# Assignment 2: Gradient Descent Algorithms

521H0489 – Hồ Hữu An

## I. The Gradient Descent algorithm for the multivariate Linear Regression problem (for multi-attribute objects).

Collecting data will always have multiple features, not just one. Then, hθ(x) will take the form:

$$h\theta(x_1,x_2,\cdots,x_n)=\theta_0 + \theta_1 x_1 + \theta_2 x_2 +\cdots+ \theta_n x_n$$

With $\theta_j$ , $x_j$ are the weights and data of the jth feature (j runs from 0 to n). We consider $x_0 = 1$ by default because $\theta_0$ is a weight that does not depend on x, called an interceptor. Cost Function remains unchanged:

$$J(\theta_0, \theta_1, \cdots, \theta_n) = \frac{1}{2m}\sum_{i=0}^{m}(h\theta\,(x_0^{(i)},\ldots,x_n^{(i)}) - y^{(i)})^2$$

However, when upgrading the weights, we need to take the partial derivative of each θ and upgrade the weights simultaneously. That means we cannot upgrade $\theta_1$ then use the value $\theta_1$ then to calculate the partial derivative of $\theta_2$. Mathematically speaking, at each upgrade step, we need to get the gradient value at position ($\theta_0$ ,$\theta_1$ , $\theta_2$, …, $\theta_n$ ) then subtract the corresponding weights,

$$\theta_j = \theta_j - \alpha\frac{\partial}{\partial\theta_j} J(\theta_0, \cdots, \theta_n)$$

However, we can calculate the formula of $\frac{\partial}{\partial\theta_j} J(\theta_0, \cdots, \theta_n)$ in more detail. by taking partial derivatives as well. You can try taking the partial derivative yourself and compare it with the results below. We have the general formula:

$$\frac{\partial}{\partial\theta_j} = \frac{1}{m} \sum_{i=1}^{m}(h\,(x_0^{(i)},\ldots,x_n^{(i)}) - y^{(i)})\, x_j^{(i)} \text{ , } 0 \le j \le n$$

However, writing as below will help us visualize more clearly:

$$\frac{\partial}{\partial\theta_j} = \frac{1}{m} \sum_{i=1}^{m}(h\,(x_0^{(i)},\ldots,x_n^{(i)}) - y^{(i)}) \text{ , with } j = 0$$

$$\frac{\partial}{\partial\theta_j} = \frac{1}{m} \sum_{i=1}^{m}(h\,(x_0^{(i)},\ldots,x_n^{(i)}) - y^{(i)})\, x_j^{(i)} \text{ , with } 1 \le j \le n$$

Finally, getting a general Gradient Descent algorithm:

1. Start by assigning $\theta_0$ ,$\theta_1$ , $\theta_2$, …, $\theta_n$ random values.

2. **Simultaneous** updates of θ new by subtracting the partial derivative of each weight at its current position.

$$\theta_j = \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h\left(x_0^{(i)},\ldots,x_n^{(i)}\right) - y^{(i)}\right) \text{ , with } j = 0$$

$$\theta_j = \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h\left(x_0^{(i)},\ldots,x_n^{(i)}\right) - y^{(i)}\right) x_j^{(i)} \text{ , with } 1 \le j \le n$$

3. Repeat step 2 until all calculated partial derivatives are 0 or an extremely small value.

**Overview:**

- Updating the weights according to the partial derivative magnitude can cause overshooting. The best way is to control the amount of updates using the α parameter called learning rate.
- In case of using Gradient Descent with multiple weights, we update the weights simultaneously using the formula

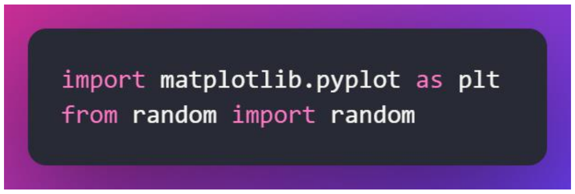
-

$$\theta_j = \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h\left(x_0^{(i)},\ldots,x_n^{(i)}\right) - y^{(i)}\right) \text{ , with } j = 0$$

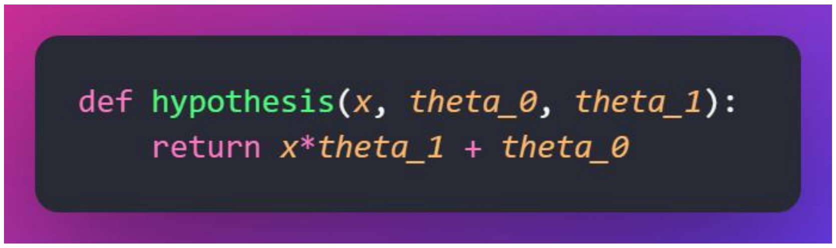
-

$$\theta_j = \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h\left(x_0^{(i)},\ldots,x_n^{(i)}\right) - y^{(i)}\right) x_j^{(i)} \text{ , with } 1 \le j \le n$$

