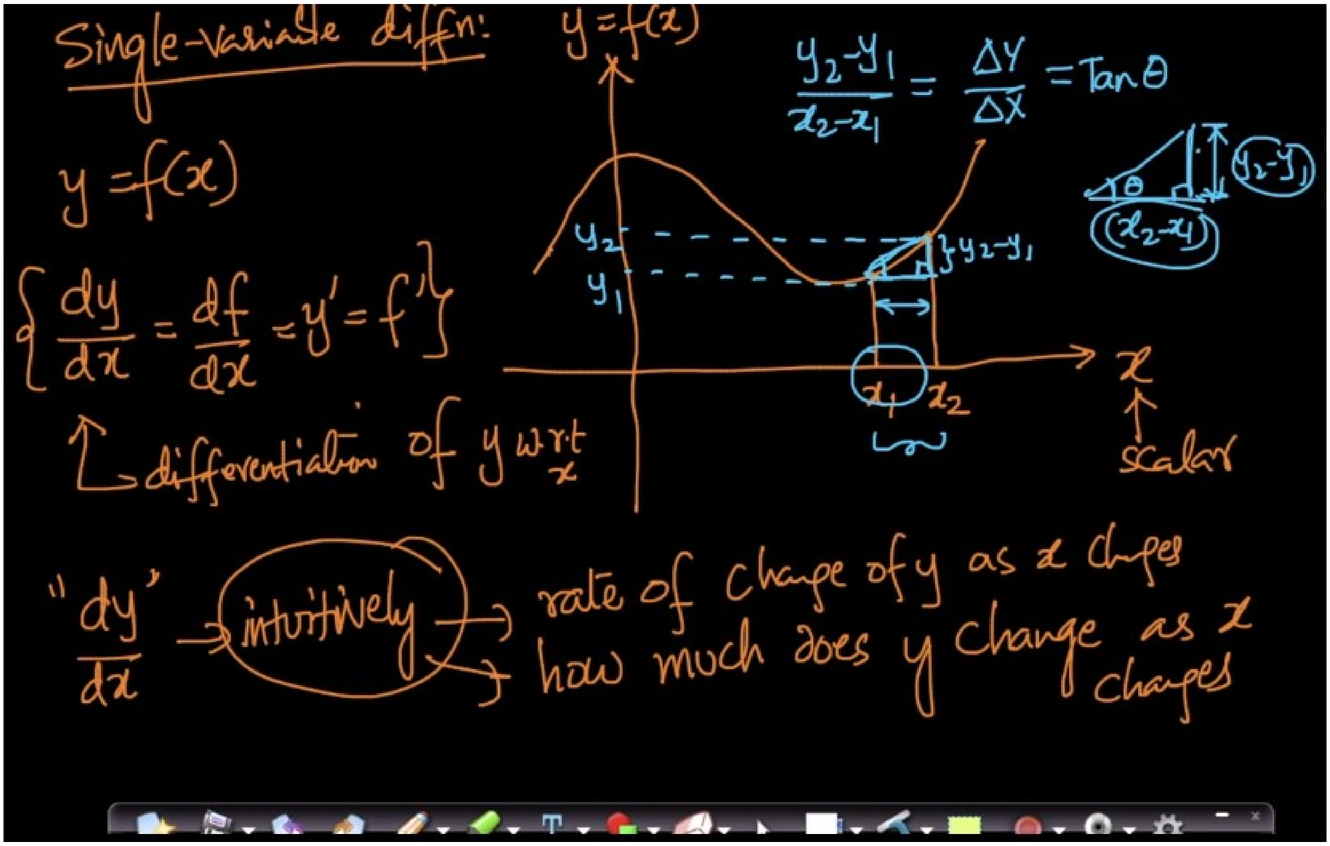
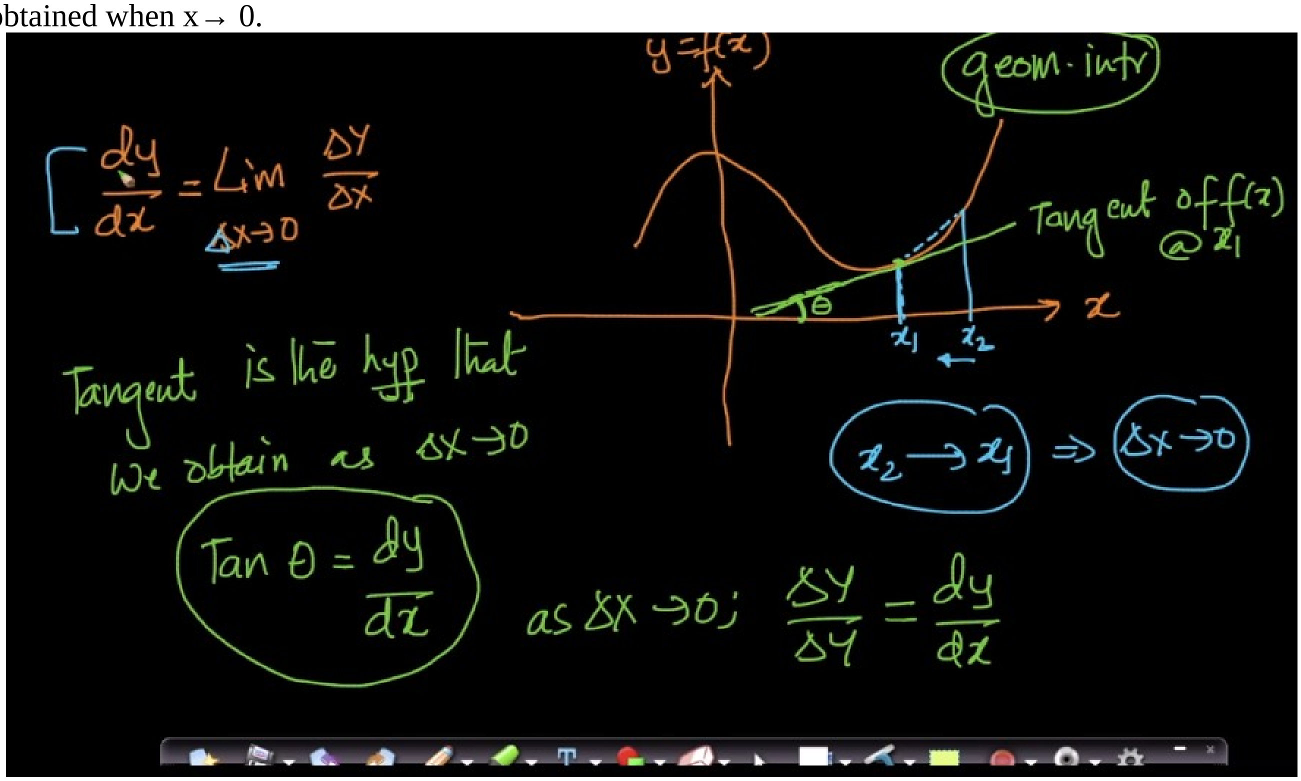
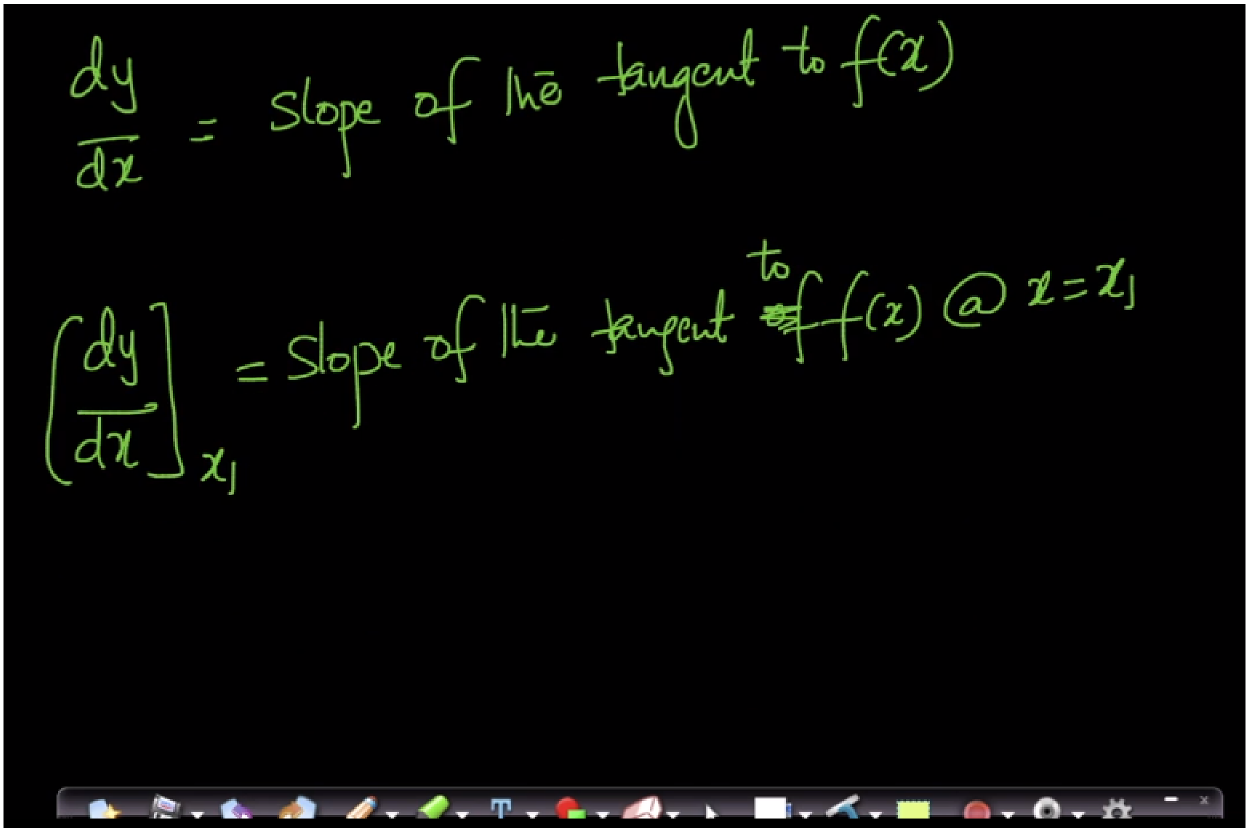
Solving optimization problems:
Single – variable differentiation:



Mathematical formulation:
When limit x → 0 , then it becomes a tangent at that point on the curve, tangent is the hyp that is obtained when x→ 0.

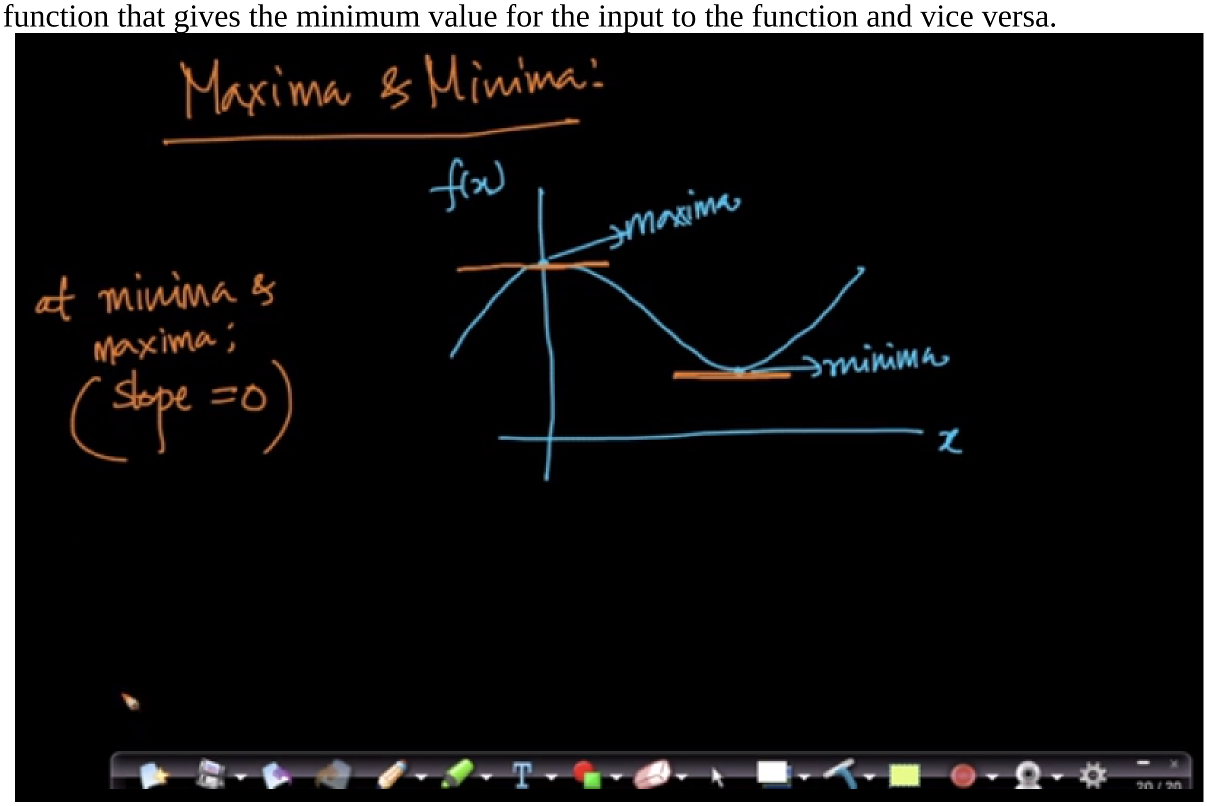




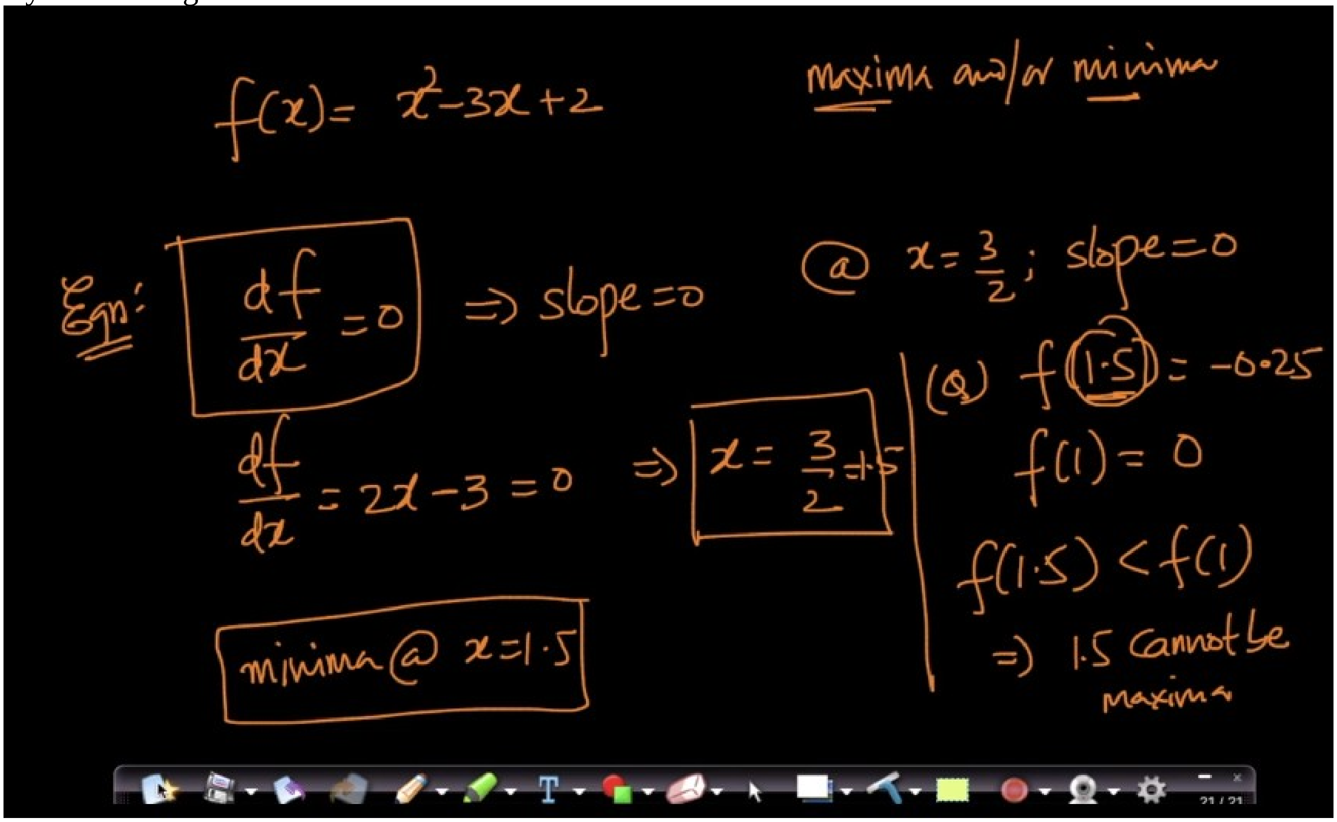
Geometrically dy / dx means the tangent at that point.

