MACHINE LEARNING

Dr. Muhammad Awais Hassan
Department of Computer Science UET, Lahore

اَللّٰهُمَّ اُرْزُقْنِی عِلْمًا نَافِعًا وَاسِعًا عَمِيْقًا

اَللّٰهُمَّ اُرْزُقْنِی رِزْقًا وَاسِعًا حَلَالًا طَيِّبًا
مُبَارَكًا مِنْ عِنْدِكَ

WEEK 06

UNIVERIATE LINEAR REGRESSION

- So finally, our objective is to find the value of θ_0 and θ_1 such that the value of $J(\theta)$ is minimized.

- $\mathrm{J}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (y^i - (\theta_0 + \theta_1 x^i))^2$

REIVEW

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

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

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

ONE APPROACH TO FIND PARAMETERS

- Try different possible values of thetas
- For possible Theta range 1 to 10, The run time shall be 10^{20} for 20 variables
- This run time is more than the time of bigbang (10^{16})

THE PROBLEM

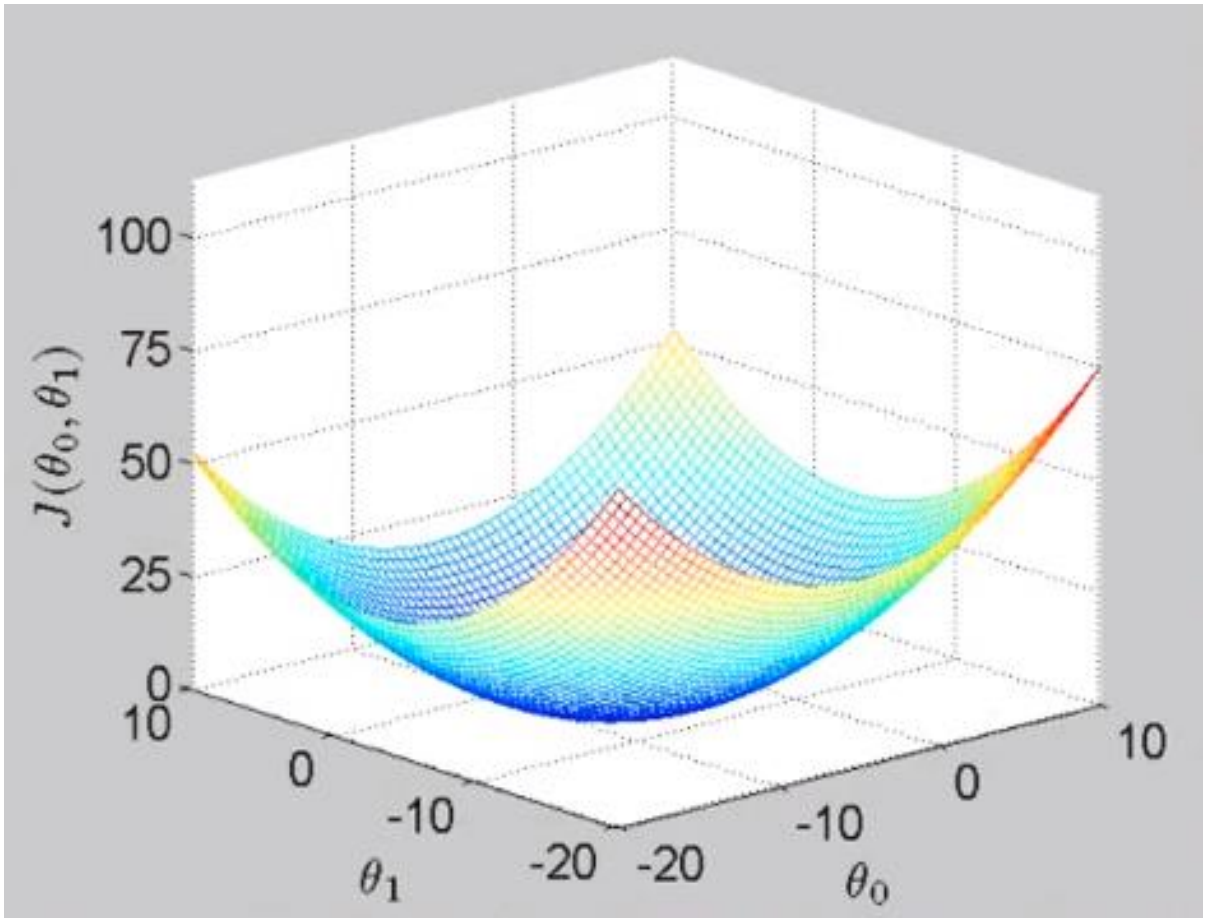
- Objective is to Minimize

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (y^i - (\theta_0 + \theta_1 x^i))^2$$

- But Brute Force is not a feasible solution. Need some better method.

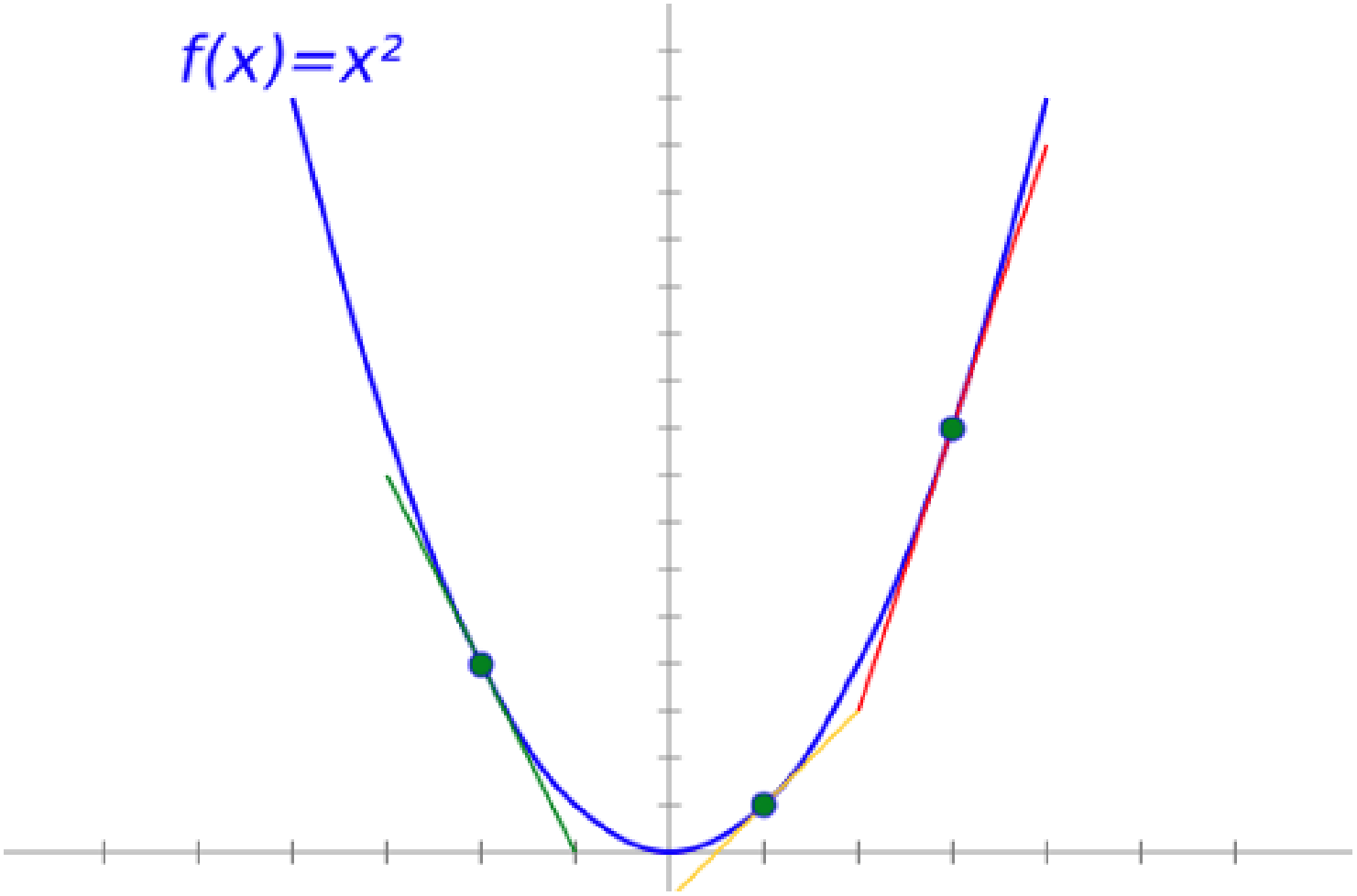
THE PROBLEM

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(y^i - (\theta_0 + \theta_1 x^i))^2$$



FIND THE RATE OF CHANGE IN Y AT DIFFERENT POINTS

REVIEW



WHY PARTIAL DERIVATIVES.

When the input of a function is made up of multiple variables, we want to see how the function changes as we *change one of those variables* while holding all the others constant

PARTIAL DERIVATIVES.

Can be thought of as "a tiny change in the function's output"

$$\frac{\partial f}{\partial x}$$

Used instead of "d" in usual $\frac{df}{dx}$ notation to emphasize that this is a partial derivative.

Multivariable function

Indicates which input variable is changed slightly.

Can be thought of as "a tiny change in x"

GRADIENT

- The gradient stores all the partial derivative information of a multivariable function.

REVIEW

WHAT IS THE GRADIENT

- The gradient of a scalar-valued multivariable function $f(x, y, \dots)$, denoted ∇f, packages all its partial derivative information into a vector:

$$\nabla f = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \\ \vdots \end{bmatrix}$$

In particular, this means ∇f is a vector-valued function.

REVIEW

GRADIENT VECTOR

Scalar-valued multivariable function

$$\nabla f(x_0, y_0, \dots) = \begin{bmatrix} \frac{\partial f}{\partial x}(x_0, y_0, \dots) \\ \frac{\partial f}{\partial y}(x_0, y_0, \dots) \\ \vdots \end{bmatrix}$$

∇f takes the same type of inputs as f

Notation for gradient, called "nabla".

∇f outputs a vector with all possible partial derivatives of f.

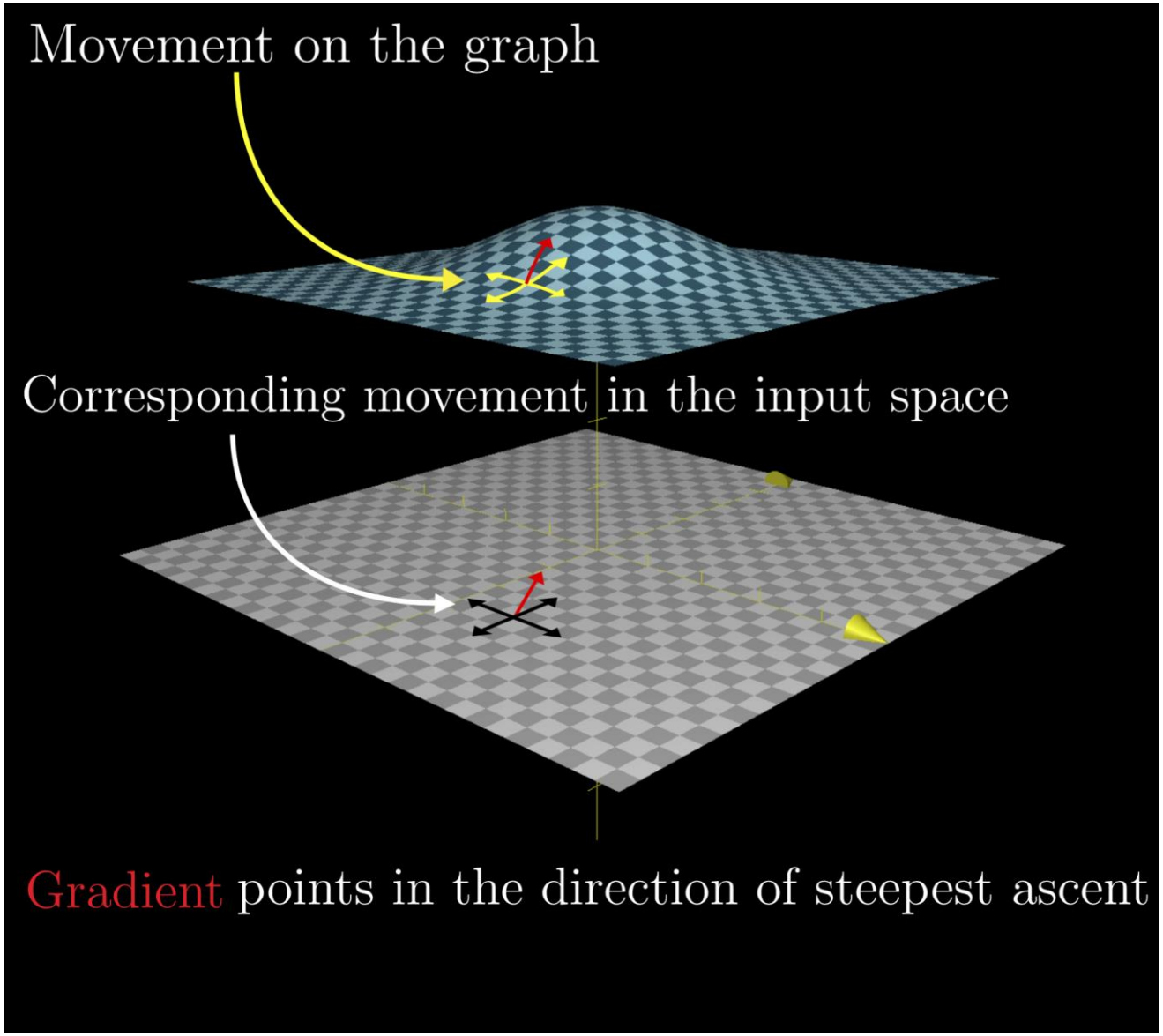
INTERPRETATION

- If you imagine standing at a point $f(x_0, y_0, \ldots)$ in the input space of f, the vector $\nabla f(x_0, y_0, \ldots)$ tells you which direction you should travel to increase the value of f most rapidly.

