MACHINE LEARNING

اَللّٰهُمَّ اُرُزُقُنِی عِلُمًا نَافِعًا وَاسِعًا عَمِیُقًا

اَللّٰهُمَّ اُرْزُقْنِى رِزْقًا وَاسِعًا حَلَالًا طَيِّبًا
مُبَارَكًا مِنْ عِنْدِكَ

WEEK 07

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

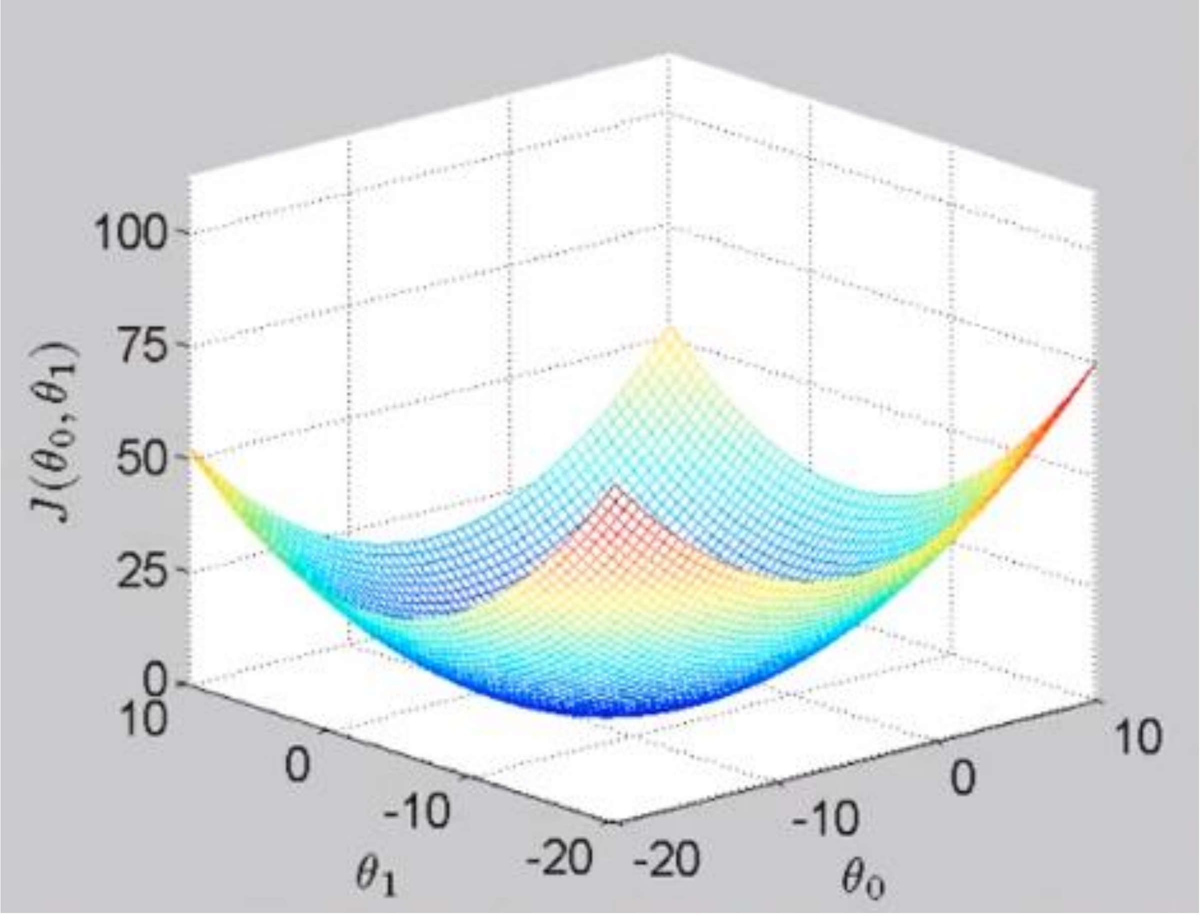
Parameters: θ_0, θ_1

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

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

THE PROBLEM

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



REVIEW

GRADIENT VECTOR

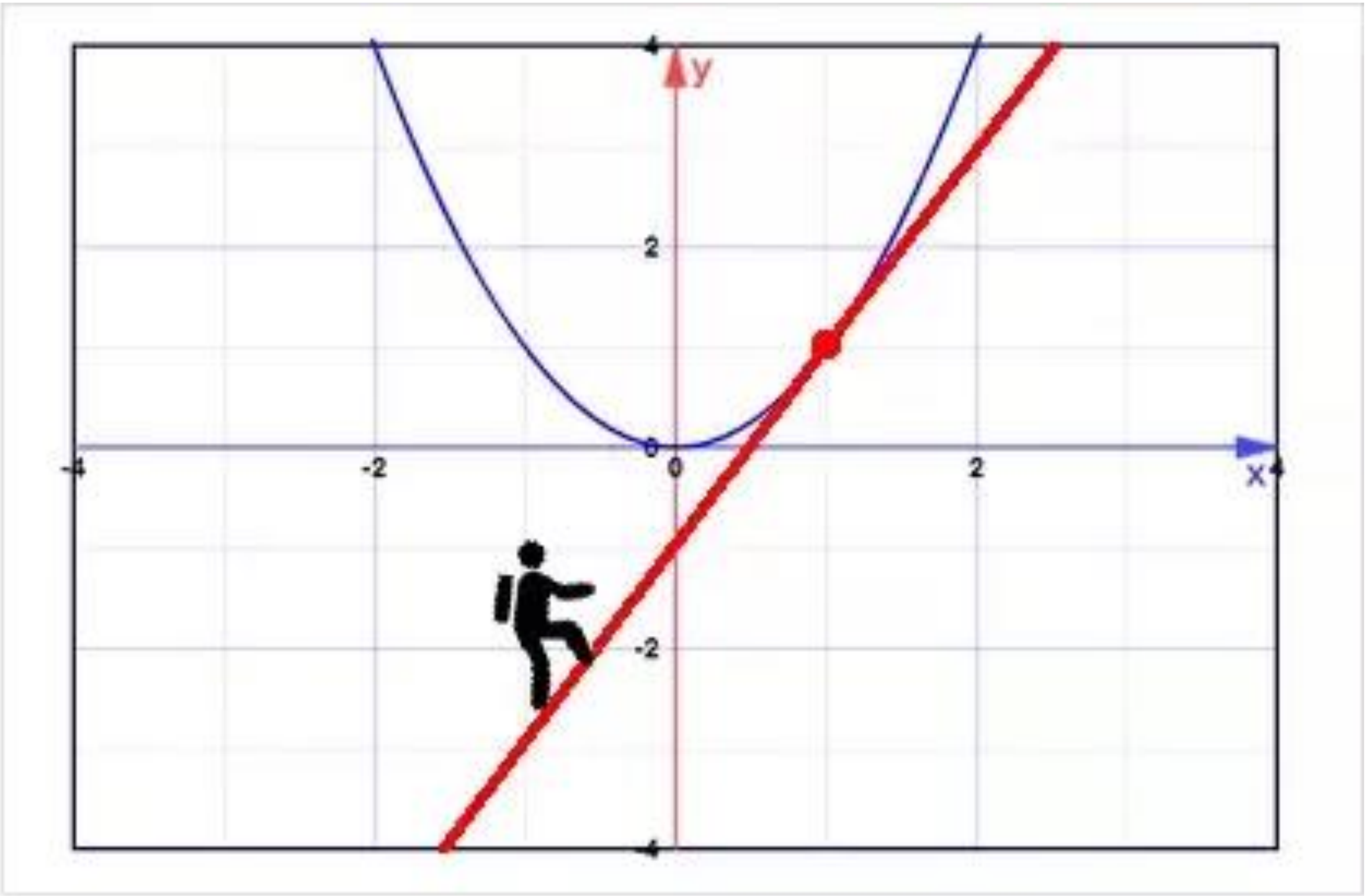
Scalar-valued multivariable function

$$\nabla f(\underbrace{x_0, y_0, \dots}_{\substack{\nabla f \text{ takes the same} \\ \text{type of inputs as } f}}) = \underbrace{\begin{bmatrix} \dfrac{\partial f}{\partial x}(x_0, y_0, \dots) \\[2ex] \dfrac{\partial f}{\partial y}(x_0, y_0, \dots) \\[2ex] \vdots \end{bmatrix}}_{\substack{\nabla f \text{ outputs a vector with} \\ \text{all possible partial derivatives of } f.}}$$

Notation for gradient, called “nabla”.

