Managing Big Data

Regression

Manolis Tzagarakis
Assistant Professor
Department of Economics
University of Patras

tzagara@upatras.gr
2610 969845
google:tzagara
Facebook: tzagara
SkypeID: tzagara
QuakeLive: DeusEx
CoD: CoDFather

Regression analysis

- Investigate and quantify the relationship between variables in a dataset using an existing dataset.
  - In particular, investigate **the effects of one (or more) variable(s) of the dataset onto the value of another variable in the dataset**.
    - How does the value of one variable change if other variable(s) change value?
- Goal: come up with a model (i.e. a function) that **predicts and explains** the value of one variable based on the values of other variables.

Regression analysis

- In regression, the relationship expressed is between one variable -called the **dependent variable**- and one or more **independent variables**
- **Important!** In regression, dependent variable **takes continuous values**
  - Independent variables can be of any type
- Relationship between variables take the **form of a function/equation**: Aims at expressing the value of the dependent variable as a function of the values of other independent variables.
  - Function also referred to as "**regression model**", "**regression equation**" or plain "**regression**".

Regression analysis

- Regression equations can take many different forms
  - But does not
- Examples of regression equations/models
  - **FoodConsumption = 0.78 Income + 1459**
    - e.g. for quantifying the relationship between annual **FoodConsumption (dependent variable)** of families and their annual **income (independent)**
  - **CarValue = PurchaseValue - $e^{(0.88*age)}$ e.g.**
    - E.g for quantifying the relationship between the **present value of a car (dependent variable)** and the variables **purchase value and age (independent variables)**.

Regression analysis

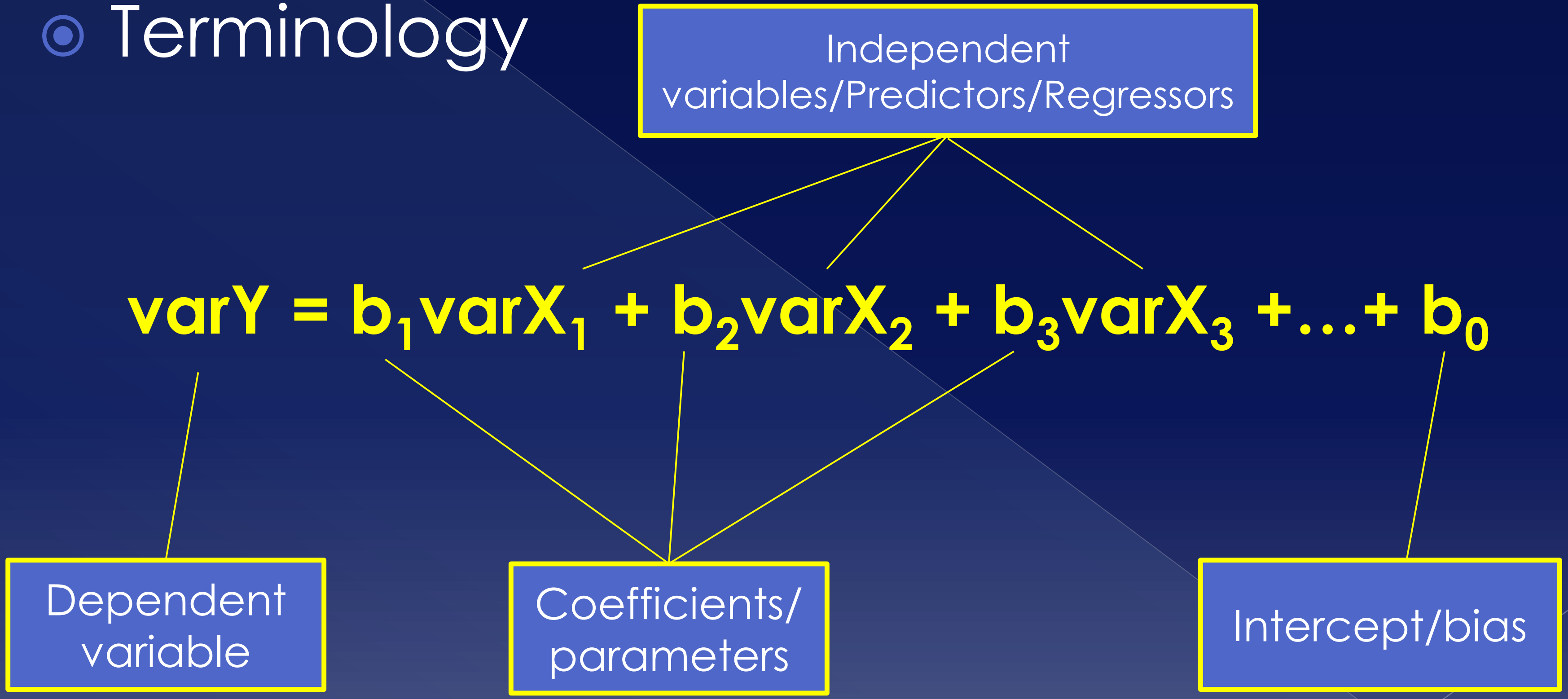
- Purpose of regression models
  - **Explain the variance** in the dependent variable based on the values of the independent variables(s) of the existing dataset
  - **Predict the value** of the dependent variable based on the values of the independent variable(s)

Regression analysis

- Regression analysis **requires a training set with observations on these variables** from which the relationship between the interested variables will be quantified.
- A regression model tries to come up with an **equation that <u>best</u> "fits"** the training set.
  - There can be many regression equations that fit the data, but we require the one that fit the best
  - This "fit" can assessed and the usefulness of the model can be determined.

Regression analysis

- Terminology



$$\text{varY} = b_1\text{varX}_1 + b_2\text{varX}_2 + b_3\text{varX}_3 + \ldots + b_0$$

Regression analysis

- In a regression model, **the unknowns are the coefficients which must be estimated** from the training set
  - Estimation of coefficients is done using the existing training dataset
  - Values of the independent variables are not unknowns-known from the training set

Regression analysis

- Types of regression models
  - Based on **how the value of the dependent variable changes** when the values of the independent variables or coefficients change (That's very, very important and always to keep in mind – determines the form of the regression model)
    - i.e. how a change in the coefficients/parameters and independent variables affect the dependent variable.
    - Expressed as rate of change: $\frac{\Delta Y}{\Delta X}, \frac{\Delta Y}{\Delta b}$
  - Two types
    - Linear regression models
    - Nonlinear regression models

Regression analysis

- Types of regression models
  - Linear regression models
    - In linear regression models the **dependent variable <u>depends linearly</u> on all the coefficients/parameters (and only the coefficients!)**
    - **"Depends linearly"** means that the **rate of change of the dependent variable** -if the coefficient changes- is **independent of the value of the coefficient (i.e. constant wrt coefficient)**.
    - Linear regression models **do not need to depend linearly on the independent variables!**

Regression analysis

- Types of regression models
  - Linear regression models
    - The model **Consumption = b_1Income + b_0** is a linear model because the value of consumption **depends linearly on all coefficients b** of the model: (e.g. assuming coefficient b_1 increases by 1):

$$\frac{\Delta \text{Consumption}}{\Delta b_1} = \frac{(b_1+1)Income + \ b_0 - b_1 Income \ - b_0}{1} = Income$$

Independent of the value of coefficient b_1 that changed. Hence the model is linear. Model happens to be also linear with respect to independent variable Income.

