Neeraj Vashistha (190573535)

ECS708 Machine Learning

Assignment 1: Part 1 - Linear Regression

Task 1 Modify the function calculate_hypothesis.m to return the predicted value for a single specified training example. Include in the report the corresponding lines from your code.

```
def calculate_hypothesis(X, theta, i):
    """
        :param X            : 2D array of our dataset
        :param theta        : 1D array of the trainable parameters
        :param i            : scalar, index of current training sample's row
    """

    #hypothesis = 0.0
    hypothesis = np.dot(X[i],theta)
    return hypothesis
```

Notice that the hypothesis function is not being used in the gradient_descent function.

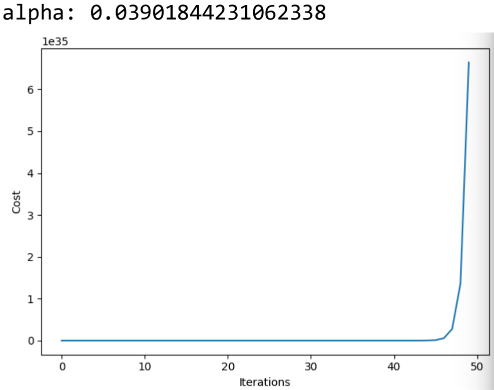
Modify it to use the calculate_hypothesis function. Include the corresponding lines of the code in your report

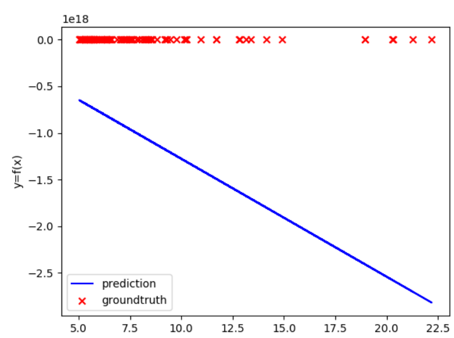
```
hypothesis = calculate_hypothesis(X, theta, i)
```

Now modify the values for the learning rate, alpha in mllab1.m.

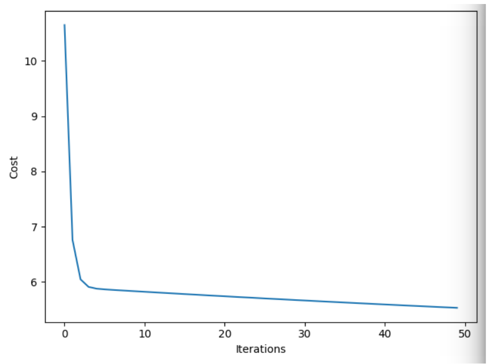
Observe what happens when you use a very high or very low learning rate. Document and comment on your findings in the report.

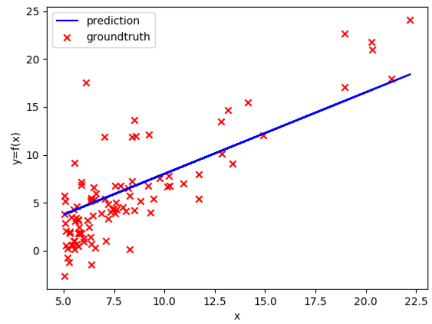
```
Minimum cost: 386.05252, on iteration #1
alpha: 0.03901844231062338
```



