1. **Linear regression with one variable**

I implement linear regression with 1 variable to predict profits for a food truck. Suppose I am the CEO of a restaurant franchise and I am considering different cities for opening a new outlet. The chain already has trucks in various cities and I have data for profits and populations from the cities.

I would like to use this data to help me select which city to expand to next.

**ex1data1.txt** contains the dataset for my linear regression problem.

The 1st column is the population of a city and 2nd column is the profit of a food truck in that city. A negative value for profit indicates a loss.

* 1. **Plotting the Data**

Before starting on any task, it’s often useful to understand the data by visualizing it.

***Script of plotting:***

*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.*

*% ====================== MY 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); % 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*

end

***script of executing plotData function:***

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

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

data = load('ex1data1.txt');

X = data(:, 1); y = data(:, 2); %e.g x axis has column 1 data

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

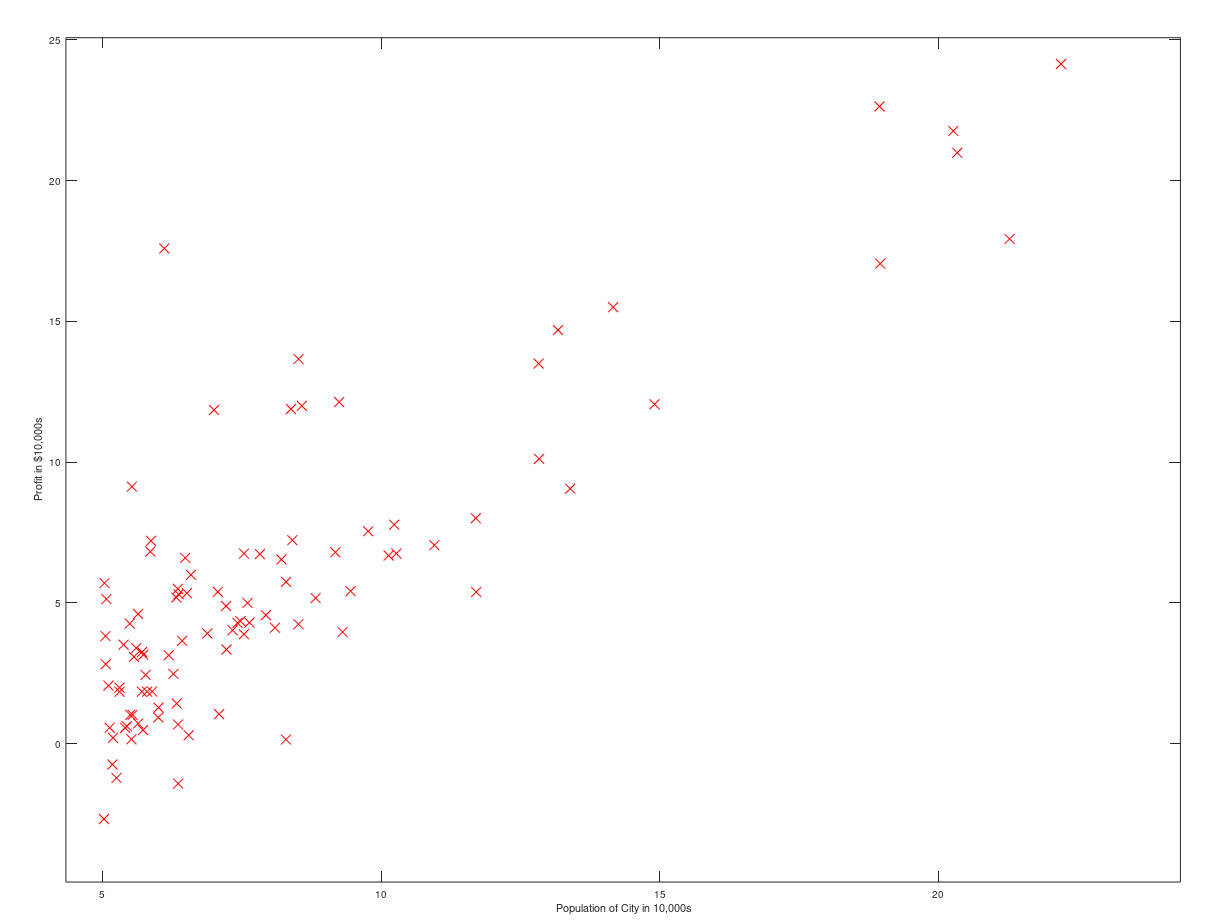
% Plot Data

plotData(X, y);