INTERPRETATION

- For example, if you step straight in the positive x direction, the slope is $\frac{\partial f}{\partial x}$
- if you step straight in the positive y-direction, the slope is $\frac{\partial f}{\partial y}$
- But most directions are some combination of the two

INTERPRETATION

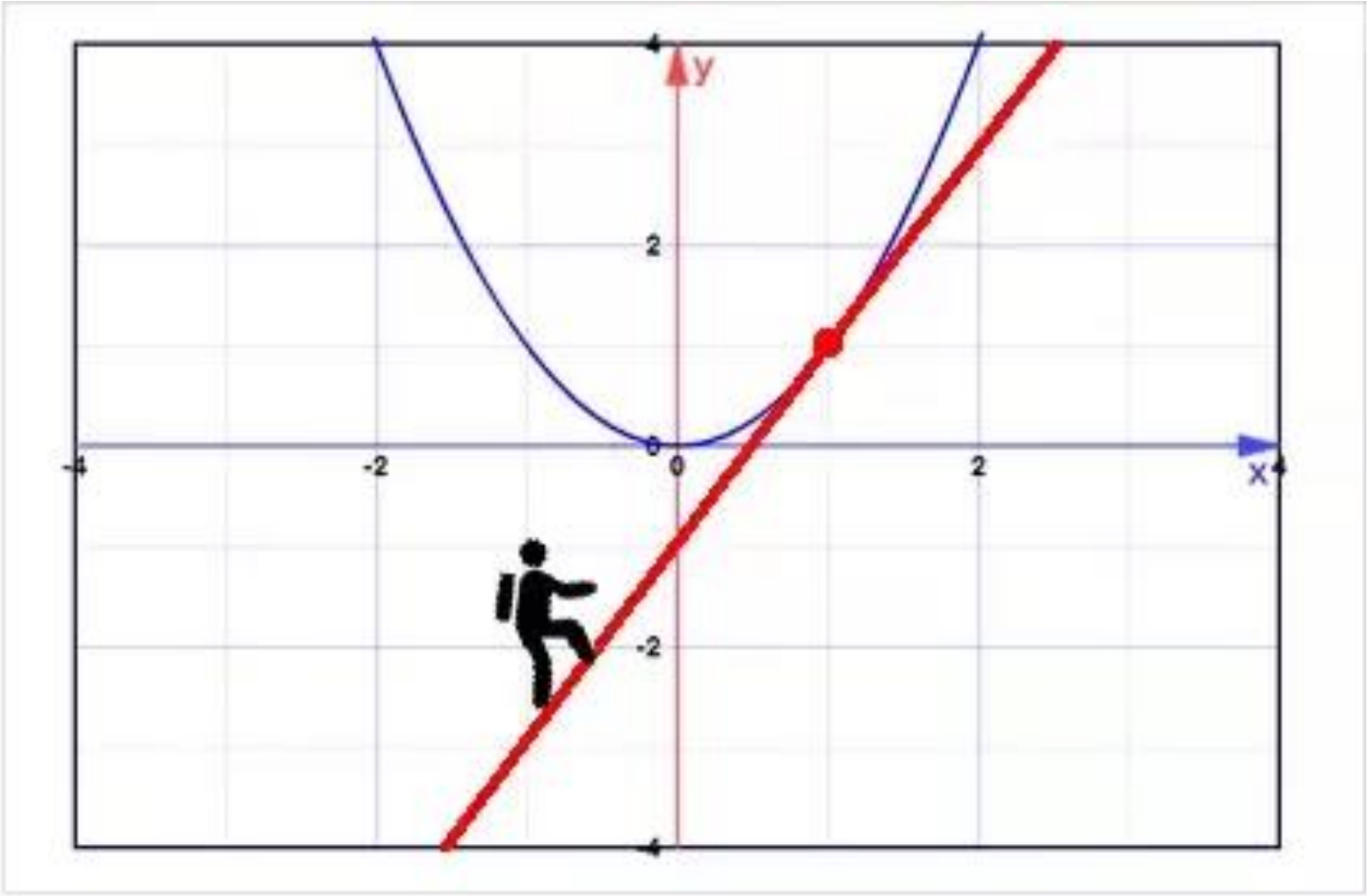


GRADIENT IS STEEPEST ASCENT

REVIEW

- **The most important thing to remember about the gradient**: The gradient of f at a points (x_0, y_0) is in the direction of steepest ascent.

INTERPRETATION



PROGRAMMING ASSIGNMENT

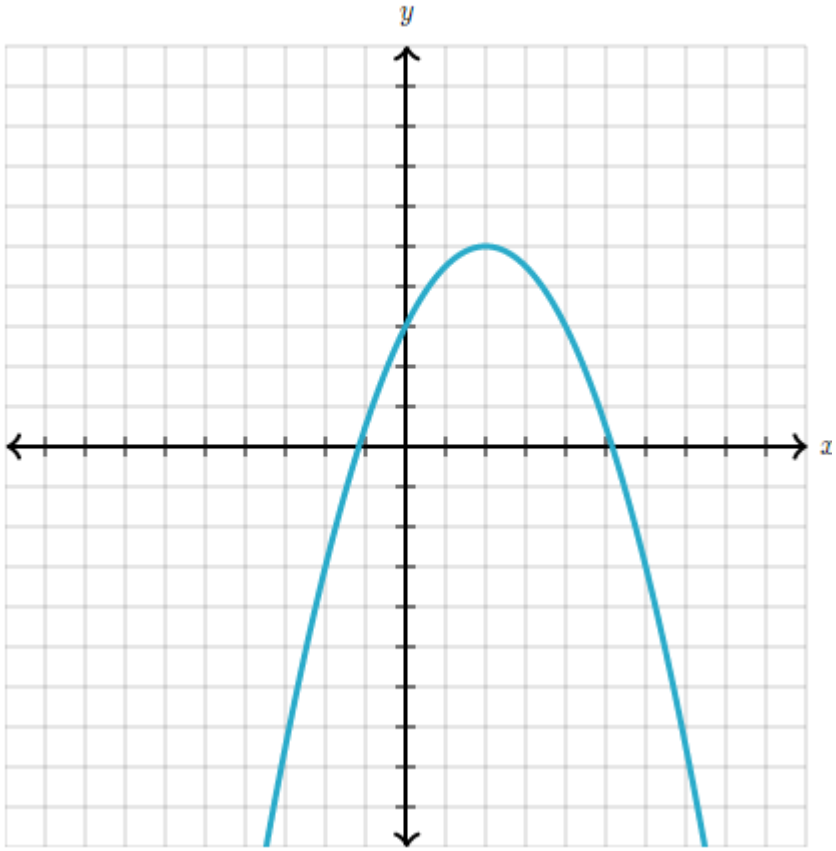
- Write a python program that calculate the gradient vector of a function at any given point. Multiply it with 2, and calculate new point in direction of the gradient ascent)
- Draw the Function on Graph and Highlight the starting point and new calculated point
- Reference: https://towardsdatascience.com/taking-derivatives-in-python-d6229ba72c64

VISUALIZING THE FUNCTION

2D REPRESENTATION

- it's common to visualize single value function in graph

$$f(x) = -\frac{1}{2}x^2 + 2x + 3$$



2D REPRESENTATION

- 2D Graphs are not the only way to visualize the functions

OTHER TYPES TO VISUALIZE

- There are four common types other than the graphs
  - Contour maps.
  - Parametric curves/surfaces
  - Vector fields.
  - Transformations

GRAPHS, OUR OLD FRIEND.

- Graphs have the benefit of showing both the input space and the output space at once, but as a result, they are highly limited by dimension.

- For this reason, they are only really useful for single-variable functions and multivariable functions with a two-dimensional input and a one-dimensional output.

CONTOUR MAPS

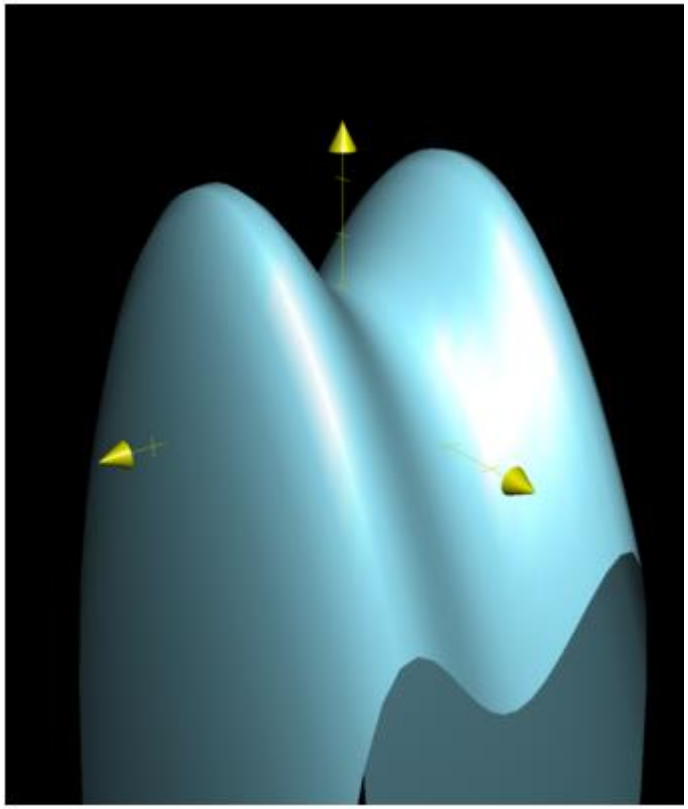
- Contour maps only show the input space and are useful for functions with a two-dimensional input and a one-dimensional output.

EXAMPLE

- Let we have multi variate function function

$$f(x, y) = x^4 - x^2 + y^2.$$

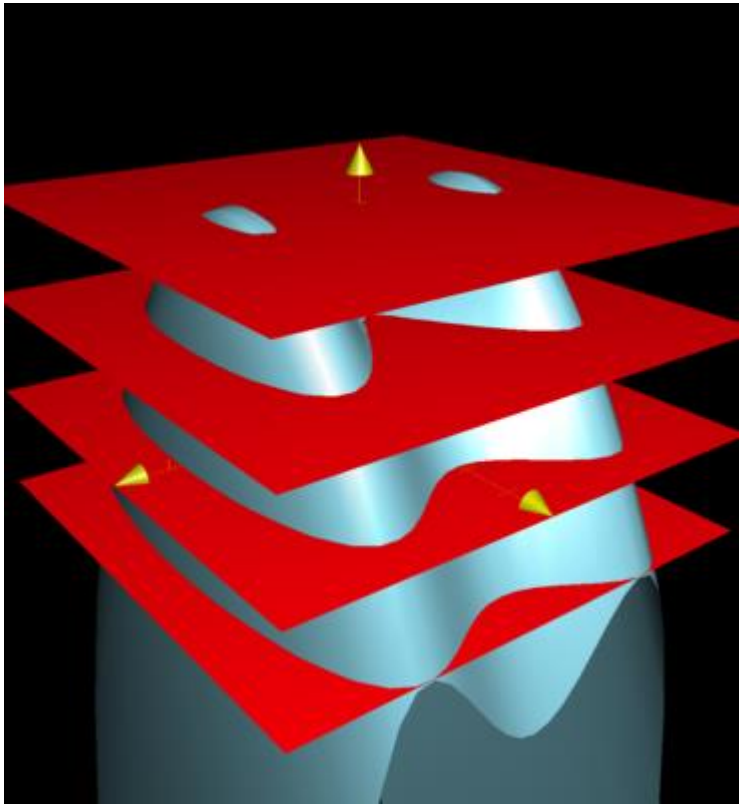
3D GRAPH OF FUNCTION IS



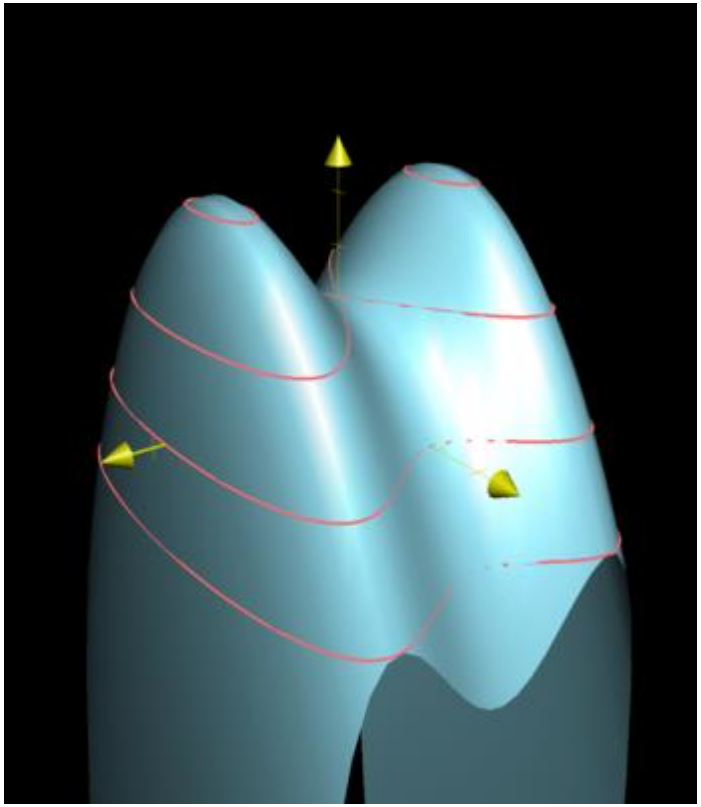
STEP 2

- Slice the graph with a few evenly-spaced level planes, each of which should be parallel to the xy-plane. You can think of these planes as the spots where z equals some given output, like $z=2z$