Regression analysis

- Types of regression models
  - Linear regression models
    - Dependent variable does **not need to be linear on the independent variables** (can be but not is not required). This means linear regression models – when plotted- can form curves.
    - This means that all the following regression models are also linear although they are not linear with respect to the variables.

$$\boldsymbol{Consumption = b_1 Income + b_2 FamiltySize^3 + b_0}$$

$$\boldsymbol{BloodPresure = b_1 Sex + b_2 \sqrt{Age} + b_0}$$

$$\boldsymbol{\ln(Income) = b_1 Experience + b_2 Experience^2 + b_3 YearsEducation + b_0}$$

These are considered linear models because you may substitute FamilySize³ with a new variable say Z

Regression analysis

- Types of regression models
  - Nonlinear regression models
    - In nonlinear regression models the **dependent variable does <u>not depend linearly</u> on all the coefficients (and only the coefficients!)**
    - **“Does not depend linearly”** means that the **rate of change of the dependent variable** -if one of the coefficient changes- is **dependent of the value of the coefficient**.

Regression analysis

- Types of regression models
  - Nonlinear regression models
    - The following examples are nonlinear regression models

$$\textit{Rate of reaction} = \frac{b_1 \textit{Concentration}}{b_2 + \textit{Concentration}}$$

Rate of a chemical reaction and the concentration of substance

$$\textit{Crop yield} = \frac{1}{(b_1 + b_2 \textit{CropSpacing})^{b_3}}$$

Regression analysis

- The notion of error in regression models
  - Regression model are **approximations** that **try to fit in the best way possible the real values** of the dependent variable in the training set.
    - Because regression models approximate the value of the dependent variable, they **never succeed in predicting the real value** of the dependent variable.
      - But what is the real value of the dependent variable?

Regression analysis

- The notion of error in regression models
  - Two types of errors in regression models
    - Errors/disturbance
      - **The difference between the (unobserved) real value of the dependent variable in the population and the observed value in the training set.** This error can never be observed or measured because we are unaware of the real value of the dependent variable in the population.
    - Residuals/fitting deviations
      - **The difference between the dependent value in the training set and the predicted/estimated value by the regression model**. This can be observed and measured

Regression analysis

- The notion of error in regression models
  - Errors and residuals are included in the regression models.
    - Adding term ε (for error) when showing the general model or ε_i (for residuals) when
  - Full specification of a regression model includes error term e.g.

$$\boldsymbol{Consumption = b_1 Income + b_2 FamiltySize^3 + b_0 + \varepsilon}$$

  - If the error term is not explicitly included in the regression model, it's implied.
    - This means, there always is an error term!

Regression analysis

- More types of regression models
  - **Simple** regression models
    - When the regression model includes only 2 variables: one dependent and one independent variable
      - E.g. **Income = b_1Education + b_0**
  - **Multiple** regression models
    - When the regression model includes more than 2 variables
      - E.g. **Income = b_1Education + b_2Experience + b_0**

Regression analysis

- Who comes up with regression models and how?
  - Domain experts (economist, statisticians, engineers, etc)
    - Theory
      - **Read the relevant literature and identify factors** that affect the value of the dependent variable
    - Look at the data and how it changes
      - From existing data, see **how** the dependent variable changes when the independent variables change
    - Trial and error
      - Begin by **trying simple regression models and assess the results.** Continue by modifying the model if results are not appropriate.

Regression analysis

- Who comes up with regression models?
  - **Don’t forget:** Regression models are approximations that try to fit the best way possible the data in the training set.

Estimating parameters in regression models

Estimating parameters

- In a regression model, the problem is **estimating the coefficients/parameters** that will **indicate the relationships** between the variables
  - Coefficients/parameters are estimated from an existing dataset (training set) which is required.

Training set.

Tid	House Price	Marital Status	Income	m^2House
1	190K	Single	125K	180
2	145K	Married	100K	154
3	101K	Single	70K	110
4	187K	Married	120K	167
5	109K	Divorced	95K	110
6	96K	Married	60K	90
7	200K	Divorced	220K	190

$$\text{Income} = b_1 m^2\text{House} + b_0$$

Unknowns are the parameters b (independent variables known from training set). The parameters b of this regression model are estimated using the training set. The goal: find the best values of b which best fit the values of the dependent variable in the training set.

Estimating parameters

- **Different methods** to estimate parameters based on the type of the regression model
  - Linear vs Nonlinear
- The general idea: Estimation of parameters in regression model (linear or nonlinear) **involves a Cost function (also called "Loss function")** that needs to be **minimized**.

Estimating parameters

- **Cost function** tries to measure **how big the error of the regression model is** when estimating the value of the dependent variable
  - Essentially, this is the sum of residuals which is to be minimized
  - Cost functions can have many different forms
    - Depending on the purpose
    - The **form of the cost function** determines the **type of regression**: Ordinary Least Squares (OLS), LASSO, Quantile etc

Estimating parameters in linear regression models

Estimating parameters

- **Linear regression models** have the following general form:

$$Y = b_1X_1 + b_2X_2 + b_3X_3 + \cdots b_kX_k + b_0 + \varepsilon$$

Where:
Y: Dependent variable
X_i : Independent variable i
b_i: Parameter to be estimated
ε: Error term

Estimating parameters

- Since linear regression models try to **fit the available training data**, the linear regression model can also be written in the form:

$$\boldsymbol{Y_i = b_1X_{1i} + b_2X_{2i} + b_3X_{3i} + \cdots + b_kX_{ki} + b_0 + e_i}$$

Where:
Y_i : Value of dependent variable in observation i in training set
X_{ki} : Value of independent variable k in observation i of the training set
b_i: Parameter to be estimated
e_i : Residual of the i-th observation in the training set

Estimating parameters

- If there are n observations in the training set, then there will be n equations, one for each observation, of the form:

$$\boldsymbol{Y_i = b_1X_{1i} + b_2X_{2i} + b_3X_{3i} + \cdots + b_kX_{ki} + b_0 + e_i}$$

Estimating parameters

- Because **parameters are estimated from the training set** and not the **truly real values of the variables (remember: the training set is just a sample)**, the estimates are mentioned in the regression model

$$\hat{Y}_i = \hat{b}_1 X_{1i} + \hat{b}_2 X_{2i} + \hat{b}_3 X_{3i} + \cdots + \hat{b}_k X_{ki} + \hat{b}_0 + e_i$$

Where:
$\hat{Y}_i$: The **estimated** value of the dependent variable
$\hat{b}_i$: The **estimated** value of the parameter i.

Estimating parameters

- Regression model in matrix notation
  - It’s customary to represent these n regression equation in matrix notation. If we define:

Matrix of values of independent variables in training set.

$$\hat{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \dots \\ Y_n \end{bmatrix} \quad X = \begin{bmatrix} 1 & X_{11} & X_{21} & \dots & X_{k1} \\ 1 & X_{12} & X_{22} & \dots & X_{k2} \\ 1 & X_{13} & X_{23} & \dots & X_{k3} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & X_{1n} & X_{2n} & \dots & X_{kn} \end{bmatrix} \quad \hat{b} = \begin{bmatrix} \hat{b}_0 \\ \hat{b}_1 \\ \hat{b}_2 \\ \dots \\ \hat{b}_k \end{bmatrix} \quad e = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ \dots \\ e_n \end{bmatrix}$$

  - Then the n linear regression equations, derived from the training set, can be written in matrix form:

$$\hat{Y} = X\hat{b} + e$$

If you carry out the operations, you’ll get tne n linear regression equations as vectors.
The matrix form makes it easier to calculate the parameters.

