Professor: Rogério Martins Gomes

Alunos: Aritana Noara Costa Santos e Victor Augusto Januário da Cruz

1. Regressão linear com uma variável

In [461…

```
![image.png](attachment:image.png)
```

```
/bin/bash: -c: linha 0: erro de sintaxe próximo ao token inesperado `attachment:image.png'
/bin/bash: -c: linha 0: `[image.png](attachment:image.png)'
```

1) Coletando o arquivo data1.txt:

In [462…

```
import pandas as pd
import numpy as np

filename = '/home/aritana/my_jupyter_notebook/linearRegressionOneMultipleVariables/data1.txt'


data = pd.read_csv(filename,sep=',',header=None).values


#definindo os eixos
x = data[:, 0]
y = data[:, 1]

print(pd.read_csv(filename,sep=',',header=None))

#fonte:http://awesci.com/reading-and-plotting-data-in-jupyter-notebook/
```

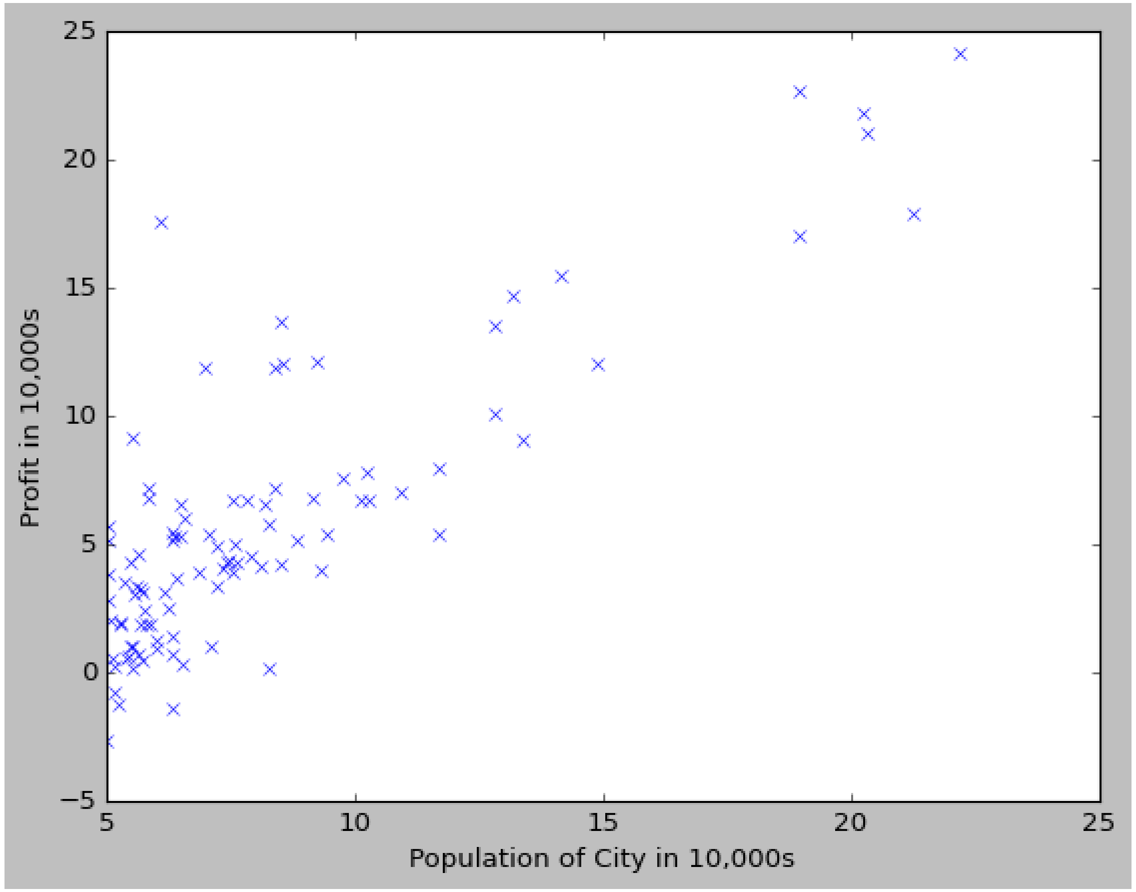
```
         0         1
0   6.1101  17.59200
1   5.5277   9.13020
2   8.5186  13.66200
3   7.0032  11.85400
4   5.8598   6.82330
..     ...       ...
92  5.8707   7.20290
93  5.3054   1.98690
```

```
94   8.2934   0.14454
95  13.3940   9.05510
96   5.4369   0.61705

[97 rows x 2 columns]
```

1) Plotando os dados:

In [463…

```
import matplotlib.pyplot as plt

plt.xlabel("Population of City in 10,000s")
plt.ylabel("Profit in 10,000s")
plt.plot(x,y,'x')
plt.show()
```



1.3 Gradiente descendente

Nesta parte, você deverá calcular os parâmetros da regressão linear e a função custo para o conjunto de dados, utilizando uma taxa de aprendizado de 0,01.

- Plote a função custo em relação ao número de iterações para ver o seu decaimento;
- Depois que terminar de calcular os parâmetros da regressão, traçar o ajuste linear, conforme a figura a seguir.

Fundamentação teórica:

a) Objetivo: minimizar função custo (Erro médio quadrático)

Hipótese: $h_\theta(x) = \theta_0 + \theta_1 x$

Parâmetros: θ_0, θ_1

Função Custo: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

Objetivo: $\underset{\theta_0, \theta_1}{\text{minimizar}} \; J(\theta_0, \theta_1)$

b) Algoritimo:

Algoritmo: Gradiente Descendente

repetir até convergir $\{$

$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ $\quad$ $(\text{for } j = 0 \text{ and } j = 1)$

$\}$ $\quad$ $\alpha \rightarrow$ *Taxa de apredizagem*

Correto: Atualização Simultânea	Incorreto:
$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$	$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$
$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$	$\theta_0 := \text{temp0}$
$\theta_0 := \text{temp0}$	$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$
$\theta_1 := \text{temp1}$	$\theta_1 := \text{temp1}$

c) Definição de alfa e das derivadas da função custo:

Alfa:

- Se α for muito pequeno: convergência Lenta.
- Se α for muito grande: $J(\theta)$ poderia não decrescer em cada iteração, ou mesmo, não convergir.

Para escolher α, tente:

…, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, …

derivadas:

$$\frac{\partial}{\partial\theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial\theta_j} \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

$$= \frac{\partial}{\partial\theta_j} \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)^2$$

$$j = 0 : \frac{\partial}{\partial\theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

$$j = 1 : \frac{\partial}{\partial\theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x^{(i)}$$

In [464…

```
#start with some Teta0 and Teta1, say (0,0)
#keep changing  Teta0 and Teta1, to reduce J(Teta0,Teta1) until end up at a minimum
#Update Teta0 and Teta1 simultaneously
```

Derivadas da função custo em relação aos parâmetros

In [465…

```
import matplotlib.pyplot as plt


def Dj_dteta0(teta0, teta1, xi, yi, m):
    resultado = 0
    for i in range (m):
        resultado = resultado + (teta0 + teta1 * xi[i] - yi[i])
    return resultado/m

def Dj_dteta1(teta0, teta1, xi, yi, m):
    resultado = 0
    for i in range (m):
        resultado = resultado + (teta0 + teta1 * xi[i] - yi[i])* xi[i]
    return resultado/m

def FuncaoCusto(teta0, teta1, xi, yi, m):
    resultado = 0
    custo = []
    for i in range (m):
        custo.append(((teta0 + teta1 * xi[i]) - yi[i])**2)
    for i in range (m):
        resultado = resultado + custo[i]
    return resultado/(2*m)

def plotGrafico(x,y,h):
    plt.xlabel("Population of City in 10,000s")
    plt.ylabel("Profit in 10,000s")
    plt.plot(x,y,'x',label="Training data")
    plt.plot(x,h,label="Linear regression")
    plt.legend(loc='upper left', frameon=False)
    plt.show()

def plotGraficoFuncaoCusto(teta1,custo):
    plt.grid()
    plt.xlabel("Teta1")
    plt.ylabel("J ( Tteta1 )")
    plt.plot(teta1,custo, label= "Teta1")
    plt.legend(loc='upper left', frameon=False)
    plt.show()
```

