Practice Linear Regression

The objective of this practice is to familiarize students with the MATLAB development environment, experience a supervised learning algorithm (linear regression), and understand the gradient descent method. The students will have two help scripts (ex1.m and ex1_multi.m) that will be executed, in parts, in each section of the practice, and will help them to check if their code provides correct results.

1. Linear regression with a variable

We have a data set containing, for some chain stores, the benefit of each store associated with the size of the city where it is located. Linear regression will predict, depending on the size of the city, the benefit we can get if we decided to open a new store in any city.

1.1. Data representation

You can check that ex1.m loads data from a file in the variables X and Y:

```
load data = ('Ex1data1.txt');
X = data (:, 1); and = data (:, 2);
m = length (y);
```

Then the script makes a call to plotData.m. You must complete plotData.m to generate the graph. Modifies the file by entering the following code:

```
plot (x, y, 'rx', 'MarkerSize', 10);
ylabel ( 'Benefit 10,000s');
xlabel ( 'City population in 10,000s');
```

1.2. Gradient descent

In this section we will adjust the parameters (θ) of linear regression to the data provided by the algorithm of the gradient descent.

1.2.1. equations

The aim of the linear regression is to minimize the cost function

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

where the hypothesis is given by the linear model

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

The parameters of our model are θ_0 and θ_1 and they have to be adjusted to minimize the value $J(\theta)$. This is done using the algorithm gradient descent. This algorithm performs the following operations on each iteration:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

for each j (here j = 0 and j = 1). In each iteration of the algorithm gradient descent, values θ_0 and θ_1 are closer to the value that minimizes the function $J(\theta)$

In ex1.m data are prepared for linear regression: one more dimension is added to accommodate the term θ_0 and the parameters are initialized to 0 and the learning rate to 0.01.

```
X = [s (m, 1), data (:, 1)];
theta = zeros (2, 1);
iterations = 1500;
alpha = 0.01;
```

1.2.2. Calculation of the cost $J(\theta)$

Once the descent of the gradient is made we should monitor the convergence of the cost function. **You must implement the function $J(\theta)$**. To do this you must enter the code necessary in the computeCost.m file, which is the function that returns the value of $J(\theta)$. Remember that the variables X and Y are not scalars but matrices whose rows represent examples of the training data.

Once you complete the function, the next step in ex1.m makes a call to computeCost, initializing θ with zeros. Once done, you will see the cost on the screen.

You get a cost *approximately* equal to 32.07.

1.2.3. Gradient descent

Now you will implement gradient descent in gradientDescent.m file. The structure of the loop is already written and only need to add the updates within each iteration. To verify that the descent gradient is running gradientDescent.m code computeCost calls at each iteration and shows the cost $J(\theta)$ per screen correctly. If the implementation is correct value $J(\theta)$ should never increase, and must converge to a stable value at the end of the algorithm.

When you have finished programming, the next step of ex1.m uses the calculated parameters with your code to print the linear function. Your final values θ will also be used to predict the benefits in populations of 35000 and 70000 people through the following lines of ex1.m.

```
predict1 = [1, 3.5] * theta;
predict2 = [1, 7] * theta;
```

1.3. viewing $J(\theta)$

To better understand the cost function $J(\theta)$, we will represent the cost on a bidimensional plane of values θ_0 and θ_1. This section does not need to write any code, but you must understand the code provided.

In the next step of ex1.m, there is a code arranged to calculate $J(\theta)$ in a set of values θ_0 and θ_1 using the computeCost function.

```
J_vals = zeros (length (theta0_vals), length (theta1_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
```

After the execution of these lines, we have an array of values $J(\theta)$. Ex1.m the script uses these values to generate a surface map levels using commands surf and contour. The purpose of these charts is to show how $J(\theta)$ varies and change θ_0 and θ_1. The cost function is completely convex.

2. Linear regression with multiple variables

In this section we apply linear regression with multiple variables to predict the price of houses. To do this we will apply regression on a set of data on homes that have recently sold. The ex1data2.txt file contains a set of data organized in 3 columns: the first contains the size of the housing, the second the number of bedrooms and the third column the price of the house. For this paragraph we will use the script ex1_multi.m.

2.1. Normalization of variables

Ex1_multi.m script loads the data and shows some of them. We see that plot sizes are more than 100 times greater than the number of bedrooms. When variables differ by several orders of magnitude it is convenient to climb them for the gradient descent converges faster.

In this section you must complete featureNormalize.m code to do the following:

- Subtract each variable the mean value of the column
- After subtraction, scale (divide) the values of each variable by the respective "standard deviation".

Standard deviation is a measure of how much variation form is in the range of values of a variable (the most points will be within the range of ± 2 standard deviation). Std can use the MATLAB function to calculate the standard deviation. For example, in featureNormalize.m the amount X (:, 1) contains all values of housing sizes of the training data, so std (X (:, 1)) calculate the standard deviation housing sizes. The code should work

for any size of the dataset (any number of variables and examples). Remember that each column of X corresponds to a variable.

2.2. Gradient descent

In this section you must complete computeCostMulti.my gradientDescentMulti.m code to implement the cost function and gradient descent for linear regression with multiple variables. If your code from the front (a variable) is ready for multiple variables, you can use it here too. Make sure your code supports any number of variables and is well vectorized. Use size (X, 2) to determine how many variables are present in the dataset. We also can help you see how you can write the cost function vectorized form:

$$J(\theta) = \frac{1}{2m} (X\theta - \vec{y})^T (X\theta - \vec{y})$$

where

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

2.2.1 Selecting learning rates

Ultimately this section will test different learning rates for data and look for a learning rate that provides rapid convergence. You can change the learning rate modifying ex1_multi.my and changing some code where alpha is specified. The next step of ex1_multi.m call the function gradientDescent.my run 50 iterations gradient descent to the selected rate. ex1_multi.m script function returns history values $J(\theta)$ in the vector J. After the last iteration, J print values versus the number of iterations.

It is advisable to choose values of the learning rate on a logarithmic scale, in multiplicative steps of approximately 3 times the previous value (ie: 0.001, 0.003, 0.01, 0.03, 0.1, 0.3). You can also change the number of iterations if you need to see the general trend of the curve J). Remember that if the learning rate is very large, $J(\theta)$ can diverge and contain too large values for numerical representation in MATLAB (NaN can mean both ∞ and -∞).

To compare different rates of learning is useful to represent J for different rates in the same figure. You can do this with the hold on command. For example, if you have 3 different values of alpha (although you should try some more), and have kept costs in J1 J2 and J3, you can use the following commands to represent them in the same figure, with different colors:

```
      plot (1:50, J1 (1:50), 'b');
hold on;
plot (1:50, J2 (1:50), 'r');
plot (1:50, J3 (1:50), 'k');
```

With the best learning rate you found, ex1_multi.m uses the script to run the gradient descent until it converges to a final value θ. Then uses that value θ to predict the price of a house 120 meters and 3 bedrooms. Do not forget to normalize these values before making the prediction.