Minima and Maxima:

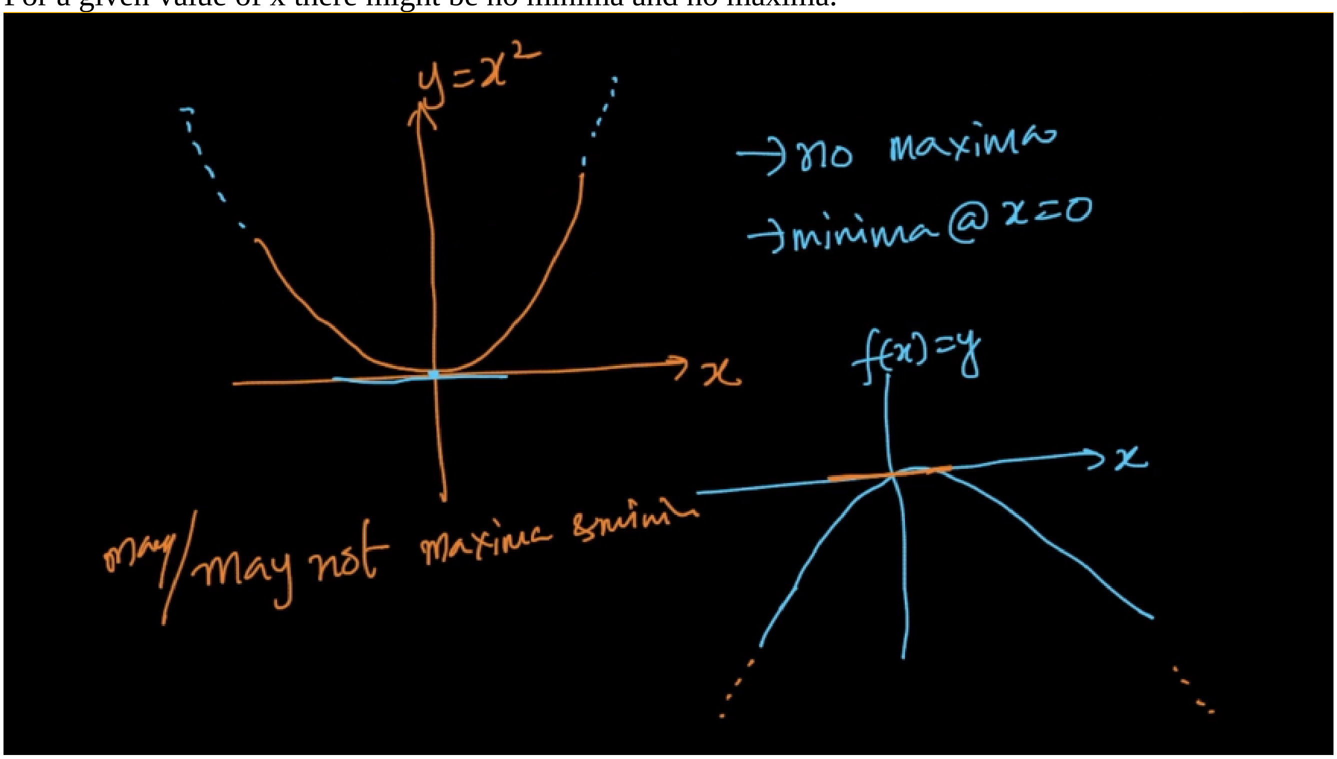
The function that gives the minimum value for the input to the function and vice versa.



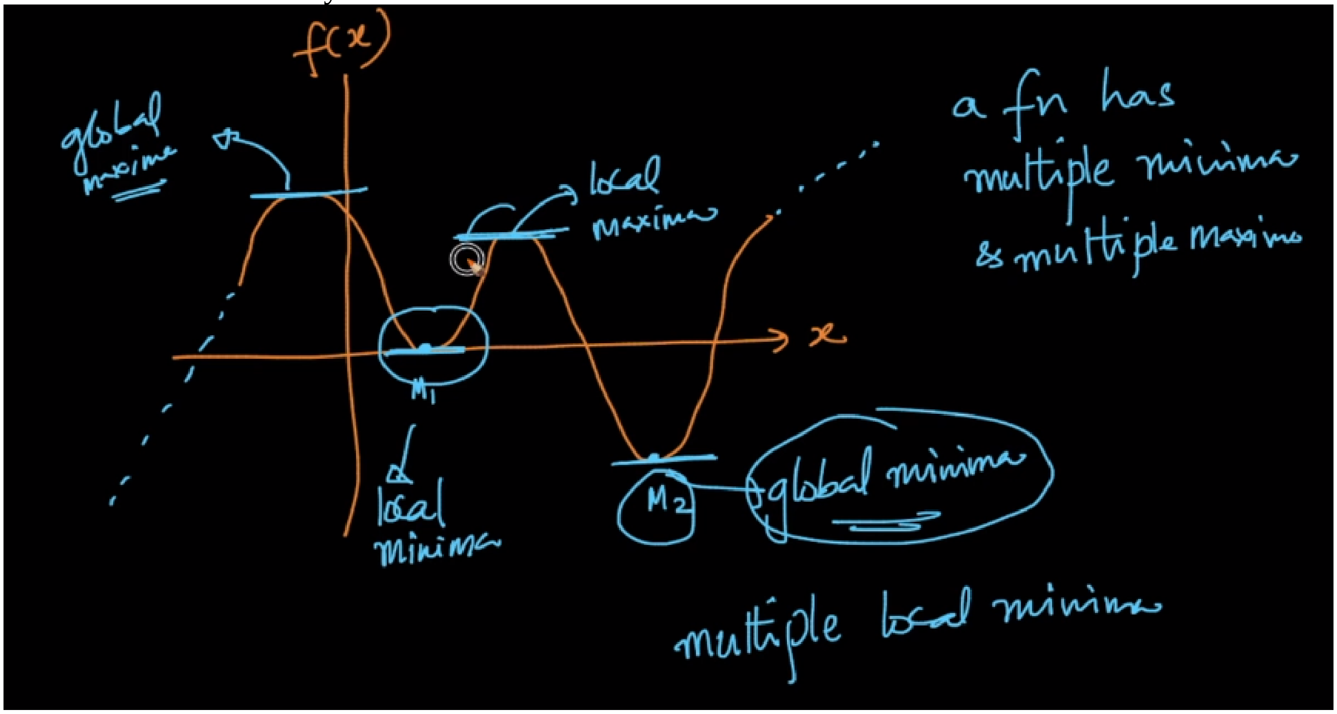
Example: The slope tangent the value of x for the derivative of the function in zero.
By substituting the values near to x we can know the nature of the function.



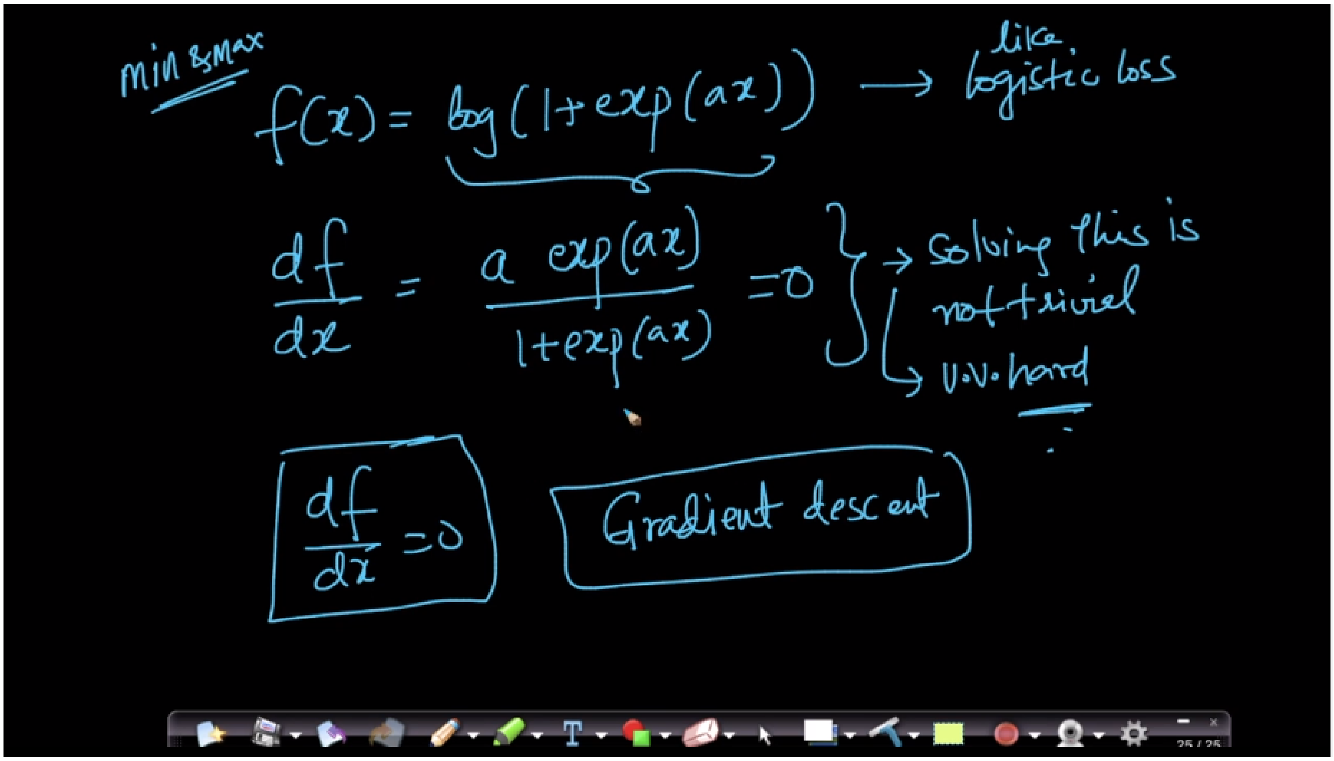
For a given value of x there might be no minima and no maxima.



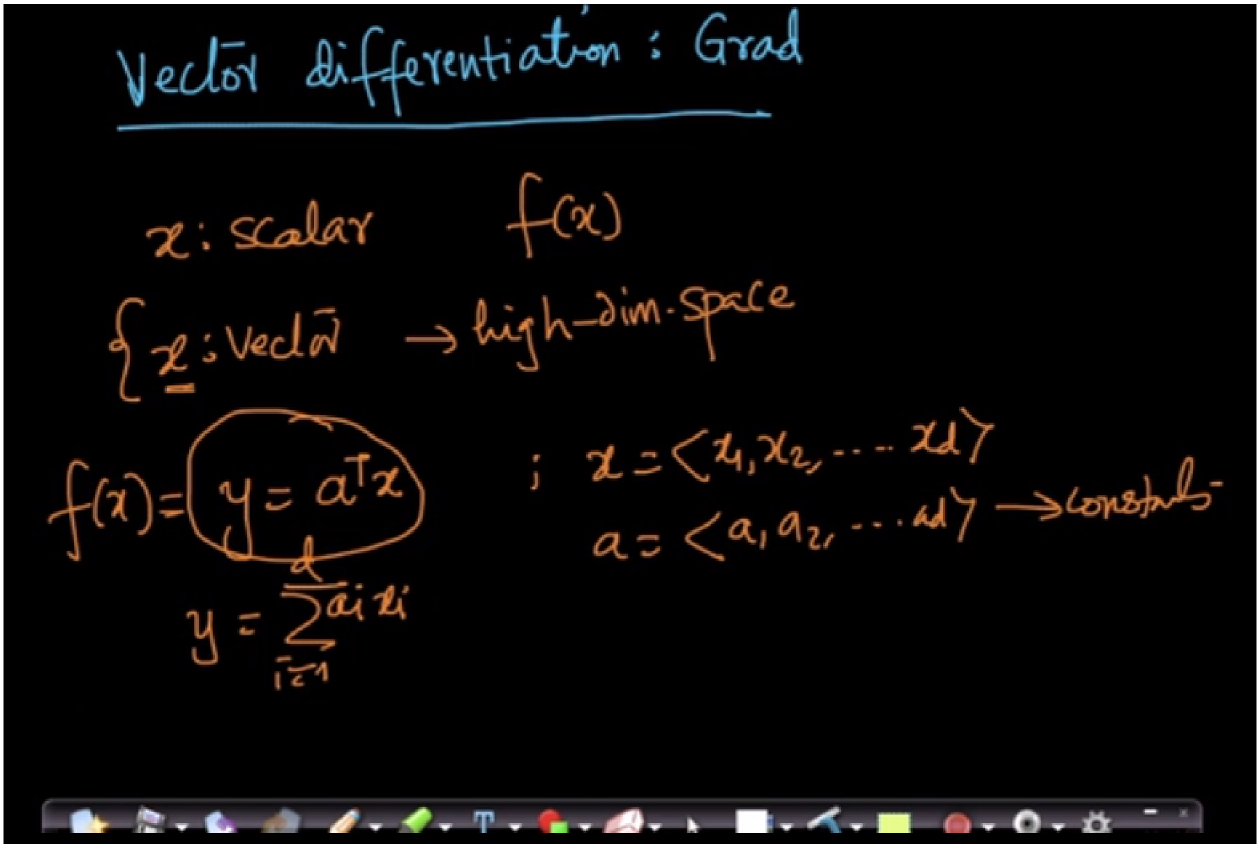
A function can have many number of minimas and maximas.



