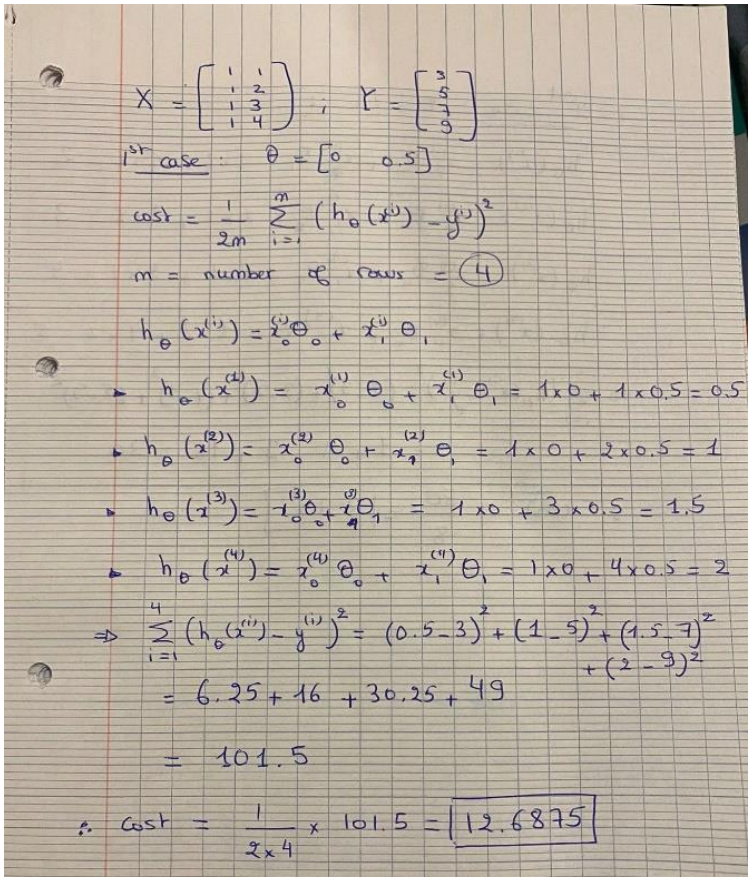
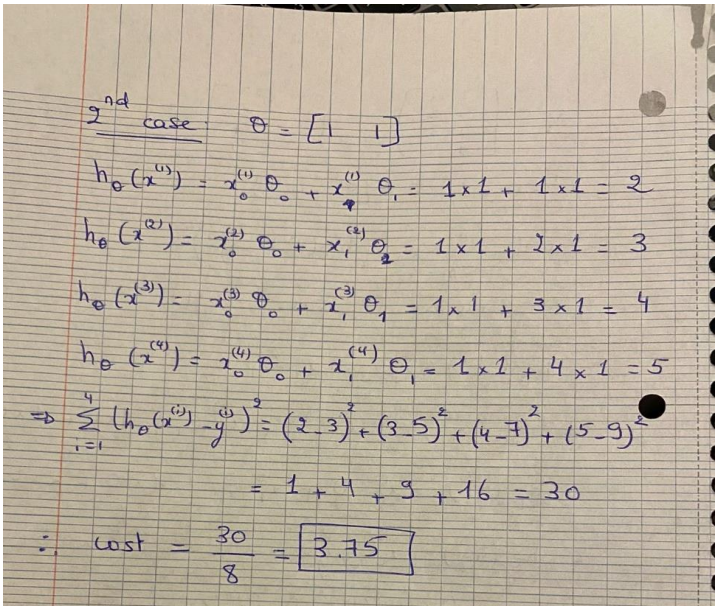
Rayan Hassan
4511021

ECE 1395 – Homework 2 report

1) Here are the manually computed costs for both cases (theta = [0,0.5] and theta = [1,1]) as well as the results I got in python using the computeCost function.






```
In [21]: runfile('C:/Users/RAYAN/OneDrive/Desktop/
ps2.py', wdir='C:/Users/RAYAN/OneDrive/Desktop')
cost with theta = [0 0.5]:  12.6875
cost with theta = [1 1]:  3.75
```