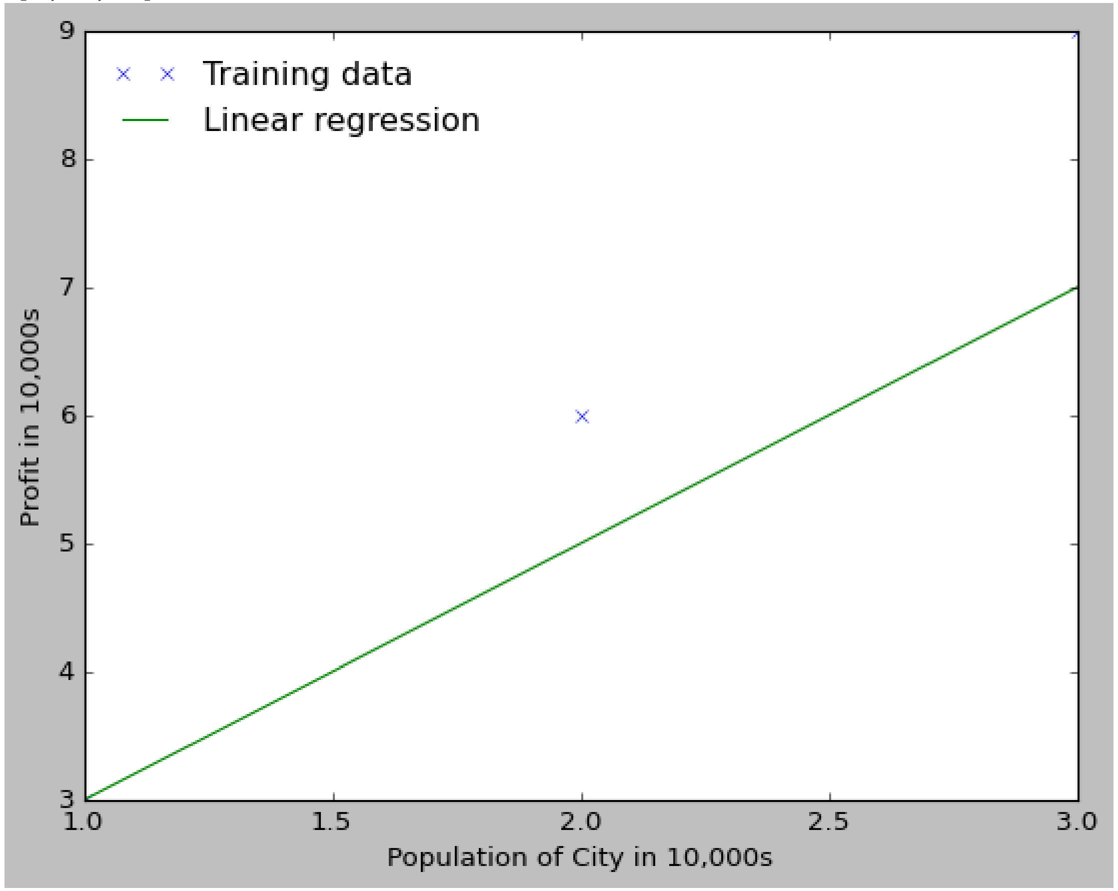
Testes das funções declaradas anteriormente.

In [466…

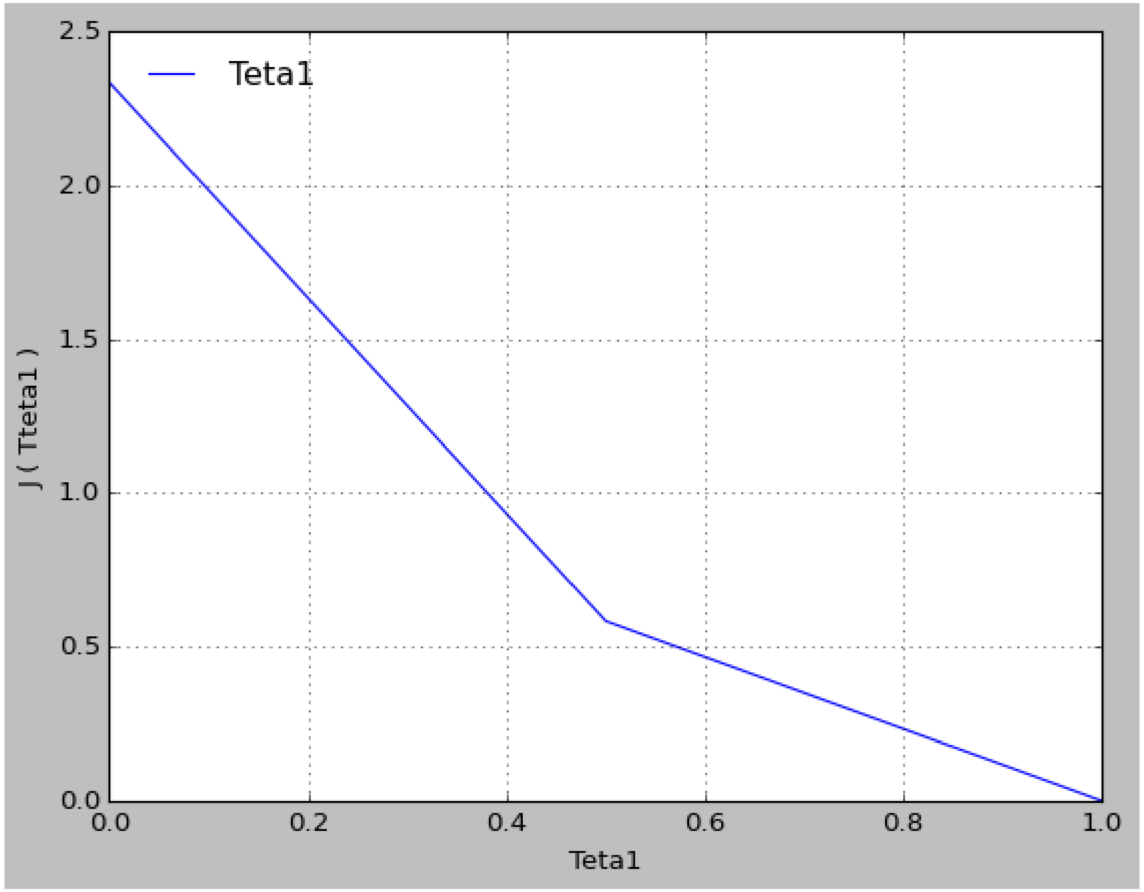
```
# ESTE ARQUIVO É APENAS PARA TESTE UNITARIO
teta0 = 2
teta1 = 9
xi = [1,2,3]
yi = [3, 6, 9]
m = 3
alfa = 0.01

#Teste das derivadas

dj_dteta0 = Dj_dteta0(teta0, teta1, xi, yi, m)
print(dj_dteta0)

if(round(dj_dteta0,11)!=14.0):
    print("Erro em dj_dteta0")


dj_dteta1 = Dj_dteta1(teta0, teta1, xi, yi, m)
print(dj_dteta1)

if(round(dj_dteta1,11)!=32.0):
    print("Erro em dj_dteta1")


#Teste da regressao

#funcao 1 + 2x
hi = []
for i in range (m):
    hi.append(1 + 2 * xi[i])

print(hi)
plotGrafico(xi,yi,hi)

#Teste da funcao custo
xi = [1,2,3]
yi = [1,2,3]
m=3
a =FuncaoCusto(0, 0, xi, yi, m)
print("funcao custo para teta = 0:",a)

#Teste plotar funcao custo
```

```
teta1=[1, 0.5, 0]
custo = []
for i in range (m):
    custo.append(FuncaoCusto(0, teta1[i], xi, yi, m))
    print("teta, custo:",teta1[i],custo[i])
plotGraficoFuncaoCusto(teta1,custo)
```

```
14.0
32.0
[3, 5, 7]
```




```
funcao custo para teta = 0: 2.3333333333333335
teta, custo: 1 0.0
teta, custo: 0.5 0.5833333333333334
teta, custo: 0 2.3333333333333335
```



In [467…

```
xi = [1,2,3]
yi = [1,2,3]
m=3
a =FuncaoCusto(0, 0, xi, yi, m)
print(a)
```

```
2.3333333333333335
```

Execução

repetir até convergir {

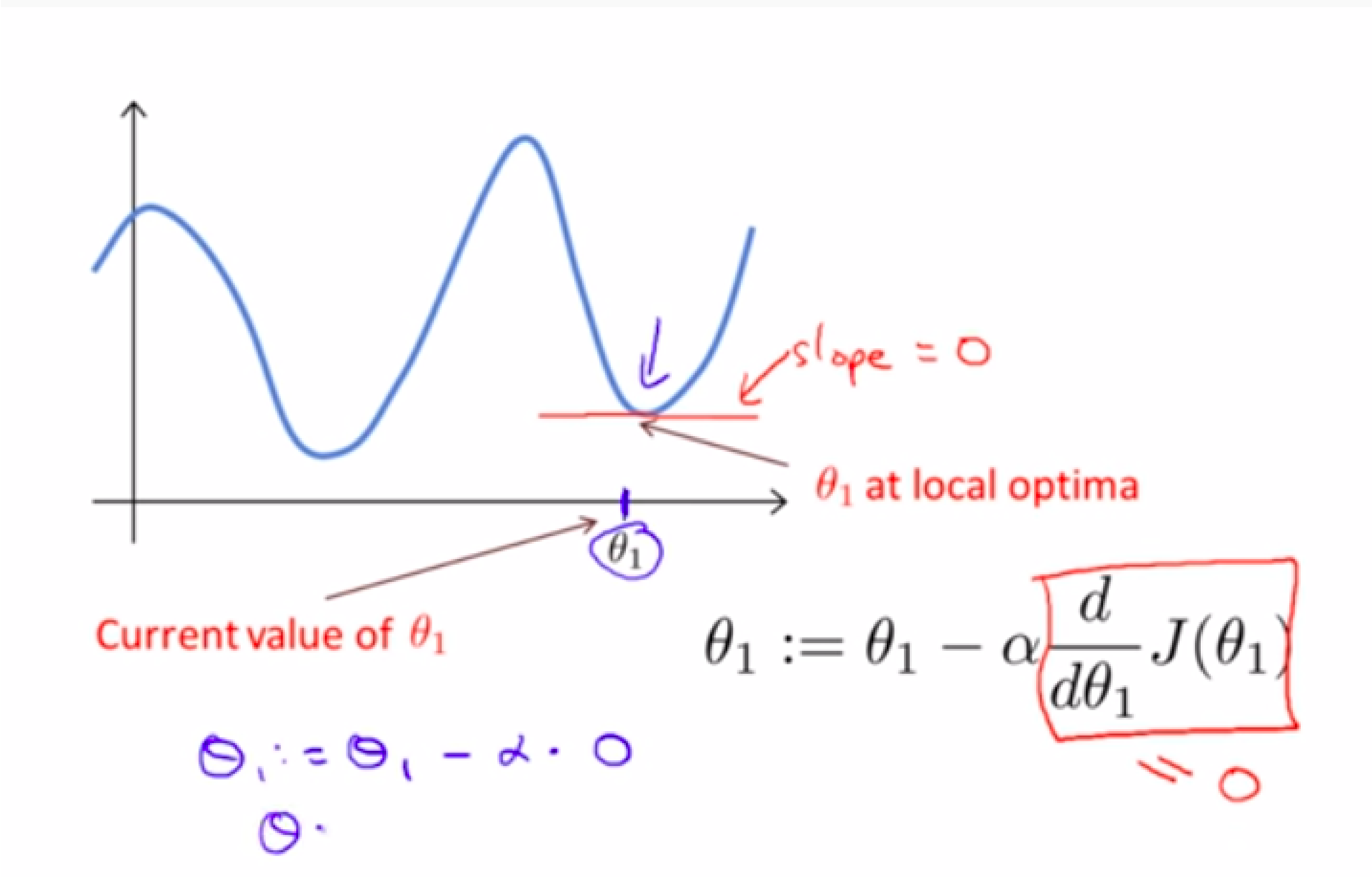
$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}$$

}

Atualize θ_0 e θ_1 simultaneamente

A convergência se dá em um mínimo local ou global onde a derivada se anula e os valores de teta0 e teta1 serão sempre constantes, pois não haverá inclinação (derivada nula).