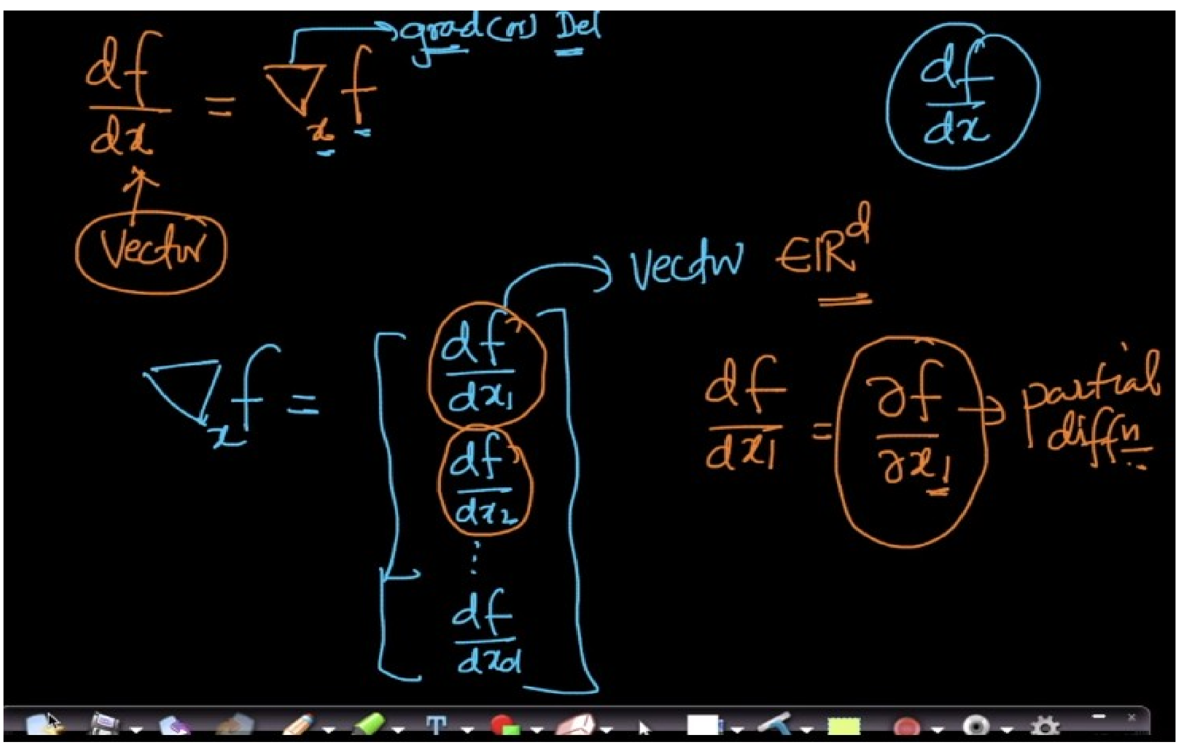
Non-trival cases of derivatives:



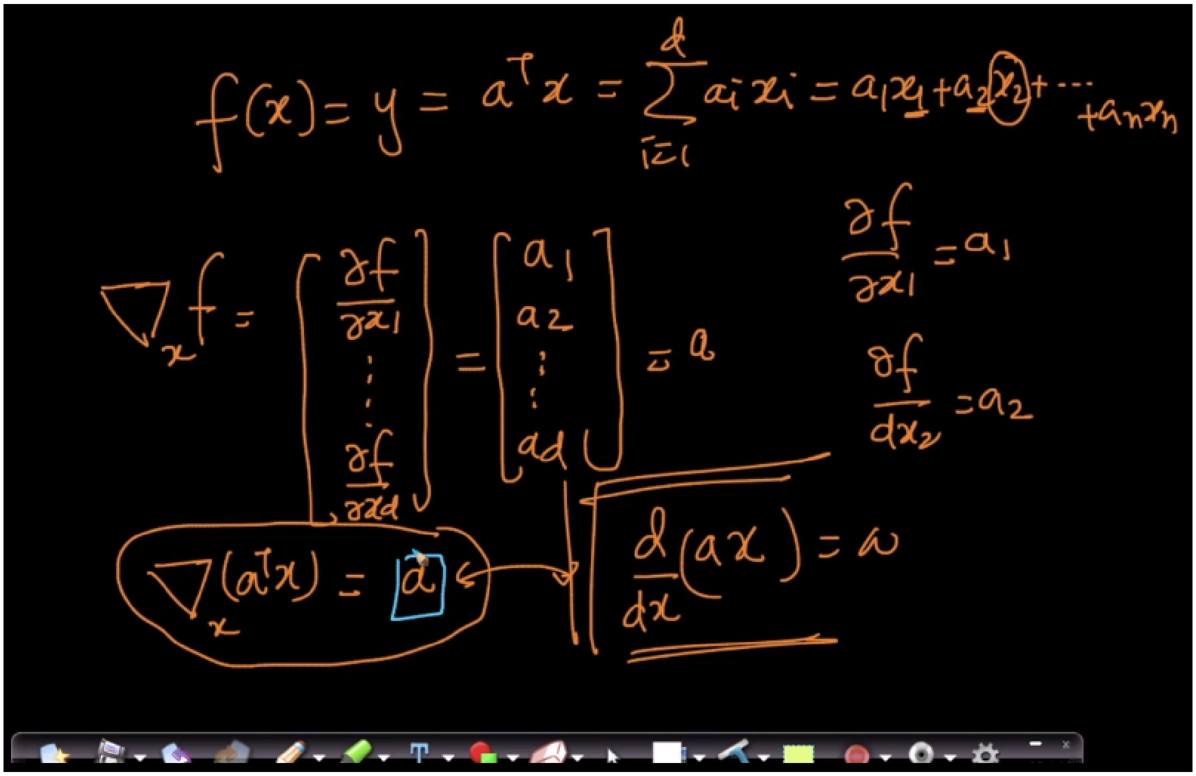
Vector differentiation:



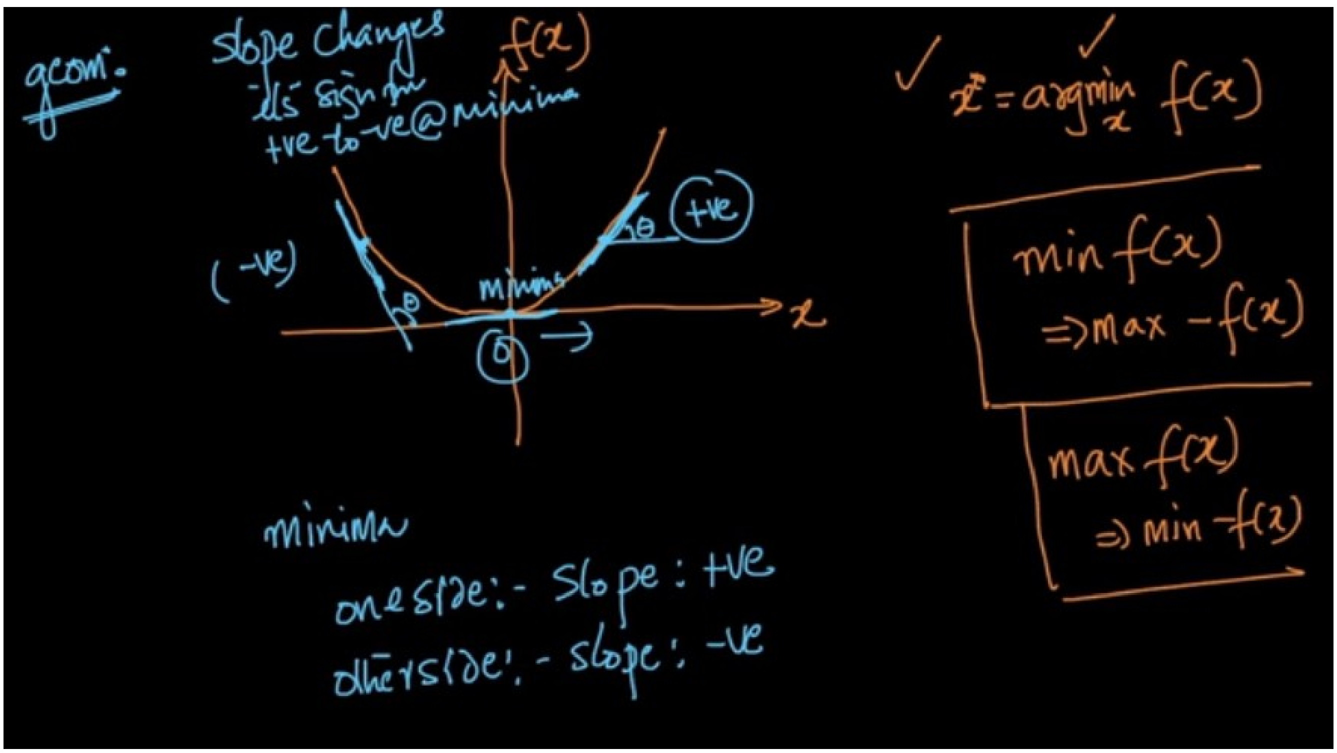
Differentiating a vector:



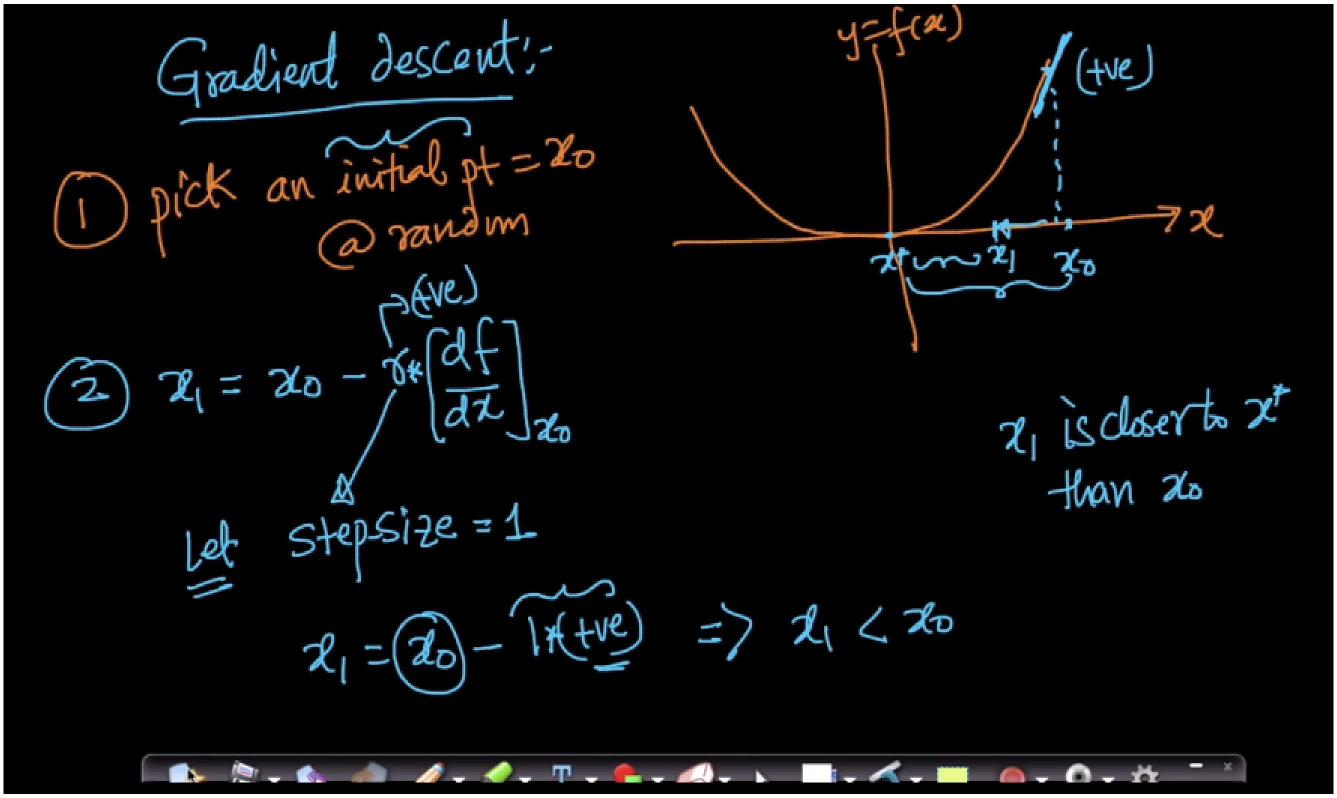
Differentiating a function f, which is a vector:



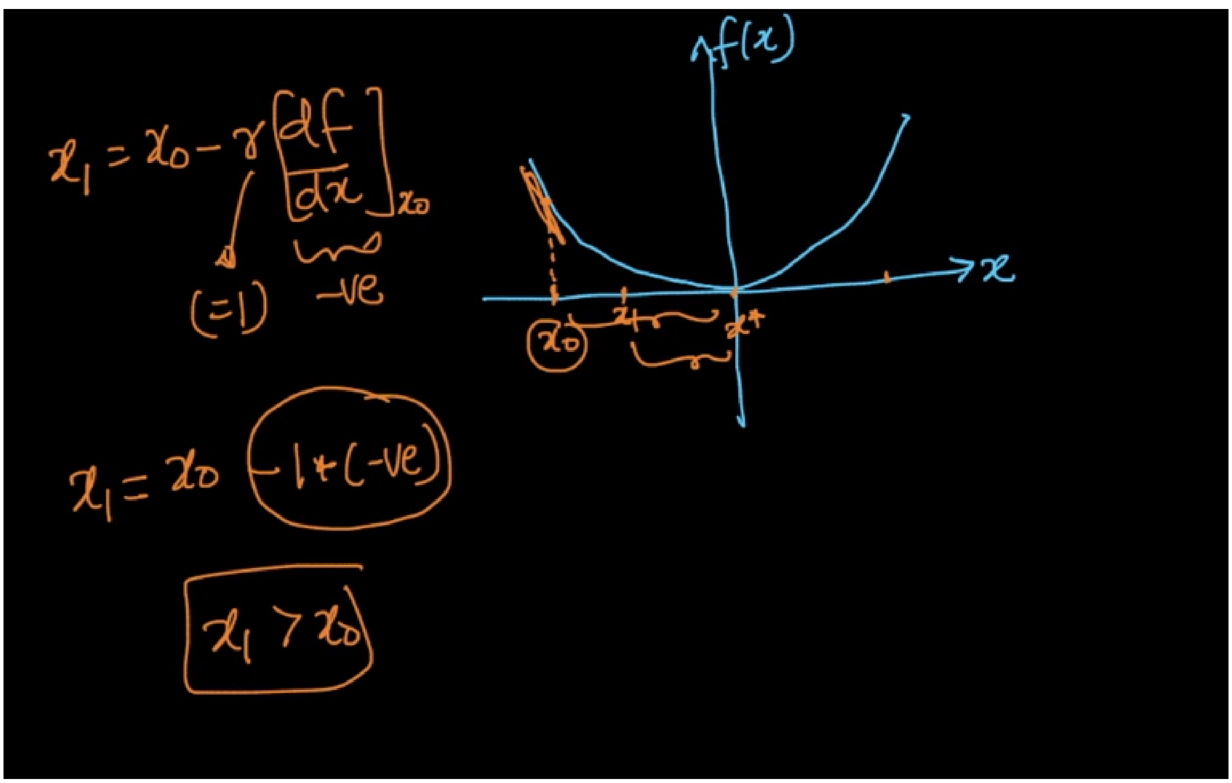
We use gradient descent for calculating the minimum of the function, because as the terms increase the functions become complex.
Gradient descent algorithm: It is an iterative algorithm. As the iterations increase the minimum value is achieved.



