Machine Learning (CS696)

Fall 2017

Assignment 1

Submitted by

Ashok Kumar Shrestha
#A25265227

Simple Linear Regression

Training data.

X	Y
2.0	5.1
2.5	6.1
3.0	6.9
3.5	7.8
4.0	9.2
4.5	9.9
5.0	11.5
5.5	12.0
6.0	12.8

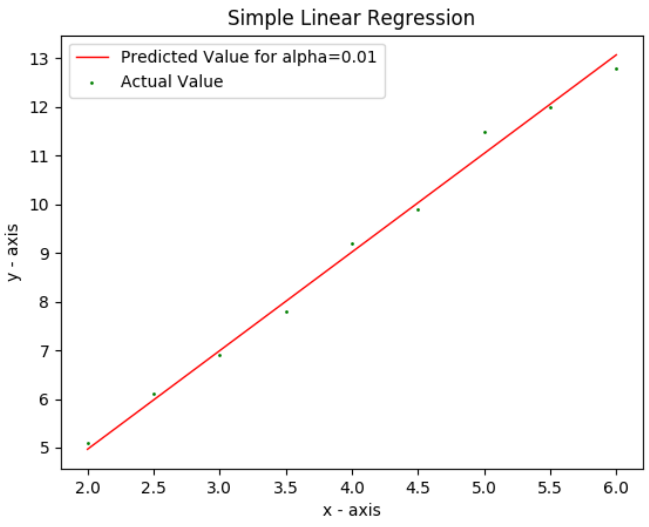
Source Code:

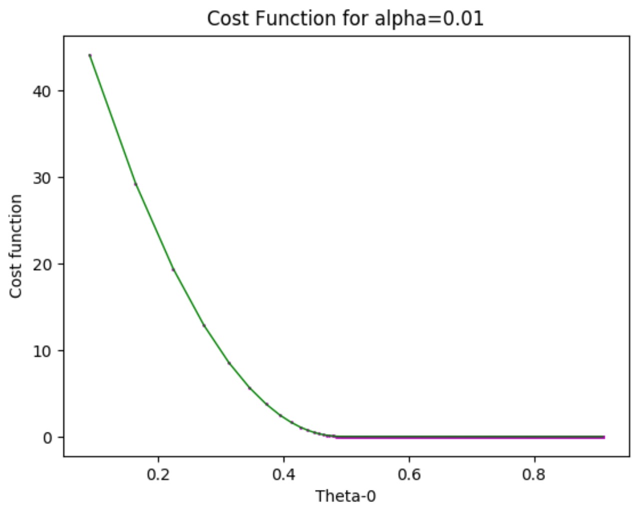
```
import matplotlib.pyplot as plt
import numpy as np

#Hypothesis function
def hypothesis(x_row,theta):
    return (theta[0] + theta[1]*x_row)

#calculate cost funtion
def compute_cost(x,y,n,m,theta):
    j_theta = sum([(hypothesis(x[i],theta)-y[i])**2 for i in range(m)])/(2*m)
    return j_theta

#calculate new values of thetas using gradient descent
def gradient_descent(x,y,m,theta,alpha):
    n_theta = [0,0]
    n_theta[0] = theta[0] - alpha*(sum([(hypothesis(x[i],theta)-y[i]) for i in range(m)]))/(m)
    n_theta[1] = theta[1] - alpha*(sum([(hypothesis(x[i],theta)-y[i])*x[i] for i in range(m)]))/(m)
    return n_theta
```