Estimating parameters

- Two methods for estimating the parameters of linear regression models
  - **Ordinary Least Squares (OLS)**
  - **Gradient Descent and its variations**
- Each of the above method appropriate in specific situations.

OLS

Estimating parameters

- Ordinary Least Square (OLS) Regression
  - In OLS the cost function is the **Sum of Squared Errors (SSE) i.e. sum of residuals** which must be minimized:

$$SSE = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2$$

Y_i = Value of the dependent variable in observation i of the training set
$\hat{Y}_i$ = Estimated value by the linear regression model for the values of the independent variables in observation i in the training set.

  - The **parameters b which minimize** the above SSE **are the parameter estimates of the linear regression model that best fit the training data.**

Estimating parameters

- Ordinary Least Square (OLS) Regression
  - How is the Cost function (SSE) minimized in OLS?
  - First write **SSE in matrix notation** as a function of the **vector b**

$$SSE(\hat{b}) = e^T e = (Y - X\hat{b})^T (Y - X\hat{b})$$

  - And then minimize the Cost function (SSE) by solving the equation of partial derivatives:

$$\frac{\partial SSE(\hat{b})}{\partial \hat{b}} = 0$$

This equation **has a closed form solution** due to the form of the linear regression. Solving this will calculate the vector b that minimizes the SSE and hence finds the parameters we are looking for.

Estimating parameters

- Ordinary Least Square (OLS) Regression
  - The **closed form solution** derived from the previous equation for estimating the parameters b of a linear regression model in matrix form is:

$$\hat{b} = \left(X^T X\right)^{-1} X^T Y$$

  - The above **closed form formula – called normal equation - gives you the vector of parameters estimates b**, based on the matrix of values of the independent variables X and the matrix of the values of the dependent variable Y in the training set, **which minimize SSE** and hence what we were looking for.

Gradient descent

Estimating parameters

- Gradient descent
  - While OLS minimizes the SSE in a very specific way (by finding the values of b who yield the partial derivative to zero) leading to a **closed form formula (the normal equation)** for estimating the parameters, Gradient descent **minimizes the cost function in a very different way.**
  - **Gradient descent is an <u>iterative, numerical optimization method</u> for minimizing the cost function and thus finding the parameter estimates**.
    - i.e. Gadient descent does not offer a closed form formula like the normal equation in OLS for calculating the parameters.
    - "iterative" ? Tries to guess the proper values of the parameters that lead to minimizing the cost function

Estimating parameters

- Gradient descent
  - Why Gradient descent?
    - OLS has **one big concern**: The normal equation requires inversion of a matrix:

$$\hat{b} = \underbrace{\left(X^T X\right)}_{\text{Matrix inversion}}{}^{-1} X^T Y$$

    - **Matrix inversion <u>is a very expensive operation</u>**. If the **$X^T X$ matrix has 100 variables** (is an 100x100 matrix), since an inversion requires n^3 operations (n=dimension of matrix) on average, it would require **~1000000 operations** to invert the matrix.
    - **OLS not suitable for big data!**

Estimating parameters

- Gradient descent
  - Why Gradient descent?
    - **Gradient descent performs much better** –in terms of execution times/number of operations- **in big data contexts** than OLS and in such situations it's exclusively used.
      - There are even versions that increase the performance of the algorithm
  - **Warning!** Gradient descent uses a different notation for the multiple linear regression model:

$$h_\theta(x^{(i)}) = \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_k x_k^{(i)} + \theta_0 + \varepsilon$$

θ_j = Parameter j (to be estimated)

$x_j^{(i)}$ = Value of independent variable j in observation i in training set

Estimating parameters

- Cost function in Gradient descent
  - In Gradient descent the **cost function is called the mean squared error, J(θ)**

$$J(\theta_0, \theta_1, \ldots, \theta_\kappa) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Where:
θ_i = (Unknown) parameter i of the linear regression model from a total of k+1 parameters
m = Number of observation in training set
$h_\theta()$ = The estimated value of the linear regression model for the values of the independent variables at observation i in training set.
$x^{(i)}$ = The values of the independent variables of observation i in training set
$y^{(i)}$ = The value of the dependent variable of observation i in training set

Estimating parameters

- Cost function in Gradient descent
  - Gradient descent **attempts to minimize the cost function J(θ)** by finding/estimating the proper values of parameters θ.

$$\underbrace{J(\theta_0, \theta_1, \dots, \theta_\kappa)}_{\text{Often abbreviated simply as } J(\theta).} = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Often abbreviated simply as J(θ).

Estimating parameters

- Cost function in Gradient descent
  - Cost function has things in common with the cost function (i.e. SSE) in OLS.

$$J(\theta_0, \theta_1, \dots, \theta_\kappa) = \frac{1}{2m} \underbrace{\sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2}_{\text{Sum of squared errors in OLS}}$$

Why divide by 2m? m because of two reasons: i) it's the **mean** squared error and ii) it yields to **smaller numbers** which is important **due to the numerical nature of the method.** Also, include the constant 2 in denominator to make things simpler as it's shown later on (hint: it will be eliminated). However, these terms do not affect the minimization process.

Form of the linear regression model, with θ the unknown parameters, is:

$$h_\theta\left(x^{(i)}\right) = \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \dots + \theta_k x_k^{(i)} + \theta_0 + \varepsilon$$

Estimating parameters

- Cost function in Gradient descent
  - Cost function in matrix form

Note: square each element of vector

$$J(\theta) = \frac{1}{2m}(X\theta - y)^T(X\theta - y) = \frac{1}{2m}(X\theta - y)^2$$

Sum of all elements to get pure number of J(Θ).

Where:

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} \quad X = \begin{bmatrix} 1 & X_{11} & X_{21} & \dots & X_{k1} \\ 1 & X_{12} & X_{22} & \dots & X_{k2} \\ 1 & X_{13} & X_{23} & \dots & X_{k3} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & X_{1n} & X_{2n} & \dots & X_{kn} \end{bmatrix} \quad \theta = \begin{bmatrix} \hat{\theta}_0 \\ \hat{\theta}_1 \\ \hat{\theta}_2 \\ \dots \\ \hat{\theta}_k \end{bmatrix}$$

