**ROCHESTER INSTITUTE OF TECHNOLOGY**

**DEPARTMENT OF COMPUTER ENGINEERING**

**CMPE 677 Machine Intelligence**

**HW #2   
  
Name:\_\_\_\_\_Kaiwen Zheng\_\_\_\_\_\_\_\_\_\_**

**Due 11:55pm, 09/13/2017, submit via Dropbox. Work alone. All questions pertain to Matlab/Octav.**

1. (6 pts) We can use gradient descent with a least squares cost function in place of the normal equations method to fit highly non-linear functions in very high dimensions. Given a very small learning parameter, , along with an arbitrary starting position, can we guarantee we can eventually converge to an optimal solution? Please explain in a few words why yes or no.

**Yes, because gradient descent for a least squares function is a convex curve which means there is only one minima. Given a very small learning rate and sufficient number of iterations, gradient descent will converge to an optimal solution.**

2. (6 pts) Please check which statements are true regarding least squares gradient descent solving for linear regression parameters 0 and 1:

1. If 0 and 1 are initialized to a local minimum, the first step will take them away from this local minimum
2. If our cost is increasing, we most likely need to increase .
3. No matter how 0 and 1 are initialized, as long as  is sufficiently small, we will converge to the global minimum

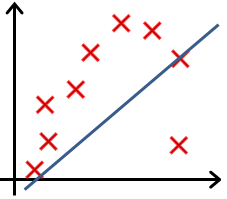
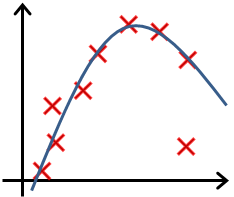
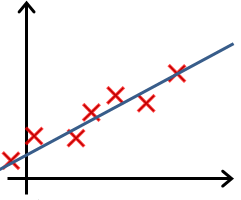
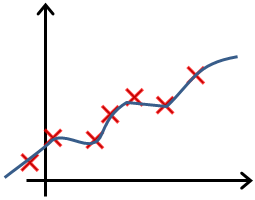
**Answer: c**

3. (6 pts) Suppose that for some linear regression problem (say, predicting life expectancy given cigarettes smoked per day as in the lecture), we have some training set, and for our training set we managed to find some *θ*0, *θ*1 such that *J*(*θ*0,*θ*1)=0. Which of the statements below must then be true? (Select all that apply.).

1. This is not possible: By the definition of *J*(*θ*0,*θ*1), it is not possible for there to exist *θ*0 and *θ*1 so that *J*(*θ*0,*θ*1)=0**;**
2. Our training set can be fit perfectly by a straight line, i.e., all of our training examples lie perfectly on some straight line
3. We can perfectly predict the value of *y* even for new examples that we have not yet seen. (e.g., we can perfectly predict prices of even new houses that we have not yet seen.)
4. Gradient descent is likely to get stuck at a local minimum and fail to find the global minimum

**Answer: b**

4. (6 pts) Each of the below figures represent how the hypothesis (blue line) compares to the training set (red X). Label the plots as under fit, over fit, or good fit.

1. **; Under fit**
2. **; Good fit**
3. **; Good fit**
4. **; Over fit**

5. (6 pts) You are training a classification model with linear regression. Which of the following statements are true? Indicate true or false for each item.

1. Adding many new features to the model helps prevent overfitting on the training set
2. Introducing regularization to the model always results in equal or better performance on examples not in the training set
3. Adding many new features to the model makes it more likely to overfit the training set
4. Adding a new feature to the model always results in equal or better performance on examples not in the training set

**Anwser: b and c**

6. (6 pts) Which of the following statements about regularization are true? Indicate true or false for each item.