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

pause;

***Output:***



* 1. **Gradient Descent**

***Script for 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;

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

J = sum((X \* theta - y).^2) / (2\*m); % X(79,2) theta(2,1)

% .^2 is to calculate each element of this X matrix to its power of two

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

end

***Script for 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

% 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.

delta = (1/m)\*sum(X.\*repmat((X\*theta – y), 1, size(X, 2)));

theta = (theta’ – (alpha \* delta))’;

% Save the cost J in every iteration

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

end

end

***Script for executing script of gradient descent:***

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

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

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

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;

%My output is like:

%Testing the cost function ...

%With theta = [0 ; 0]

%Cost computed = 32.072734

%Expected cost value (approx) 32.07

%With theta = [-1 ; 2]

%Cost computed = 54.242455

%Expected cost value (approx) 54.24

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: ');

fprintf('%f %f \n', theta(1), theta(2));

%My output is like:

%Running Gradient Descent ...

%Theta found by gradient descent:

%-3.630291

%1.166362

%Expected theta values (approx)

%-3.6303

%1.1664

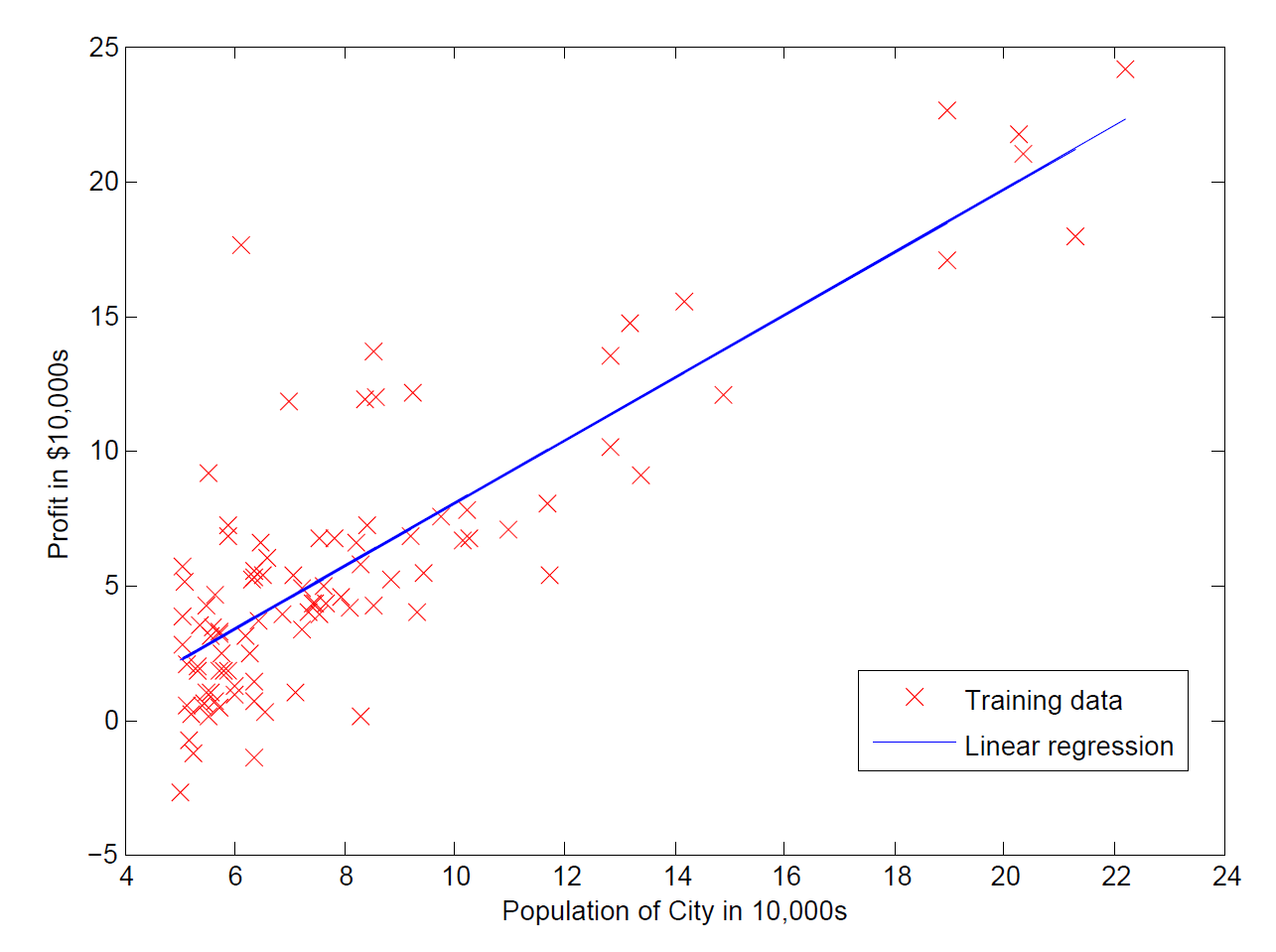
% Plot the linear fit

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;

%My output is like:

% For population = 35,000, we predict a profit of 4519.767868

%For population = 70,000, we predict a profit of 45342.450129

%Program paused. Press enter to continue.

* 1. **Visualizing J**

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

% Grid over which we will calculate J

theta0\_vals = linspace(-10, 10, 100); % theta0\_vals has 100 numbers ranging from -10 to 10. It’s a row vector

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

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

J\_vals = zeros(length(theta0\_vals), length(theta1\_vals)); %create n\*n zero matrix

% Fill out J\_vals

for i = 1:length(theta0\_vals)

for j = 1:length(theta1\_vals)

t = [theta0\_vals(i); theta1\_vals(j)]; %“；”is creating column

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; %create a new plot

surf(theta0\_vals, theta1\_vals, J\_vals)%plot，J is determined by theta0 and theta1

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

% Contour plot

figure;