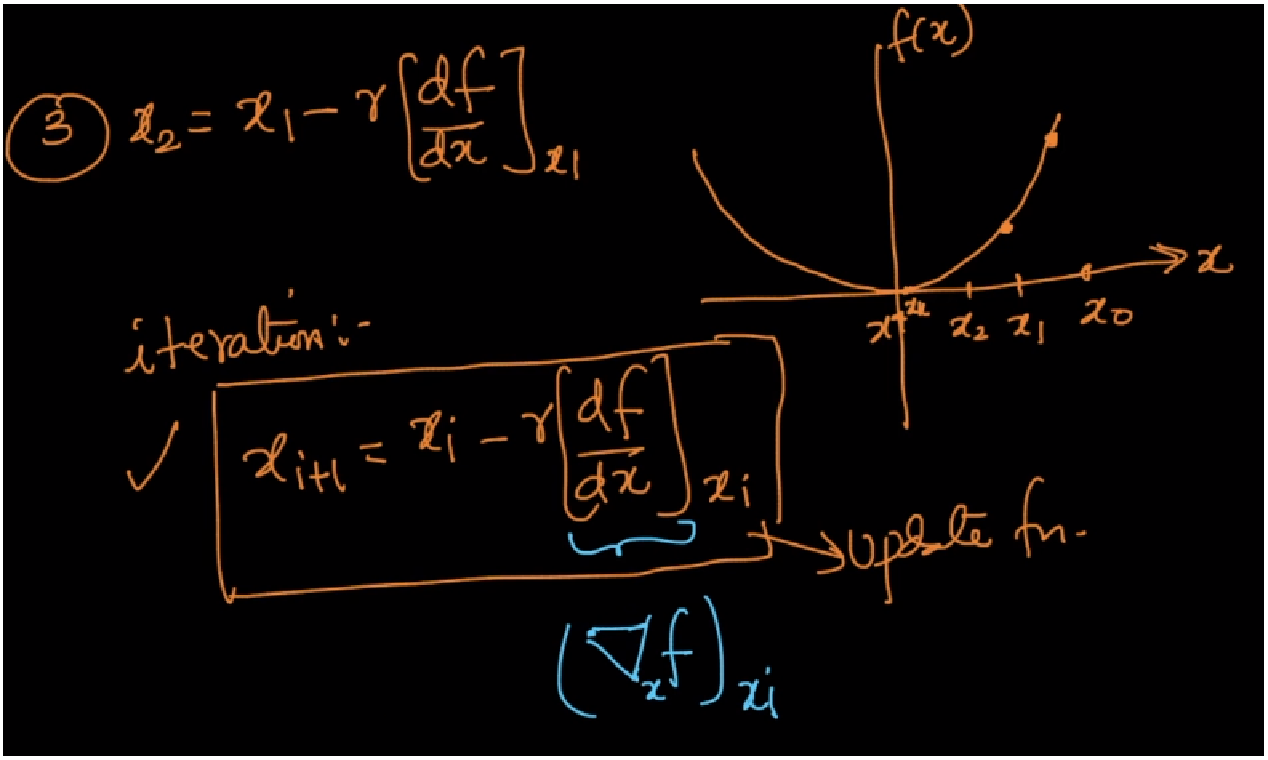
For positive slope.



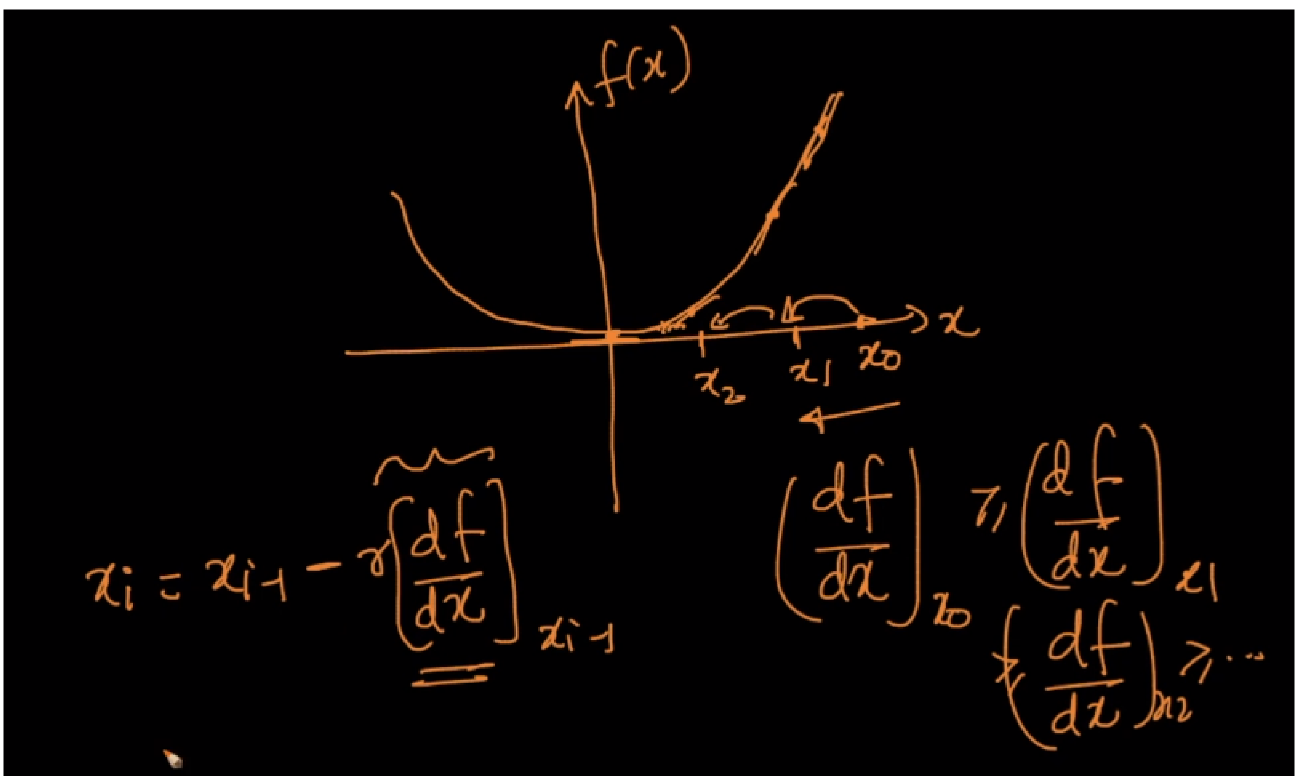
For negative slope.



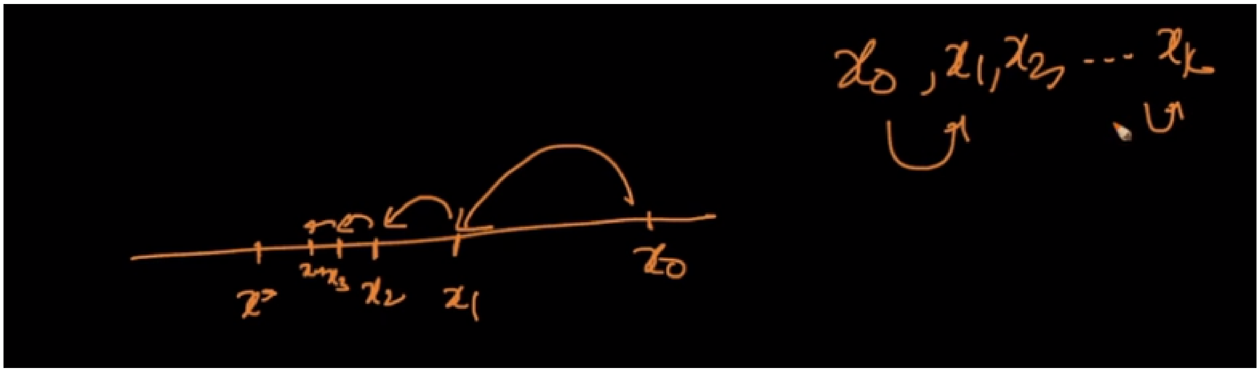
Another point on the curve is achieved by updating the step size by derivative at the previous point.



Property of Gradient descent:

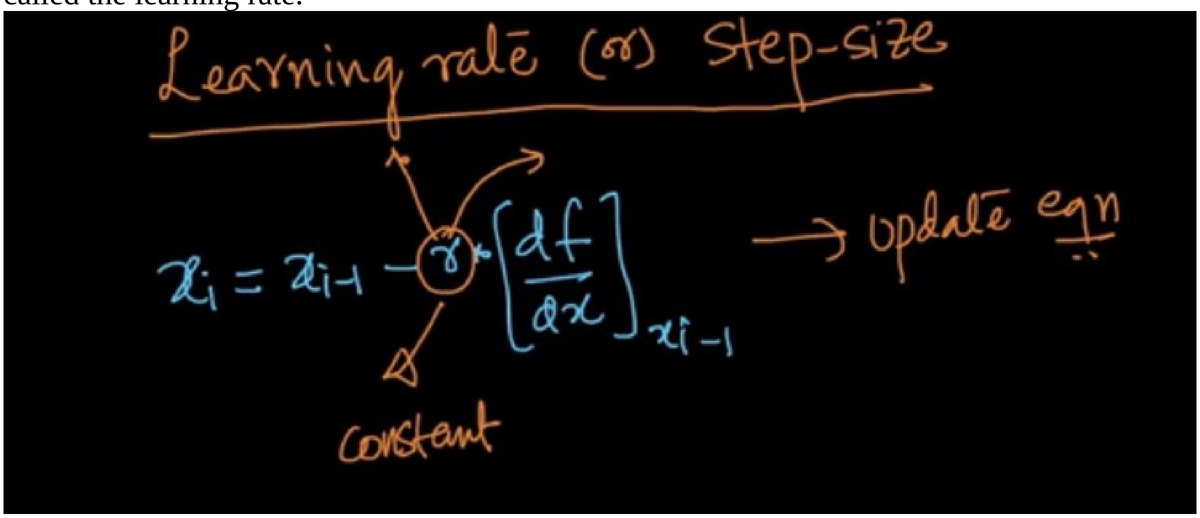


The first step will the big, the next steps will be decreased eventually to attain the minimum of the function, that is achieved when the derivative is relatively small.

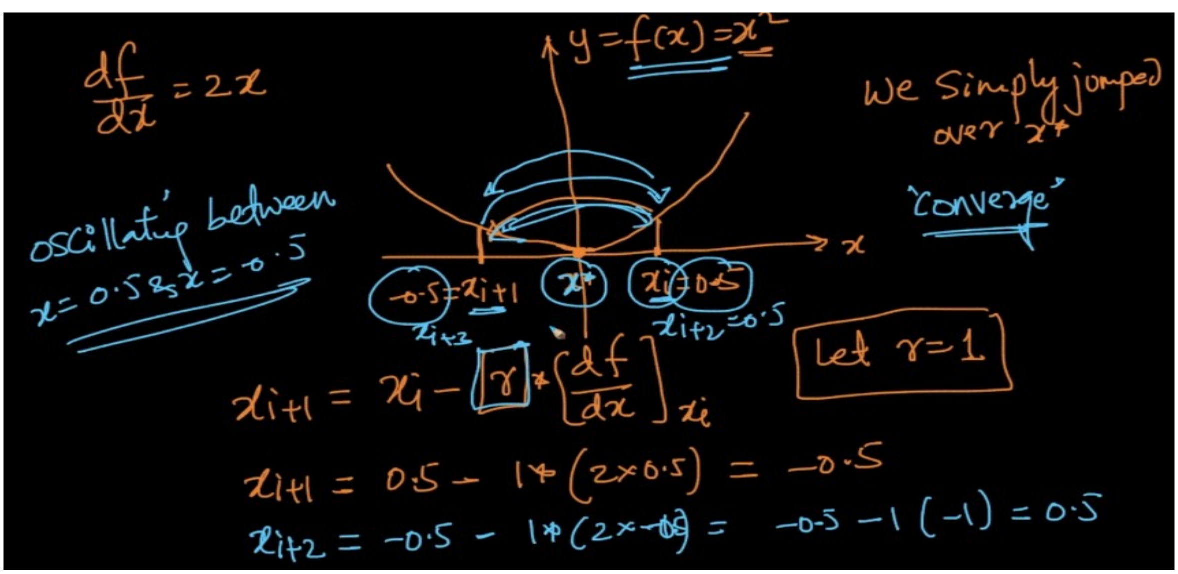


Learning rate (or) step-size:

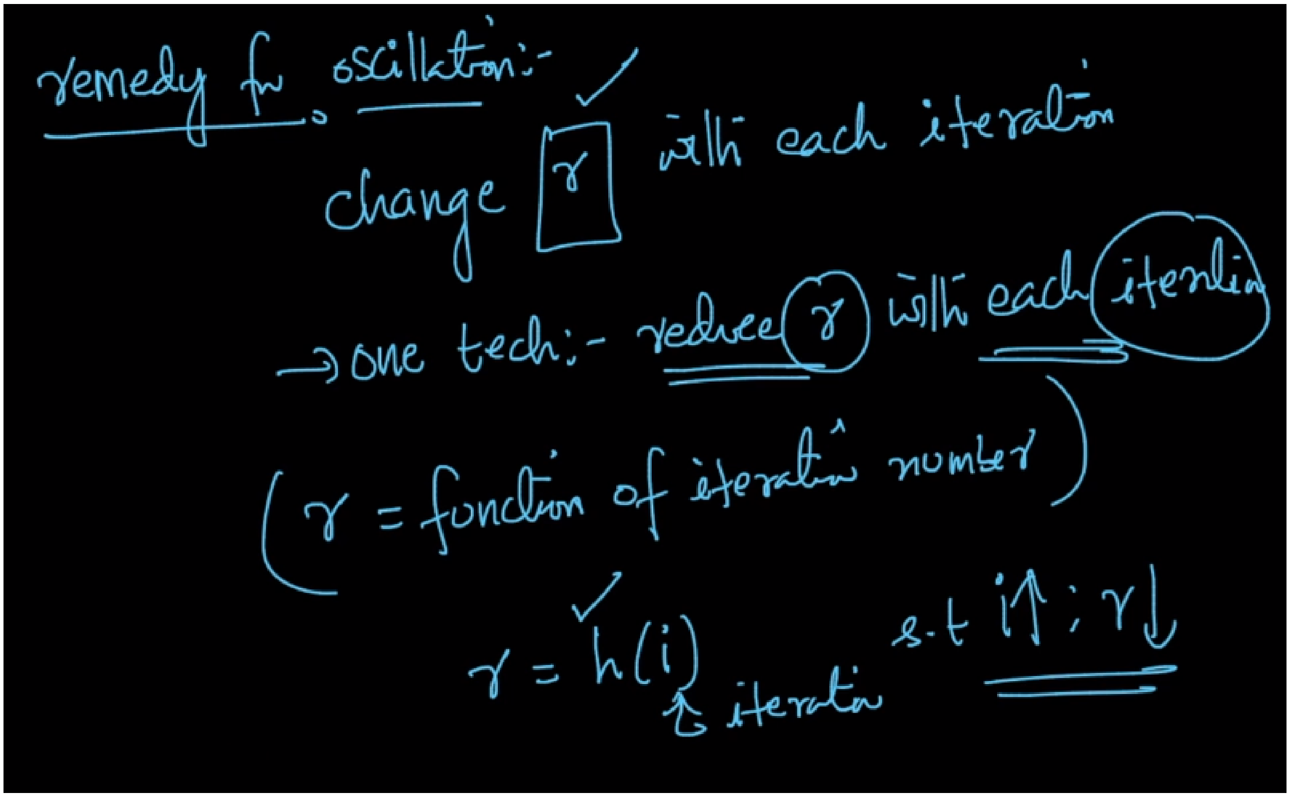
‘r’ is called the learning rate.



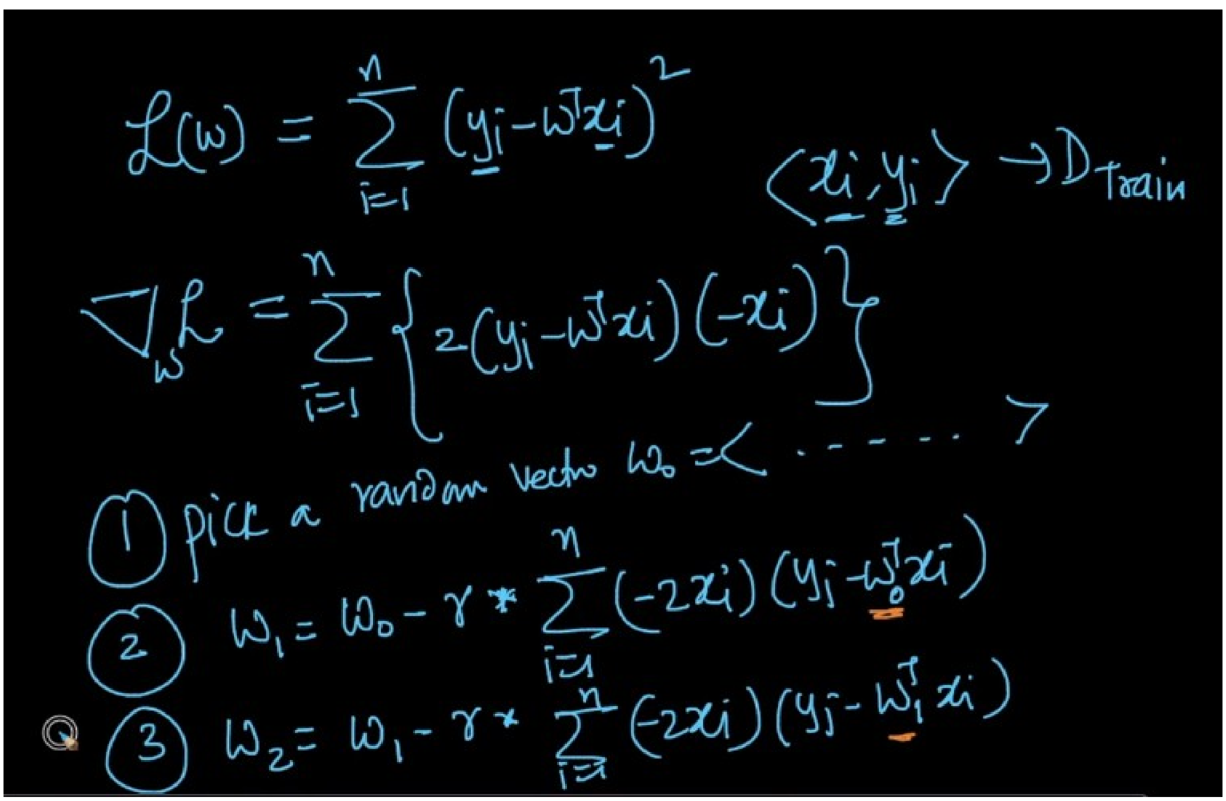
If learning rate is not appropriate then the lowest point in not achieved.



This problem is called oscillation problem, the only solution for this problem is changing ‘r’ with each iteration. We can create a function such that ‘i’ increases ‘r’ should decrease.



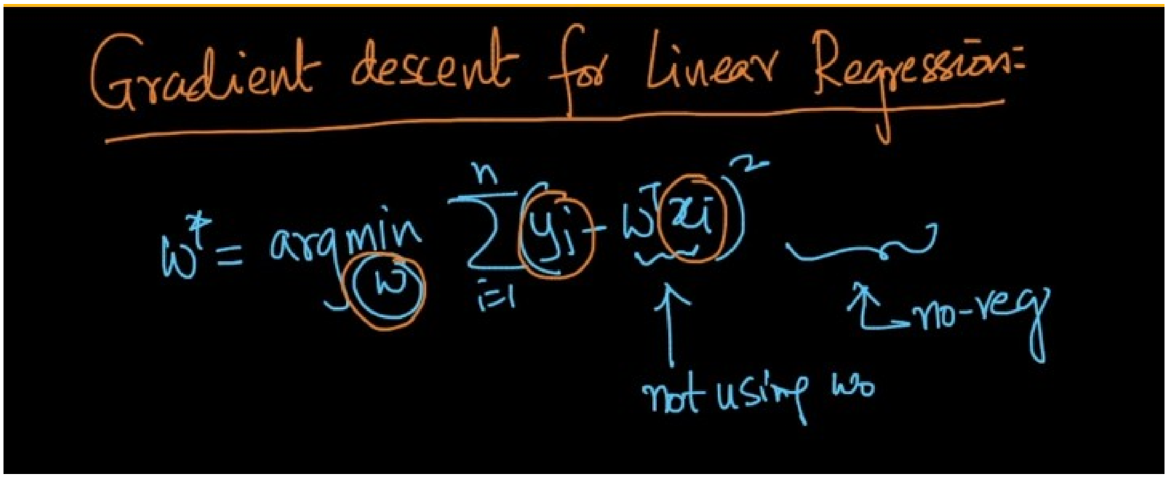
Gradient descent for linear regression:



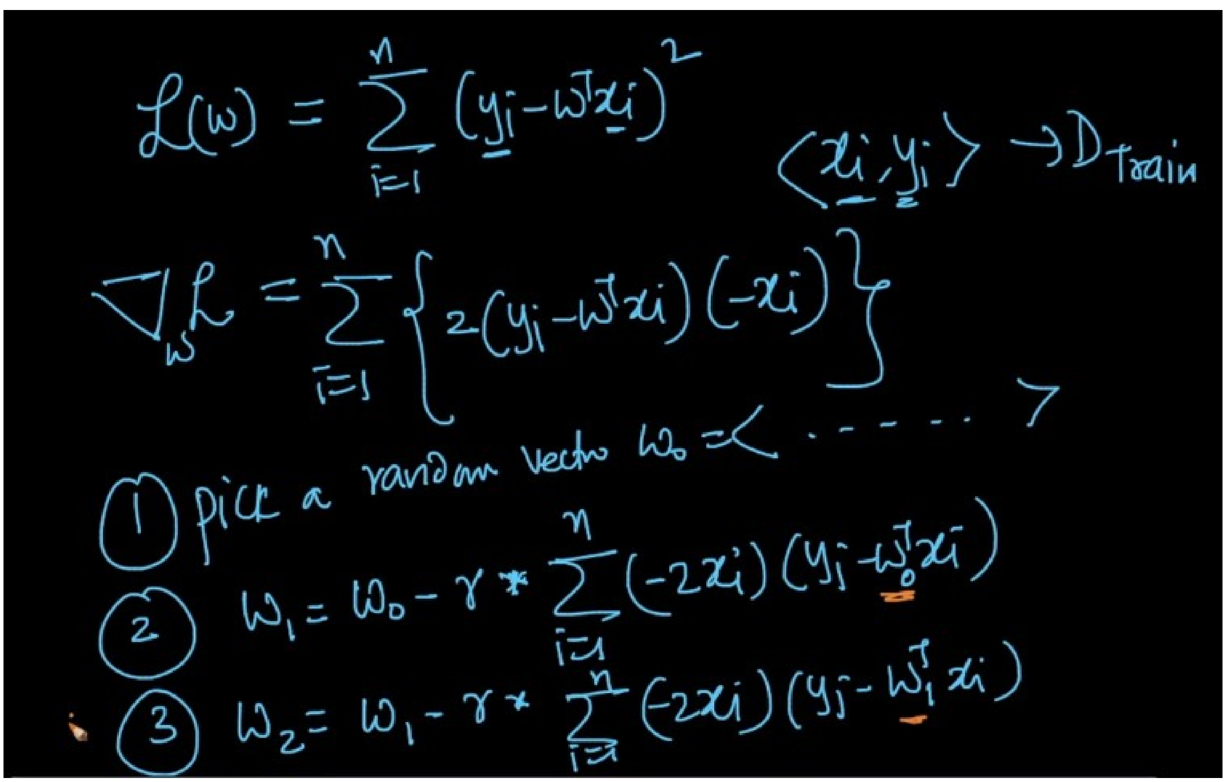
The iterations are made till the difference of the previous step and next step is low as and the ‘w’ is finalized.

If n is large for every step calculations of the total data points must be done, this is computationally expensive.

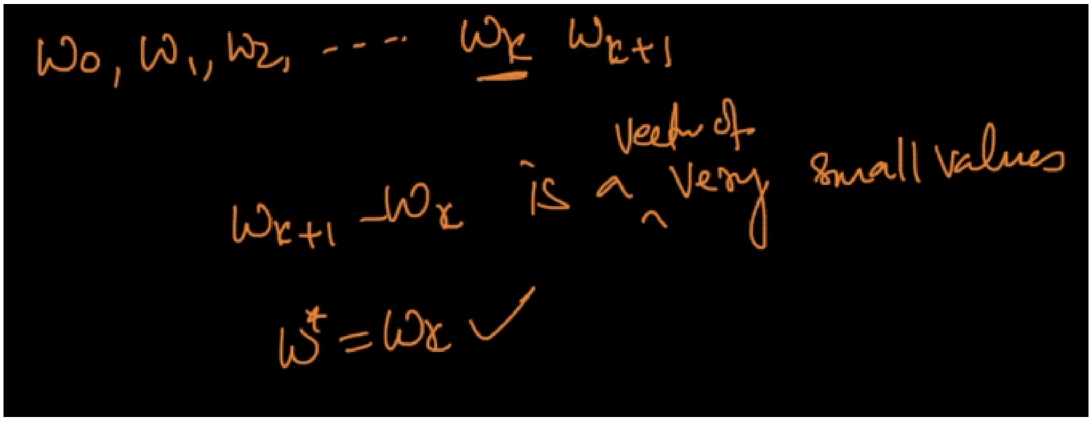
Gradient descent for linear regression:



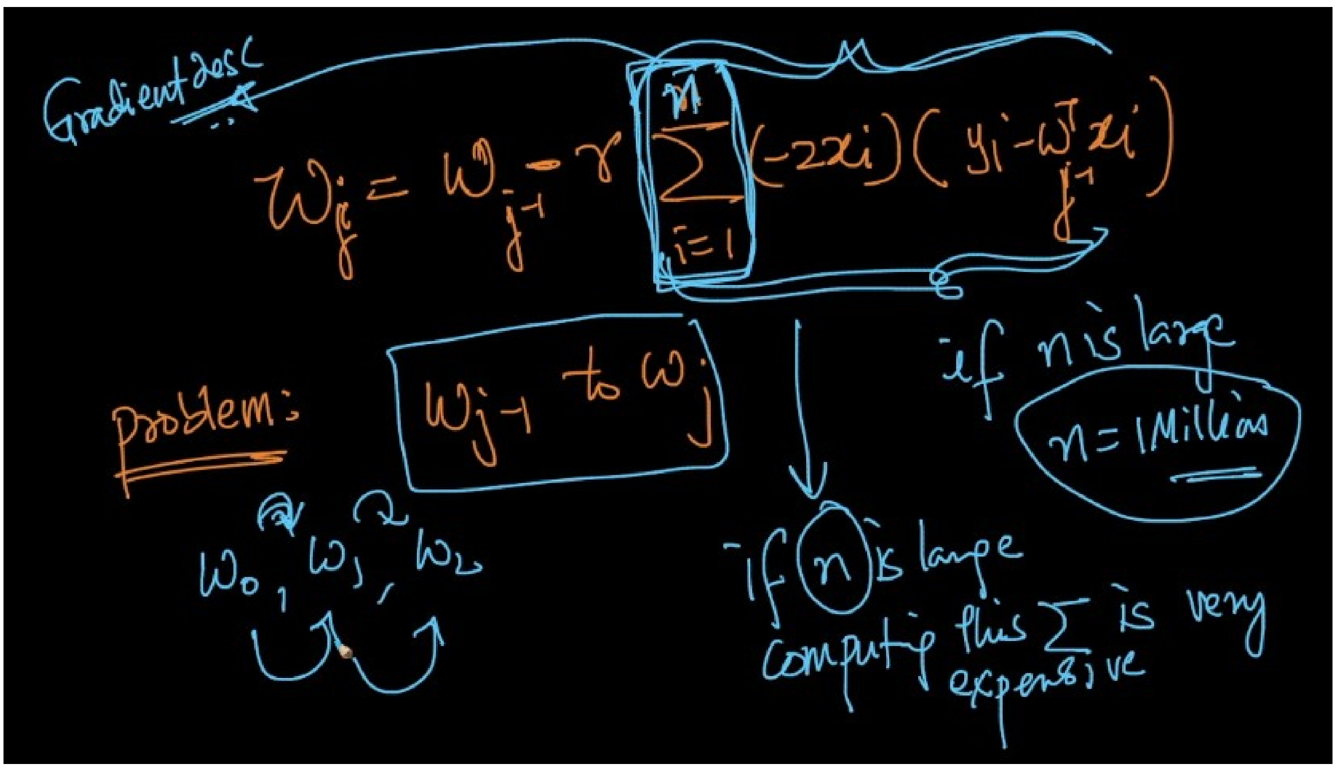
The loss function is written as the function of L.



