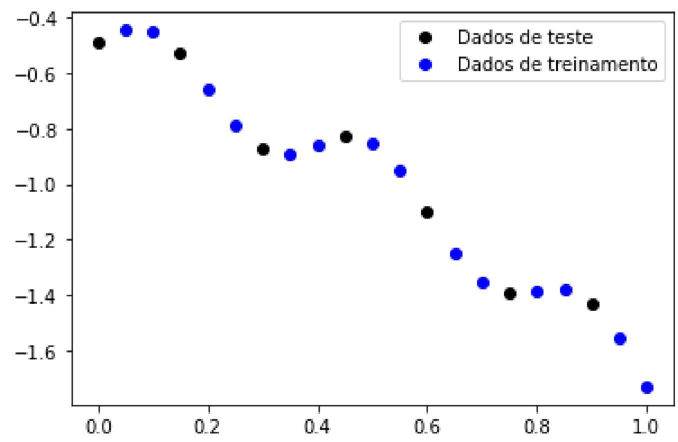
Alunos : Victor Calebe e João Victor Rocha Muhamad

Divisão Dos Exempalres em Set de treinamento e Set de Teste:

Nessa estapa, dividimos 2/3 dos dados para um vetor de treinamento e 1/3 para teste. Foi proposto pelo professor que fizessemos essa divisão de forma alternada, ou seja, a cada 3 elementos do vetor 2 serão para treino e 1 será para teste.

```
%matplotlib inline
from pylab import plot, grid, xlabel, ylabel, title, figure, subplot, tight_layout, axis, legend, text, xli
import numpy as np

exemplares = 21
h = lambda x: -0.85*np.exp(0.9*x) + 0.36 + 0.12*np.sin(5*np.pi*x)
x = np.round(np.linspace(0,1,exemplares),5)

xte = []
xtr = []

for i in range(0,len(x),3):
  xte.append(x[i])
  xtr.append(x[i+1]),xtr.append(x[i+2])

xte = np.array(xte)
xtr = np.array(xtr)

hte = h(xte)
htr = h(xtr)

plot(xte,hte,'o',color = 'black')
plot(xtr,htr,'o',color = 'blue')
legend(['Dados de teste','Dados de treinamento'],)
```

```
<matplotlib.legend.Legend at 0x7fde1730cc90>
```



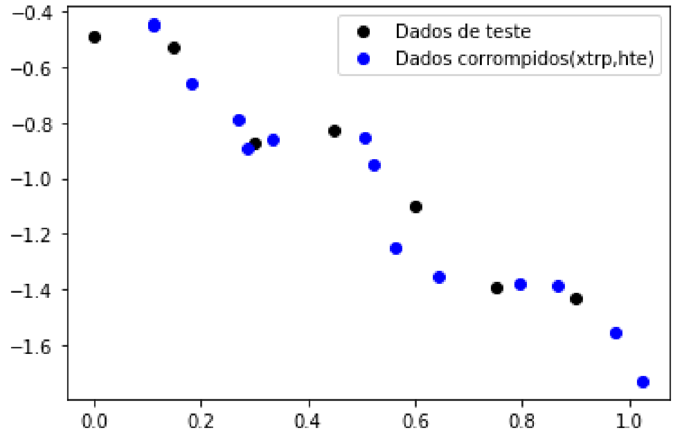
Geração de Valores Aleatórios de *Distribuição Gaussiana*:

Adição do "noise" aos valores de "xtr".

```
mu, sigma = 0, 0.05
noise = np.random.normal(mu,sigma,14)
xtrp = noise + xtr
```

```
plot(xte,hte,'o',color = 'black')
plot(xtrp,htr,'o',color = 'blue')
legend(['Dados de teste','Dados corrompidos(xtrp,hte)'],)
```

```
<matplotlib.legend.Legend at 0x7fde16e01450>
```



Treinamento:

Nessa etapa, treinamos as curvas para ficarem melhor ajustadas aos pontos de "xtrp".

```
%matplotlib inline
from numpy import linspace, zeros, pi, cos, sin, exp
from pylab import plot, grid, xlabel, ylabel, title, figure, subplot, tight_layout, axis, legend, text, xli

peso1_tr = np.polyfit(xtrp,htr,1)#Pesos da curva_1
peso3_tr = np.polyfit(xtrp,htr,3)#Pesos da curva_3
peso5_tr = np.polyfit(xtrp,htr,5)#Pesos da curva_5
peso7_tr = np.polyfit(xtrp,htr,7)#Pesos da curva_7
peso9_tr = np.polyfit(xtrp,htr,9)#Pesos da curva_9
peso11_tr = np.polyfit(xtrp,htr,11)#Pesos da curva_11
peso13_tr = np.polyfit(xtrp,htr,13)#Pesos da curva_13
```