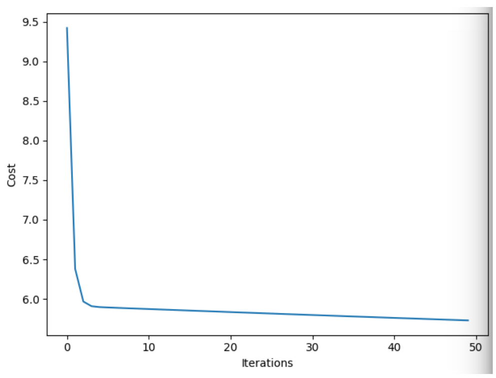


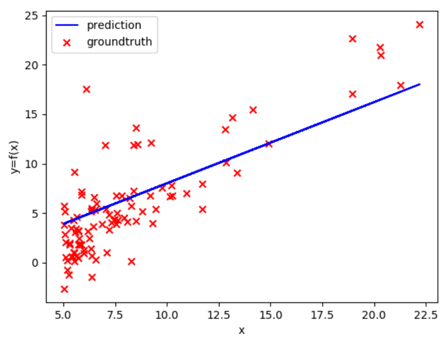

```
Gradient descent finished.
Minimum cost: 133.44523, on iteration #1
alpha: 0.02601229487374892
Gradient descent finished.
Minimum cost: 39.82595, on iteration #1
alpha: 0.017341529915832612
```






```
Gradient descent finished.
Minimum cost: 5.52904, on iteration #50
alpha: 0.011561019943888409
Gradient descent finished.
Minimum cost: 5.64490, on iteration #50
```
alpha: 0.007707346629258939 6





Gradient descent finished.
Minimum cost: 5.72906, on iteration #50

```
alpha: 0.005138231086172626
Gradient descent finished.
Minimum cost: 5.78848, on iteration #50
alpha: 0.0034254873907817508
Gradient descent finished.
Minimum cost: 5.82964, on iteration #50
```

From above we can see the cost was lowest when alpha was 0.017341529915832612 and it was approximately 5.52904

As we have a higher learning rate, we fail to achieve convergence and the cost is very high, but as we decrease the learning rate, we attain an optimum, here the learning rate was between 0.01 to 0.02, but if we decrease the learning rate too low the rate of convergence becomes too slow and for the number of iterations (50) it fails to converge again.

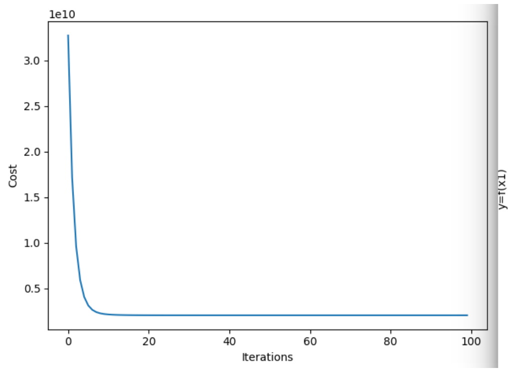
Task 2 Modify the functions calculate_hypothesis and gradient_descent to support the new hypothesis function. This should be sufficiently general so that we can have any number of extra variables. Include the relevant lines of the code in your report.

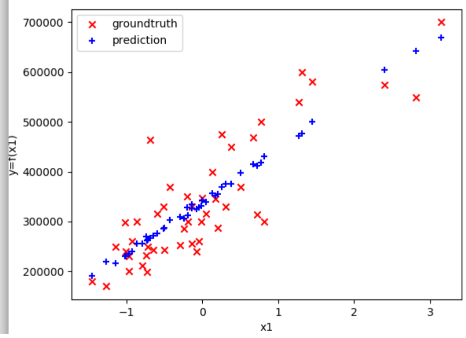
```
for it in range(iterations):
        # initialize temporary theta, as a copy of the existing theta array
```

```
        theta_temp = theta.copy()
        # print(len(theta_temp))
        sigma = np.zeros((len(theta)))
        # print(sigma)
        for index in range(len(theta_temp)):
            for i in range(m):
                hypothesis = calculate_hypothesis(X, theta, i)
                output = y[i]
                sigma[index] = sigma[index] + (hypothesis - output) * X[i, index]

            theta_temp[index] = theta_temp[index] - (alpha/m) * sigma[index]
        # copy theta_temp to theta
        theta = theta_temp.copy()
        # append current iteration's cost to cost_vector
        iteration_cost = compute_cost(X, y, theta)
        cost_vector = np.append(cost_vector, iteration_cost)

        # plot predictions for current iteration
        if do_plot==True:
            plot_hypothesis(X, y, theta, ax1)
```

