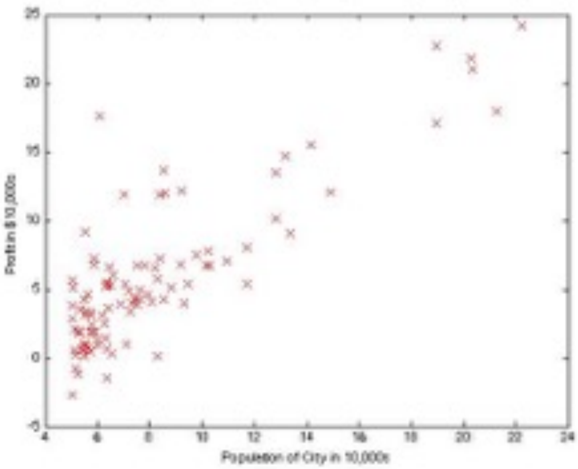
Chris TANG
Andrew Ng
Machine Learning
15/2/2

Programming Exercise 1

2 Linear regression with one variable



2.2.3 Computing the cost J(θ)

Your next task is to complete the code in the file computeCost.m, which is a function that computes J(θ). As you are doing this, remember that the variables X and y are not scalar values, but matrices whose rows represent the examples from the training set.

computeCost.m

function J = computeCost(X, y, theta)

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

J = 0;

% Instructions: Compute the cost of a particular choice of theta

% You should set J to the cost.

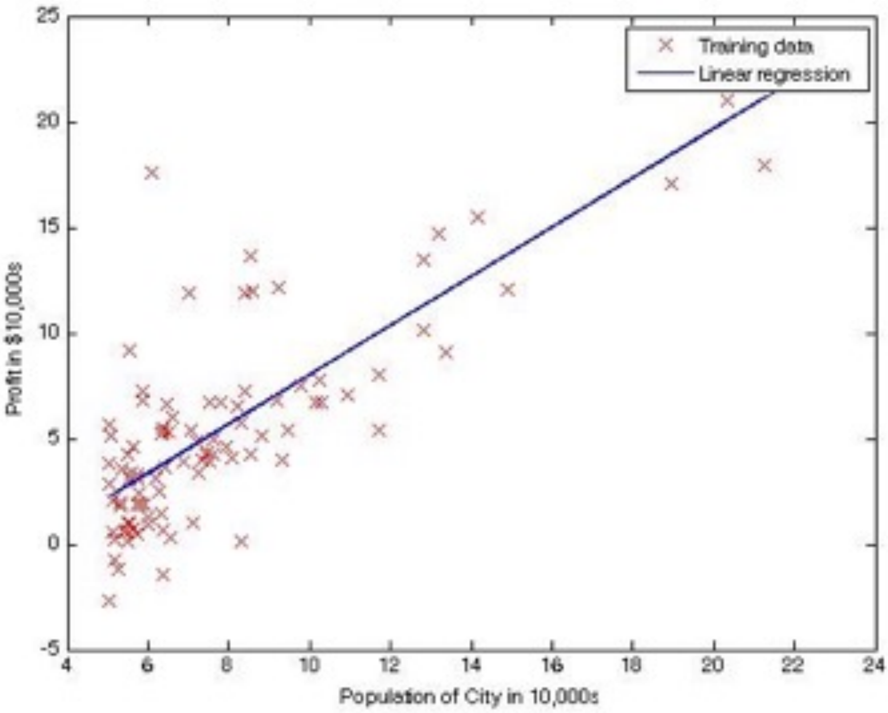
htheta = X*theta;% **htheta is vector of m*2 * 2*1 --> m*1 In this we have to compute the hypothesis h(x).**

cost = htheta - y;

powcost = power(cost,2);% add all to J

J = sum(powcost)/(2*m);