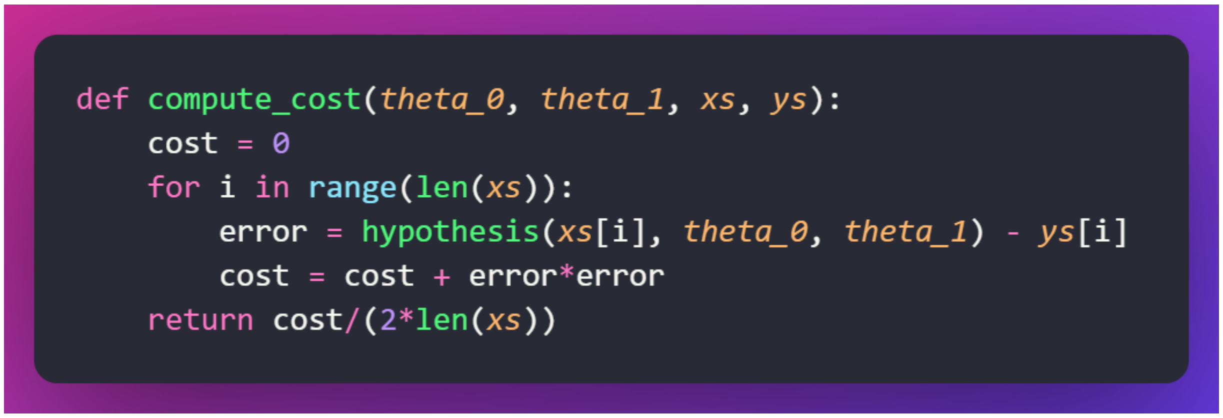
## II. Code the algorithm



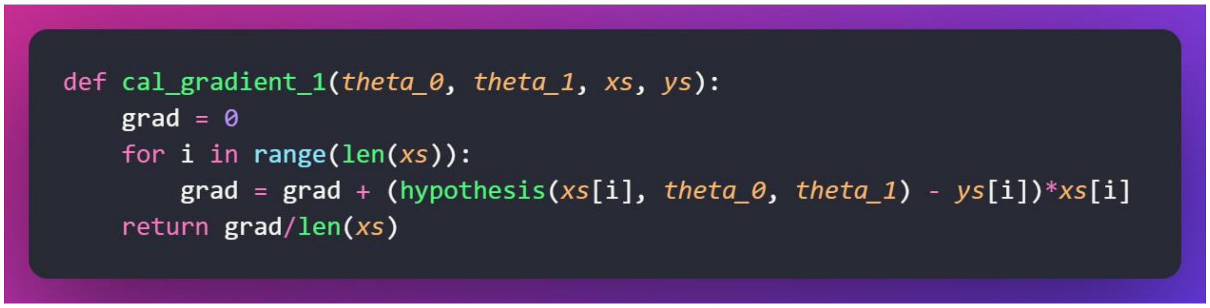
```python
import matplotlib.pyplot as plt
from random import random
```



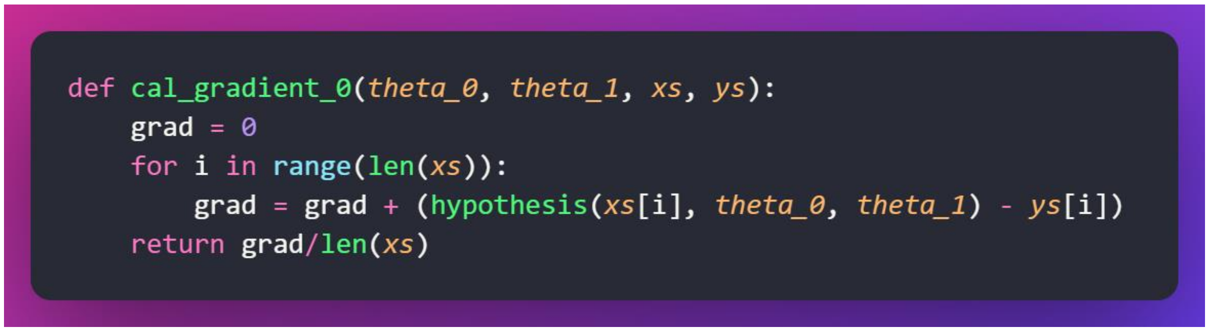
```python
def hypothesis(x, theta_0, theta_1):
    return x*theta_1 + theta_0
```