The calculation of w’s are made till the vector W does not change.



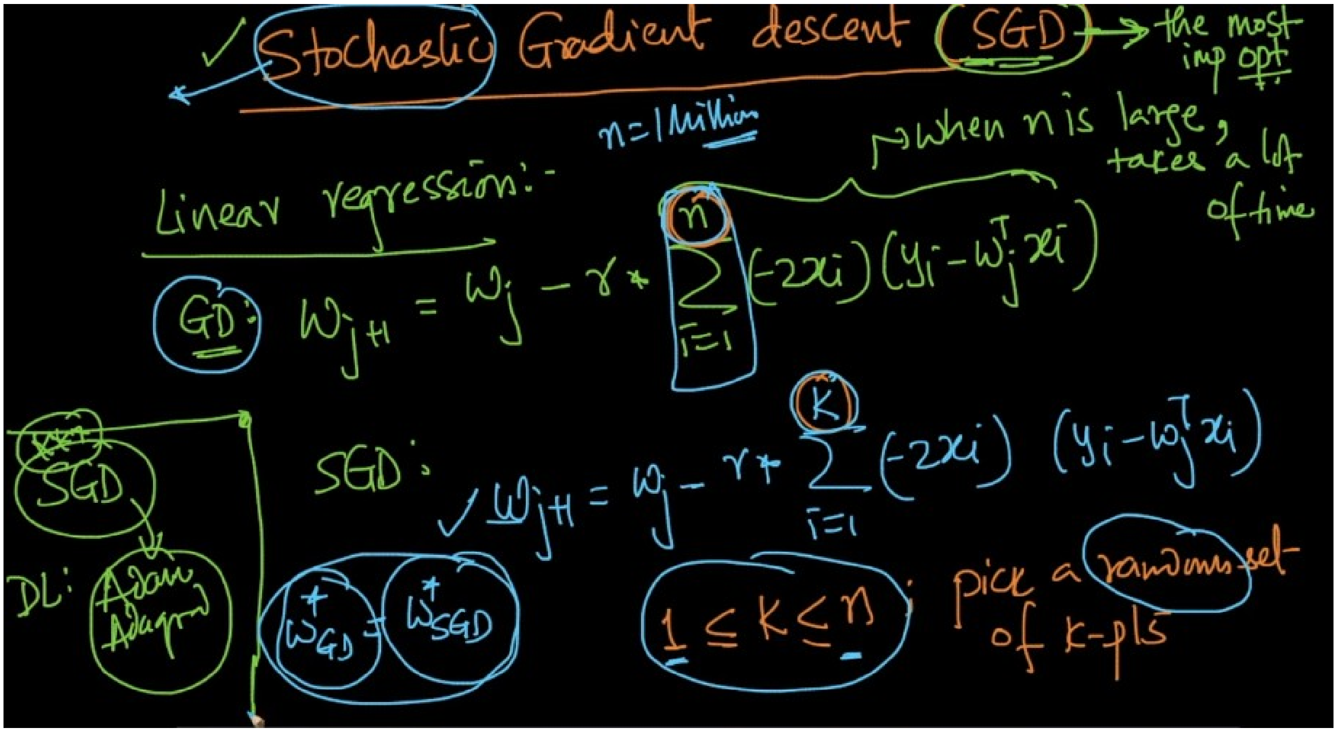
If ‘n’ is large, gradient descent becomes time consuming process.



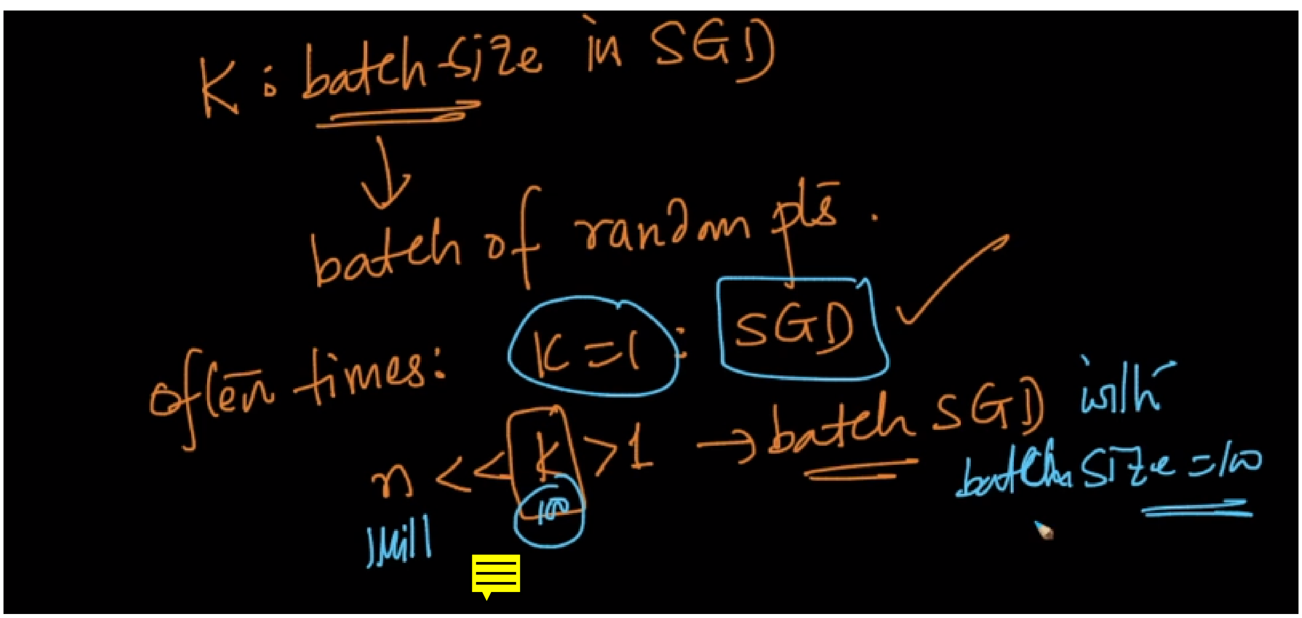
To solve this problem, we use **stochastic gradient descent.**

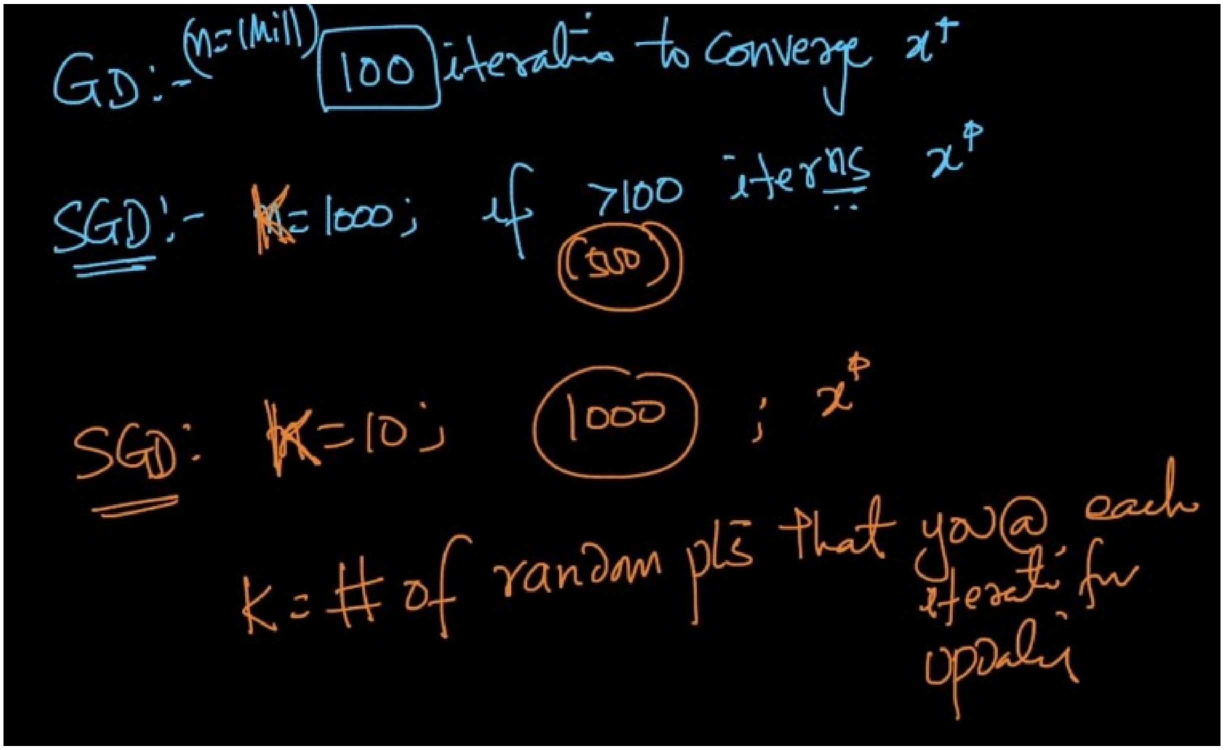
SGD Algorithm:

SGD chooses random set of data points from the whole data set where the number of points are less than whole points, the chosen points can even be one.



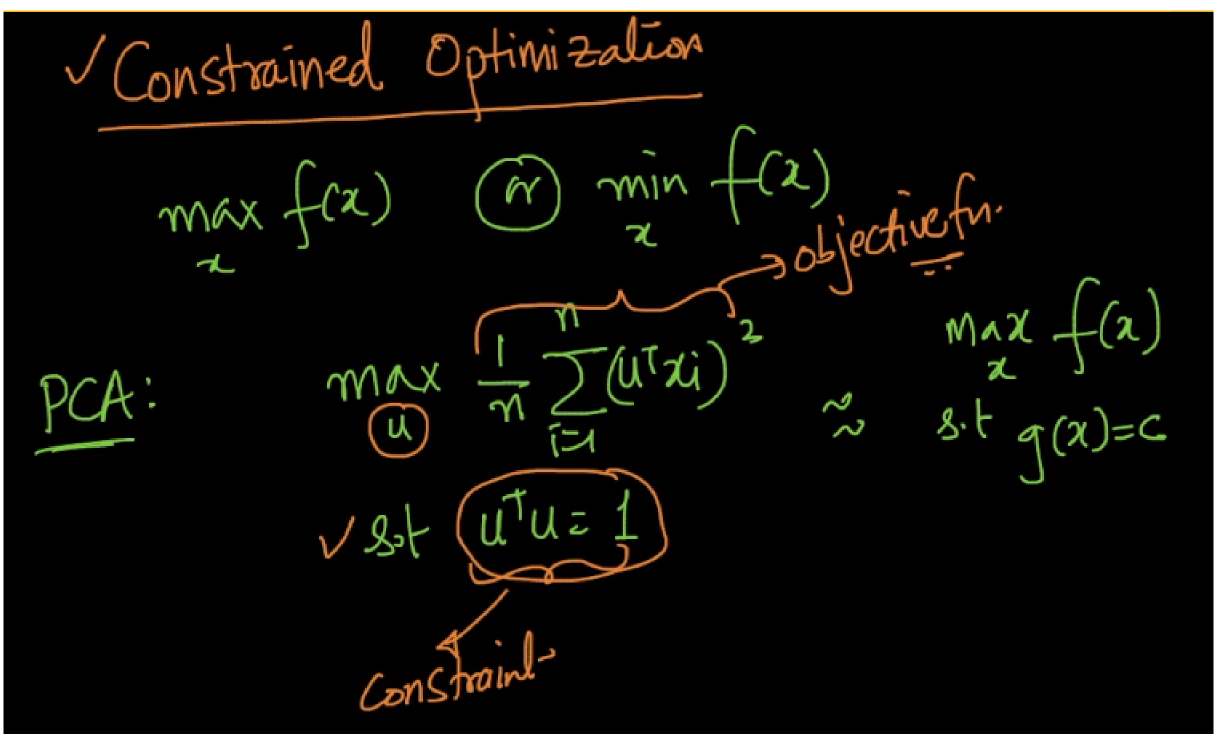
We will use the different set of random points for every iteration, this is also called batch size in SGD. When k in greater that 1 we will batch gradient descent.





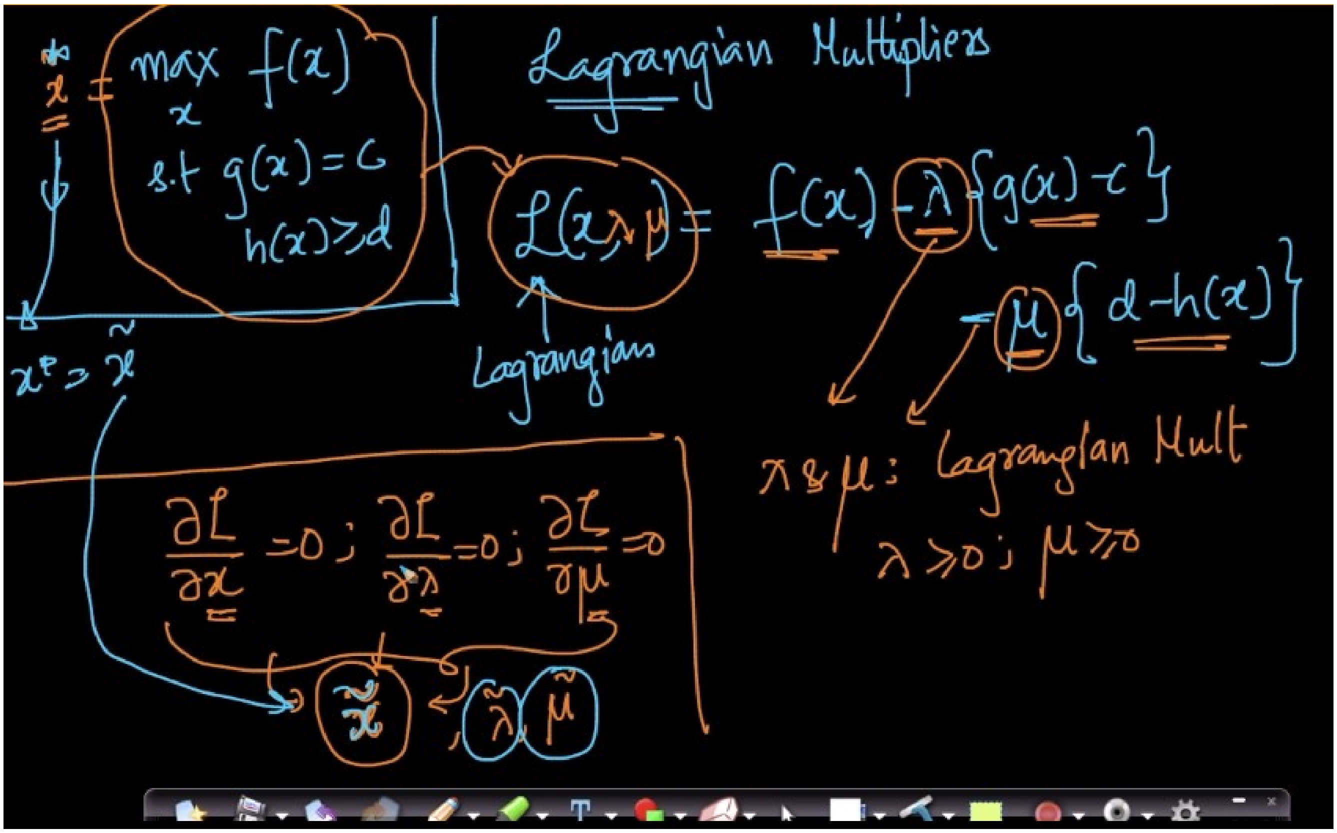
From one iteration to another iteration we will use different set of points.