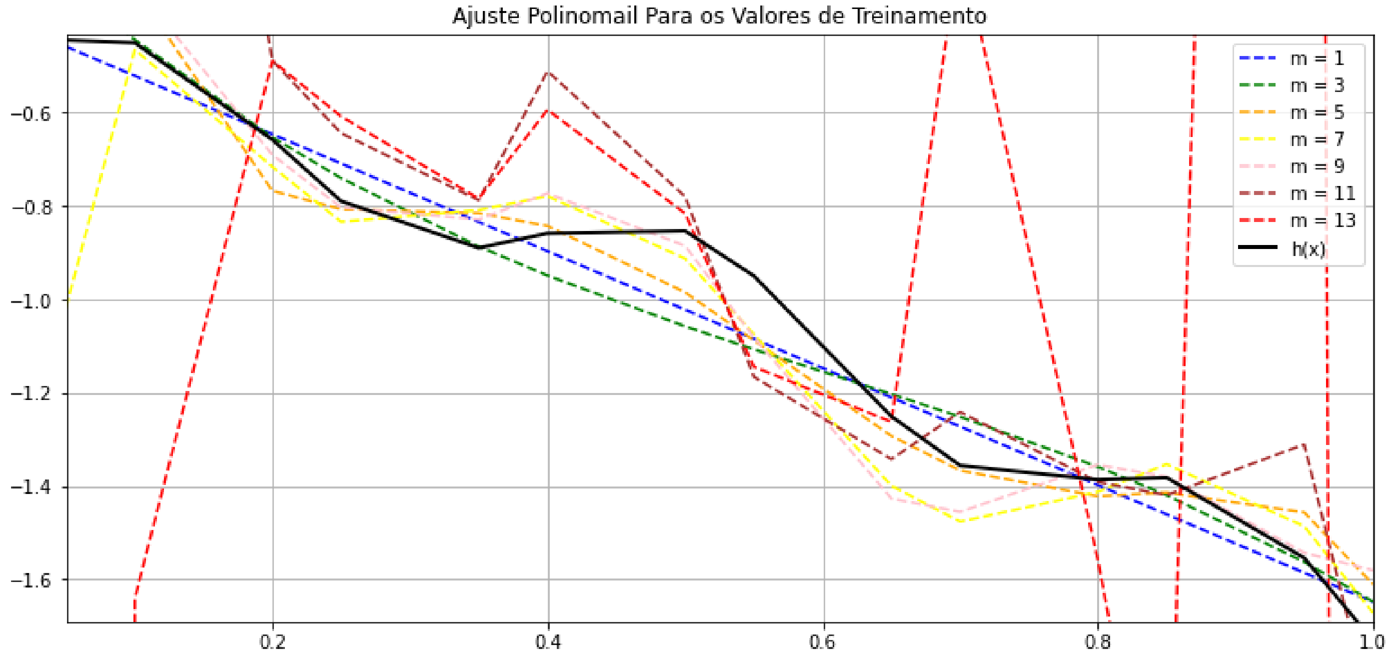
Plotagem de gráfico(treinamento):

Nessa seção vamos plotar os gráficos dos ajustes obtidos na etapa anterior ao lado da curva original para uma melhor análise visual.

```
ajuste1_tr = np.polyval(peso1_tr,xtr)#Cuva ajustada com os pontos
ajuste3_tr = np.polyval(peso3_tr,xtr)#...
ajuste5_tr = np.polyval(peso5_tr,xtr)#...
ajuste7_tr = np.polyval(peso7_tr,xtr)#...
ajuste9_tr = np.polyval(peso9_tr,xtr)#...
ajuste11_tr = np.polyval(peso11_tr,xtr)#...
ajuste13_tr = np.polyval(peso13_tr,xtr)#Cuva ajustada com os pontos


figure(figsize=(13,6))

plot(xtr, ajuste1_tr,'--',color = 'blue') #Fitting
plot(xtr, ajuste3_tr,'--',color = 'green') #Fitting
plot(xtr, ajuste5_tr,'--', color = 'orange') #Fitting
plot(xtr, ajuste7_tr,'--', color = 'yellow') #Fitting
plot(xtr, ajuste9_tr,'--',color = 'pink') #Fitting
plot(xtr, ajuste11_tr,'--',color = 'brown') #Fitting
plot(xtr, ajuste13_tr,'--',color = 'red') #Fitting
plot(xtr,htr,color = 'black',linewidth=2) # h(x)
```

```
#
grid(True)
#xlabel('$x$',fontsize=16)
#ylabel('$ajuste(x)$',fontsize=16)
title('Ajuste Polinomail Para os Valores de Treinamento')
legend(['m = 1','m = 3', 'm = 5','m = 7', 'm = 9', 'm = 11', 'm = 13','h(x)'],)
xlim(xtr[0],xtr[-1])
ylim(h(xte)[0]-1.2,h(xte)[-1]+1)
#ylim(-4.25,8)
```