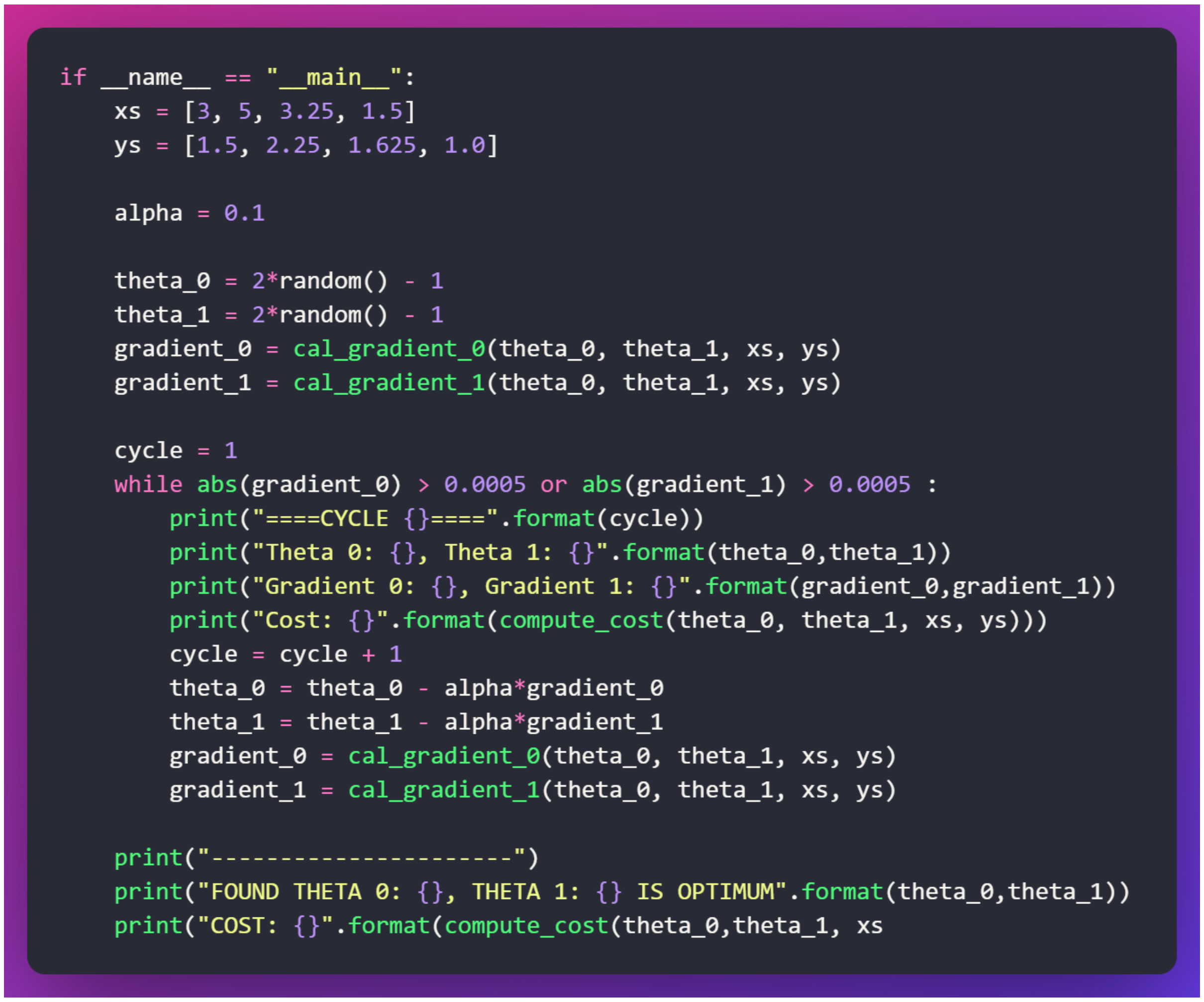
```python
def compute_cost(theta_0, theta_1, xs, ys):
    cost = 0
    for i in range(len(xs)):
        error = hypothesis(xs[i], theta_0, theta_1) - ys[i]
        cost = cost + error*error
    return cost/(2*len(xs))
```



```python
def cal_gradient_1(theta_0, theta_1, xs, ys):
    grad = 0
    for i in range(len(xs)):
        grad = grad + (hypothesis(xs[i], theta_0, theta_1) - ys[i])*xs[i]
    return grad/len(xs)
```

