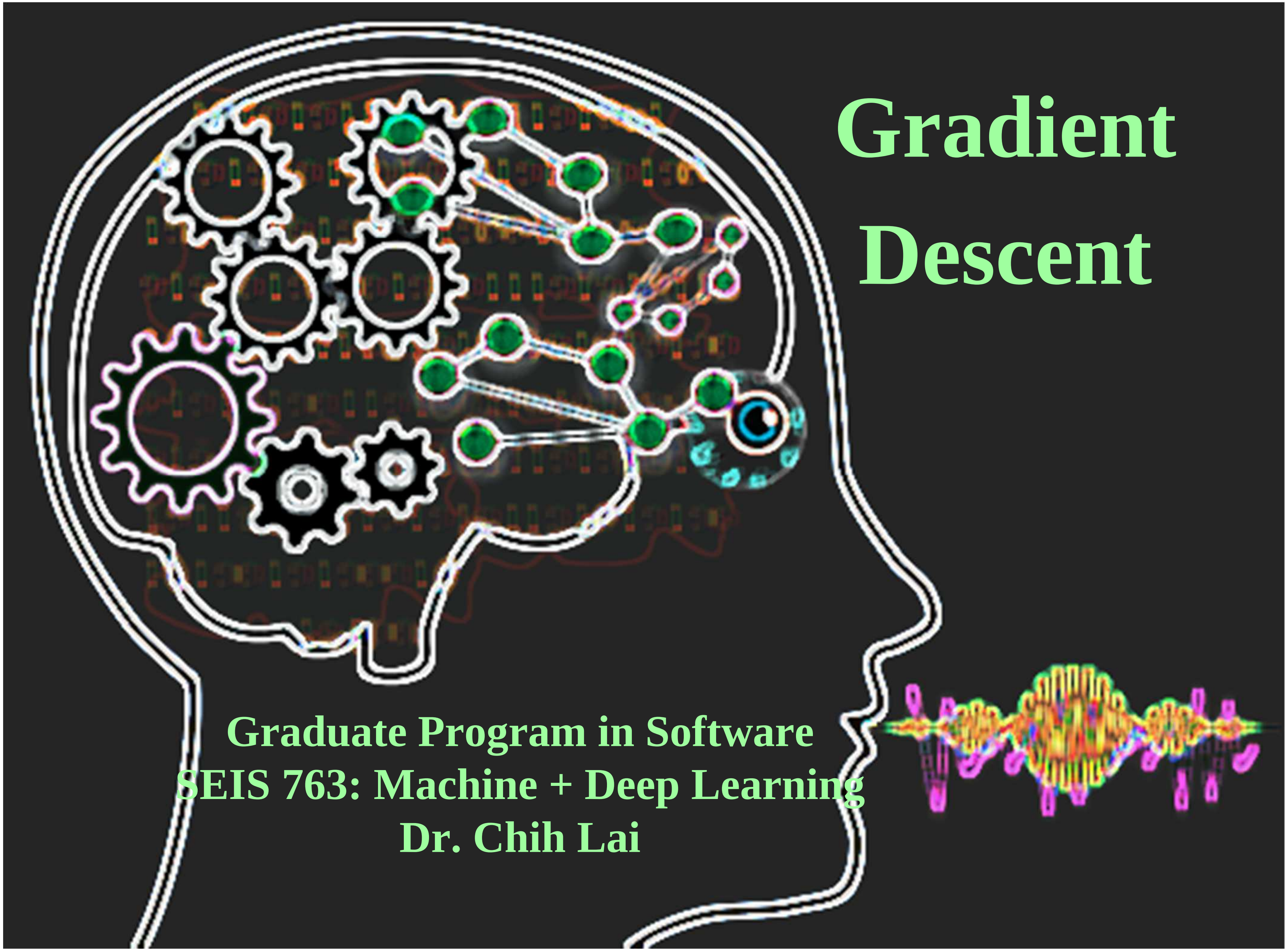
Gradient Descent



Graduate Program in Software

SEIS 763: Machine + Deep Learning

Dr. Chih Lai

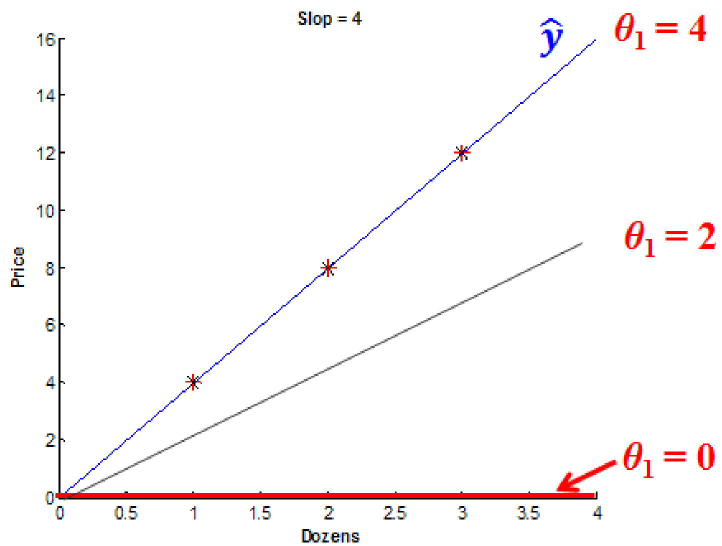
Outline

- Choosing Parameters ***θ*** to Minimize MSE (or **<u>other</u>** error functions).
  - Cost / Objective / Loss Function.
  - Machine Learning by ***Gradient Descent*** to Minimize MSE.
- The Impact of Learning Rate α.
- The Impact of Feature Scaling.

How to Choose Parameters θs for LR?

- LR basic concept: data ➔ $h_\theta(x)$ ***hypothesis*** ➔ estimates θ.
  - $h_\theta(x) = \hat{y} = \theta^T \boldsymbol{X} = \theta_0 \times \boldsymbol{x}_0 + \theta_1 \times \boldsymbol{x}_1 + \ldots + \theta_n \times \boldsymbol{x}_n$
  - Measure ***square sum of error*** ($\boldsymbol{SS_E}$) or **Residuals** as $\boldsymbol{SS_E} = \sum_i (y_i - \hat{y}_i)^2$.
    - **Purposes of squaring in the cost function: To make cost all positive.**
  - How much difference between LR model and training data.

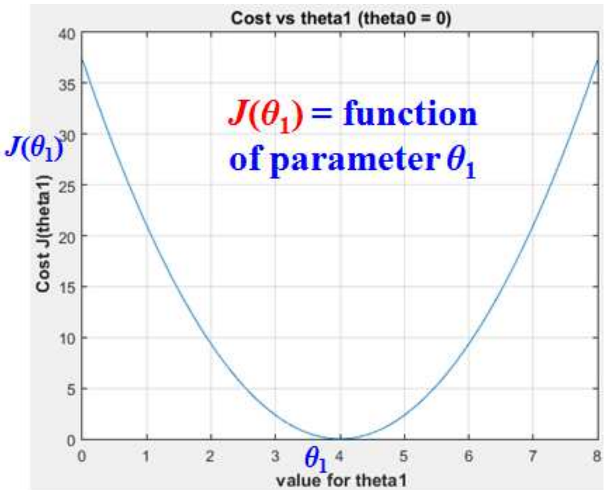
- How to choose θ parameters (i.e. ***hypothesis***) to minimize MSE?



Choosing Parameters θs to Minimize MSE

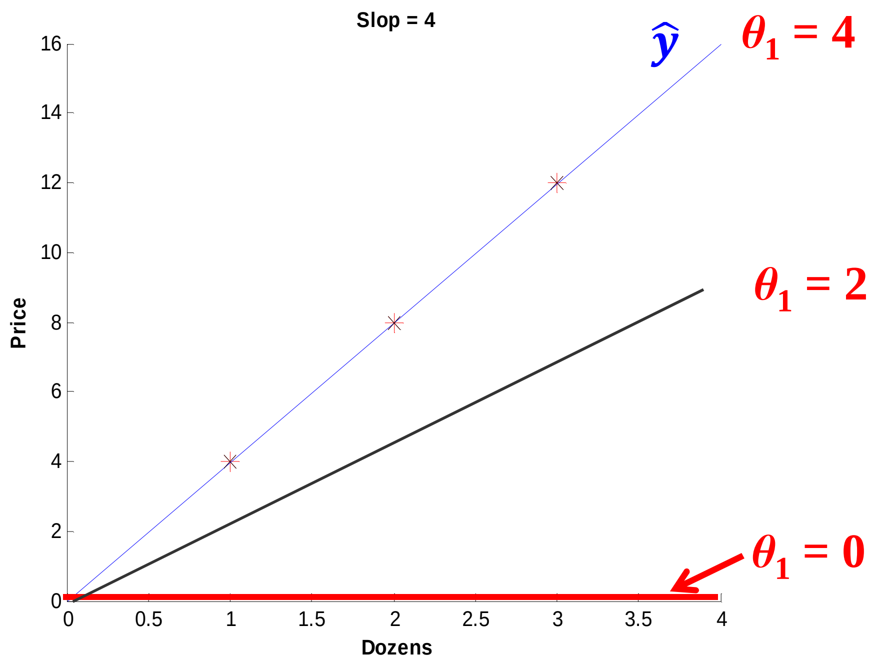
- $h_\theta(x) = \hat{y} = \theta^T \boldsymbol{X} = \theta_0 \times \boldsymbol{x}_0 + \theta_1 \times \boldsymbol{x}_1 + \ldots + \theta_n \times \boldsymbol{x}_n$

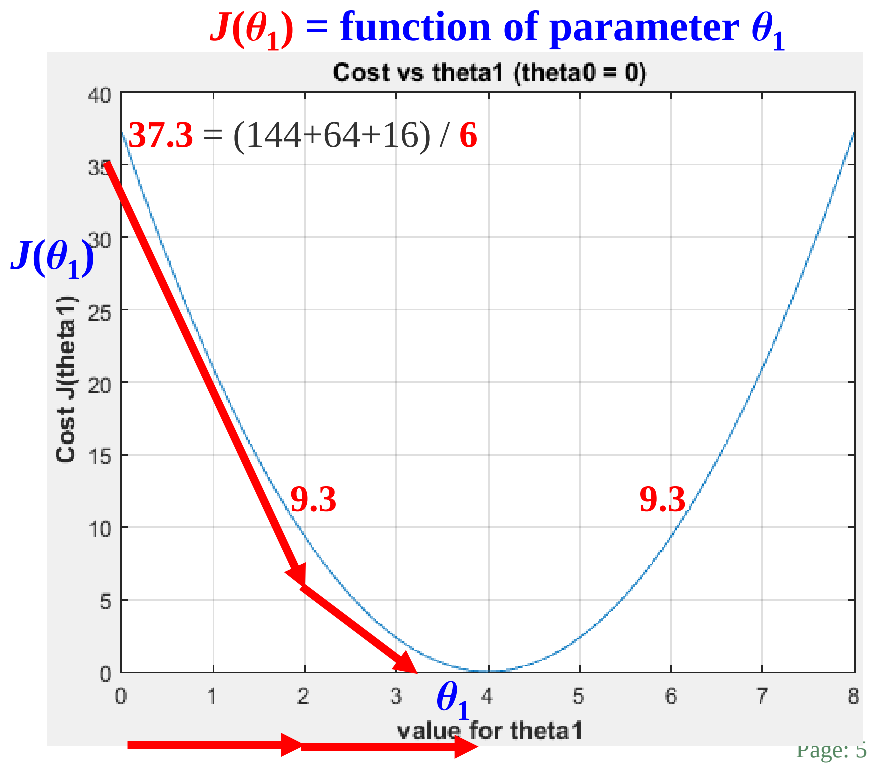
- Define a ***cost function*** (***objective*** function, ***loss*** function, ***penalty***)
  - Assume we have ***m*** records (instances), each record has ***p*** attributes.
    - $\boldsymbol{\theta}$ is a $(p \times 1)$ vector, $\boldsymbol{X}$ is a $(p \times m)$ matrix of all ***m*** input records.
    - $\boldsymbol{Y}$ is a $(1 \times m)$ vector, containing values for target ***responses***.
  - $SSE = \sum_{j=1}^{m}(y_j - \hat{y}_j)^2 = \sum_{j=1}^{m}\left(y_j - \sum_{i=1}^{p}\theta_i x_{ij}\right)^2 = (Y - X\theta)^T(Y - X\theta)$.
  - **MSE** (***cost function***) $\boldsymbol{J(\theta)} = \frac{1}{\mathbf{2}m}\sum_{j=1}^{m}(y_j - \hat{y}_j)^2$
    - Outliers may have great impact on linear regression.
  - How does the cost function look like for 1 predictor?
  - To find MSE = **minimize** the cost function



Visualizing Cost Function

- MSE (***cost function***) = ***J(θ)*** = $\frac{1}{2m}\sum_{j=1}^{m}(y_j - \hat{y}_j)^2$
  - Minimization based on **slope** or **gradient** of ***J(θ)***
  - Why not moving ***θ*** "🡐" that direction?