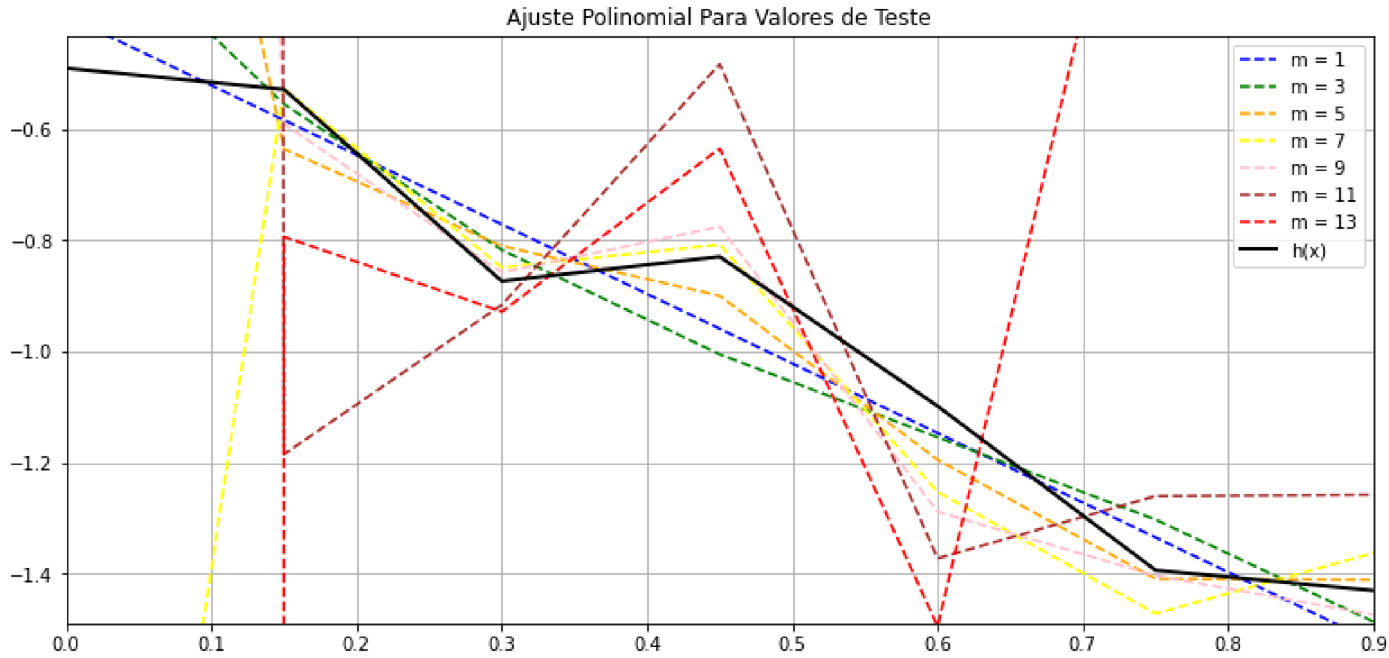
```
(-1.69, -0.430721788675001)
```



Teste e Plotagem das Curvas:

```
ajuste1_te = np.polyval(peso1_tr,xte)
ajuste3_te = np.polyval(peso3_tr,xte)
ajuste5_te = np.polyval(peso5_tr,xte)
ajuste7_te = np.polyval(peso7_tr,xte)
ajuste9_te = np.polyval(peso9_tr,xte)
ajuste11_te = np.polyval(peso11_tr,xte)
ajuste13_te = np.polyval(peso13_tr,xte)

figure(figsize=(13,6))


plot(xte, ajuste1_te,'--',color = 'blue') #Fitting
plot(xte, ajuste3_te,'--',color = 'green') #Fitting
plot(xte, ajuste5_te,'--', color = 'orange') #Fitting
plot(xte, ajuste7_te,'--', color = 'yellow') #Fitting
plot(xte, ajuste9_te,'--', color = 'pink') #Fitting
plot(xte, ajuste11_te,'--', color = 'brown') #Fitting
plot(xte, ajuste13_te,'--', color = 'red') #Fitting
plot(xte,hte,color = 'black',linewidth=2) # correto

grid(True)
#xlabel('$x$',fontsize=16)
#ylabel('$ajuste(x)$',fontsize=16)
title('Ajuste Polinomial Para Valores de Teste')
legend(['m = 1', 'm = 3', 'm = 5', 'm = 7', 'm = 9','m = 11','m = 13','h(x)'])
xlim(xte[0],xte[-1])
ylim(h(xte)[0]-1,h(xte)[-1]+1)
```

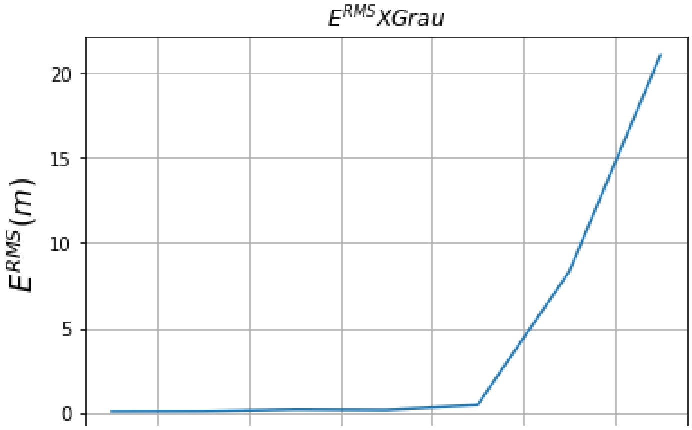
```
(-1.49, -0.430721788675001)
```



Treino e Calculo do Erro :

```
#treino
import math

erms_tr_wx = []
m_tr = [1,3,5,7,9,11,13]

erms_w1 = (ajuste1_tr - htr)**2
erms_tr_wx.append(math.sqrt(sum(erms_w1)/len(xtr)))

erms_w3 = (ajuste3_tr - htr)**2
erms_tr_wx.append(math.sqrt(sum(erms_w3)/len(xtr)))

erms_w5 = (ajuste5_tr - htr)**2
erms_tr_wx.append(math.sqrt(sum(erms_w5)/len(xtr)))

erms_w7 = (ajuste7_tr - htr)**2
erms_tr_wx.append(math.sqrt(sum(erms_w7)/len(xtr)))

erms_w9 = (ajuste9_tr - htr)**2
erms_tr_wx.append(math.sqrt(sum(erms_w9)/len(xtr)))

erms_w11 = (ajuste11_tr - htr)**2
erms_tr_wx.append(math.sqrt(sum(erms_w11)/len(xtr)))

erms_w13 = (ajuste13_tr - htr)**2
erms_tr_wx.append(math.sqrt(sum(erms_w13)/len(xtr)))

plot(m_tr,erms_tr_wx)

grid(True)
xlabel('$m$',fontsize=16)
ylabel('$E^{RMS}(m)$',fontsize=16)
title('$E^{RMS} X Grau$')

print(erms_tr_wx)
```