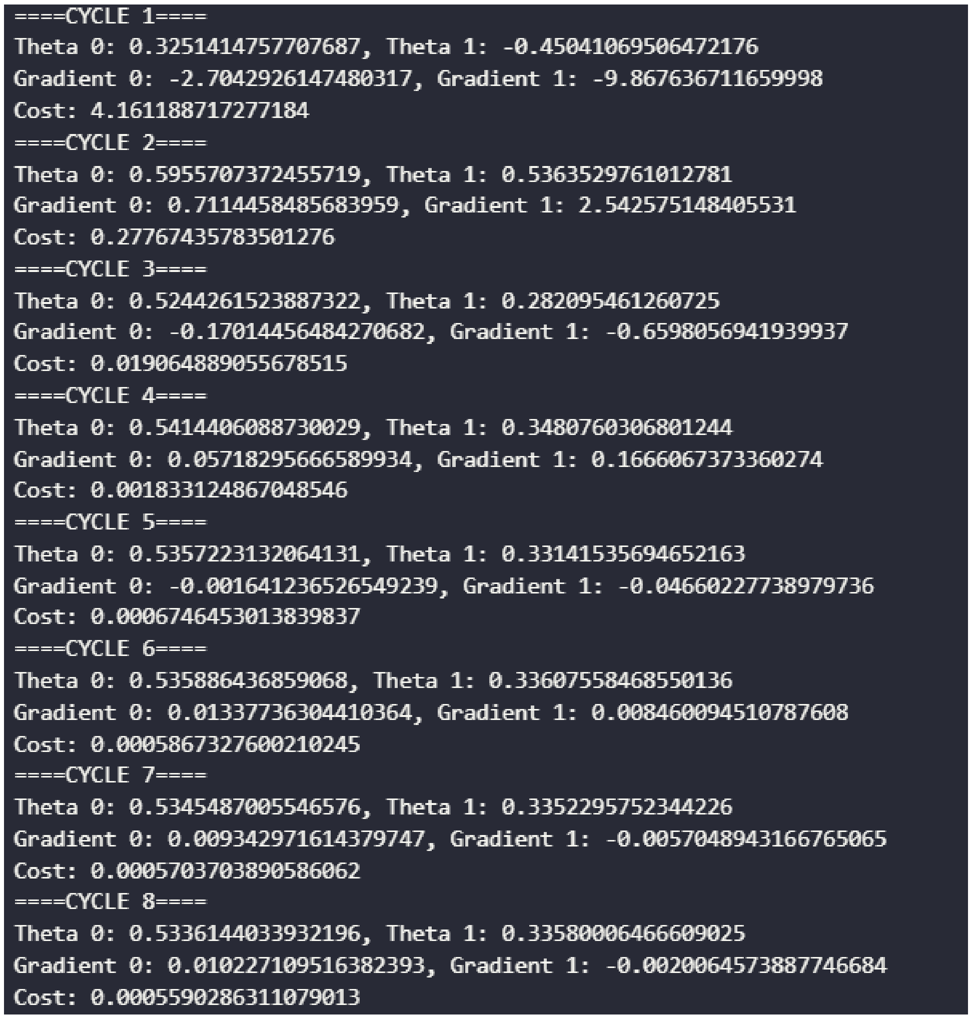


```python
def cal_gradient_0(theta_0, theta_1, xs, ys):
    grad = 0
    for i in range(len(xs)):
        grad = grad + (hypothesis(xs[i], theta_0, theta_1) - ys[i])
    return grad/len(xs)
```

