Compute Cost Function

function J = computeCost(X, y, theta)

%COMPUTECOST Compute cost for linear regression

% J = COMPUTECOST(X, y, theta) computes the cost of using theta as the

% parameter for linear regression to fit the data points in X and y

% Initialize some useful values

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

% You need to return the following variables correctly

J = 0;

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

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

% You should set J to the cost.

pred = X\*theta;

err = (pred-y).^2

J = (1/(2\*m))\*sum(err);

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

end

Feature Normalization

function [X\_norm, mu, sigma] = featureNormalize(X)

%FEATURENORMALIZE Normalizes the features in X

% FEATURENORMALIZE(X) returns a normalized version of X where

% the mean value of each feature is 0 and the standard deviation

% is 1. This is often a good preprocessing step to do when

% working with learning algorithms.

% You need to set these values correctly

X\_norm = X;

mu = zeros(1, size(X, 2));

sigma = zeros(1, size(X, 2));

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

% Instructions: First, for each feature dimension, compute the mean

% of the feature and subtract it from the dataset,

% storing the mean value in mu. Next, compute the

% standard deviation of each feature and divide

% each feature by it's standard deviation, storing

% the standard deviation in sigma.

%

% Note that X is a matrix where each column is a

% feature and each row is an example. You need

% to perform the normalization separately for

% each feature.

%

% Hint: You might find the 'mean' and 'std' functions useful.

%

nof = size(X,2);

for x=1:nof,

mu(x) = mean(X(:,x));

sigma(x) = std(X(:,x));

X\_norm(:,x) = (X\_norm(:,x)-mu(x))/sigma(x);

end;

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

end

Gradient Descent

function [theta, J\_history] = gradientDescent(X, y, theta, alpha, num\_iters)

%GRADIENTDESCENT Performs gradient descent to learn theta

% theta = GRADIENTDESCENT(X, y, theta, alpha, num\_iters) updates theta by

% taking num\_iters gradient steps with learning rate alpha

% Initialize some useful values

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

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

for iter = 1:num\_iters

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

% Instructions: Perform a single gradient step on the parameter vector

% theta.

%

% Hint: While debugging, it can be useful to print out the values

% of the cost function (computeCost) and gradient here.

%

pred = X\*theta;

delta = (1/m)\*(X'\*(pred-y));

theta = theta - (alpha\*delta);

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

% Save the cost J in every iteration

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

end

end

Normal Equation

function [theta] = normalEqn(X, y)

%NORMALEQN Computes the closed-form solution to linear regression

% NORMALEQN(X,y) computes the closed-form solution to linear

% regression using the normal equations.

theta = zeros(size(X, 2), 1);

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

% Instructions: Complete the code to compute the closed form solution

% to linear regression and put the result in theta.

%

% ---------------------- Sample Solution ----------------------

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

% -------------------------------------------------------------

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

end

Plot Data

function plotData(x, y)

%PLOTDATA Plots the data points x and y into a new figure

% PLOTDATA(x,y) plots the data points and gives the figure axes labels of

% population and profit.

figure; % open a new figure window

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

% Instructions: Plot the training data into a figure using the

% "figure" and "plot" commands. Set the axes labels using

% the "xlabel" and "ylabel" commands. Assume the

% population and revenue data have been passed in

% as the x and y arguments of this function.

%

% Hint: You can use the 'rx' option with plot to have the markers

% appear as red crosses. Furthermore, you can make the

% markers larger by using plot(..., 'rx', 'MarkerSize', 10);

plot(x, y, 'rx', 'MarkerSize', 10);

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

end

Driver Code

%% Machine Learning Online Class - Exercise 1: Linear Regression

% Instructions

% ------------

%

% This file contains code that helps you get started on the

% linear exercise. You will need to complete the following functions

% in this exericse:

%

% warmUpExercise.m

% plotData.m

% gradientDescent.m

% computeCost.m

% gradientDescentMulti.m

% computeCostMulti.m

% featureNormalize.m

% normalEqn.m

%

% For this exercise, you will not need to change any code in this file,

% or any other files other than those mentioned above.

%

% x refers to the population size in 10,000s

% y refers to the profit in $10,000s

%

%% Initialization

clear ; close all; clc

%% ==================== Part 1: Basic Function ====================

% Complete warmUpExercise.m

fprintf('Running warmUpExercise ... \n');

fprintf('5x5 Identity Matrix: \n');

warmUpExercise()

fprintf('Program paused. Press enter to continue.\n');

pause;

%% ======================= Part 2: Plotting =======================

fprintf('Plotting Data ...\n')

data = load('ex1data1.txt');

X = data(:, 1); y = data(:, 2);

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

% Plot Data

% Note: You have to complete the code in plotData.m

plotData(X, y);

fprintf('Program paused. Press enter to continue.\n');

pause;

%% =================== Part 3: Cost and Gradient descent ===================

X = [ones(m, 1), data(:,1)]; % Add a column of ones to x

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

% Some gradient descent settings

iterations = 1500;

alpha = 0.01;

fprintf('\nTesting the cost function ...\n')

% compute and display initial cost

J = computeCost(X, y, theta);

fprintf('With theta = [0 ; 0]\nCost computed = %f\n', J);

fprintf('Expected cost value (approx) 32.07\n');

% further testing of the cost function

J = computeCost(X, y, [-1 ; 2]);

fprintf('\nWith theta = [-1 ; 2]\nCost computed = %f\n', J);

fprintf('Expected cost value (approx) 54.24\n');

fprintf('Program paused. Press enter to continue.\n');

pause;

fprintf('\nRunning Gradient Descent ...\n')

% run gradient descent

theta = gradientDescent(X, y, theta, alpha, iterations);

% print theta to screen

fprintf('Theta found by gradient descent:\n');

fprintf('%f\n', theta);

fprintf('Expected theta values (approx)\n');

fprintf(' -3.6303\n 1.1664\n\n');

% Plot the linear fit

hold on; % keep previous plot visible

plot(X(:,2), X\*theta, '-')

legend('Training data', 'Linear regression')

hold off % don't overlay any more plots on this figure

% Predict values for population sizes of 35,000 and 70,000

predict1 = [1, 3.5] \*theta;

fprintf('For population = 35,000, we predict a profit of %f\n',...

predict1\*10000);

predict2 = [1, 7] \* theta;

fprintf('For population = 70,000, we predict a profit of %f\n',...

predict2\*10000);

fprintf('Program paused. Press enter to continue.\n');

pause;

%% ============= Part 4: Visualizing J(theta\_0, theta\_1) =============

fprintf('Visualizing J(theta\_0, theta\_1) ...\n')

% 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(X, 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');

% Contour plot

figure;

% Plot J\_vals as 15 contours spaced logarithmically between 0.01 and 100

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);