**Rapport Descente de gradient**

[**Variables**](#_cr3d5chba6gk) **1**

[**Moindres Carrés**](#_a4pvv7mtv8te) **2**

[**Descente de Gradient**](#_q7z9m2g5mbww) **3**

[**Brute Force**](#_rb6byrt113st) **6**

[**Autres**](#_yqyyi5pngbs2) **8**

# **Variables**

Pour s’y retrouver et tester nos codes facilement avec plusieurs méthodes différentes et des variables que l’on aimerait changer manuellement nous avons créé une région avec des variables et les variables que l’on peut changer sont pas convention en uppercase.

dataset = **None**

*# choosing the linear regression method*

*# 0 : least squares*

*# 1 : gradient descent*

*# 2 : brute force*

METHOD = 1

*# regression line coefficients*

a, b = 0, 0 *# coefficients*

STARTING\_A = 0 *# starting value for a in gradient descent method*

STARTING\_B = 0 *# starting value for b in gradient descent method*

*# linear regression with gradient descent*

EPOCHS = 100 *# number of epoch before stop running*

LEARNING\_RATE = 0.005 *# learning rate*

DISPLAY\_EPOCH = 1 *# display plot each DISPLAY\_EPOCH*

epoch = 0 *# to count epoch during the run*

*# linear regression with brute force*

A\_MIN = -50 *# a min to test*

A\_MAX = 50 *# a max to test*

B\_MIN = -10 *# b min to test*

B\_MAX = 10 *# b max to test*

ACCURACY = 0.01 *# step of accuracy to test a and b*

*# plot*

fig = plt.figure(1) *# to display the linear regression*

ax = fig.add\_subplot(1, 1, 1) *# to plot the linear regression*

ani = **None** *# to animate the plot*

started = **False** *# turn to True when Enter key is pressed*

# Moindres Carrés

Nous avons créé tout d’abord une fonction de moindre carrés pour tester nos résultats

**def** least\_squares():

*"""*

*Return the coefficient of the linear regression using the least squares method*

**:return***: a, b (coefficients as f(x) = a\*x + b)*

*"""*

length = len(dataset[0])

mean\_x = 0

mean\_y = 0

*# calculate the means of the features*

**for** i **in** range(length):

mean\_x += dataset[0][i]

mean\_y += dataset[1][i]

mean\_y /= length

mean\_x /= length

*# as a reminder :*

*# f(x) = COV(X,Y)/VAR(X)\*x + b*

*# b = mean(Y) - a \* mean(X)*

cov\_xy = 0

var\_x = 0

*# Calculation of COV(X, Y) and of VAR(X)*

**for** i **in** range(length):

cov\_xy += (dataset[0][i] - mean\_x) \* (dataset[1][i] - mean\_y)

var\_x += (dataset[0][i] - mean\_x) \*\* 2

*# Calculation of the linear regression coefficients*

a = cov\_xy / var\_x

b = mean\_y - a \* mean\_x

**return** a, b

# 

# Descente de Gradient

La méthode la plus intéressante pour la descente de gradient et la méthode pour effectuer une étape de cet algorithme. En effet la méthode de la descente de gradient répète une et une seule même étape comme ci-dessous jusqu’à convergence et/ou un certain nombre d’étapes effectuée défini au préalable. Nous avons décidé d'arrêter la descente de gradient seulement après un certain nombre défini d’étape et de ne pas prendre en considération sa convergence.

*# initialize a and b coefficient*

a = STARTING\_A

b = STARTING\_B

*# loop over the number of epochs*

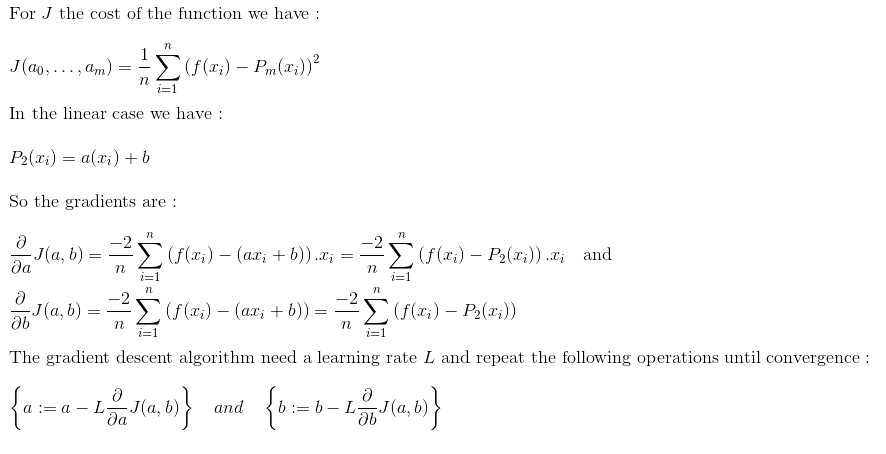
**while** epoch < EPOCHS:

*# step by step gradient descent*

a, b, squared\_error = step\_gradient\_descent(dataset[0], dataset[1], a, b)

epoch += 1

Pour chaque étape, on calcule le gradient du coût des paramètres actuels par rapport à chacun de ses paramètres pour en estimer vers “quelle direction” il faut changer les paramètres actuels pour faire baisser cette fonction de coût. Nous avons décidé d’utiliser comme fonction de coût la moyenne des distances au carré entre les prédictions et les valeurs réelles. Ci-dessous nous montrons comment calculer les gradients. Il est à noter que nous faisons les modifications de valeurs en fonction d’un paramètre qui est le taux d’apprentissage. Ce taux d’apprentissage permet d’éviter que la modification soit trop brusque et donc que l’on ne puisse pas atteindre un minimum.



Ainsi nous avons créé la méthode effectuer une étape de la descente de gradient.

***def*** *step\_gradient\_descent(X, Y, a, b):*

*"""*

*Return the coefficient of the linear regression using the gradient descent method*

***:return****: a, b (coefficients as f(x) = a\*x + b)*

***:param*** *X: X array*

***:param*** *Y: Y array*

***:param*** *a: coefficient a*

***:param*** *b: coefficient b*

***:return****: a, b, squared\_error (coefficients as f(x) = a\*x + b)*

*"""*

*N = len(X)*

*# calculate our current predictions*

*predictions = [(a \* X[i]) + b* ***for*** *i* ***in*** *range(N)]*

*# calculate the errors*

*error = [(Y[i] - predictions[i])* ***for*** *i* ***in*** *range(N)]*

*# calculate the gradients*

*a\_gradient = -(2 / N) \* sum([X[i] \* error[i]* ***for*** *i* ***in*** *range(N)])*

*b\_gradient = -(2 / N) \* sum([error[i]* ***for*** *i* ***in*** *range(N)])*

*# update the coefficients*

*a -= LEARNING\_RATE \* a\_gradient*

*b -= LEARNING\_RATE \* b\_gradient*

*cost = sum([e\*\*2* ***for*** *e* ***in*** *error]) # if needed*

***return*** *a, b, cost*

Nous avons donc testé cette méthode avec les paramètres :

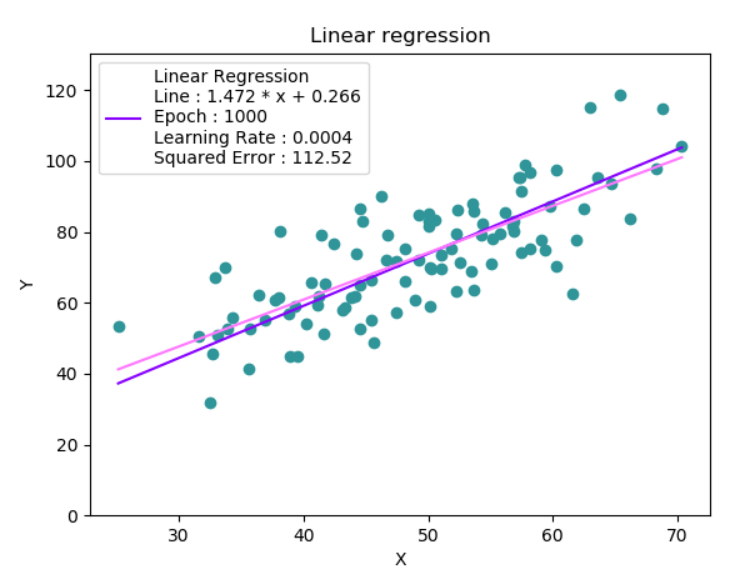
STARTING\_A = 0

STARTING\_B = 0

EPOCHS = 1000

LEARNING\_RATE = 4\*10\*\*-4

Nous obtenons à la fin des 1000 itérations le graphique ci-dessous. En rose nous avons la droite de régression linéaire calculée par la méthode des moindres carrés et en violet nous avons la droite de régression calculée par la méthode de la descente de gradient. Nous pouvons donc remarquer qu’au bout des 1000 itérations nous avons nos deux droites qui se distinguent encore bien. Nous avons donc deux possibilités qui peuvent impliquer cet écart. Premièrement, le nombre d’itération n’est pas assez élevé par rapport à la valeur de LEARNING\_RATE. Deuxièmement il est possible que la méthode de la descente de gradient soit tombée dans un extremum local.



Ainsi nous avons décidé de changer le paramètre EPOCHS tel que :

EPOCHS = 100000

Nous pouvons constater que nos deux droites de régression sont maintenant quasi confondues.

# 

# Brute Force

Avant de créer la méthode pour définir les meilleurs valeurs de a et de b. Nous avons besoin de créer la fonction ci-dessous qui est la fonction de coût vue pendant la méthode de la descente de gradient.