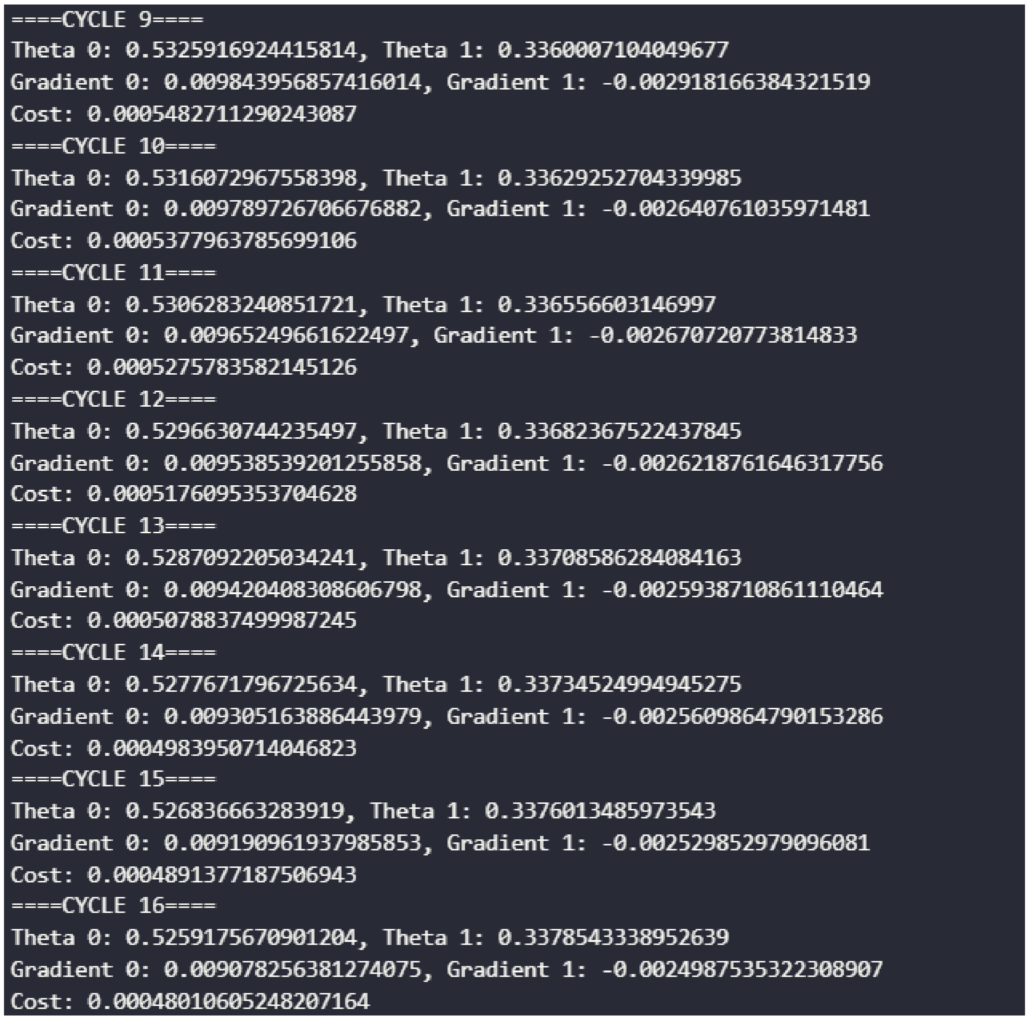


```python
if __name__ == "__main__":
    xs = [3, 5, 3.25, 1.5]
    ys = [1.5, 2.25, 1.625, 1.0]

    alpha = 0.1

    theta_0 = 2*random() - 1
    theta_1 = 2*random() - 1
    gradient_0 = cal_gradient_0(theta_0, theta_1, xs, ys)
    gradient_1 = cal_gradient_1(theta_0, theta_1, xs, ys)

    cycle = 1
    while abs(gradient_0) > 0.0005 or abs(gradient_1) > 0.0005 :
        print("====CYCLE {}====".format(cycle))
        print("Theta 0: {}, Theta 1: {}".format(theta_0,theta_1))
        print("Gradient 0: {}, Gradient 1: {}".format(gradient_0,gradient_1))
        print("Cost: {}".format(compute_cost(theta_0, theta_1, xs, ys)))
        cycle = cycle + 1
        theta_0 = theta_0 - alpha*gradient_0
        theta_1 = theta_1 - alpha*gradient_1
        gradient_0 = cal_gradient_0(theta_0, theta_1, xs, ys)
        gradient_1 = cal_gradient_1(theta_0, theta_1, xs, ys)

    print("----------------------")
    print("FOUND THETA 0: {}, THETA 1: {} IS OPTIMUM".format(theta_0,theta_1))
    print("COST: {}".format(compute_cost(theta_0,theta_1, xs
```