Run ml_assgn1_2.py and see how different values of alpha affect the convergence of the algorithm. Print the theta values found at the end of the optimization. Include the values of theta and your observations in your report.

```
Dataset normalization complete.
alpha:  0.01
Gradient descent finished.
Minimum cost: 10596969344.16698, on iteration #100
theta_final:  [215810.61679138  61446.18781361  20070.13313796]
[183865.19798769]
[316034.47300652]

Dataset normalization complete.
alpha:  0.1
Gradient descent finished.
Minimum cost: 2043462824.61817, on iteration #100
theta_final:  [340403.61773803 108803.37852266  -5933.9413402 ]
[293214.16354571]
[472159.9884142]

Dataset normalization complete.
alpha:  0.2
Gradient descent finished.
Minimum cost: 2043280065.35581, on iteration #100
theta_final:  [340412.65950512 109442.00621882  -6572.56460334]
[293082.73783407]
[472276.87730296]
```





Dataset normalization complete.
alpha: 0.3
Gradient descent finished.
Minimum cost: 2043280050.60358, on iteration #100
theta_final: [340412.65957447 109447.75525931 -6578.31364383]
[293081.47339913]
[472277.84818736]

Dataset normalization complete.
alpha: 0.4
Gradient descent finished.
Minimum cost: 2043280050.60283, on iteration #100
theta_final: [340412.65957447 109447.79624289 -6578.35462741]
[293081.46438477]
[472277.85510807]

Dataset normalization complete.
alpha: 0.5
Gradient descent finished.
Minimum cost: 2043280050.60283, on iteration #78
theta_final: [340412.65957447 109447.7964687   -6578.35485322]
[293081.4643351]
[472277.8551462]

Dataset normalization complete.
alpha: 10
Gradient descent finished.
Minimum cost: 5591996951312.50488, on iteration #1
theta_final: [-7.18869008e+105 -1.39819070e+121 -1.39819070e+121]
[9.39776019e+120]
[-3.3182923e+121]

As we increase the value of alpha from 0.1 to 0.5 we find the lowest cost value is obtained when alpha is 0.3 and after 0.3 the cost does not decrease further meaning the model has converged but if we set value of alpha to be too high or too low the model cost is very high or the model fails to converge.

Finally, we would like to use our trained theta values to make a prediction. Add some lines of code in mllab2.m to make predictions of house prices.

How much does your algorithm predicts that a house with 1650 sq. ft. and 3 bedrooms cost?

293081.47339913

How about 3000 sq. ft. and 4 bedrooms?

472277.84818736

```
X1_new = ([[1650,3]] - mean_vec)/std_vec
X_normalized = np.append(np.ones((X1_new.shape[0], 1)), X1_new, axis=1)
print(np.dot(X_normalized,theta_final))



X2_new = ([[3000,4]] - mean_vec)/std_vec
# print(X_normalized)
X_normalized = np.append(np.ones((X2_new.shape[0], 1)), X2_new, axis=1)
print(np.dot(X_normalized,theta_final))
```

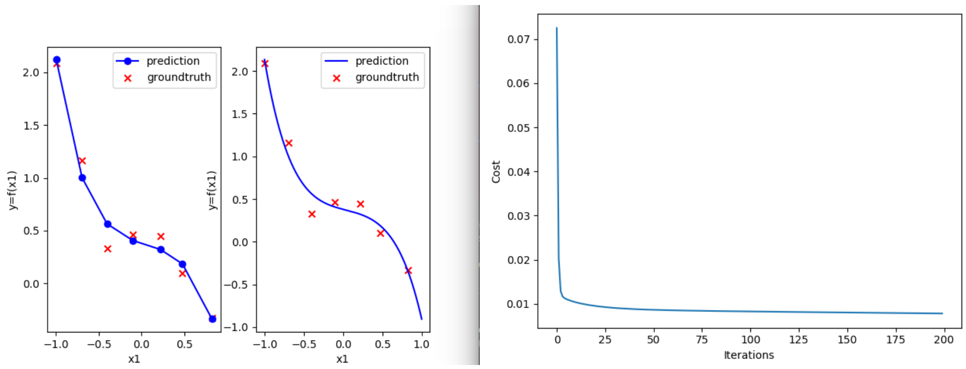
output

```
[293081.47339913]
[472277.84818736]
```