INTERPRETATION

REVIEW



Linear regression with one variable

Gradient descent

Machine Learning

REVIEW

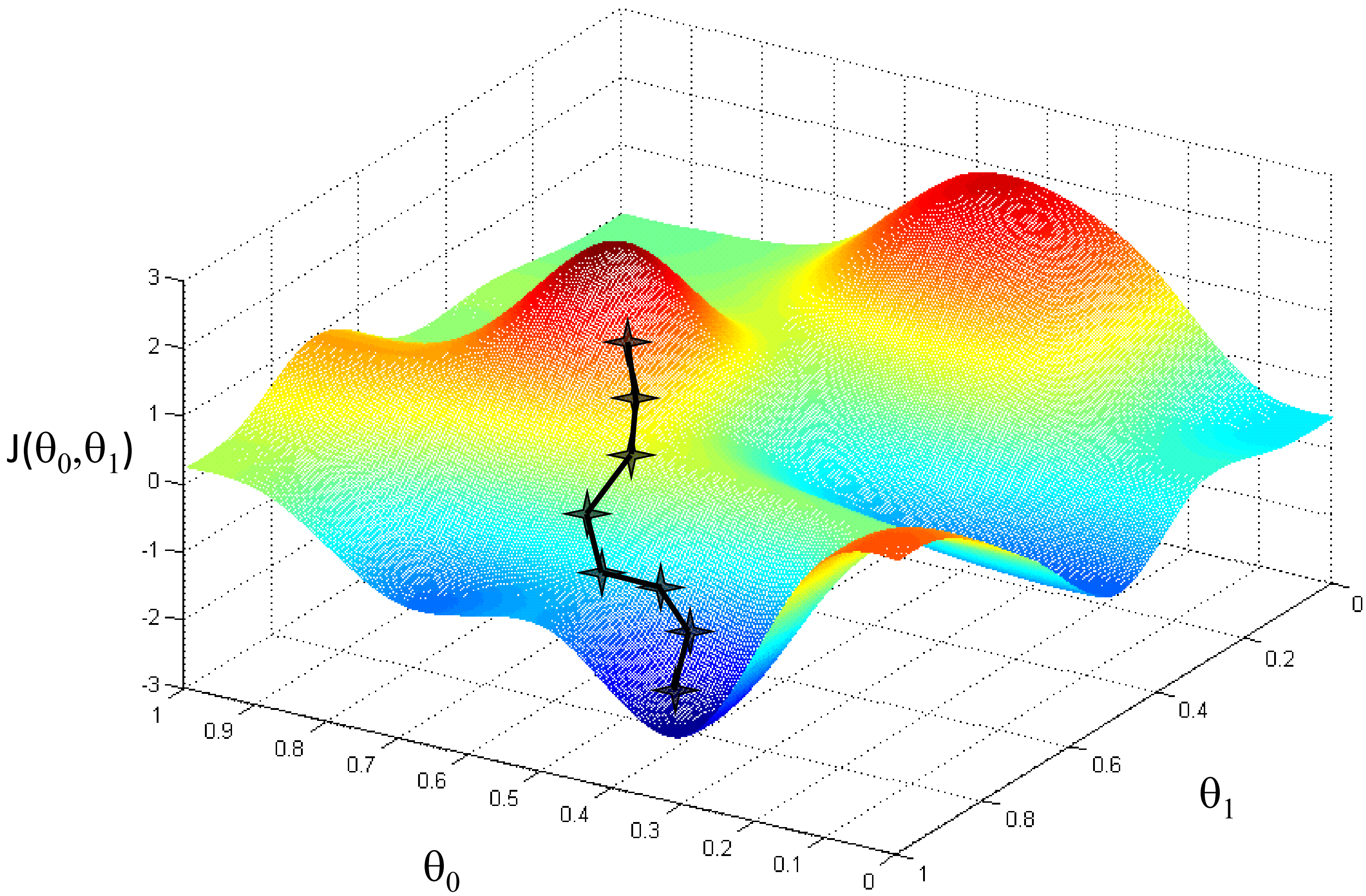
Have some function $J(\theta_0, \theta_1)$

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

REVIEW



$J(\theta_0,\theta_1)$
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

$$
\begin{array}{l}
\text{repeat until convergence } \{ \\
\quad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad (\text{for } j = 0 \text{ and } j = 1) \\
\}
\end{array}
$$

Correct: Simultaneous update

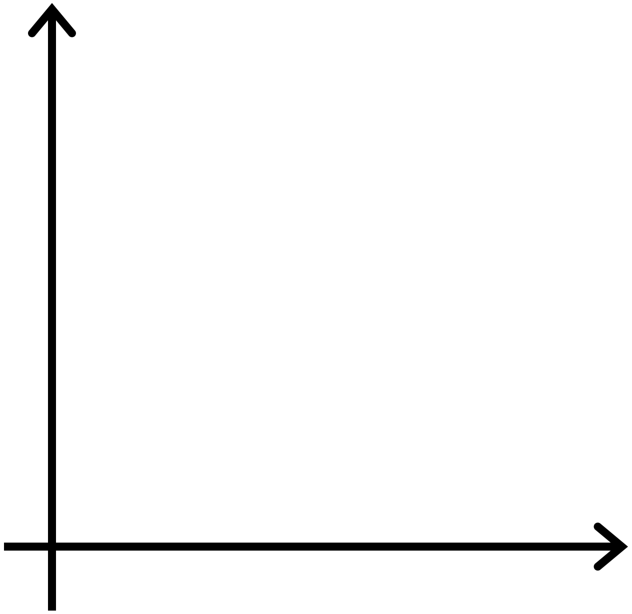
$$
\begin{array}{l}
\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\
\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \\
\theta_0 := \text{temp0} \\
\theta_1 := \text{temp1}
\end{array}
$$

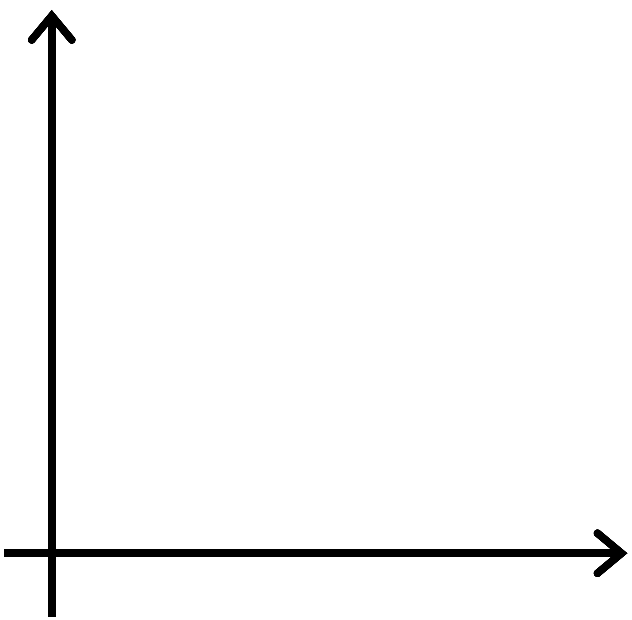
Incorrect:

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

Gradient descent algorithm

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

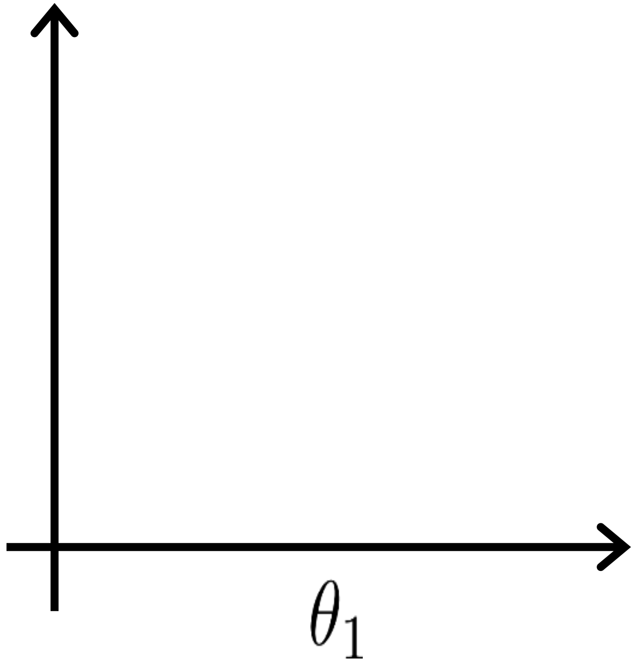




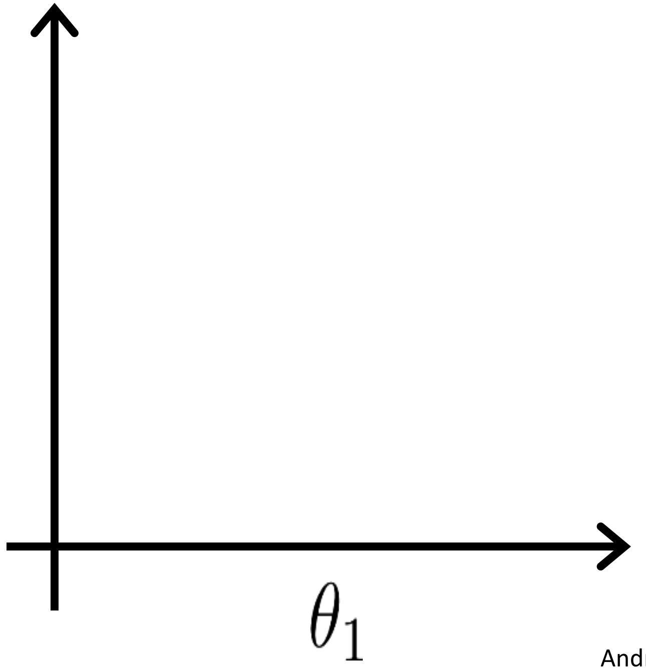
REVIEW

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

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.