1. Using too large a value of *λ* can cause your hypothesis to overfit the data; this can be avoided by reducing *λ*.**;**
2. Using a very large value of *λ* cannot hurt the performance of your hypothesis; the only reason we do not set *λ* to be too large is to avoid numerical problems**;**
3. Using too large a value of *λ* can cause your hypothesis to underfit the data**;**
4. Elastic net regularization can always find a better than or equal to solution as compared to ridge or lasso regression**;**

**Answer: c and d**

7. (12 pts) Use the following code in matlab:

clear ; close all;

data = load('ex1data1.txt'); % Dataset from Andrew Ng, Machine Learning MOOC

X = data(:, 1);

y = data(:, 2);

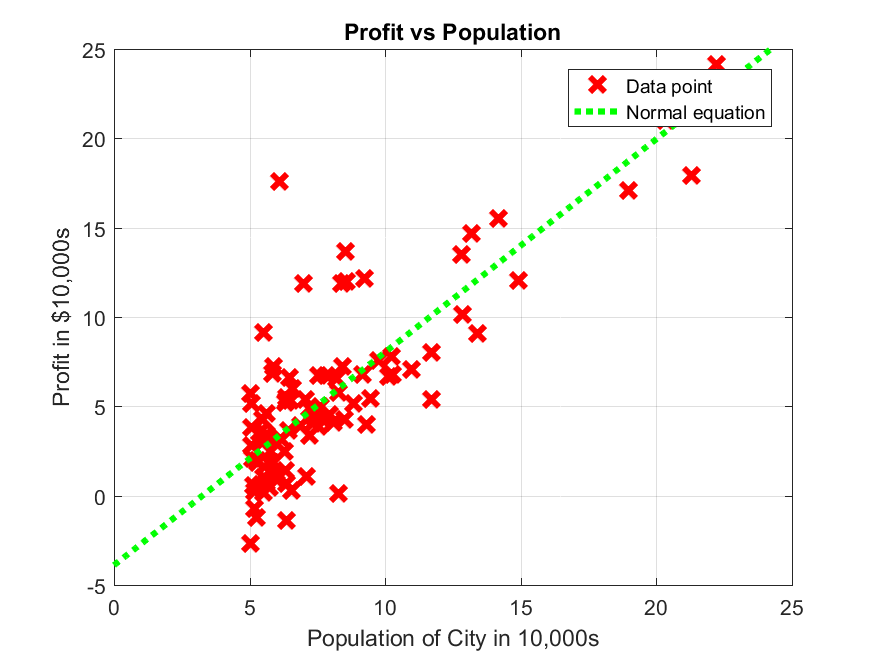
plot(X, y, 'rx', 'MarkerSize', 10,'LineWidth',3); % Plot the data

ylabel('Profit in $10,000s'); % Set the y axis label

xlabel('Population of City in 10,000s'); % Set the x axis label

grid on

Use the normal equations method to solve for a linear line that goes through the data. Show code and include plot here (size=50%) showing the solution as a green dotted line- make sure linewidth is set to 3. x-axis should go from 0 to 25 and y-axis should go from -5 to 25.



figure

data = load('ex1data1.txt'); % Dataset from Andrew Ng, Machine Learning MOOC

X = data(:, 1);

y = data(:, 2);

M = [ones(length(data),1) X];

W = ((M'\*M)\M')\*y;

graphX = 0:0.01:25;

hy = W(1)+graphX.\*W(2); % y=mx+b

plot(X, y, 'rx', 'MarkerSize',10,'LineWidth',3); % Plot the data

hold on

plot(graphX,hy,'g:', 'MarkerSize', 10,'LineWidth',3);

axis([0 25 -5 25])

ylabel('Profit in $10,000s'); % Set the y axis label

xlabel('Population of City in 10,000s'); % Set the x axis label

title('Profit vs Population');

grid on

print('cmpe677\_hwk2\_7','-dpng')

8. (6 pts) We now will compare the normal equation method from problem 7 to gradient descent with the cost function: , where . Before you can implement gradient descent, you need a way to track your performance, which computes J(). Write a matlab function computeCost.m, that takes in three arguments:

*Xdata*, size *n×D*

*Theta*, size *D×1*

*Y*, size *n×1*

Where *n* is the number of samples, and *D* is the dimension of the sample plus 1 (the plus 1 accounts for the constant column: *X* will be 97x2 in this example).

and returns a single overall cost.