2) The following shows the estimate for theta and its corresponding cost value with alpha=0.001 and 15 iterations.

```
In [22]: runfile('C:/Users/RAYAN/OneDrive/Desktop/
ps2.py', wdir='C:/Users/RAYAN/OneDrive/Desktop')
theta =  [0.32107811 1.11529019]
cost =  4.6672575695826435
```

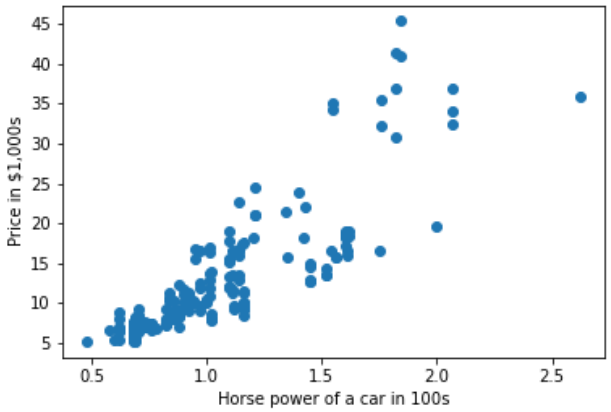
3) The output of the normalEqn function is seen below.

```
In [27]: runfile('C:/Users/RAYAN/OneDrive/Desktop/
normalEqn.py', wdir='C:/Users/RAYAN/OneDrive/
Desktop')
Theta value using normal equation =  [1. 2.]
Cost value using normal equation =  0.0
```

There is a big difference between the theta vectors obtained in 2 and 3. The values in theta vector obtained in 2 are much smaller (the optimal ones) than the ones obtained in 3. That is because the algorithm still hasn't reached the optimal theta that results in cost = 0 (the minima). To remedy that, we should increment the number of iterations.

Rayan Hassan
4511021

4) a) b) The following figure shows the scatter plot of the data given in the problem (ps2-4-b.png).



c) The following shows what the matrix X looks like (we can clearly see that it has 1 feature x1, the first column corresponds to x0 which is equal to 1).

```
Matrix X:
[[1, 1.11], [1, 1.11], [1, 1.54], [1, 1.02], [1, 1.15], [1, 1.1], [1, 1.1], [1, 1.1], [1, 1.4], [1, 1.01], [1,
1.01], [1, 1.21], [1, 1.21], [1, 1.21], [1, 1.82], [1, 1.82], [1, 1.82], [1, 0.48], [1, 0.7], [1, 0.7], [1,
0.68], [1, 0.68], [1, 1.02], [1, 0.68], [1, 0.68], [1, 0.68], [1, 1.02], [1, 0.88], [1, 1.45], [1, 0.58], [1,
0.76], [1, 0.6], [1, 0.76], [1, 0.76], [1, 0.76], [1, 0.76], [1, 0.86], [1, 0.86], [1, 0.86], [1, 0.86], [1,
1.01], [1, 1.0], [1, 0.78], [1, 0.9], [1, 1.76], [1, 1.76], [1, 2.62], [1, 0.68], [1, 0.68], [1, 0.68], [1,
0.68], [1, 0.68], [1, 1.01], [1, 1.01], [1, 1.01], [1, 1.35], [1, 0.84], [1, 0.84], [1, 0.84], [1, 0.84], [1,
0.84], [1, 1.2], [1, 1.55], [1, 1.55], [1, 1.84], [1, 1.84], [1, 1.75], [1, 0.68], [1, 0.68], [1, 0.68], [1,
1.02], [1, 1.16], [1, 0.88], [1, 1.45], [1, 1.45], [1, 1.45], [1, 0.88], [1, 0.88], [1, 1.16], [1, 1.16], [1,
0.69], [1, 0.69], [1, 0.69], [1, 0.69], [1, 0.69], [1, 0.69], [1, 0.69], [1, 0.69], [1, 0.69], [1, 0.97], [1,
0.97], [1, 1.52], [1, 1.52], [1, 1.52], [1, 1.6], [1, 2.0], [1, 1.6], [1, 0.97], [1, 0.97], [1, 0.95], [1, 0.95],
[1, 0.97], [1, 1.42], [1, 0.68], [1, 1.02], [1, 0.68], [1, 0.68], [1, 0.68], [1, 0.88], [1, 1.45], [1, 1.43], [1,
2.07], [1, 2.07], [1, 1.1], [1, 1.1], [1, 1.1], [1, 1.1], [1, 1.6], [1, 1.6], [1, 0.69], [1, 0.73],
[1, 0.73], [1, 0.82], [1, 0.82], [1, 0.94], [1, 0.82], [1, 1.11], [1, 0.82], [1, 0.94], [1, 0.82], [1, 1.11], [1,
0.62], [1, 0.62], [1, 0.62], [1, 0.62], [1, 0.62], [1, 0.62], [1, 0.7], [1, 0.7], [1, 0.7], [1, 0.7], [1, 0.7],
[1, 0.7], [1, 0.7], [1, 1.12], [1, 1.12], [1, 1.16], [1, 1.16], [1, 1.16], [1, 1.16], [1, 1.16], [1, 1.16], [1,
0.92], [1, 0.92], [1, 0.92], [1, 0.92], [1, 1.61], [1, 1.61], [1, 1.56], [1, 1.56], [1, 0.85], [1, 0.85], [1,
0.85], [1, 1.0], [1, 0.9], [1, 0.9], [1, 1.1], [1, 0.88], [1, 1.14], [1, 1.14], [1, 1.14], [1, 1.14], [1, 1.62],
[1, 1.62], [1, 1.14], [1, 1.6], [1, 1.34], [1, 1.14]]
```