Gradient descent algorithm

$$\text{repeat until convergence } \{$$

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

$$(\text{for } j = 1 \text{ and } j = 0)$$

$$\}$$

Linear Regression Model

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

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

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

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

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

Gradient descent algorithm

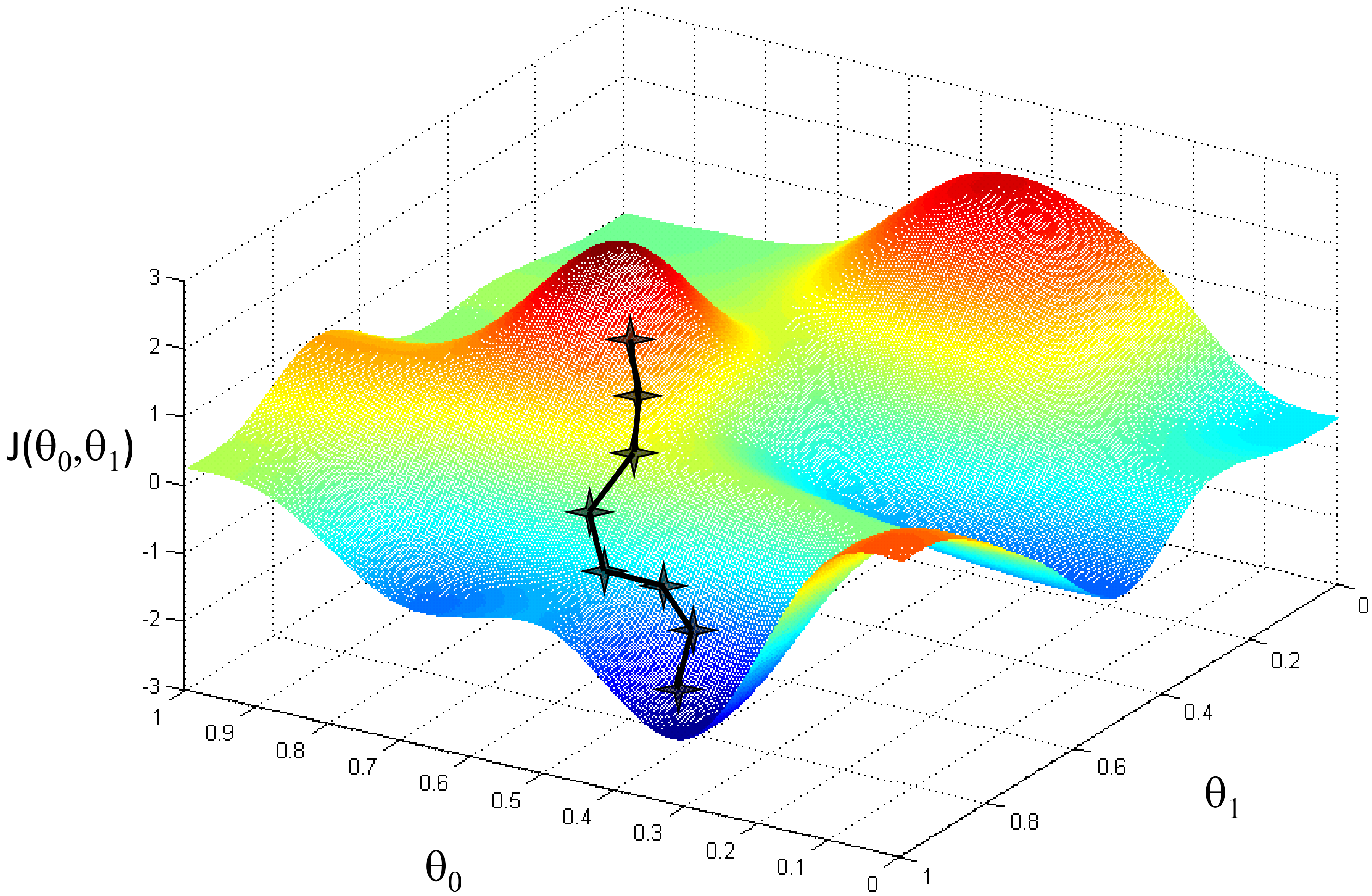
$$\text{repeat until convergence } \{$$

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

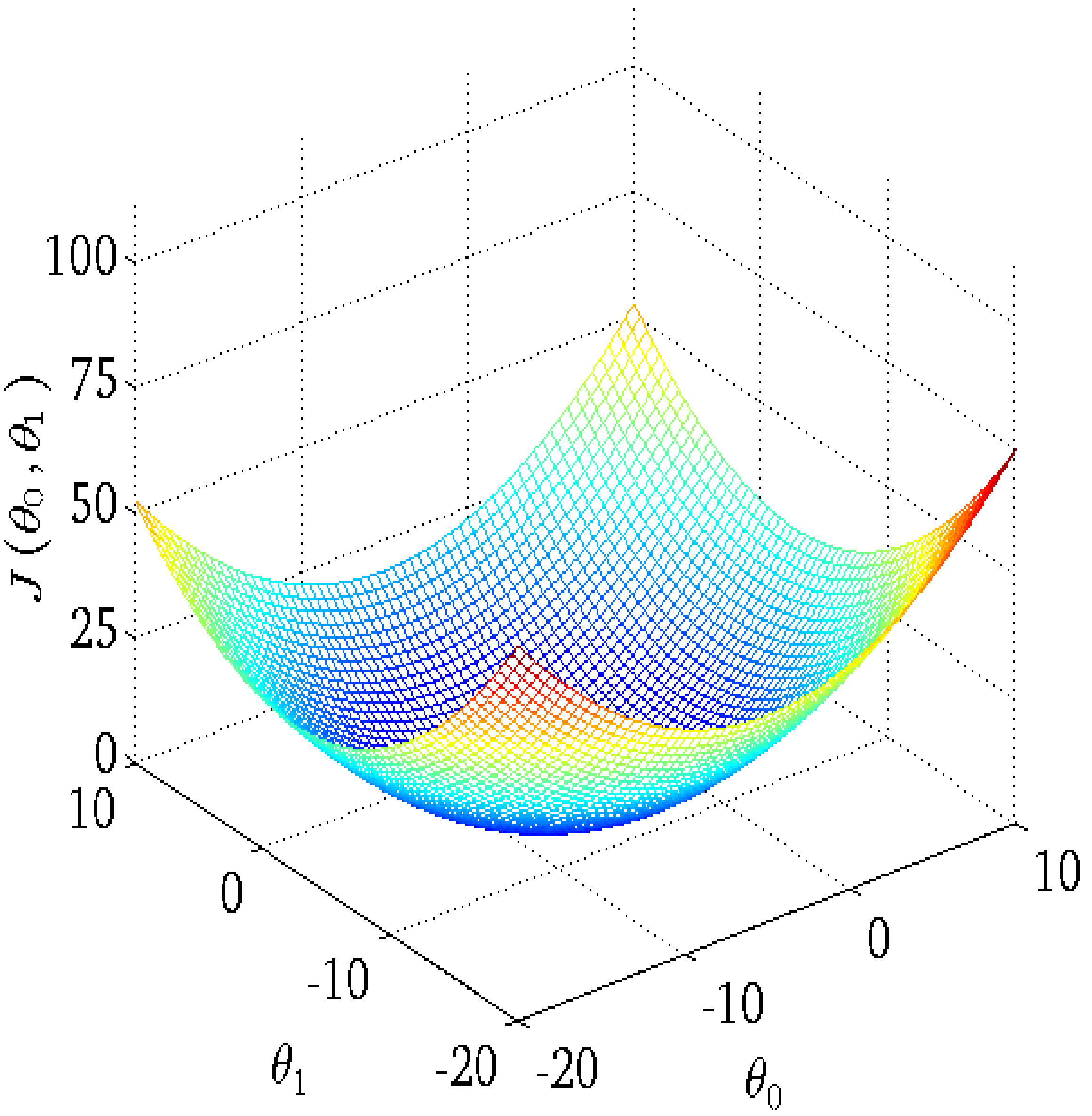
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

$$\}$$

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

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

GRADIENT DESCENT IMPLEMENTATION

MAIN MEHTOD

```
if __name__ == '__main__':
    #Step1. Read Data From File
    data= readData() #abstraction
    #Step2. Initialize Global Variables.
    p=params(t0=20,t1=10,lr=0.00001)
    #Step3 Take 100 Gradient Steps
    gradientDescent(data,p,iterations=300)
    #Display History
    displayHistory()
    #plot cost function with theta0
    plot_thet0()
    #plot cost function with theta1
    plot_thet1()
    #plot surface plot
    showSurfacePlot(data)
```

