****

**GRADIENT DESCENT**

**ARVIND PAWAR**

**NORTHEASTERN UNIVERSITY**

**WEEK 1**

**ALY6020 PREDICTIVE ANALYTICS**

**INSTRUCTOR NAME- DR. MARCO MONTE DE OCA**



Machine Learning is an application of Artificial Intelligence that makes systems able to automatically learn and improve from past experiences without being explicitly programmed. We train machine learning models with known input and output data to make future predictions and based on that we can solve complex real-world problems. However, how do we know our model will work well on new data. In machine learning, we often face **prediction errors**, which is the difference between the observed value and predicted value. We need to minimize the error to get accurate predictions. **Gradient Descent** is the most common optimization algorithm and one of the most critical concepts in Data Science. It is a **first-order optimization algorithm** refers to the task of **minimizing the cost/loss function**. First order optimization means it only considers the first derivative when performing the updates on the parameters. Cost functions are used to estimate how badly statistical models are performing and helps the model to correct or change behavior to minimize errors. It measures how wrong the model is in terms of its ability to determine the relationship between independent and response variables. We can estimate the cost function by iteratively running a model to compare the predictions with the observed values. Gradient descent algorithm **minimizes** the function by **iteratively** moving in **steepest descent** direction to find **local or global minima** of a function. To find the optimal point, we iterate the function by setting a learning rate. **Learning rate** is a step size using which we can cover each step to reach the optimal point. If we set high learning rate, then we can cover more ground steps, but it arises a risk of overshooting the lowest point since we would be considering slope which changes continuously. With a low learning rate, we can confidently move in the negative gradient direction since we are recalculating it. A low learning rate is more precise to get the optimal point but calculating gradient at small step size is time consuming and degrade the performance. A cost/loss function is a learning parameter and has its gradients. To minimize the error and make our model more accurate, we study the slope of the cost function’s curve that tells us in which direction to move and how to update parameters.

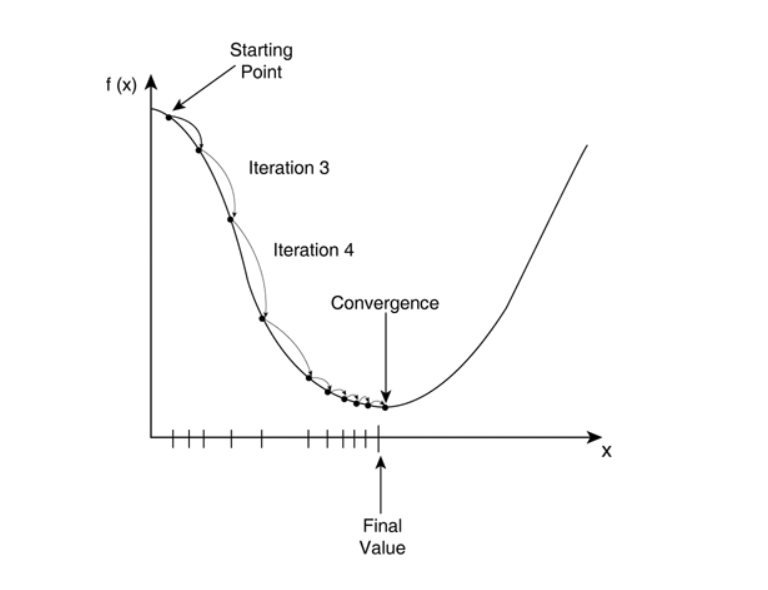


Fig. 1

In Fig. 1, We are starting from a point and running the cost function in iterations. Initially Gradient Descent takes long steps and as it comes near local minimum it automatically takes small steps and reach to the optimal point.

Let us consider the **gradient descent algorithm**, which starts with some initial θ, and repeatedly updates the parameters:**θj: = θj − α ∂/∂θj \*J(θ).**

Where, θ**j** is a parameter we want to choose to minimize J(θ). We have to start with random initial guess for θ, we will iteratively run the function with learning rate (α) to update θ to make J(θ) smaller until we converge to a value of θ that minimizes the cost function J(θ).