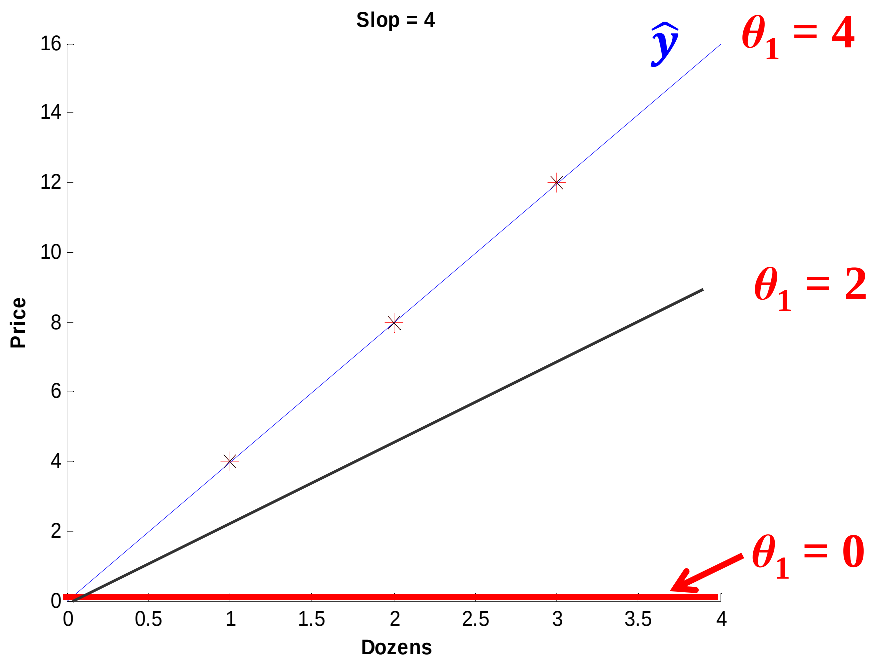
Machine Learning to Minimize Cost Function

- Minimize MSE (*cost function*) = $\boldsymbol{J(\theta)} = \frac{1}{2m}\sum_{j=1}^{m}(y_j - \hat{y}_j\ )^2$ by…
  - Set **partial derivatives** (**slope, <u>gradient</u>**) of $\boldsymbol{J(\theta) = 0}$ to find parameters θ.

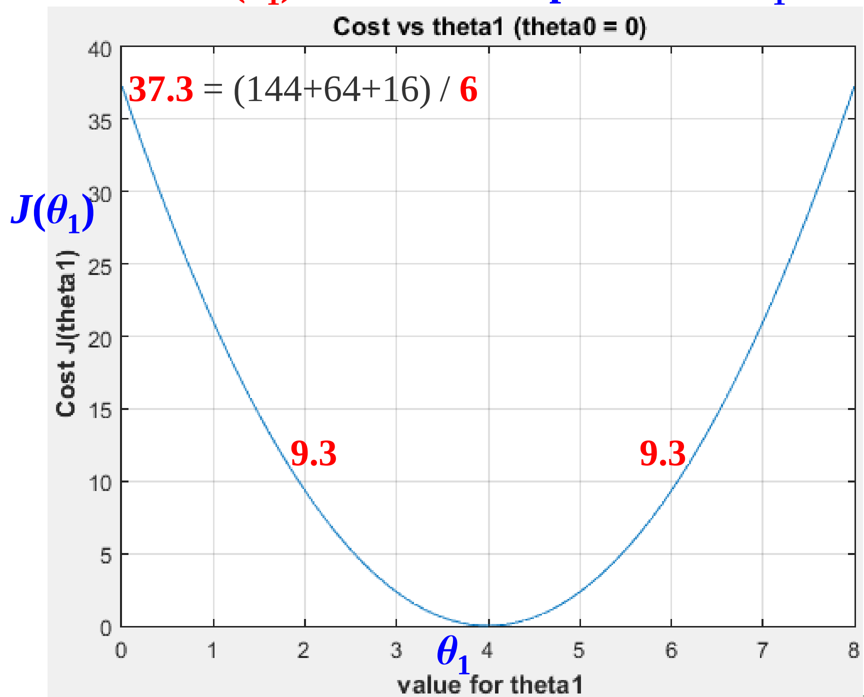
- Machine Learning **Steps:**
  1. Randomly set $\boldsymbol{\theta}$, and then
  2. **<u>Repeatedly</u>** adjust $\boldsymbol{\theta}$ based on ***gradient***.

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

$$\boldsymbol{\theta_1 = \theta_1 - \alpha \frac{1}{m}\sum_{j=1}^{m}(y_j - h_\theta(x_j)) \times x_1}$$



$J(\theta_1)$ = function of parameter θ_1



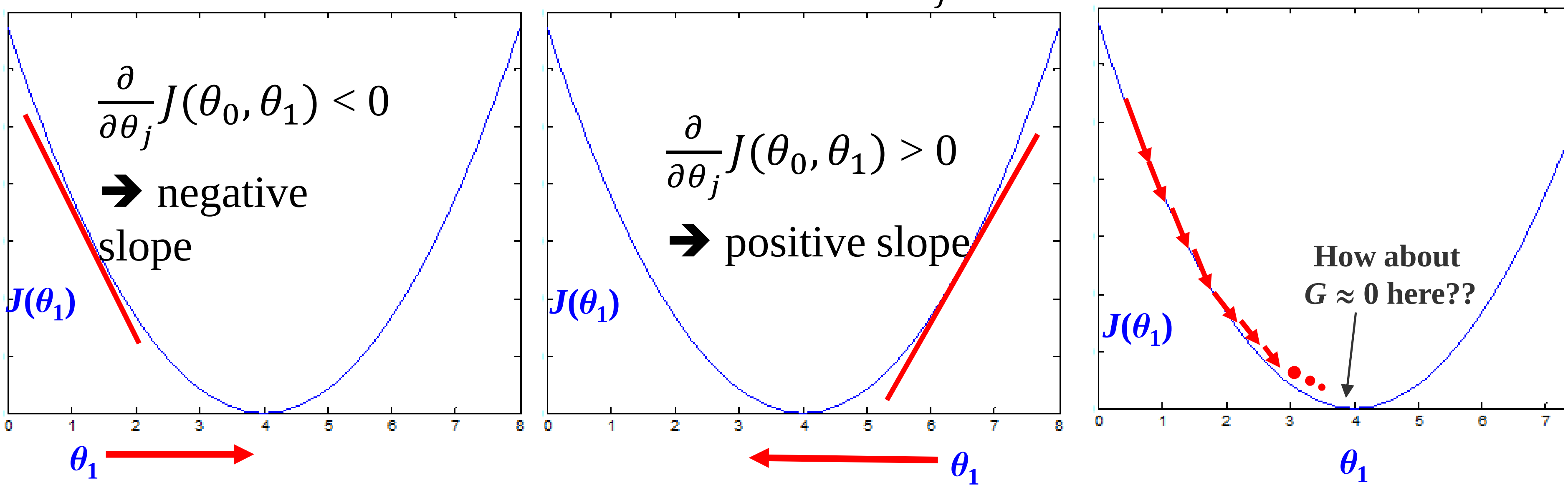
Summary– Gradient Descent in Reducing Cost Function

1) Define some cost function $J(\theta_0, \theta_1) = J(\theta) = \frac{1}{2m}\sum_{j=1}^{m}(y_j - \hat{y}_j)^2$ or ***other*** functions

2) Randomly begin with some θ_0 , θ_1.

3) Keep changing θ_0 , θ_1 to reduce $J(\theta_0, \theta_1)$ ➔ until reaching a "***good***" minimum.
   - For **–** slope, θ_1 will move right. For **+** slope, θ_1 will move left.
   - Closer to minimum, gradient descent will slow down **automatically**. **Learning rate decay**
   - Slope becomes smaller…

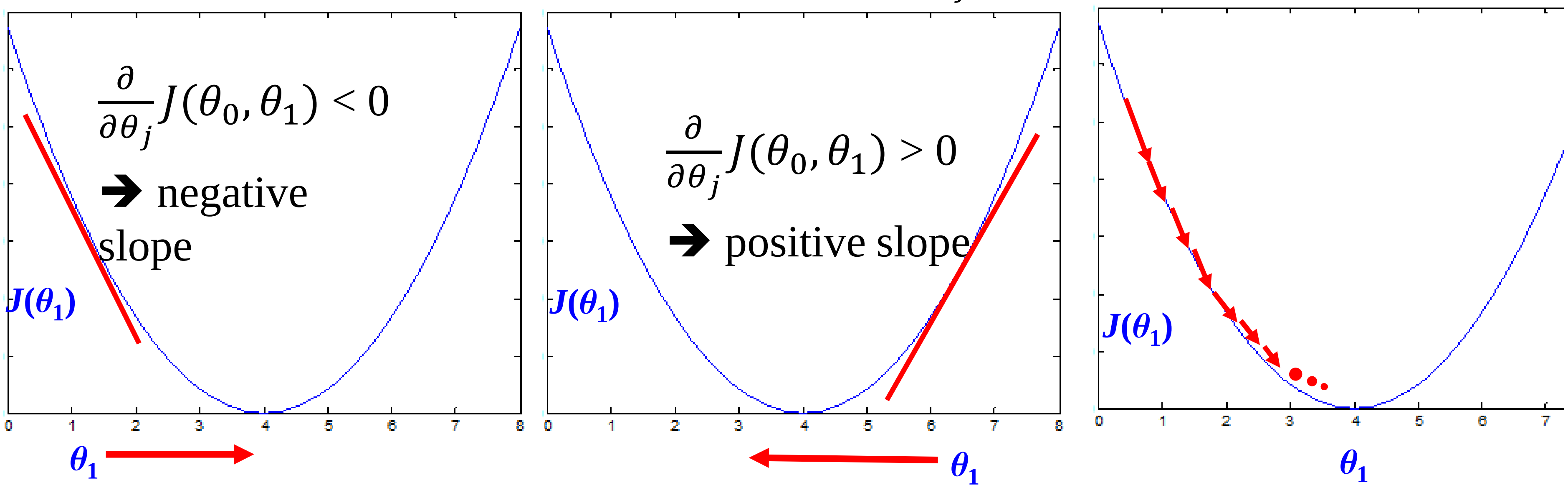
$$\theta_j = \theta_j - \boldsymbol{G} = \theta_j - \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$



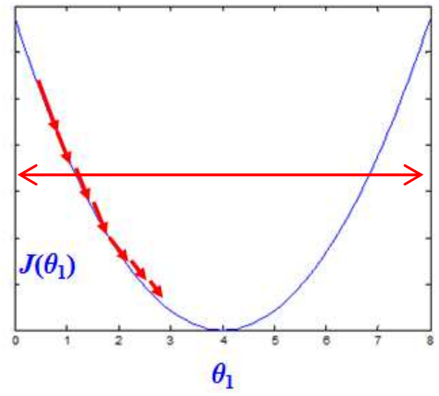
Gradient in Plain English

- Compute a gradient for each weight θ_j and bias θ_0.
  - A gradient is just a number like -0.83.
- Magnitude of the gradient gives you a hint about how much θ should change.
- The sign of the gradient tells us to increase or decrease θ
  - So that $\hat{y}$ will get closer to the target output y to minimize ***MSE***.

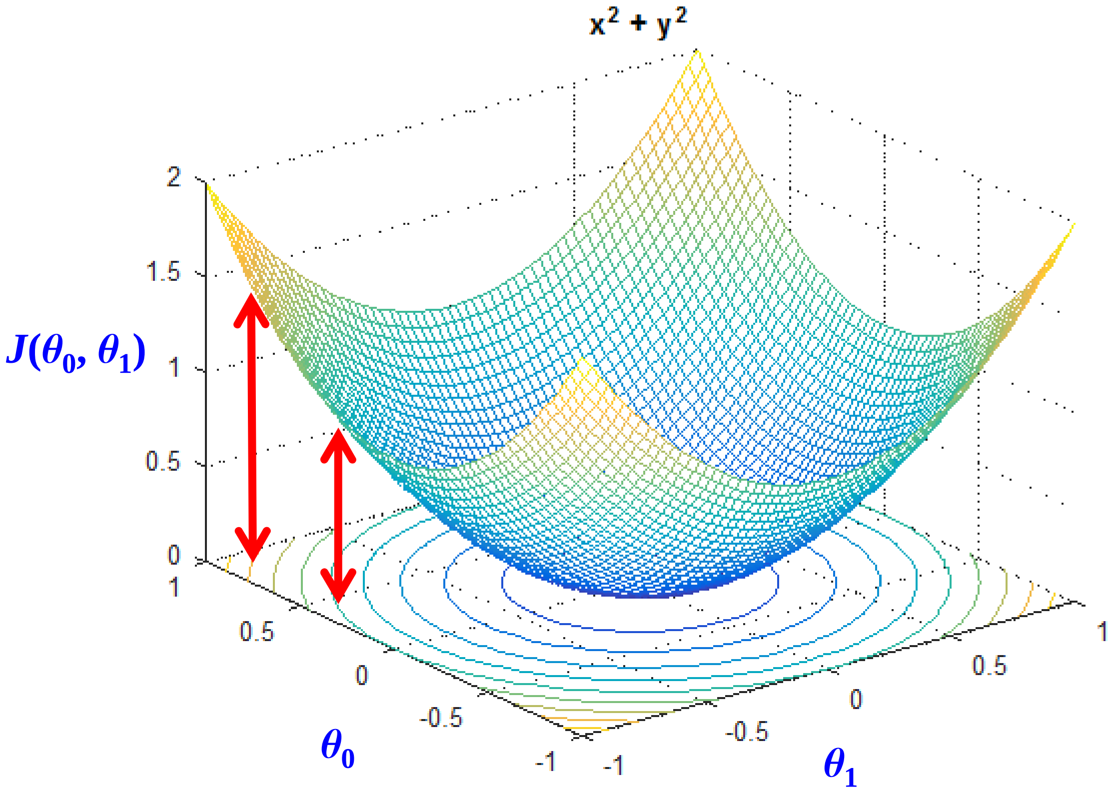
$$\theta_j = \theta_j - \mathbf{G} = \theta_j - \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$



Other Cost Function Example



- Gradient Descent on θ_1 and θ_0.
- How to plot a surface in Matlab?
  - With **contour** on the bottom.
  - **NOT** just make it fancier.