end



**Please noticed here:

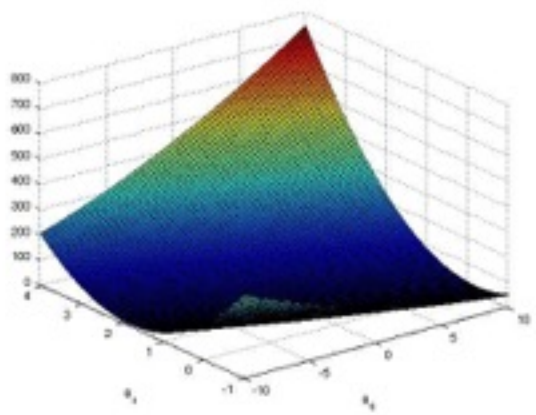
data: m by 2 X: m by 2 (1 column with all one is added to the left)
Y: m by 1

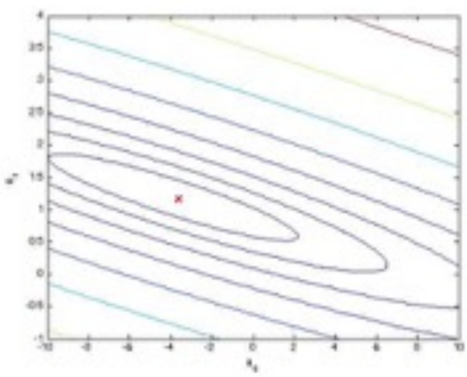
2.2.4 Gradient descent

A good way to verify that gradient descent is working correctly is to look at the value of J(θ) and check that it is decreasing with each step.

```
function [theta, J_history] = gradientDescent(X, y, theta, alpha, num_iters)

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

J_history = zeros(num_iters, 1); Each time the J_history is updated

for iter = 1:num_iters % inside each iteration, we have to update the value of theta according to the

  % ====================== YOUR CODE HERE ======================

  htheta = X*theta;% htheta is vector of m*2 * 2*1 --> m*1

  cost = htheta - y;
```

```
    temp0 = cost'*X(:,1);

    theta0 = theta(1) - alpha/m * temp0;

    temp1 = cost'*X(:,2);

    theta1 = theta(2)- alpha/m * temp1;

    theta = [theta0;theta1];

   % ============================================================

    % Save the cost J in every iteration

    J_history(iter) = computeCost(X, y, theta);

end

end
```





These two figures above show J(θ) various based on various θ0 and θ1.

Extra Credit Exercises.

3.1 Feature Normalization

1.Subtract the mean value of each feature from the dataset.

2.After subtracting the mean, additionally scale (divide) the feature values by their respective "standard deviations."

```
function [X_norm, mu, sigma] = featureNormalize(X)

X_norm = X; % X is a m*3 matrix(47*3)

mu = zeros(1, size(X, 2));% mu is a 1*3

sigma = zeros(1, size(X, 2)); % sigma is a 1*3

% ====================== YOUR CODE HERE ======================

% Following code is edited by Chris TANG

  mu = mean(X); % by column, mu stores the mean of X.

  sigma = std(X); % shows the standard deviation

  temp = ones(length(X),1);% please notice the minus only works for same dimension.

  X_norm = (X - (temp*mu))./(temp*sigma);

% ============================================================

end
```

3.2 Gradient Descent

The cost function is same as previous one in one single variable

For gradient descent.

```
function [theta, J_history] = gradientDescentMulti(X, y, theta, alpha, num_iters)

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

J_history = zeros(num_iters, 1);

for iter = 1:num_iters

   % =========== YOUR CODE HERE ===


   % Edited by Chris TANG
```