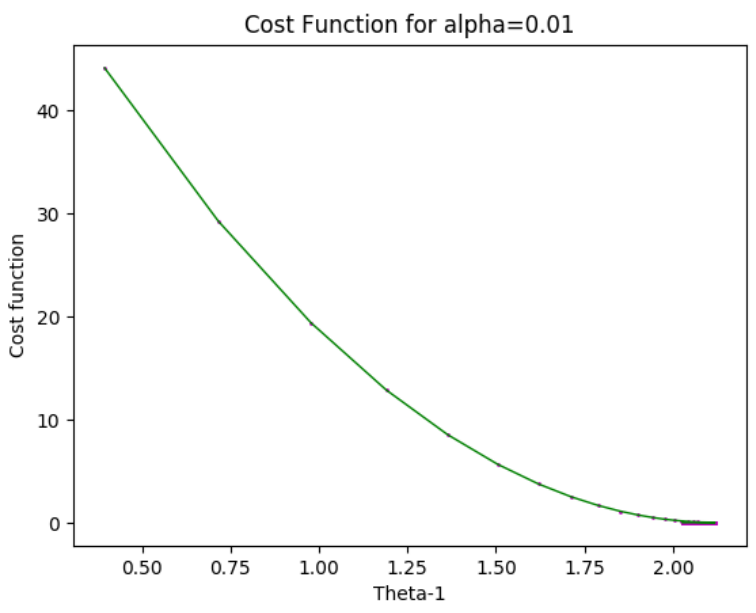
```
#Plot given data and predicted value in graph
def draw_xymodel(x,y,alpha):
    plt.title('Simple Linear Regression')
    plt.xlabel('x - axis')
    plt.ylabel('y - axis')
    plt.scatter(x,y,label="Actual Value",color="g",marker="o",s=1)

    approx_value = [hypothesis(x_row,theta) for x_row in x]
    plt.plot(x, approx_value, label="Predicted Value for alpha="+str(alpha), color='r', linewidth = 1)

    plt.legend()
    plt.show()

#plot cost function in graph
def draw_cost_function(x_arr, j_list, xlable_name, alpha):
    plt.title('Cost Function for alpha='+str(alpha))
    plt.xlabel(xlable_name)
    plt.ylabel('Cost function')
    plt.plot(x_arr, j_list, color='g', linewidth = 1)
    plt.scatter(x_arr, j_list, color="m",marker="o",s=1)
    #plt.legend()
    plt.show()

if __name__ == '__main__':

    x = np.array([2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0])
    y = np.array([5.1,6.1,6.9,7.8,9.2,9.9,11.5,12.0,12.8])

    np.seterr(all = 'raise')
    #assume theta0 and theta1 values initially
    theta = np.array([0.0,0.0])
    iterations = 1500
    alpha = 0.01

    j_list = []         #list of j-theta value
    t_list = []         #list of thetas value
    x_arr = []          #list of no. of iterations
    m = len(y)          #training samples
    n = len(theta)      #features
    is_converged = True;

    print('Alpha: ' + str(alpha))
    print('-----------Initial Parameters-----------')
    print('Theta-0: {0:0.2f} \nTheta-1: {1:0.2f}'.format(*theta))

    try:
        for i in range(iterations):
            j_theta = compute_cost(x,y,n,m,theta)
            j_list.append(j_theta)
            theta = gradient_descent(x,y,m,theta,alpha)
            t_list.append(theta)
    except FloatingPointError as e:
        #print(e)
        is_converged = False;
            print('\nNot able to converge!')
            theta = [0,0]

    if(is_converged):
        print('\n-----------Final Parameters-----------')
        print('Theta-0: {0:0.2f} \nTheta-1: {1:0.2f}'.format(*theta))
            print('\ny = '+str(theta[0])+' + '+str(theta[1])+'*x')

    draw_xymodel(x,y,alpha)

    zipped = zip(*t_list)
    draw_cost_function(zipped[0],j_list,'Theta-0',alpha)
    draw_cost_function(zipped[1],j_list,'Theta-1',alpha)

    x_arr = np.array([i for i in range(len(j_list))])
    draw_cost_function(x_arr,j_list,'Iterations',alpha)
```

