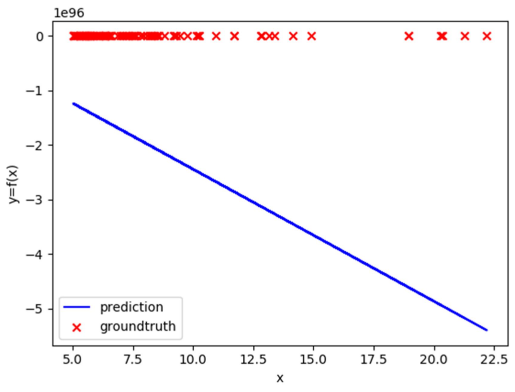
1. Linear Regression with One Variable

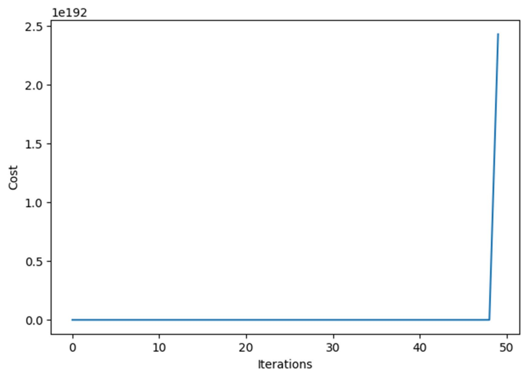
a. Task 1

```
#########################################
# Write your code here
# You must calculate the hypothesis for the i-th sample of X, given X, theta and
i.
hypothesis = X[i].dot(theta)
#########################################
```

```
#########################################
# 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)
#########################################/
```

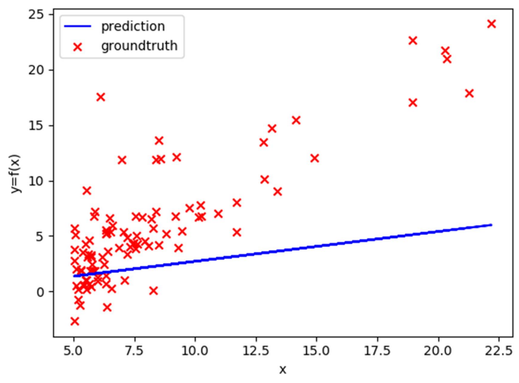
At alpha = 1, iterations = 50

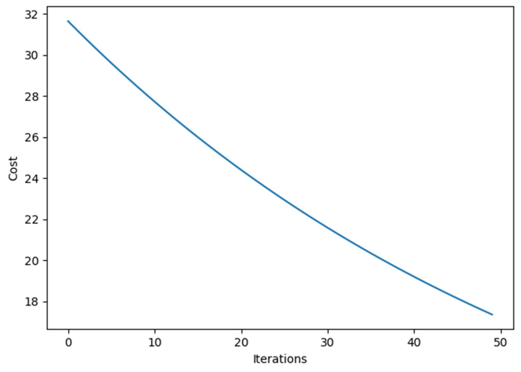




When alpha is a large quantity, the gradient overshoots and the optimal solution is missed.

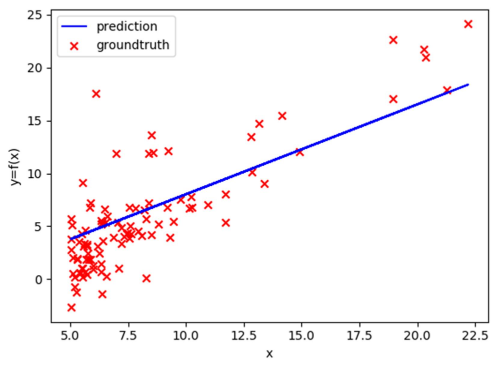
At alpha = 0.0001, iteration = 50

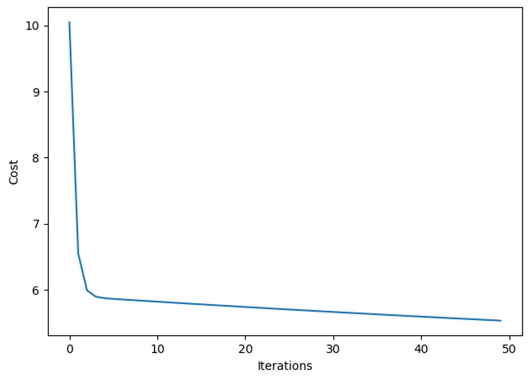




When the gradient is very, the optimal value is not reached because of the limited iterations.