Matrix Y is shown below. It has 1 column correspond to the costs.

```
Matrix Y:
[[13.495], [16.5], [16.5], [13.95], [17.45], [15.25], [17.71], [18.92], [23.875], [16.43], [16.925], [20.97],
[21.105], [24.565], [30.76], [41.315], [36.88], [5.151], [6.295], [6.575], [5.572], [6.377], [7.957], [6.229],
[6.692], [7.609], [8.558], [8.921], [12.964], [6.479], [6.855], [5.399], [6.529], [7.129], [7.295], [7.295],
[7.895], [9.095], [8.845], [10.295], [12.945], [10.345], [6.785], [11.048], [32.25], [35.55], [36.0], [5.195],
[6.095], [6.795], [6.695], [7.395], [10.945], [11.845], [13.645], [15.645], [8.845], [8.495], [10.595], [10.245],
[11.245], [18.28], [34.184], [35.056], [40.96], [45.4], [16.503], [5.389], [6.189], [6.669], [7.689], [9.959],
[8.499], [12.629], [14.869], [14.489], [6.989], [8.189], [9.279], [9.279], [5.499], [6.649], [6.849], [7.349],
[7.299], [7.799], [7.499], [7.999], [8.249], [8.949], [9.549], [13.499], [14.399], [13.499], [17.199], [19.699],
[18.399], [11.9], [12.44], [15.58], [16.695], [16.63], [18.15], [5.572], [7.957], [6.229], [6.692], [7.609],
[8.921], [12.764], [22.018], [32.528], [34.028], [37.028], [11.85], [12.17], [15.04], [15.51], [18.15], [18.62],
[5.118], [7.053], [7.603], [7.126], [7.775], [9.96], [9.233], [11.259], [7.463], [10.198], [8.013], [11.694],
[5.348], [6.338], [6.488], [6.918], [7.898], [8.778], [6.938], [7.198], [7.738], [8.358], [9.258], [8.058],
[8.238], [9.298], [9.538], [8.449], [9.639], [9.989], [11.199], [11.549], [17.669], [8.948], [9.988], [10.898],
[11.248], [16.558], [15.998], [15.69], [15.75], [7.975], [8.195], [8.495], [9.995], [11.595], [9.98], [13.295],
[12.29], [12.94], [13.415], [15.985], [16.515], [18.42], [18.95], [16.845], [19.045], [21.485], [22.625]]
```