**Result:**

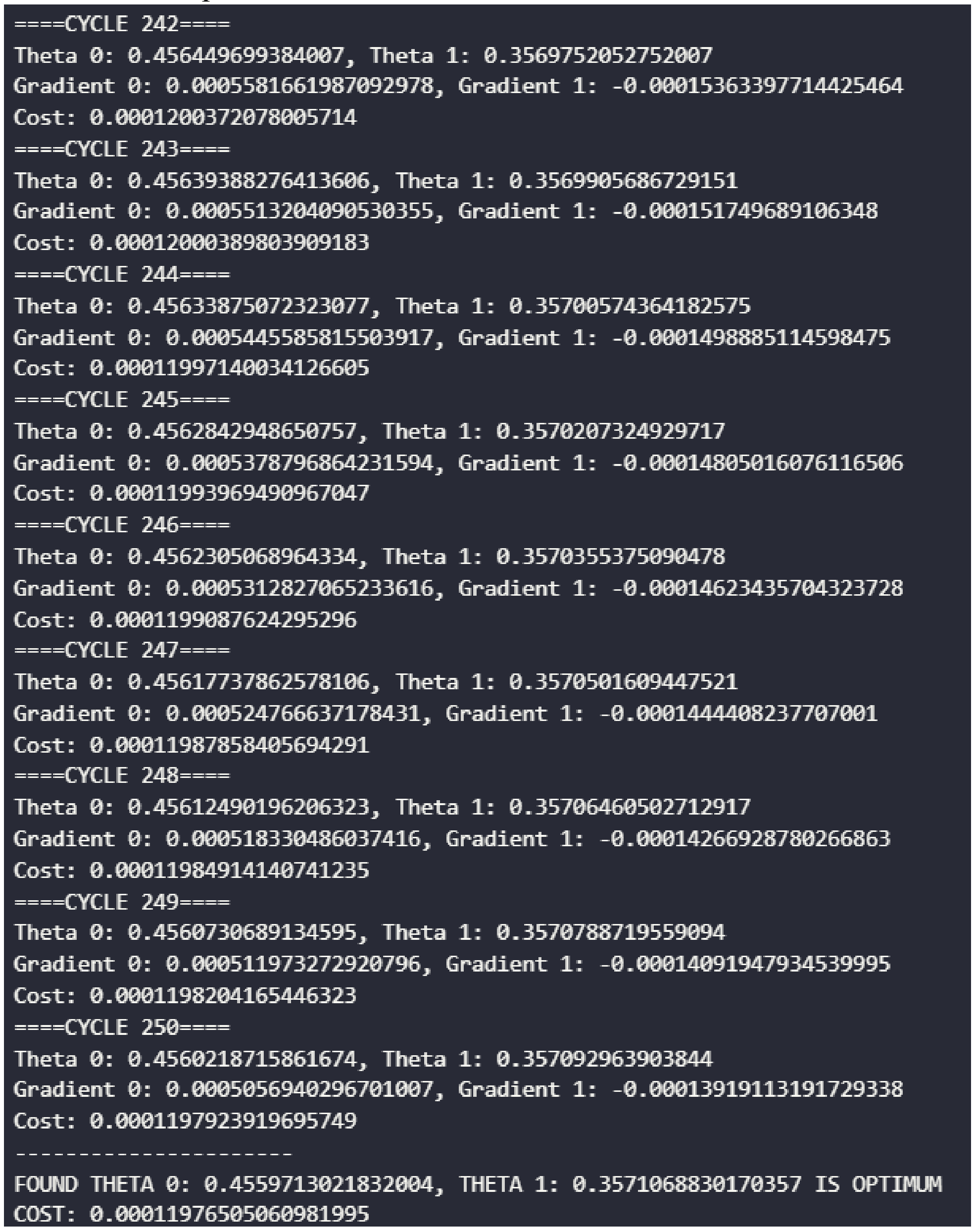


```
====CYCLE 1====
Theta 0: 0.3251414757707687, Theta 1: -0.45041069506472176
Gradient 0: -2.7042926147480317, Gradient 1: -9.867636711659998
Cost: 4.161188717277184
====CYCLE 2====
Theta 0: 0.5955707372455719, Theta 1: 0.5363529761012781
Gradient 0: 0.7114458485683959, Gradient 1: 2.542575148405531
Cost: 0.27767435783501276
====CYCLE 3====
Theta 0: 0.5244261523887322, Theta 1: 0.282095461260725
Gradient 0: -0.17014456484270682, Gradient 1: -0.6598056941939937
Cost: 0.019064889055678515
====CYCLE 4====
Theta 0: 0.5414406088730029, Theta 1: 0.3480760306801244
Gradient 0: 0.05718295666589934, Gradient 1: 0.1666067373360274
Cost: 0.001833124867048546
====CYCLE 5====
Theta 0: 0.5357223132064131, Theta 1: 0.33141535694652163
Gradient 0: -0.001641236526549239, Gradient 1: -0.04660227738979736
Cost: 0.0006746453013839837
====CYCLE 6====
Theta 0: 0.535886436859068, Theta 1: 0.33607558468550136
Gradient 0: 0.01337736304410364, Gradient 1: 0.008460094510787608
Cost: 0.0005867327600210245
====CYCLE 7====
Theta 0: 0.5345487005546576, Theta 1: 0.3352295752344226
Gradient 0: 0.009342971614379747, Gradient 1: -0.0057048943166765065
Cost: 0.0005703703890586062
====CYCLE 8====
Theta 0: 0.5336144033932196, Theta 1: 0.33580006466609025
Gradient 0: 0.010227109516382393, Gradient 1: -0.0020064573887746684
Cost: 0.0005590286311079013
```



