Report of Assignment_1_part_1

Task 1:

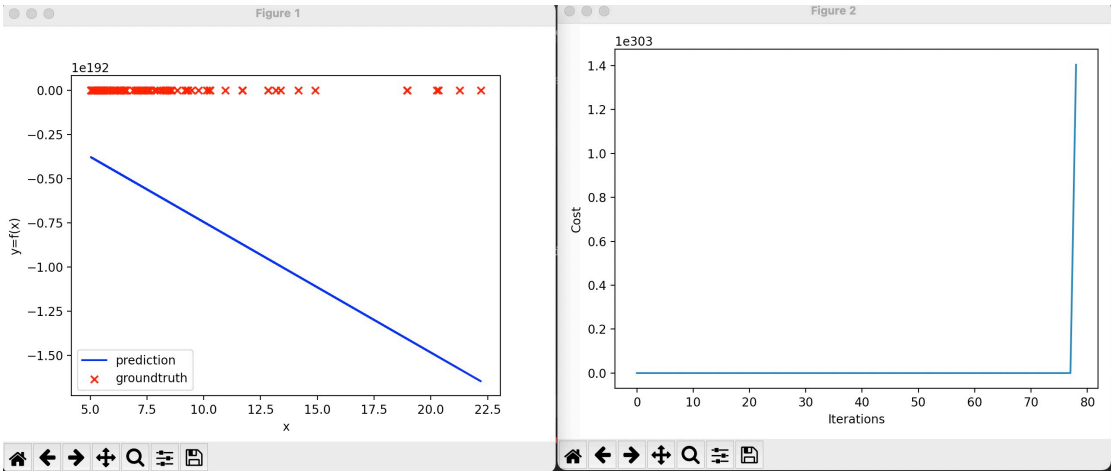
Modify the function *caculated_hypothesis.py* :

```
hypothesis = 0.0
########################################
# Write your code here
# You must calculate the hypothesis for the i-th sample of X, given X, theta and i.
hypothesis = X[i, 0] * theta[0] + X[i, 1] * theta[1]
#hypothesis = np.transpose(theta) * X [i,:]
#hypothesis = sum(hypothesis)
########################################
```

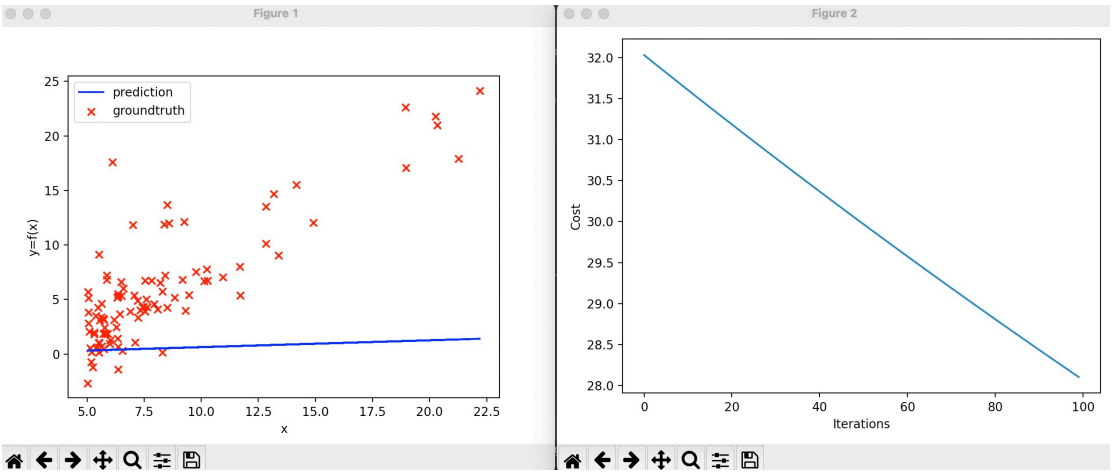
Modify *gradient_descent.py* to use the *caculated_hypothesis.py* :

```
########################################
# Write your code here
# Replace the above line that calculates the hypothesis, with a call to the "calculate_hypothesis" function
hypothesis = calculate_hypothesis(X, theta, i)
########################################/
```