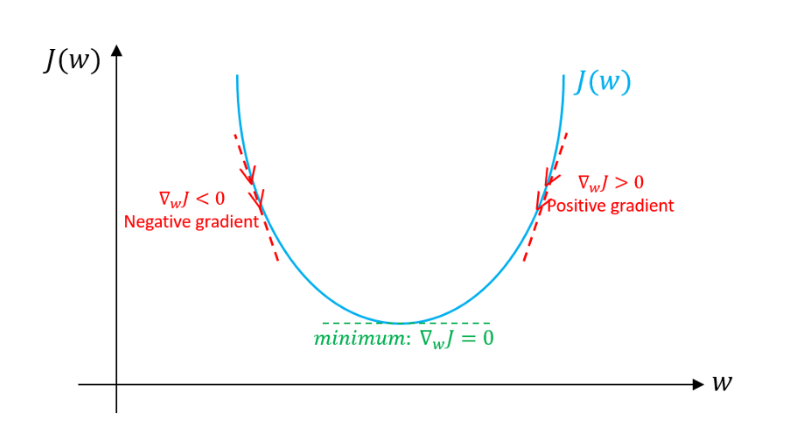


Fig. 2

In Fig. 2, If we intitialise gradient decsent with theta on the top right of the curve, we will get the positive slope, so our derivative will be positive. If we put that in the the function

θ**j**: = θj − α ∂/∂θj \*J(θ).

= θj − α \* (positive number)

So, theta minus positive number will make us move the theta 1 to the left direction which is heading towards the minimum point.

Similarly, if we start from the top left of the curve, we will get negative slope,

θ**j**: = θj − α ∂/∂θj \*J(θ).

= θj − α \* (negative number)

So, from the above calculation we are likely to increase our theta and motive iteration in the right direction toward minimum point.

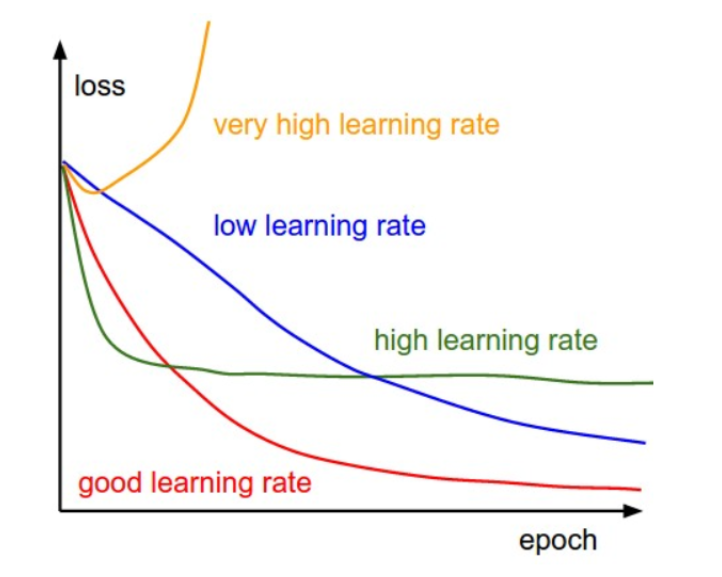
  
Fig. 3

Fig. 3-line graph is representing the curves of convergence of theta using different level of learning rate. Learning rate should not be very high, high learning rate, covers more ground steps, but it arises a risk of missing the lowest point since we would be considering slope which changes continuously.

Let us take an example of linear regression.

Approximate y as a linear function of x:

hθ(x) = θ0 + θ1x1 + θ2x2

Where, θi’s are the parameter and xi is the series of independent variables.

Now, given a training data set, how do we pick, or learn, the parameters θ? We want to determine one reasonable method to make h(x) (predicted values) close to y (observed values), at least for the training examples we have. To formalize this, we will define a function that measures, for each value of the θ’s, how close the h(x(i))’s are to the corresponding y(i)’s. We define the cost function: J(θ) = 1/2 \* m ∑ i=1 ( hθ(x (i) ) − y (i) ) 2 .

