**Implement Gradient Descent in Python**

**What is gradient descent ?**

It is an optimization algorithm to find the minimum of a function. We start with a random point on the function and move in the **negative direction** of the **gradient of the function**to reach the **local/global minima**.

# ****Example by hand :****

**Question** : Find the local minima of the function y=(x+3)² starting from the point x=2

**Solution :**We know the answer just by looking at the graph. y = (x+3)² reaches it’s minimum value when x = -3 (i.e when x=-3, y=0). Hence x=-3 is the local and global minima of the function.

Now, let’s see how to obtain the same numerically using gradient descent.

**Step 1** : Initialize x =2. Then, find the gradient of the function, dy/dx = 2\*(x+3).

**Step 2** : Move in the direction of the negative of the gradient. Stopping condition: how much to move? For that, we require a learning rate. Let us assume the **learning rate → 0.01**

**Step 3**: Let’s perform 2 iterations of gradient descent

Initialize Parameters:

Learning rate = 0.01

Iteration 1:

Iteration 2:

**Step 4** : We can observe that the X value is slowly decreasing and should converge to -3 (the local minima). However, how many iterations should we perform?

Let us set a precision variable in our algorithm which calculates the difference between two consecutive “x” values . If the difference between x values from 2 consecutive iterations is lesser than the precision we set, stop the algorithm !

# ****Gradient descent in Python :****

**Step 1** : Initialize parameters

cur\_x = 2 # The algorithm starts at x=2  
rate = 0.01 # Learning rate  
precision = 0.000001 #This tells us when to stop the algorithm  
previous\_step\_size = 1 #  
max\_iters = 10000 # maximum number of iterations  
iters = 0 #iteration counter  
df = lambda x: 2\*(x+3) #Gradient of our function

**Step 2**: Run a loop to perform gradient descent :

i. Stop loop when difference between x values from 2 consecutive iterations is less than 0.000001 or when number of iterations exceeds 10,000

.

while previous\_step\_size > precision and iters < max\_iters:

prev\_x = cur\_x #Store current x value in prev\_x

cur\_x = cur\_x - rate \* df(prev\_x) #Grad descent

previous\_step\_size = abs(cur\_x - prev\_x) #Change in x

iters = iters+1 #iteration count

print("Iteration",iters,"\nX value is",cur\_x) #Print iterations  
  
print("The local minimum occurs at", cur\_x)

**Output**: From the output below, we can observe the x values for the first 10 iterations- which can be cross checked with our calculation above. The algorithm runs for 571 iterations before it terminates. The code and solution is embedded below for reference.

X value is -2.9999364493143186

Iteration 559

X value is -2.9999377203280324

Iteration 560

X value is -2.999938965921472

Iteration 561

X value is -2.9999401866030424

Iteration 562

X value is -2.9999413828709818

Iteration 563

X value is -2.999942555213562

Iteration 564

X value is -2.999943704109291

Iteration 565

X value is -2.999944830027105

Iteration 566

X value is -2.999945933426563

Iteration 567

X value is -2.999947014758032

Iteration 568

X value is -2.9999480744628713

Iteration 569

X value is -2.999949112973614

Iteration 570

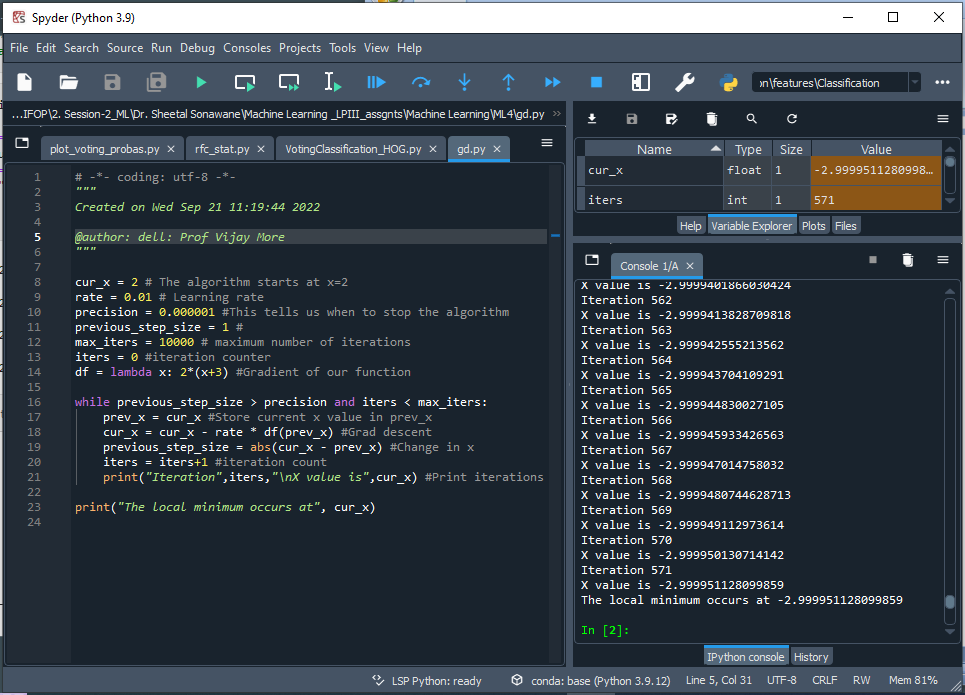
X value is -2.999950130714142

Iteration 571

X value is -2.999951128099859

The local minimum occurs at -2.999951128099859

Screenshot of program and output



Graph of output values

Source: https://towardsdatascience.com/implement-gradient-descent-in-python-9b93ed7108d1