Assignment_1_P1

Yang Tian 200054403

1. Linear Regression with One Variable

Task 1.
Modify *calculate_hypothesis.py*

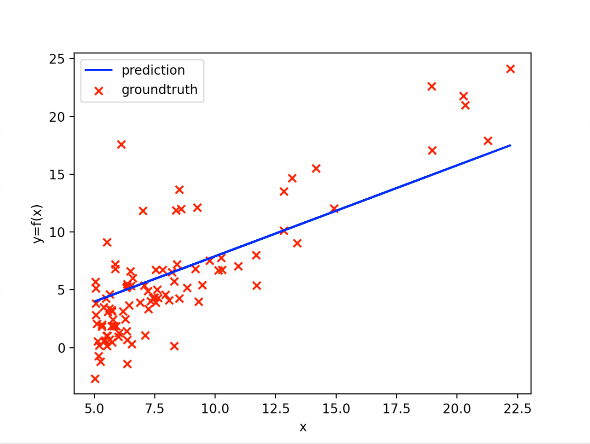
```
def calculate_hypothesis(X, theta, i):
    """
        :param X            : 2D array of our dataset
        :param theta        : 1D array of the trainable parameters
        :param i            : scalar, index of current training sample's row
    """

    hypothesis = 0.0
    ##########################################
    # 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]
    # print("i : " ,i)
    # print("theta 0 : ", theta[0])
    # print("theta 1 : ", theta[1])
    # print(hypothesis)

    return hypothesis
```

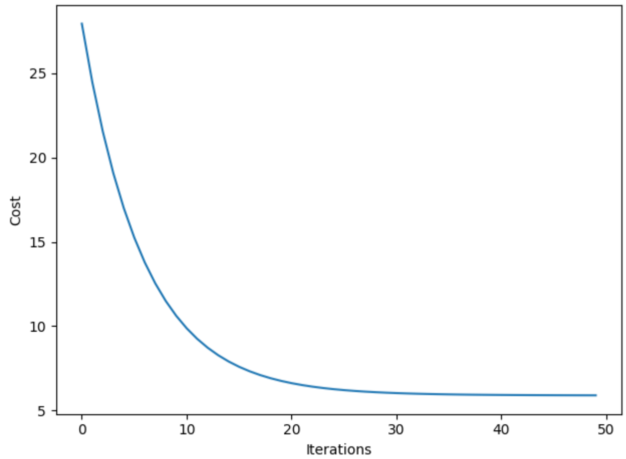
Modify *gradient_descent*

```
  for i in range(m):
      # hypothesis = X[i, 0] * theta[0] + X[i, 1] * theta[1]
      ##########################################
      # Write your code here
      # Replace the above line that calculates the hypothesis, with a call to the "calcula
      ##########################################/

      hypothesis = calculate_hypothesis(X,theta,i)

      output = y[i]
      sigma = sigma + (hypothesis - output)
  theta_0 = theta_0 - (alpha/m) * sigma
```