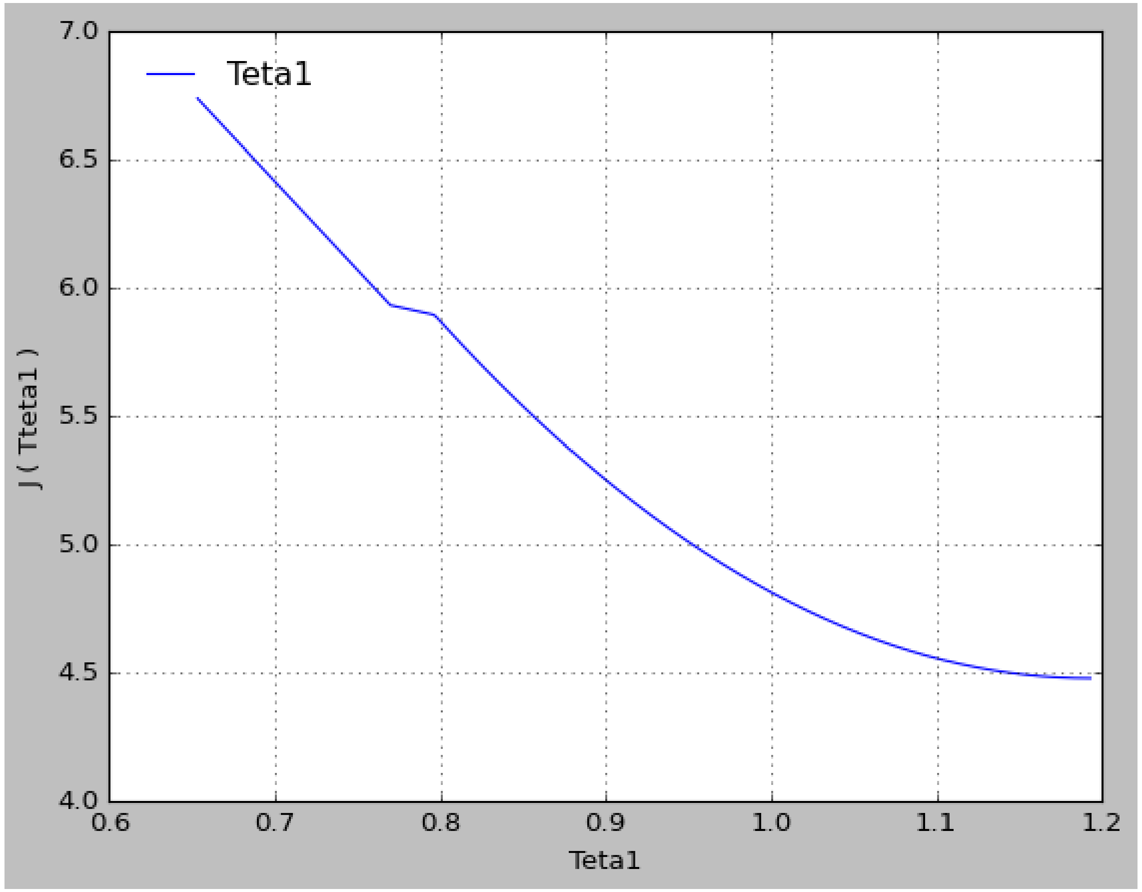


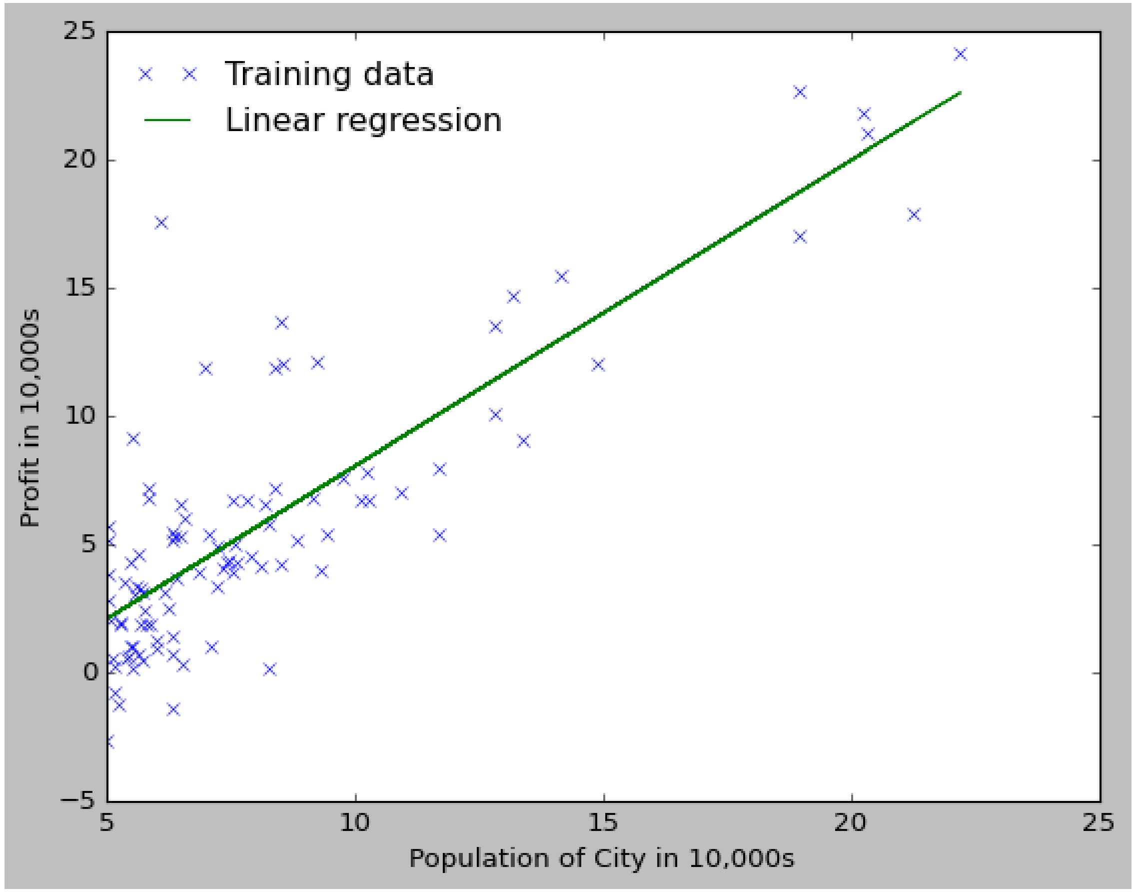
Fonte: https://www.youtube.com/watch?v=YovTqTY-PYY&list=PLLssT5z_DsK-h9vYZkQkYNWcItqhlRJLN&index=9

In [468…

```
#Definir Valores iniciais
# vetor x e vetor y ja foram definidos anteriormente

x = x #vetorx
y = y #vetory
teta0 = 0
teta1 = 0
alfa = 0.01
m = len(x)
h = []
#para impressao
```

```
vetorCusto = []
vetorTeta1 = []

while(1):
    dj_dteta0 = Dj_dteta0(teta0, teta1, x, y, m)
    temp0 = teta0 - alfa * dj_dteta0

    dj_dteta1 = Dj_dteta1(teta0, teta1, x, y, m)
    temp1 = teta1 - alfa * dj_dteta1

    if(temp1 == teta1 ):
        break
    teta0 = temp0
    teta1 = temp1

    vetorTeta1.append(teta1)
    vetorCusto.append(FuncaoCusto(teta0, teta1, x, y, m))#atualiza custo

#imprimir grafico do teta1
plotGraficoFuncaoCusto(vetorTeta1,vetorCusto)

#imprimir grafico da regressão linear
for i in range(m):
    h.append(teta0 + teta1 * x[i])
plotGrafico(x,y,h)
```


Teta1
J (Tteta1)
Teta1
7.0
6.5
6.0
5.5
5.0
4.5
4.0
0.6
0.7
0.8
0.9
1.0
1.1
1.2



Conclusão da parte 01: Regressão linear com uma variável

O gráfico da variação da função de custo, ajuste linear, que é o erro médio quadrático, em relação a Teta 01 evidencia que através do método do gradiente descendente, foi obtido, a um teta0 qualquer, um teta01 que minimizasse a fução custo. Foi definido no código que quando Teta01 parasse de variar, característica de derivada nula, fosse o ponto de parada.Quando se aproxima do mínimo local, o gradiente descendente automaticamente conduzirá asteps menores. Assim, não é necessário diminuir o valor de α ao longo do tempo. Com posse dessas informações no gráfico "population x profit" acima, foi evidenciada reta de regressão linear:

Hipóstese = teta0 + teta1 * x

que melhor se adequa aos dados do treinamento.