X has 179 rows (training samples) and 2 columns (x0 and feature x1). Y has 179 rows and 1 column (prices). This is shown below:

```
Size of matrix X: (179, 2)
Size of matrix Y:  (179, 1)
```

Rayan Hassan
4511021

Note that the matrices shown above are lists, but I convert them to numpy arrays in my code to match the functions in gradientDescent and computeCost. (so this is just for presentation purpose, so that you can see them clearly, since the numpy arrays are very long given than we have 179 rows).

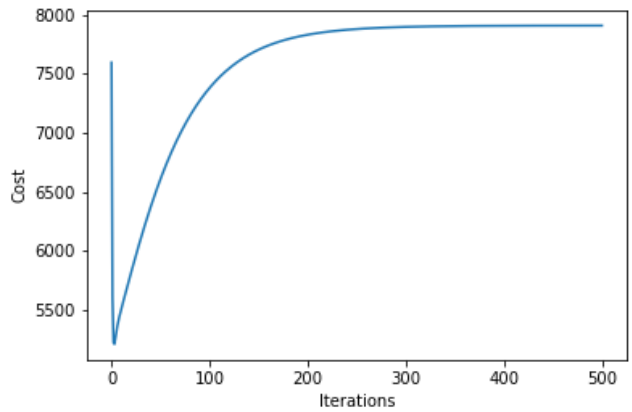
d) The matrices X_train and Y_train are shown below. (Again, they are lists here but are converted to numpy arrays in the code)

```
Reloaded modules: computeCost, gradientDescent, normalEqn
X_train is:
[[1, 1.6], [1, 1.11], [1, 0.48], [1, 0.68], [1, 0.7], [1, 1.16], [1, 1.45], [1, 0.86], [1, 0.69], [1, 0.85], [1, 0.97], [1, 0.76], [1,
0.92], [1, 1.16], [1, 1.43], [1, 1.1], [1, 1.11], [1, 1.82], [1, 2.07], [1, 1.82], [1, 1.14], [1, 0.84], [1, 1.02], [1, 0.88], [1, 1.16],
[1, 0.69], [1, 2.07], [1, 0.97], [1, 1.35], [1, 0.68], [1, 1.01], [1, 1.6], [1, 0.7], [1, 1.6], [1, 1.75], [1, 0.88], [1, 2.07], [1, 1.0],
[1, 1.01], [1, 2.62], [1, 1.45], [1, 1.16], [1, 0.58], [1, 0.68], [1, 0.92], [1, 1.0], [1, 0.68], [1, 1.16], [1, 0.7], [1, 0.82], [1, 1.02],
[1, 0.7], [1, 1.82], [1, 0.69], [1, 0.88], [1, 0.9], [1, 1.52], [1, 0.82], [1, 1.42], [1, 0.7], [1, 0.62], [1, 1.02], [1, 0.7], [1, 1.02],
[1, 0.69], [1, 1.16], [1, 0.85], [1, 0.62], [1, 0.68], [1, 0.68], [1, 1.56], [1, 0.97], [1, 0.62], [1, 0.68], [1, 1.56], [1, 0.82], [1,
0.84], [1, 1.16], [1, 0.68], [1, 1.01], [1, 0.68], [1, 0.69], [1, 0.88], [1, 1.14], [1, 0.97], [1, 0.6], [1, 0.82], [1, 0.62], [1, 0.7], [1,
0.92], [1, 1.02], [1, 1.76], [1, 0.95], [1, 0.94], [1, 1.55], [1, 1.1], [1, 0.9], [1, 1.34], [1, 1.45], [1, 1.52], [1, 0.9], [1, 0.84], [1,
0.69], [1, 0.73], [1, 0.68], [1, 1.12], [1, 1.55], [1, 1.2], [1, 1.16], [1, 0.84], [1, 1.1], [1, 2.0], [1, 0.76], [1, 1.1], [1, 1.84], [1,
1.14], [1, 0.76], [1, 0.7], [1, 0.69], [1, 1.52], [1, 1.1], [1, 1.76], [1, 1.84], [1, 1.61], [1, 1.1], [1, 0.97], [1, 1.61], [1, 0.95], [1,
1.1], [1, 1.14], [1, 0.84], [1, 0.88], [1, 0.69], [1, 1.14], [1, 1.21], [1, 1.1], [1, 0.7], [1, 1.11], [1, 0.86], [1, 1.21], [1, 0.85], [1,
0.68], [1, 0.68], [1, 0.78], [1, 1.14], [1, 1.01], [1, 0.68], [1, 1.11], [1, 0.92], [1, 0.69], [1, 0.69], [1, 0.68], [1, 1.12], [1, 1.6],
[1, 0.94], [1, 0.68], [1, 1.45], [1, 1.16], [1, 0.88], [1, 1.6], [1, 1.62]]


Y_train is:
[[18.62], [11.259], [5.151], [5.572], [9.258], [9.989], [12.629], [9.095], [5.118], [8.195], [11.9], [6.529], [11.248], [9.959], [22.018],
[15.51], [13.495], [36.88], [37.028], [41.315], [16.515], [10.595], [7.689], [8.921], [8.449], [5.499], [34.028], [12.44], [15.645],
[6.695], [10.945], [18.15], [8.058], [17.199], [16.503], [8.499], [32.528], [9.995], [11.845], [36.0], [12.964], [9.639], [6.479], [7.395],
[9.988], [10.345], [5.572], [9.279], [7.198], [7.775], [7.957], [7.738], [30.76], [7.999], [8.189], [9.98], [13.499], [7.463], [18.15],
[6.575], [6.918], [8.558], [6.938], [7.957], [7.349], [17.669], [7.975], [8.778], [6.377], [6.189], [15.75], [16.63], [5.348], [6.692],
[15.69], [9.233], [8.495], [11.199], [6.095], [16.925], [7.609], [6.649], [8.921], [12.94], [9.549], [5.399], [7.126], [7.898], [6.295],
[8.948], [13.95], [32.25], [15.58], [10.198], [34.184], [17.71], [11.595], [21.485], [14.869], [14.399], [11.048], [10.245], [7.299],
[7.603], [5.389], [9.538], [35.056], [18.28], [9.279], [8.845], [11.85], [19.699], [7.295], [13.295], [40.96], [15.985], [7.295], [8.238],
[6.849], [13.499], [15.25], [35.55], [45.4], [16.558], [12.17], [8.949], [15.998], [16.695], [15.04], [22.625], [11.245], [6.989], [7.799],
[16.845], [24.565], [18.92], [8.358], [11.694], [7.895], [21.105], [8.495], [6.669], [7.609], [6.785], [13.415], [16.43], [5.195], [16.5],
[10.898], [8.249], [7.499], [6.229], [9.298], [19.045], [9.96], [6.795], [12.764], [11.549], [12.29], [18.399], [18.42]]
```