Vector of dependent variables

Matrix of independent variables with first columns all 1s

Vector of estimated parameters

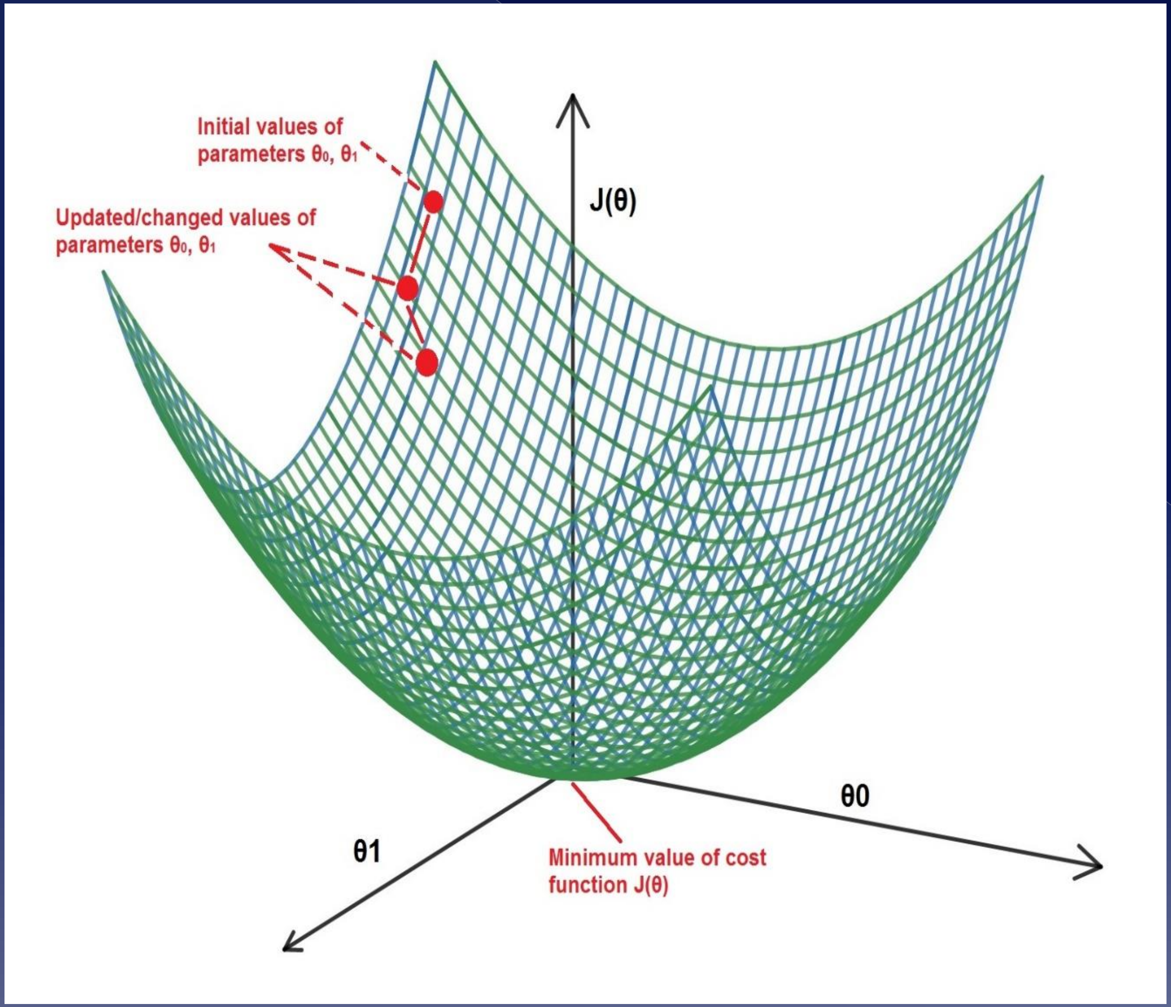
Estimating parameters

- Cost function
  - A note on **notation**: cost function in **gradient descent uses different notation** (**θ** instead of **b** for parameters, **h_θ()** for linear regression model, **J(θ)** for cost function)
  - This is because Gradient descent originated from a different field. One of the **first algorithms which founded the area of machine learning** in applied mathematics
  - We use the same notation used by contemporary literature.

Estimating parameters

- General idea of estimating the parameters θ with Gradient descent which minimize the cost function J(θ)
  - Start with **initial, random values for the parameters θ**
  - **Update/Change the values of the parameters θ** in a way that **yield to smaller value of the cost function J(θ)**
  - Continue changing values of θ **iteratively until the smallest value of J(θ)** is attained.

Estimating parameters


Initial values of parameters θ0, θ1
Updated/changed values of parameters θ0, θ1
J(θ)
θ0
θ1
Minimum value of cost function J(θ)

The general idea of Gradient descent.

Assume a simple linear regression model $h_\theta(x) = \theta_0 + \theta_1 x$

The cost function of such linear regression model, J(θ), will be convex and an example cost function is depicted on the left.

Gradient descent tries to modify the values of all the parameters θ iteratively, towards the smallest value of J(θ).

Estimating parameters

- How to change the values of parameters θ in order to minimize J(θ)?
  - The **value of the first partial derivative** of the cost function J(θ) with respect to a parameter θ_j will tell us **how we need to modify the parameter θ_j** (leaving all other parameters constant) to achieve a smaller value of J(θ).
- Remember from your math:
  - If the **value of the first derivative of a function f(x)** with respect to x is at some point x_0
    - **positive (>0)**, an increase of x_0 leads to an increase of f(x). A decrease of x_0 leads to decrease of f(x)
    - **negative (<0)**, an increase of x_0 leads to an decrease of f(x). A decrease of x_0 leads to increase of f(x)
    - **is equal to zero (=0)**, an increase of x_0 leads to an increase or decrease of f(x) (has a point of deflection at x_0).

Estimating parameters

- How to change the values of parameters θ in order to minimize J(θ)?
  - In essence, the value of the first derivative with respect to some x tells us if x needs to increase or decrease in order to achieve a smaller value of the function f(x).
    - Value of the first derivative **tells us the direction of change of variable x (increase or decrease).**

Estimating parameters

- How to change the values of parameters θ in order to minimize J(θ)?
  - **J(θ) is a multivariate function**, where the parameters θ_j are the unknown variables.
  - To apply the derivative technique, we will use the **first partial derivative wrt to one θ_j** parameter and leaving other θs constant i.e. calculate the value of $\frac{\partial J(\theta)}{\partial \theta_j}$. If this value is positive, a decrease of θ_j will decrease the cost function, if it's negative, an increase of θ_j will decrease the cost function J(θ).
    - **Do this for all parameters θ** to see how they need to change **i.e. calculate $\nabla J(\theta)$**
    - **Do this iteratively** to get an even smaller value J(θ)

Estimating parameters

- How to change the values of parameters θ in order to minimize J(θ)?
  - A more **clear example**
    - if initial parameters of θ = (θ_0, θ_1, θ_2, ..., θ_k) and at that point the cost function is J(θ), then if the value of $\frac{\partial J(\theta)}{\partial \theta_2}$ is negative, this means that **a small increase** (update/change) of parameter θ_2 leading to parameters θ' = (θ_0, θ_1, **θ_2+ε** ..., θ_k) (leaving all other θs the same) **will decrease J(θ)**. If it's positive, decrease θ_2 to get θ' = (θ_0, θ_1, **θ_2-ε** ..., θ_k) , to get a smaller J(θ).
      - i.e. J(θ') < J(θ)
    - **Do the same for each and all θs** in the linear regression model and update their values accordingly.
    - **Do such update for each θ iteratively** (i.e. many times over).

Estimating parameters

- How to change the values of parameters θ in order to minimize J(θ)?
  - In Gradient descent, each parameter is updated/changed, at each iteration, using the following formula:

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$$

a is a real value > 0, called the learning rate. It is a constant given as input to gradient descent. While the partial derivative will give us the direction in which the cost function will decrease, it does not specify how big the increase of θ should be. This is specified by the value of the learning rate a. Can be imagined as the step by which the θ will change. **Setting the appropriate value for a is very important and affects the significantly the algorithm.**

Estimating parameters

- How to change the values of parameters θ in order to minimize J(θ)?
  - For a **multiple linear regression model**, you can actually calculate $\frac{\partial J(\theta)}{\partial \theta}$ for all θs, resulting in the following update formulas for the parameters θ:

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right)$$

Update for the constant term/intercept

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right) x_j^{(i)}$$

Update for all other parameters in the linear regression model

Where
m: number of observations in training set
$h_\theta(x^{(i)})$: value of linear regression model for the values of independent variables in observation i of the training set
$y^{(i)}$: value of the dependent variable in observation i of the training set
α : learning rate
$x_j^{(i)}$: value of independent variable x_j in observation i of the training set

Form of linear regression model: $h_\theta\left(x^{(i)}\right) = \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_k x_k^{(i)} + \theta_0 + \varepsilon$