At alpha = 0.0178, iteration = 50





2. Linear Regression with Multiple Variables

a. Task 2

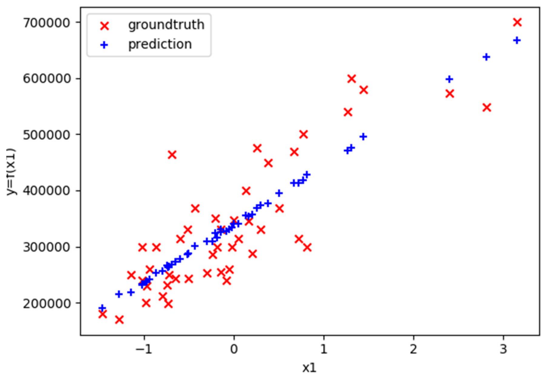
```
########################################
# Write your code here
# You must calculate the hypothesis for the i-th sample of X, given X, theta and
i.
hypothesis = X[i].dot(theta)
########################################/
```

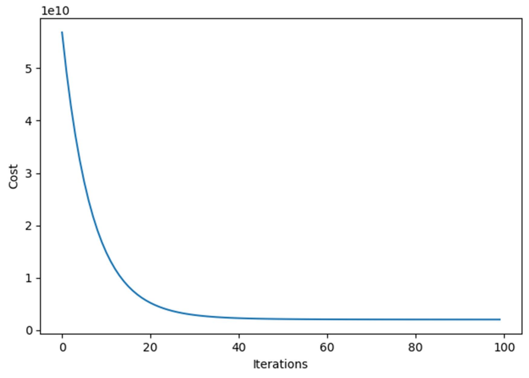
```
########################################
# Write your code here
# Calculate the hypothesis for the i-th sample of X, with a call to the
"calculate_hypothesis" function
hypothesis = calculate_hypothesis(X, theta, i)
########################################/
```

```
########################################
# 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]
########################################/
```

```
########################################
# Write your code here
# Update theta_temp, using the values of sigma
theta_temp = theta_temp - (alpha / m) * sigma
########################################/
```

At alpha = 0.07 and iterations = 100





The final_theta values are [340172.61021136 106907.71761397 -4039.25417931].

It can be noticed that the theta value is very large.

```
##########################################
# Write your code here
# Create two new samples: (1650, 3) and (3000, 4)
# Calculate the hypothesis for each sample, using the trained parameters
theta_final
# Make sure to apply the same preprocessing that was applied to the training data
# Print the predicted prices for the two samples

new_sample = np.array([[1, 1650,3], [1, 3000,4]])
x1_mean = mean_vec[0][0]
x2_mean = mean_vec[0][1]
x1_standard = std_vec[0][0]
x2_standard = std_vec[0][1]
for i in range(len(new_sample)):
    new_sample[i][1] = (new_sample[i][1] - x1_mean)/x1_standard
    new_sample[i][2] = (new_sample[i][2] - x2_mean) / x2_standard
print("The prediction for sample [1650, 3] is : ", new_sample[0].dot(theta_final))
print("The prediction for sample [3000, 4] is : ", new_sample[1].dot(theta_final))
#########################################/
```

The prediction for sample [1650, 3] is : 340172.61021136347

The prediction for sample [3000, 4] is : 443041.0736460278

3. Regularized Linear Regression

a. Task 3

```
##########################################
# 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)
#########################################/
```

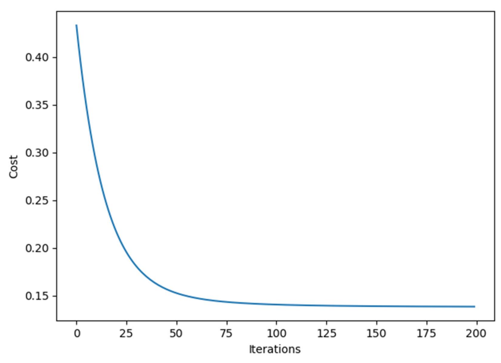
```
##########################################
# Write your code here
# Adapt the code, to compute the values of sigma for all the elements of theta
```

