Ex 1 Submission

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.


	% ====================== 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);


	figure; % open a new figure window


	plot(x, y, 'rx', 'MarkerSize', 10);
	xlabel('Population of City in 10,000s');
	ylabel('Profit in $10,000s');

============================================================
end
============================================================
```

COMPUTE COST

```
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.




	J =  sum((X * theta - y) .^ 2) / (2 * m);

=============================================
end
=============================================
```

GRADIENT DESCENT

```
function [theta, J_history] = gradientDescent(X, y, theta, alpha, num_iters)
	%GRADIENTDESCENT Performs gradient descent to learn theta
	%   theta = GRADIENTDESENT(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.
	    %
```

```
        population = X(:, 2);
        temp_theta = zeros(2, 1);

        temp_theta(1) = theta(1) - alpha * sum(X * theta - y) / m;
        temp_theta(2) = theta(2) - alpha * sum((X * theta - y) .* population) / m;
        theta = temp_theta;

==================================================
        % Save the cost J in every iteration
        J_history(iter) = computeCost(X, y, theta);


    end

    end
==================================================
```

COMPUTE COST MULTI

```
function J = computeCostMulti(X, y, theta)
    %COMPUTECOSTMULTI Compute cost for linear regression with multiple
variables
    %   J = COMPUTECOSTMULTI(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.


    J = sum((X * theta .- y) .^ 2) / (2 * m);
```

```
% ============================================================

end
============================================================

GRADIENT DESCENT MULTI


function [theta, J_history] = gradientDescentMulti(X, y, theta, alpha, num_iters)
    %GRADIENTDESCENTMULTI Performs gradient descent to learn theta
    %   theta = GRADIENTDESCENTMULTI(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 (computeCostMulti) and gradient here.
      %


      theta = theta - (alpha * ((X * theta .- y)' * X))' / m;




      % ============================================================
      % Save the cost J in every iteration
      J_history(iter) = computeCostMulti(X, y, theta);

    end

    end
```

==

FEATURE NORMALIZE

```
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.
    %


    m = size(X, 1);
    mu = mean(X);
    sigma = std(X);


    for i = 1:m
        X_norm(i, :) = (X(i, :) .- mu) ./ sigma;

   end
```

```
==================================================
       end
==================================================
```

NORMALIZE EQN

```
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
==================================================
```