The function cannot include any loops (it can be done with one line of code). Show the code here. Do not continue until you get a cost value of 32.0727.

clear ;

data = load('ex1data1.txt'); % Dataset from Andrew Ng, Machine Learning MOOC

X = data(:, 1);

y = data(:, 2);

Xdata = [ones(length(X),1) X];

theta = zeros(2, 1); % initialize fitting parameters to zero

computeCost(Xdata,y,theta);

function cost = computeCost(Xdata, Ydata, theta)

% computes the cost

cost = sum((Ydata-Xdata\*theta).^2)/(2\*length(Ydata));

end

9. (12 pts) Given the computeCost.m from the previous problem, you can now implement gradient descent using the given function gradientDescentLinear.m. Use the following matlab code:

clear ; close all;

data = load('ex1data1.txt'); % Dataset from Andrew Ng, Machine Learning MOOC

X = data(:, 1);

y = data(:, 2);

M = [ones(length(X),1) X];

theta\_init = zeros(2, 1); % initialize fitting parameters to zero

% Some gradient descent settings

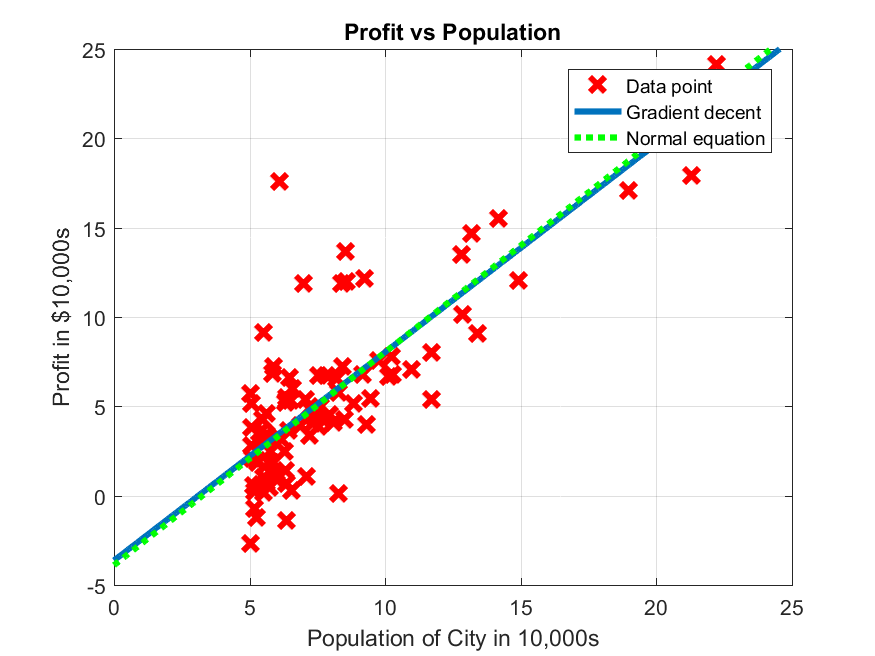
iterations = 1500;

alpha = 0.01;

% run gradient descent

theta = gradientDescentLinear(M, y, theta\_init, alpha, iterations);

Do not continue until you get a theta value of [-3.6303 1.1664]. For this problem, plot this as a blue solid line, linewidth=3, along with the green line from problem 7. Show code and include plot here (size=50%) x-axis should go from 0 to 25 and y-axis should go from -5 to 25.



figure

data = load('ex1data1.txt'); % Dataset from Andrew Ng, Machine Learning MOOC

X = data(:, 1);

y = data(:, 2);

M = [ones(length(X),1) X];

theta\_init = zeros(2, 1); % initialize fitting parameters to zero

% Some gradient descent settings

iterations = 1500;

alpha = 0.01;

% run gradient descent

theta = gradientDescentLinear(M, y, theta\_init, alpha, iterations);

plot(X, y, 'rx', 'MarkerSize', 10,'LineWidth',3); % Plot the data

hold on

W = ((M'\*M)\M')\*y;

graphX = 0:0.01:25;

ly = W(1)+graphX.\*W(2); % y=mx+b

gy = theta(1)+graphX.\*theta(2); % y=mx+b

plot(graphX,gy, 'MarkerSize', 10,'LineWidth',3);

plot(graphX,ly, 'g:', 'MarkerSize', 10,'LineWidth',3);

axis([0 25 -5 25])

ylabel('Profit in $10,000s'); % Set the y axis label

xlabel('Population of City in 10,000s'); % Set the x axis label

title('Profit vs Population');

grid on

print('cmpe677\_hwk2\_9','-dpng')

10. (12 pts) What is the profit for a population of 35K and 70K using the normal equations vs. gradient descent.