**∂/∂θj J(θ) = ∂/∂θj 1/ 2 \* (hθ(x) − y) 2**

**= 2 · 1/2 (hθ(x) − y)·∂/∂θj (hθ(x) − y)**

**= (hθ(x) − y)· ∂/∂θj (n∑ i=0 θixi – y)**

**= (hθ(x) − y)xj**

For a single training example, this gives the update rule 1:

θj := θj + α (y (i) − hθ(x (i) )) x (i) j .

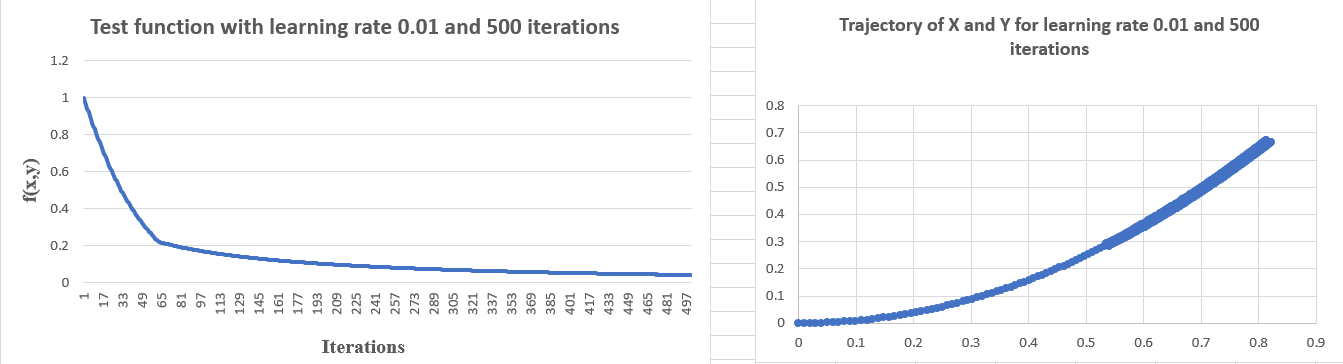
The update rule is called “least mean squares”

We have studied the Gradient Descent Algorithm in detail, let us see how it works.

**Task 1: We have given a test function “Rosenbrock function” and it is defined as: f(x,y) = (1-x)2+100(y-x2)2**

We have analyzed the update in parameters and convergence of a given test function using different learning rate and iterations.