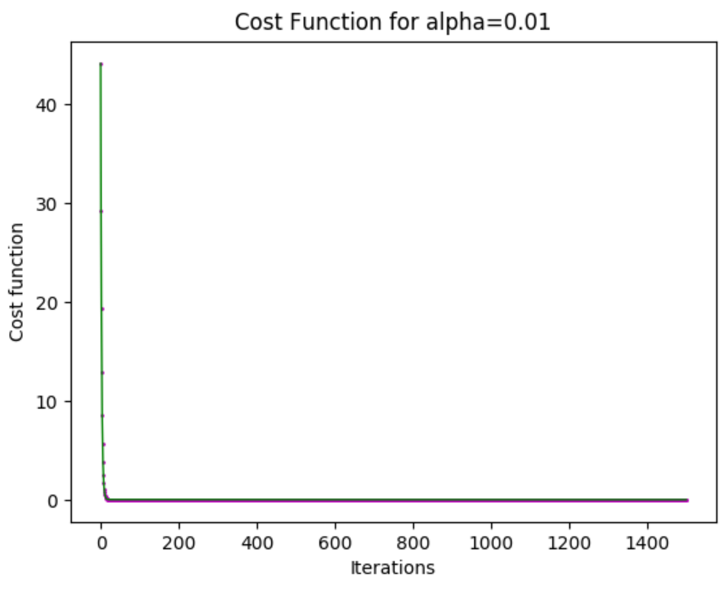
Output:

```
Alpha: 0.01
-----------Initial Parameters-----------
Theta-0: 0.00
Theta-1: 0.00

-----------Final Parameters-----------
Theta-0: 0.91
Theta-1: 2.03
y = 0.909997827398 + 2.02746944335*x
```