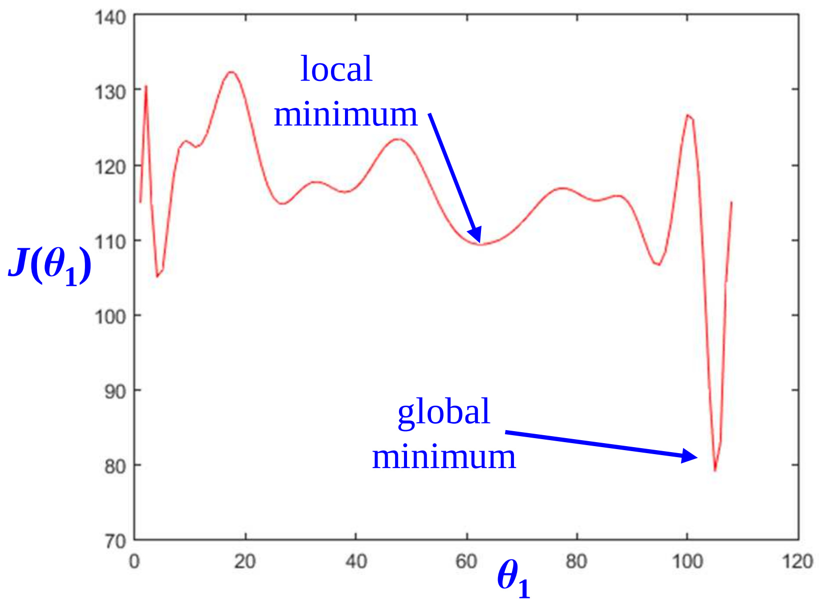


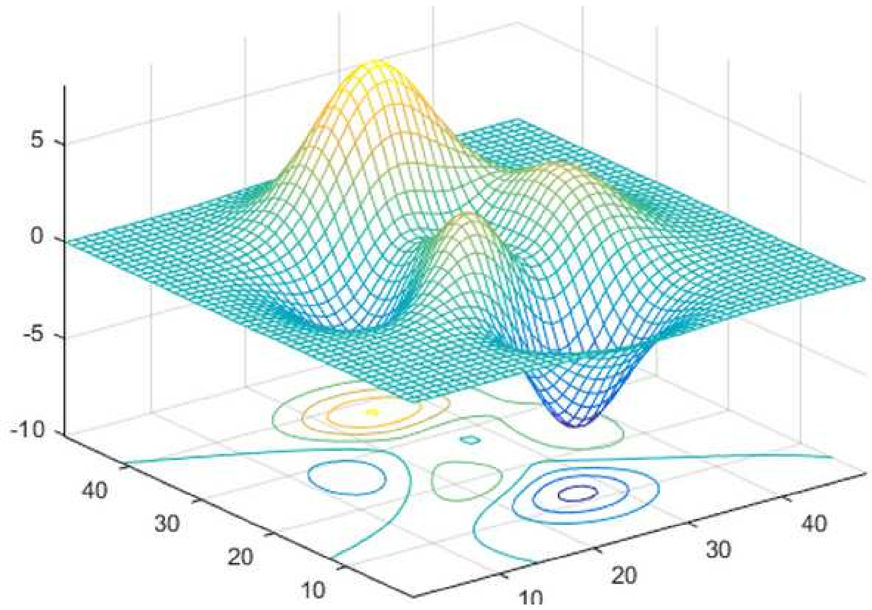

```
syms x y
figure
ezsurf(x^2 + y^2, [-1, 1])

figure
ezmeshc(x^2 + y^2, [-1, 1])

figure
ezmeshc(x^2 + y^2, [-1, 1, -0.5, 1.5])

figure
ezcontour(x^2 + y^2, [-1, 1])

figure
ezcontourf(x^2 + y^2, [-1, 1])
```

See detail Matlab functions for plotting in Appendix

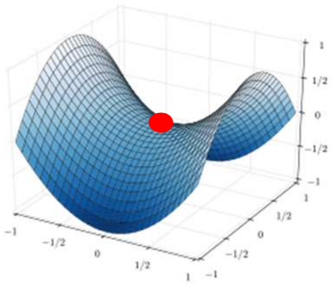
Local Minimum

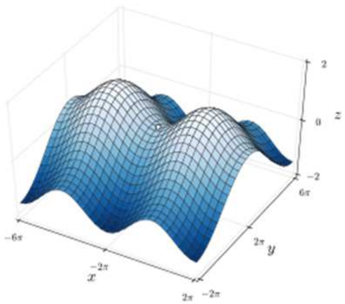
- Gradient descent may find local minimum, **<u>NOT</u>** **<u>GLOBAL</u>** minimum.
  - **<u>IF</u>** your cost function is **non-convex**.






```
[X,Y] = meshgrid(-3:.125:3);
Z = peaks(X,Y);
meshc(Z)
```





https://en.wikipedia.org/wiki/Saddle_point

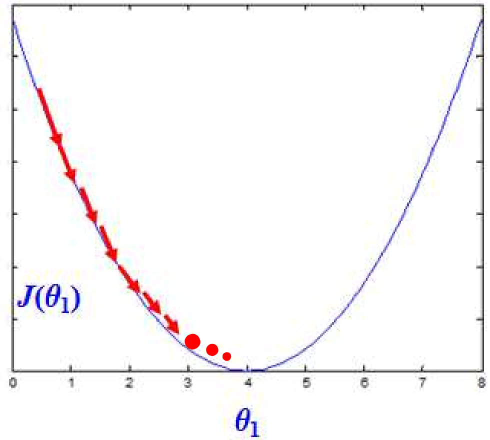
Outline

- Choosing Parameters θ to Minimize MSE (or **other** error functions).
  - Cost / Objective / Loss Function.
  - Machine Learning by ***Gradient Descent*** to Minimize MSE.
- The Impact of Learning Rate α.
- The Impact of Feature Scaling.

Learning Rate α

- Minimize $J(\theta) = \frac{1}{2m}\sum_{j=1}^{m}(y_j - \hat{y}_j)^2$
  - Set **partial derivatives** (**slope, gradient**) of ***J*(*θ*) = 0**.

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



- α is the learning rate = how big each downhill step.
  - $\alpha > \mathbf{0}$ must always hold.

repeat until convergence for ***j*** = 0 and ***j*** = 1:

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

repeat until convergence for ***j*** = 0 and ***j*** = 1:

derivative result

$$\{ \ \theta_0 = \theta_0 - \alpha \frac{1}{m}\sum_{j=1}^{m}\left(y_j - h_\theta(x_j)\right) \times \boxed{x_0}$$

NOTE: X_0 = 1

$$\theta_1 = \theta_1 - \alpha \frac{1}{m}\sum_{j=1}^{m}(y_j - h_\theta(x_j)) \times x_1$$

$$\}$$

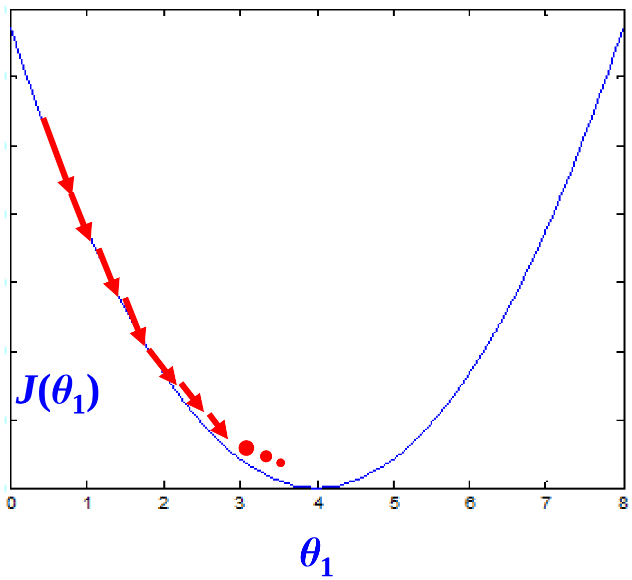
(see Appendix for details)

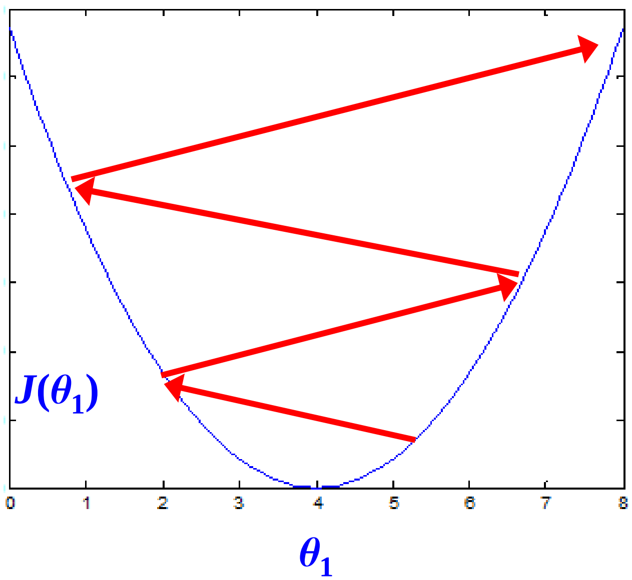
Impact of Learning Rate α

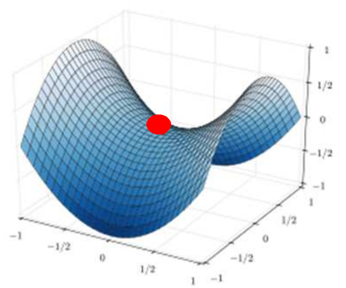
- If α too small, gradient descent takes long time to converge to find best θ.
  - **Unable to learn** or stop learning.

- If too big, gradient descent ***overshoot*** local minimum & may fail to converge.
  - **Unstable** learning.

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

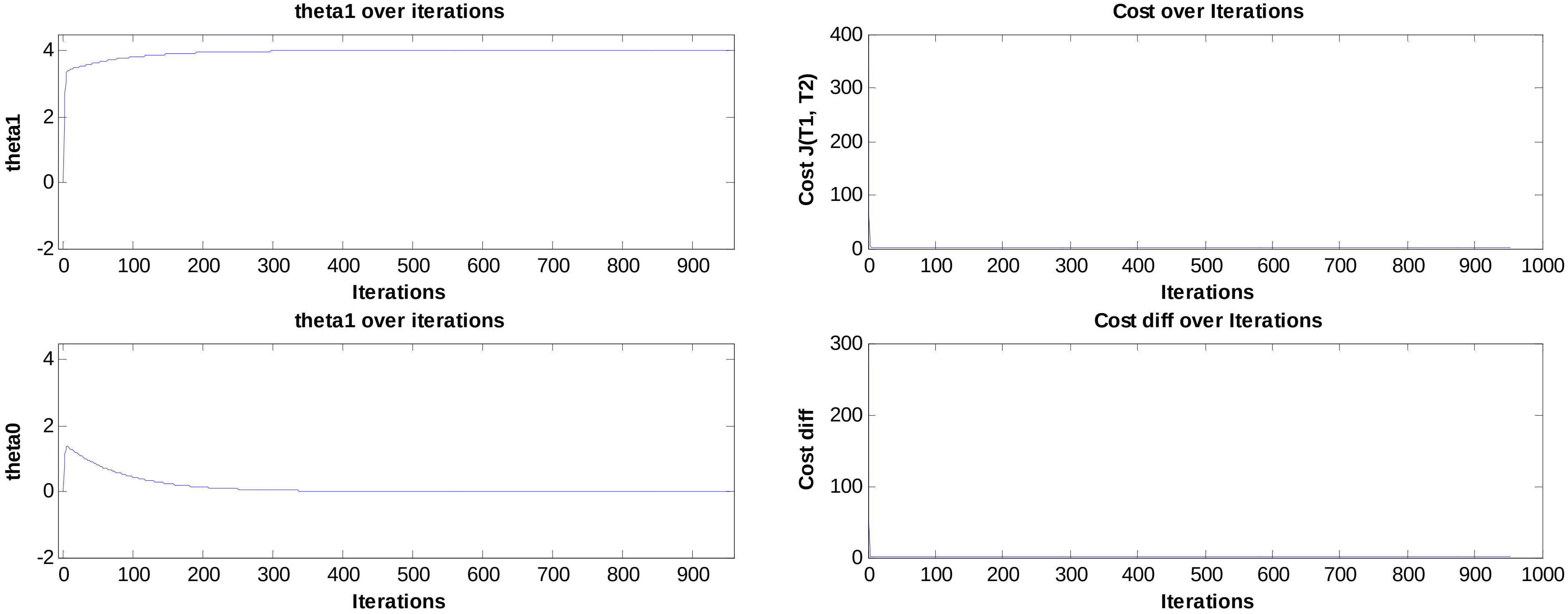






https://en.wikipedia.org/wiki/Saddle_point

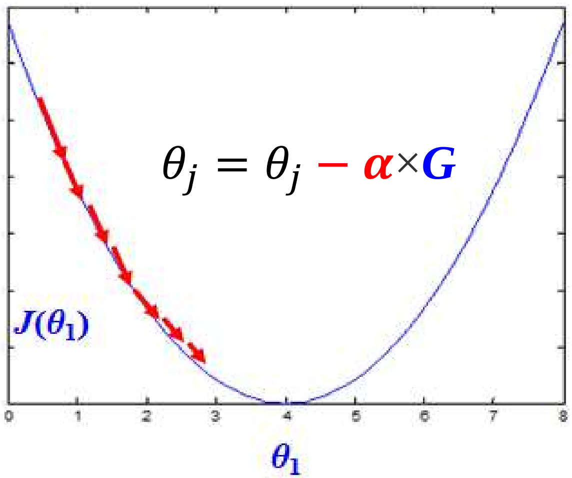
Gradient Descent, Learning Rate α = 0.1, Converged