Constrained optimization & PCA:



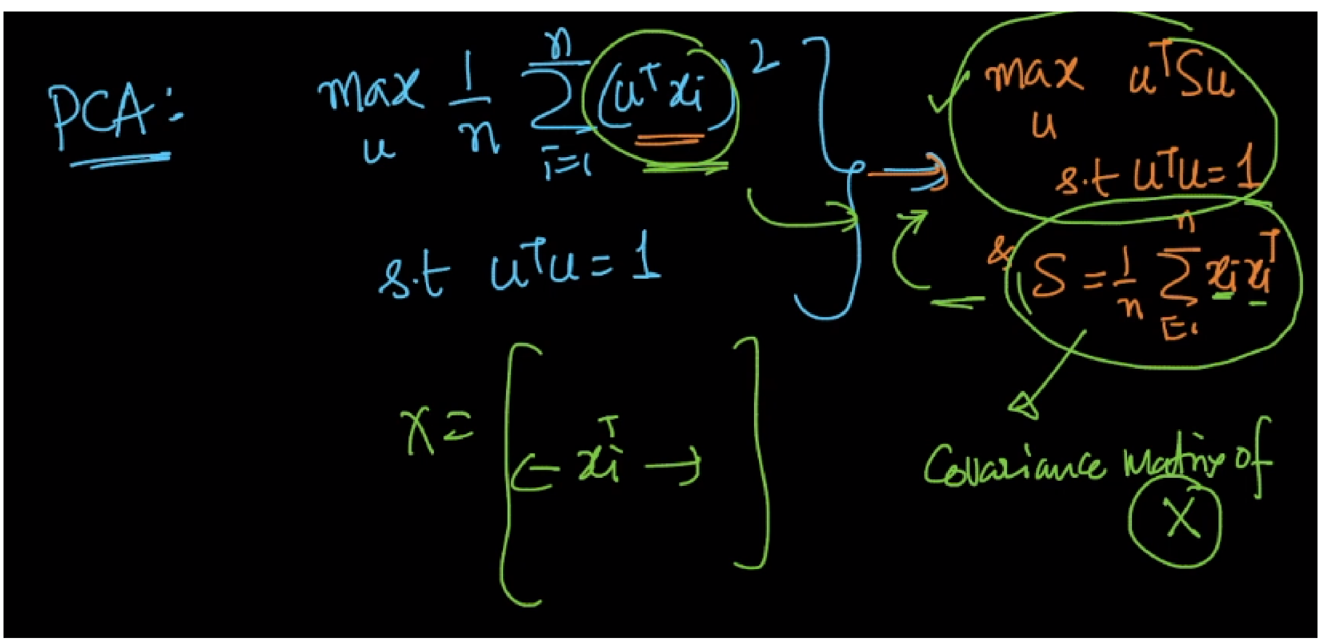
Here the constraint is U is a unit vector.

We will solve the constraint based optimization functions using lagrangean multipliers.



PCA:

Modification of PCA.



PCA :- $\max_{u} \; u^T S u$

s.t $u^T u = 1$

$$S = \frac{1}{n} \sum_{i=1}^{n} x_i x_i^T$$

$\hookrightarrow$ Co-var matrix of X

$$\mathcal{L}(u, \lambda) = u^T S u - \lambda (u^T u - 1)$$

$$\frac{\partial \mathcal{L}}{\partial u} = 0 \Rightarrow \frac{\partial}{\partial u}\left(u^T S u - \lambda u^T u - \lambda\right) = 0$$

$$\Rightarrow \boxed{Su - \lambda u = 0} \Rightarrow \boxed{Su = \lambda u}$$

$Su = \lambda u$ $\longrightarrow$ definition of eigen-val & eigenvec

S: Cov. Matrix

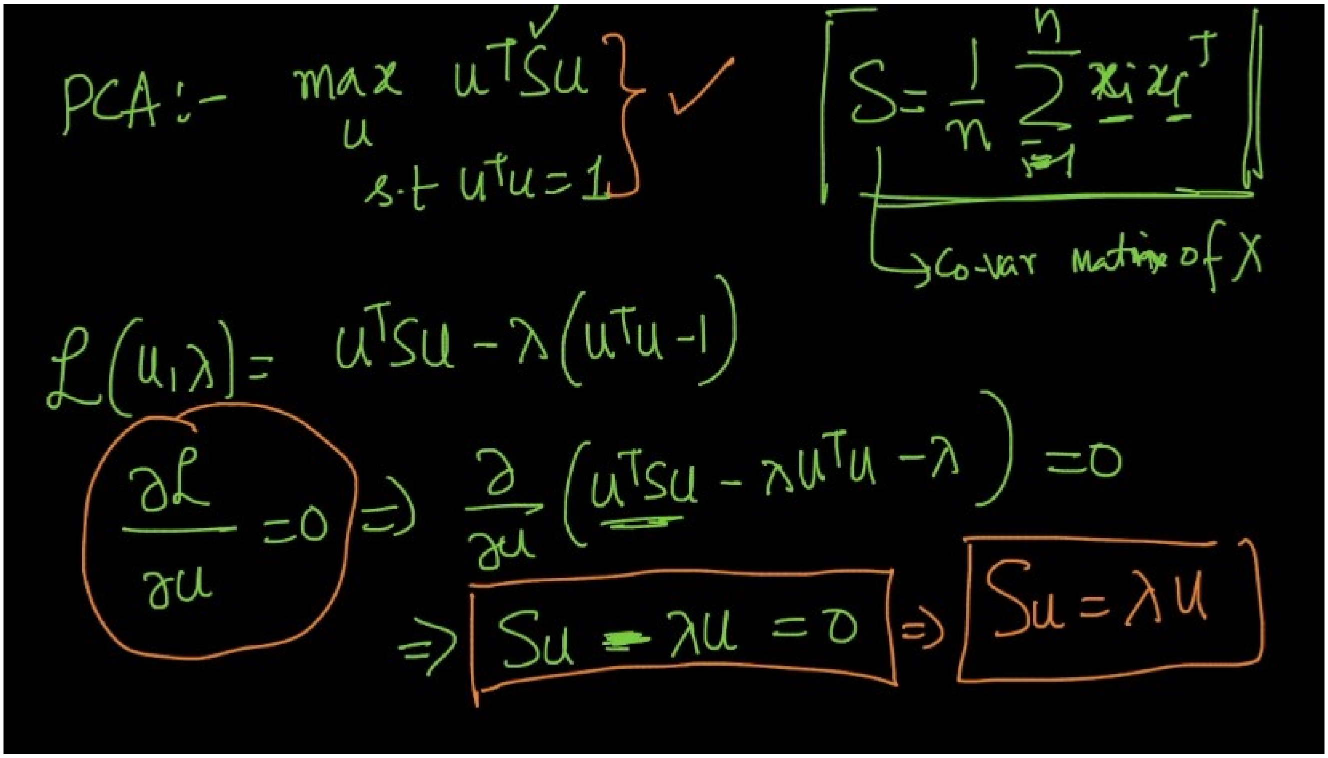
u: vector

$\boxed{A} \, u = \lambda u$

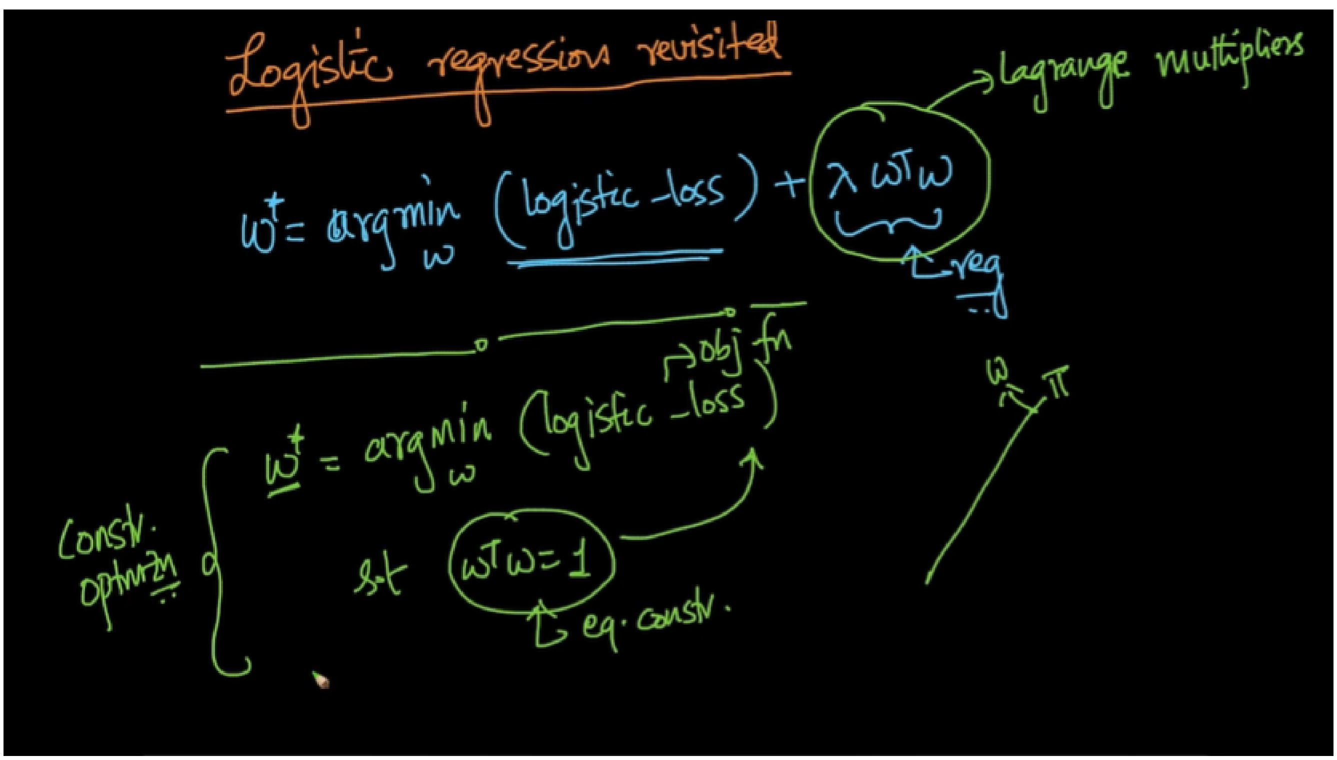
u: eig-vec

λ: eigen-val

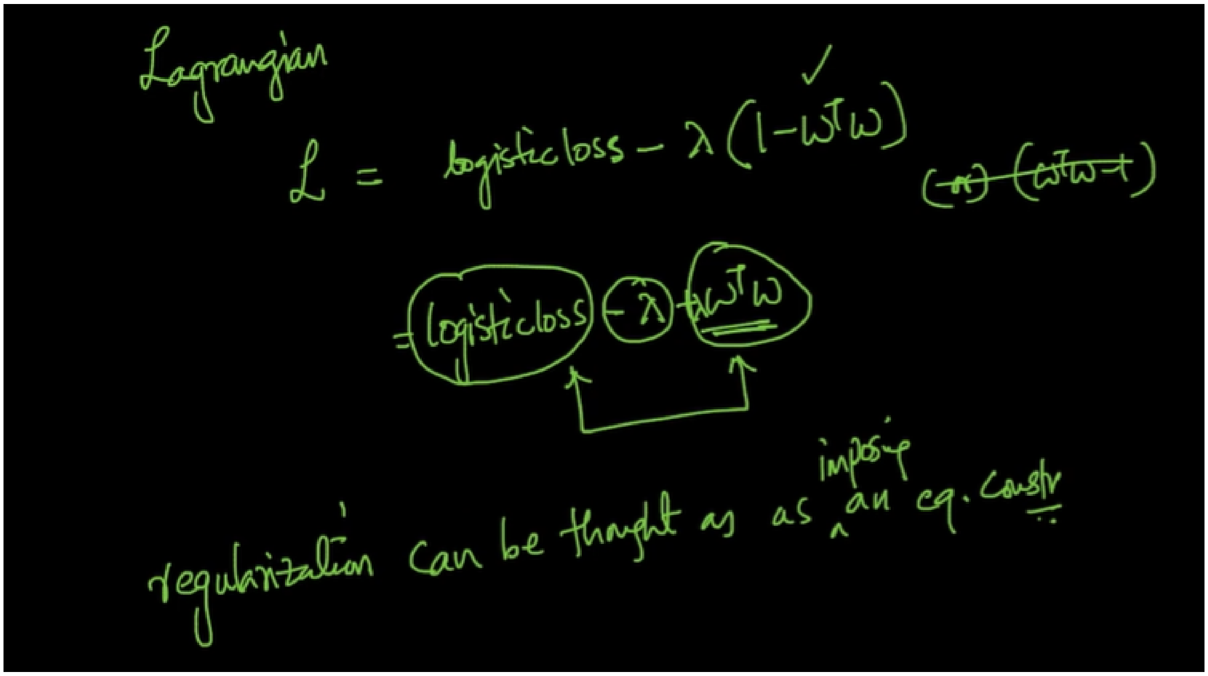
$\left\{ \begin{array}{l} u \text{ is the eigen vec of } S \\ \lambda \text{ is the eigen val of } S \end{array} \right\}$