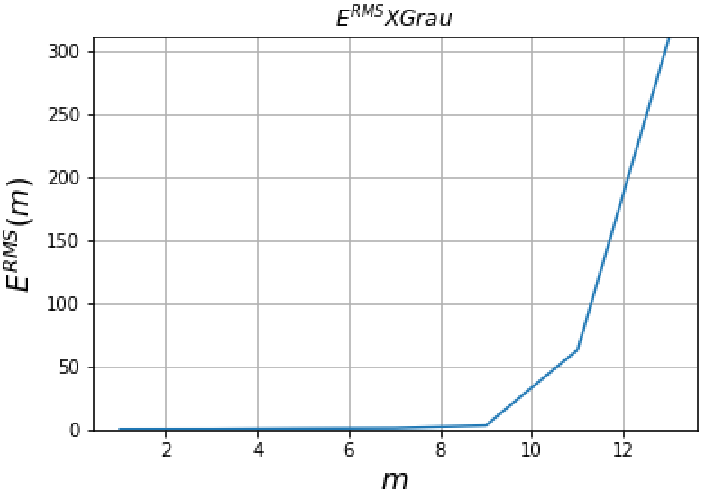
[0.07829927904489294, 0.09078870346679237, 0.20196667112295524, 0.1716926240144812, 0.4662131908076835



Teste e Calculo do erro :

```
erms_te_wx = []
te_wx = [1, 3, 5, 7, 9, 11, 13]

erms_w1 = (ajuste1_te - hte)**2
erms_te_wx.append(math.sqrt(sum(erms_w1)/len(xte)))

erms_w3 = (ajuste3_te - hte)**2
erms_te_wx.append(math.sqrt(sum(erms_w3)/len(xte)))

erms_w5 = (ajuste5_te - hte)**2
erms_te_wx.append(math.sqrt(sum(erms_w5)/len(xte)))

erms_w7 = (ajuste7_te - hte)**2
erms_te_wx.append(math.sqrt(sum(erms_w7)/len(xte)))

erms_w9 = (ajuste9_te - hte)**2
erms_te_wx.append(math.sqrt(sum(erms_w9)/len(xte)))

erms_w11 = (ajuste11_te - hte)**2
erms_te_wx.append(math.sqrt(sum(erms_w11)/len(xte)))

erms_w13 = (ajuste13_te - hte)**2
erms_te_wx.append(math.sqrt(sum(erms_w13)/len(xte)))

plot(te_wx, erms_te_wx)

grid(True)
xlabel('$m$',fontsize=16)
ylabel('$E^{RMS}(m)$',fontsize=16)
title('$E^{RMS} X Grau$')
#xlim(0,13)
ylim(0,erms_te_wx[-1]+1)
print(erms_te_wx)
```

```
[0.08755124926445236, 0.14448627084319945, 0.6531596143309244, 0.99740898617704, 3.15536488088417, 62.
```



Ajuste Ponimial de Curva (200 exemplares)

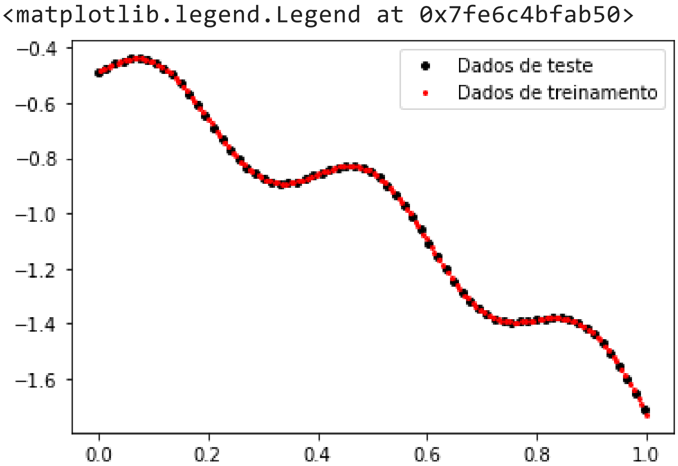
Alunos : Victor Calebe e João Victor Rocha Muhamad

Divisão Dos Exempalres em Set de treinamento e Set de Teste:

Nessa estapa, dividimos 2/3 dos dados para um vetor de treinamento e 1/3 para teste. Foi proposto pelo professor que fizessemos essa divisão de forma alternada, ou seja, a cada 3 elementos do vetor 2 serão para treino e 1 será para teste.

```
%matplotlib inline
from pylab import plot, grid, xlabel, ylabel, title, figure, subplot, tight_layout, axis, legend, text, xli
import numpy as np

exemplares = 200
h = lambda x: -0.85*np.exp(0.9*x) + 0.36 + 0.12*np.sin(5*np.pi*x)
x = np.round(np.linspace(0,1,exemplares),5)

xte = []
xtr = []

for i in range(0,len(x),3):
  try:
    xte.append(x[i])
    xtr.append(x[i+1]),xtr.append(x[i+2])
  except: print('')

xte = np.array(xte)
xtr = np.array(xtr)

hte = h(xte)
htr = h(xtr)

plot(xte,hte,'o',color = 'black',markersize=4)
plot(xtr,htr,'o',color = 'red',markersize=2)


legend(['Dados de teste','Dados de treinamento'],)
```

```
<matplotlib.legend.Legend at 0x7fe6c4bfab50>
```