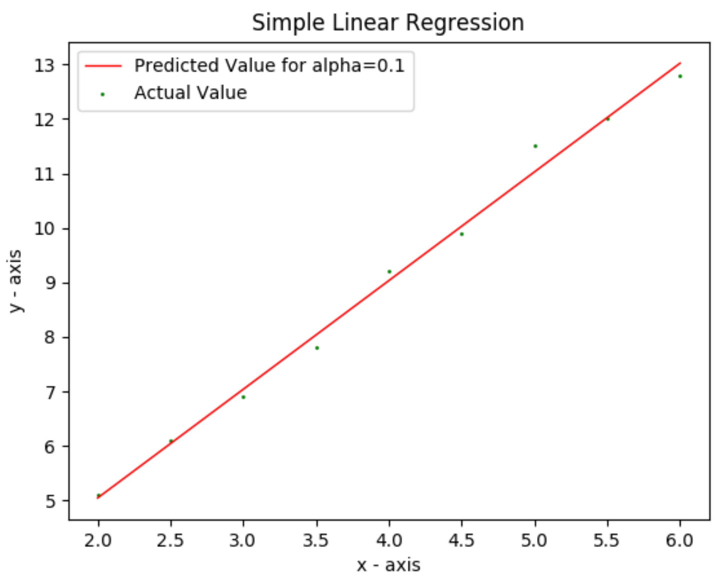


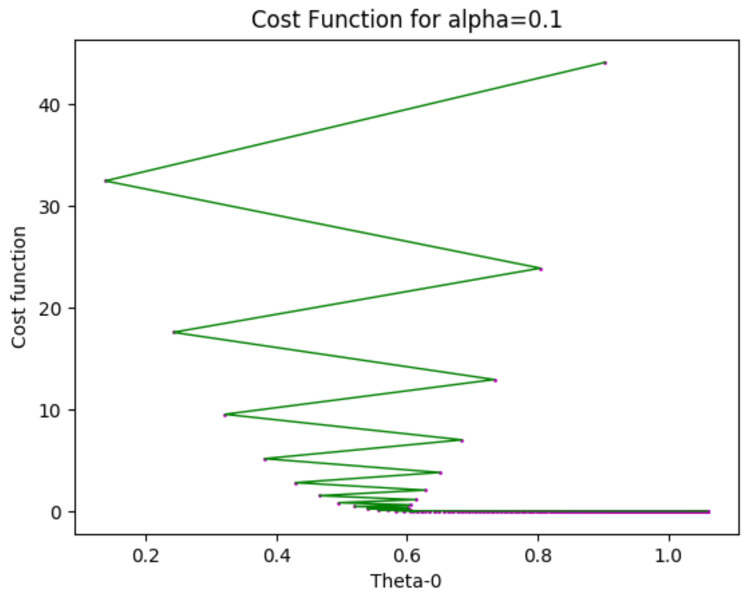


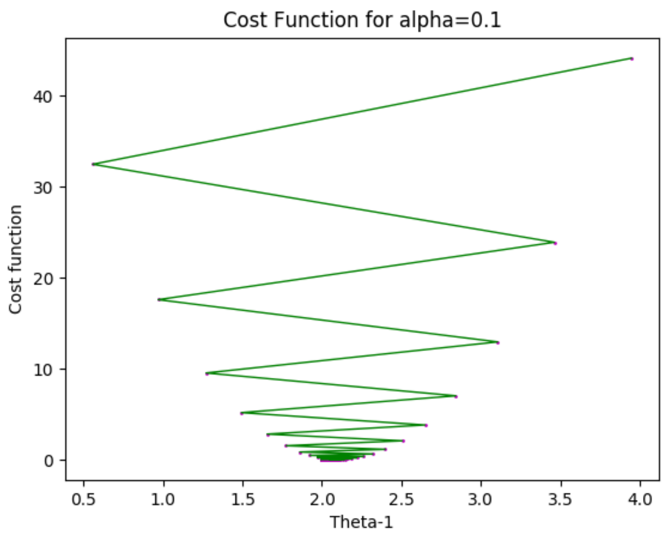


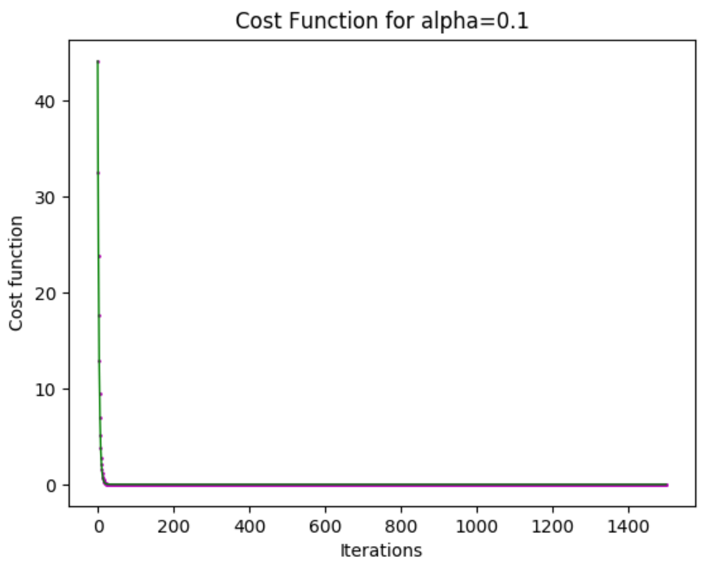


```
Alpha: 0.1
-----------Initial Parameters-----------
Theta-0: 0.00
Theta-1: 0.00

-----------Final Parameters-----------
Theta-0: 1.06
Theta-1: 1.99

y = 1.05999922393 + 1.99333350995*x
```