Run ml_ml_assgn1_1

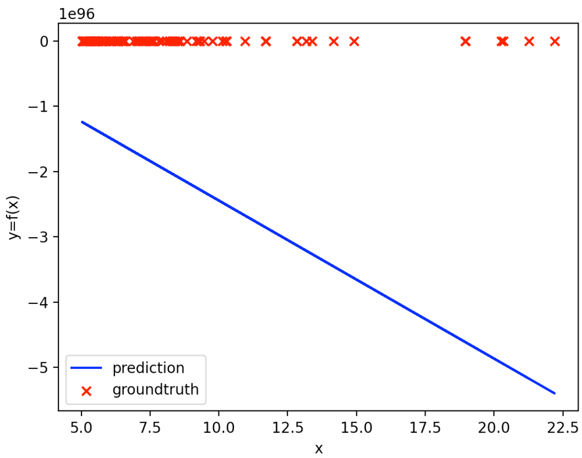
alpha = 0.001, iterations = 50

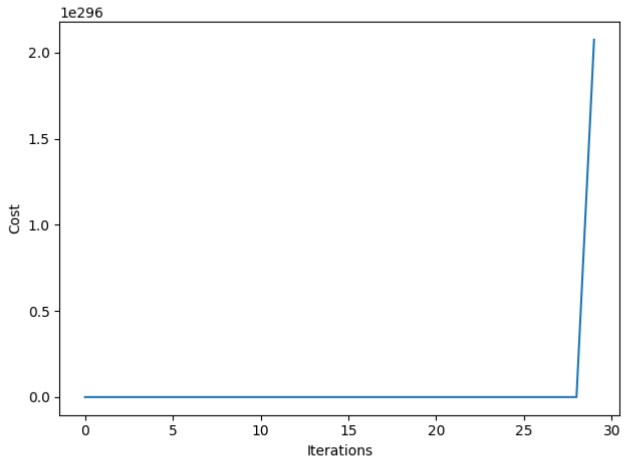





```
Gradient descent finished.
Minimum cost: 5.89503, on iteration #50

Process finished with exit code 0
```

alpha = 1, iterations = 50






```
Gradient descent finished.
Minimum cost: 176835674674.88126, on iteration #1

Process finished with exit code 0
```

With using the same iteration, the smaller alpha leads to a decreasing trend of cost , while the bigger alpha leads to an increasing trend. And it is more accurate using small alpha

2. Linear Regression with Multiple Variables

Task2.

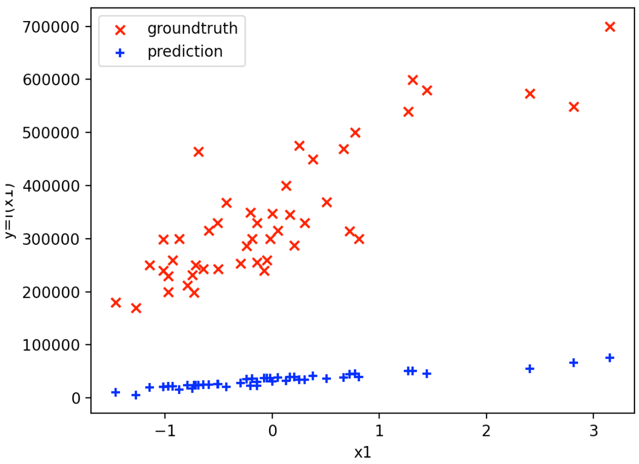
Modify the functions *calculate_hypothesis* and *gradient_descent* to support the new hypothesis function.