Show your code and answer.

data1 = [1, 3.5];

data2 = [1, 7];

linearY1 = data1\*W

gradientY1 = data1\*theta

linearY2 = data2\*W

gradientY2 = data2\*theta

linearY1 = 0.2798

gradientY1 = 0.4520

linearY2 = 4.4555

gradientY2 = 4.5342

11. (6 pts) We have argued that the cost function is convex. Use the below code to visualize the cost surface as a mesh and a contour. No need to show any work for this problem…just understand the code.

% Grid over which we will calculate J

theta0\_vals = linspace(-10, 10, 100);

theta1\_vals = linspace(-1, 4, 100);

% initialize J\_vals to a matrix of 0's

J\_vals = zeros(length(theta0\_vals), length(theta1\_vals));

% Fill out J\_vals

for i = 1:length(theta0\_vals)

for j = 1:length(theta1\_vals)

t = [theta0\_vals(i); theta1\_vals(j)];

J\_vals(i,j) = computeCost(M, y, t);

end

end

% Because of the way meshgrids work in the surf command, we need to

% transpose J\_vals before calling surf, or else the axes will be flipped

J\_vals = J\_vals';

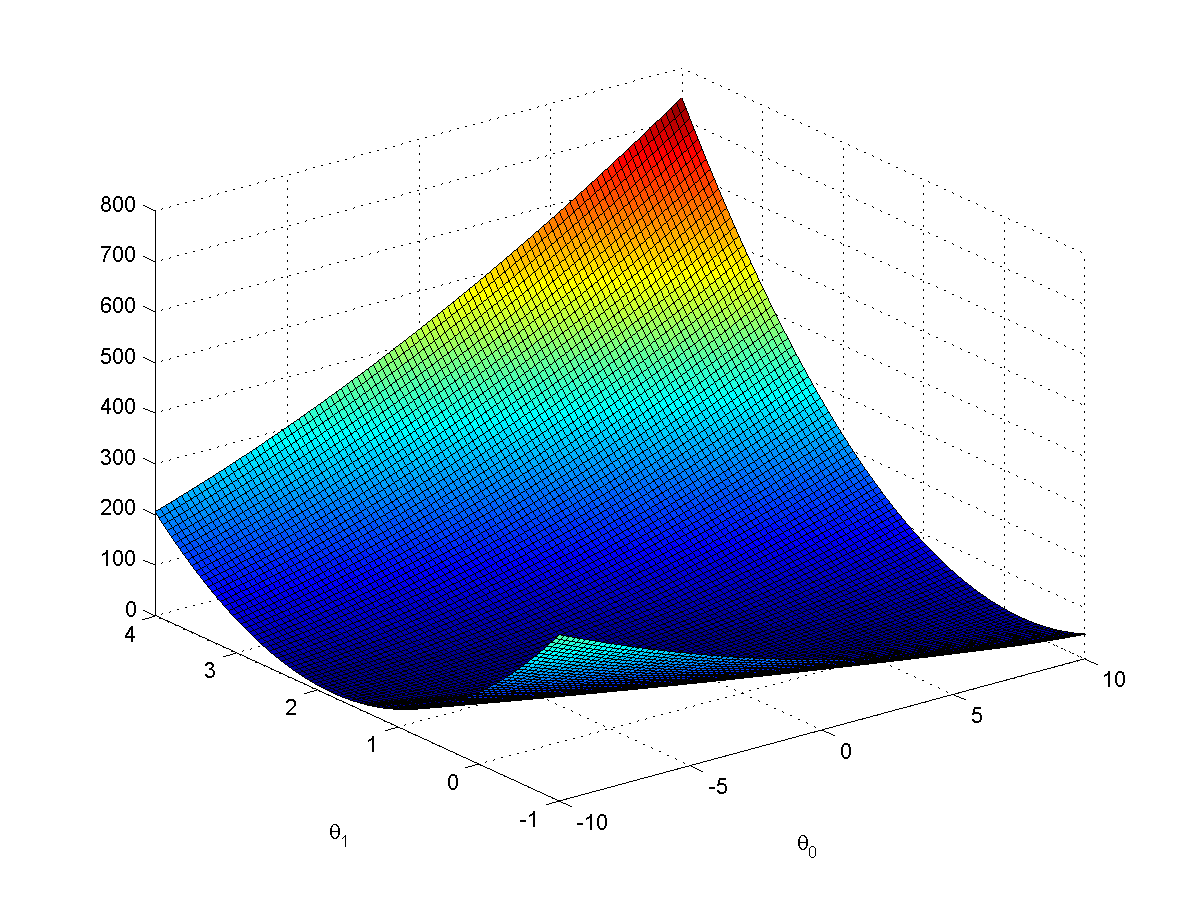
% Surface plot

figure;

surf(theta0\_vals, theta1\_vals, J\_vals)

xlabel('\theta\_0'); ylabel('\theta\_1');