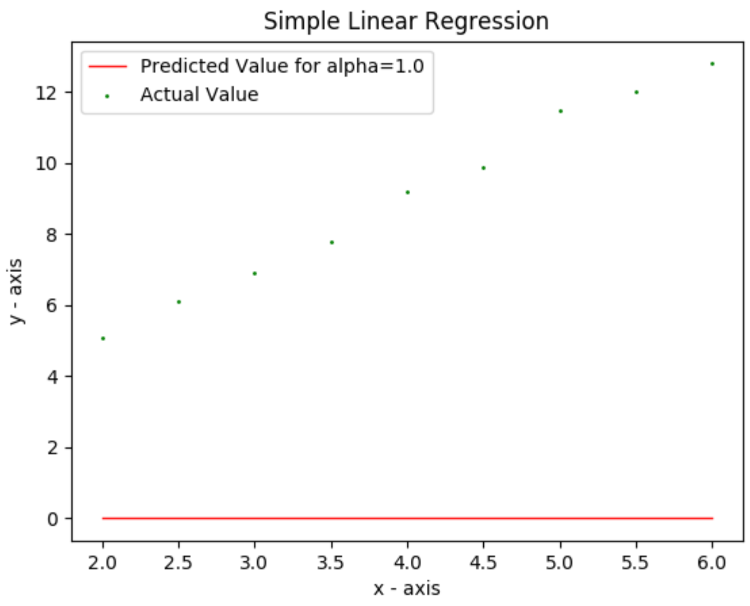


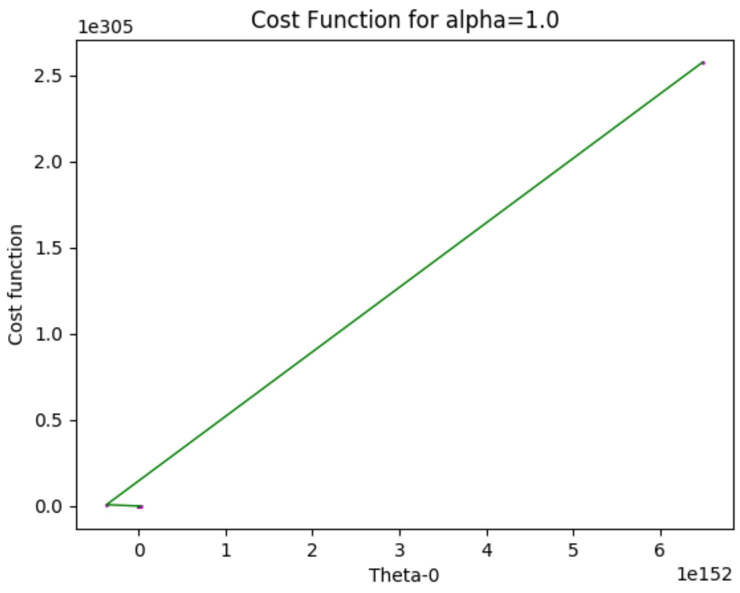


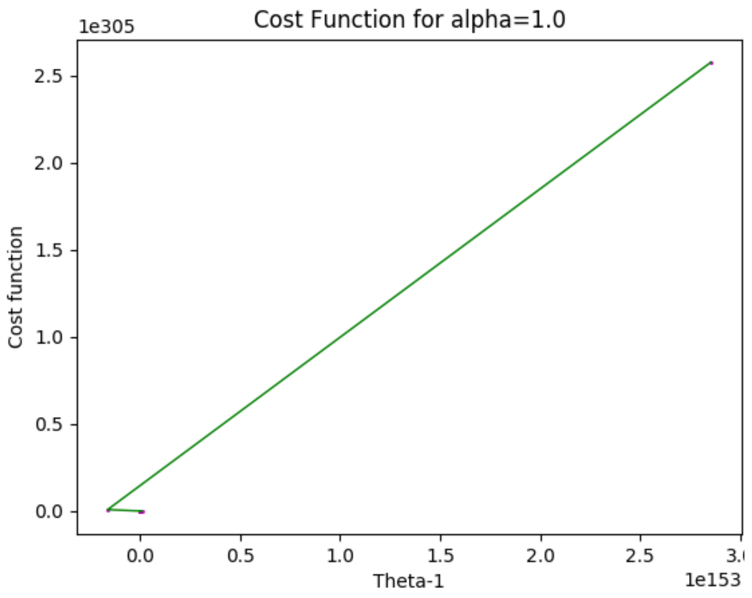


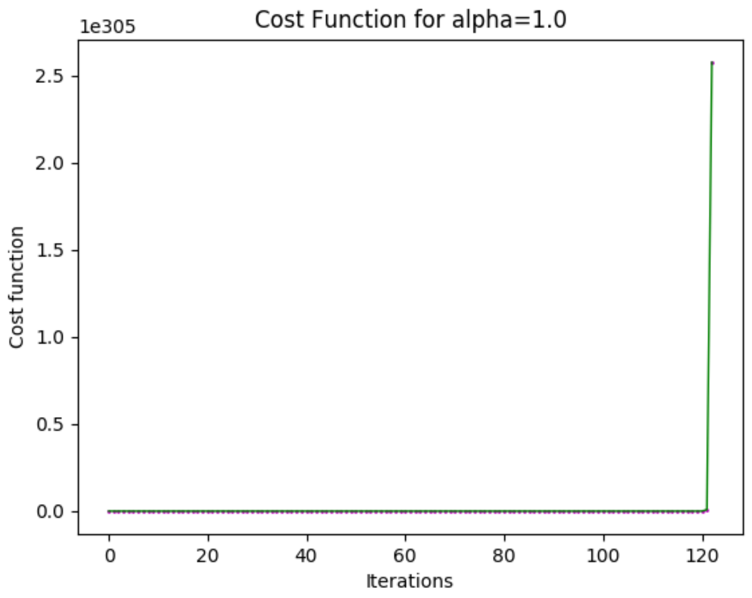


```
Alpha: 1.0
-----------Initial Parameters-----------
Theta-0: 0.00
Theta-1: 0.00

Not able to converge!
```