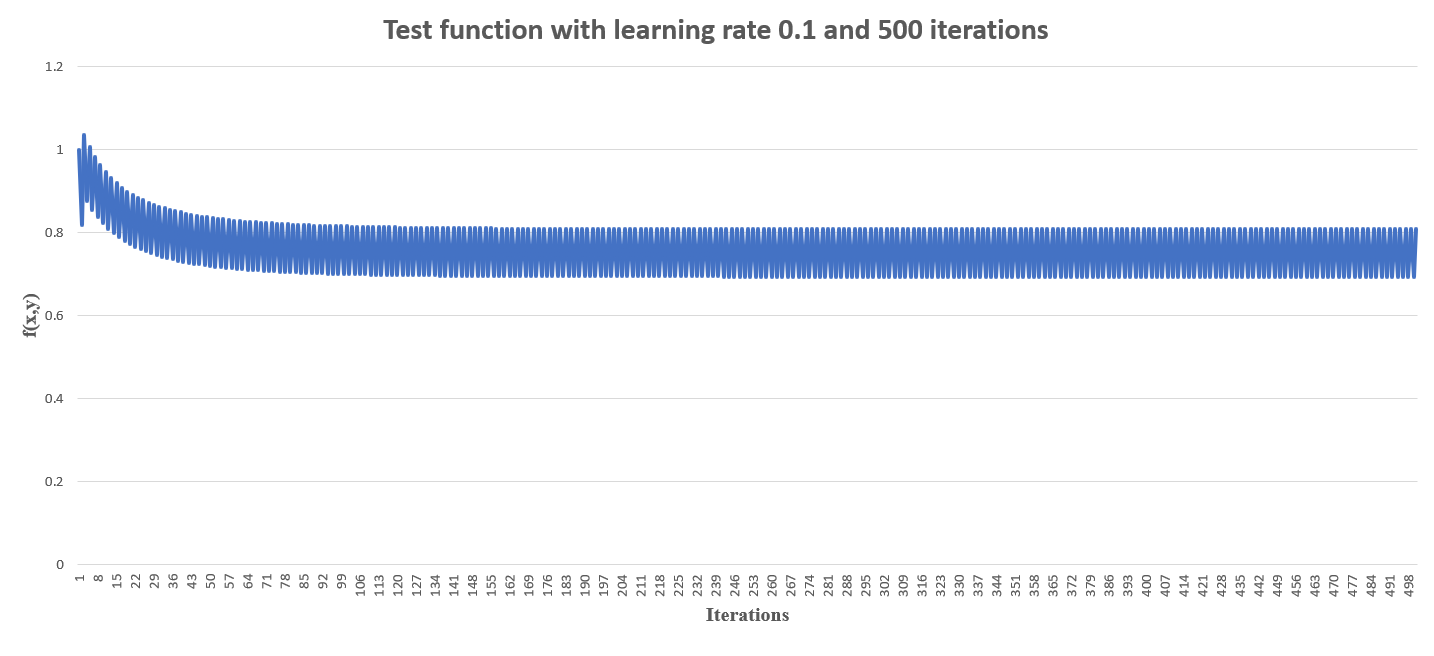
1. **Learning rate = 0.01 and Iterations = 500.**

****

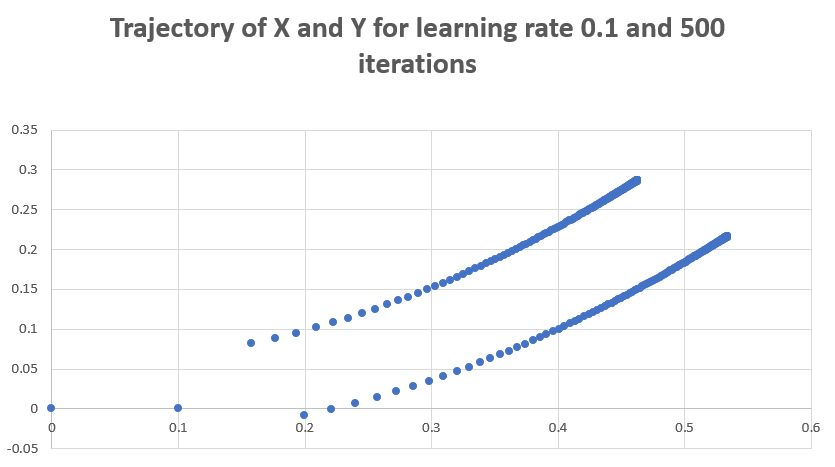
**Fig. 4**

In the 1st step, we have set the learning rate at 0.01, and we have run the function with 500 iterations. After around 337 iterations our loss function is moving toward the minimum point. From the trajectory of x and y graph, we can infer that it could identify the initial point from (0,0). As the number of iterations increasing our cost function is decreasing, and the solution estimate trajectory is moving towards the optimal point. If we have chosen starting points, let us say, (2, -2) then we can observe how (x,y) traverses through the negative axis to (0,0).