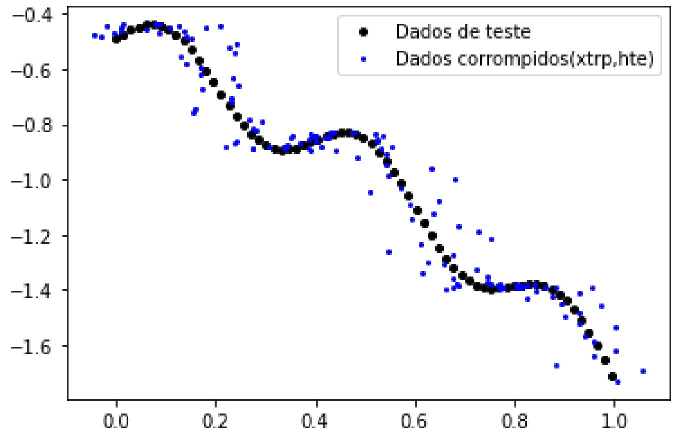


Geração de Valores Aleatórios de *Distribuição Gaussiana*:

Adição do "noise" aos valores de "xtr".

```
mu, sigma = 0, 0.05
noise = np.random.normal(mu,sigma,len(xtr))
xtrp = noise + xtr

plot(xte,hte,'o',color = 'black',markersize=4)
plot(xtrp,htr,'o',color = 'blue',markersize=2)
legend(['Dados de teste','Dados corrompidos(xtrp,hte)'],)
```

```
<matplotlib.legend.Legend at 0x7fe6c4bfb650>
```



Treinamento:

Nessa etapa, treinamos as curvas para ficarem melhor ajustadas aos pontos de "xtrp".

```
%matplotlib inline
from numpy import linspace, zeros, pi, cos, sin, exp
from pylab import plot, grid, xlabel, ylabel, title, figure, subplot, tight_layout, axis, legend, text, xli

peso1_tr = np.polyfit(xtrp,htr,1)#Pesos da curva_1
peso3_tr = np.polyfit(xtrp,htr,3)#Pesos da curva_3
peso5_tr = np.polyfit(xtrp,htr,5)#Pesos da curva_5
peso7_tr = np.polyfit(xtrp,htr,7)#Pesos da curva_7
peso9_tr = np.polyfit(xtrp,htr,9)#Pesos da curva_9
peso11_tr = np.polyfit(xtrp,htr,11)#Pesos da curva_11
peso13_tr = np.polyfit(xtrp,htr,13)#Pesos da curva_13
```

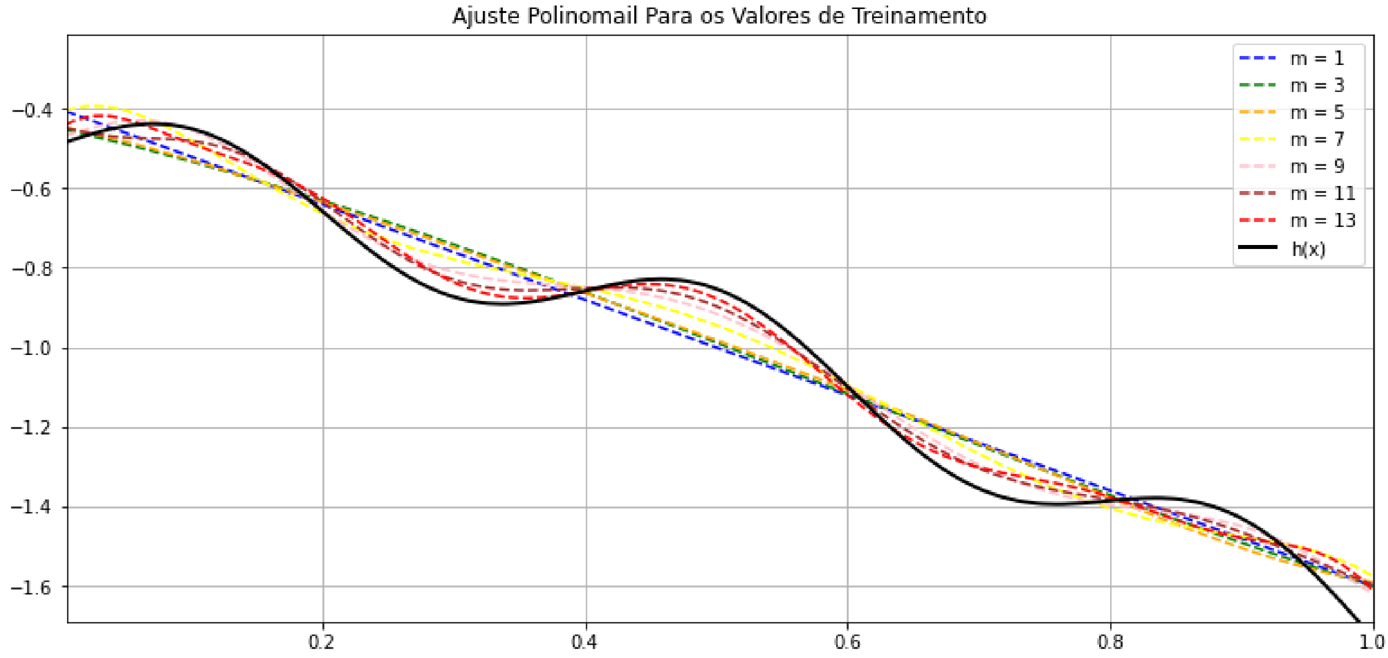
Plotagem de gráfico(treinamento):