2. Regressão linear com múltiplas variáveis

Nesta parte, você implementará a regressão linear com múltiplas variáveis para prever o preço de uma casa. Suponha que você está vendendo sua casa e você queira saber qual seria um bom preço de acordo com o mercado. Dessa forma, uma maneira de fazer isso seria coletar informações sobre as casas que foram vendidas recentemente e fazer um modelo de previsão de preços.

O arquivo "data2.txt" contém um conjunto de treinamento de preços de casas em Portland, Oregon. A primeira coluna representa o tamanho da casa (em metros quadrados), a segunda o número de quartos e terceira o preço da casa.

2.1 *Feature Normalization*

Comece carregando e exibindo alguns valores a partir deste conjunto de dados. Olhando para os valores, note que a dimensão relativa ao tamanho das casas é cerca de 1000 vezes maior que o número de quartos. Quando as features diferem por uma elevada ordem de magnitude, reescalonar a dimensão das features pode fazer a descida do gradiente convergir muito mais rapidamente.

- Subtraia o valor médio de cada feature do conjunto de dados.
- Após subtrair a média, divida os valores das features pelos seus respectivos desvios-padrão

In [469…

```
#Funções
```

```
def DesvioPadrao(mean,numberOfPoints,points):
    serie = 0
    for i in range(numberOfPoints):
        serie = serie + (points[i] - mean)**2
    sd = math.sqrt(serie/numberOfPoints)
    return sd
```

In [470…

```
import pandas as pd
import numpy as np


filename = '/home/aritana/my_jupyter_notebook/linearRegressionOneMultipleVariables/data2.txt'


data = pd.read_csv(filename,sep=',',header=None).values


print(pd.read_csv(filename,sep=',',header=None))
```

```
       0  1       2
0   2104  3  399900
1   1600  3  329900
2   2400  3  369000
3   1416  2  232000
4   3000  4  539900
5   1985  4  299900
6   1534  3  314900
7   1427  3  198999
8   1380  3  212000
9   1494  3  242500
10  1940  4  239999
11  2000  3  347000
12  1890  3  329999
13  4478  5  699900
14  1268  3  259900
15  2300  4  449900
16  1320  2  299900
17  1236  3  199900
18  2609  4  499998
19  3031  4  599000
20  1767  3  252900
21  1888  2  255000
```

```
22  1604  3  242900
23  1962  4  259900
24  3890  3  573900
25  1100  3  249900
26  1458  3  464500
27  2526  3  469000
28  2200  3  475000
29  2637  3  299900
30  1839  2  349900
31  1000  1  169900
32  2040  4  314900
33  3137  3  579900
34  1811  4  285900
35  1437  3  249900
36  1239  3  229900
37  2132  4  345000
38  4215  4  549000
39  2162  4  287000
40  1664  2  368500
41  2238  3  329900
42  2567  4  314000
43  1200  3  299000
44   852  2  179900
45  1852  4  299900
46  1203  3  239500
```

In [471…

```
#Funções
import math

def media(vetor):
    resultado = 0
    for i in range(len(vetor)):
        resultado = resultado + vetor[i]
    return resultado / len(vetor)

def desvioPadrao(mean,numberOfPoints,points):
    serie = 0
    for i in range(numberOfPoints):
        serie = serie + (points[i] - mean)**2
    sd = math.sqrt(serie/numberOfPoints)
    return sd

def subtrairValorMedio(vetor,media):
    resultado = []
    for i in range(len(vetor)):
```

```
        resultado.append(vetor[i]-media)
    return resultado

def dividirPeloDesvioPadrao(vetor, desvioPadrao):
    resultado = []
    for i in range(len(vetor)):
        resultado.append(vetor[i]/desvioPadrao)
    return resultado
```

Feature Normalization

In [472…

```
# Calculating mean and standard deviation
import statistics


#definindo os eixos , teta0 + teta1*x1 + teta2 * x2
x0 = 1
x1 = data[:, 0]#tamanho
x2 = data[:, 1]#quartos
y = data[:, 2]#preço
n = 2 # numero de features [x1,x2]
size = len(x1)

#Media e desvio Padrão


meanX1 = media(x1)
meanX2 = media(x2)

sdX1 = desvioPadrao(meanX1,size,x1)
sdX2 = desvioPadrao(meanX2,size,x2)

x1 = subtrairValorMedio(x1,meanX1)
x2 = subtrairValorMedio(x2,meanX2)

x1 = dividirPeloDesvioPadrao(x1, sdX1)
x2 = dividirPeloDesvioPadrao(x2, sdX2)
#Features foram normalizadas
```

2.2 Gradiente descendente

Nesta parte, você deverá calcular os parâmetros da regressão linear e a função custo para o conjunto de dados, utilizando diferentes taxas de aprendizado para estudar o seu efeito na convergência.

In [473…

```
##Funcoes
import matplotlib.pyplot as plt


def Dj_dteta0(teta0, teta1, teta2, x1, x2, y, m):
    resultado = 0
    for i in range (m):
        resultado = resultado + (teta0 + teta1 * x1[i]+ teta2 * x2[i] - y[i])
    return resultado/m

def Dj_dteta1(teta0, teta1, teta2, x1, x2, y, m):
    resultado = 0
    for i in range (m):
        resultado = resultado + (teta0 + teta1 * x1[i]+ teta2 * x2[i] - y[i])*x1[i]
    return resultado/m

def Dj_dteta2(teta0, teta1, teta2, x1, x2, y, m):
    resultado = 0
    for i in range (m):
        resultado = resultado + (teta0 + teta1 * x1[i]+ teta2 * x2[i] - y[i])*x2[i]
    return resultado/m

def FuncaoCusto(teta0, teta1,teta2, x1,x2, y, m):
    resultado = 0
    custo = []
    for i in range (m):
        custo.append(((teta0 + teta1 * x1[i] + teta2 * x2[i])- y[i])**2)
    for i in range (m):
        resultado = resultado + custo[i]
    return resultado/(2*m)

def plotGrafico(x,y,h):
    plt.xlabel("Population of City in 10,000s")
    plt.ylabel("Profit in 10,000s")
```

```
    plt.plot(x,y,'x',label="Training data")
    plt.plot(x,h,label="Linear regression")
    plt.legend(loc='upper left', frameon=False)
    plt.show()

def plotGraficoFuncaoCusto(iteracoes,custo):
    plt.grid()
    plt.xlabel("No. de iterações")
    plt.ylabel("min J (teta)")
    plt.plot(iteracoes,custo)
    plt.show()
```

Execução

In [474…

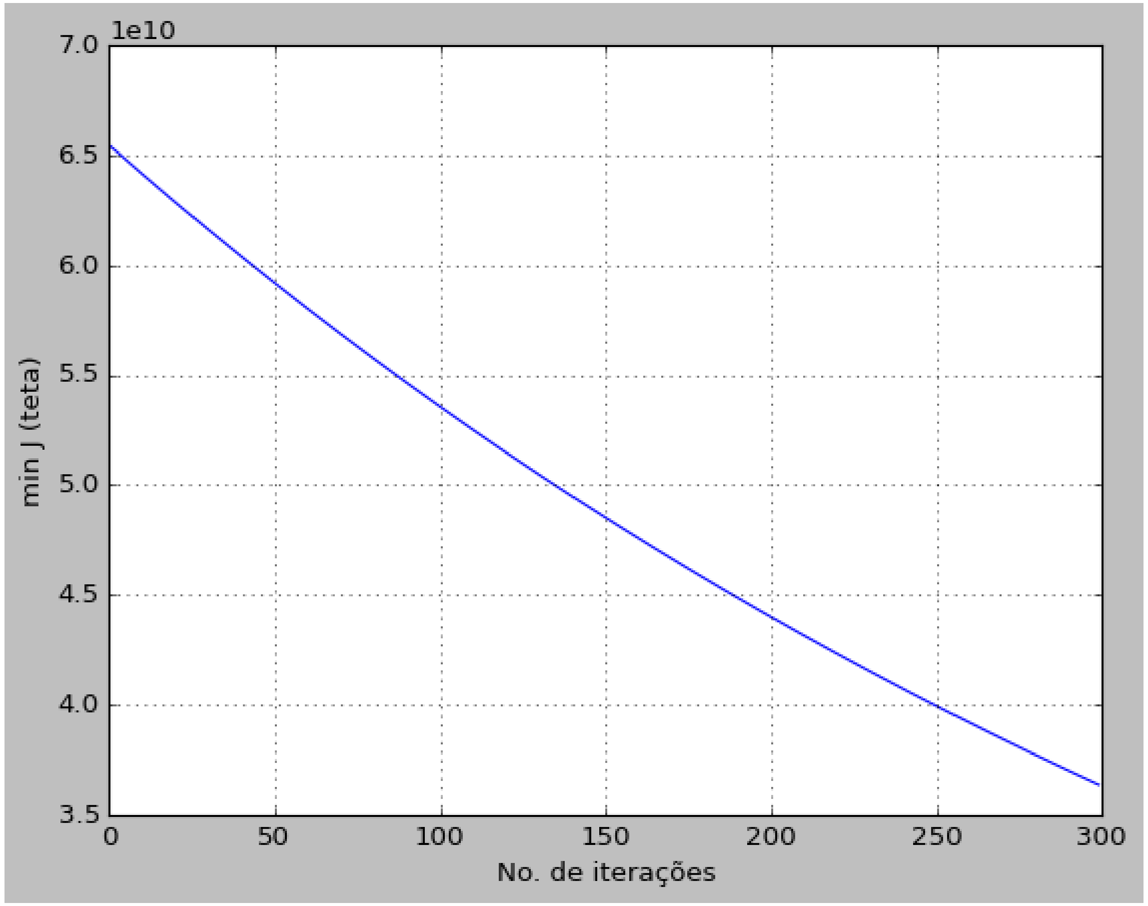
```
#Definir Valores iniciais
# vetor x e vetor y ja foram definidos anteriormente

def executarIteracoes(alfa,NoIteracoes,x1,x2,teta0,teta1,teta2):
    m = len(x1)
    h = []
    #para impressao
    vetorCusto = []
    vetorNumeroIteracoes = []

    #teta0 + teta1*x1 + teta2 * x2
    for i in range(NoIteracoes):
        dj_dteta0 = Dj_dteta0(teta0, teta1, teta2, x1, x2, y, m)
        temp0 = teta0 - alfa * dj_dteta0

        dj_dteta1 = Dj_dteta1(teta0, teta1, teta2, x1, x2, y, m)
        temp1 = teta1 - alfa * dj_dteta1

        dj_dteta2 = Dj_dteta2(teta0, teta1, teta2, x1, x2, y, m)
        temp2 = teta1 - alfa * dj_dteta2

        #atualiza parametros
        teta0 = temp0
        teta1 = temp1
        teta2 = temp2

        #custo e iteracoes para serem plotados
        vetorNumeroIteracoes.append(i)
```

```
        vetorCusto.append(FuncaoCusto(teta0, teta1,teta2, x1,x2, y, m))#atualiza custo

    #Plote a função custo em relação ao número de iterações para ver o seu decaimento;
    plotGraficoFuncaoCusto(vetorNumeroIteracoes,vetorCusto)
    return vetorCusto


def convergente(vetorCusto):
    iteracoes = len(vetorCusto)-1
    ultimaIteracao = vetorCusto[iteracoes]
    penultimaIteracao = vetorCusto[iteracoes -1]

    calculoDIvergencia = penultimaIteracao - ultimaIteracao
    if(calculoDIvergencia <= 0.001):
        print("Convergente")
    else:
        print("Divergente")
```