X = [1 2   3]

Y =[4 8 12]

$\theta = \begin{bmatrix} 0 \\ 4 \end{bmatrix}$

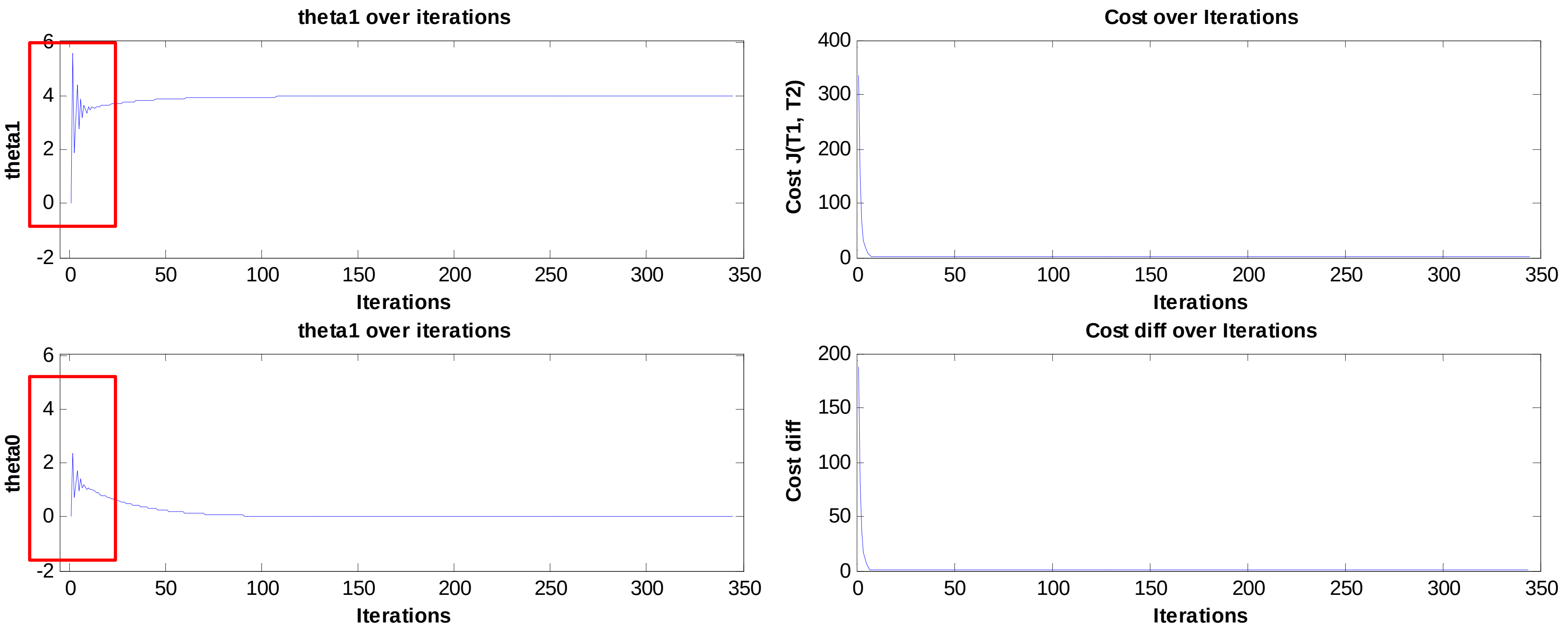


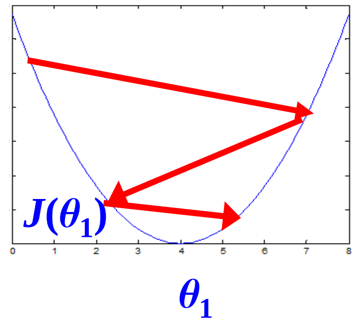
% Pseudo code for monitoring progress

```
if Err > 10^6
      disp('Looks like it is diverging ' );        break
end

if abs(diff_error_from_two_iterations) <= 10^-7,
      disp('converge, error improvement is small enough.');
      break
end
```

Gradient Descent, Learning Rate α = 0.3, Converged

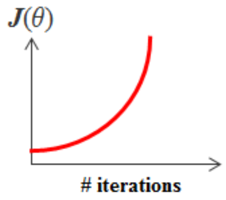
- But, notice the convolution at the first few iterations.

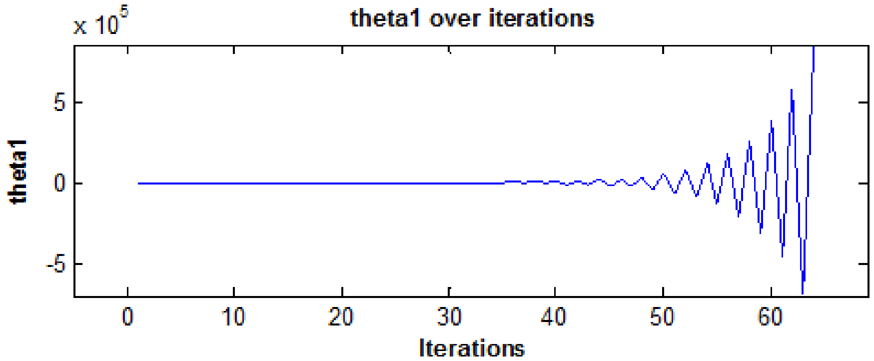


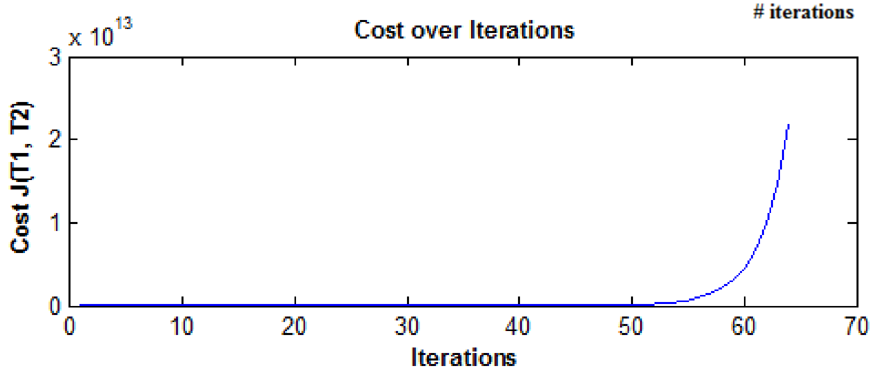


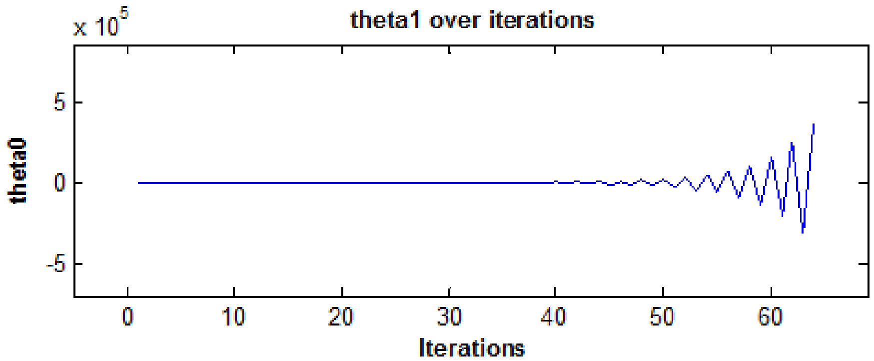
$$\theta_j = \theta_j - \alpha \times G$$

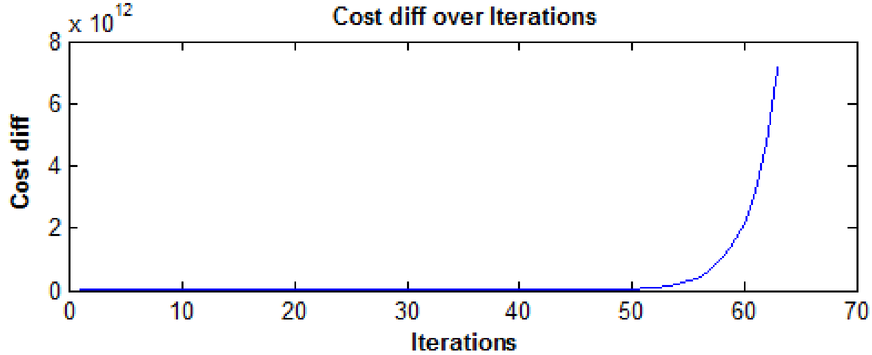
Gradient Descent α = 0.4 **Diverge** Learning

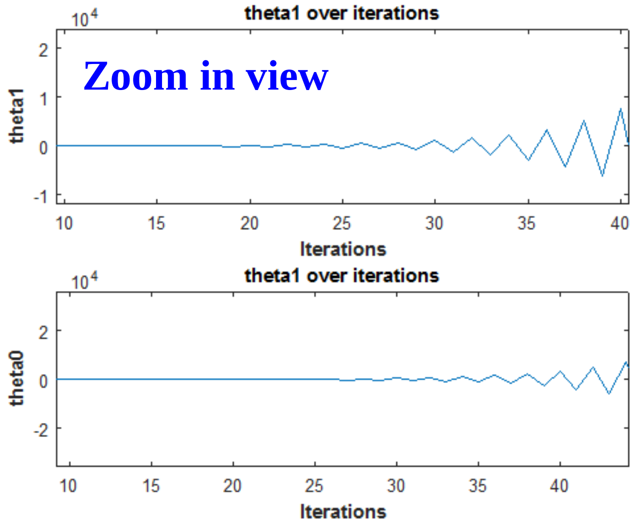


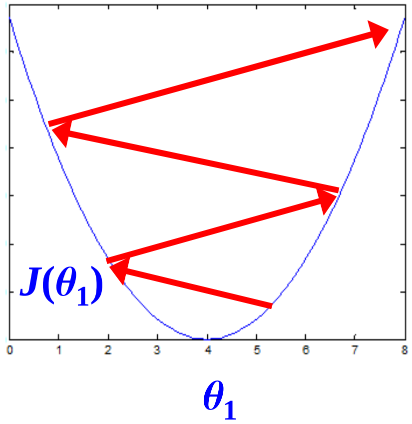












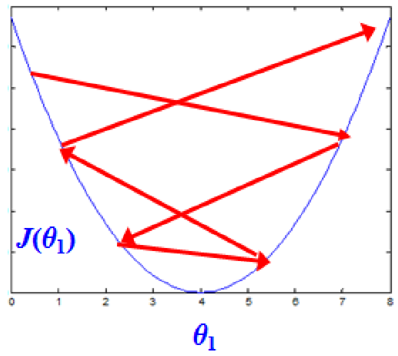
$$\theta_j = \theta_j - \alpha \times G$$

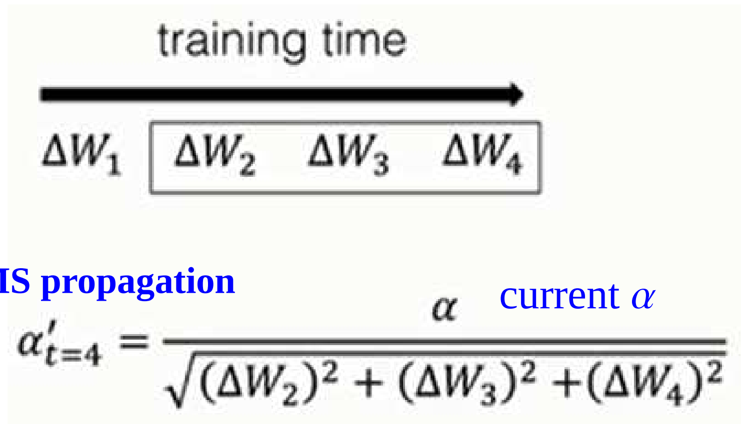
Stochastic Gradient Descent

- Begin with a random α, then **<u>gradually</u>** modify α.