Nessa seção vamos plotar os gráficos dos ajustes obtidos na etapa anterior ao lado da curva original para uma melhor análise visual.

```
ajuste1_tr = np.polyval(peso1_tr,xtr)#Cuva ajustada com os pontos
ajuste3_tr = np.polyval(peso3_tr,xtr)#...
ajuste5_tr = np.polyval(peso5_tr,xtr)#...
ajuste7_tr = np.polyval(peso7_tr,xtr)#...
ajuste9_tr = np.polyval(peso9_tr,xtr)#...
ajuste11_tr = np.polyval(peso11_tr,xtr)#...
ajuste13_tr = np.polyval(peso13_tr,xtr)#Cuva ajustada com os pontos


figure(figsize=(13,6))

plot(xtr, ajuste1_tr,'--',color = 'blue') #Fitting
plot(xtr, ajuste3_tr,'--',color = 'green') #Fitting
plot(xtr, ajuste5_tr,'--', color = 'orange') #Fitting
```

```
plot(xtr, ajuste7_tr,'--', color = 'yellow') #Fitting
plot(xtr, ajuste9_tr,'--',color = 'pink') #Fitting
plot(xtr, ajuste11_tr,'--',color = 'brown') #Fitting
plot(xtr, ajuste13_tr,'--',color = 'red') #Fitting
plot(xtr,htr,color = 'black',linewidth=2) # h(x)
#
grid(True)
#xlabel('$x$',fontsize=16)
#ylabel('$ajuste(x)$',fontsize=16)
title('Ajuste Polinomail Para os Valores de Treinamento')
legend(['m = 1','m = 3', 'm = 5','m = 7', 'm = 9', 'm = 11', 'm = 13','h(x)'],)
xlim(xtr[0],xtr[-1])
ylim(h(xte)[0]-1.2,h(xte)[-1]+1.5)
#ylim(-4.25,8)
```

```
(-1.69, -0.21174814035134482)
```