Verificar diferentes taxas de aprendizado

In [475…

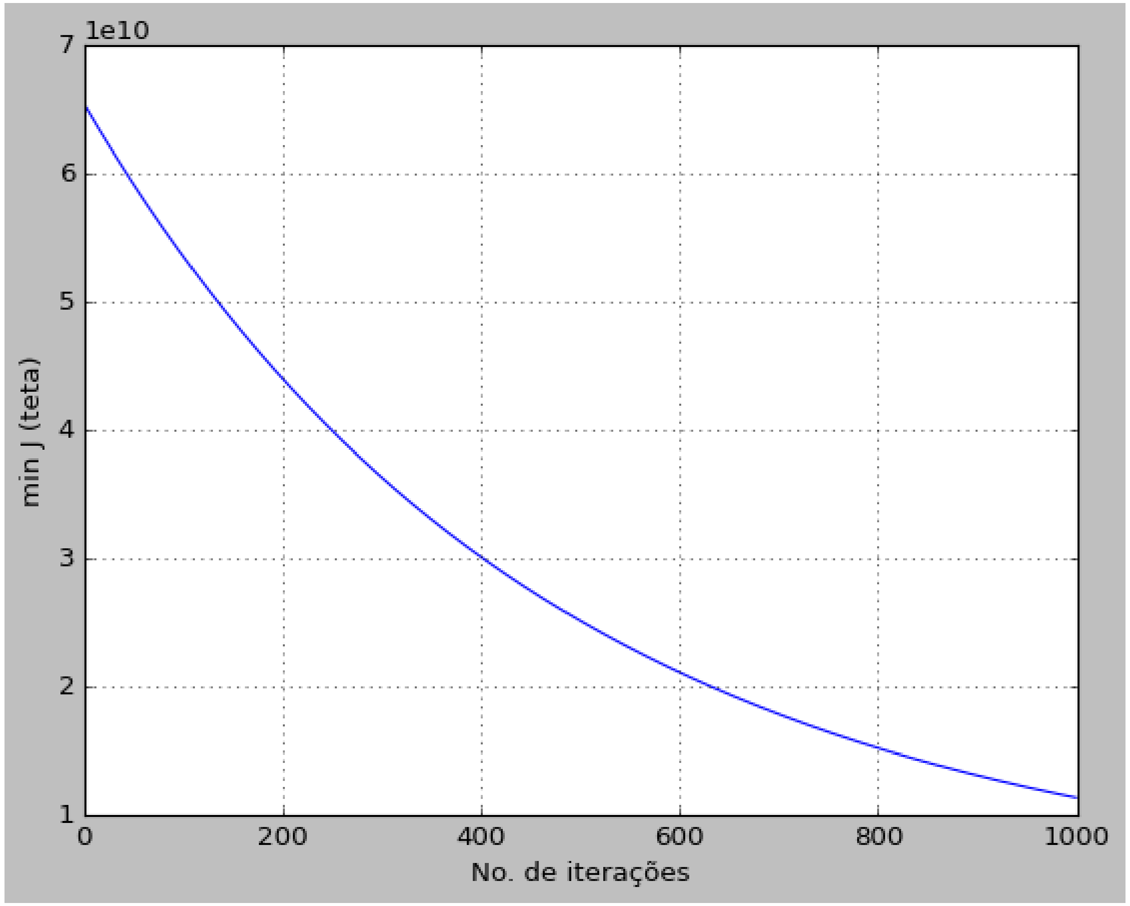
```
x1 = x1 #vetorx
x2 = x2 #vetory
y = y
teta0 = 0
teta1 = 0
teta2 = 0
alfa = 0.001
NoIteracoes = 300

vetorCusto = executarIteracoes(alfa,NoIteracoes,x1,x2,teta0,teta1,teta2)
convergente(vetorCusto)
```




```
Divergente
```

In [476…

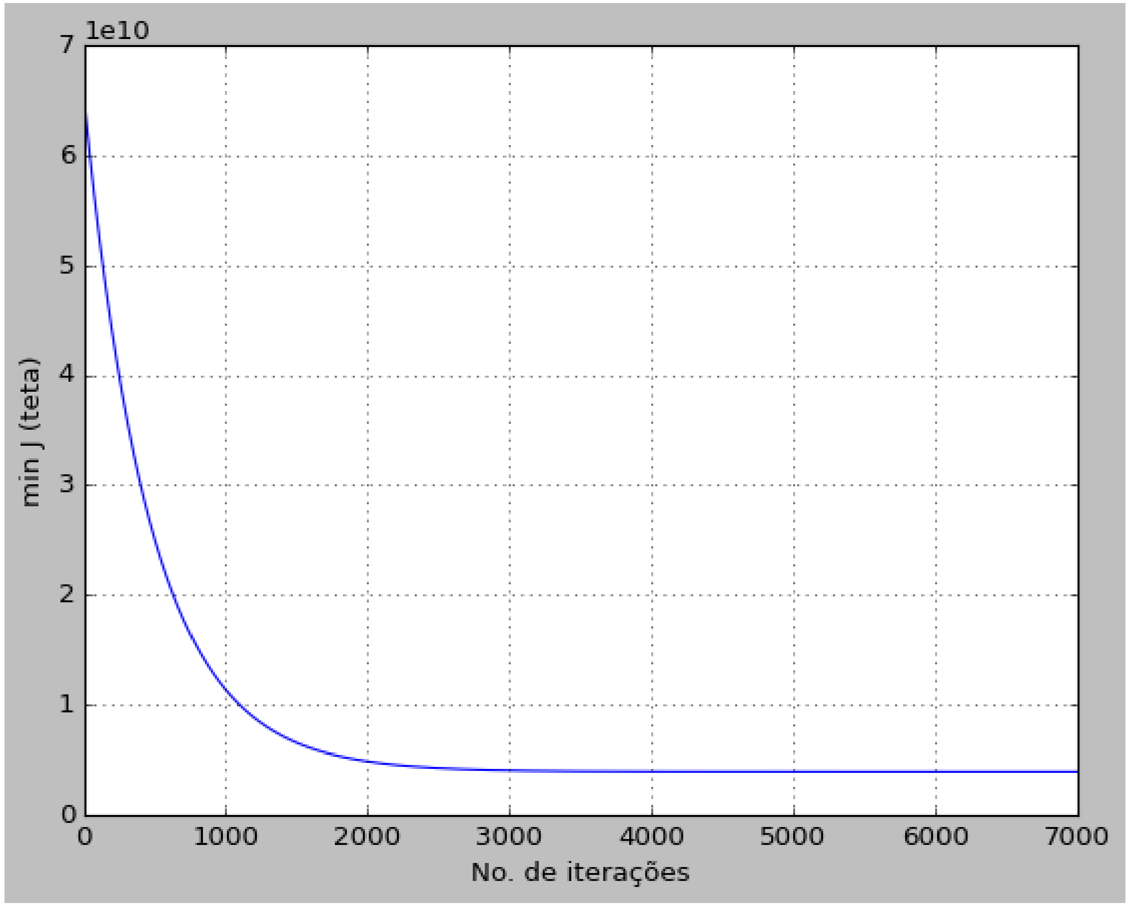
```
alfa = 0.001
NoIteracoes = 1000

vetorCusto = executarIteracoes(alfa,NoIteracoes,x1,x2,teta0,teta1,teta2)
convergente(vetorCusto)
```




```
Divergente
```

In [477…

```
alfa = 0.001
NoIteracoes = 7000

vetorCusto = executarIteracoes(alfa,NoIteracoes,x1,x2,teta0,teta1,teta2)
convergente(vetorCusto)
```