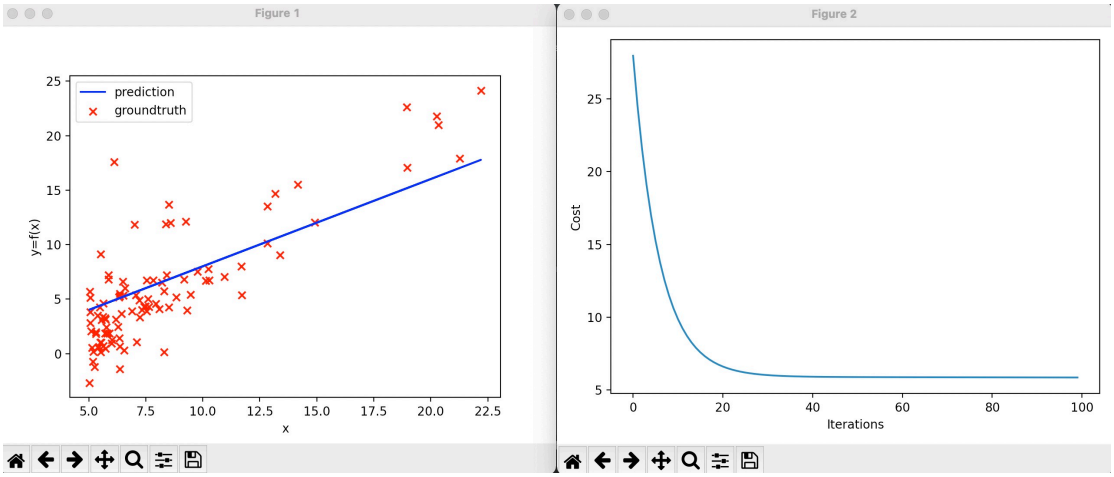
When use a very high learning rate(1.0):



When use a very low learning rate(0.0001):



When use a appropriate learning rate(0.1):



Comments:

using different learning rate will get different prediction lines. When used a very high or low learning rate, the prediction lines were both not good for fitting the data. Meanwhile the cost values of prediction results were high.

Task 2:

Modify the functions *calculate_hypothesis.py* and *gradien_descent.py*:

```
hypothesis=0
########################################
# Write your code here
# You must calculate the hypothesis for the i-th sample of X, given X, theta and i.
#hypothesis=theta[0]+theta[1]*X[i,0]+theta[2]*X[i,1]
hypothesis = theta * X[i,:]
hypothesis = sum(hypothesis)
########################################/
```

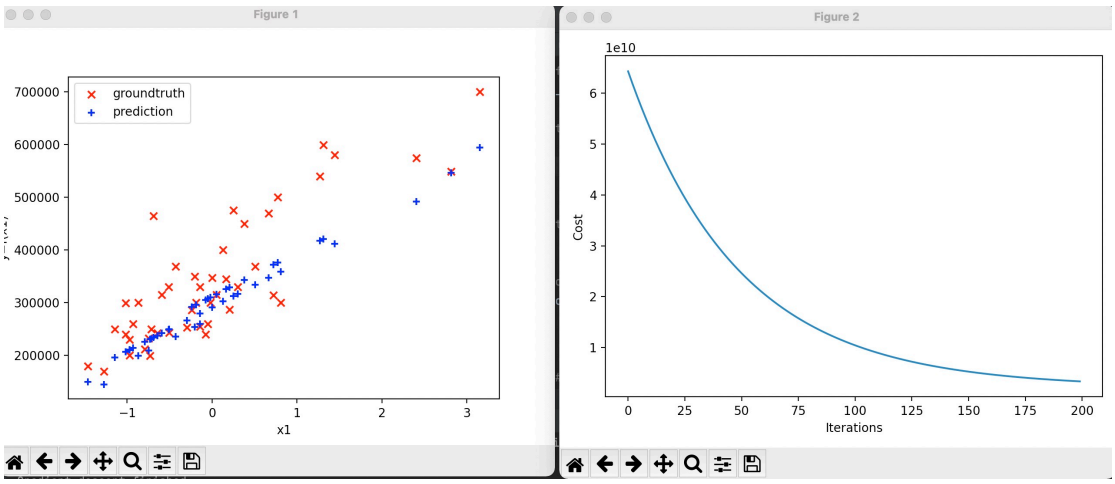
```
for i in range(m):
    ########################################
    # Write your code here
    # Calculate the hypothesis for the i-th sample of X, with a call to the "calculate_hypothesis" function
    hypothesis = calculate_hypothesis(X, theta, i)
    ########################################/
    output = y[i]
    ########################################
    # Write your code here
    # Adapt the code, to compute the values of sigma for all the elements of theta
    #sigma[0]=sigma[0]+(hypothesis-output)*1
    #sigma[1]=sigma[1]+(hypothesis-output)*X[i,0]
    #sigma[2]=sigma[2] + (hypothesis - output) * X[i, 1]
    sigma = sigma + (hypothesis - output) * X[i,:]
    ########################################/
```

```
# update theta_temp
########################################
# Write your code here
# Update theta_temp, using the values of sigma
#theta_temp[0]=theta[0] - (alpha / m) * sigma[0]
#theta_temp[1] = theta[1] - (alpha / m) * sigma[1]
#theta_temp[2] = theta[2] - (alpha / m) * sigma[2]
theta_temp = theta_temp - (alpha / m) * sigma
########################################/
# copy theta_temp to theta
theta = theta_temp.copy()
```