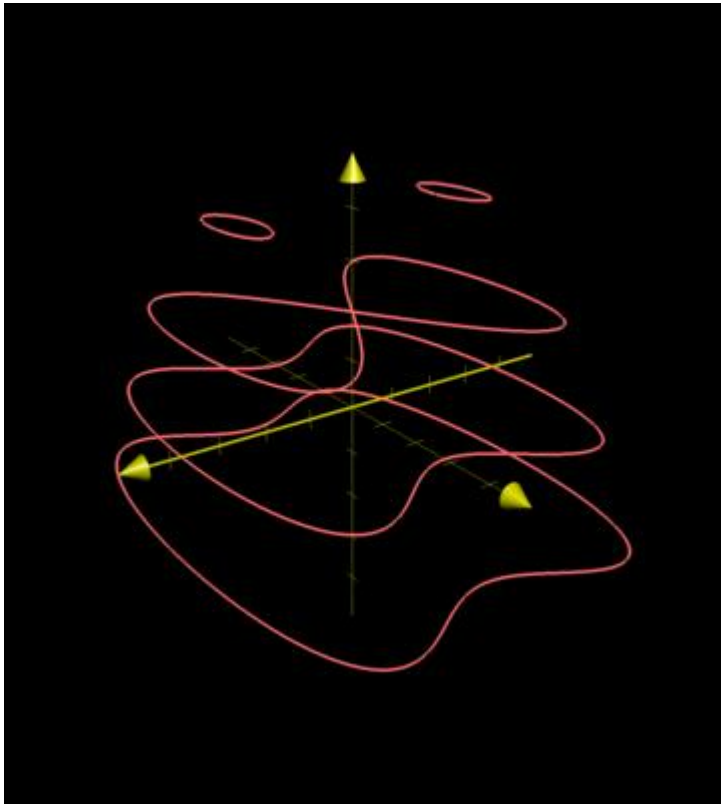
STEP 2



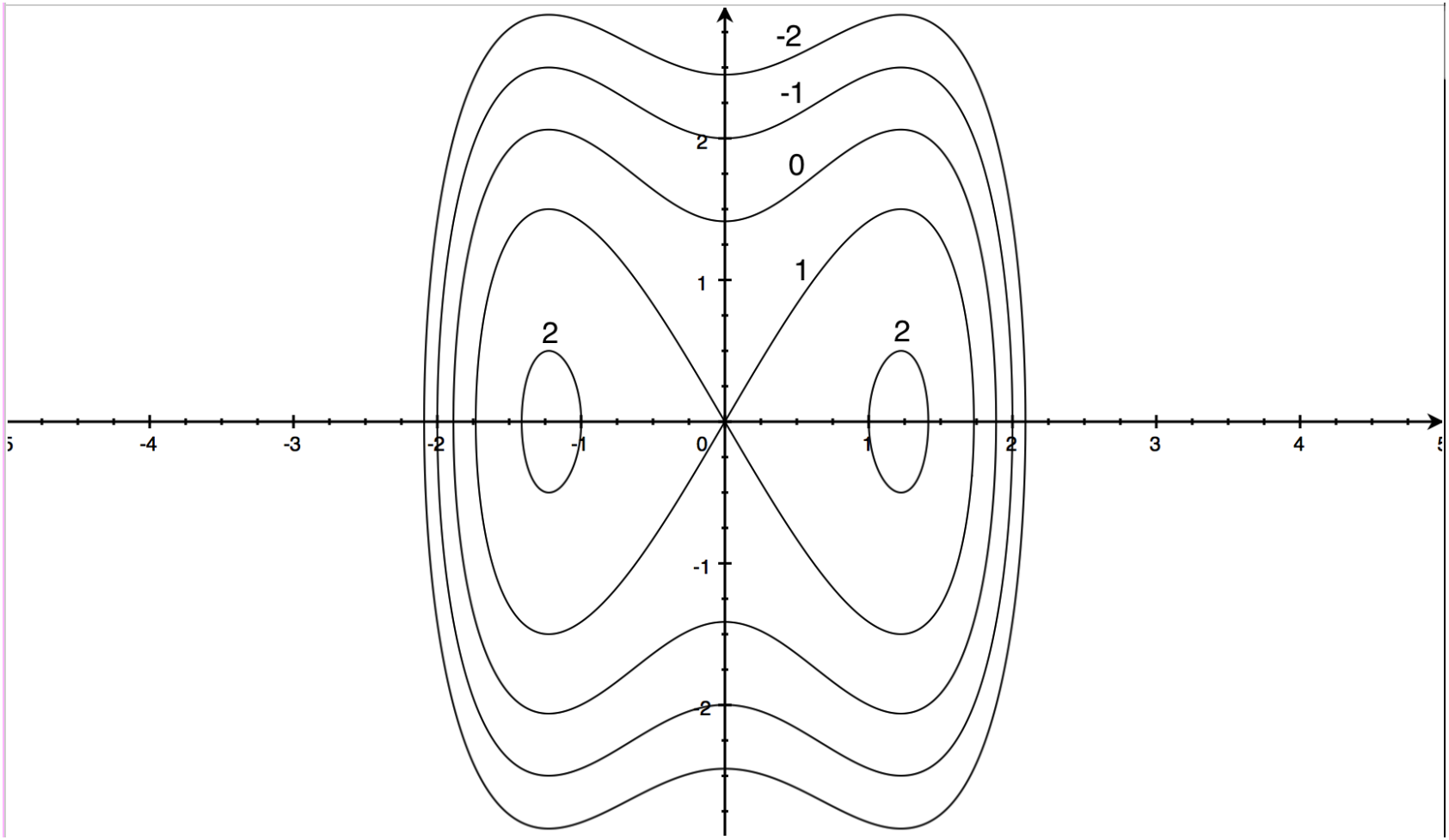
STEP 3: MARK THE GRAPH WHERE THE PLANES CUT INTO IT.



STEP 4: PROJECT THESE LINES ONTO THE XY-PLANE, AND LABEL THE HEIGHTS THEY CORRESPOND TO.



CONTOUR MAPS



GRADIENT DESCENT

Machine Learning

Linear regression with one variable

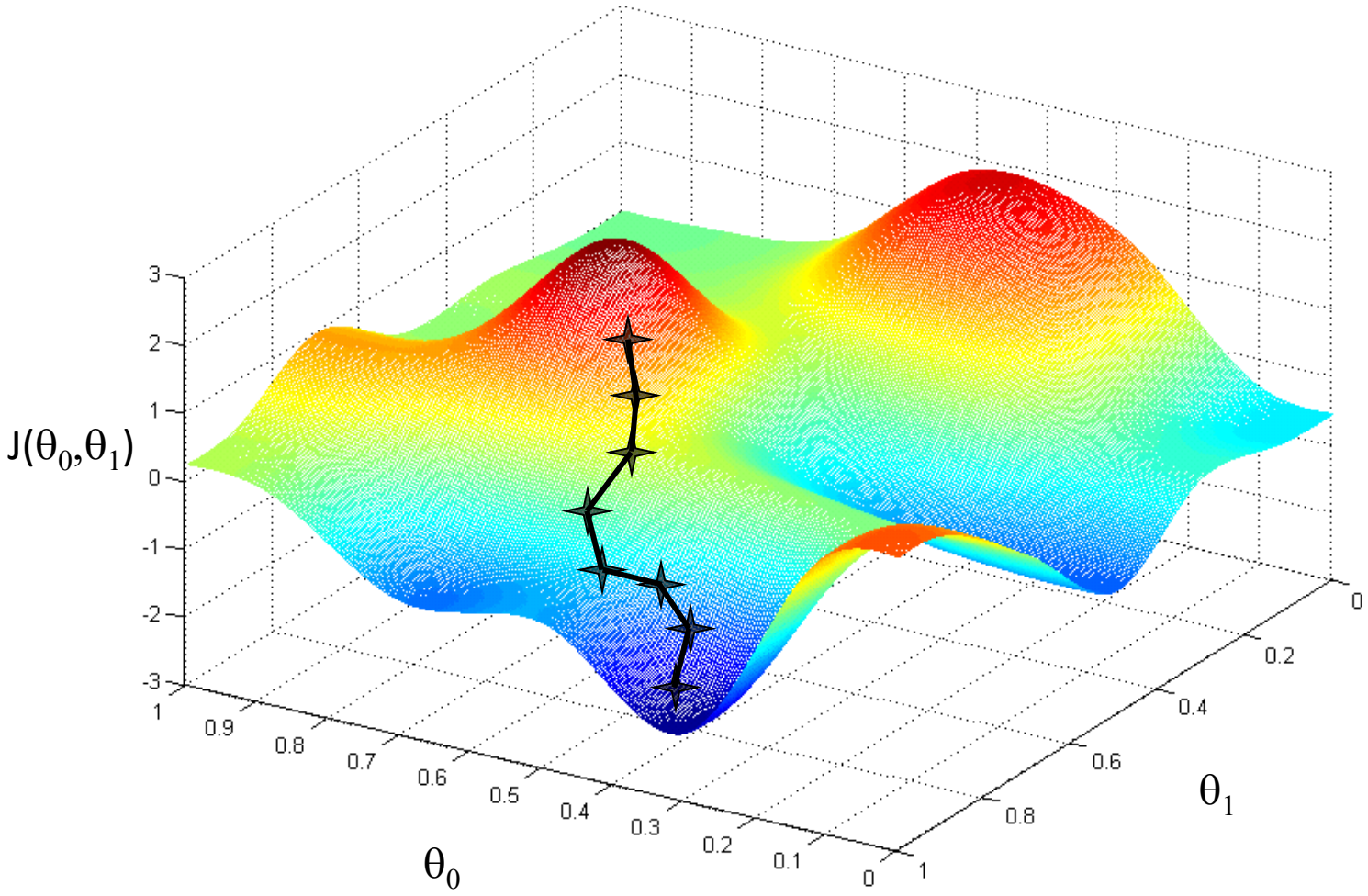
Gradient descent

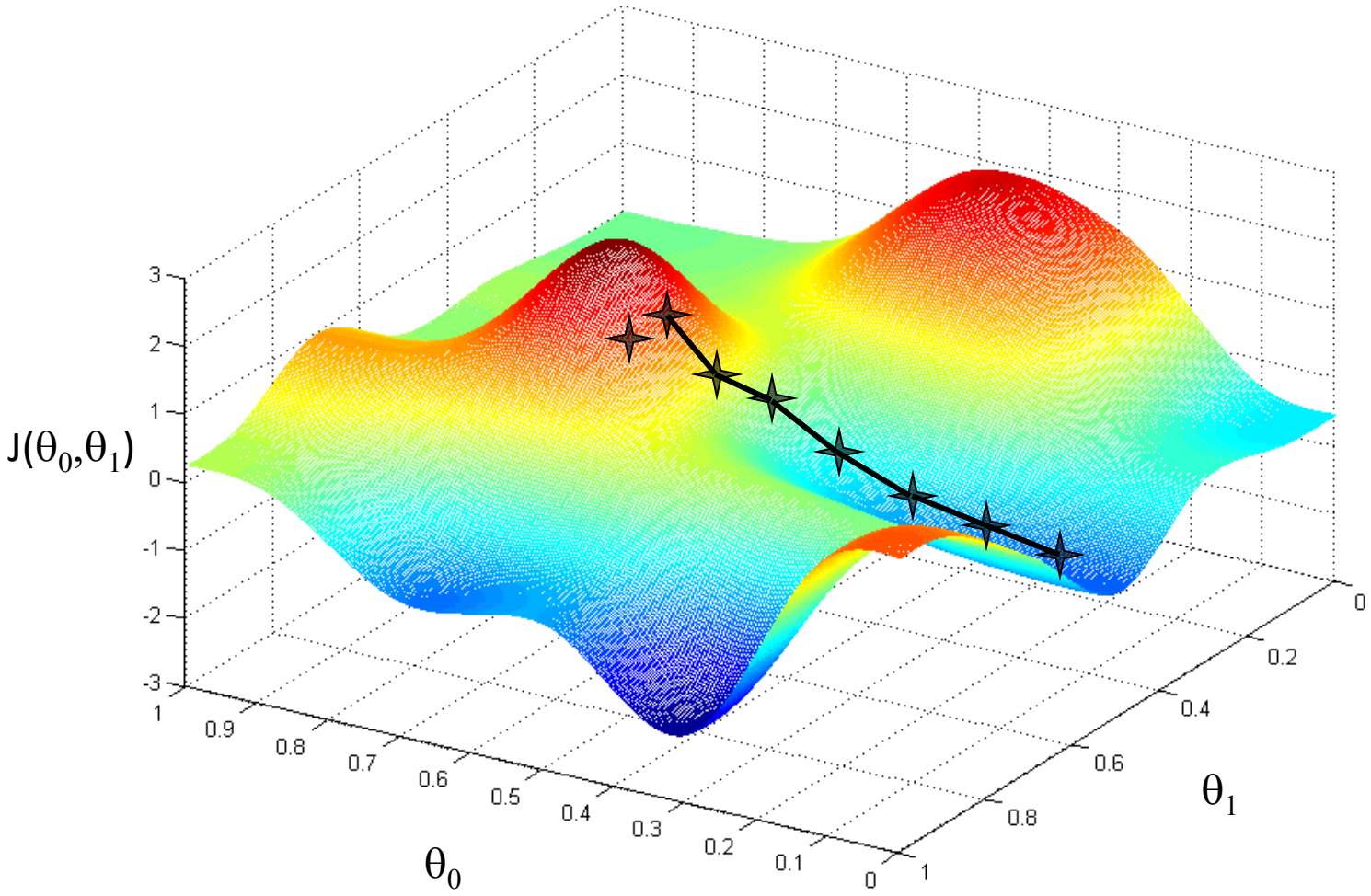
Have some function $J(\theta_0, \theta_1)$

Want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

Outline:

- Start with some
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum θ_0, θ_1


$J(\theta_0,\theta_1)$
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ_0
0
0.2
0.4
0.6
0.8
1
θ_1


$J(\theta_0,\theta_1)$
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ_0
0
0.2
0.4
0.6
0.8
1
θ_1

Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad (\text{for } j = 0 \text{ and } j = 1)$$

}

Correct: Simultaneous update

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\theta_1 := \text{temp1}$$

Incorrect:

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_1 := \text{temp1}$$

Machine Learning

Linear regression with one variable

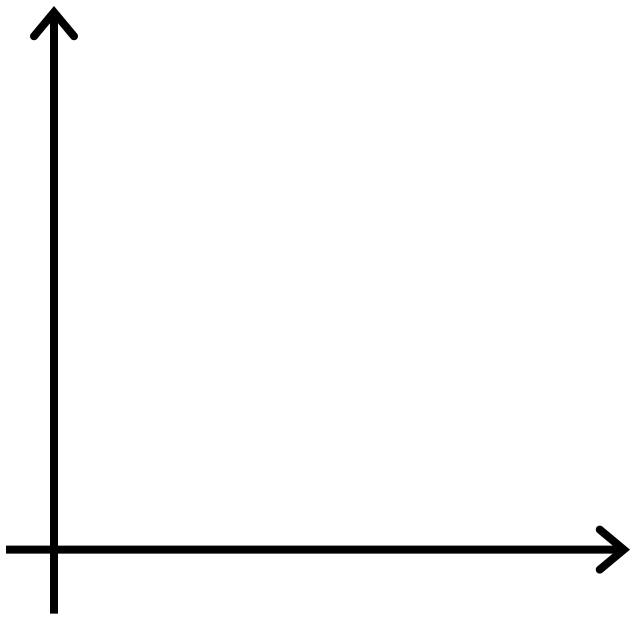
Gradient descent intuition

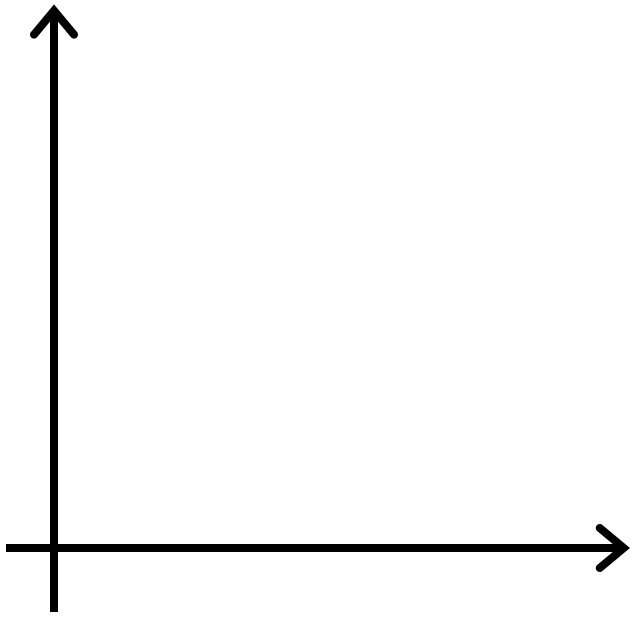
Gradient descent algorithm

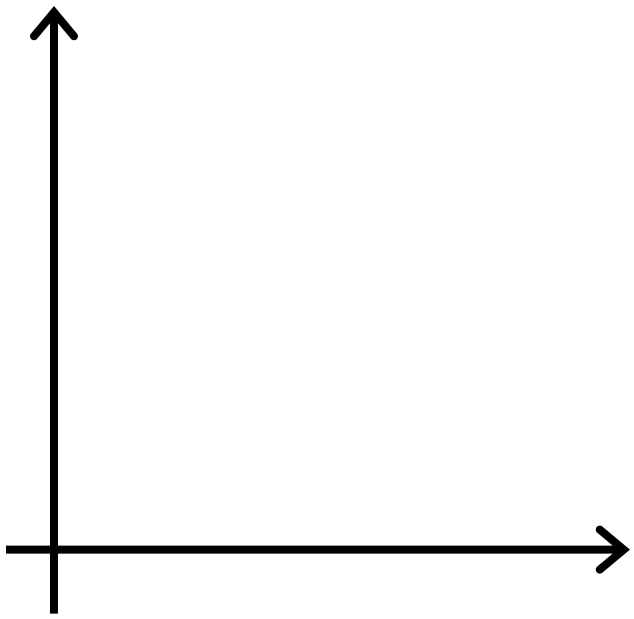
repeat until convergence {

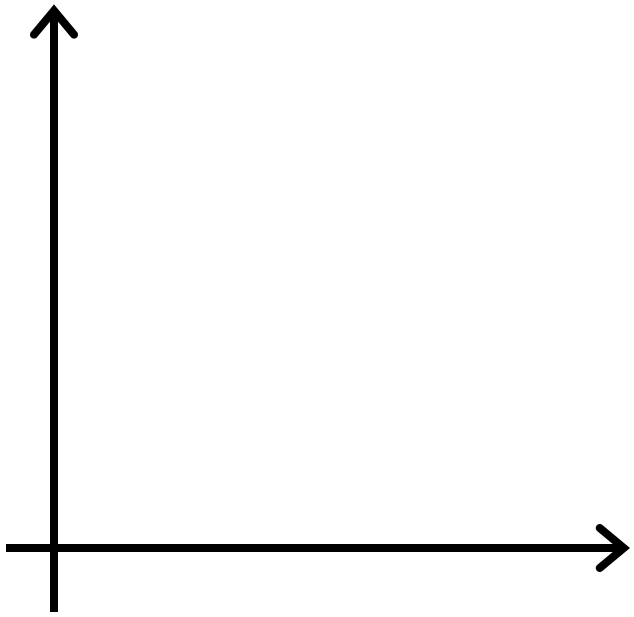
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad \text{(simultaneously update } j = 0 \text{ and } j = 1\text{)}$$

}



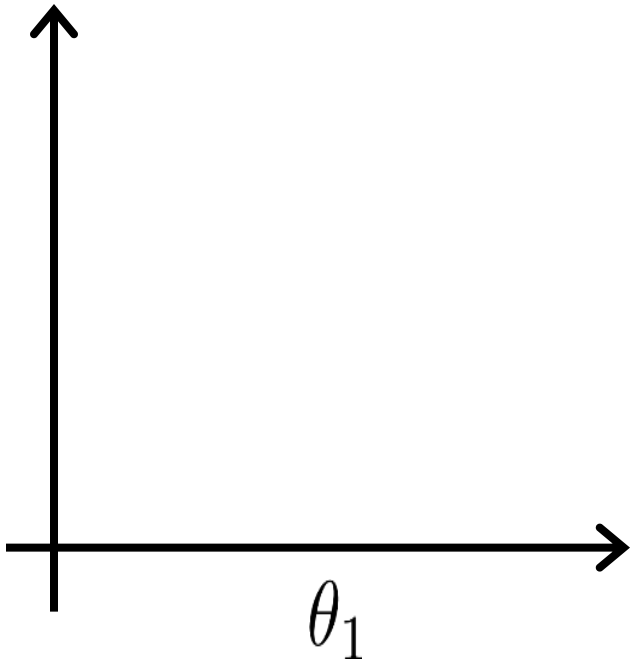




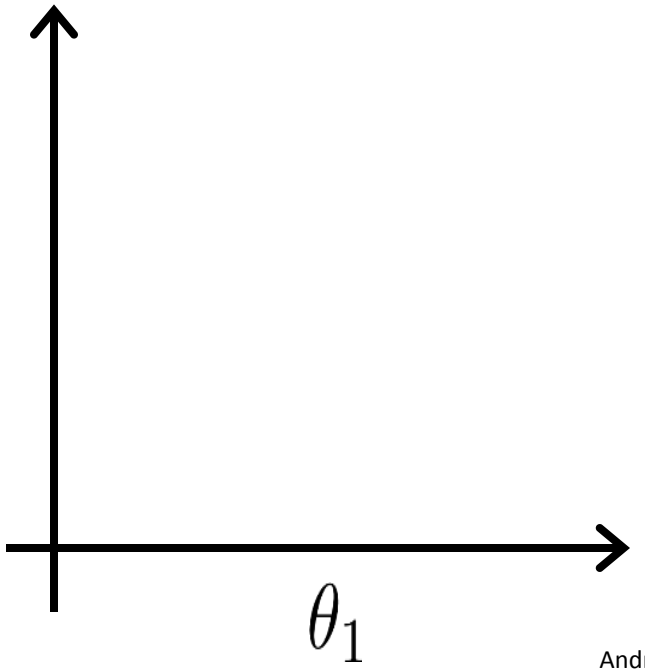


$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.

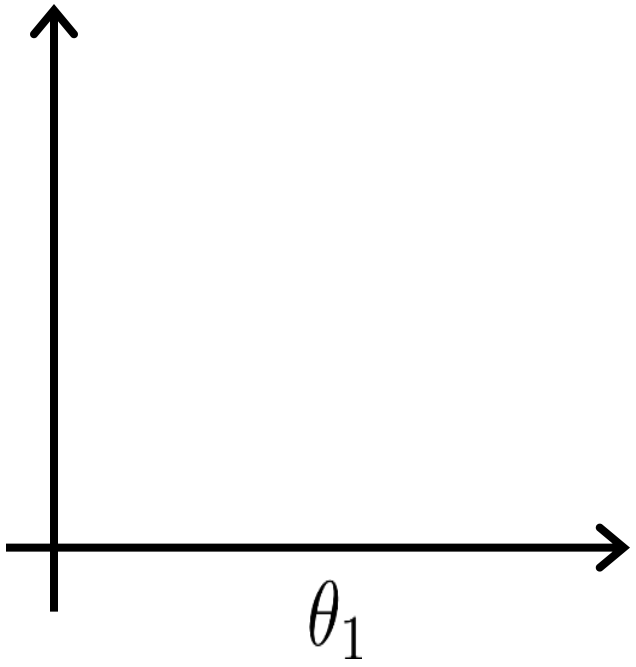


If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.

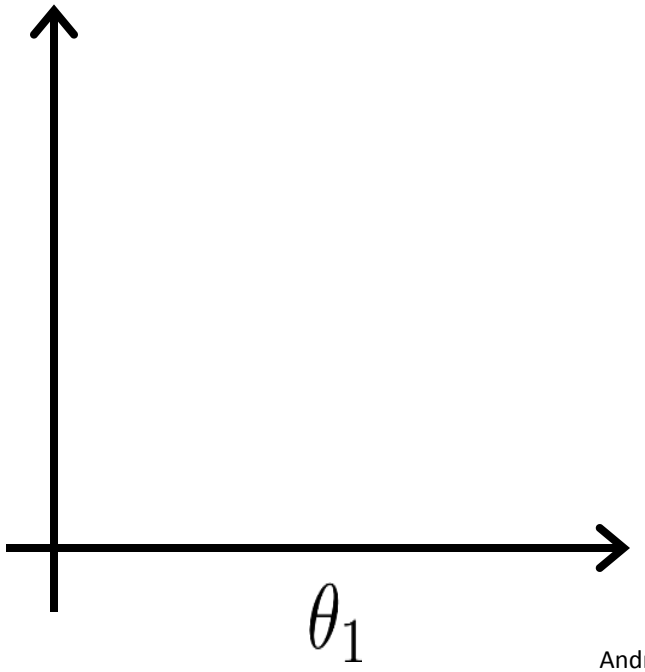


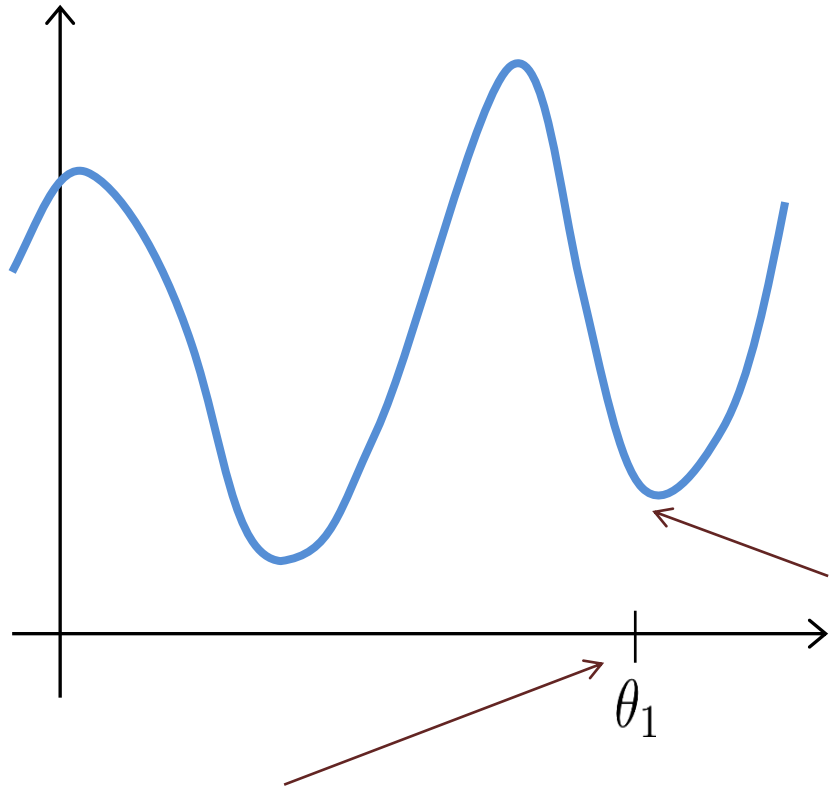
$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.



If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



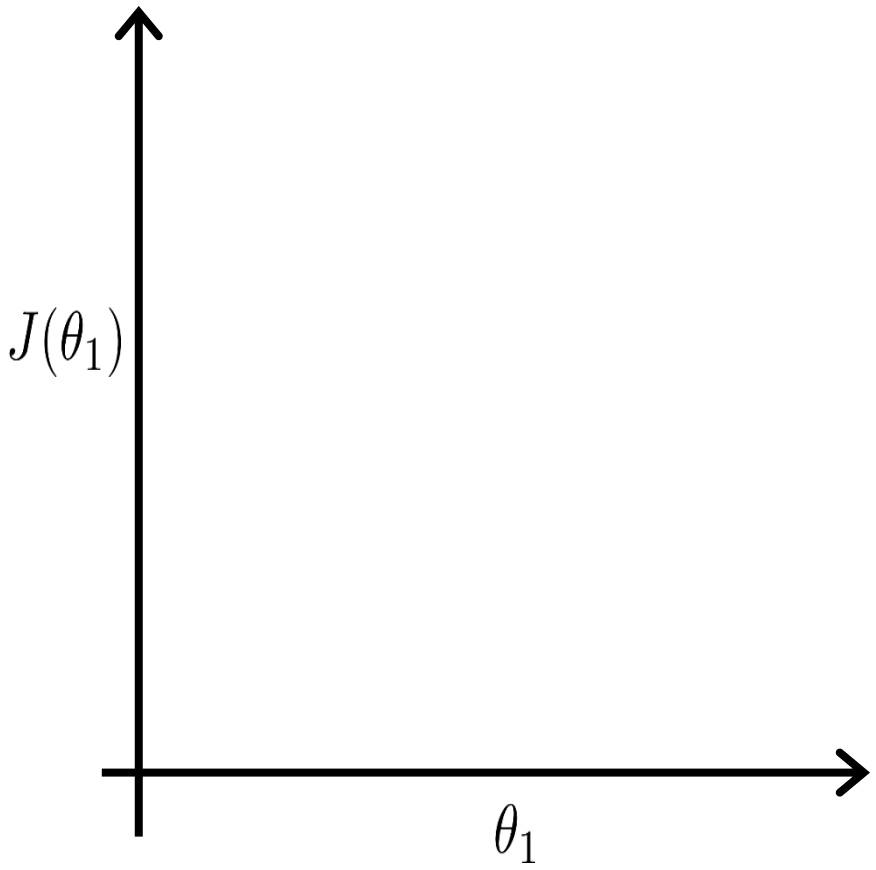


$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

Gradient descent can converge to a local minimum, even with the learning rate α fixed.