**print -dpng surfaceCost.png**



% Contour plot

figure;

% Plot J\_vals as 15 contours spaced logarithmically between 0.01 and 1000

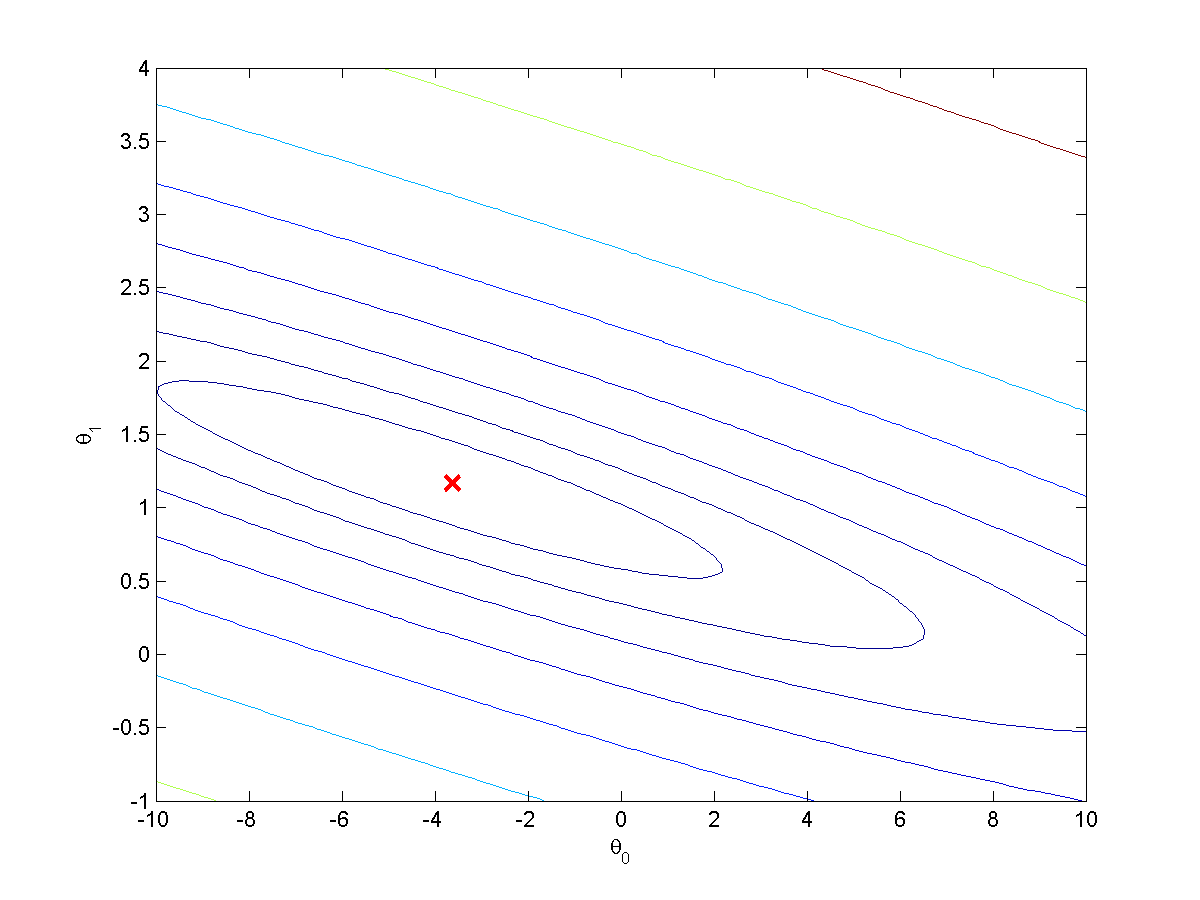
contour(theta0\_vals, theta1\_vals, J\_vals, logspace(-2, 3, 20))

xlabel('\theta\_0'); ylabel('\theta\_1');

hold on;

plot(theta(1), theta(2), 'rx', 'MarkerSize', 10, 'LineWidth', 2);

**print -dpng surfaceContour.png**



Read the code and understood it.