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

$$\text{new } \alpha = \frac{\text{current } \alpha}{\Sigma(\text{past } gradient\mathbf{sssss})}$$

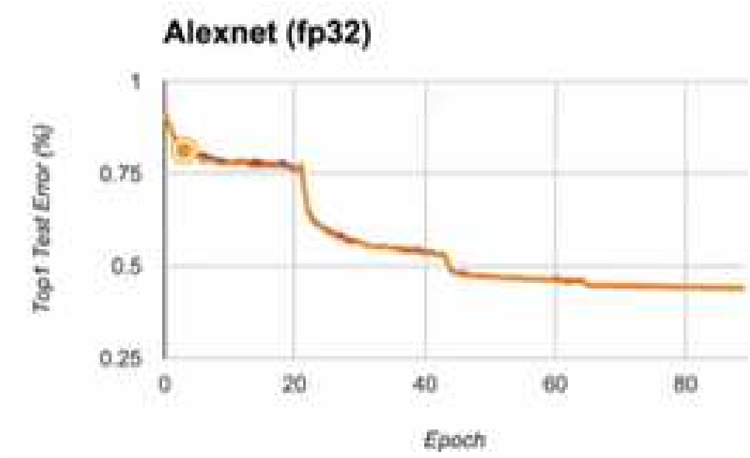




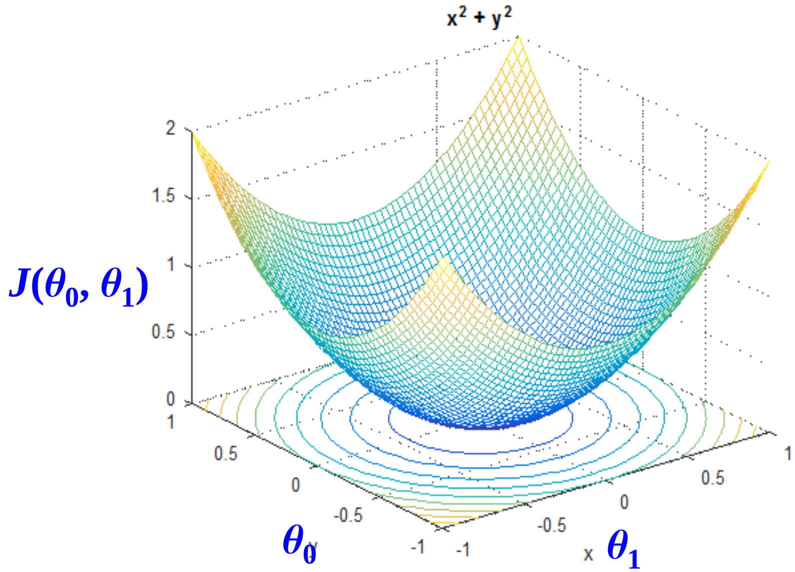
RMS propagation

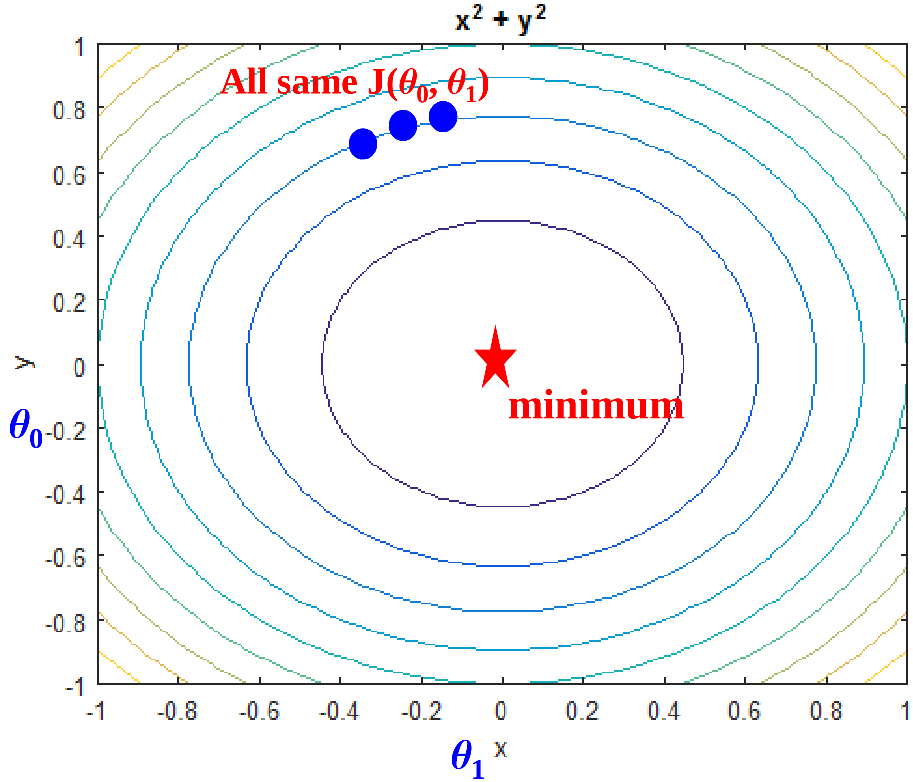
current α

new $\alpha\uparrow$ ← past gradient↓

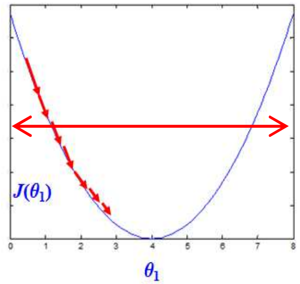


Interpretation of A Contour Plot over θ_1 and θ_0





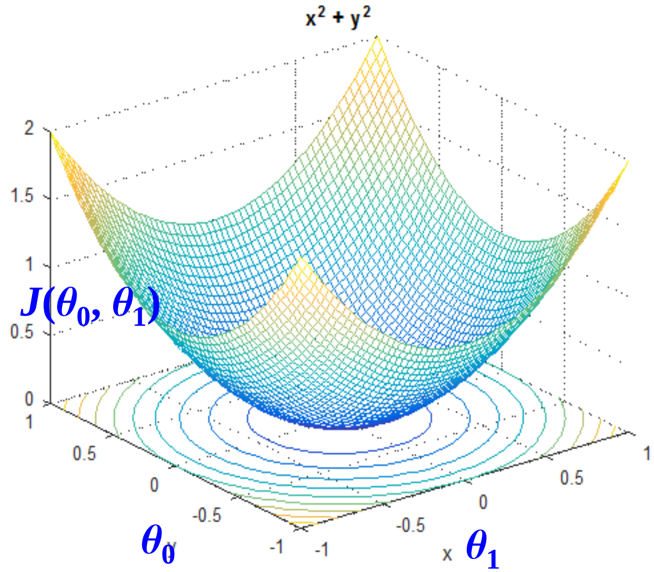
Each eclipse/circle line shows the contour curve with the same cost value.

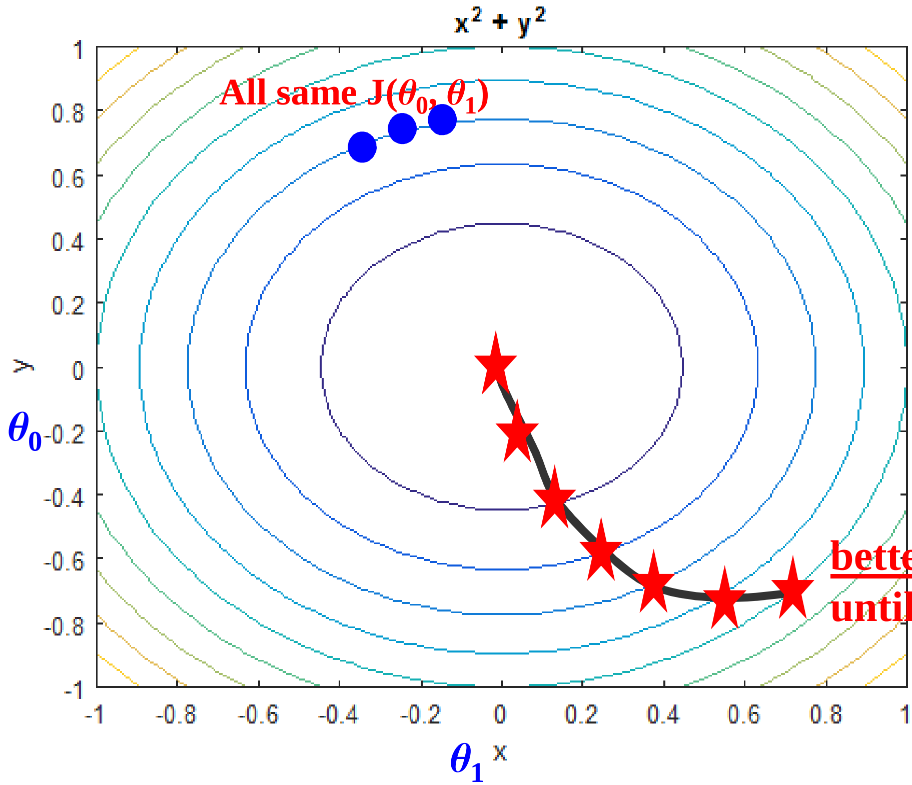



```
syms x y
figure
ezcontour(x^2 + y^2, [-1, 1])

figure
ezcontourf(x^2 + y^2, [-1, 1])
```

Animation of Gradient Descent on A Contour Plot over θ_1 and θ_0

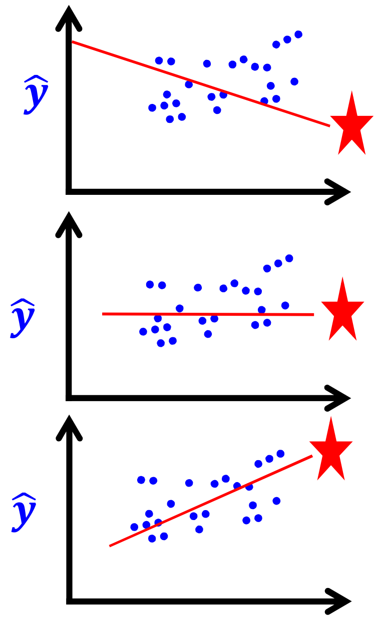
- GD improves loss by small changes in θ.





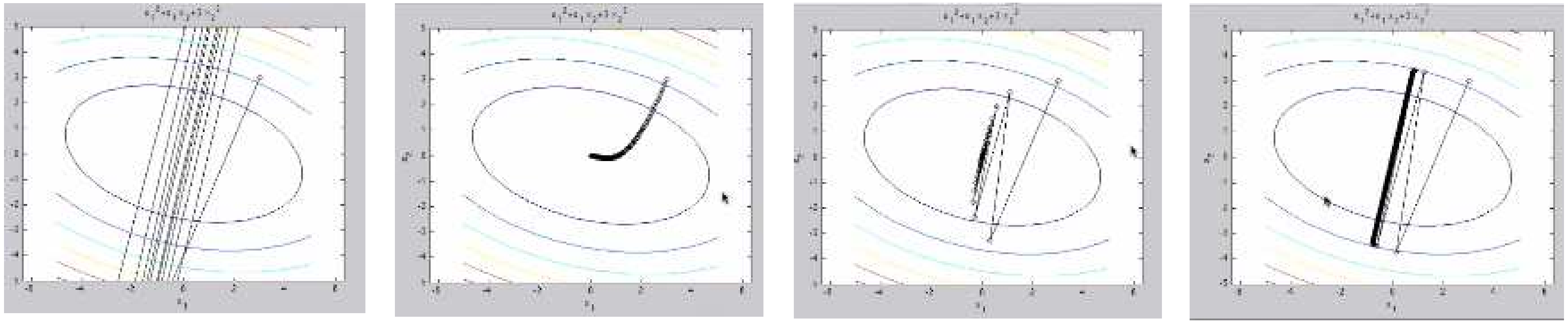
Each eclipse line shows the contour curve with the same cost value.

better & better until minimum

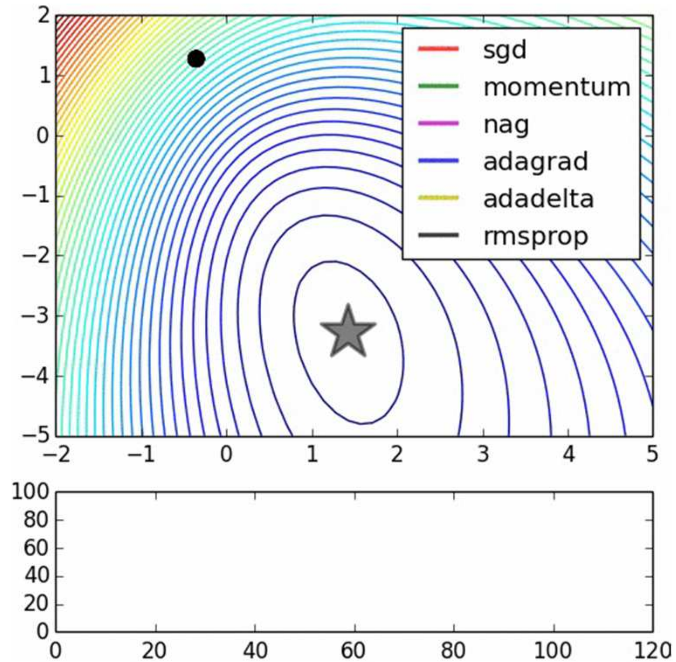


Matlab File Exchange Gradient Descent Tool

- Good animation 1
  - http://www.mathworks.com/matlabcentral/fileexchange/35535-simplified-gradient-descent-optimization