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

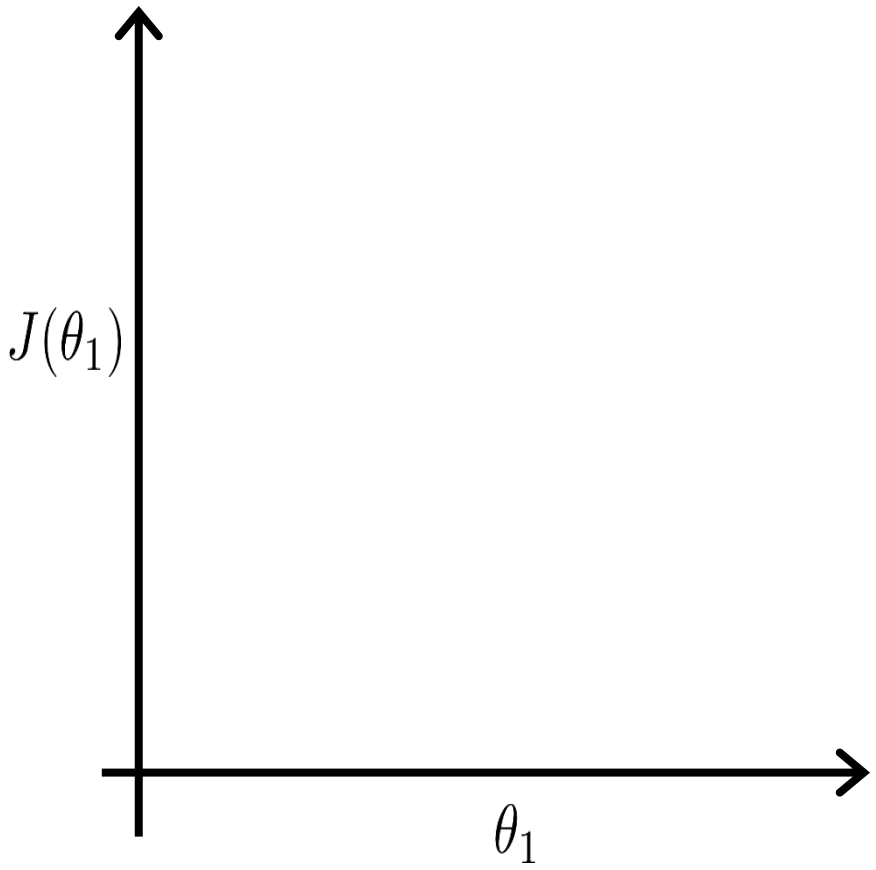
As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease α over time.



Gradient descent can converge to a local minimum, even with the learning rate α fixed.

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease α over time.



Machine Learning

Linear regression with one variable

Gradient descent for linear regression

Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(for $j = 1$ and $j = 0$)

}

Linear Regression Model

$$h_\theta(x) = \theta_0 + \theta_1 x$$

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

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) =$$

$$j = 0 : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) =$$

$$j = 1 : \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) =$$

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) =$$

$$j = 0 : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) =$$

$$j = 1 : \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) =$$

Gradient descent algorithm

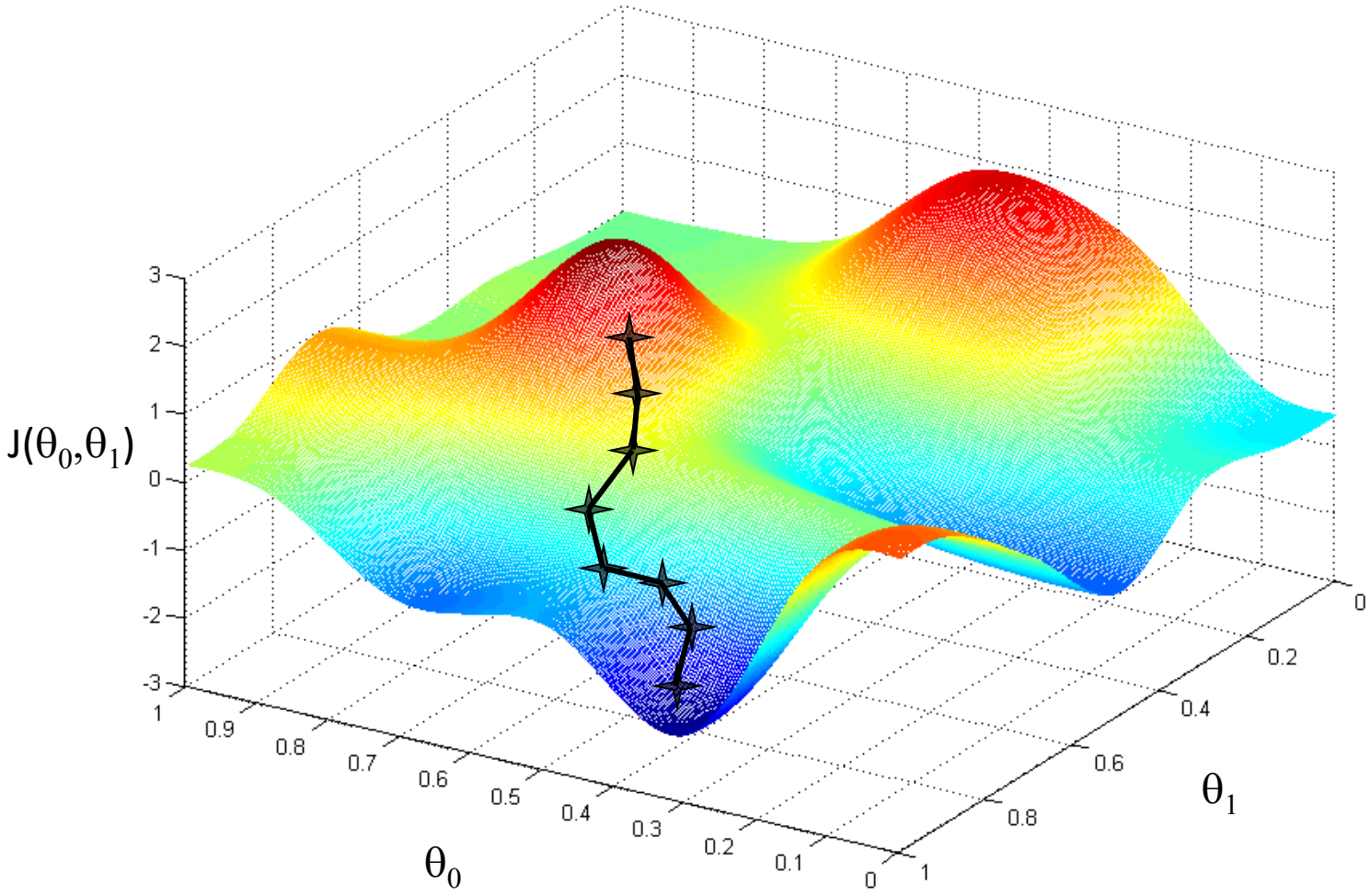
repeat until convergence {

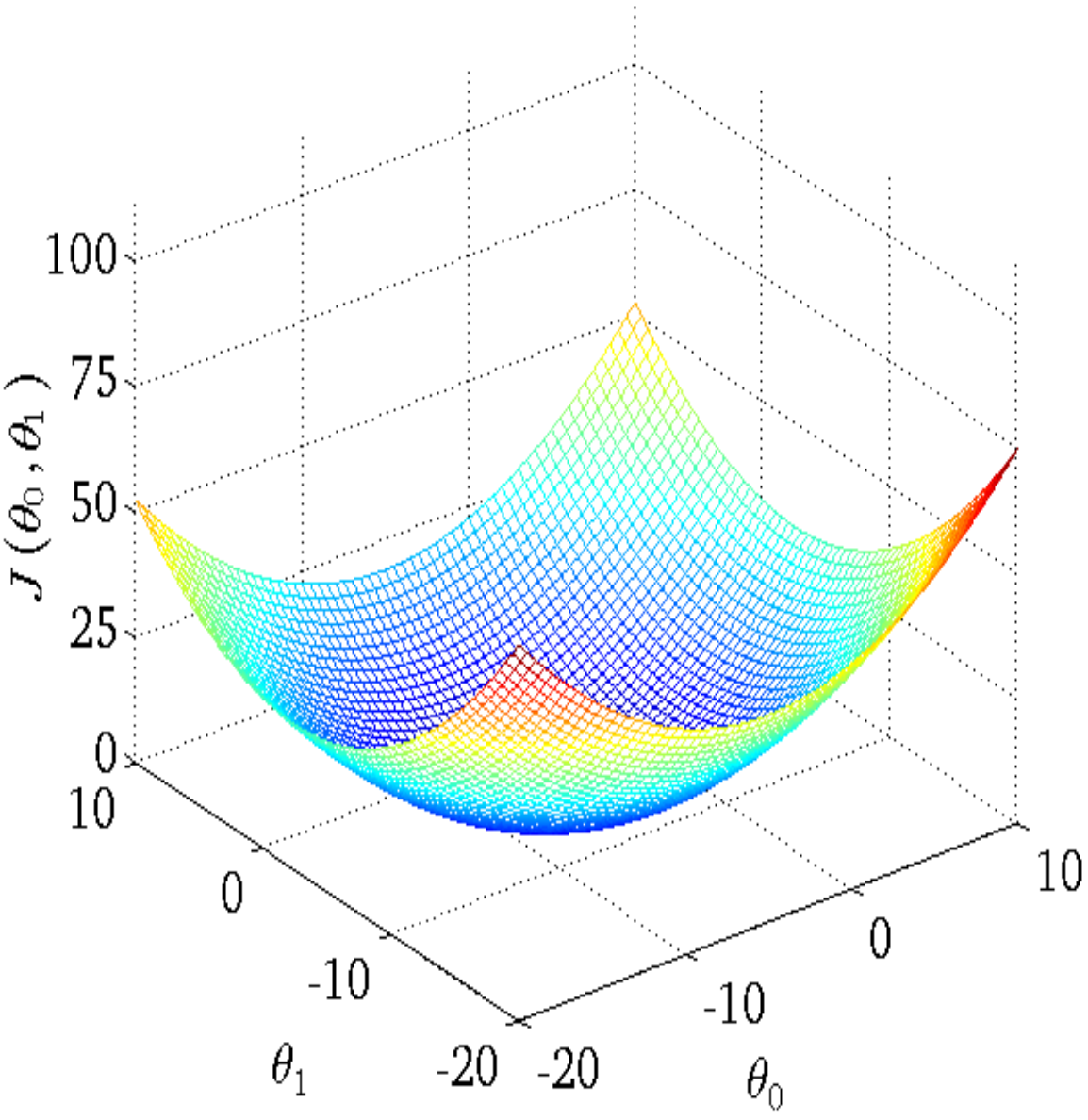
$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

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

}

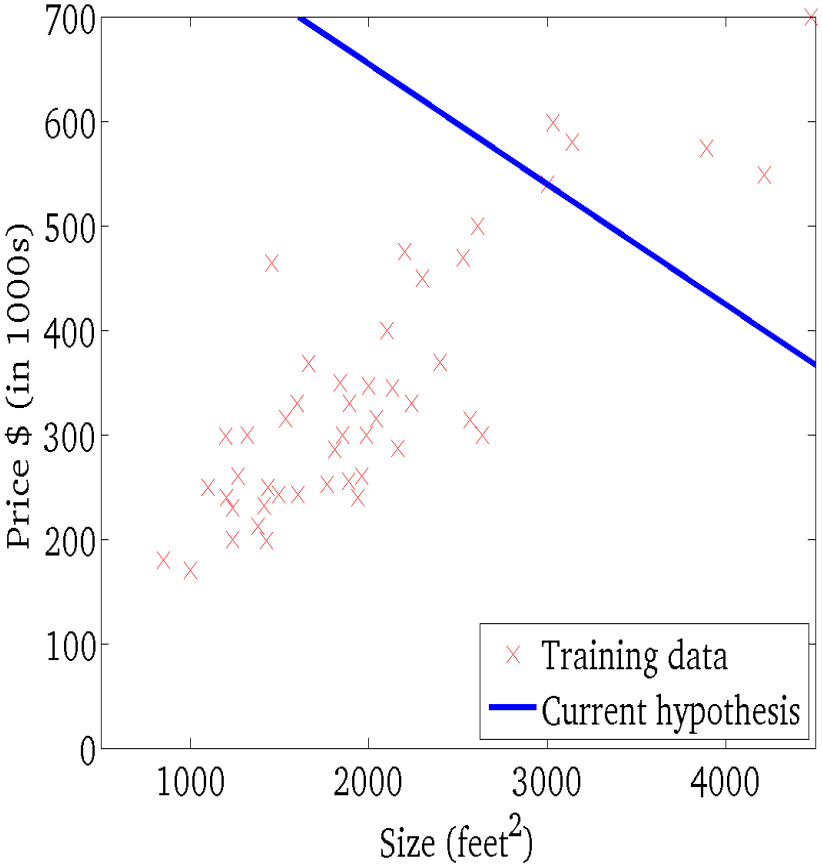
update and θ_0 θ_1 simultaneously


$J(\theta_0,\theta_1)$
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ_0
0
0.2
0.4
0.6
0.8
1
θ_1


100
75
50
25
0
$J(\theta_0, \theta_1)$
10
0
-10
-20
θ_1
-20
-10
0
10
θ_0

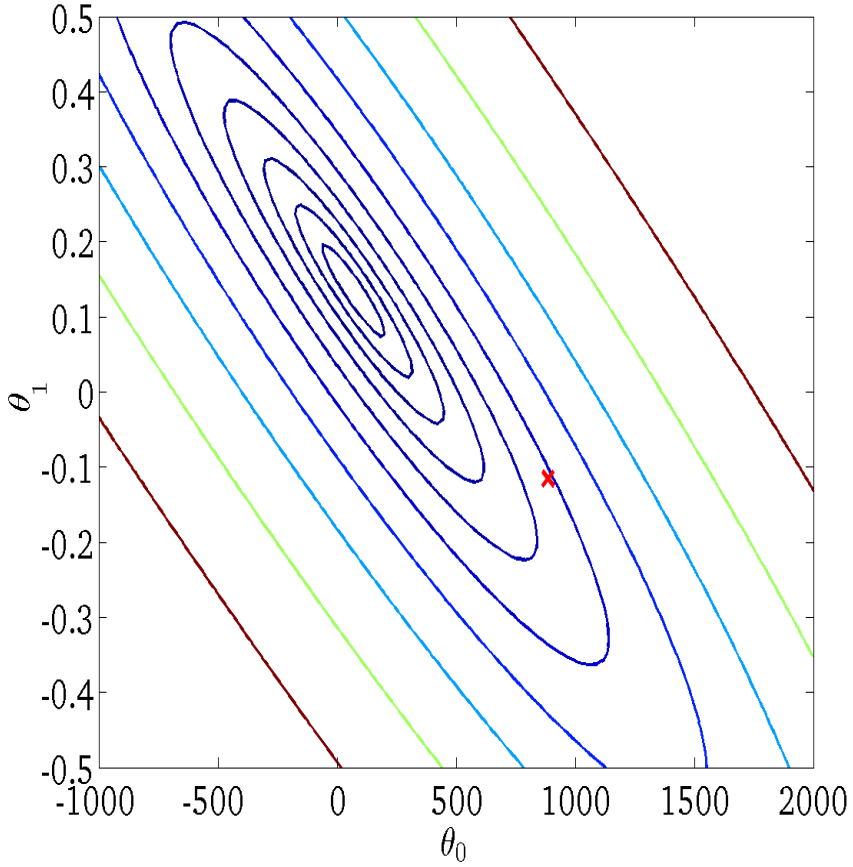
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