12. (16 pts) We will now experiment with data with more than one dimension. Copy gradientDescentLinear.m to gradientDescentMulti.m. Modify gradientDescentMulti.m such that it can handle an arbitrary number of xData dimensions. Execute the matlab code:

clear ; close all;

%% Load Data

data = load('ex1data2.txt');

X = data(:, 1:2);

y = data(:, 3);

% Scale features and set them to zero mean with std=1

% Write a function featureNormalize.m which computes  
% the mean and std of X, then returns a normalized version

% of X, where we substract the mean form each feature,

% then scale so that std dev = 1

[Xnorm mu sigma] = featureNormalize(X);

% Add intercept term to X

Xdata = [ones(length(X),1) Xnorm];

% Choose some alpha value

alpha = 0.01;

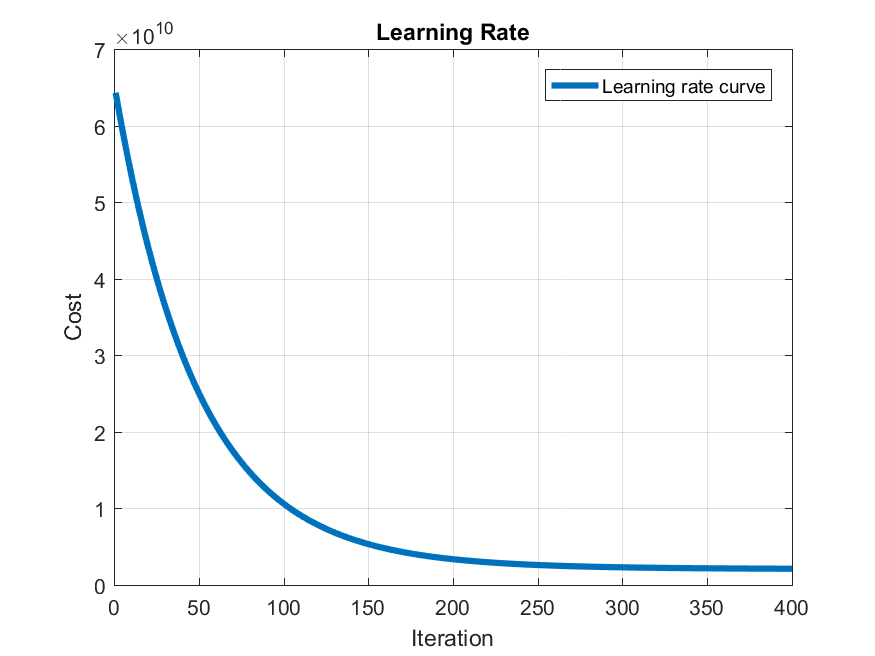
num\_iters = 400;

% Init Theta and Run Gradient Descent

theta = zeros(3, 1);

[theta, J\_history] = gradientDescentMulti(Xdata, y, theta, alpha, num\_iters);

Plot the convergence graph of the cost function over the 400 iterations- include at 50% size. Do you think the learning parameter is too low, too high, or just about right? Show the value of theta for gradient descent as well as the normal equations.

****

function [theta, J\_history] = gradientDescentMulti(Xdata, y, theta, alpha, num\_iters)

n = length(y); % number of training examples

J\_history = zeros(num\_iters, 1);

temp = zeros(1,size(theta,1));

for iter = 1:num\_iters

for i = 1:size(theta,1);

temp(i) = theta(i) - (alpha/n)\*sum((Xdata\*theta-y).\*Xdata(:,i));

end

theta = temp';

% Save the cost J in every iteration

J\_history(iter) = computeCost(Xdata, y, theta);

end

end

**The learning rate shown above is about right because the cost decreases over time and reaches a flat line.**

**Theta for gradient descent**

theta = 1.0e+05 \*

3.3430

1.0009

0.0367

**Theta for gradient descent**

theta = 1.0e+05 \*

3.4041

1.1063

-0.0665