READDATA

```python
def readData():

    with open('data.csv', newline='') as f:
        reader = csv.reader(f)
        data = list(reader)
        for d in data:
            d[0]=float(d[0])
            d[1]=float(d[1])
        return data
```

PARAM CLASS TO HOLD GLOBAL DATA

```
class params:
    t0=20
    t1=10
    lr=0.00001
    t0_cost=0 #partial derivative
    t1_cost=0 #partial derivative
    total_cost=0 #MSE
    def __init__(self,t0=20,t1=10,lr=0.01):
        self.t0=t0
        self.t1=t1
        self.lr=lr
```

GRADIENT DESCCENT

```
def gradientDescent(data,p,iterations=100):
    for i in range(0,iterations):
        #Saving History of Previous Step
        p.t0_cost=getCostTheta0(data,p)
        p.t1_cost=getCostTheta1(data,p)
        temp0=p.t0 - (p.lr * p.t0_cost)
        temp1=p.t1 - (p.lr * p.t1_cost)
        p.t0=temp0
        p.t1=temp1
        p.total_cost=getTotalCost(data, p)
        SaveHistory(p)
```

PARTIAL DERIVATIVE COST

```python
def getCostTheta0(data,p):
    cost = 0

    for record in data:
        cost = cost+ getLoss(p, record)
    m = len(data)
    cost=cost/m
    return cost


def getCostTheta1(data,p):
    cost = 0
    for record in data:
        cost =cost+ (getLoss(p, record) * record[0])
    m = len(data)
    cost=cost/m
    return cost
```

GET LOSS

```
def getPrediction(x,p):
    pred=p.t0+p.t1*x #hypothesis
    return pred

def getLoss(p, record):
    y_pred = getPrediction(record[0],p)
    y_actual= record[1]
    loss = (y_pred - y_actual) #loss
    return loss
```

TOTAL COST FUNCTION

```python
def getTotalCost(data,p):
    #Mean Square Cost
    cost=0
    for record in data:
        loss = getLoss(p, record)
        loss=pow(loss,2)
        cost=cost+loss
    m=len(data)
    cost = 1/2*1/m*(cost)
    return cost
```

SAVED VALUES

```
def SaveHistory(p):
    log_theta0.append(p.t0)
    log_theta1.append(p.t1)
    log_theta0_cost.append(p.t0_cost)
    log_theta1_cost.append(p.t1_cost)
    log_total_cost.append(p.total_cost)
    log_lr.append(p.lr)

def displayHistory():
  top="idx.\tTheta1\tTheta2\tLR\tCostT0\tCostT1\t\tTotal_Cost"
  print(top)
  for i in range(0,len(log_theta0)):
    msg="{0}.\t{1:.2f}\t{2:.2f}\t{3}\t{4:.2f}\t{5:.2f}\t\t{6:.
   2f}"
        print(msg.format(i+1,log_theta0[i],log_theta1[i],log_lr
[i],log
   _theta0_cost[i],log_theta1_cost[i],log_total_cost[i]))
```

PLOTTING GRAPH

```
def plot_thet0():
    plt.plot(log_theta0,log_total_cost)
    plt.gca().invert_xaxis()
    plt.show()
def plot_thet1():
    plt.plot(log_theta1,log_total_cost)
    plt.gca().invert_xaxis()
    plt.show()
```

SHOW SURFACE PLOT

```
def showSurfacePlot(data):
    x,y,J_vals=prepareSurfaceData(data)
    X, Y = np.meshgrid(x, y)
    fig = plt.figure()
    ax = plt.axes(projection="3d")
    ax.scatter(log_theta0,log_theta1,log_total_cost,marker="+")
    ax.plot_surface(X, Y, J_vals, cmap="plasma")
    ax.set_xlabel('Theta0')
    ax.set_ylabel('Theta1')
    ax.set_zlabel('Cost')
    plt.title("3D plot of COST function for different theta");
    plt.show()
```

PREPARE SURFACE DATA

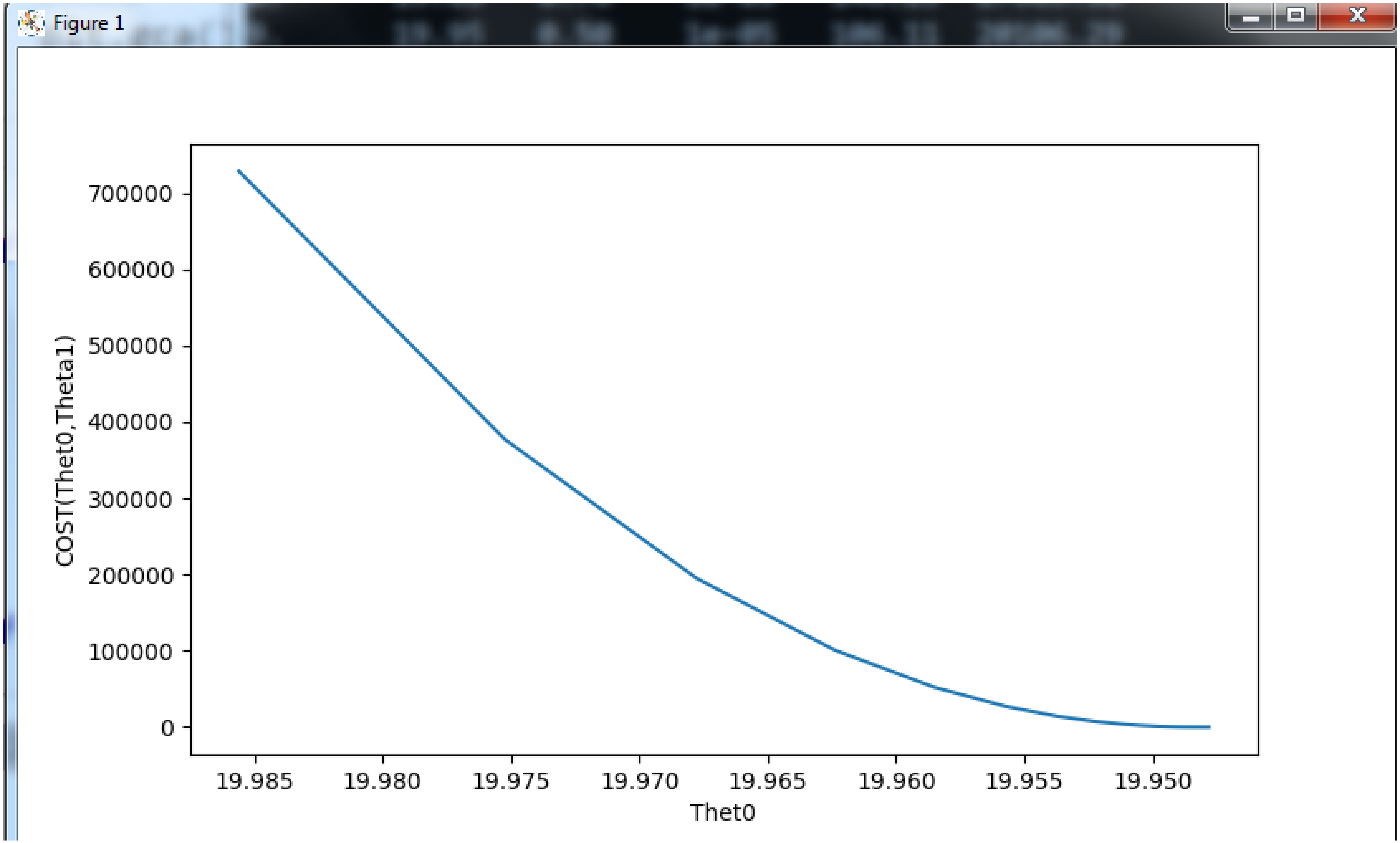
```
def prepareSurfaceData(data):
    x= np.arange(-20, 20,1)
    y= np.arange(-20, 20,1)
    J_vals = np.zeros((len(x), len(y)))
    p=params()
    c1=0;c2=0
    for i in x:
        for j in y:
            p.t0=i
            p.t1=j
            J_vals[c1][c2] = int(getTotalCost(data, p))
            c2=c2+1
        c1=c1+1
        c2=0 # reinitialize to 0
    return x,y,J_vals
```