Estimating parameters

- How to change the values of parameters θ in order to minimize J(θ)?
  - In **matrix form**, the previous update formulas for parameters θ can be written as

$$\boldsymbol{\theta := \theta - \alpha \frac{1}{m} X^T \,(X\theta - Y)}$$

Where:
m: Number of observations in training set
$\boldsymbol{\theta}$: The vector of (k+1) parameters of the linear regression model
$\boldsymbol{X}$: The mx(k+1) matrix of values of independent variables in the linear regression model, with the first column all 1 (ones).
$\boldsymbol{Y}$: The vector of m values of the dependent variables in the training set
a : the learning rate, given as input

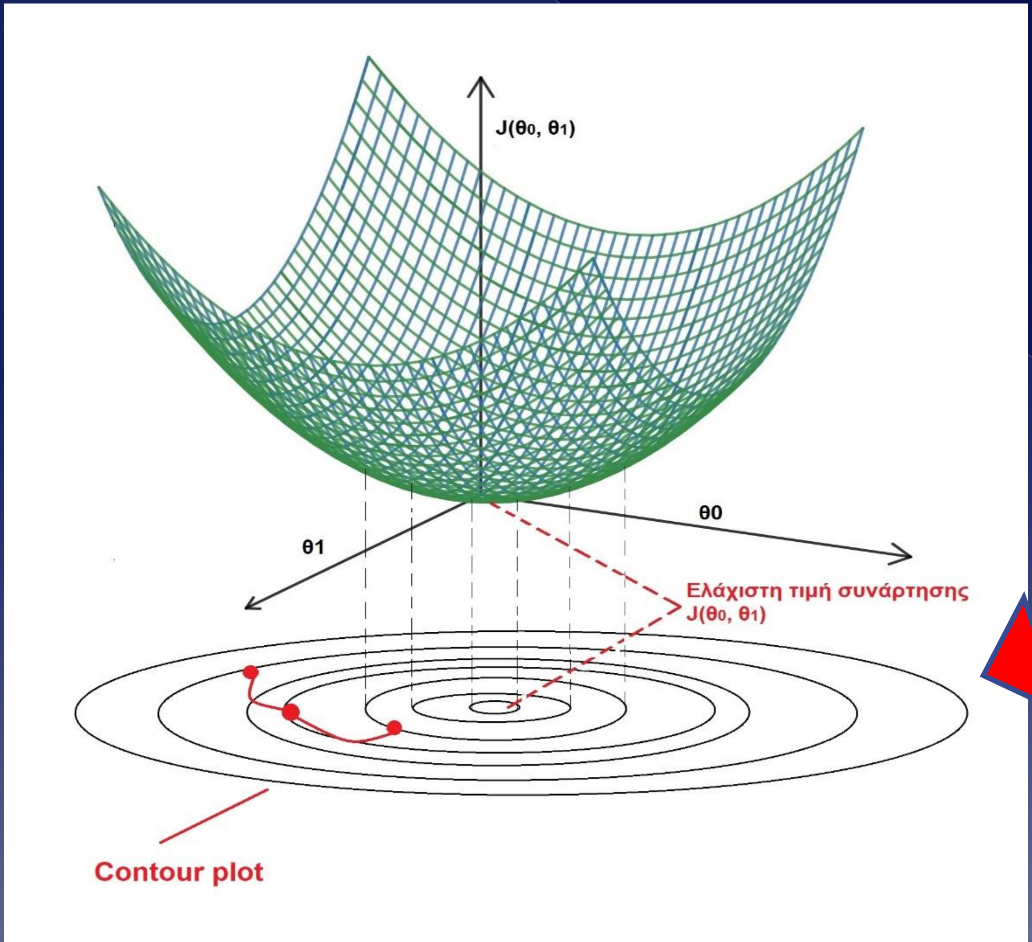
Estimating parameters

- Gradient descent algorithm
  - Pseudocode

```
Initialize vector of parameters θ with random values
Initialize costVector # We will store the value of the cost function for each
                        iteration in this vector.
α = 0.01 # Set learning rate. See later how to come up with an appropriate value.

# Start iterations of Gradient descent
while termination conditions not met {

      update θ vector with  $\theta := \theta - \alpha \frac{1}{m} X^T (X\theta - Y)$
      calculate value of cost function J(θ) for the newly calculated values of θ
      Store value of cost function into vector costVector
}

# Vector θ will contain the estimated parameters which minimize J(θ)
print θ vector # Print the estimated parameters
plot costVector # Plot the costVector
```

Estimating parameters

- Gradient descent usually depicted as a contour plot



Convergence of θs to the values which minimize J(θ) is usually depicted as Contour plot.

In a Contour plot, each circle represents the values of θ that lead to the same value of J(θ).

Estimating parameters

- When does Gradient descent terminate?
  - 3 possible termination conditions
    - **When a predefined number of iterations have been completed.** Typical number of iterations are n=50, 20000 or greater depending how fast the algorithm converges
    - **When the improvement of the cost function is smaller than a predefined value**
    - **Early stopping**. With the current "version" of the cost function, **calculate the cost on a validation set (different from training set)** at each iteration. Compare the two consecutive values of J(θ) and if J(θ) starts to increase, terminate the algorithm. Used to **address overfitting**.

Estimating parameters

- Gradient descent algorithm with predefined number of iterations as termination condition

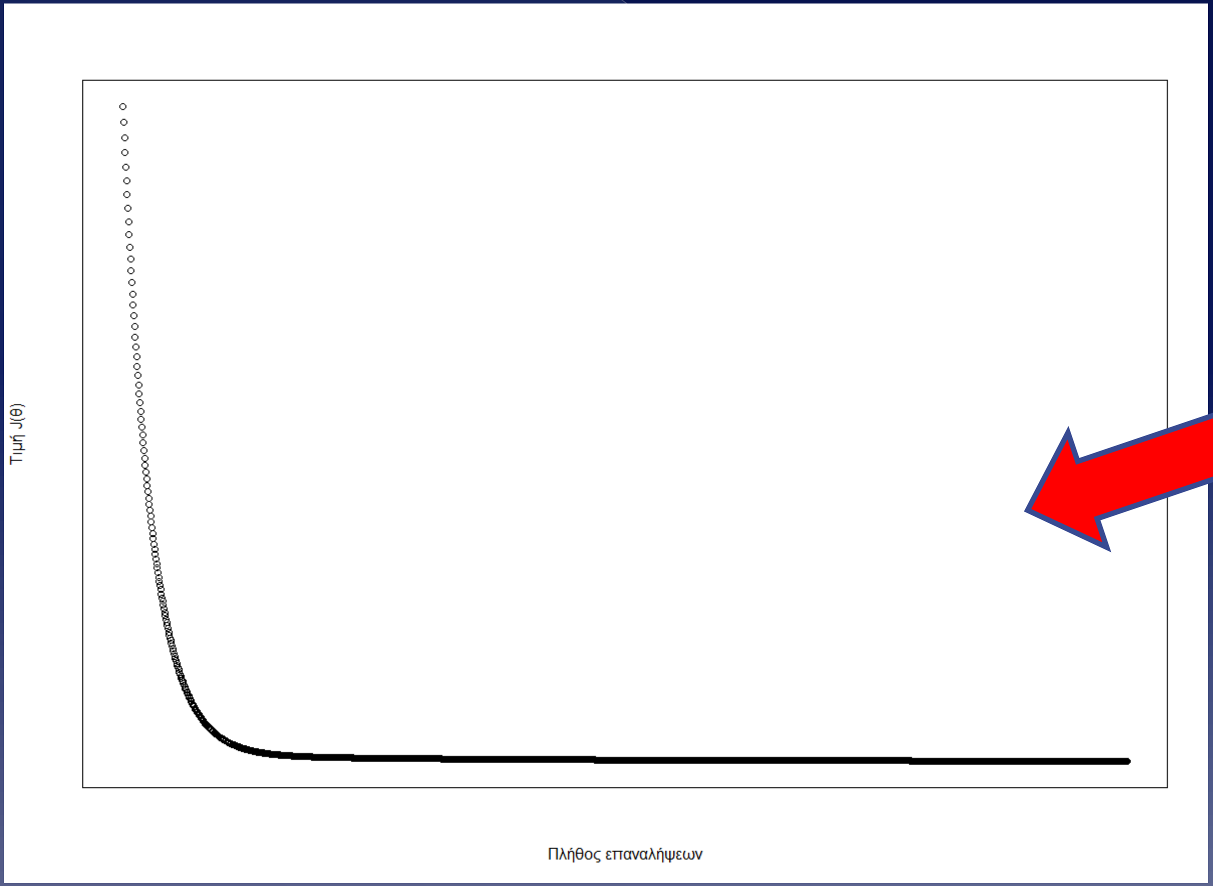
```
Initialize vector of parameters θ with random values
Initialize costVector # We will store the value of the cost function J(θ) for each iteration here
α = 0.01 # Setting the learning rate
numIterations = 10000 # Number of iterations to carry out
n = 0 # How many iterations we have done
while n < numIterations {
      update θ vector with  $\theta := \theta - \alpha \frac{1}{m} X^T (X\theta - Y)$
      calculate value of cost function J(θ) for the newly estimated values of θ
      Store value of cost function into vector costVector
      n = n + 1   # Next iteration
}

# Vector θ will contain the estimated parameters which minimize J(θ)
print θ vector
plot costVector
```