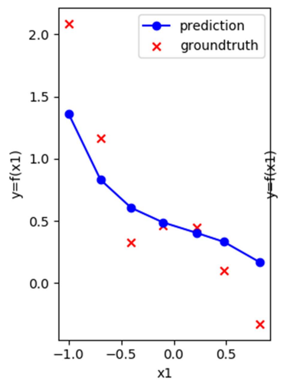
```
sigma = sigma + (hypothesis - output) * X[i]
###########################################/
```

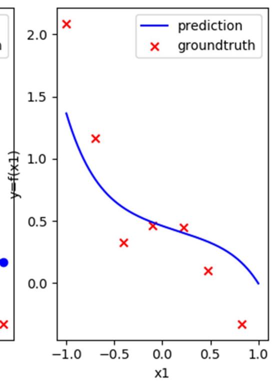
```
###########################################
# Write your code here
# Update theta_temp, using the values of sigma
# Make sure to use lambda, if necessary
temp_bias = theta_temp[0]
theta_temp = theta_temp*(1 - (alpha*(l/m))) - (alpha / m) * sigma
theta_temp[0] = temp_bias - (alpha / m) * sigma[0]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

```
# 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)
```

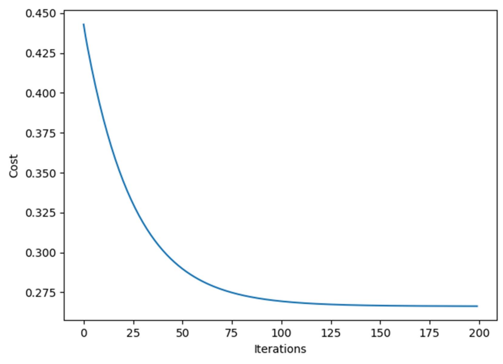
Alpha = 0.02, l = 5, iteration = 200

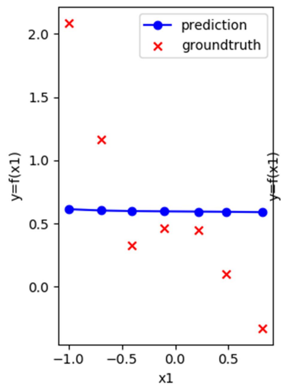


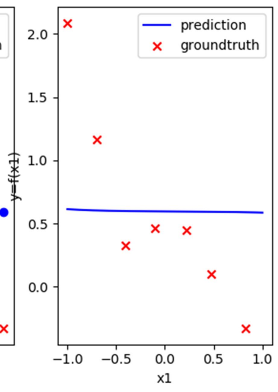




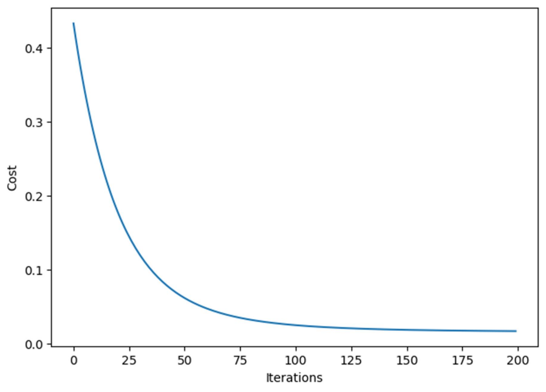
Alpha = 0.02, l = 500, iteration = 200

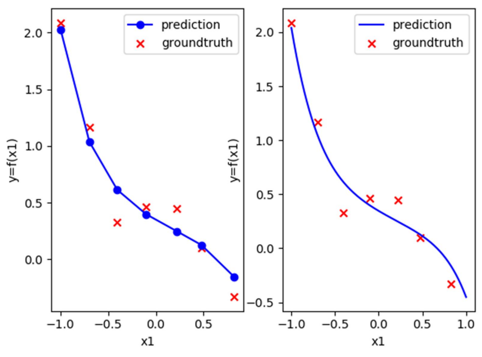






Alpha = 0.02, l = 0.1, iteration = 200


Cost
0.0
0.1
0.2
0.3
0.4
0
25
50
75
100
125
150
175
200
Iterations


prediction
groundtruth
y=f(x1)
x1
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
prediction
groundtruth
y=f(x1)
x1
−0.5
0.0
0.5
1.0
1.5
2.0