RESULTS OF CONVERGENCE

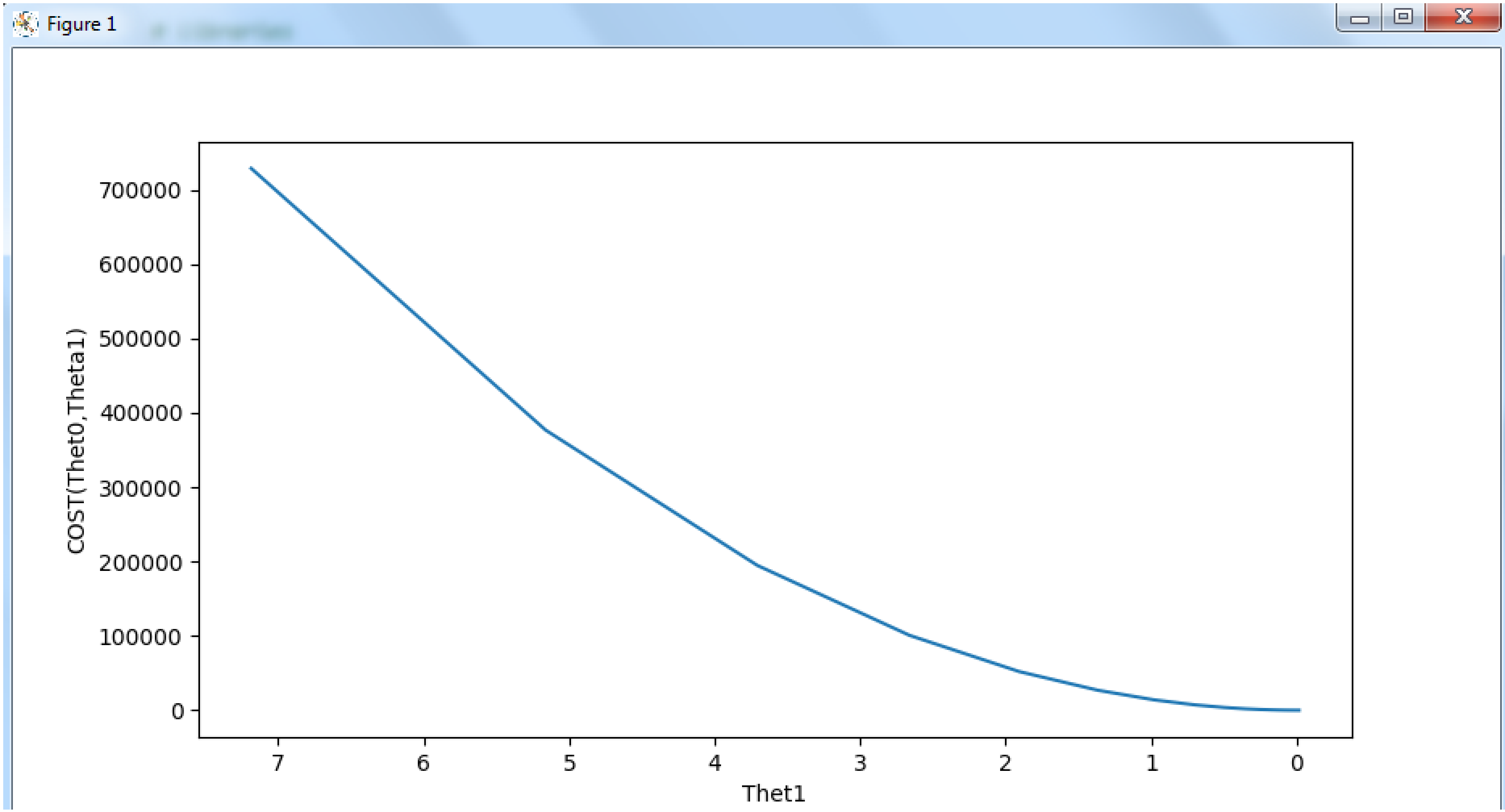
Command Prompt - python URLGD.py

idx.	Theta1	Theta2	LR	CostT0	CostT1	Total_Cost
1.	19.99	7.19	1e-05	1440.69	281453.83	728814.10
2.	19.98	5.16	1e-05	1036.85	202370.78	376801.83
3.	19.97	3.71	1e-05	746.48	145508.52	194815.36
4.	19.96	2.66	1e-05	537.70	104623.46	100730.36
5.	19.96	1.91	1e-05	387.58	75226.30	52089.45
6.	19.96	1.37	1e-05	279.64	54089.17	26942.63
7.	19.95	0.98	1e-05	202.03	38891.16	13942.01
8.	19.95	0.70	1e-05	146.23	27963.50	7220.83
9.	19.95	0.50	1e-05	106.11	20106.29	3746.05
10.	19.95	0.35	1e-05	77.26	14456.81	1949.63
11.	19.95	0.25	1e-05	56.51	10394.72	1020.90
12.	19.95	0.17	1e-05	41.60	7474.00	540.76
13.	19.95	0.12	1e-05	30.87	5373.95	292.53
14.	19.95	0.08	1e-05	23.16	3863.97	164.20
15.	19.95	0.05	1e-05	17.62	2778.26	97.85
16.	19.95	0.03	1e-05	13.63	1997.62	63.55
17.	19.95	0.02	1e-05	10.77	1436.32	45.82
18.	19.95	0.01	1e-05	8.70	1032.74	36.65
19.	19.95	0.00	1e-05	7.22	742.55	31.91
20.	19.95	-0.00	1e-05	6.16	533.91	29.46
21.	19.95	-0.01	1e-05	5.39	383.88	28.19
22.	19.95	-0.01	1e-05	4.84	276.02	27.54
23.	19.95	-0.01	1e-05	4.44	198.46	27.20
24.	19.95	-0.01	1e-05	4.16	142.69	27.02
25.	19.95	-0.01	1e-05	3.95	102.59	26.93

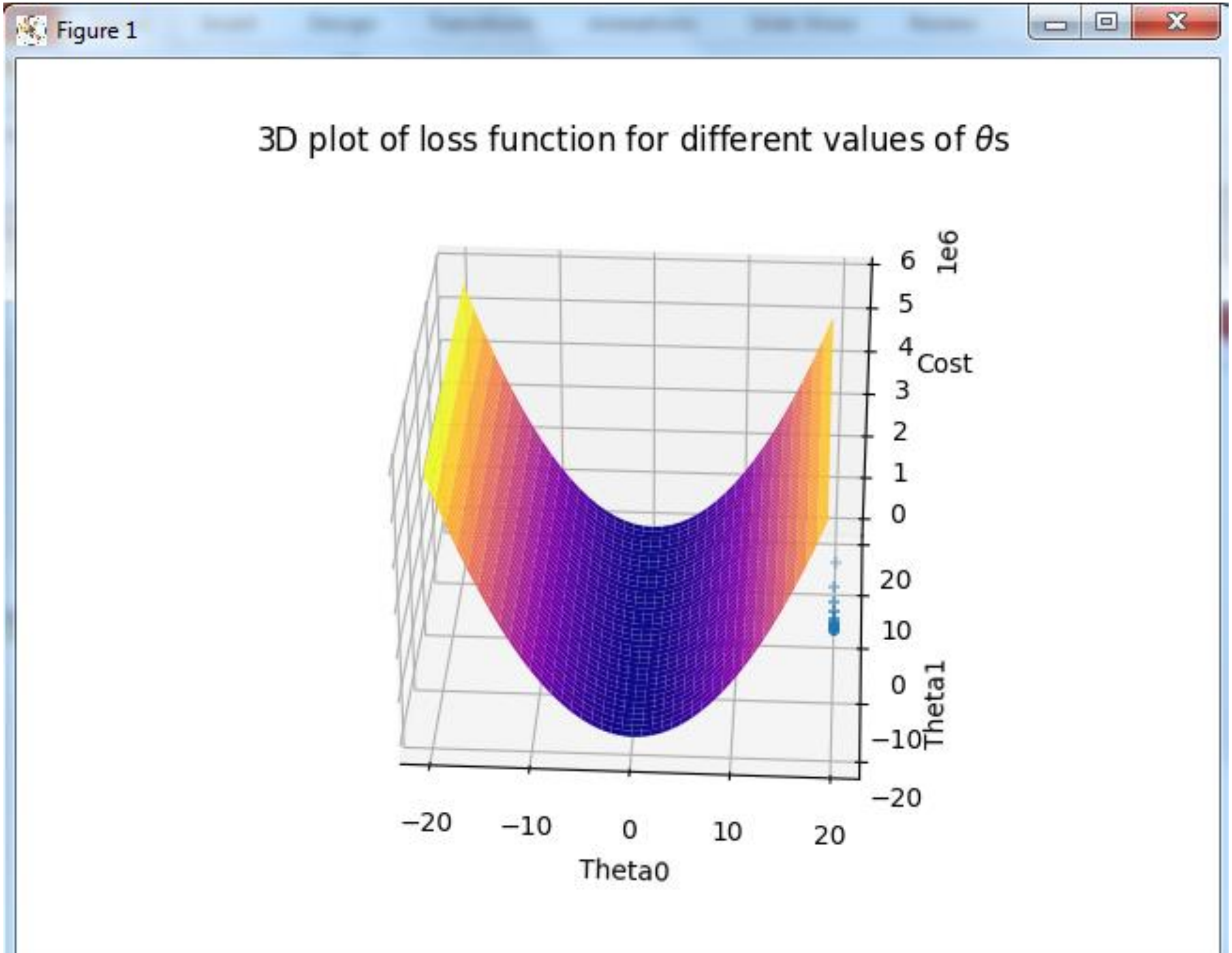
CONVERGENCE OF THETA0



CONVERGENCE OF THETA1



SURFACE GRAPH



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.