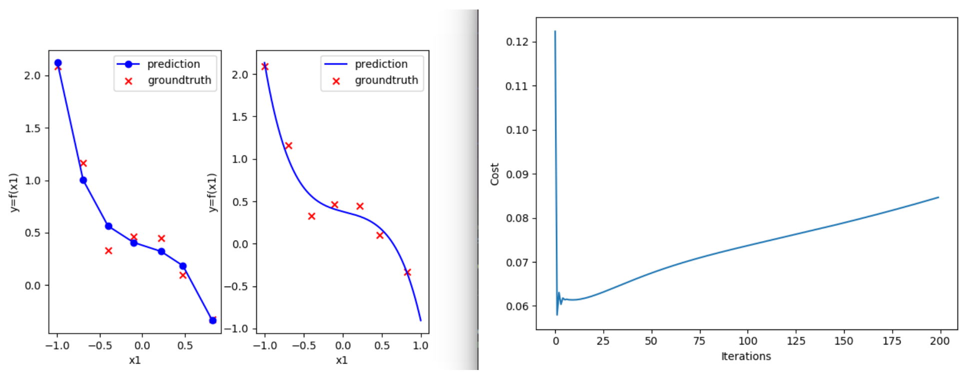
Task 3 Note that the punishment for having more terms is not applied to the bias. This cost function has been implemented already in the function compute_cost_regularised. Modify gradient_descent to use the compute_cost_regularised method instead of compute_cost. Include the relevant lines of the code in your report and a brief explanation.

```
# Gradient Descent loop
    for it in range(iterations):

        # initialize temporary theta, as a copy of the existing theta array
        theta_temp = theta.copy()

        sigma = np.zeros((len(theta)))
        for index in range(len(theta_temp)):
            for i in range(m):
                hypothesis = calculate_hypothesis(X, theta, i)
                output = y[i]
                sigma[index] = sigma[index] + (hypothesis - output) * X[i, index]

            theta_temp[index] = theta_temp[index] - (alpha/m) * sigma[index]
        # copy theta_temp to theta
        theta = theta_temp.copy()


        # append current iteration's cost to cost_vector
        # iteration_cost = compute_cost(X, y, theta)
        iteration_cost = compute_cost_regularised(X, y, theta, l)
```

```
        cost_vector = np.append(cost_vector, iteration_cost)
```




```
Unregularized alpha = 1, iterations = 200
Gradient descent finished.
Minimum cost: 0.00780, on iteration #200
```




```
Regularized alpha = 1, iteration = 200, lambda = 0.0
Gradient descent finished.
Minimum cost: 0.05795, on iteration #2
```

The cost has nearly increased 4 times, when we tune the hyperparameter lambda it penalizes the model and we see this increased cost.

Next, modify gradient_descent to incorporate the new cost function. Again, we do not want to punish the bias term.

This means that we use a different update technique for the partial derivative of θ 0 , and add the regularization to all of the others:

$$\theta_0 = \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right) x_0^{(i)} \qquad \text{j=0}$$

$$\theta_j = \theta_j \left(1 - \alpha \frac{\lambda}{m} \right) - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right) x_j^{(i)} \qquad \text{j>0}$$