1. **Learning rate = 0.1 and Iterations = 500.**

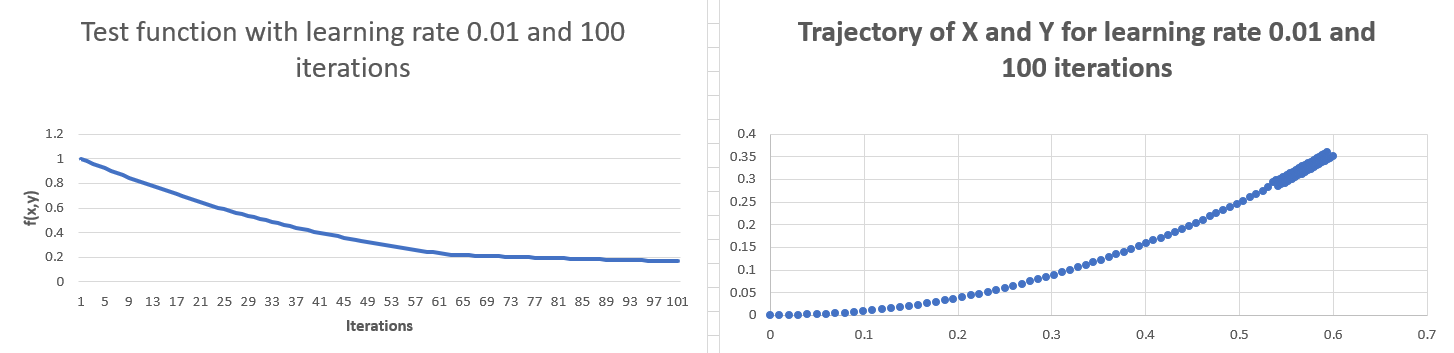
****

**Fig. 5**

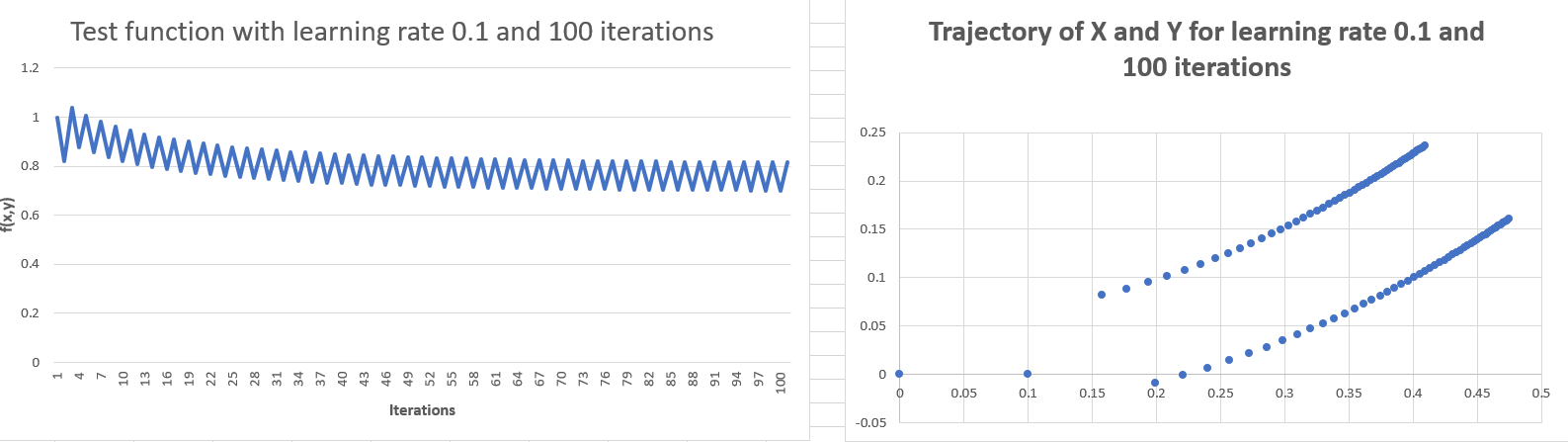
In the 2nd step, we have increased the learning rate from 0.01 to 0.1. We can see in fig 5 that our cost function is oscillating because of high step length and with these 500 iterations it is not able to reach to the minimum point. ****

**Fig. 6**

From the fig 6, we can see that initial iterations are taking steps by rate 0.1 and as it is moving toward the optimal point the magnitude of the slope reduces and the result between the product of the magnitude of slope and step length minimizes. Hence, it is taking slower steps. After much more iterations it can get converged.