Observation:

For the given training samples for simple linear regression: with low value of alpha (0.01), it took longer to converge whereas for value of alpha (0.1) it took less time/iteration to converge. But, for value of alpha (1.0) it was not able to converge at all. Convergence was observed when j-theta (cost function) value decreased by very small amount for few no. of iterations. Varying initial assumed values of the parameters do not effect the convergence.

Conclusion:

The learning rate (alpha) is key factor in obtaining convergence over no. of iterations. If the alpha value is small j-theta decreases for every iteration but if the alpha value is too small then gradient descent algorithm will take large no. of iterations to reach convergence. But if the alpha value is too large j-theta value increases by huge magnitude and may never reach convergence for even large no. of iterations.

Multiple Linear Regression

Training data.:

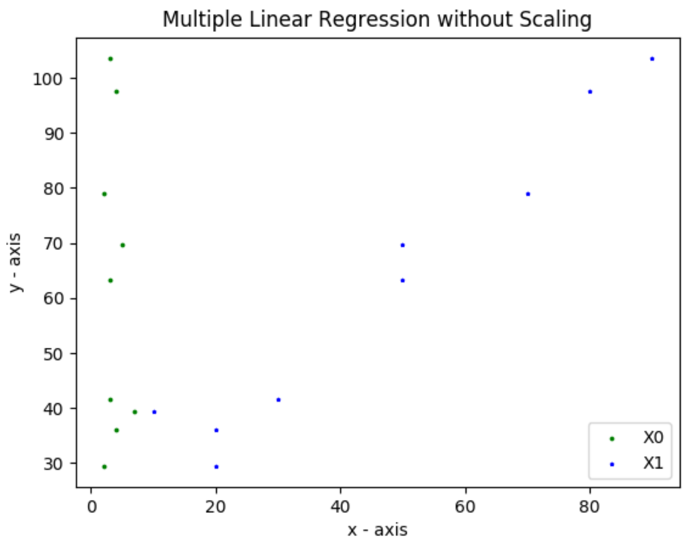
X_1	X_2	y
2.0	70.0	79.4
3.0	30.0	41.5
4.0	80.0	97.5
4.0	20.0	36.1
3.0	50.0	63.2
7.0	10.0	39.5
5.0	50.0	69.8
3.0	90.0	103.5
2.0	20.0	29.5

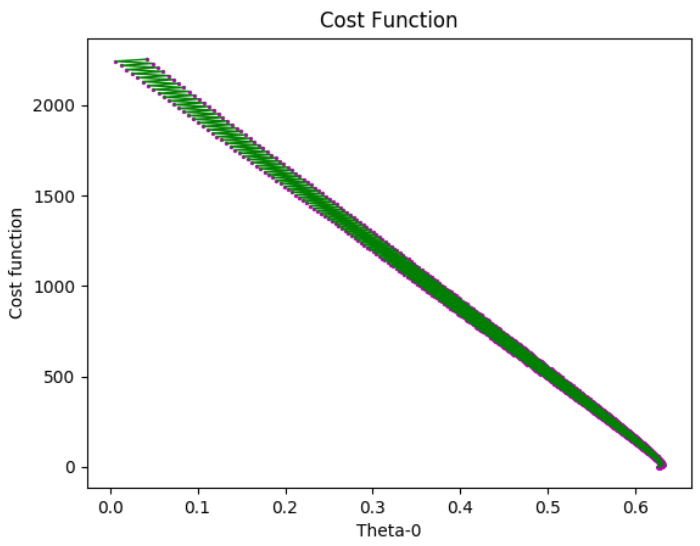
Source Code:

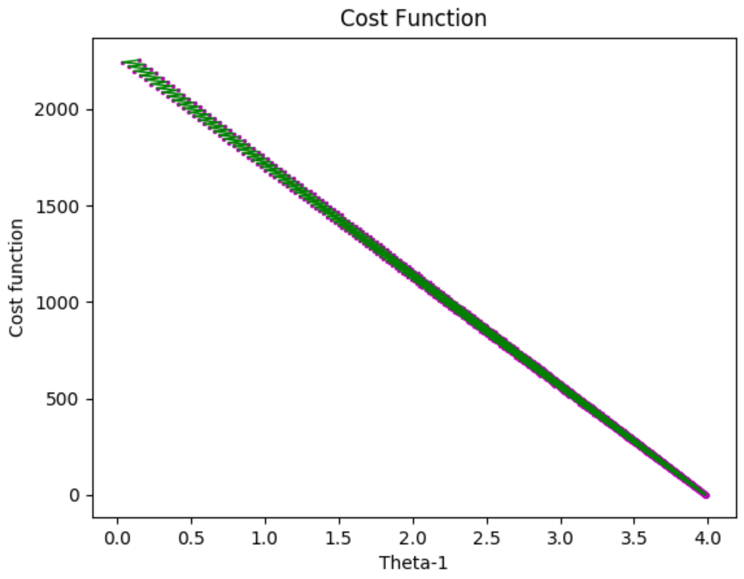
```
import matplotlib.pyplot as plt
import numpy as np

#Hypothesis function
def hypothesis(x_row,n,theta):
    return sum([theta[i]*x_row[i] for i in range(n)])

#calculate cost funtion
def compute_cost(x,y,n,m,theta):
    j_theta = sum([(hypothesis(x[i],n,theta)-y[i])**2 for i in range(m)])/(2*m)
    return j_theta

#calculate new values of thetas using gradient descent
def gradient_descent(x,y,n,m,theta,alpha):
    n_theta = np.zeros((n,1))
    for j in range(n):
        n_theta[j] = theta[j] - alpha*(sum([(hypothesis(x[i],n,theta)-y[i])*x[i][j] for i in range(m)]))/(m)

    return n_theta

#Feature scaling: Dividing each feature value by the range of the feature
def with_scaling(x,y,n,m,theta,alpha):
    t_list = []
    j_list = []
    is_converged = True;
    range_i = x.max(axis=0)-x.min(axis=0)
    scale_x = x
    for i in range(1,n,1):
        scale_x[:,i] = x[:,i]/range_i[i]
    try:
        for i in range(iterations):
            theta = gradient_descent(scale_x,y,n,m,theta,alpha)
            t_list.append(theta)
            j_theta = compute_cost(scale_x,y,n,m,theta)
            j_list.append(j_theta)
    except FloatingPointError as e:
        #print(e)
            is_converged = False;
        theta = np.array([0 for i in range(n)])
            j_list.append(j_theta)
            print('\nNot able to converge!')

    if(is_converged):
        #scaling theta values back
            for i in range(1,n,1):
                theta[i] = theta[i]/range_i[i]

        print('\n-----------Final Parameters with Scaling-----------')
        for i in range(n):
                print('theta-'+str(i)+': '+str(*theta[i]))

            eqn = '\ny = '
            eqn += str(*theta[0])
            for i in range(1,n,1):
                eqn += ' + '+str(*theta[i]) + '*x'+str(i)
            print(eqn)

    return t_list,j_list

#Without scaling the predictor variables
def without_scaling(x,y,n,m,theta,alpha):
    is_converged = True;
    try:
        for i in range(iterations):
            theta = gradient_descent(x,y,n,m,theta,alpha)
            t_list.append(theta)
            j_theta = compute_cost(x,y,n,m,theta)
            j_list.append(j_theta)
    except FloatingPointError as e:
        #print(e)
            is_converged = False;
        theta = np.array([0 for i in range(n)])
            j_list.append(j_theta)
            print('\nNot able to converge!')
    if(is_converged):
        print('\n-----------Final Parameters without Scaling-----------')
        for i in range(n):
                print('theta-'+str(i)+': '+str(*theta[i]))

            eqn = '\ny = '
            eqn += str(*theta[0])
            for i in range(1,n,1):
                eqn += ' + '+str(*theta[i]) + '*x'+str(i)
            print(eqn)

    return t_list,j_list

#Plot given data sets in graph
def draw_xymodel(x,y,is_scaling):
    zipped = zip(*x)
    plt.title('Multiple Linear Regression '+is_scaling+' Scaling')
    plt.xlabel('x - axis')
    plt.ylabel('y - axis')
    plt.scatter(zipped[0],y,label="X0",color="g",marker="o",s=3)
    plt.scatter(zipped[1],y,label="X1",color="b",marker="*",s=3)
    plt.legend()
    plt.show()

#plot cost function in graph
def draw_cost_function(x_arr, j_list,x_lable):
    plt.title('Cost Function')
    plt.xlabel(x_lable)
    plt.ylabel('Cost function')
    plt.plot(x_arr, j_list, color='g', linewidth = 1)
    plt.scatter(x_arr, j_list, color="m", marker="o", s=2)
    plt.show()

if __name__ == '__main__':

    x = np.array([[2.0,3.0,4.0,4.0,3.0,7.0,5.0,3.0,2.0],
     [70.0,30.0,80.0,20.0,50.0,10.0,50.0,90.0,20.0]])

    y = np.array([79.0,41.5,97.5,36.1,63.2,39.5,69.8,103.5,29.5])

    np.seterr(all = 'raise')     #for raising Runtime overflow for large computation
    m = len(y)                   #training parameters
    n = x.shape[0]+1             #features
    iterations = 1500
    alpha = 0.000684

    #assume theta0 and theta1 values initially
    theta = np.array([0 for i in range(n)])

    print('Alpha: '+str(alpha))
    print('-----------Initial Parameters-----------')
    for i in range(n):
        print('theta-'+str(i)+': '+str(theta[i]))

    j_list = []     #list of j-theta value
    t_list = []     #list of thetas value
    #Adding X0=1 to first column of X data
    it = np.ones((m,n))
    it[:,1:] = np.array(zip(*x))

    t_list, j_list = without_scaling(it,y,n,m,theta,alpha)
    #t_list, j_list = with_scaling(it,y,n,m,theta,alpha)

    draw_xymodel(np.array(zip(*x)),y,'without')

    theta_list = np.array(zip(*t_list))
    for i in range(n):
      draw_cost_function(theta_list[i],j_list,'Theta-'+str(i))

    x_arr = [i for i in range(len(j_list))]          #list of no. iterations
    draw_cost_function(x_arr,j_list,'Iterations')
```