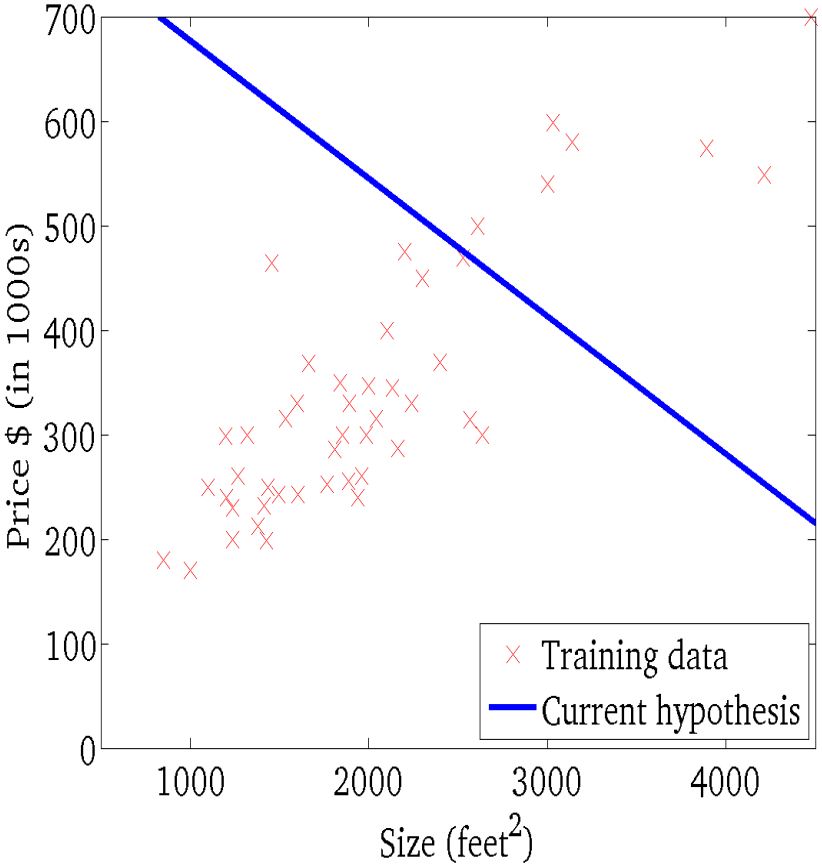
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



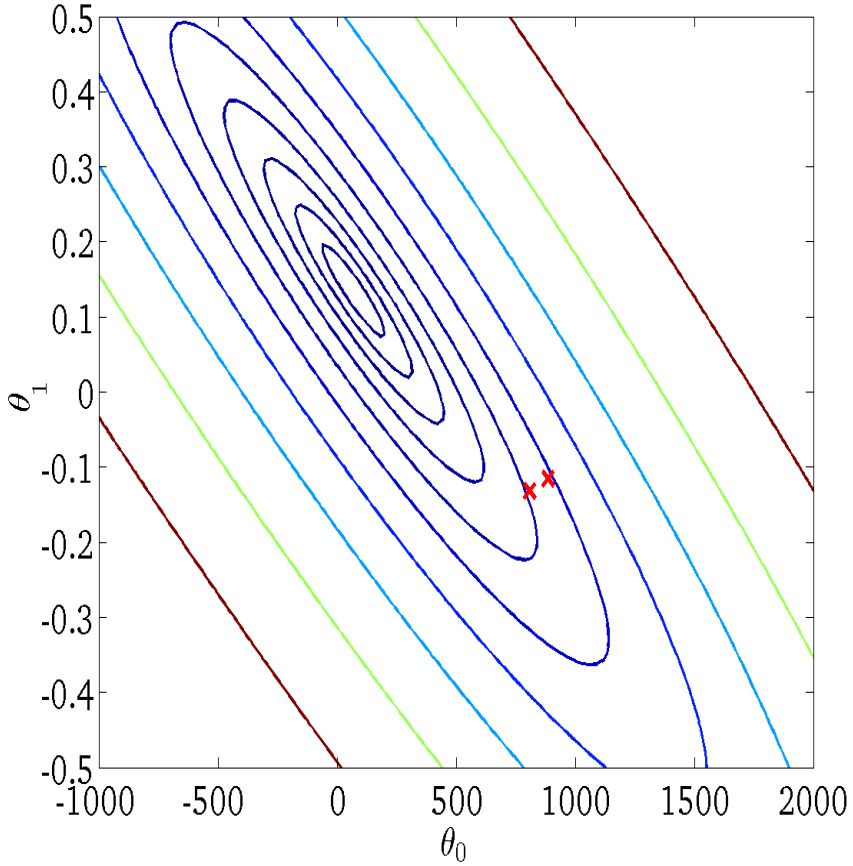
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



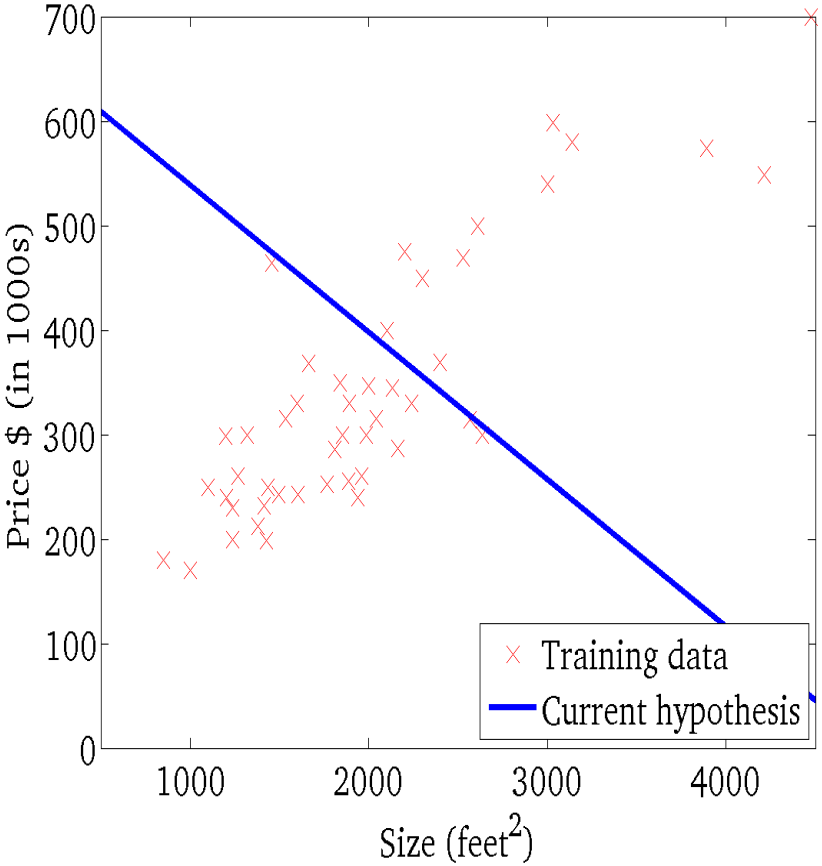
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



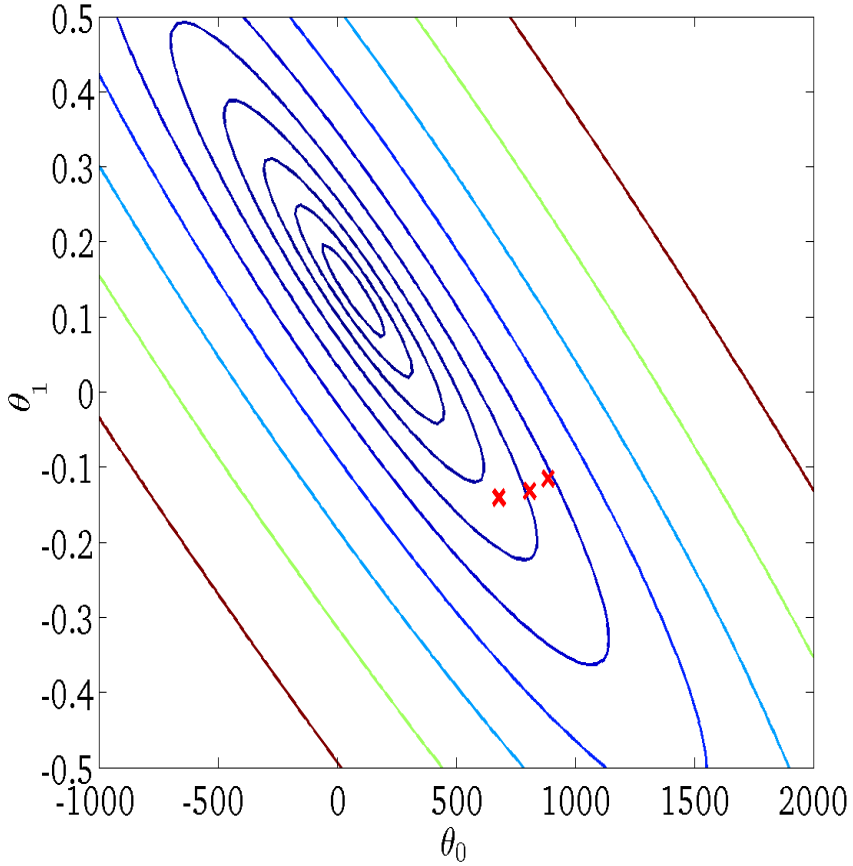
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



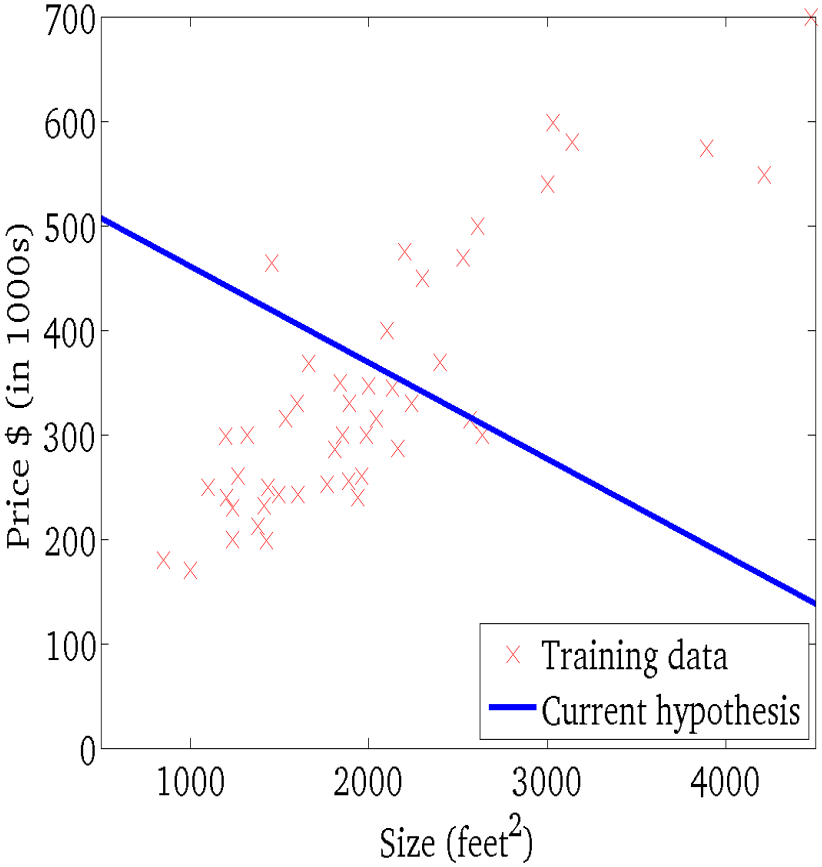
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



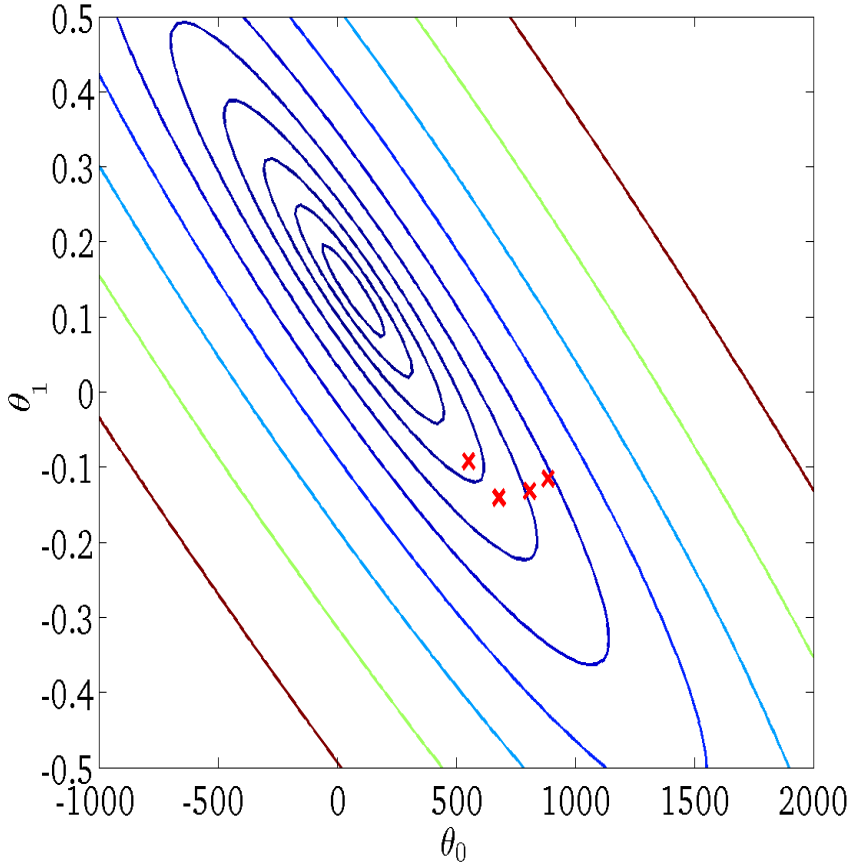
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