****

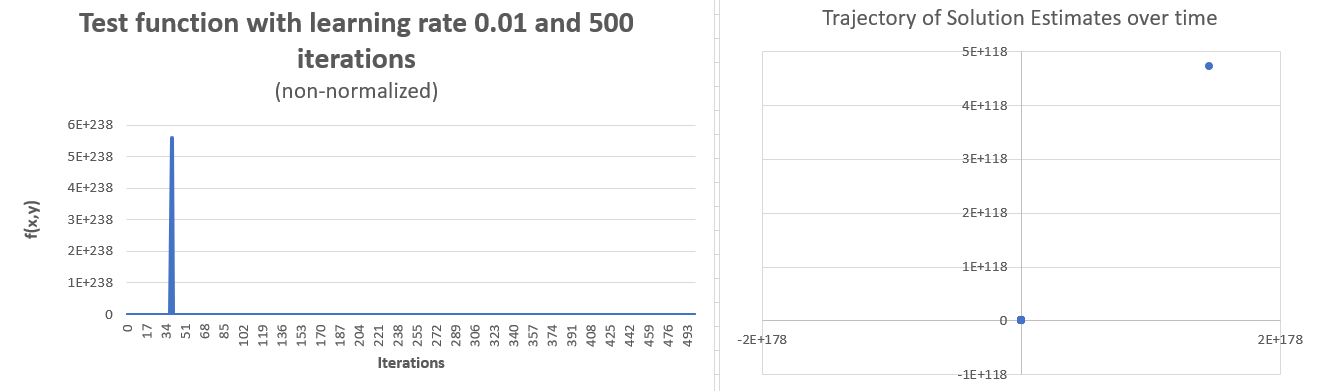
**Fig. 7**

****

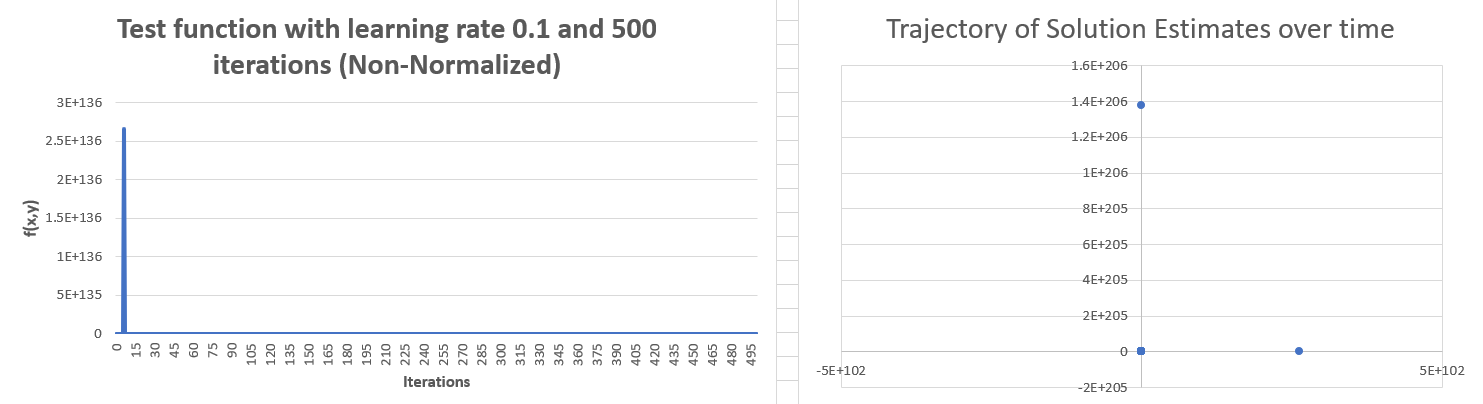
**Fig. 8**

In the fig. 7 and 8, we have minimized the number of iterations to 100. We observe the same patterns as we observed with 500 iterations. We can observe the graph with more details now.