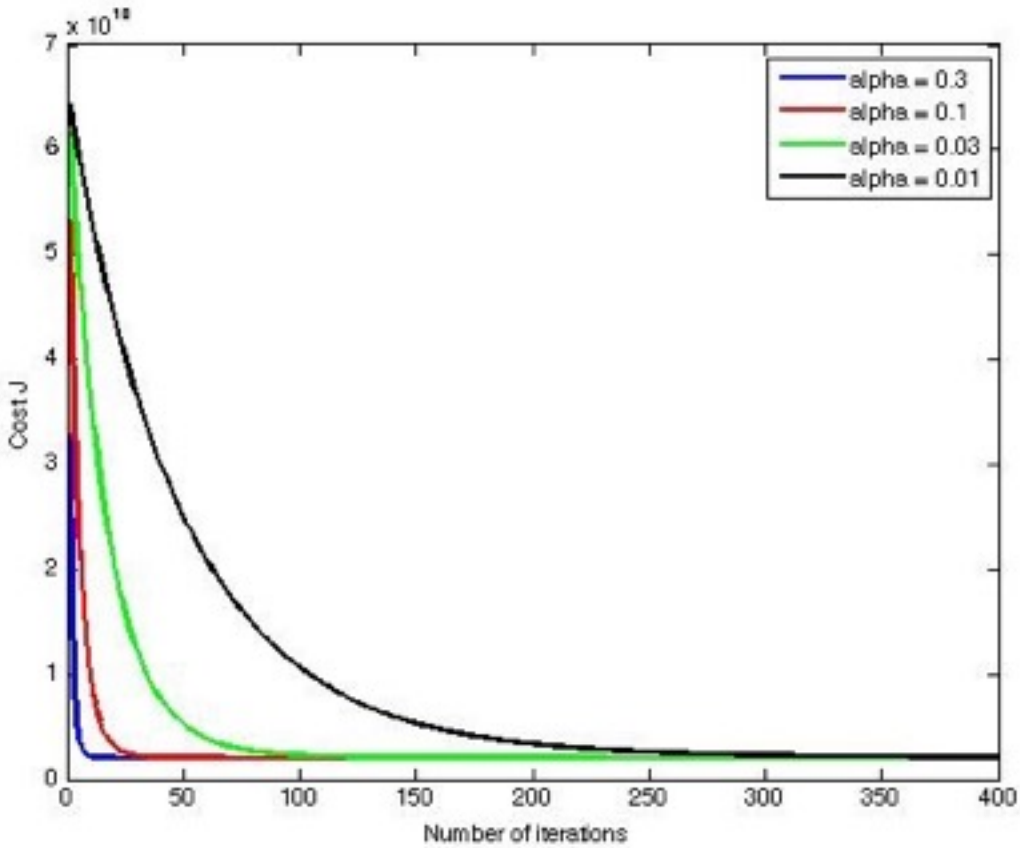
```
    % alpha is a number 1*1
    % m is a number 1*1
    % theta is a n*1(n is number of feature, here n is 3)
    % htheta is a m * 1
    % X is a m by n (n is 3 here)
      htheta = X*theta - y;% m by 1 vector
    [rtheta,ctheta] = size(theta);
    theta = theta - alpha/m * (htheta'*X)';
      % ============================================================
J_history(iter) = computeCostMulti(X, y, theta);


end
end
```

3.2.1 Selecting the learning rate

In **ex1_multi.m**

```
alpha = 0.3;
alpha1 = 0.1;
alpha2 = 0.03;
alpha3 = 0.01;
num_iters = 400;


% Init Theta and Run Gradient Descent
```

```
theta = zeros(3, 1);

[theta, J1] = gradientDescentMulti(X, y, theta, alpha, num_iters);

theta = zeros(3, 1);% please note that here we have to reset the theta

[theta, J2] = gradientDescentMulti(X, y, theta, alpha1, num_iters);

theta = zeros(3, 1);

[theta, J3] = gradientDescentMulti(X, y, theta, alpha2, num_iters);

theta = zeros(3, 1);

[theta, J4] = gradientDescentMulti(X, y, theta, alpha3, num_iters);


% Plot the convergence graph

figure;

plot(1:numel(J1), J1, '-b', 'LineWidth', 2);
```



```
hold on;

plot(1:numel(J2), J2, '-r', 'LineWidth', 2);

plot(1:numel(J3), J3, '-g', 'LineWidth', 2);

plot(1:numel(J2), J4, '-k', 'LineWidth', 2);
```

3.3 Normal Equations

```
theta = (pinv(X'*X))*X'*y;
```