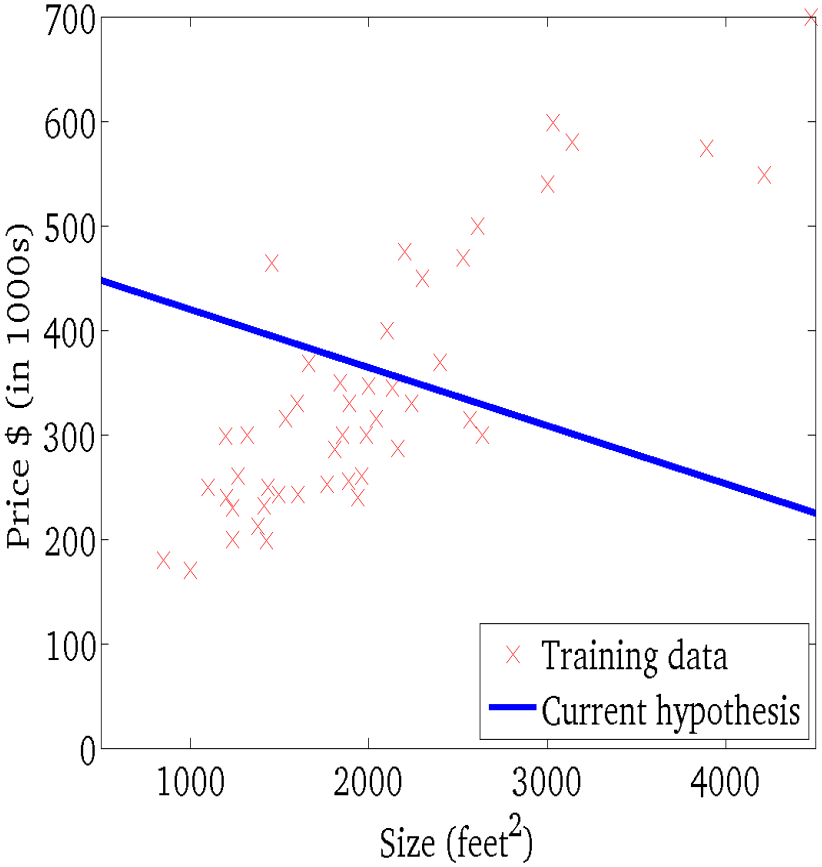
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



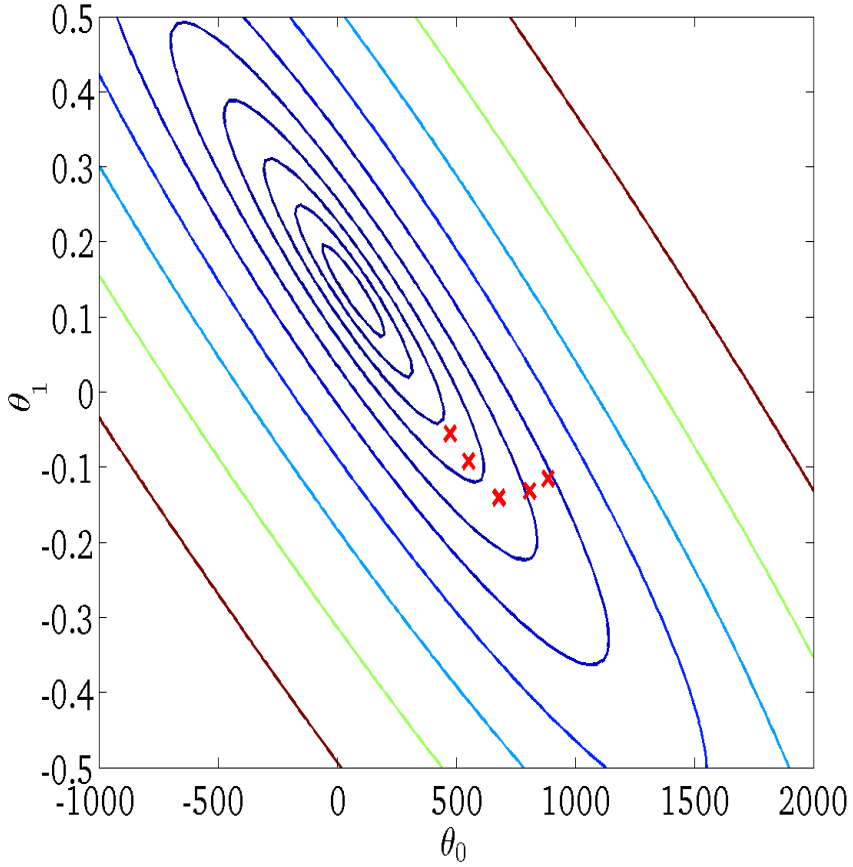
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



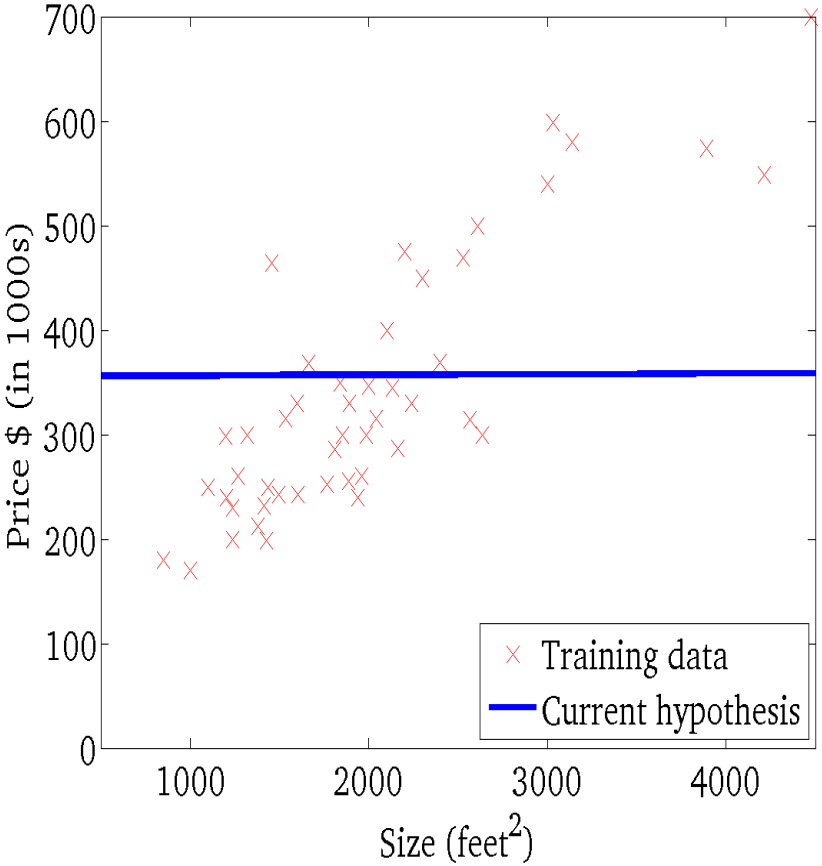
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



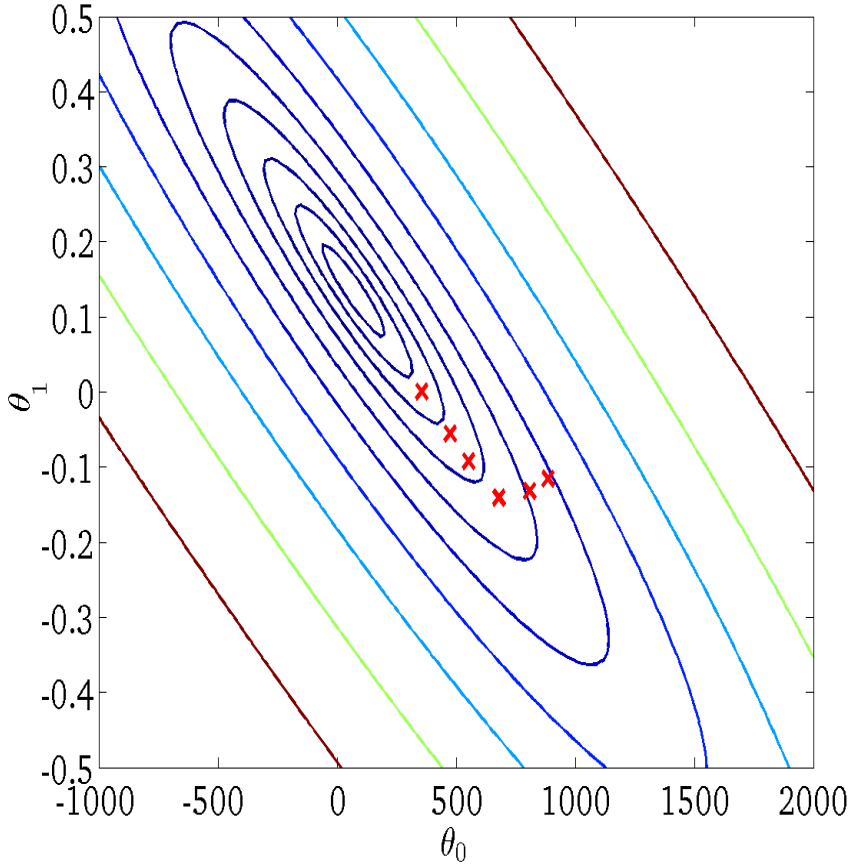
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



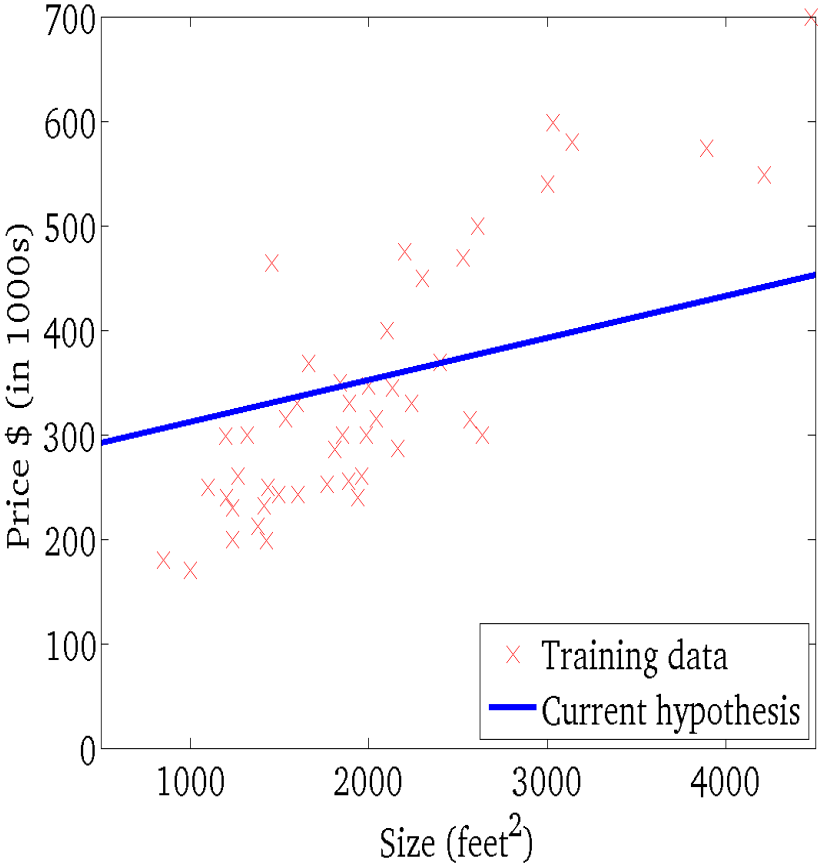
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



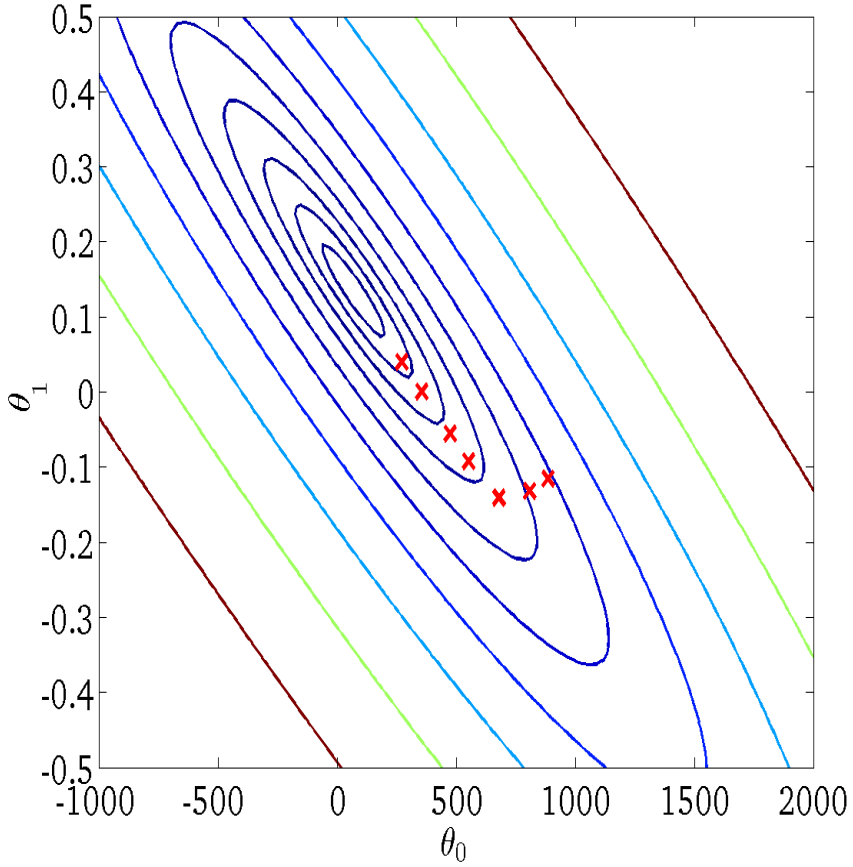
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



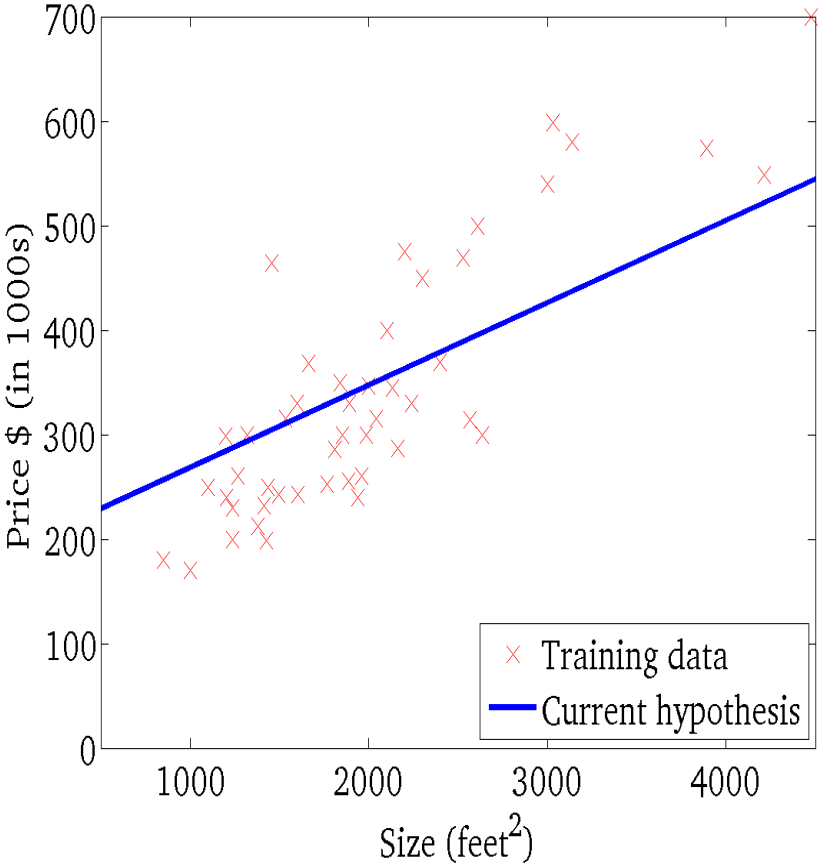
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