Estimating parameters

- The learning rate a
  - Setting the **learning rate a** to the proper value is critical!
    - Determines **if and how fast Gradient descent** converges to the minimum of the cost function.
    - If the **value of the learning parameter is too small**, Gradient descent may **converge very slowly**
    - If the **value of the learning parameter is too large**, Gradient descent may **not converge at all** to the proper values of θ which minimize $J(\theta)$
  - How to check if learning parameter a is too small, too big or just appropriate?
    - Empirically, plot the cost function and see its shape

Estimating parameters

- The learning rate a
  - **Appropriate value** of learning rate

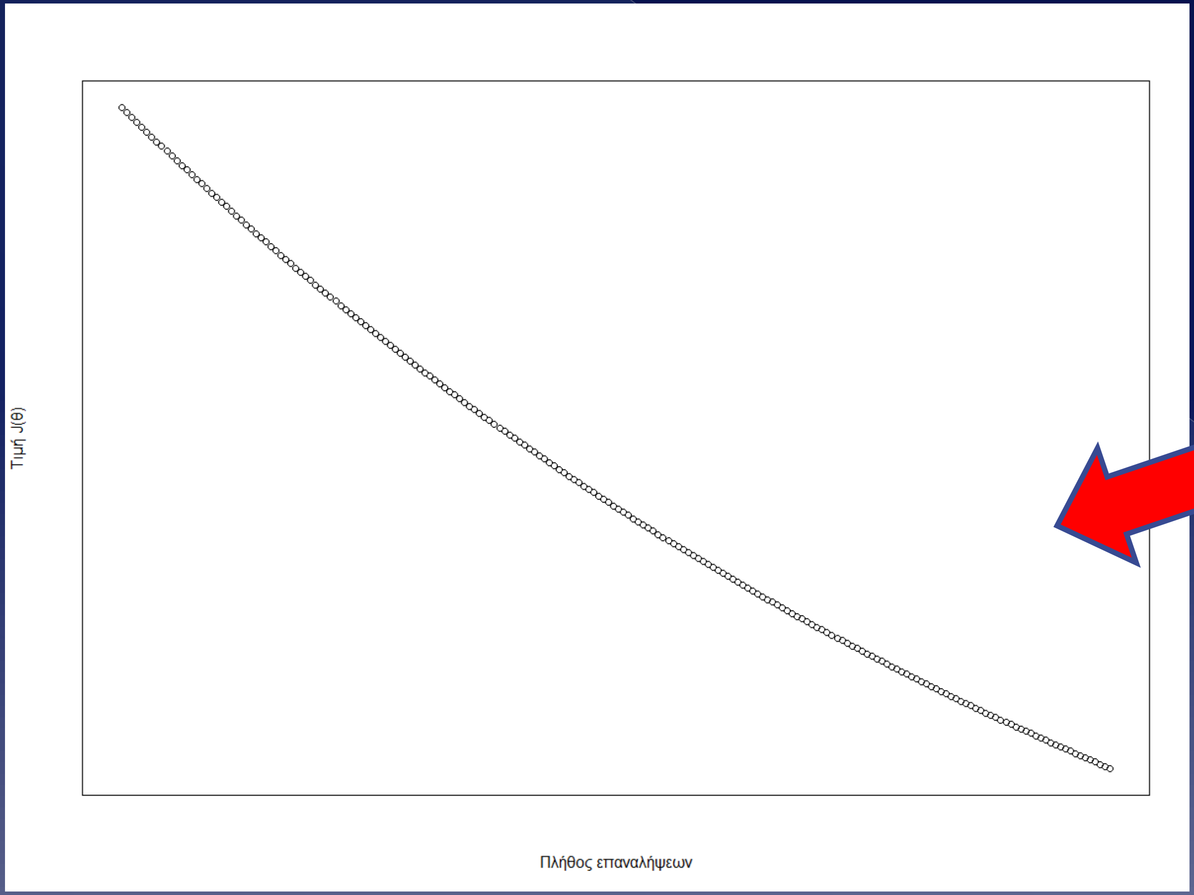


If the value of the learning rate is appropriate, the cost function J(θ) plotted against the number of iterations will have such shape. Cost function shows a steep drop and then a gradual improvement.

To check if the selected value is appropriate, run Gradient descent and plot the cost function.