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

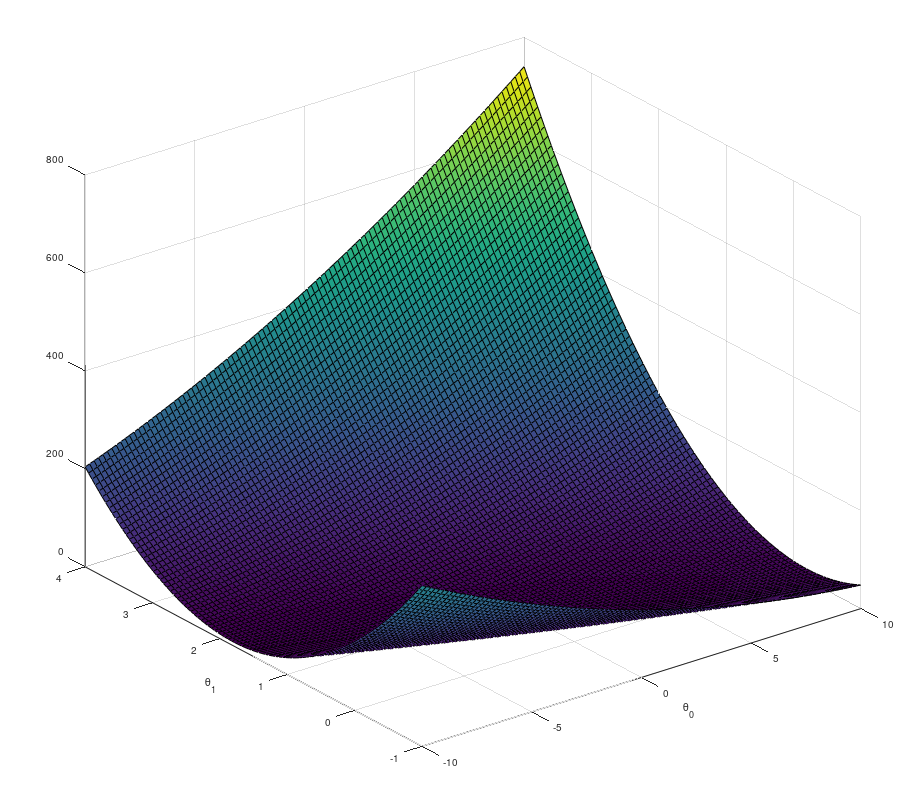
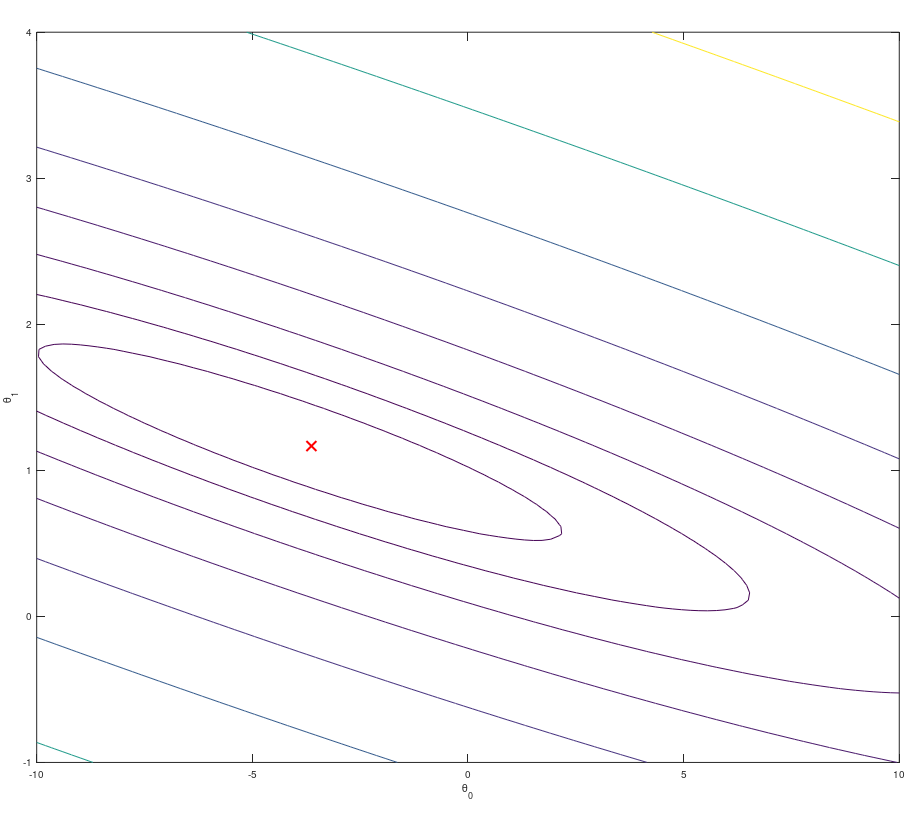
contour(theta0\_vals, theta1\_vals, J\_vals, logspace(-2, 3, 20))

% plot 20 contour lines，each altitude is determined by theta0,theta1,J, ranging from 10^(-2) to 10^3

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

hold on;

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



1. **Linear regression with multiple variable**

I implement linear regression with multiple variables to predict the prices of houses. I am selling my house and I want to know what a good market price would be.

One way to do this is to first collect information on recent houses sold and make a model of housing prices.

The file **ex1data2.txt** contains a training set of housing prices in Port-land, Oregon. The first column is the size of the house (in squar feet).The second column is the number of bedrooms, and the third column is the price of the house.

* 1. **Feature Normalization**

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

%% Clear and Close Figures

clear ; close all; clc

fprintf('Loading data ...\n');

%% Load Data

data = load('ex1data2.txt');

X = data(:, 1:2);

y = data(:, 3);

m = length(y);

% Print out some data points

fprintf('First 10 examples from the dataset: \n');

fprintf(' x = [%.0f %.0f], y = %.0f \n', [X(1:10,:) y(1:10,:)]');

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

pause;

% Scale features and set them to zero mean

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

[X mu sigma] = featureNormalize(X);

% Add intercept term to X

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

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));

% 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.

mu = mean(X);

sigma = std(X);

X\_norm = (X\_norm - mu) ./ sigma;

end

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

End

* 1. **Gradient Descent**

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

% Hint: By using the 'hold on' command, you can plot multiple

% graphs on the same figure.