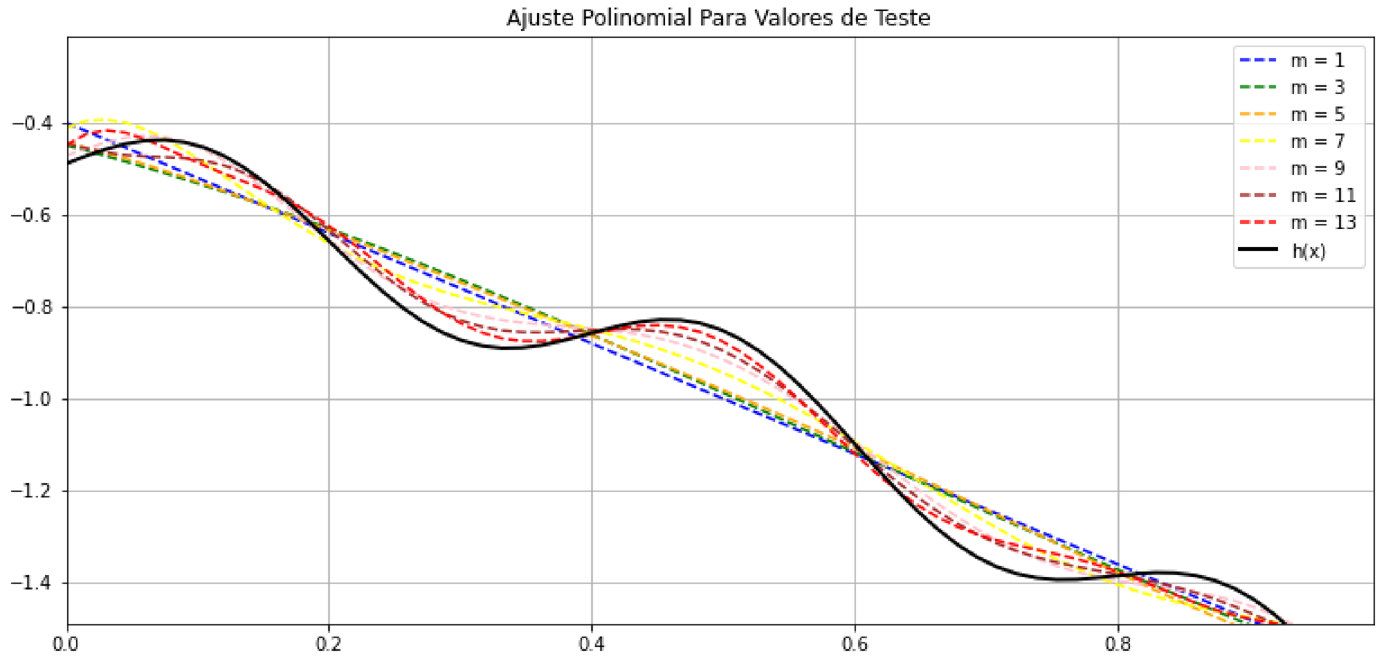


Teste e Plotagem das Curvas:

```
ajuste1_te = np.polyval(peso1_tr,xte)
ajuste3_te = np.polyval(peso3_tr,xte)
ajuste5_te = np.polyval(peso5_tr,xte)
ajuste7_te = np.polyval(peso7_tr,xte)
ajuste9_te = np.polyval(peso9_tr,xte)
ajuste11_te = np.polyval(peso11_tr,xte)
ajuste13_te = np.polyval(peso13_tr,xte)

figure(figsize=(13,6))


plot(xte, ajuste1_te,'--',color = 'blue') #Fitting
plot(xte, ajuste3_te,'--',color = 'green') #Fitting
plot(xte, ajuste5_te,'--', color = 'orange') #Fitting
plot(xte, ajuste7_te,'--', color = 'yellow') #Fitting
plot(xte, ajuste9_te,'--', color = 'pink') #Fitting
plot(xte, ajuste11_te,'--', color = 'brown') #Fitting
plot(xte, ajuste13_te,'--', color = 'red') #Fitting
plot(xte,hte,color = 'black',linewidth=2) # correto

grid(True)
#xlabel('$x$',fontsize=16)
#ylabel('$ajuste(x)$',fontsize=16)
title('Ajuste Polinomial Para Valores de Teste')
```

```
legend(['m = 1', 'm = 3', 'm = 5', 'm = 7', 'm = 9','m = 11','m = 13','h(x)'])
xlim(xte[0],xte[-1])
ylim(h(xte)[0]-1,h(xte)[-1]+1.5)
```

```
(-1.49, -0.21174814035134482)
```

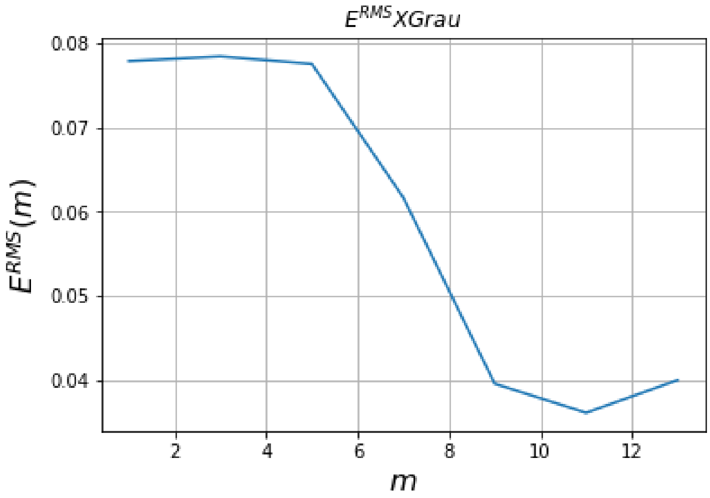


Treino e Calculo do Erro :

```
#treino
import math

erms_tr_wx = []
m_tr = [1,3,5,7,9,11,13]

erms_w1 = (ajuste1_tr - htr)**2
erms_tr_wx.append(math.sqrt(sum(erms_w1)/len(xtr)))

erms_w3 = (ajuste3_tr - htr)**2
erms_tr_wx.append(math.sqrt(sum(erms_w3)/len(xtr)))

erms_w5 = (ajuste5_tr - htr)**2
erms_tr_wx.append(math.sqrt(sum(erms_w5)/len(xtr)))

erms_w7 = (ajuste7_tr - htr)**2
erms_tr_wx.append(math.sqrt(sum(erms_w7)/len(xtr)))

erms_w9 = (ajuste9_tr - htr)**2
erms_tr_wx.append(math.sqrt(sum(erms_w9)/len(xtr)))

erms_w11 = (ajuste11_tr - htr)**2
erms_tr_wx.append(math.sqrt(sum(erms_w11)/len(xtr)))

erms_w13 = (ajuste13_tr - htr)**2
erms_tr_wx.append(math.sqrt(sum(erms_w13)/len(xtr)))

plot(m_tr,erms_tr_wx)

grid(True)
xlabel('$m$',fontsize=16)
ylabel('$E^{RMS}(m)$',fontsize=16)
title('$E^{RMS} X Grau$')
```

```
print(erms_tr_wx)
```

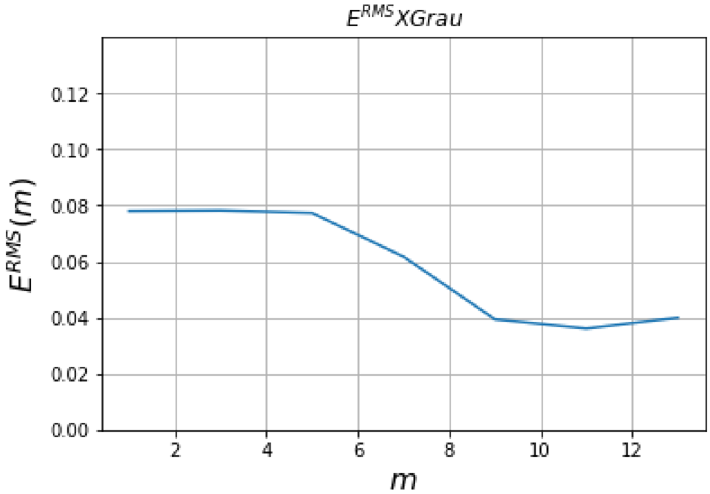
```
[0.07788712930502126, 0.07844299417926184, 0.07755736360516553, 0.061669468771162535, 0.03953812226395
```