- Good animation 2
  - http://danielnouri.org/notes/2014/12/17/using-convolutional-neural-nets-to-detect-facial-keypoints-tutorial/



Different gradient descent algorithms:

http://ruder.io/optimizing-gradient-descent/

https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f

https://www.analyticsvidhya.com/blog/2017/03/introduction-to-gradient-descent-algorithm-along-its-variants/

Outline

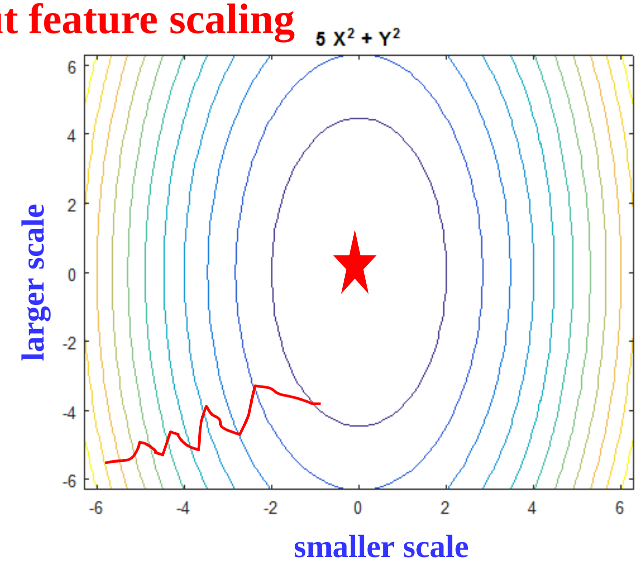
- Choosing Parameters ***θ*** to Minimize MSE (or **<u>other</u>** error functions).
  - Cost / Objective / Loss Function.
  - Machine Learning by ***Gradient Descent*** to Minimize MSE.
- The Impact of Learning Rate α.
- The Impact of Feature Scaling.

Speedup Gradient Descent

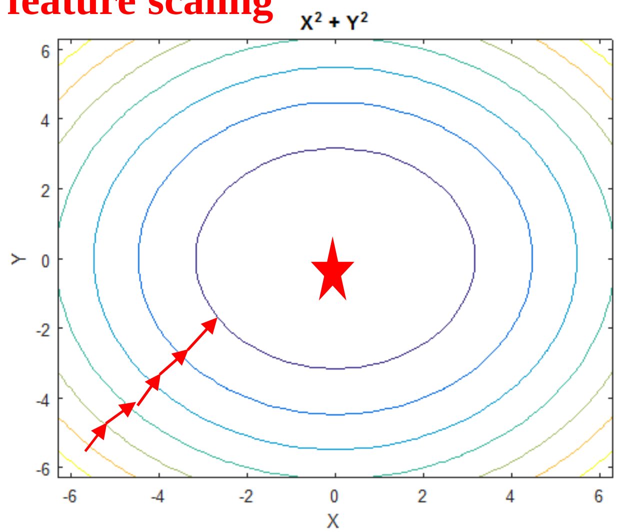
- Feature scaling–
  - If features are on the similar scale, gradient descent will converge faster.
  - Scale every numeric feature.
  - i.e. scale these two features➔ $0 \leq \boldsymbol{x}_1 \leq 15$ (# rooms) $-50{,}000 \leq \boldsymbol{x}_2 \leq 800{,}000$ ($$).

- The effect of standardization for machine learning algorithms
  - http://sebastianraschka.com/Articles/2014_about_feature_scaling.html

without feature scaling



with feature scaling



Feature Scaling

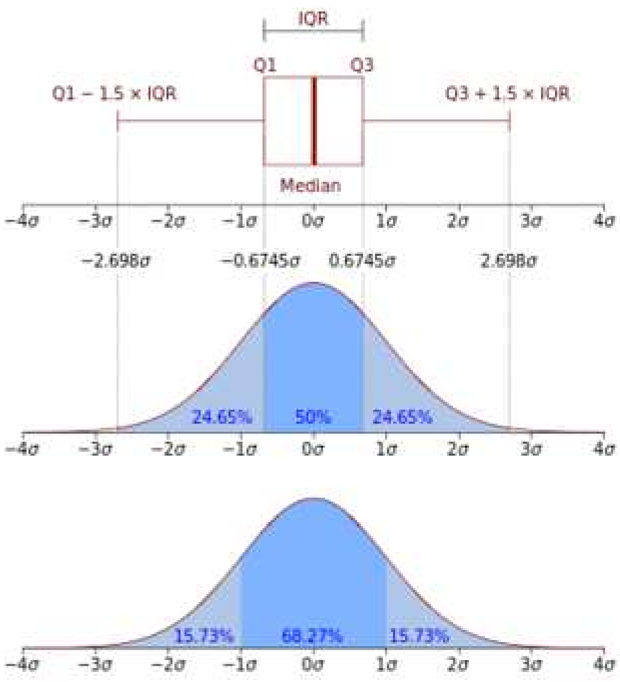
- Mean normalization (with feature scaling)–
  - Replace all features $\boldsymbol{x_i}$ with $\boldsymbol{x_i} - \mu_i$ so all features have 0-mean, not including $\boldsymbol{x}_0 = 1$.

- $\chi_i = \dfrac{\boldsymbol{x_i} - \mu_i}{\sigma_i}$ σ_i = SD of $\boldsymbol{x}$. ***Standardization***.
  - After **Z-score**, $\boldsymbol{\mu \approx 0}$ **AND** $\boldsymbol{\sigma = 1}$.

- $\chi_i = \dfrac{\boldsymbol{x_i} - \mu_i}{\max(\boldsymbol{x_i}) - \min(\boldsymbol{x_i})}$ ***Normalization***.
  - After, $\boldsymbol{\mu \approx 0}$.

- $\chi_i = \dfrac{\boldsymbol{x_i} - \min(\boldsymbol{x_i})}{\max(\boldsymbol{x_i}) - \min(\boldsymbol{x_i})} = [0 .. 1]$.

- $\chi_i = \dfrac{\boldsymbol{x_i} - \tilde{x}}{Q3(\boldsymbol{x_i}) - Q1(\boldsymbol{x_i})}$ $\tilde{x}$ = median of $\boldsymbol{x}$.

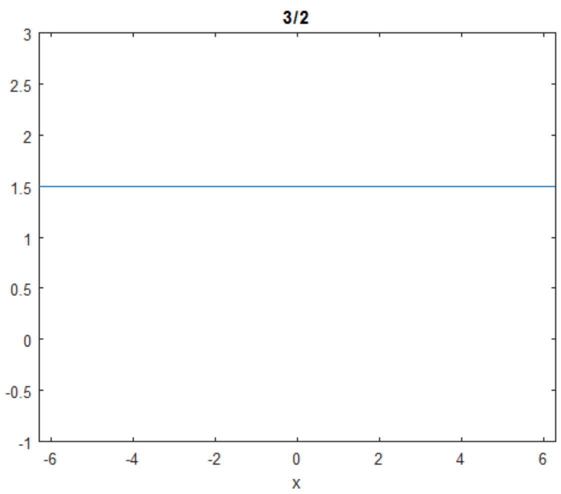


https://en.wikipedia.org/wiki/Interquartile_range

Appendix

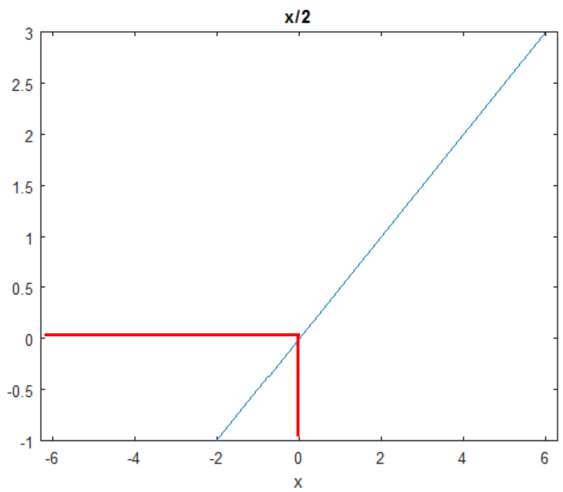
Easy Way to Visualize Hypothesis in Matlab

$$\theta_0 = 1.5$$
$$\theta_1 = 0$$



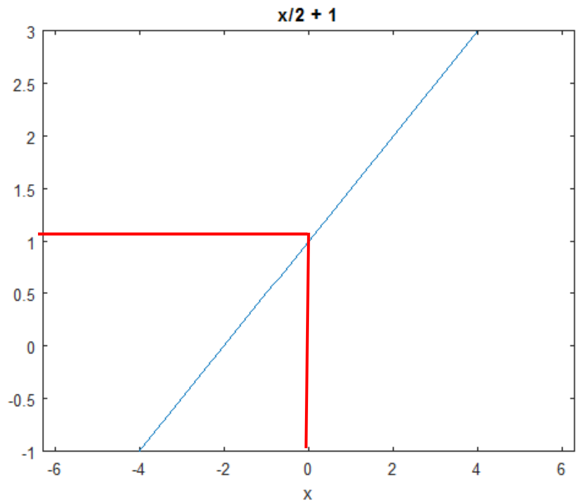

```
syms x y
ezplot(0 * x + 1.5),
ylim([-1 3])
```

$$\theta_0 = 0$$
$$\theta_1 = 0.5$$




```
syms x y
ezplot(0.5 * x + 0),
ylim([-1 3])
```

$$\theta_0 = 1$$
$$\theta_1 = 0.5$$




```
syms x y
ezplot(0.5 * x + 1),
ylim([-1 3])
```

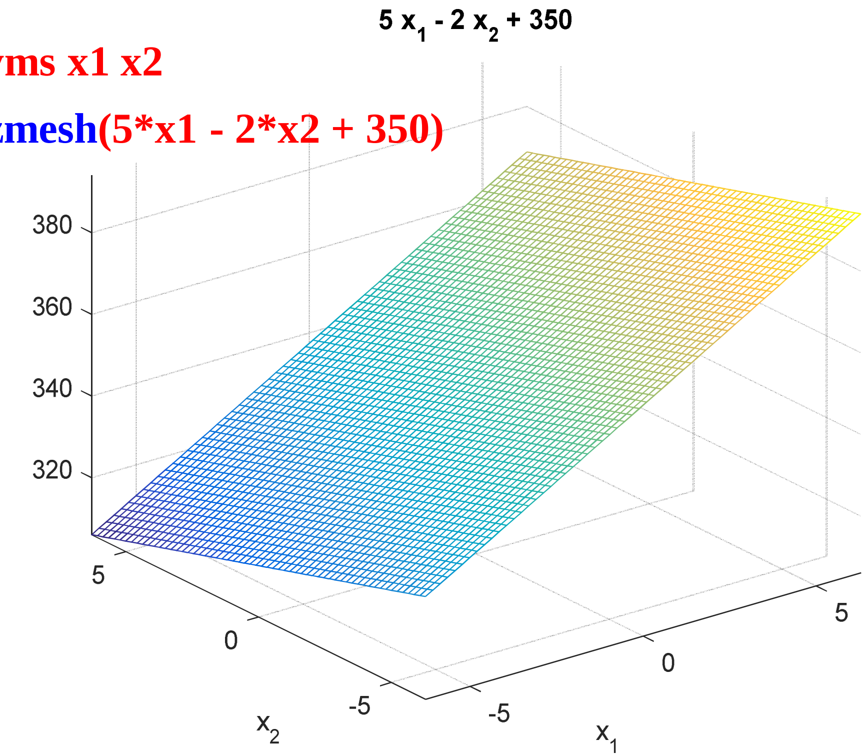
Visualizing LR Results in Matlab

- $h_\theta(x) = \hat{y} = \theta^T \boldsymbol{X} = \theta_0 \times \boldsymbol{x}_0 + \theta_1 \times \boldsymbol{x}_1 + \ldots + \theta_n \times \boldsymbol{x}_n$
  - with $\theta_0 = \mathbf{350}$, $\theta_1 = 5$, $\theta_2 = \mathbf{-2}$.

with $\theta_0 = \mathbf{0}$, $\theta_1 = \mathbf{1}$, $\theta_2 = \mathbf{1}$.