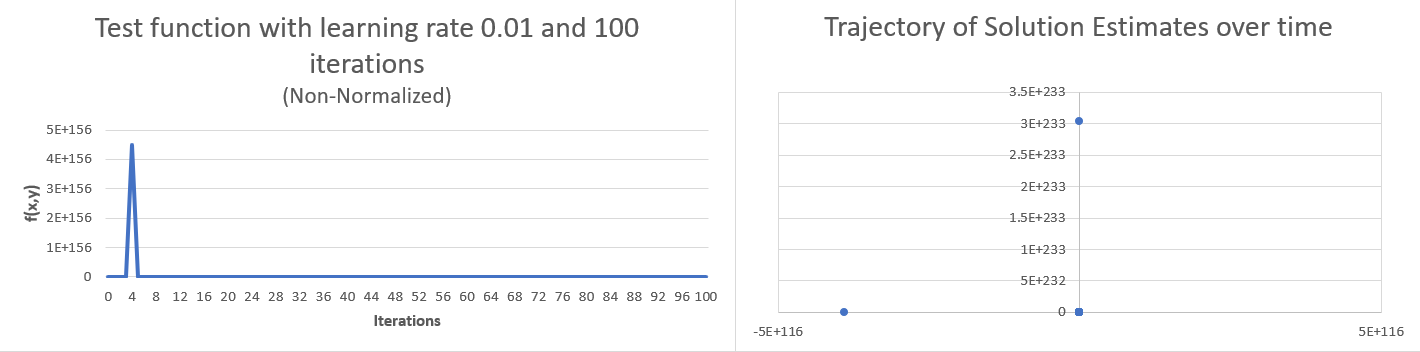
1. Analysis of non-normalized parameters.