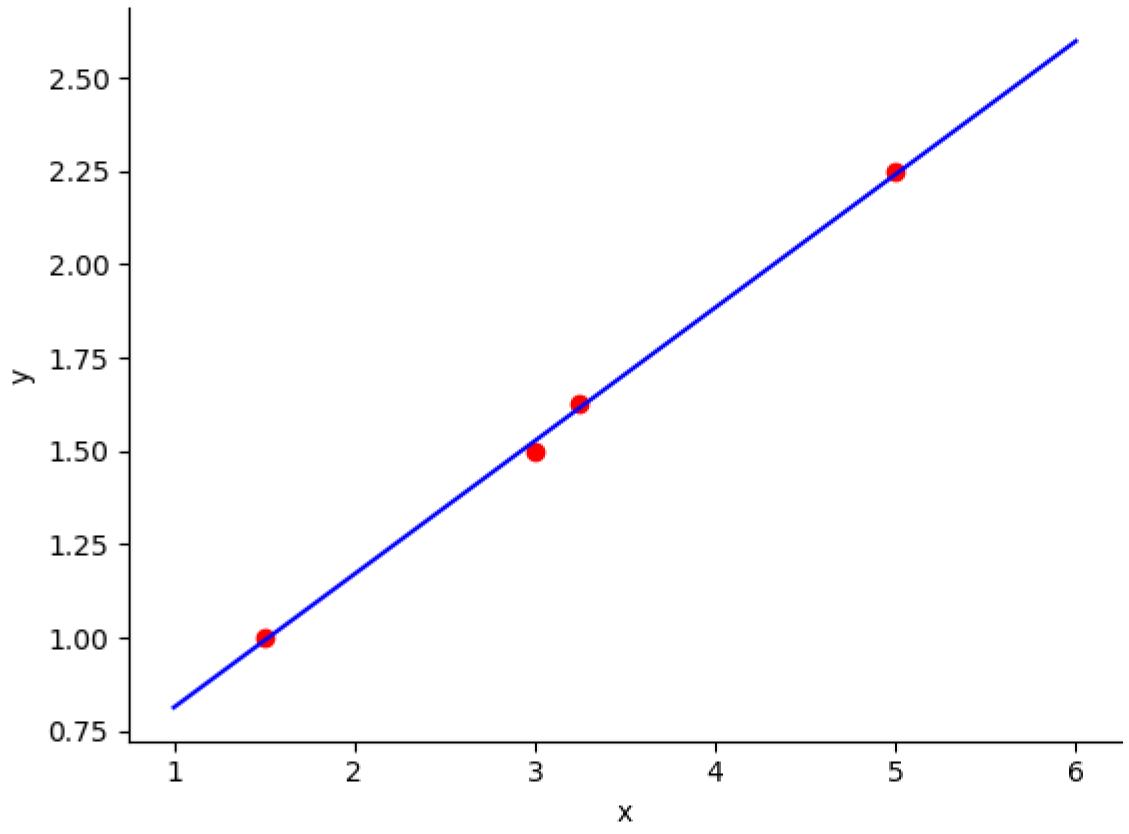
```
====CYCLE 9====
Theta 0: 0.5325916924415814, Theta 1: 0.3360007104049677
Gradient 0: 0.009843956857416014, Gradient 1: -0.002918166384321519
Cost: 0.0005482711290243087
====CYCLE 10====
Theta 0: 0.5316072967558398, Theta 1: 0.33629252704339985
Gradient 0: 0.009789726706676882, Gradient 1: -0.002640761035971481
Cost: 0.0005377963785699106
====CYCLE 11====
Theta 0: 0.5306283240851721, Theta 1: 0.336556603146997
Gradient 0: 0.00965249661622497, Gradient 1: -0.002670720773814833
Cost: 0.0005275783582145126
====CYCLE 12====
Theta 0: 0.5296630744235497, Theta 1: 0.33682367522437845
Gradient 0: 0.009538539201255858, Gradient 1: -0.0026218761646317756
Cost: 0.0005176095353704628
====CYCLE 13====
Theta 0: 0.5287092205034241, Theta 1: 0.33708586284084163
Gradient 0: 0.009420408308606798, Gradient 1: -0.0025938710861110464
Cost: 0.0005078837499987245
====CYCLE 14====
Theta 0: 0.5277671796725634, Theta 1: 0.33734524994945275
Gradient 0: 0.009305163886443979, Gradient 1: -0.0025609864790153286
Cost: 0.0004983950714046823
====CYCLE 15====
Theta 0: 0.526836663283919, Theta 1: 0.3376013485973543
Gradient 0: 0.009190961937985853, Gradient 1: -0.002529852979096081
Cost: 0.0004891377187506943
====CYCLE 16====
Theta 0: 0.5259175670901204, Theta 1: 0.3378543338952639
Gradient 0: 0.009078256381274075, Gradient 1: -0.0024987535322308907
Cost: 0.00048010605248207164
```