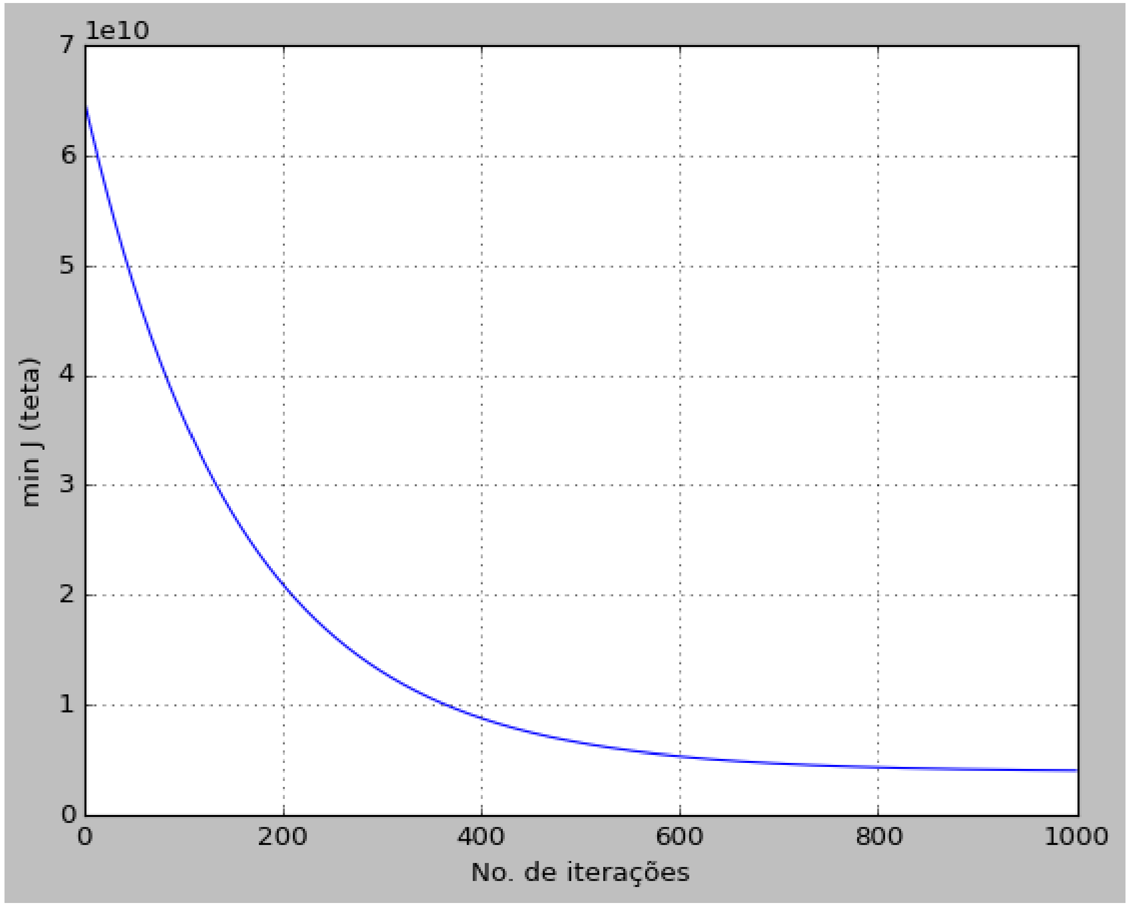



```
Convergente
```

Para valores de alfa = 0.001 a convergencia é muito lenta e exige no mínimo 7000 iteracoes.

In [478…

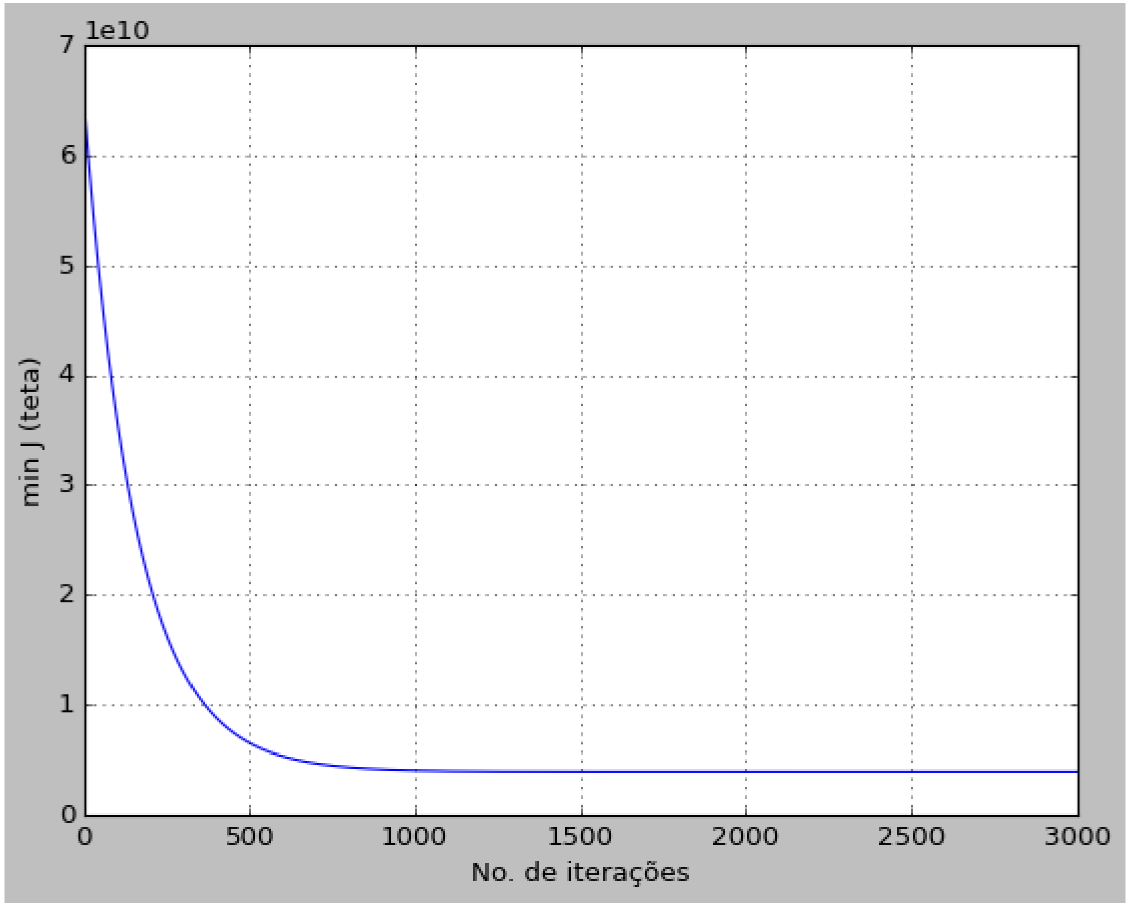
```
alfa = 0.003
NoIteracoes = 1000

vetorCusto = executarIteracoes(alfa,NoIteracoes,x1,x2,teta0,teta1,teta2)
convergente(vetorCusto)
```




```
Divergente
```

In [479…

```
alfa = 0.003
NoIteracoes = 3000

vetorCusto = executarIteracoes(alfa,NoIteracoes,x1,x2,teta0,teta1,teta2)
convergente(vetorCusto)
```

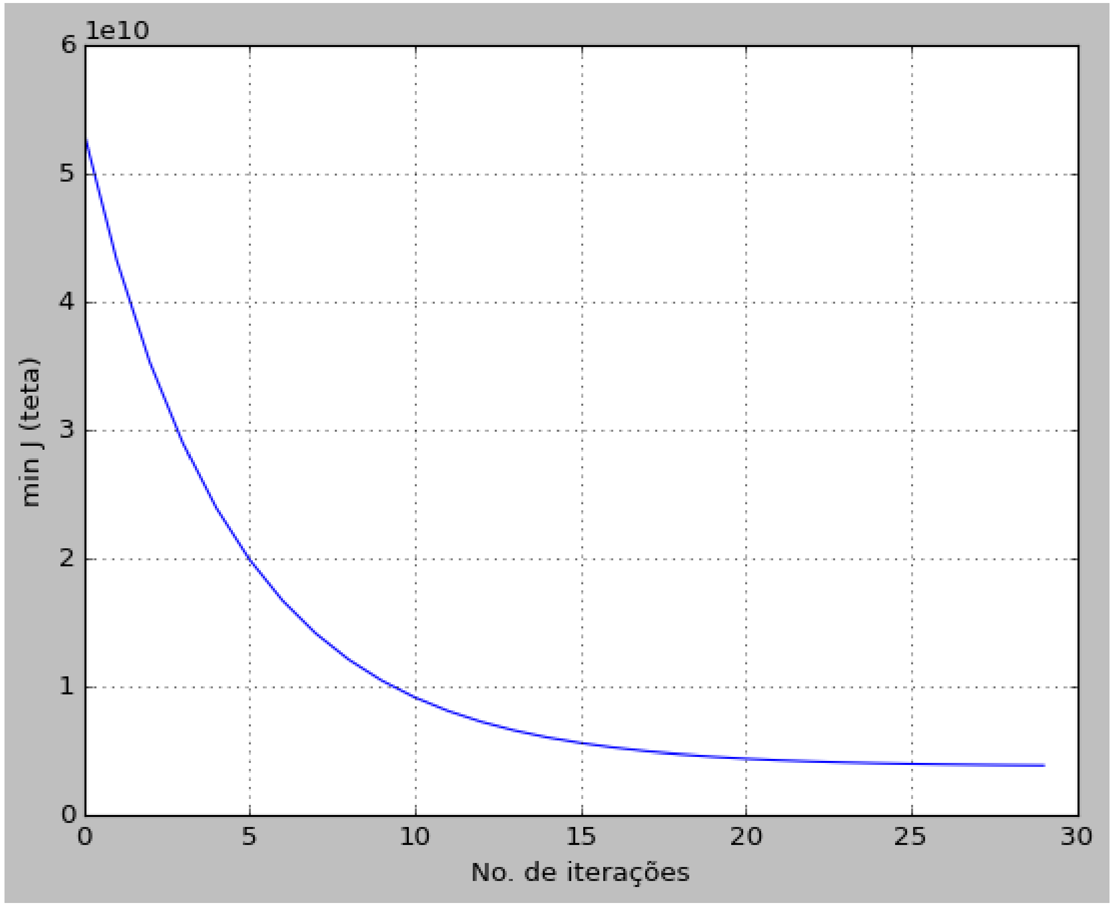



```
Convergente
```

Para valores de alfa = 0.003 a convergencia é mais rápida comparadaa ao alfa=0.001 lenta e exige no mínimo 3000 iteracoes.