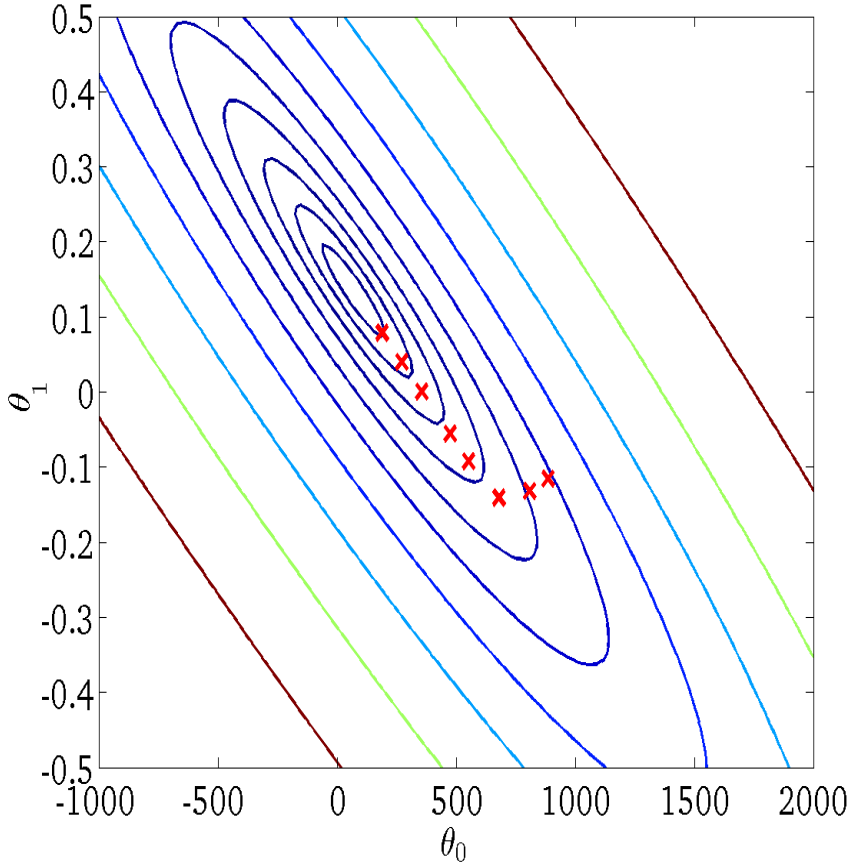
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



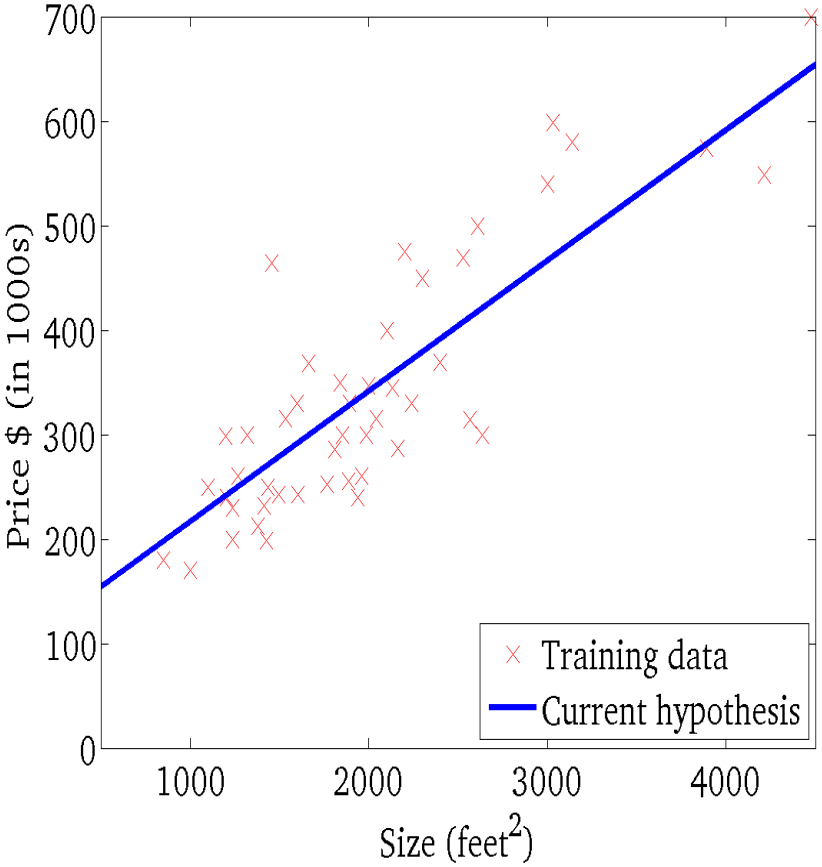
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



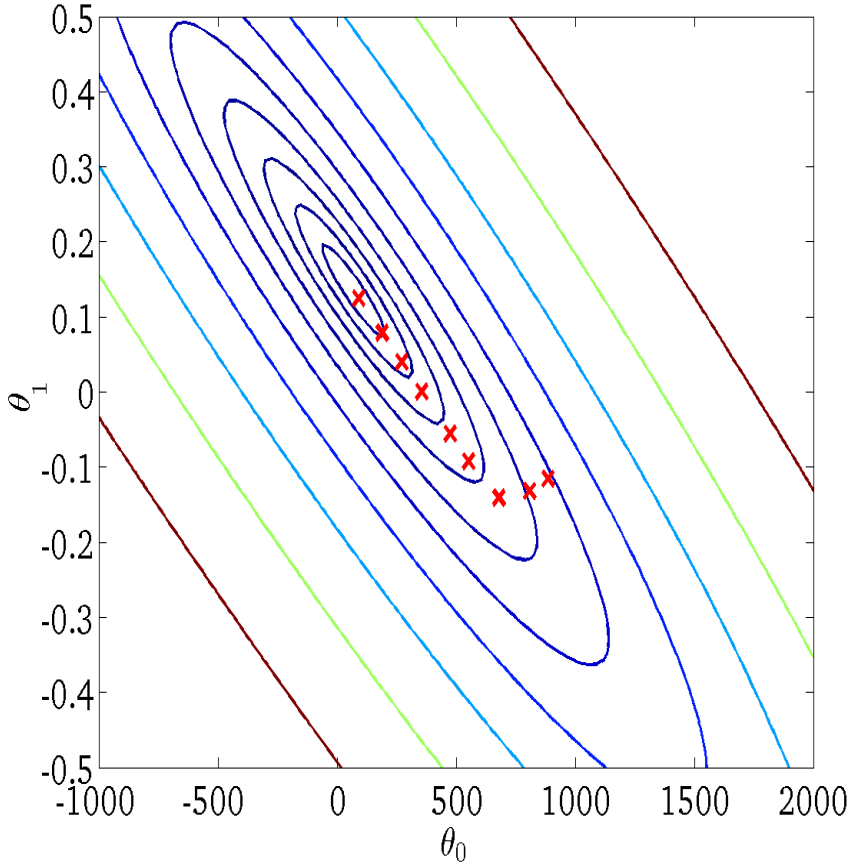
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



“Batch” Gradient Descent

“Batch”: Each step of gradient descent uses all the training examples.

PROGRAMMING ASSIGNMENT

- Write a python code to find the best parameters for univeriate linear regression problem using the Gradient Descent Algorithm.
- You shall run the program with following command
  - Python UNI_LR_GD.py
- The details are given at next slide

INPUT OF THE PROGRAM

- **Enter the Name of Train Data File:** data.csv
  - the data.csv file shall have value for x and y. All the values shall be use to find the parameters
- **Enter the Value of Learning Rate:** 0.9
- **Enter the Name of Test Data: test.csv**
  - The test.csv shall contain the values that are required to predict.

OUTPUT OF THE PROGRAM

Output	Explanation
Predictions.csv	This file shall show the prediction values for each example if test.csv- The first row shall show the header and all other rows show the individual values. Finally, the last row shall print the average error X-Value, Actual-Value, Predicted-Value, Lest Square Error 4,16,14, 4 3,9,12,9 Average Error: 6.5
CostFunction_ Theta0.csv	The file show the value of cost function with respect to different values of theta 0. the fist row of the file contains header information, all other rows contains the different values till convergence, and last row shall print the learning rate value Theta0_Value, CostFunction 0.5,30 Learning Rate: 0.9
CostFunction_ Theta1.csv	Same as above

SUBMISSIONS

- **Program and files**
  - You need to submit the UNI_LR_GD.py file along with data, test file and output files
- **Documentation (A MS Word file that contains)**
  - A graph for different theta0 value during the convergence of gradient descent algorithm and values of cost function on y-axis when alpha is 0.9
  - Another graph same as previous one but theta1 at x-axis and cost function at y-axis when alpha is 0.9
  - Add two more graphs same as above two graphs with alpha 0.5
  - **Discussion**: The affect of learning rate on convergence

LINEAR REGRESSION

ANOTHER WAY TO LOOK AT THE LINEAR REGRESSION

Linear Regression

Linear Regression Model:

$$f(x) = \beta_0 + \sum_{j=1}^{d} \beta_j x_j \quad \text{with} \quad \beta_j \in \mathbb{R}, \quad j \in \{1, \ldots, d\}$$

β 's are called parameters or coefficients or weights.

Learning the linear model $\longrightarrow$ learning the $\beta' s$

Estimation with Least squares:

Use least square loss: $\ell oss(y_i, f(x_i)) = (y_i - f(x_i))^2$

We want to minimize the loss over all examples, that is minimize the *risk or cost function* R:

$$R = \frac{1}{2n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

Linear Regression

A simple case with one feature ($d = 1$):

$$f(x) = \beta_0 + \beta_1 x$$

We want to minimize:

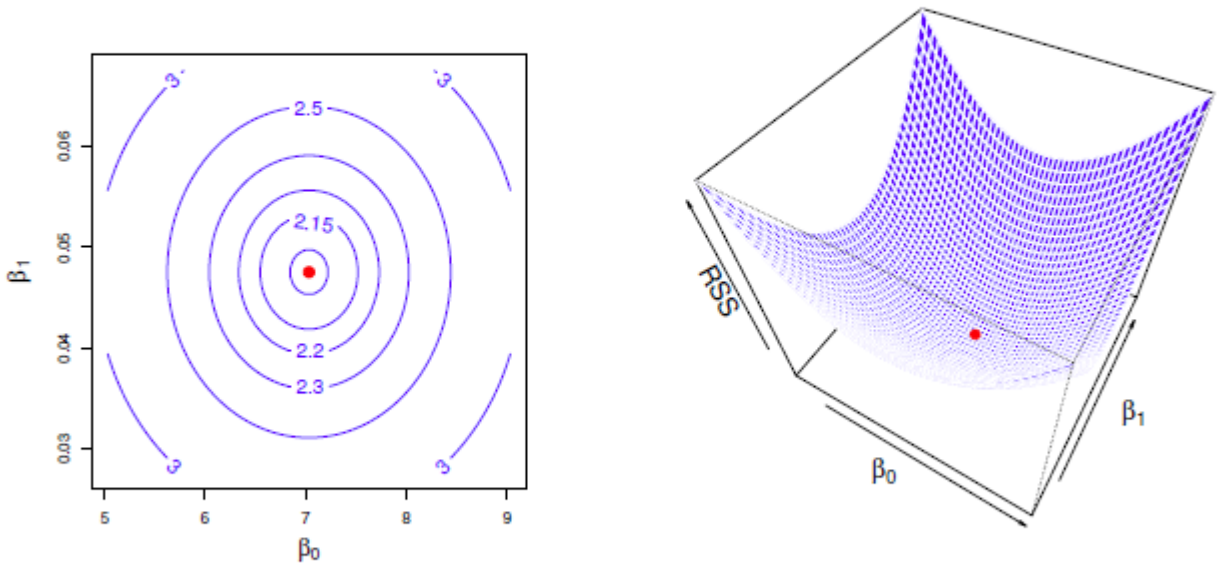
$$R = \frac{1}{2n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

$$R(\beta) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2$$

Find β_0 and β_1 that minimize:

$$R(\beta) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2$$

Linear Regression



Credit: Introduction to Statistical Learning.

Linear Regression

Find β_0 and β_1 so that:

$$argmin_\beta(\frac{1}{2n}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2)$$

Minimize: $R(\beta_0, \beta_1)$, that is: $\frac{\partial R}{\partial \beta_0} = 0 \qquad \frac{\partial R}{\partial \beta_1} = 0$

Find β_0 and β_1 so that:

$$argmin_\beta(\frac{1}{2n}\sum_{i=1}^{n}(y_i-\beta_0-\beta_1 x_i)^2)$$

Minimize: $R(\beta_0,\beta_1)$, that is: $\frac{\partial R}{\partial \beta_0}=0 \qquad \frac{\partial R}{\partial \beta_1}=0$

$$\frac{\partial R}{\partial \beta_0}=2\times\frac{1}{2n}\sum_{i=1}^{n}(y_i-\beta_0-\beta_1 x_i)\times\frac{\partial}{\partial \beta_0}(y_i-\beta_0-\beta_1 x_i)$$

$$\frac{\partial R}{\partial \beta_0}=\frac{1}{n}\sum_{i=1}^{n}(y_i-\beta_0-\beta_1 x_i)\times(-1)=0$$

$$\beta_0=\frac{1}{n}\sum_{i=1}^{n}y_i-\beta_1\frac{1}{n}\sum_{i=1}^{n}x_i$$

Linear Regression

$$\frac{\partial R}{\partial \beta_1} = 2 \times \frac{1}{2n} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i) \times \frac{\partial}{\partial \beta_1} (y_i - \beta_0 - \beta_1 x_i)$$

$$\frac{\partial R}{\partial \beta_1} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i) \times (-x_i) = 0$$

$$\beta_1 \sum_{i=1}^{n} {x_i}^2 = \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} \beta_0 x_i$$

Plugging β_0 in β_1:

$$\beta_1 = \frac{\sum_{i=1}^{n} y_i x_i - \frac{1}{n} \sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i}{\sum_{i=1}^{n} x_i^2 - \frac{1}{n} \sum_{i=1}^{n} x_i \sum x_i}$$

- In simple words these are

```
B1 = Correlation * (Std. Dev. of y/ Std. Dev. of x)

B0 = Mean(Y) - B1 * Mean(X)
```