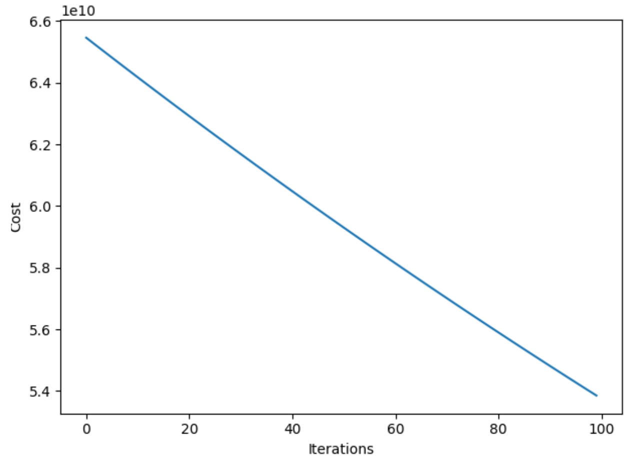
```python
def calculate_hypothesis(X, theta, i):
    """
        :param X            : 2D array of our dataset
        :param theta        : 1D array of the trainable parameters
        :param i            : scalar, index of current training sample's row
    """

    ########################################
    # 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 = 0
    m = X.shape[1]
    for j in range(m):
        hypothesis = hypothesis + X[i, j] * theta[j]
    return hypothesis
```

```python
    sigma = np.zeros((len(theta)))
    for i in range(m):
        ########################################
        # Write your code here
        # Calculate the hypothesis for the i-th sample of X, with a call to t
        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
        for j in range(len(theta)):
            sigma[j] = sigma[j] + (hypothesis - output) * X[i, j]
        ########################################/

    # update theta_temp
    ########################################
    # Write your code here
    # Update theta_temp, using the values of sigma
    for i in range(len(sigma)):
        theta_temp[i] = theta_temp[i] - (alpha / m) * sigma[i]

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

Run ml_assgn1_2.py

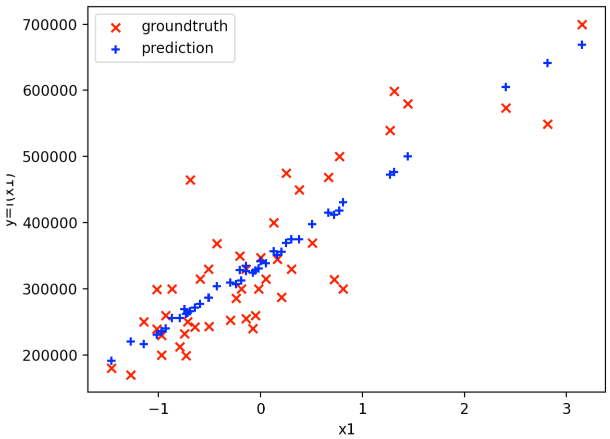
alpha = 0.001, iterations = 100

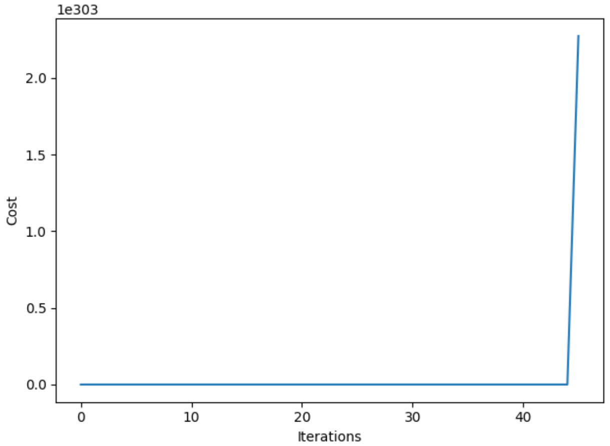





```
theta_final : [32409.9584134   9932.44103785  4936.53500492]
```

alpha = 1, iterations = 100






```
theta_final : [-2.05951806e+105 -1.68803653e+114 -1.68803653e+114]
```

With using the same iteration, the smaller alpha leads to a big theta, while the bigger alpha leads to a small theta.

How much does your algorithm predicts that a house with 1650 sq. ft. and 3 bedrooms cost? How about 3000 sq. ft. and 4 bedrooms?

```
##########################################/
# print("Mean", mean_vec)
# print("std", std_vec)
X1_new = ([[1650,3]] - mean_vec)/std_vec
X1_normalized = np.append(np.ones((X1_new.shape[0], 1)), X1_new, axis=1)
print('the price with 1650 sq.ft. and 3 rooms',np.dot(X1_normalized,theta_final))

X2_new = ([[3000,4]] - mean_vec)/std_vec
# print(X_normalized)
X2_normalized = np.append(np.ones((X2_new.shape[0], 1)), X2_new, axis=1)
print('the price with 3000 sq.ft. and 4 rooms',np.dot(X2_normalized,theta_final))
```

the price with 1650 sq. ft. and 3 rooms [26863.53624697]

the price with 3000 sq. ft. and 4 rooms [50475.86803996]

3. Regularised Linear Regression

Task 3.
Modify *gradient_descent* to use the compute_cost_regularised method instead of *compute_cost*