In [480…

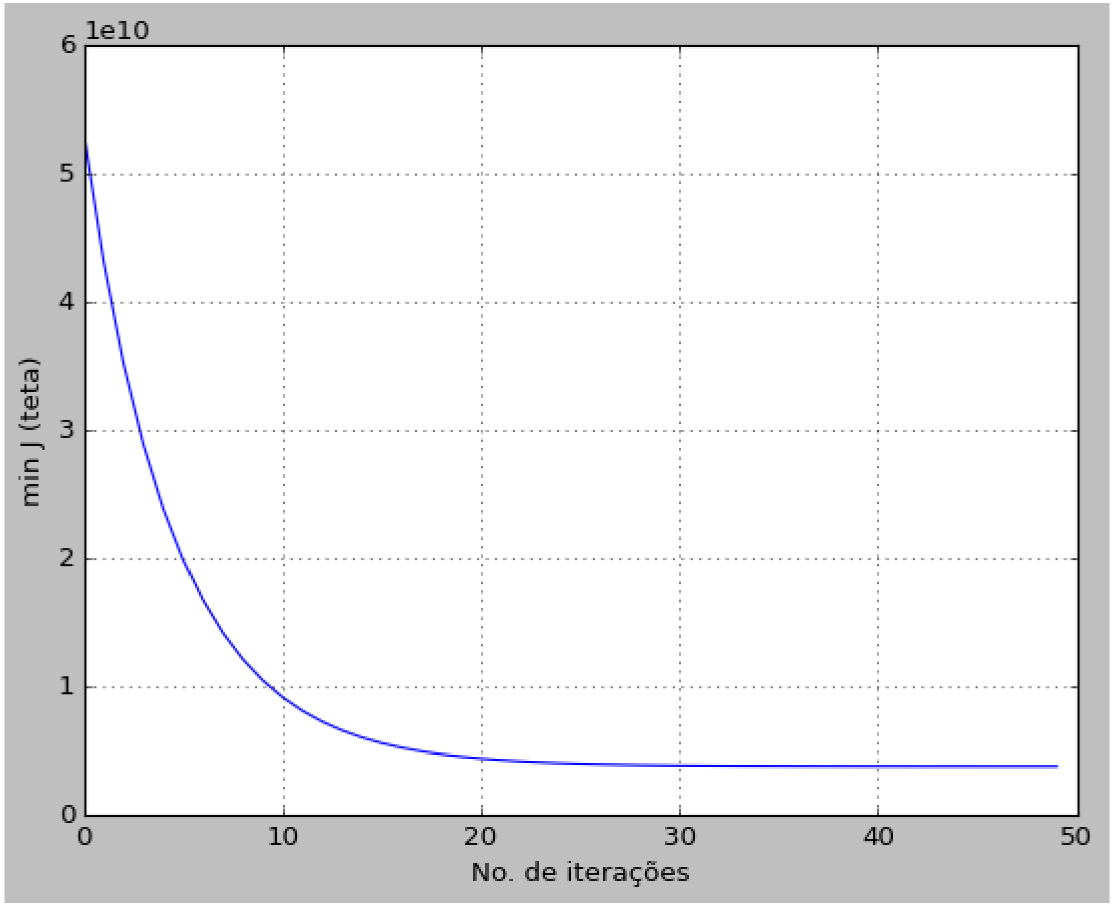
```
alfa = 0.1
NoIteracoes = 30

vetorCusto = executarIteracoes(alfa,NoIteracoes,x1,x2,teta0,teta1,teta2)
convergente(vetorCusto)
```




```
Divergente
```

In [481…

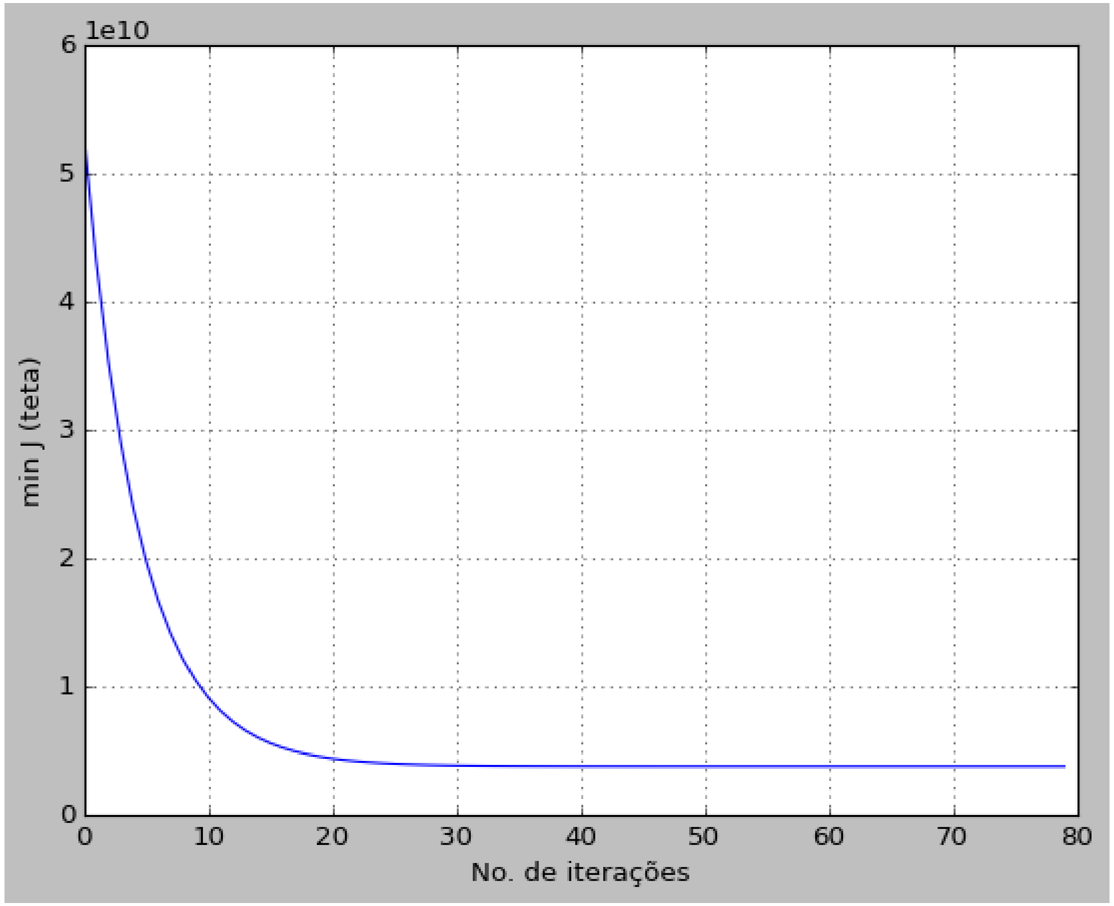
```
alfa = 0.1
NoIteracoes = 50

vetorCusto = executarIteracoes(alfa,NoIteracoes,x1,x2,teta0,teta1,teta2)
convergente(vetorCusto)
```




```
Divergente
```

In [482…

```
alfa = 0.1
NoIteracoes = 80

vetorCusto = executarIteracoes(alfa,NoIteracoes,x1,x2,teta0,teta1,teta2)
convergente(vetorCusto)
```