$$\boxed{\text{Learning the linear model} \longrightarrow \text{learning the } \beta's}$$

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:

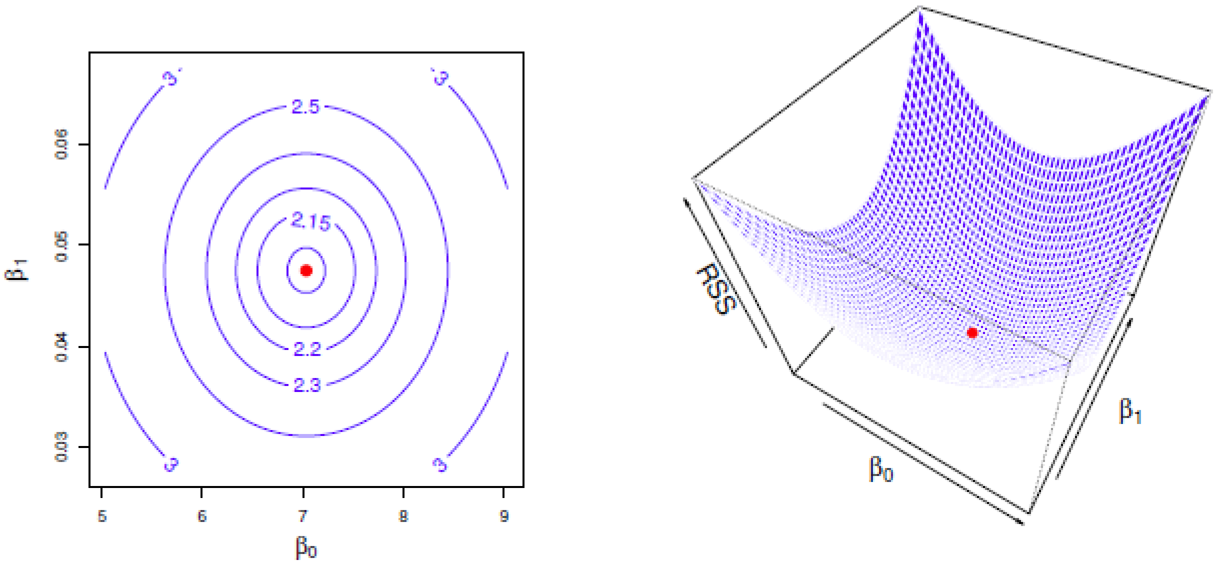
$$R = \frac{1}{2n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

