**EX1.m – Multivariate Linear Regression**

%% ================ Part 1: Feature Normalization ================

data = load('ex1data2.txt');

X = data(:, 1:2);

y = data(:, 3);

m = length(y);

fprintf('Normalizing Features ...\n');

[X mu sigma] = featureNormalize(X);

X = [ones(m, 1) X];

%% ================ Part 2: Gradient Descent ================

alpha = .1;

num\_iters = 50;

theta = zeros(3, 1);

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

figure;

plot(1:numel(J\_history), J\_history, '-b', 'LineWidth', 2);

xlabel('Number of iterations');

ylabel('Cost J');

% Estimate the price of a 1650 sq-ft, 3 br house

P\_Val = [1650.,3.];

P\_Norm = zeros(1,2);

for i=1:2

P\_Norm(i) = (P\_Val(i)-mu(i))/sigma(i);

end

P\_Norm = [1 P\_Norm];

price = P\_Norm\*theta;

%% ================ Part 3: Normal Equations ================

data = csvread('ex1data2.txt');

X = data(:, 1:2);

y = data(:, 3);

m = length(y);

X = [ones(m, 1) X];

theta = normalEqn(X, y);

price = [1,1650,3]\*theta;

**FEATURE NORMALIZER**

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.

X\_norm = X;

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

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

dim = size(X);

for i=1:dim(2)

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

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

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

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

m = length(y);

J = 0;

T = (X\*theta - y);

J = sum(T'\*T)\*1/(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

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

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

dim = size(X);

for iter = 1:num\_iters

theta\_temp = theta;

for i=1:dim(2)

theta(i) = theta(i) - alpha/m \* (sum((X\*theta\_temp-y).\*X(:,i)));

end

J\_history(iter) = computeCostMulti(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);

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

end