```
# append current iteration's cost to cost_vector
iteration_cost = compute_cost_regularised(X, y, theta,lambda_value)
cost_vector = np.append(cost_vector, iteration_cost)
```

Modify *gradient_descent* to incorporate the new cost function

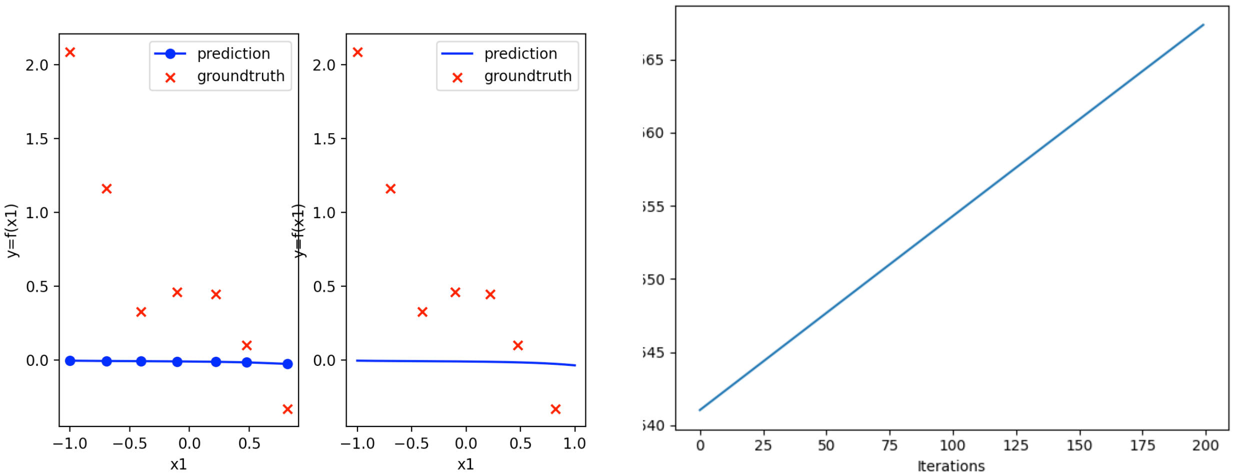
```
for j in range(len(theta)):
    sigma[j] = sigma[j] + (hypothesis - output) * X[i, j]