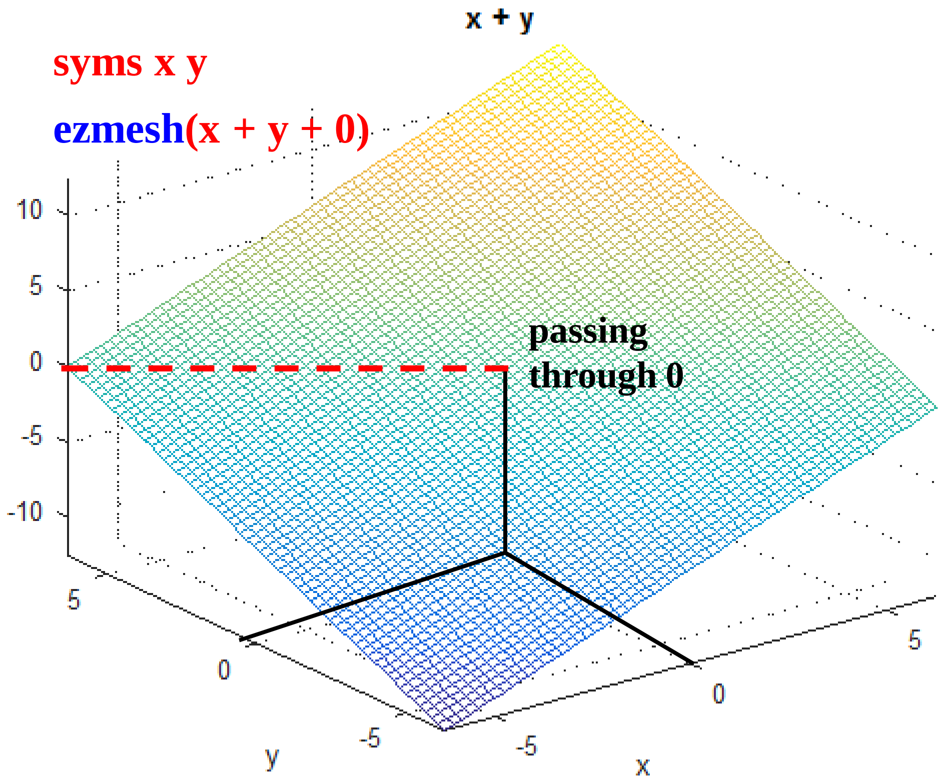
syms x1 x2

ezmesh(5*x1 - 2*x2 + 350)



syms x y

ezmesh(x + y + 0)



[X,Y] = meshgrid(-5:0.1:5, -5:0.1:5);

Z=X+Y;

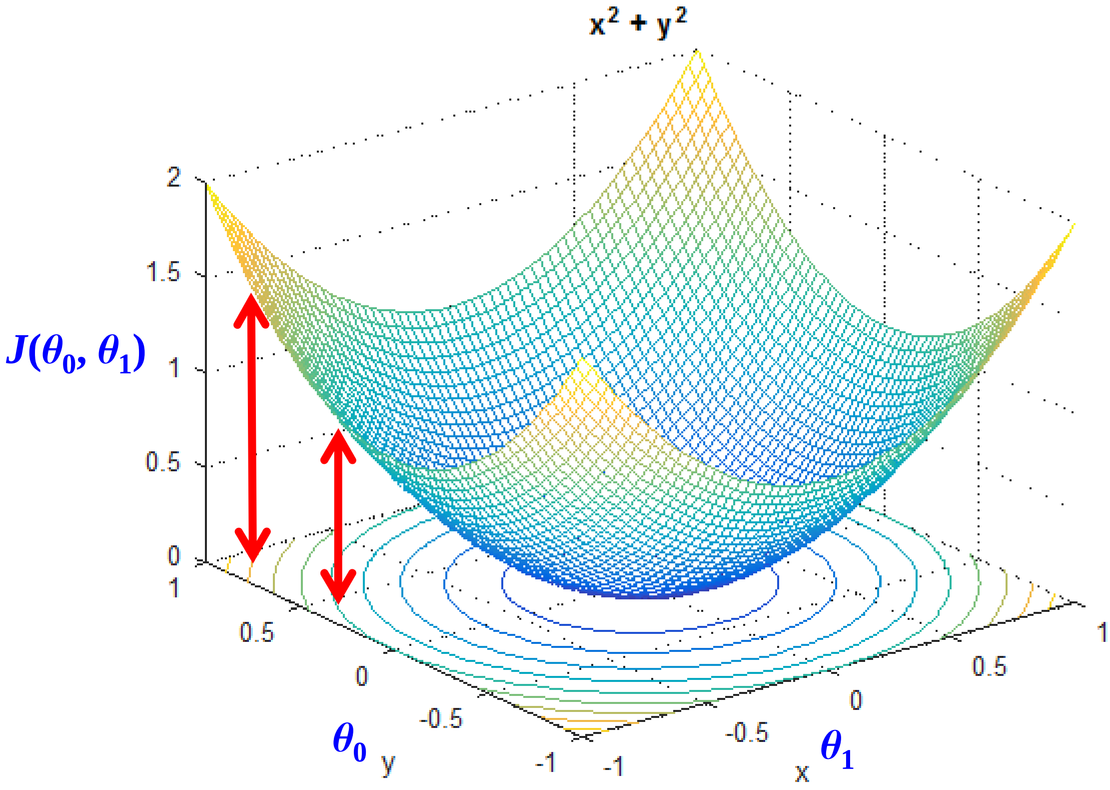
figure, mesh(X,Y,Z)

http://mathesaurus.sourceforge.net/octave-r.html

Learning Cost Function with 2 Predictors

- Gradient Descent on θ_1 and θ_0.

- How to plot a surface in Matlab?
  - With contour on the bottom.
  - **NOT** just make it fancier.




```
syms x y
figure
ezsurf(x^2 + y^2, [-1, 1])

figure
ezmeshc(x^2 + y^2, [-1, 1])

figure
ezmeshc(x^2 + y^2, [-1, 1, -0.5, 1.5])

figure
ezcontour(x^2 + y^2, [-1, 1])

figure
ezcontourf(x^2 + y^2, [-1, 1])
```

Representing Data as a Surface in Matlab

- Representing Data as a Surface
  - http://www.mathworks.com/help/matlab/visualize/representing-a-matrix-as-a-surface.html

Function	Used to Create
mesh, surf	Surface plot
meshc, surfc	Surface plot with contour plot beneath it
meshz	Surface plot with curtain plot (reference plane)
pcolor	Flat surface plot (value is proportional only to color)
surfl	Surface plot illuminated from specified direction
surface	Low-level function (on which high-level functions are based) for creating surface graphics objects

Function	Used to Create
meshgrid	Rectangular grid in 2-D and 3-D space
griddata	Interpolate scattered data
griddedInterpolant	Interpolant for gridded data
scatteredInterpolant	Interpolate scattered data

Gradient Descent on **Your Own Cost Function**

- Define your cost function first.
  - Cost function = $J(\theta) = \frac{1}{2m}\sum_{j=1}^{m}(y_j - h_\theta(x_j))^2$.

- Pass **your cost function** to a Matlab **fminunc()** to automate the GD process.
  - Find minimum of unconstrained multivariable function using BFGS Quasi-Newton method
  - See next slide.

R package – "trust"
https://cran.r-project.org/web/packages/trust/index.html
https://cran.r-project.org/web/views/Optimization.html

```
function [Cost_J gradient] = Your_Cost_Func(X, y, alpha, lambda, theta)
   [n m] = size(X);

   theta_T_X = theta' * X;    % hypothesis of "theta0 + theta1 * x"
   Err = theta_T_X - y;        % residual
   Cost_J = (1 / (2 * m)) * sum(Err.^2);       % Cost from MSE

   % next, compute partial derivate
   gradient(1) = (alpha * (1 / m) * sum(Err));
   gradient(2) = (alpha * (1 / m) * sum(Err * X'));
end
```

repeat until convergence for j = 0 and j = 1:

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

repeat until convergence for j = 0 and j = 1:

$$\{ \quad \theta_0 = \theta_0 - \alpha \frac{1}{m}\sum_{j=1}^{m}\left(y_j - h_\theta(x_j)\right) \times x_0$$

derivative

$$\theta_1 = \theta_1 - \alpha \frac{1}{m}\sum_{j=1}^{m}(y_j - h_\theta(x_j)) \times x_1$$

$\}$

Gradient Descent using Default Matlab Function

- Pass your cost function to a Matlab **fminunc()** to automate the GD process.
  - Find minimum of unconstrained multivariable function using BFGS Quasi-Newton method.
  - **http://www.mathworks.com/help/optim/ug/fminunc.html**

```
X = [1 2 3];
x0 = ones(1, size(X, 2));   % # of records (data points)
X=[x0; [1 2 3]]
Y = [4 8 12];
[n, m] = size(X);   % n = # features,   m = # of data points

% set optimization options
options = optimset('GradObj', 'on', 'MaxIter', 10000, ...
      'TolFun', 1e-10, 'Display', 'iter');
theta0 = zeros(n, 1);
alpha = 0.3;                lambda = 0;

% http://www.mathworks.com/help/optim/ug/fminunc.html
[optTheta, functionVal, exitflag, output, grad] = ...
    fminunc(@(t) LR_fminunc(X, Y, alpha, lambda, t), theta0, options)
```

Pass your cost function here.

Iteration	f(x)	Norm of step	First-order optimality	CG-iterations
0	37.3333		8	
1	4.70129	10	1.43	1
2	3.12966	1	0.166	1
3	2.35436	1	0.203	1
4	1.66409	1	0.201	1
5	1.09206	1	0.19	1
6	0.641035	1	0.18	1
7	0.311171	1	0.169	1
8	0.102481	1	0.158	1
9	0.0149657	1	0.147	1
10	0.0149657	0.892161	0.147	1
11	0.000698877	0.22304	0.0336	0
12	0.000698877	0.0859564	0.0336	1
13	0.00015845	0.0214891	0.0028	0
14	2.75879e-05	0.0429782	0.00624	1
15	2.75879e-05	0.0386887	0.00624	1
16	1.276e-06	0.00967219	0.00146	0

optTheta =

0.0000

4.0000

functionVal =1.3281e-12

Partial Differential (of 2 variables) in Mathematica

$J(\theta) = \frac{1}{2m}\sum_{j=1}^{m}(y_j - h_\theta(x_j))^2$

- D[(1/(2m)) * ((T0*x0 + T1*x1) - y)^2, T0]

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

```
In[5]:= D[(1/(2 m)) * ((T0 * x0 + T1 * x1) - y)^2, T0]
```

$$\frac{x0\,(T0\,x0 + T1\,x1 - y)}{m}$$

$$\theta_0 = \theta_0 - \alpha\frac{1}{m}\sum_{j=1}^{m}\left(y_j - h_\theta(x_j)\right) \times x_0$$

- [(1/(2m)) * ((T0*x0 + T1*x1) - y)^2,T1]

```
In[10]:= D[(1/(2 m)) * ((T0 * x0 + T1 * x1) - y)^2, T1]
```

Out[10]= $$\frac{x1\,(T0\,x0 + T1\,x1 - y)}{m}$$

$$\theta_1 = \theta_1 - \alpha\frac{1}{m}\sum_{j=1}^{m}(y_j - h_\theta(x_j)) \times x_1$$

- https://reference.wolfram.com/language/tutorial/Differentiation.html

Gradient Descent (derivative on multiple attributes)

- Derivative on 2 attributes.

```
In[5]:= D[(1/(2 m)) * ((T0 * x0 + T1 * x1) - y) ^2, T0]
```

$$\frac{x0\,(T0\,x0 + T1\,x1 - y)}{m}$$

```
In[10]:= D[(1/(2 m)) * ((T0 * x0 + T1 * x1) - y) ^2, T1]
```

Out[10]= $\frac{x1\,(T0\,x0 + T1\,x1 - y)}{m}$

repeat until convergence for all ***j***s:

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

derivative

repeat until convergence for ***j*** = 0 and ***j*** = 1:

$$\{ \quad \theta_0 = \theta_0 - \boldsymbol{\alpha} \frac{1}{m} \sum_{j=1}^{m} \left(y_j - h_\theta(x_j)\right) \times x_0$$

$$\theta_1 = \theta_1 - \boldsymbol{\alpha} \frac{1}{m} \sum_{j=1}^{m} (y_j - h_\theta(x_j)) \times x_1$$

$$\}$$

- Derivative on 3 attributes.

```
In[1]:= D[(1/(2 m)) * ((T0 * x0 + T1 * x1 + T2 * x2) - y) ^2, T0]
```

Out[1]= $\frac{\boxed{x0}\,(T0\,x0 + T1\,x1 + T2\,x2 - y)}{m}$

NOTE: X_0 = 1

```
In[2]:= D[(1/(2 m)) * ((T0 * x0 + T1 * x1 + T2 * x2) - y) ^2, T1]
```

Out[2]= $\frac{\boxed{x1}\,(T0\,x0 + T1\,x1 + T2\,x2 - y)}{m}$

```
In[3]:= D[(1/(2 m)) * ((T0 * x0 + T1 * x1 + T2 * x2) - y) ^2, T2]
```

Out[3]= $\frac{\boxed{x2}\,(T0\,x0 + T1\,x1 + T2\,x2 - y)}{m}$

Gradient Descent Illustration

- https://reference.wolfram.com/language/tutorial/Differentiation.html
- $J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2 \quad \text{for } \theta = \begin{bmatrix} \theta 1 \\ \theta 2 \end{bmatrix}$
- $\frac{\partial}{\partial \theta_1} J(\theta) = 2(\theta_1 - 5)$
- $\frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5)$