%

% Choose some alpha value

alpha = 0.01;

num\_iters = 400;

% Init Theta and Run Gradient Descent

theta = zeros(3, 1);

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

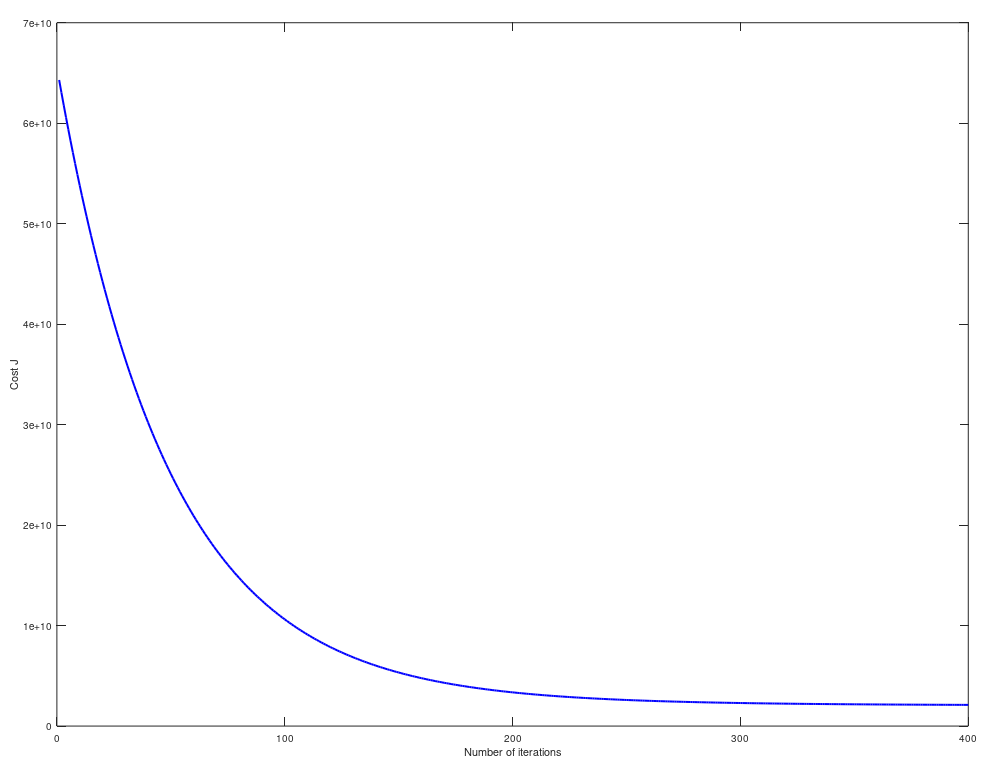
% Plot the convergence graph

figure;

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

xlabel('Number of iterations');

ylabel('Cost J');



% Display gradient descent's result

fprintf('Theta computed from gradient descent: \n');

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

fprintf('\n');

因为在单变量线性回归中，使用的是向量化的计算方法，对于多变量线性回归同样适用。不需要重新写: computeCostMulti.m 和 computCost.m 一样，gradientDescentMulti.m 和gradientDescent.m 一样

% Hint: At prediction, make sure you do the same feature normalization.

fprintf('Running gradient descent ...\n');

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

predict\_x = [1650,3];

predict\_x = (predict\_x - mu) ./ sigma;

price = [1, predict\_x] \* theta;

Output: Normalizing Features ...

Running gradient descent ...

Theta computed from gradient descent:

334302.063993

100087.116006

3673.548451

Predicted price of a 1650 sq-ft, 3 br house (using gradient descent):

$289314.620338

* 1. **Normal Equation**

X = data(:, 1:2);

y = data(:, 3);

m = length(y);

% Add intercept term to X

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

% Calculate the parameters from the normal equation

theta = normalEqn(X, y);

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

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

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

end

Output:

Theta computed from the normal equations:

89597.909542

139.210674

-8738.019112

Predicted price of a 1650 sq-ft, 3 br house (using normal equations):

$293081.464335