Teste e Calculo do erro :

```
erms_te_wx = []
te_wx = [1, 3, 5, 7, 9, 11, 13]

erms_w1 = (ajuste1_te - hte)**2
erms_te_wx.append(math.sqrt(sum(erms_w1)/len(xte)))

erms_w3 = (ajuste3_te - hte)**2
erms_te_wx.append(math.sqrt(sum(erms_w3)/len(xte)))

erms_w5 = (ajuste5_te - hte)**2
erms_te_wx.append(math.sqrt(sum(erms_w5)/len(xte)))

erms_w7 = (ajuste7_te - hte)**2
erms_te_wx.append(math.sqrt(sum(erms_w7)/len(xte)))

erms_w9 = (ajuste9_te - hte)**2
erms_te_wx.append(math.sqrt(sum(erms_w9)/len(xte)))

erms_w11 = (ajuste11_te - hte)**2
erms_te_wx.append(math.sqrt(sum(erms_w11)/len(xte)))

erms_w13 = (ajuste13_te - hte)**2
erms_te_wx.append(math.sqrt(sum(erms_w13)/len(xte)))

plot(te_wx, erms_te_wx)

grid(True)
xlabel('$m$',fontsize=16)
ylabel('$E^{RMS}(m)$',fontsize=16)
title('$E^{RMS} X Grau$')
#xlim(0,13)
ylim(0,erms_te_wx[-1]+0.1)
print(erms_te_wx)
```

[0.0779891074020805, 0.07819012540213215, 0.07731318429936222, 0.061730091091244985, 0.039360014031856



INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE GOIÁS

ENGENHARIA DE CONTROLE E AUTOMAÇÃO

TRABALHO DE INTELIGÊNCIA ARTIFICIAL

Fevereiro de 2021

DESCRIÇÃO

Este trabalho destina-se ao conteúdo de Ajuste de Curvas Polinomial, sendo objeto de análise a Complexidade do Modelo e sua influência na Capacidade de Generalização (observação de dados novos). Os passos para execução do trabalho são os seguintes:

1) Utilizar a função geradora ***h(x)= - 0.85*exp(0.90*x) + 0.36 + 0.12*sen(5*pi*x)***

2) Gerar inicialmente um conjunto de dados, atribuindo valores a *x* ***(sugestão: x=(0.0:0.05:1.0)***, com **N** exemplares, neste caso 21 exemplares.

3) Separar os dados gerados em dois conjuntos: Treinamento (2/3 dos exemplares – ***xtr e htr***) e Teste (1/3 dos exemplares – ***xte e hte***).

4) A cada ponto de dado (valor de x) do **Conjunto de Treinamento** (***xtr***) ser acrescentado um ruído gaussiano com média zero e desvio padrão igual a 0,05, obtendo um novo Conjunto de Treinamento ***xtrp***.
   (a) - Para gerar números aleatórios com distribuição gaussiana, utilize a seguinte equação: r = 0.0 + 0.05*randn(N,1), onde 0.0 é média, 0.05 é o desvio padrão. A função *randn* é a função do MATLAT que possibilita gerar números aleatórios normalmente distribuídos. O valor de N é o número de exemplares que correspondem a 2/3 de ***xtr***. Logo, ***xtrp=xtr+r***.

5) Para obtenção dos parâmetros livres **W*** (parâmetros ótimos), através do conjunto de treinamento, deve ser utilizada a função POLYFIT(***xtrp***,***htr***,**M**), onde **M** é a ordem do polinômio, obtendo-se $y(xtrp)= w_1x^m + w_2x^{m-1} +...+ w_mx + w_{m+1}$. A função POLYFIT retorna um vetor com os parâmetros dos livres do polinômio (ex.: m=1, vetor $\mathbf{W}^*$<w_1 w_0>, m=3, $\mathbf{W}^*$<w_4 w_3 w_2 w_1 w_0>).

6) Fazer simulações para m=1, m=3, m=5, m=7, m=9 e m=11 e superior, se necessário.

7) Uma vez obtido **W*** para cada ordem, plotar, para cada ordem, a curva <u>***htr***/*y(xtrp)*</u> (curva ajustada para os dados de treinamento). Use a função POLYVAL para ***testar*** o polinômio obtido. Para efeito de análise deve ser também plotado <u>***hte***/*y(xte)*</u>’ (curva da função ajustada do conjunto de teste).

8) Computar os E^{RMS} (*root-mean-square error*) para o conjunto de treinamento e para o conjunto de teste. Vale ressaltar que ***htr*** e ***hte*** são os valores desejados e *y(xtrp)* e *y(xte)* são os valores calculados. Comparar, para cada grau de polinômio simulado, o erro em função dos graus do polinômio (curva E^{RMS} x grau do polinômio).

9) Fazer as análises necessárias quanto a relação da complexidade do modelo pela capacidade de generalização. Escolher o melhor grau do polinômio que ajusta os dados. Utilizar o Cap. 1, item 1.5 do livro '*Neural Networks for Pattern Recognition*', Christopher M. Bishop, como referência para o trabalho.

10) Repetir a operação(itens 2 ao 9) para **N** = 200 exemplares

11) Utilizar a Regularização para a melhor equação ajustada (melhor valor de M) para este conjunto com 200 exemplares. Utilize 3 valores de λ (slides).

12) Apresentar o trabalho escrito com os resultados no formato *paper* (duas colunas)

13) O trabalho pode ser feito em dupla (somente!!)