The theta values found at the end of the optimization :

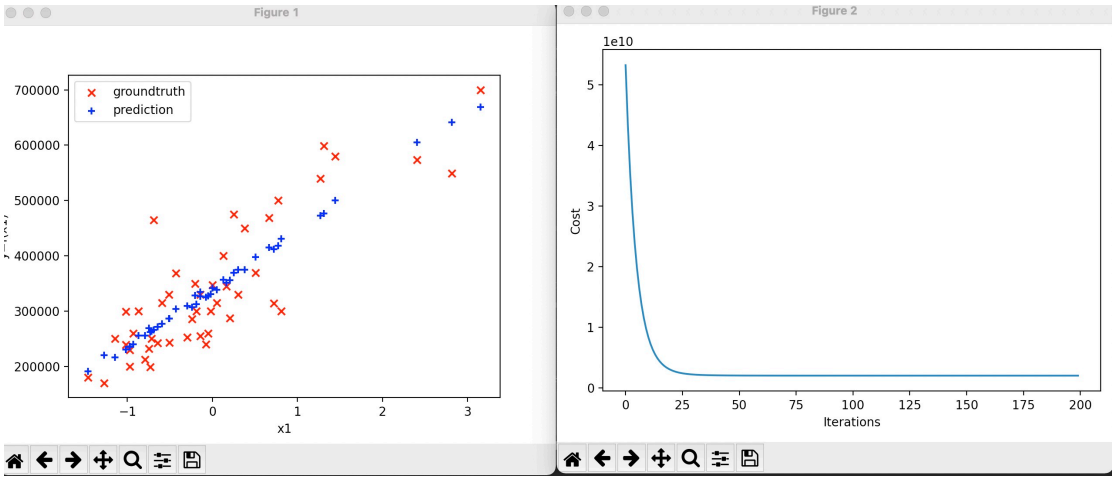
Alpha=0.01

```
Dataset normalization complete.
Gradient descent finished.
Minimum cost: 3344770635.49166, on iteration #200
print the final theta
[294804.28212715  83217.03697925  15220.03137818]
```



Alpha=0.1

```
Minimum cost: 2043280073.15030, on iteration #200
print the final theta
[340412.6593343  109440.63822766  -6571.19661218]
```



The final theta values are so large. And with the rise of alpha value, the value of the final theta were rising too.

The prices of the houses :

Based on alpha=0.1

```
Dataset normalization complete.
Gradient descent finished.
Minimum cost: 2043280073.15030, on iteration #200
print the final theta
[340412.6593343  109440.63822766  -6571.19661218]
the prediction prices for sample_1 is 293083.0385535835
the prediction prices for sample_2 is 472276.6461255231
```

Task 3:

Modify the *gradient_descent.py* to use the *compute_cost_regularised* method :

```
    # 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)
```

Modify the *gradient_descent.py* to incorporate the new cost function :

```
for i in range(m):
    ########################################
    # Write your code here
    # Calculate the hypothesis for the i-th sample of X, with a call to the "calculate_hypothesis" function
    hypothesis = calculate_hypothesis(X, theta, i)
    ########################################/
    output = y[i]
    ########################################
    # Write your code here
    # Adapt the code, to compute the values of sigma for all the elements of theta
    sigma=sigma+(hypothesis-output)*X[i,:]
    ########################################/

# update theta_temp
########################################
# Write your code here
# Update theta_temp, using the values of sigma
# Make sure to use lambda, if necessary
theta_temp[0]=theta_temp[0]-(alpha/m)*sigma[0]
theta_temp[1:]=theta_temp[1:]*(1-(alpha/m)*l)-(alpha/m)*sigma[1:]
########################################/

# copy theta_temp to theta
theta = theta_temp.copy()
```

```
    # 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)
```

Find the best value of alpha :

Alpha=0.05