It repeats more times until optimum



```
====CYCLE 242====
Theta 0: 0.456449699384007, Theta 1: 0.3569752052752007
Gradient 0: 0.0005581661987092978, Gradient 1: -0.00015363397714425464
Cost: 0.0001200372078005714
====CYCLE 243====
Theta 0: 0.45639388276413606, Theta 1: 0.3569905686729151
Gradient 0: 0.0005513204090530355, Gradient 1: -0.000151749689106348
Cost: 0.00012000389803909183
====CYCLE 244====
Theta 0: 0.45633875072323077, Theta 1: 0.35700574364182575
Gradient 0: 0.0005445585815503917, Gradient 1: -0.0001498885114598475
Cost: 0.00011997140034126605
====CYCLE 245====
Theta 0: 0.4562842948650757, Theta 1: 0.3570207324929717
Gradient 0: 0.0005378796864231594, Gradient 1: -0.00014805016076116506
Cost: 0.00011993969490967047
====CYCLE 246====
Theta 0: 0.4562305068964334, Theta 1: 0.3570355375090478
Gradient 0: 0.0005312827065233616, Gradient 1: -0.00014623435704323728
Cost: 0.0001199087624295296
====CYCLE 247====
Theta 0: 0.45617737862578106, Theta 1: 0.3570501609447521
Gradient 0: 0.000524766637178431, Gradient 1: -0.0001444408237707001
Cost: 0.00011987858405694291
====CYCLE 248====
Theta 0: 0.45612490196206323, Theta 1: 0.35706460502712917
Gradient 0: 0.000518330486037416, Gradient 1: -0.00014266928780266863
Cost: 0.00011984914140741235
====CYCLE 249====
Theta 0: 0.4560730689134595, Theta 1: 0.3570788719559094
Gradient 0: 0.000511973272920796, Gradient 1: -0.00014091947934539995
Cost: 0.0001198204165446323
====CYCLE 250====
Theta 0: 0.4560218715861674, Theta 1: 0.357092963903844
Gradient 0: 0.0005056940296701007, Gradient 1: -0.00013919113191729338
Cost: 0.0001197923919695749
----------------------
FOUND THETA 0: 0.4559713021832004, THETA 1: 0.3571068830170357 IS OPTIMUM
COST: 0.00011976505060981995
```

Optimal weights



The straight line is found using Gradient Descent

## III. Different methods to determine the learning rate

1. Trial and error method

   This method is to try different values and observe their effects on the training process.
   - starting with a small value, such as 0.01, and increasing or decreasing it by a factor of 10 until you find a value that works well for network
   - monitoring the learning curves, which plot the loss and accuracy values against the number of epochs or iterations, to see how the network behaves with different learning rates
   - checking the gradients and weights of the network to see if they are converging or exploding.

   Note: The automation of this approach relies on reinforcement learning techniques like learning automata or Q-learning. By defining the continuous action space as a reinforcement learning problem and using errors to determine the reinforcement signal, it becomes possible to automate the entire process of parameter tuning.

2. Adaptive Learning Rate

   This method adjusts the learning rate during training based on the characteristics of the data and the optimization process
   - the initial network output and error are calculated. At each epoch new weights and biases are calculated using the current learning rate.
   - New outputs and errors are then calculated

   If the new error exceeds the old error by more than a predefined ratio (typically 1.04), the new weights and biases are discarded ➔ the learning rate is decreased

   Otherwise, the new weights are kept. If the new error is less than the old error, the learning rate is increased (typically by multiplying by 1.05)

3. Learning rate decay method

   This method is to reduce the learning rate over time as the network gets closer to the minimum of the loss function
   - starting with a relatively high learning rate to speed up the initial learning
   - Then gradually lower it to avoid overshooting and oscillating around the minimum

   There are different types of learning rate decay methods, such as step decay, exponential decay, inverse time decay, and adaptive decay

   Note: Using a predefined schedule or a dynamic rule to adjust the learning rate according to the progress of the training.

4. Learning rate finder method

   This method uses a heuristic to find a range of good learning rates for network
   - starting with a very low learning rate and increase it exponentially until the loss starts to increase rapidly
   - plot the loss against the learning rate and look for the point where the loss decreases the most.

   This point is usually a good estimate of the optimal learning rate, or at least a lower bound for it