The X_test and Y_test matrices are shown here:

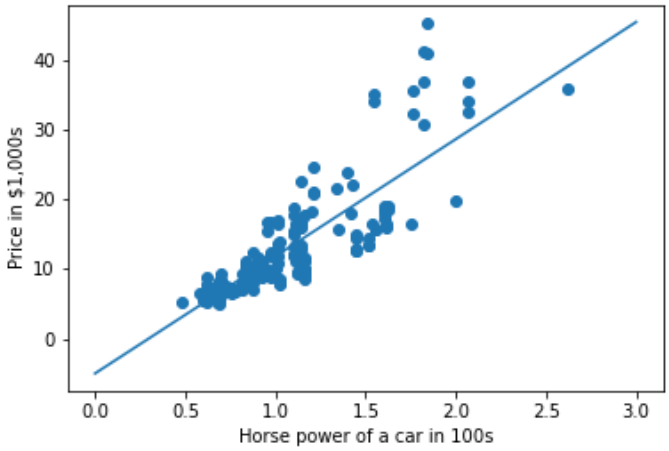
```
X_test is:
[[1, 0.76], [1, 0.68], [1, 1.4], [1, 0.86], [1, 0.86], [1, 1.54], [1, 1.45], [1, 1.01], [1, 0.73], [1, 1.21], [1, 0.68], [1, 1.62],
[1, 0.76], [1, 1.15], [1, 0.62], [1, 0.62], [1, 0.82], [1, 1.01]]


Y_test is:
[[7.129], [6.692], [23.875], [10.295], [8.845], [16.5], [14.489], [12.945], [7.053], [20.97], [6.229], [18.95], [6.855], [17.45],
[6.488], [6.338], [8.013], [13.645]]
```