Estimating parameters

- The learning rate a
  - **Too small value** of the learning rate a

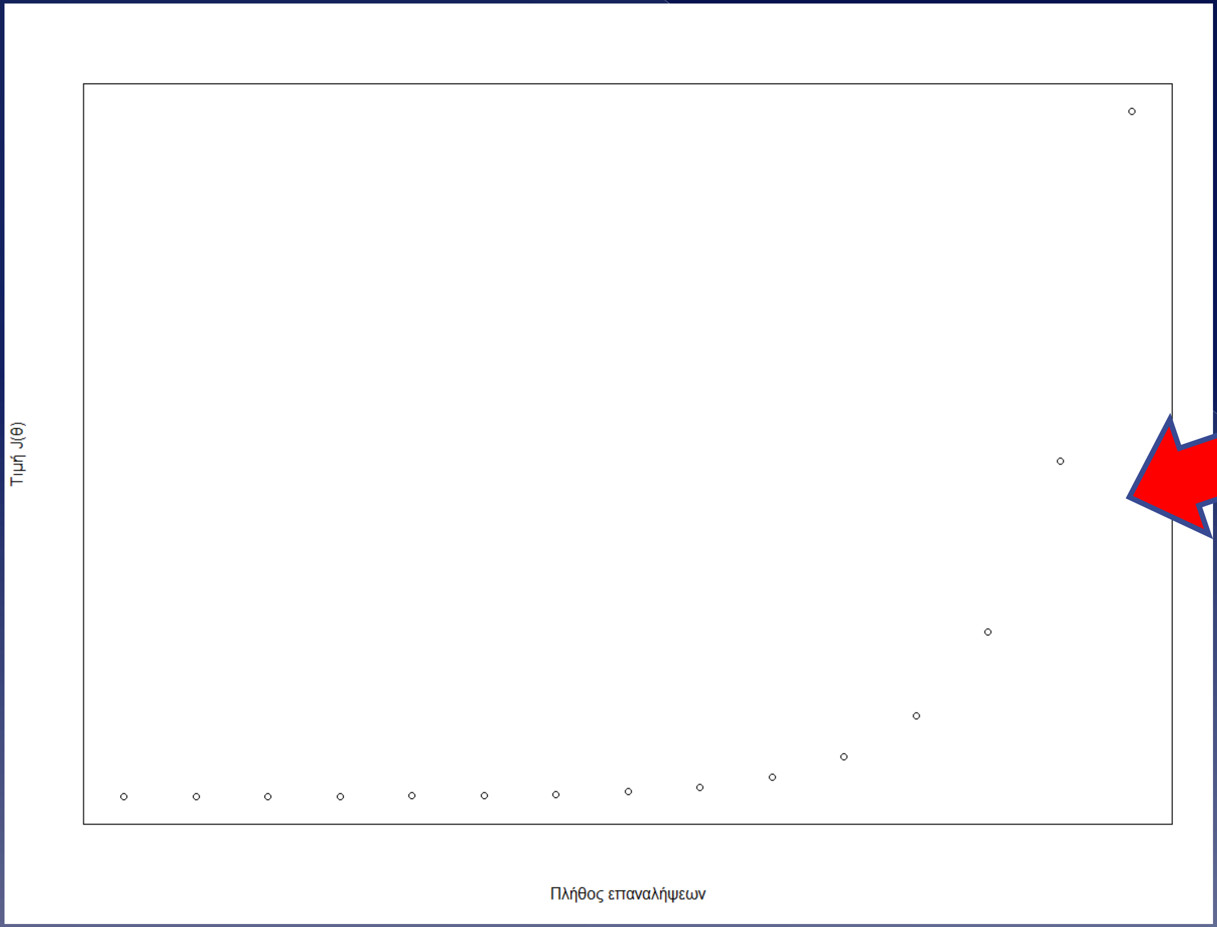


If the value of the learning rate is too small, the cost function J(θ) plotted against the number of iterations will have such shape.

This means Gradient descent will converge very, very slowly to the appropriate values of θs that minimize J(θ).

Estimating parameters

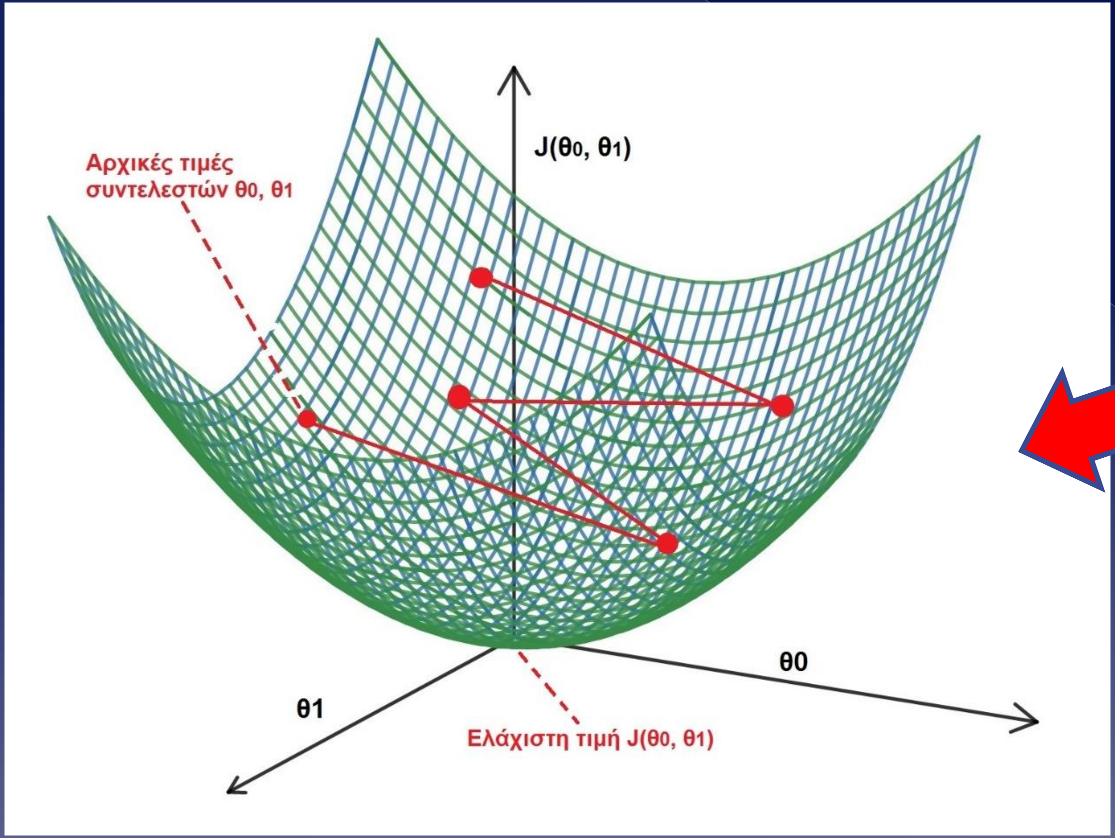
- The learning rate a
  - **Too big value** of the learning rate a



If the value of the learning rate is too big, the cost function J(θ) plotted against the number of iterations will have such a shape. The cost function increases with each iteration.

Estimating parameters

- The learning rate a
  - **Too big value** of the learning rate a



If the value of the learning rate a is too big, Gradient descent may overshoot the proper values of θ that minimize the cost function.

Overshooting happens because the value of a is too big and hence the update $\theta_j = \theta_j - a \frac{\partial J(\theta)}{\partial j}$ the new values of θ_j may increase by too much, missing the values for which J(θ) is minimized. Gradient descent then diverges from the proper values of θs.

Estimating parameters

- The learning rate a
  - Appropriate values for learning rate?
    - Typical values of the **learning rate a are 0.001, 0.01, 0.1**
    - Execute Gradient descent **with such values of the learning rate a** and **plot the cost function J(θ)** as a function of the number of iterations. Compare the shape of the plot with the plots shown previously.
    - If learning rate is too small, increase it by some amount e.g. from 0.01 to 0.03. Execute Gradient descent again and plot the cost function. Stop if the plot of the cost function has the appropriate shape.

Estimating parameters

- The version of Gradient descent discussed previously is the "plain vanilla" style of the algorithm also known as **"Batch Gradient Descent"**
- Two other **versions of Gradient Descent** available that improve performance dramatically in Big data settings:
  - **Stochastic gradient descent - SGD**
  - **Mini-Batch gradient descent - MBGD**

Versions of Gradient descent

- Why the need to improve the performance of Gradient descent?
  - If **number of observations in training set is large** (e.g. **10000000 observations/records** or more), there are two main concerns with Batch Gradient descent:
    - Entire **training set** must be **stored into memory (RAM)**
    - **Update formulas** must **iterate over the entire training set** to calculate on step for all parameters **in each iteration**.
    - In such settings, **Batch Gradient descent is computationally expensive!**

Versions of Gradient descent

- **Concern**: Entire training set into memory
  - Looking at the matrix form of the update formula: **Does is fit into memory?**

$$\theta := \theta - \alpha \frac{1}{m} X^T (X\theta - Y)$$

To execute this calculation, the entire matrix of the values of the independent variables X must be loaded into the main memory (RAM). What if it does not fit into RAM? E.g. if there are 10000000 observations and 50 numeric variables, you'll need to store 10000000 * 50 = 500000000 numbers and since each number requires at least 4 bytes you need 500000000 * 4= 2000000000 bytes of data in RAM or ~1.8GB of RAM. Do you have it?