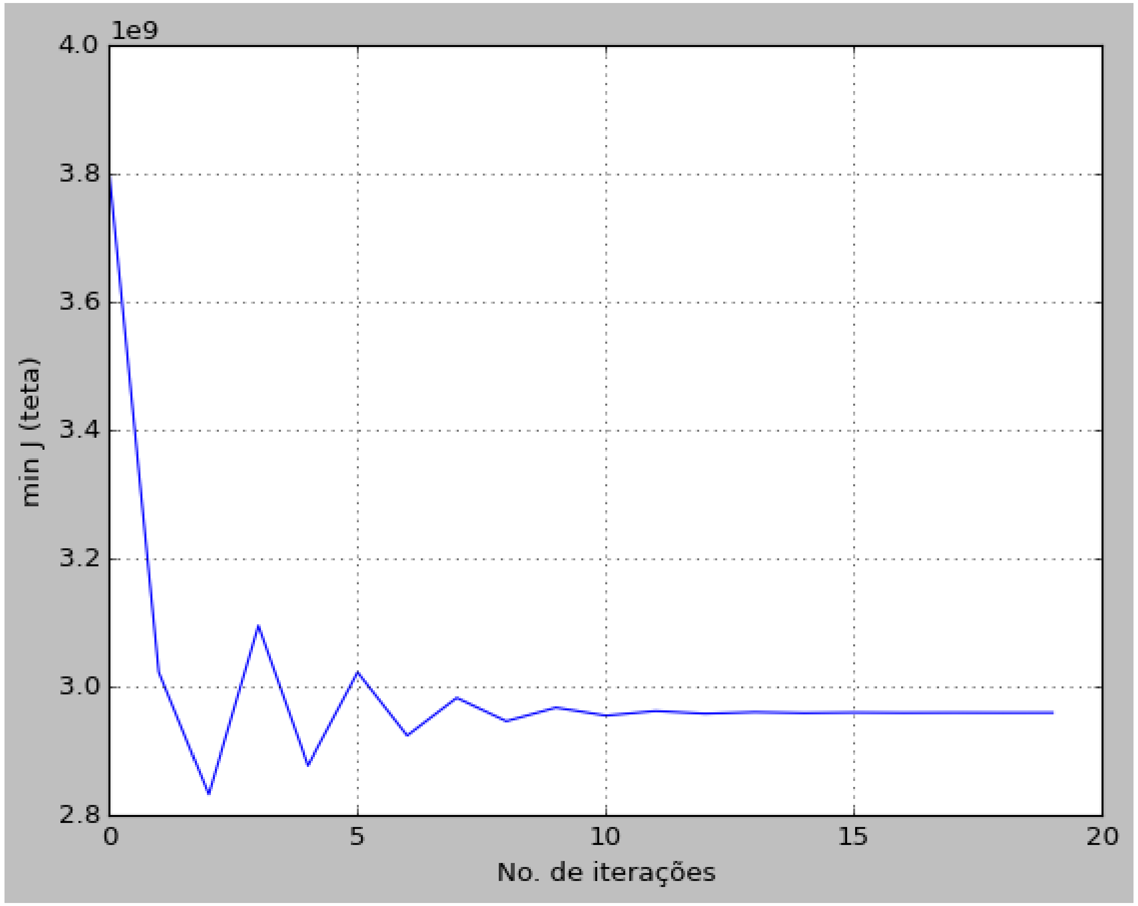



```
Convergente
```

Para valores de alfa = 0.1 a convergencia é mais rápida em relação aos resultados anteriores e exige no mínimo 80 iteracoes.

In [483…

```
alfa = 1
NoIteracoes = 20

vetorCusto = executarIteracoes(alfa,NoIteracoes,x1,x2,teta0,teta1,teta2)
convergente(vetorCusto)
```

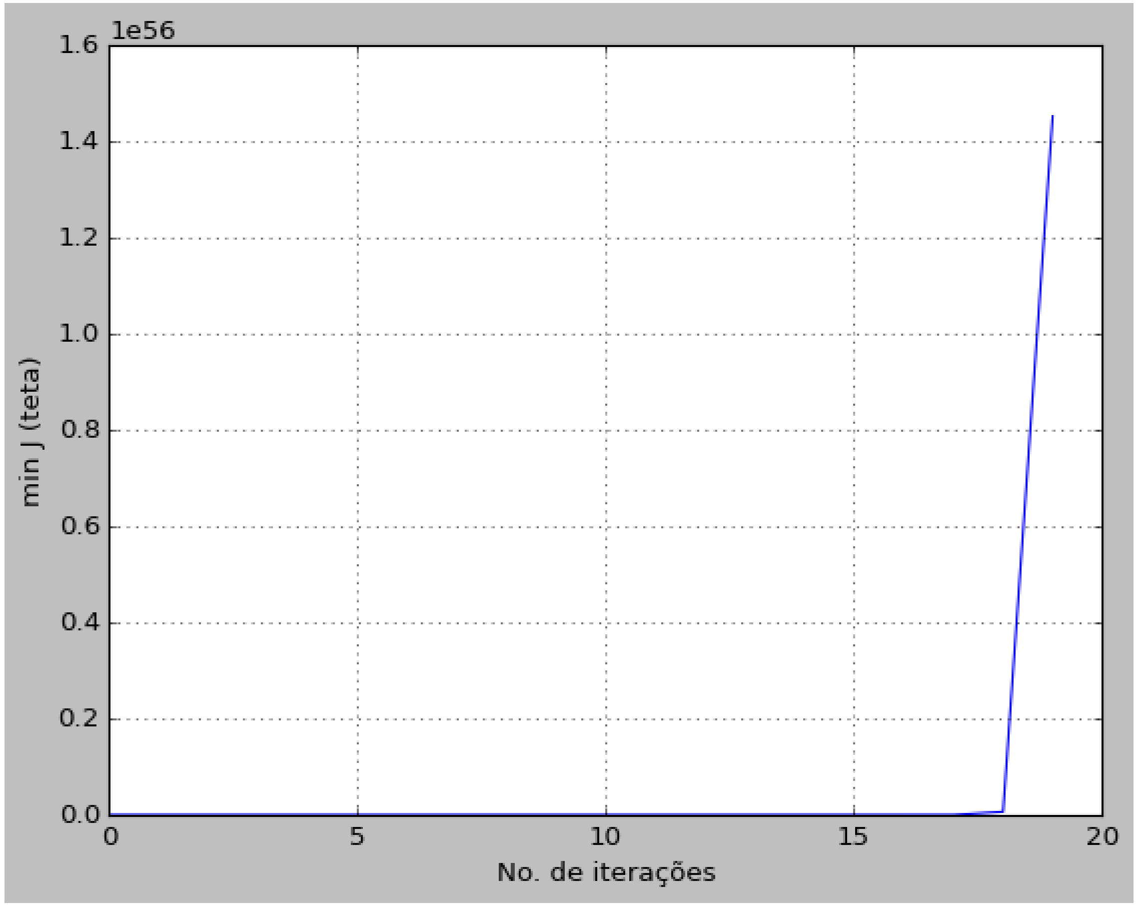



```
Convergente
```

Para valores de alfa = 1 a convergencia é mais rápida em relação aos resultados anteriores e exige no mínimo 20 iteracoes.

In [484…

```
alfa = 10
NoIteracoes = 20

vetorCusto = executarIteracoes(alfa,NoIteracoes,x1,x2,teta0,teta1,teta2)
convergente(vetorCusto)
```




```
Convergente
```

Com alfas maiores, como 10, não é possível executar a operação para minizar a função custo. Os valores não conseguem alcançar os mínimos locais. Ocorre o que na Lecture 4.4 é chamada de overshooting.

Fonte: https://www.youtube.com/watch?v=CYlR9oYhYuY&list=PLLssT5z_DsK-h9vYZkQkYNWcItqhlRJLN&index=21

2.3 Veja que agora não é possível traçar o ajuste linear como no exercício anterior. Por quê?

Conclusão da parte 02: 2.1 Feature Normalization

A normalização dos dados auxilia o método gradiente a convergir com maior rapidez. Para testar de o gradiente descendente fucionou corretamente, utilizamos o gráfico de número de iterações por minimização da função custo e testamos se decresce pelo menos 10^-3 em cada iteração, como pode ser observado nos gráficos e comentários anteriores a esta seção.

2.3: O exercício trabalha com três dimensões e seria apenas traçar as curvas de níveis da superfície.

3. Equação Normal

Você aprendeu nos vídeos que a solução matemática para a regressão linear pode ser dada pela equação normal:

$$\theta = \left(X^T X\right)^{-1} X^T \vec{y}.$$

É importante disser que, o uso desta fórmula, não requer nenhum redimensionamento das *features*. Além disso, você obterá uma solução exata (não há *loops* até a convergência, como no gradiente descendente).

Calcule os parâmetros da regressão linear utilizando a equação normal e compare o resultado com os parâmetros obtidos por meio do método do gradiente descendente.

Sugestão: você poderia utilizar a base de dados para prever o valor das casas utilizando os parâmetros da regressão encontrados com os dois métodos (gradiente e equação normal). Após esta etapa, basta calcular o erro quadrático médio das previsões em relação aos valores reais para cada um dos métodos e fazer a comparação.

In [485…

```
import pandas as pd
import numpy as np

filename = '/home/aritana/my_jupyter_notebook/linearRegressionOneMultipleVariables/data3.txt'
```

```
data = pd.read_csv(filename,sep=',',header=None).values


#definindo os eixos

#print(pd.read_csv(filename,sep=',',header=None))

X = data
print(X)
```

```
[[   1 2104    3]
 [   1 1600    3]
 [   1 2400    3]
 [   1 1416    2]
 [   1 3000    4]
 [   1 1985    4]
 [   1 1534    3]
 [   1 1427    3]
 [   1 1380    3]
 [   1 1494    3]
 [   1 1940    4]
 [   1 2000    3]
 [   1 1890    3]
 [   1 4478    5]
 [   1 1268    3]
 [   1 2300    4]
 [   1 1320    2]
 [   1 1236    3]
 [   1 2609    4]
 [   1 3031    4]
 [   1 1767    3]
 [   1 1888    2]
 [   1 1604    3]
 [   1 1962    4]
 [   1 3890    3]
 [   1 1100    3]
 [   1 1458    3]
 [   1 2526    3]
 [   1 2200    3]
 [   1 2637    3]
 [   1 1839    2]
 [   1 1000    1]
 [   1 2040    4]
 [   1 3137    3]
```

```
[   1 1811    4]
[   1 1437    3]
[   1 1239    3]
[   1 2132    4]
[   1 4215    4]
[   1 2162    4]
[   1 1664    2]
[   1 2238    3]
[   1 2567    4]
[   1 1200    3]
[   1  852    2]
[   1 1852    4]
[   1 1203    3]]
```

In [486…

```
# importamos a bibliteca NumPy
import numpy as np

def matrizTransposta(X):
  # vamos declarar e construir uma matrix
  # 2x3 (duas linhas e três colunas
  matriz = np.array(X)

  # como temos uma matriz 2x3, a transposta deverá ser
  # 3x2, ou seja, três linhas e duas colunas
  linhas = np.shape(matriz)[0] # linhas da matriz original
  colunas = np.shape(matriz)[1] # colunas da matriz original
  transposta = np.empty((colunas, linhas))

  # e agora vamos preencher a matriz transposta
  for i in range(np.shape(matriz)[0]):
    for j in range(np.shape(matriz)[1]):
      transposta[j][i] = matriz[i][j]

    return transposta

#if __name__== "__main__":
#  main()
```

In [487…

```
# X = X0 X1 X2


size = len(X)
y = y
```

```
xTransposta = matrizTransposta(X)
```

Observação: A última parte não foi possível desenvolver mais.