e) The following shows the plot cost vs iterations (ps2-4-e-1.png)



The next figure shows the data points along with the line h(x) = $\theta_0 + \theta_1 x$ (ps2-4-e-2.png)

Rayan Hassan
4511021



The parameter value (theta) is shown below.

```
Theta =  [-5.80562456 17.83647811]
```

f) The predicted costs are shown below (y predicted). I got it using θ.X_test (.dot function, and the theta that is shown in question above).

```
Y_predicted:
[ 7.7500988   6.32318055 19.1654448   9.53374662  9.53374662 21.66255173
 20.0572687  12.20921833  7.21500446 15.77651396  6.32318055 23.08946998
  7.7500988  14.70632527  5.25299187  5.25299187  8.82028749 12.20921833]
```

For the mean squared error, I couldn't import the function mean_squared_error (from sklearn library) for some reason. So I used the formula $\frac{1}{n}\sum_{i=1}^{n}(Y(test) - Y(predicted))^2$. And to find the mean I simply used np.mean(error) (see corresponding section in my code). Prediction error:

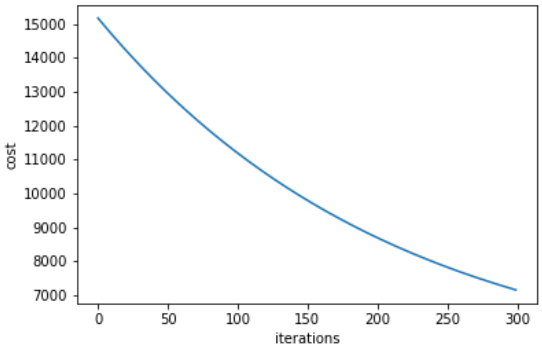
```
Prediction error:  1824.55388278397
```

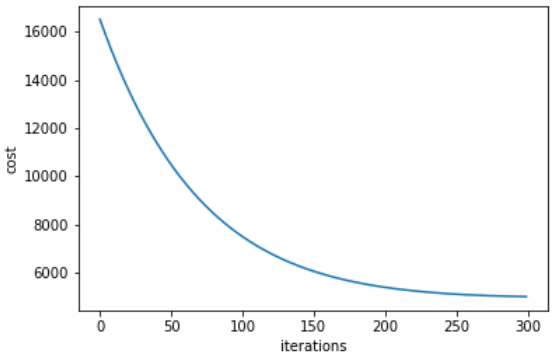
g) Using the normal equation method, we obtain the following theta vector [-5.5889, 17,6409], which is almost exactly the same as the one obtained using the gradient descent method. After that, I calculate a new y_predict (using same .dot method), and then calculated the average mean squared error in the same way as before. Prediction error here is almost equal to 16 which is way smaller than the one obtained in the previous question. This means that although the theta values obtained are similar, there is large error for our predicted Y vector. This is because our alpha value (0.3) is relatively high, so the model converges too quickly to a solution.