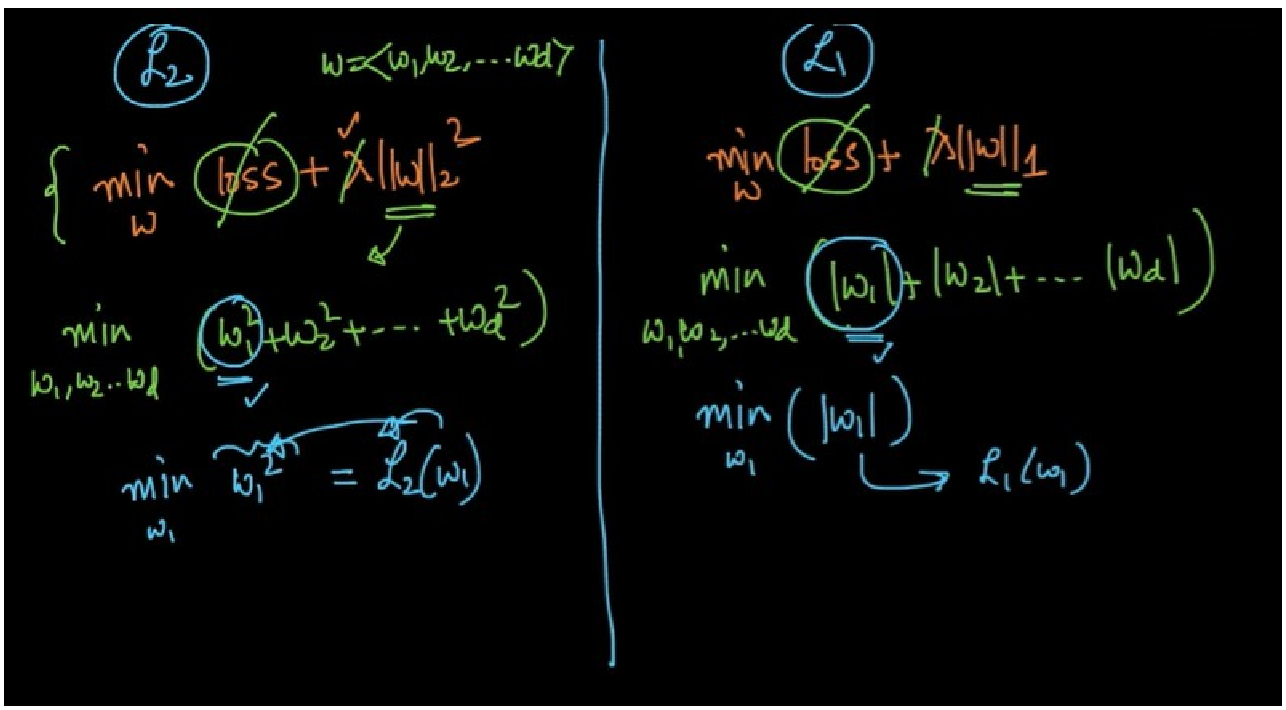
Logistic regression formulation revisited:



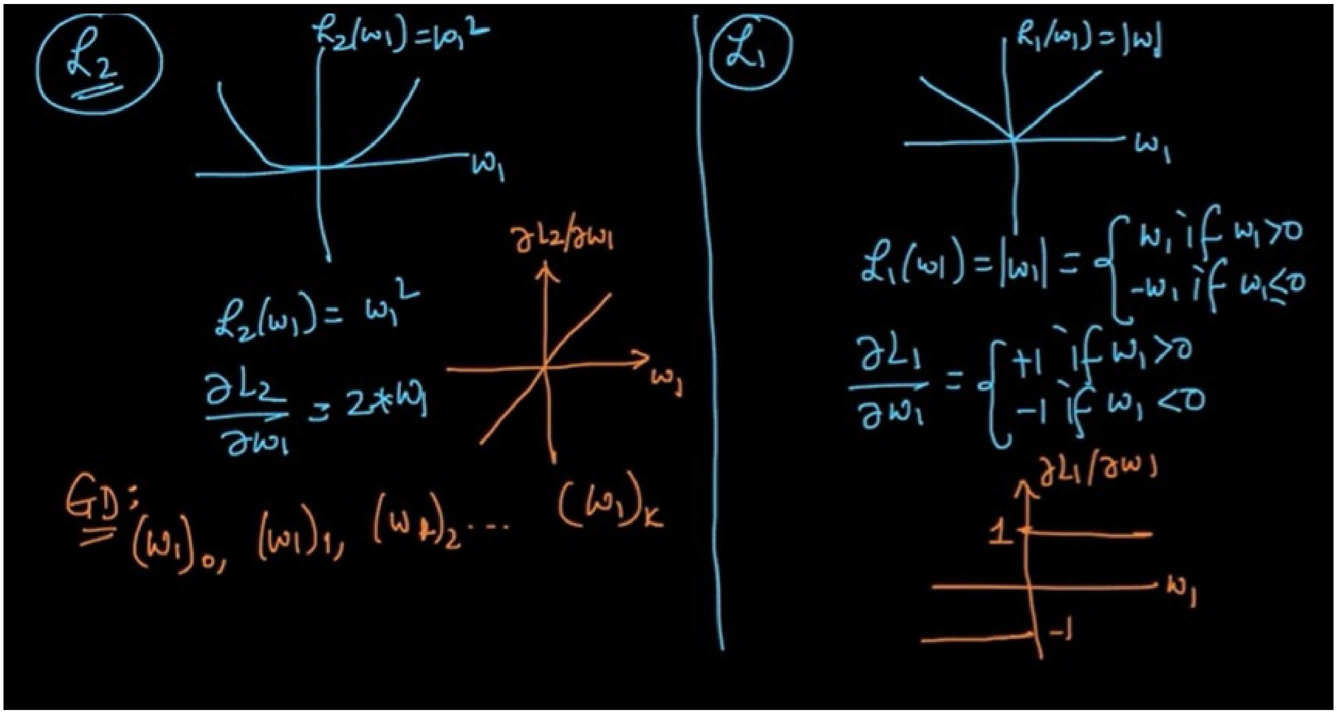
Regularization can be taught as an equality constraint that is being imposed on logistic loss.



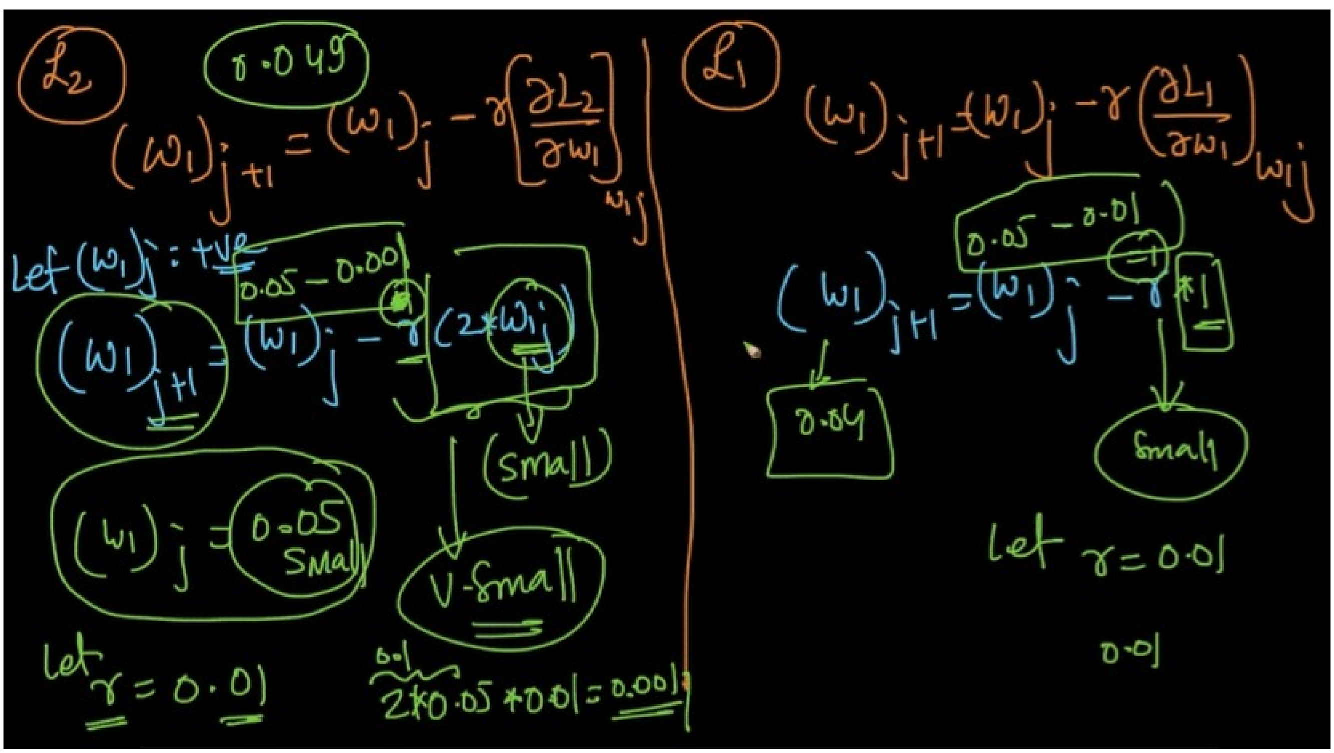
Why L1 regularization creates sparsity?



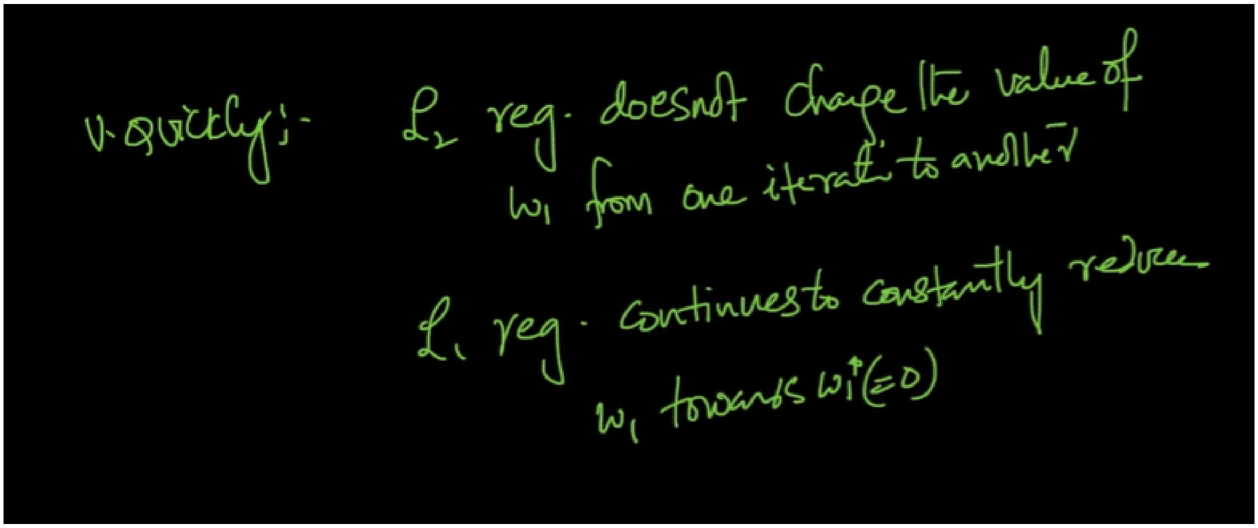
Plotting the L2 and L1 of each value of the weight vector.



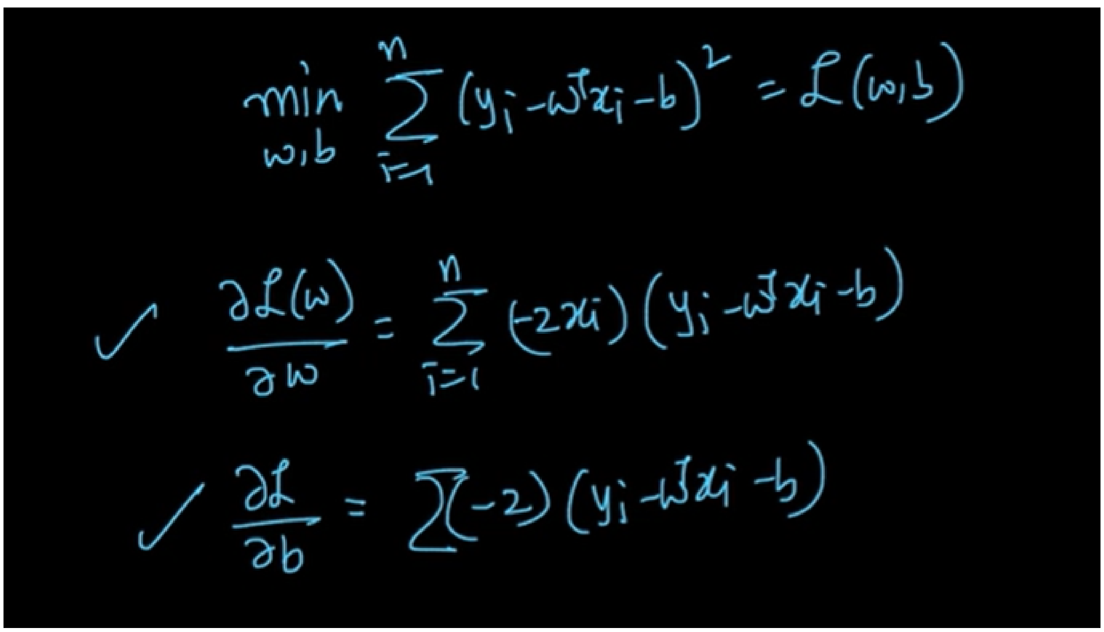
The rate of convergence will decrease in L2.



L2 regularizer doesn’t change the value of w1 from one iteration to another iteration, because the gradient reduce towards w1* = 0.


v. quickly:-
$\mathcal{L}_2$ reg. doesn't change the value of w_1 from one iteration to another
$\mathcal{L}_1$ reg. continues to constantly reduce w_1 towards w_1^* $(=0)$

Exercise:


$\min_{w,b} \sum_{i=1}^{n} (y_i - w^T x_i - b)^2 = \mathcal{L}(w,b)$
✓ $\frac{\partial \mathcal{L}(w)}{\partial w} = \sum_{i=1}^{n} (-2x_i)(y_i - w^T x_i - b)$
✓ $\frac{\partial \mathcal{L}}{\partial b} = \sum (-2)(y_i - w^T x_i - b)$

$$\vec{w}_{j+1} = \vec{w}_j - r\left(\frac{\partial L}{\partial w}\right)_{w_j}$$

$$b_{j+1} = b_j - r\left(\frac{\partial L}{\partial b}\right)_{b_j}$$

reduce r with each itern

$r_0 = 1$

iter 0 → $r = 1$
1 → $r = r/2$
2 → $r = r/2$
⋮

(100 iteratⁿ)

k_iteratⁿ → b^*

w^* $w_k \approx w_{k+1}$ $\quad$ $b_k \approx b_{k+1}$

Ex: Implement SGD for linear regression

Lr. regrsn :- code-sample (boston home price dataset)

↳ LinearRegression() in Sklearn

* Implement SGD for lr. regrsn