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

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)
```

MULTIVARIATE LINEAR REGRESSION

LINEAR REGRESSION IN SINGLE VARIABLE

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

With more than one feature:

$$f(x) = \beta_0 + \sum_{j=1}^{d} \beta_j x_j$$

Find the β_j that minimize:

$$R = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{d} \beta_j x_{ij}))^2$$

Let's write it more elegantly with matrices!

Multiple features (variables).

Size (feet²) x	Price ($1000) y
2104	460
1416	232
1534	315
852	178
…	…

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

Multiple features (variables).

Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

Notation:

n = number of features

$x^{(i)}$ = input (features) of i^{th} training example.

$x_j^{(i)}$ = value of feature j in i^{th} training example.

Previous Hypothesis:

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

Hypothesis:

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

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

For convenience of notation, define $x_0 = 1$.

Multivariate linear regression.

Linear Regression with multiple variables

GRADIENT DESCENT FOR MULTIPLE VARIABLES

Machine Learning

Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

Parameters: $\theta_0, \theta_1, \ldots, \theta_n$

Cost function:

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

Gradient descent:

Repeat $\{$

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

$\}$ (simultaneously update for every $j = 0, \ldots, n$)

Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

Parameters: $\theta_0, \theta_1, \ldots, \theta_n$

Cost function:

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

Gradient descent:

Repeat $\{$

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

$\}$ (simultaneously update for every $j = 0, \ldots, n$)

Gradient Descent

Previously (n=1):

Repeat $\{$

$$\theta_0 := \theta_0 - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})}_{\frac{\partial}{\partial \theta_0} J(\theta)}$$

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

(simultaneously update θ_0, θ_1)

$\}$

New algorithm $(n \geq 1)$:

Repeat $\{$

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

(simultaneously update θ_j for $j = 0, \ldots, n$)

$\}$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

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

$$\theta_2 := \theta_2 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_2^{(i)}$$

$\ldots$

Gradient Descent

Previously (n=1):

Repeat $\{$

$$\theta_0 := \theta_0 - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})}_{\frac{\partial}{\partial\theta_0}J(\theta)}$$

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

(simultaneously update θ_0, θ_1)

$\}$

New algorithm $(n \geq 1)$:

Repeat $\{$

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

(simultaneously update θ_j for $j = 0, \ldots, n$)

$\}$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

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

$$\theta_2 := \theta_2 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_2^{(i)}$$

$\ldots$

Linear Regression with multiple variables

GRADIENT DESCENT IN PRACTICE I: FEATURE SCALING

Machine Learning

Feature Scaling

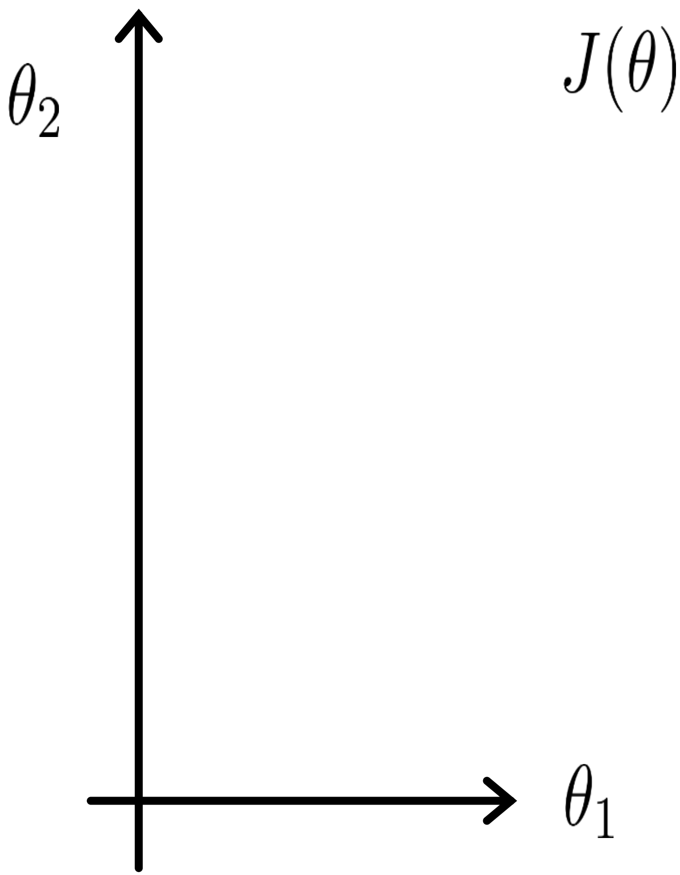
Idea: Make sure features are on a similar scale.

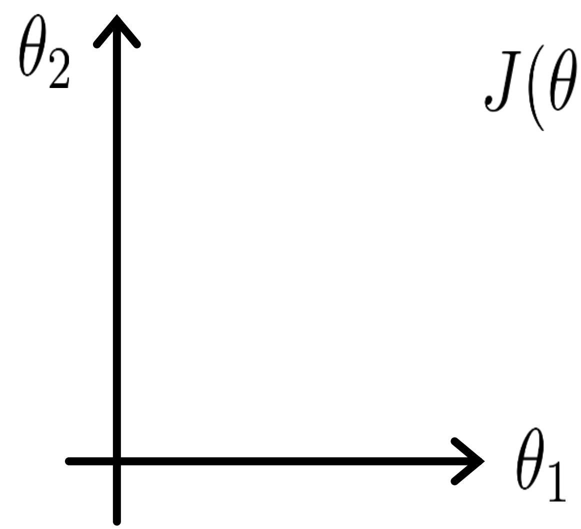