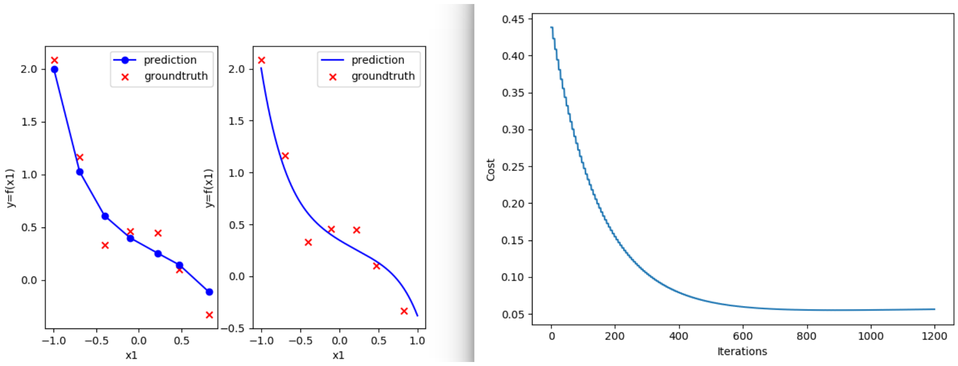
Include the relevant lines of the code in your report.

```
def compute_cost_regularised(X, y, theta, alpha, l):

    """
    :param X        : 2D array of our dataset
    :param y        : 1D array of the groundtruth labels of the dataset
    :param theta    : 1D array of the trainable parameters
    :param l        : scalar, regularization parameter
    """

    # initialize costs
    # total_squared_error = 0.0
    # total_regularised_error = 0.0

    # get number of training examples
    m = y.shape[0]

    def ret_total_squared_error(X, y, theta):
        total_squared_error = 0.0
        m = y.shape[0]
        for i in range(m):
            hypothesis = calculate_hypothesis(X, theta, i)
            output = y[i]
            squared_error = (hypothesis - output)**2
            total_squared_error += squared_error
        return total_squared_error

    def ret_total_regularised_error(theta):
        total_regularised_error = 0.0
        for i in range(1,len(theta)):
            if i == 0:
                total_regularised_error += (theta[i] - ((alpha/m) * (ret_total_squared_error(X,y,theta) *
X[0])))**2
            else:
                total_regularised_error += (theta[i]*(1 - alpha*(l/m)) - ((alpha)*(1/m) *
(ret_total_squared_error(X,y,theta) * X[i])))**2
            # total_regularised_error += theta[i]**2
        return total_regularised_error

    total_squared_error = ret_total_squared_error(X, y, theta)
    total_regularised_error = ret_total_regularised_error(theta)
    J = (total_squared_error + total_regularised_error)/(2*m)
    return J
```

After gradient_descent has been updated, run ml_assgn1_3.py. This will plot the hypothesis function found at the end of the optimization.

First of all, find the best value of alpha to use in order to optimize best. Report the value of alpha that you found in your report.

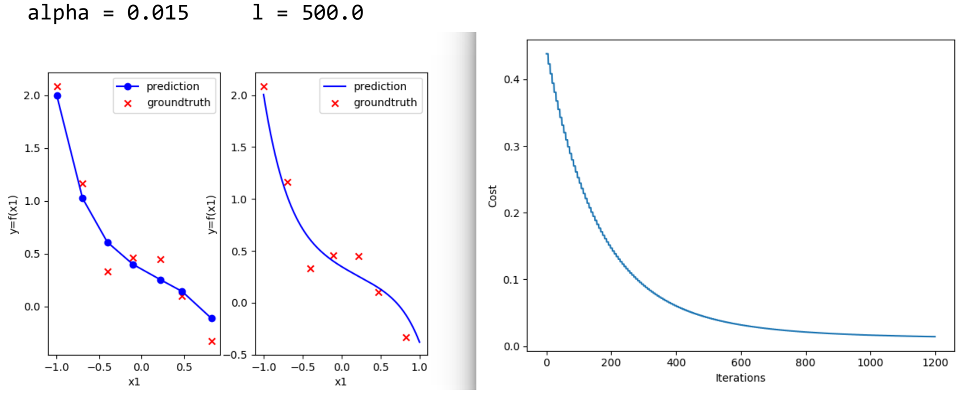
```
alpha = 0.015 ,    l = 0.0
```




```
Gradient descent finished.
Minimum cost: 0.05497, on iteration #890
```

Next, experiment with different values of λ and see how this affects the shape of the hypothesis. Note that gradient_descent will have to be modified to take an extra parameter, l (which represents λ). Include in your report the plots for a few different values of λ and comment.

```
alpha = 0.015     l = 500.0
```




```
Gradient descent finished.
Minimum cost: 0.01429, on iteration #1195
```

After we add a penalty(lambda = 500) the model has stopped memorizing the points instead it has become more generalized.