****

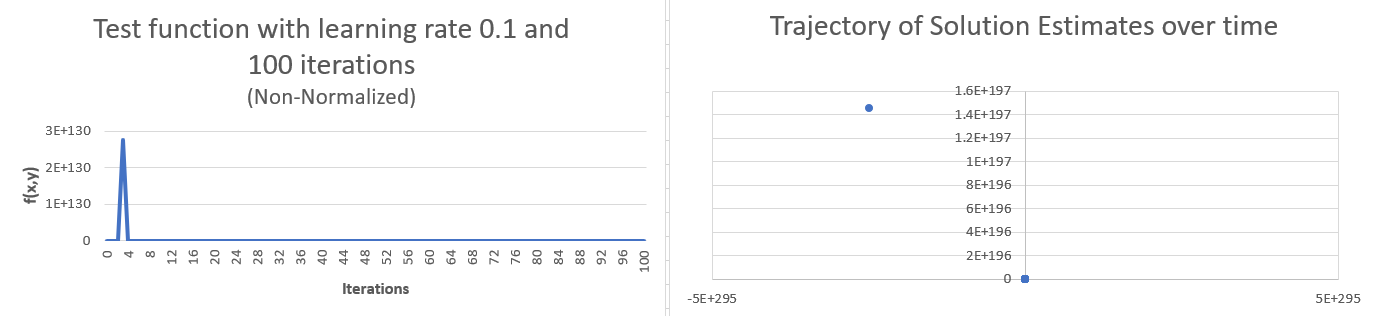
**Fig. 9**

****

**Fig. 10**

****

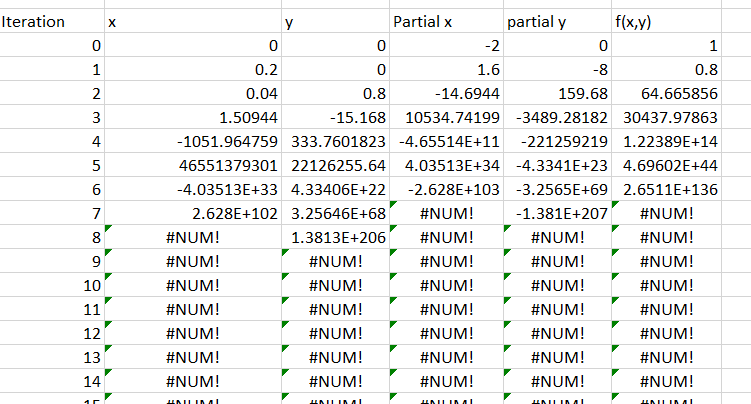
**Fig. 11**

****

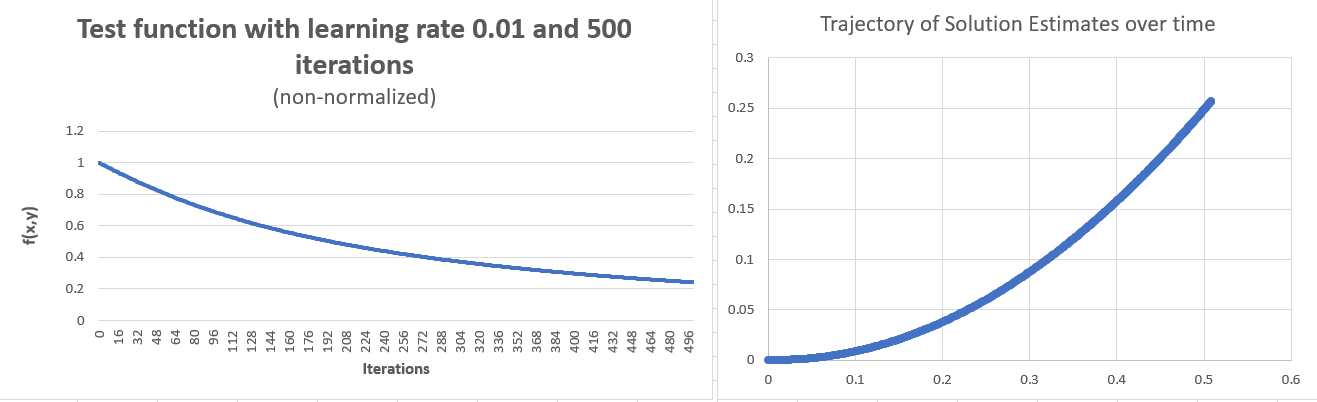
**Fig. 12**

The analysis for the above four figures is as below.

* The magnitude of the non-normalized parameters is usually high. Learning rates 0.01 and 0.1 that we have set for the given non-normalized derivatives of x and y are also high, that is why we are observing an increase in the cost function. It is leading to overshooting.
* Also, after some iterations, we are getting #NUM! Error because excel is not able to handle large, scaled values.



* If we choose the learning rate smaller than 0.01, then we would get better results. In the fig. 13, we have set the learning rate to 0.01 and iterations to 500, and we can observe better results. The x and y parameters have not reached the optimal point, and cost function has not converged yet. After some more iterations, it will get converged.

****

**Fig. 13**

**Conclusion:**

We have observed that our cost function can be minimized using gradient descent by tuning the learning rate and a number of iterations. From the visualization, we can infer that as we decrease the number of iterations, our cost function changes. We have run the cost functions for both normalized and non-normalized parameters. In the given cost function, both x and y are in different scales; therefore, we need to do normalization for optimization. From non-normalized parameters, we observed that it overshoots the cost function and keeping small learning rate we can get better results. However, a smaller learning rate for non-normalized parameters may take many iterations, and it may degrade the performance and time consuming to reach to the minimum point.

**References:**

* Gradient Descent Algorithm and Its Variants. (2017). Towards Data Science. Retrieved 30 May 2019, from <https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3>
* (2019). Cs229.stanford.edu. Retrieved 30 May 2019, from <http://cs229.stanford.edu/notes/cs229-notes1.pdf>
* Machine learning fundamentals (I): Cost functions and gradient descent. (2017). Towards Data Science. Retrieved 30 May 2019, from <https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220>