D[(x-5)^2+(y-5)^2, x] $\quad \frac{\partial}{\partial \theta_1} J(\theta)$

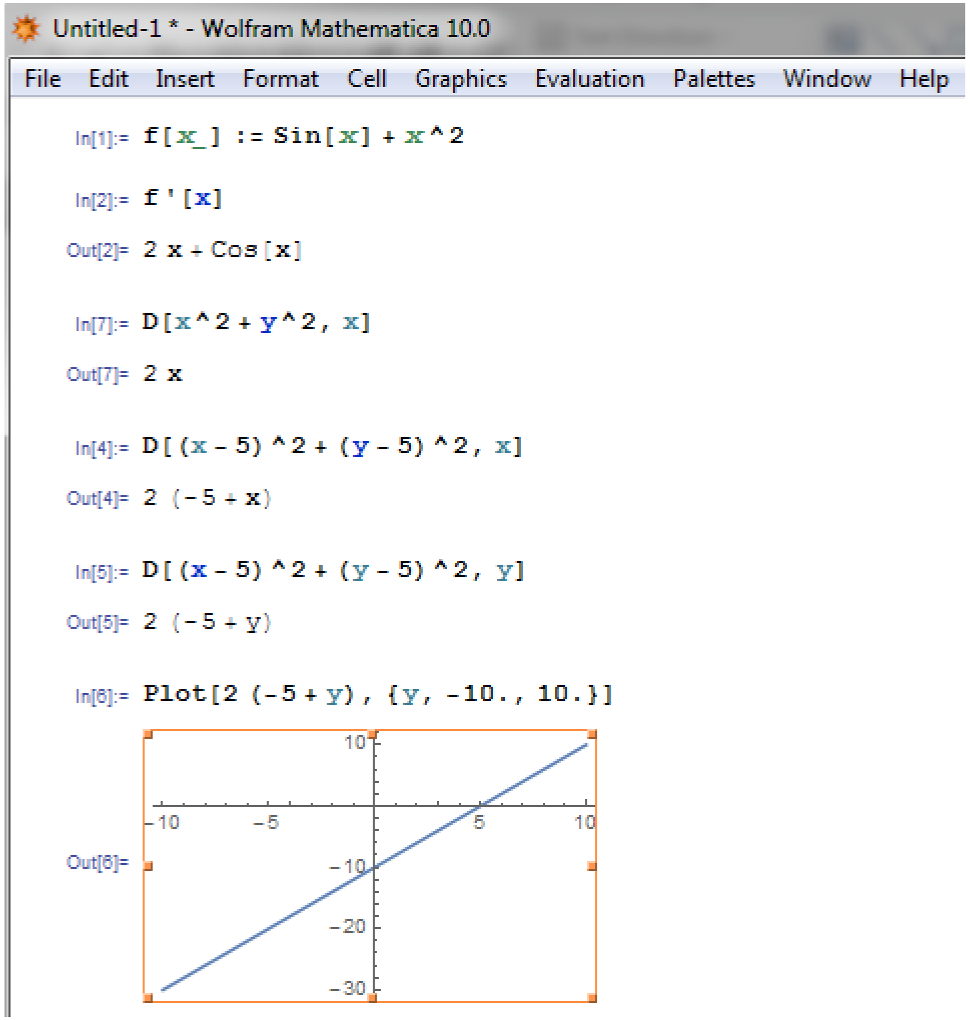
D[(x-5)^2+(y-5)^2, y] $\quad \frac{\partial}{\partial \theta_2} J(\theta)$

Plot[2 (-5+y),{y,-10.,10.}]

f[x_]:=Sin[x]+x^2

f'[x]

D[x^2+y^2,x]



Other Optimization Algorithms

- Gradient descent is one optimization method to minimize cost function $\boldsymbol{J}(\theta)$.

- Other more complex methods for **much larger** machine learning problem:
  - Conjugate gradient.
  - BFGS.
  - L-BFGS.

- Matlab Optimization Toolbox functions:
  - **fminsearch()**
    - Find minimum of unconstrained multivariable function using derivative-free method.
  - **fminunc()**
    - Find minimum of unconstrained multivariable function using BFGS Quasi-Newton method.

Another Method– Normal Equation

$$\theta = (X^T X)^{-1} X^T y$$

- Gradient descent **works well on large dataset**.

- Normal equation $(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$
  - **NO** need to do feature scaling.
  - **No** need to choose α, no need for iteration, slow for large dataset.

- Normal equation works for regression, but **NOT** work for classification.
  - Need to use gradient descent for LR w/ large features, and for classification.

Normal Equation

Given a matrix equation

http://mathworld.wolfram.com/NormalEquation.html

$$\mathsf{A}\,\mathbf{x} = \mathbf{b},$$

the normal equation is that which minimizes the sum of the square differences between the left and right sides:

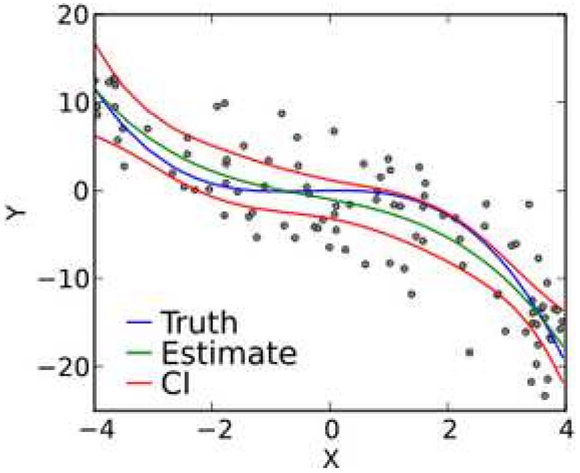
$$\mathsf{A}^{\mathrm{T}}\,\mathsf{A}\,\mathbf{x} = \mathsf{A}^{\mathrm{T}}\,\mathbf{b}.$$

It is called a normal equation because $\mathbf{b} - \mathsf{A}\,\mathbf{x}$ is normal to the range of A.

Here, $\mathsf{A}^{\mathrm{T}}\,\mathsf{A}$ is a normal matrix.

For Gradient Descent Only? Also for Polynomial Regression

- Model *y* as an *n*th degree polynomial, yielding the polynomial regression model.
- Feature scaling becomes more important when using polynomial regression.
  - New features (i.e. $\boldsymbol{x}^3$) grow fast from the original features.



http://en.wikipedia.org/wiki/Polynomial_regression

$y = a_0 + a_1 x + \varepsilon$ **LR**

$y = a_0 + a_1 x + a_2 x^2 + \varepsilon$ **polynomial regression**

$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n + \varepsilon$

http://en.wikipedia.org/wiki/Polynomial_regression

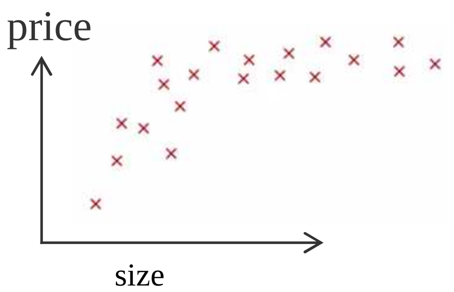
$$\theta_0 + \theta_1 x + \theta_2 x^2$$

$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

$$\theta_0 + \theta_1 (size) + \theta_2 \sqrt{(size)}$$

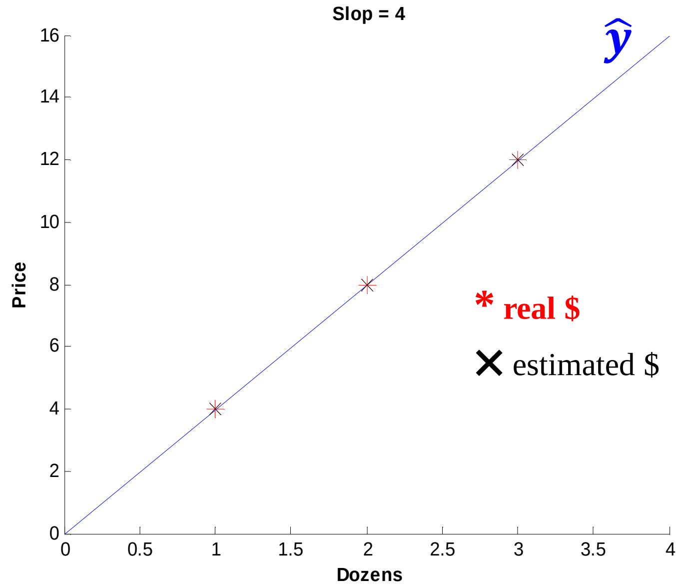
$$\begin{aligned} h_\theta(x) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \\ &= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3 \end{aligned}$$

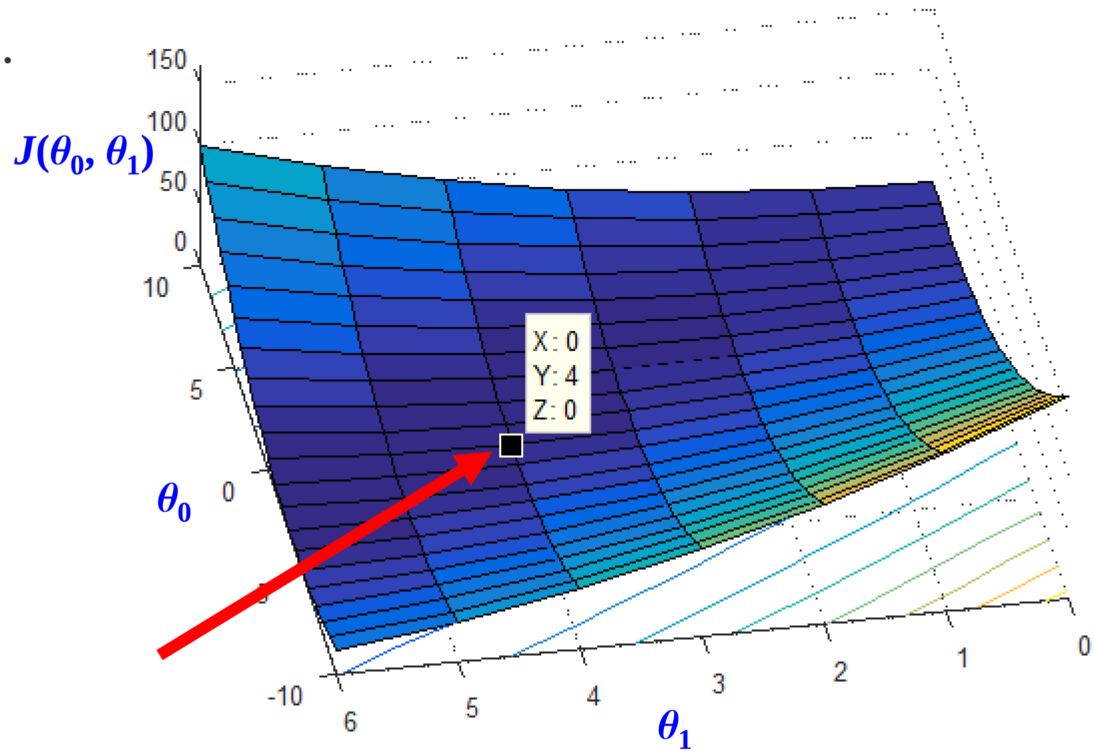
$$x_1 = (size)$$
$$x_2 = (size)^2$$
$$x_3 = (size)^3$$



Gradient Descent on Both θ_1 and θ_0

- For our healthcare cost example.





repeat until convergence for ***j*** = 0 and ***j*** = 1:

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

repeat until convergence for ***j*** = 0 and ***j*** = 1:

derivative

$$\{\ \theta_0 = \theta_0 - \alpha \frac{1}{m} \sum_{j=1}^{m} (y_j - h_\theta(x_j)) \times x_0$$

NOTE: X_0 = 1

$$\theta_1 = \theta_1 - \alpha \frac{1}{m} \sum_{j=1}^{m} (y_j - h_\theta(x_j)) \times x_1$$

$$\}$$

Gradient Descent on θ_1 Only

```
X = [1 2 3];
x0 = ones(1, size(X, 2));   % size(X, 2) = # of records (data points)
X=[x0; [1 2 3]]
Y = [4 8 12];

[n, m] = size(X);   % n = # features,   m = # of data points

theta_Vec = [0; 0];
loops = 0;
theta_Collect = [];
Cost_J = [];
for theta_change = 0 : 0.1 : 8       % change θ from 0 to 8
   theta_Vec(2) = theta_change;
   theta_Collect(end + 1) = theta_change;  % record all θs for plot
   loops = loops + 1;
   theta_T_X = theta_Vec' * X;   % loop of “theta0 + theta1 * x”
   Err = theta_T_X - Y;          % compute Residual
   Cost_J(end + 1) = (1 / 2 * m) * sum(Err.^2);   % compute COST
   if Err > 10^6                 % check converge or diverge
      disp(['Looks like it is diverging at loop ' num2str(loops) '.'])
      break
   end
end

figure,                          % plot cost over θs
plot(theta_Collect, Cost_J)
xlabel('\bf value for theta1')
ylabel('\bf Cost J(theta1)'), grid on
title('\bf Cost vs theta1 (theta0 = 0)')
```