**def** MSE(X, Y, a, b):

*"""*

*Return the cost of the linear regression*

**:return***: a, b (coefficients as f(x) = a\*x + b)*

**:param** *X: X array*

**:param** *Y: Y array*

**:param** *a: coefficient a*

**:param** *b: coefficient b*

**:return***: squared\_error (coefficients as f(x) = a\*x + b)*

*"""*

N = len(X)

**return** sum([(Y[i] - (a \* x + b))\*\*2 / N **for** i, x **in** enumerate(X)])

Enfin il ne nous reste plus qu’à créer la fonction qui va retourner couple de paramètres a et b parmis les couples à tester tel qu’ils aient le coût le plus faible.

**def** brute\_force(X, Y, a\_min, a\_max, b\_min, b\_max, accuracy):

*"""*

*Return the coefficient of the linear regression using the brute force method*

**:param** *X: X array*

**:param** *Y: Y array*

**:param** *a\_min: coefficient a minimum to test*

**:param** *a\_max: coefficient a maximum to test*

**:param** *a\_min: coefficient a minimum to test*

**:param** *a\_max: coefficient a maximum to test*

**:param** *accuracy: step of accuracy to test a and b*

**:return***: a, b (coefficients as f(x) = a\*x + b)*

*"""*

best\_a , best\_b, best\_cost = a\_min, b\_min, MSE(X, Y, a\_min, a\_max)

current\_a = a\_min

print(**"There are {0} possibilities"**

.format(int((a\_max - a\_min) \* (b\_max - b\_min) / accuracy\*\*2)))

**while** current\_a <= a\_max:

current\_b = b\_min

**while** current\_b <= b\_max:

current\_cost = MSE(X, Y, current\_a, current\_b)

**if** current\_cost < best\_cost:

best\_a, best\_b, best\_cost = current\_a, current\_b, current\_cost

current\_b += accuracy

current\_a += accuracy

**return** best\_a, best\_b

En lançant la méthode on note que pour les paramètres :

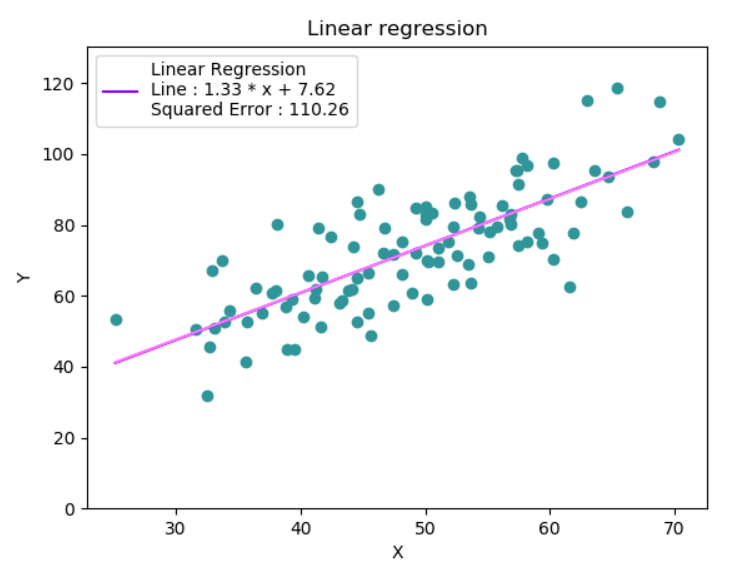
A\_MIN = -50 *# a min to test*

A\_MAX = 50 *# a max to test*

B\_MIN = -10 *# b min to test*

B\_MAX = 10 *# b max to test*

ACCURACY = 0.01 *# step of accuracy to test a and b*



On a 20 000 000 couples de valeurs à tester. C’est donc un algorithme qui est très très long à faire fonction bien qu’au final la droite de régression trouvée est quasi confondues avec celle calculée par la méthode des moindres carrés. D’autre part on peut noter que la précision n’est pas très élevée et plus on l'augmente cette précision et plus l’algorithme sera long. Aussi, rien n’indique que la meilleure valeur pour a appartient à et que la meilleure valeur pour b appartient à .

# Autres

Nous vous invitons à aller tester directement les codes. Il suffit de changer une seule valeur pour passer d’une méthode à l’autre tel que :

*# choosing the linear regression method*

*# 0 : least squares*

*# 1 : gradient descent*

*# 2 : brute force*

METHOD = 1

D’autre part il est intéressant de lancer la méthode de la descente de gradient car l’évolution de l’algorithme est animé grâce aux imports :

**import** matplotlib.pyplot **as** plt

**import** matplotlib.animation **as** animation

On peut donc constater visuellement comment l’algorithme tend vers une solution par la méthode de la descente de gradient. Il suffit d’appuyer sur la touche **ENTER** pour lancer l’algorithme une fois le programme lancé.