Versions of Gradient descent

- **Concern**: Iterate over the entire training set at each iteration
  - Looking at the analytic formula indicates better the problem (Note: the same argument holds for the matrix form, but it's clearer in the analytic form of the update formula): **Can be slow in big data contexts**

If the training set has m=10000000 observations, we iterate over all 10000000 observations just make one (1) update to one (1) parameter at one (1) iteration! Considering that we have many parameters, we traverse the 10000000 observations many times at each iteration. This makes Batch Gradient descent slow.

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j{}^{(i)}$$

Versions of Gradient descent

- Why the need to improve the performance of Gradient descent?
  - The solution in such big data environments is simply **not to iterate over the entire training set at each iteration!**
  - The two other versions of Bath Gradient descent treat/scan the training set differently

Versions of Gradient descent

- Stochastic Gradient Descent - SGD
  - At **each iteration, SGD uses only one observation** of the training set to update the parameters (instead of the entire training set in GD)

```
Initialize all parameters θ_j with random values
Initialize costVector # We will store the value of the cost function J(θ) for each iteration here
α = 0.01 # Setting the learning rate
Randomly shuffle the training set # To ensure that the observations do not have some kind of order
while termination criteria not met{
     Calculate cost function and store its value in costVector
     for each observation i in training set {
       for each parameter θ_j {
           Set new value of parameter $new_{\theta_j} := \theta_j - \alpha\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$
      }
      calculate value of cost function J(θ) for the newly estimated values of θ
      Store value of cost function into vector costVector
    }
     θ_j := new_θj
     n = n + 1    # Next iteration
}

# Vector θ will contain the estimated parameters which minimize J(θ)
print θ_j
plot costVector
```

Versions of Gradient descent

- Pros/Cons of SGD
  - Pros
    - It's a so-called **online algorithm** – you **see the update of parameters immediately**, in a sequential fashion, during their estimation i.e. in real time. That's not possible with Batch GD
    - Does **not require entire training set in memory**
    - **Avoids local minima of $J(\theta)$**
  - Cons
    - Can be **noisy** i.e. parameters jump around at each epoch with greater variance between epochs (epoch = one update of all parameters)

Versions of Gradient descent

- Mini-Batch Gradient Descent - MBGD
  - **MBGD** does not use one single observation of the training set to update the parameters. **It uses a "small batch"** of training set observations – typically between 2 and 100 observation in each batch.
    - To do this, we cut the large training set into smaller training subsets, and use these to update the parameters at each step
    - It's a method **"between" the extremes** of Batch Gradient descent (which uses entire training set for each parameter update) and Stochastic Gradient descent which uses only one observation to update the parameters.

Versions of Gradient descent

- Mini-Batch Gradient descent

```
Initialize all parameters θ_j with random values
Initialize costVector # We will store the value of the cost function J(θ) for each iteration here
α = 0.01 # Setting the learning rate
Randomly shuffle the training set # To ensure that the observations do not have some kind of order
Cut the training set into batches/subsets b_i each of size n_b such that b_i*n_b =m # Note: last batch
might be smaller than n_b
while termination criteria not met{
     Calculate cost function and store its value in costVector
     for each batch b_i {
       for each parameter θ_j {
           Set new value of parameter $new_{\theta_j} := \theta_j - \alpha \sum_{i=1}^{n_b}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$
      }
      calculate value of cost function J(θ) for the newly estimated values of θ
      Store value of cost function into vector costVector
    }
     θ_j := new_θj # update all parameters with new values
     n = n + 1   # Next iteration
}

# Vector θ will contain the estimated parameters which minimize J(θ)
print print θ_j
plot costVector
```

OLS vs Gradient descent

OLS	Gradient descent
Estimates the **same, unbiased parameters** for the same training set if the linear regression assumptions hold (No or little multicollinearity, No of auto-correlation of residuals, Homoscedasticity etc)	Estimates of **parameters are** approximations and biased. May result in different parameter estimates for the same training set and regression model.
Computationally **expensive in big data contexts**.	**Suitable in big data contexts** where number of variables and number of observation are very large
Can be used to estimate parameters **only for linear regression models**	Can be **used (and is used) to estimate parameters in nonlinear regression models.**
Offers **closed formulas (the normal equation)** for calculating the parameters	Does **not offer closed formula**. Parameter estimates are iteratively calculated
Requires **entire training set in RAM**	Versions of Gradient descent **do not require entire training set in RAM** (e.g. Stochastic Gradient descent)
Belongs to the field of **linear algebra**	Belongs to the field of **machine learning**
Taught and used mainly in social sciences	Taught and used mainly in computer science and engineering

Appendices

Deriving update formulas for parameters in linear regression models

- We will derive as an example the update for parameter θ$_1$ (parameter for an independent variable) - the same analysis holds for all other parameters

$$\frac{\partial J(\theta)}{\partial \theta_1} = \frac{\partial \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2}{\partial \theta_1} = \frac{\frac{1}{2m} \partial \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right)^2 - 2h_\theta\left(x^{(i)}\right)y^{(i)} + {y^{(i)}}^2\right)}{\partial \theta_1}$$

$$= \frac{1}{2m}\left[\frac{\partial \sum_{i=1}^{m} h_\theta\left(x^{(i)}\right)^2}{\partial \theta_1} - 2\frac{\partial \sum_{\iota=1}^{\mu} h_\theta\left(x^{(i)}\right)y^{(i)}}{\partial \theta_1} + \frac{\partial \sum_{\iota=1}^{\mu} {y^{(i)}}^2}{\partial \theta_1}\right] = < \textit{see next slide} >$$

Deriving update formulas for parameters in linear regression models

$$= \frac{1}{2m}\left[\frac{\partial \sum_{i=1}^{m} h_\theta\left(x^{(i)}\right)^2}{\partial \theta_1} - 2\,\frac{\partial \sum_{\iota=1}^{\mu} h_\theta\left(x^{(i)}\right) y^{(i)}}{\partial \theta_1}\right]$$

$$= \frac{1}{2m}\left[2\sum_{i=1}^{m} h_\theta\left(x^{(i)}\right)\frac{\partial h_\theta\left(x^{(i)}\right)}{\partial \theta_1} - 2\,\frac{\partial \sum_{\iota=1}^{\mu} h_\theta\left(x^{(i)}\right) y^{(i)}}{\partial \theta_1}\right]$$

$$= \frac{1}{2m}\left[2\sum_{i=1}^{m} h_\theta\left(x^{(i)}\right) x_1^{(i)} - 2\sum_{i=1}^{m} x_1^{(i)} y^{(i)}\right] = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_1^{(i)}$$

Thus the update formula for parameter θ1 becomes

$$\boxed{\theta_1 := \theta_1 - \alpha\,\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) {x_1}^{(i)}}$$

Now do the same for all other parameters θ0, θ2, θ3,…. and from this we get the closed form formulas for the updates of all parameters θ.