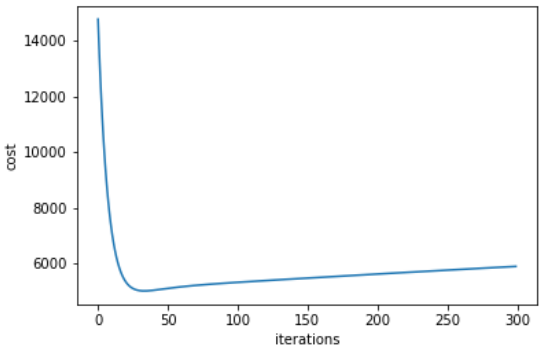
```
Parameter theta from normal equation:  [[-5.58899888]
 [17.6409884 ]]
Prediction error (after using normal equation):  16.24590131003712
```

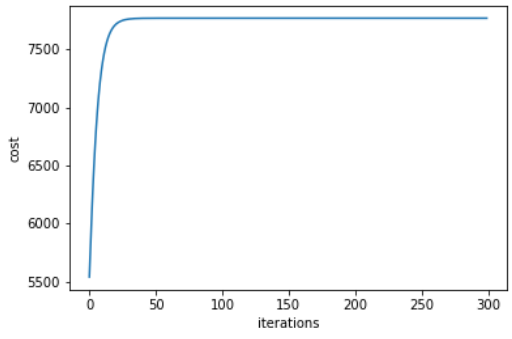
Rayan Hassan
4511021

h) The following figures show cost vs iterations for alpha = 0.001, 0.003, 0.03 and 3 respectively. (corresponding to ps2-4-h-1.png through ps2-4-h-4.png respectively).









Rayan Hassan
4511021

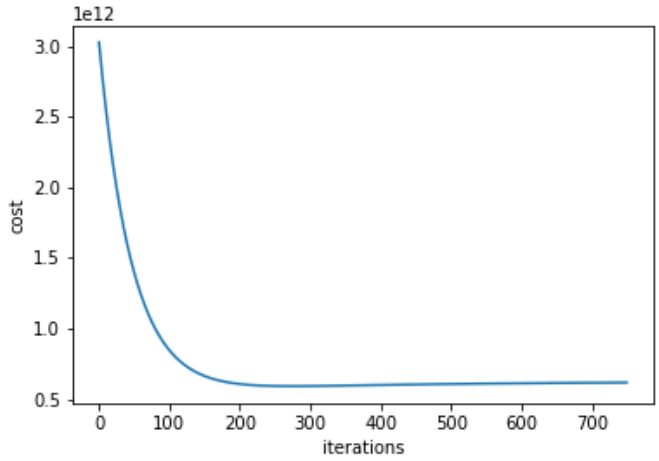
We can see that for the first figure, we have a low learning rate. For alpha = 0.003, the figure shows a good convergence (so alpha is just right). As alpha increases (0.03), alpha is large (converges too quickly to a minima). In the last figure it is clear that alpha is very high and the curve itself is not right (no convergence).

5) a) The mean and standard deviation of each vector is shown below. For vector that has the size of houses, mean is almost 2000.68, whereas for the one that contains the number of bedrooms, it is almost 3.17. The respective standard deviations are 794.7 and 0.7609 (almost).

Also, the screenshot shows the sizes of X and Y. X has 47 rows (samples) and 3 columns (2 features x1 and x2, and x0=all ones). Similarly, Y has 47 rows and 1 column (corresponding price for each sample).

```
mean of size_house vector (first column):  2000.6808510638298
mean of nbre_bedrooms vector (second column):  3.1702127659574466
standard deviation of size_house:  794.7023535338897
standard deviation of nbre_bedrooms:  0.7609818867800998
Size of feature matrix X:  (47, 3)
Size of label vector Y:  (47, 1)
```

b) The next figure shows cost vs iterations (ps2-5-b.png)



The computed model parameter theta is shown next:

θ_0 = 340231.3582597, θ_1 = 108352.80802054, θ_2 = -4377.20000942

```
Computed model paramter theta =  [340231.3582597  108352.80802054
-4377.20000942]
```

c) Using the obtained theta vector, and after normalizing the input feature, the predicted price for a 1080 square feet house with 2 bedrooms is 221433.26552738 $. I used X.theta to find it.

```
Predicted price:  [221433.26552738]
```