E.g. x_1 = size (0-2000 feet^2)

x_2 = number of bedrooms (1-5)

$$x_1 = \frac{\text{size (feet}^2\text{)}}{2000}$$

$$x_2 = \frac{\text{number of bedrooms}}{5}$$



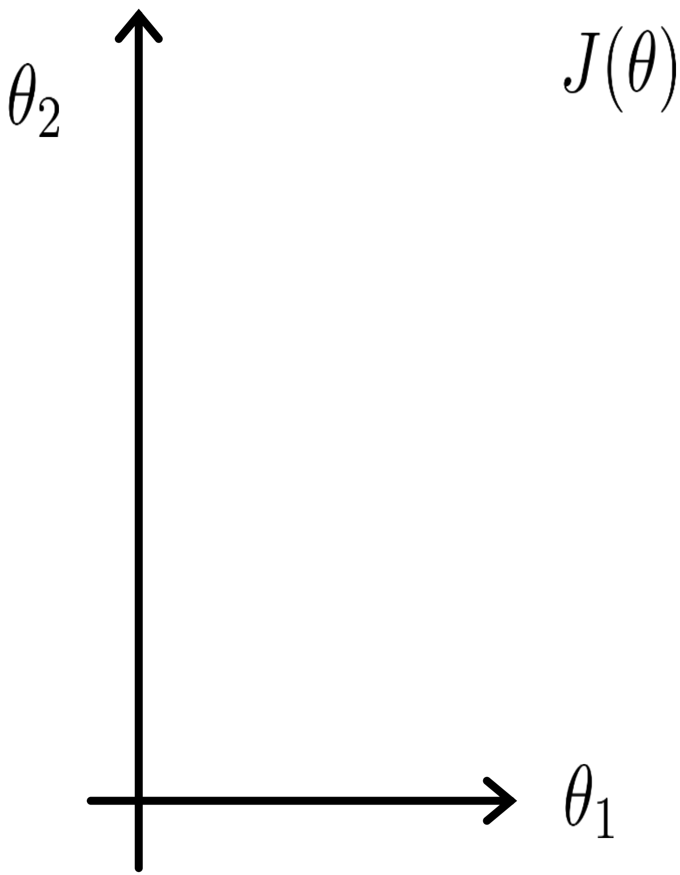


Feature Scaling

Idea: Make sure features are on a similar scale.

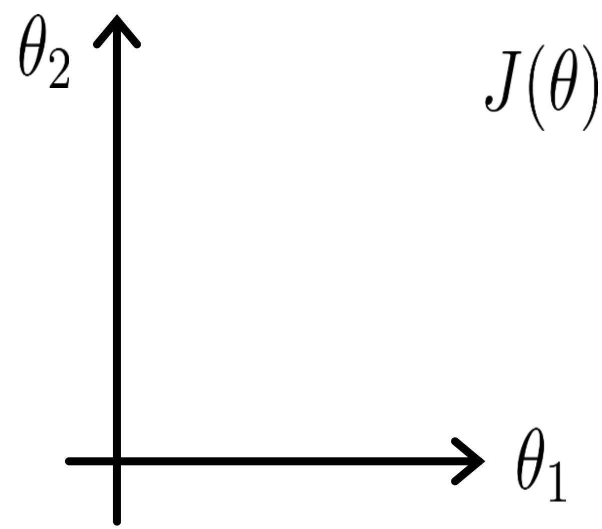
E.g. x_1 = size (0-2000 feet^2)

x_2 = number of bedrooms (1-5)



$$x_1 = \frac{\text{size (feet}^2\text{)}}{2000}$$

$$x_2 = \frac{\text{number of bedrooms}}{5}$$



Feature Scaling

Get every feature into approximately a $-1 \leq x_i \leq 1$ range.

Feature Scaling

Get every feature into approximately a $-1 \leq x_i \leq 1$ range.

Mean normalization

Replace x_i with $x_i - \mu_i$ to make features have approximately zero mean (Do not apply to $x_0 = 1$).

E.g. $$x_1 = \frac{size-1000}{2000}$$

$$x_2 = \frac{\#bedrooms-2}{5}$$

$$-0.5 \leq x_1 \leq 0.5, -0.5 \leq x_2 \leq 0.5$$

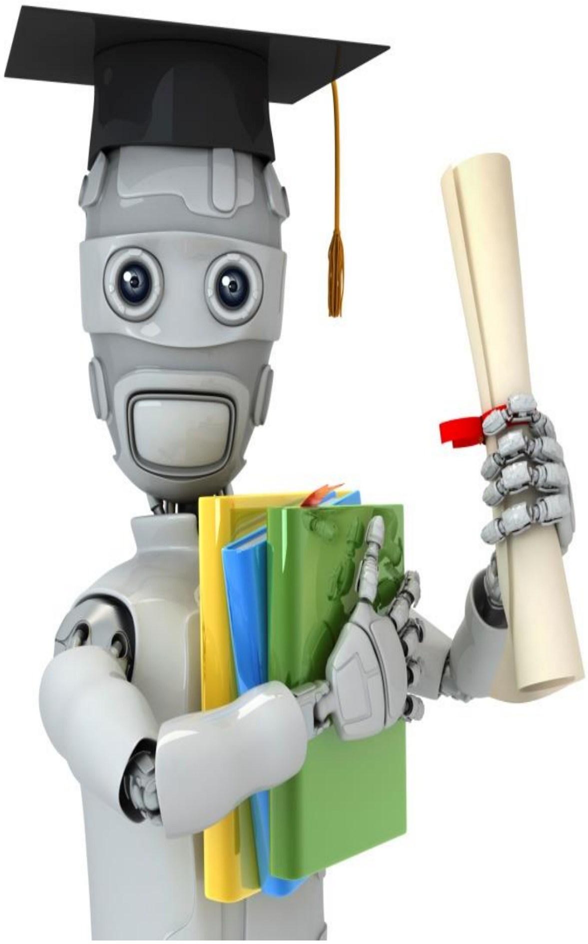
Mean normalization

Replace x_i with $x_i - \mu_i$ to make features have approximately zero mean (Do not apply to $x_0 = 1$).

E.g. $$x_1 = \frac{size - 1000}{2000}$$

$$x_2 = \frac{\#bedrooms - 2}{5}$$

$$-0.5 \leq x_1 \leq 0.5, -0.5 \leq x_2 \leq 0.5$$



Machine Learning

Linear Regression with multiple variables

GRADIENT DESCENT IN PRACTICE II: LEARNING RATE

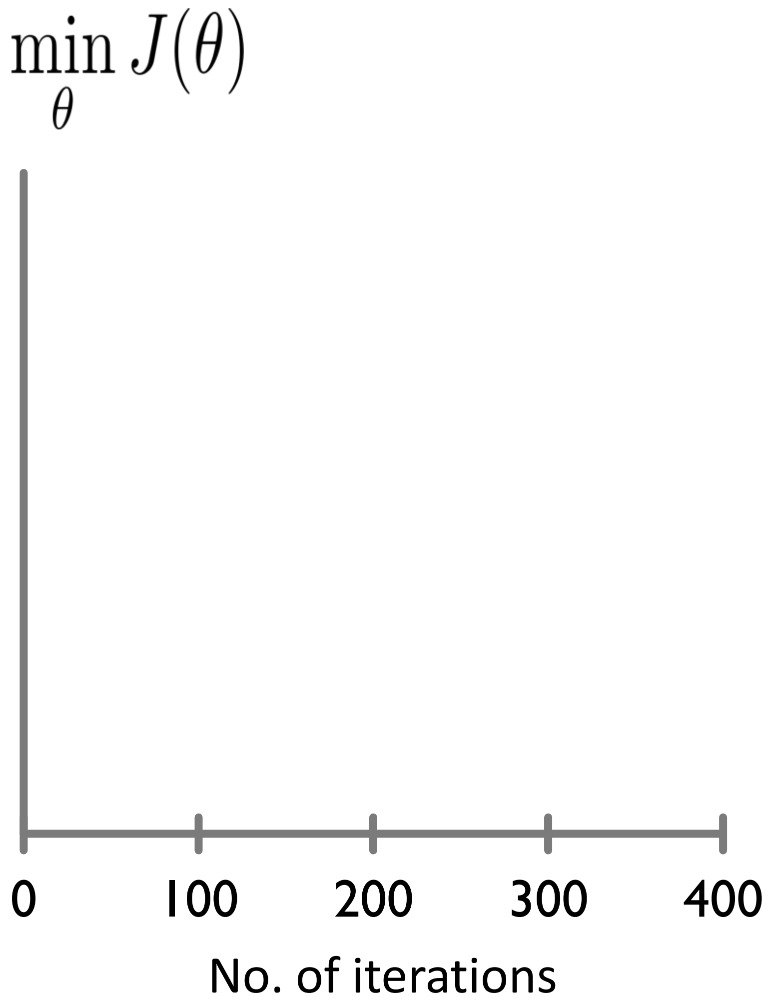
Gradient descent

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

- "Debugging": How to make sure gradient descent is working correctly.

- How to choose learning rate α.

Making sure gradient descent is working correctly.

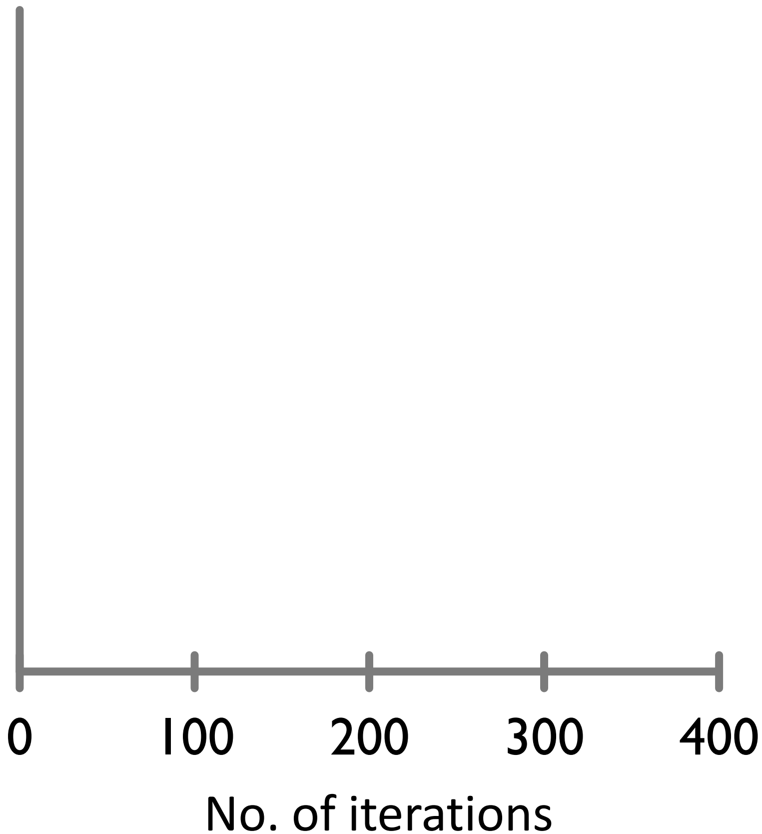


Example automatic convergence test:

Declare convergence if $J(\theta)$ decreases by less than 10^{-3} in one iteration.

Making sure gradient descent is working correctly.

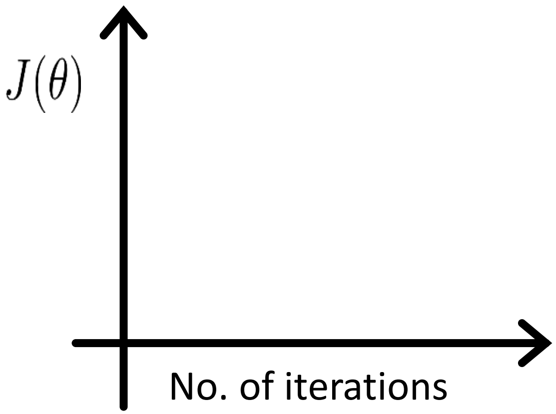
$$\min_\theta J(\theta)$$



Example automatic convergence test:

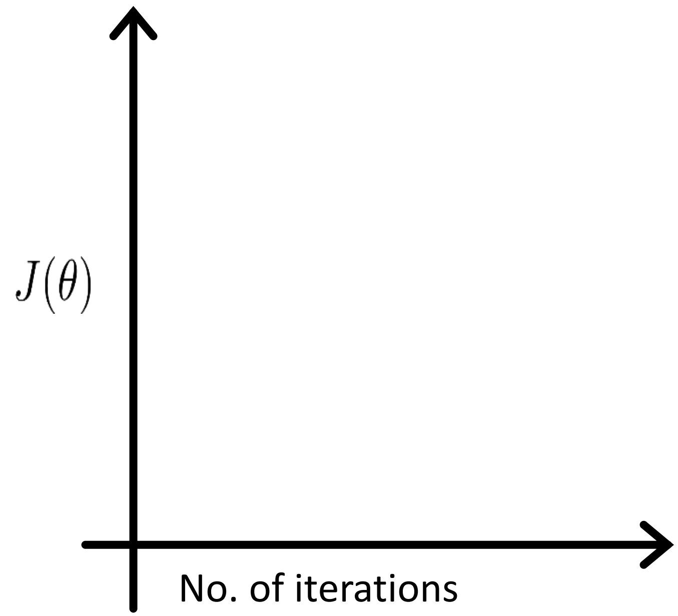
Declare convergence if $J(\theta)$ decreases by less than 10^{-3} in one iteration.

Making sure gradient descent is working correctly.



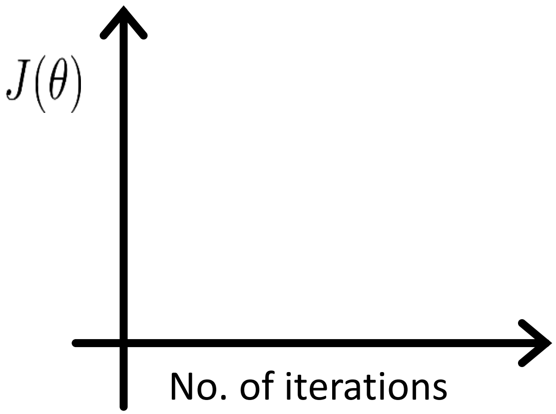
Gradient descent not working.

Use smaller α.



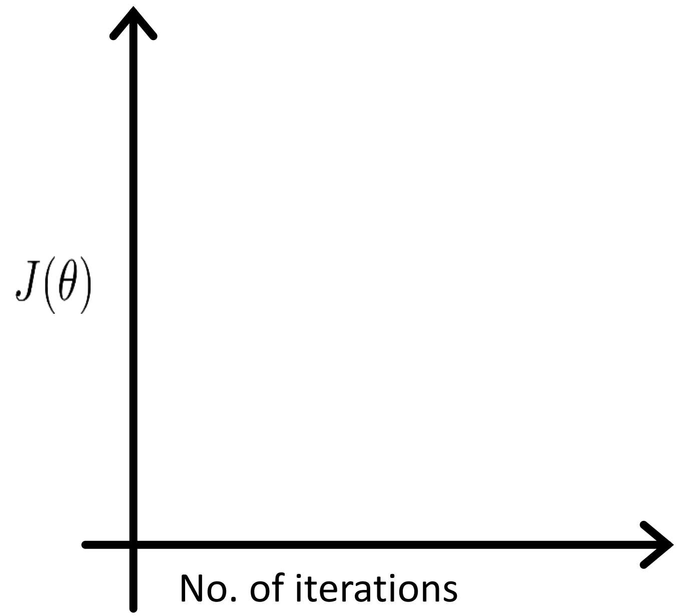
- For sufficiently small α, $J(\theta)$ should decrease on every iteration.
- But if α is too small, gradient descent can be slow to converge.

Making sure gradient descent is working correctly.



Gradient descent not working.

Use smaller α.



- For sufficiently small α, $J(\theta)$ should decrease on every iteration.
- But if α is too small, gradient descent can be slow to converge.

Summary:

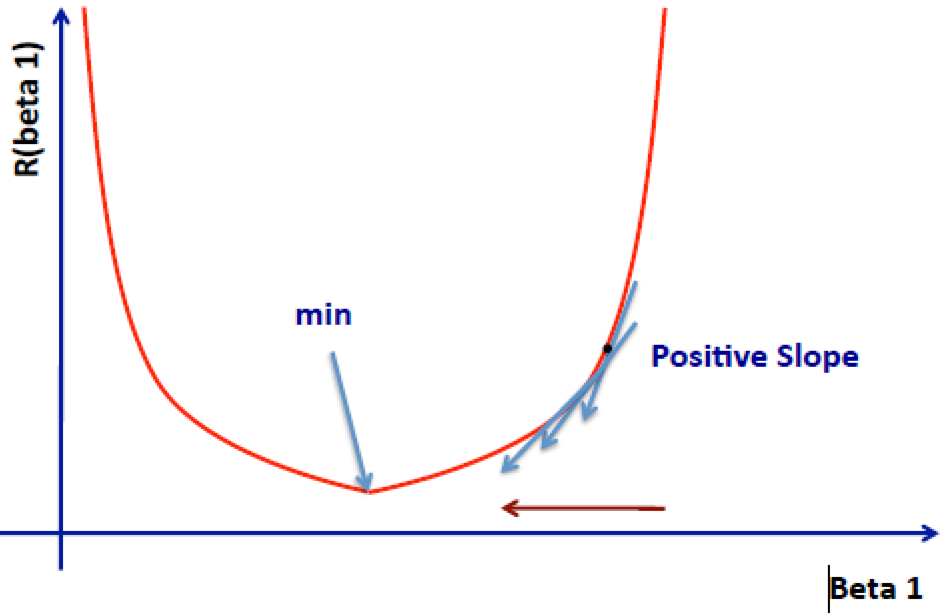
- If α is too small: slow convergence.
- If α is too large: $J(\theta)$ may not decrease on every iteration; may not converge.

To choose α, try

$$\ldots, 0.001, \quad , 0.01, \quad , 0.1, \quad , 1, \ldots$$

CONCLUSION

Gradient descent



ASSIGNMENT 07

- Extend the GradientDescent Algorithm for Multivariate Linear Regression.
- You need to report following convergence Resutls for both with Normalization and without Normaalization

ASSIGNMENT 07

- All Convergence Values (Command Prompt as shown in slides)
- All Theta Graphs (Separate for ach theta)
- Surface Graph (use any two theta values) with the Gradient Descent Calculated values of the thetas
- Report the alpha rate, initial theta values along with your results.
- Algorithm (.py file)