# update theta_temp
##########################################
# Write your code here
# Update theta_temp, using the values of sigma
# Make sure to use lambda, if necessary
for i in range(len(sigma)):
    if i is 0:
        theta_temp[i] = theta_temp[i] - (alpha / m) * sigma[i]
    else:
        theta_temp[i] = (theta_temp[i] * (1 - (alpha * lambda_value) / m)) - (alpha / m) * sigma[i]
##########################################/
```

Add param to *gradient_descent*

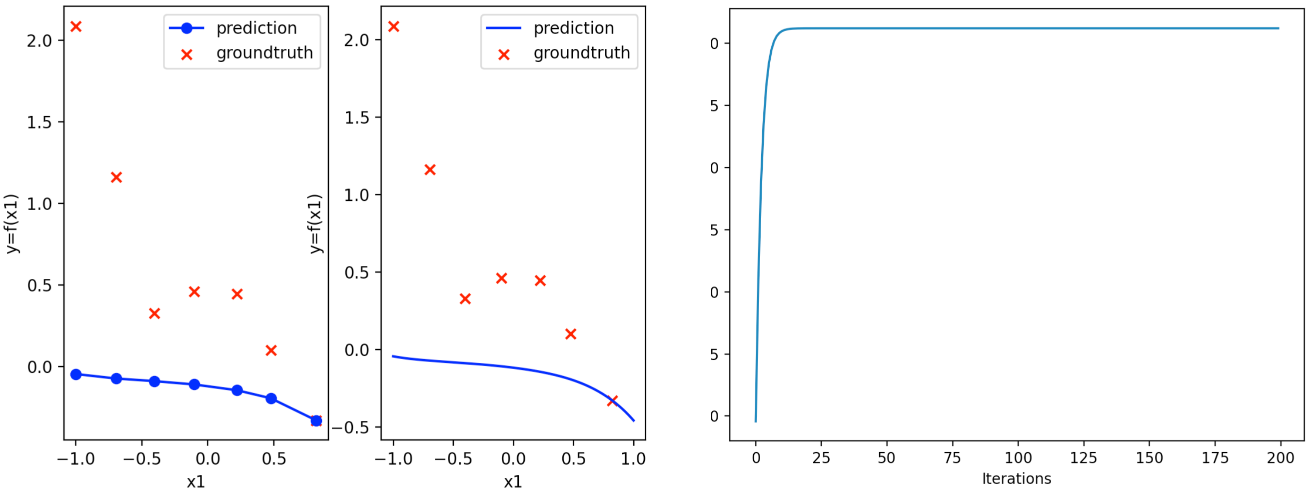
```
def gradient_descent(X, y, theta, alpha, iterations, do_plot,lambda_value):
    """
        :param X            : 2D array of our dataset
        :param y            : 1D array of the groundtruth labels of the dataset
        :param theta        : 1D array of the trainable parameters
        :param alpha        : scalar, learning rate
        :param iterations   : scalar, number of gradient descent iterations
        :param do_plot      : boolean, used to plot groundtruth & prediction values during the gradient descent i
    """
```

Run ml_assgn1_3.py

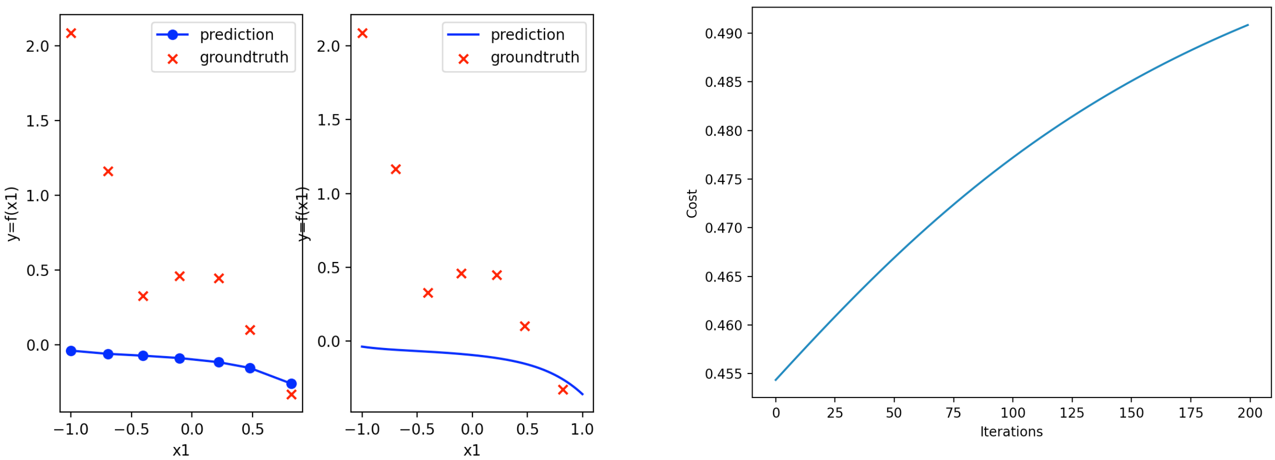
Alpha = 0.001 iterations = 200 lambda = 0.2



Alpha = 1 iterations = 200 lambda = 0.0



Alpha = 0.02 iterations = 200 lambda = 0.1



According to the patterns, the best lambda value is 0, with increase of iteraions, the cost becoming 0.