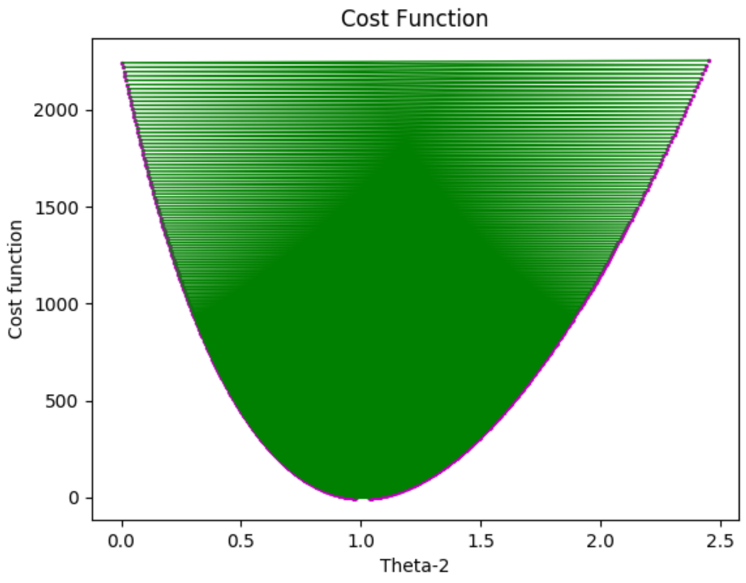
Output:

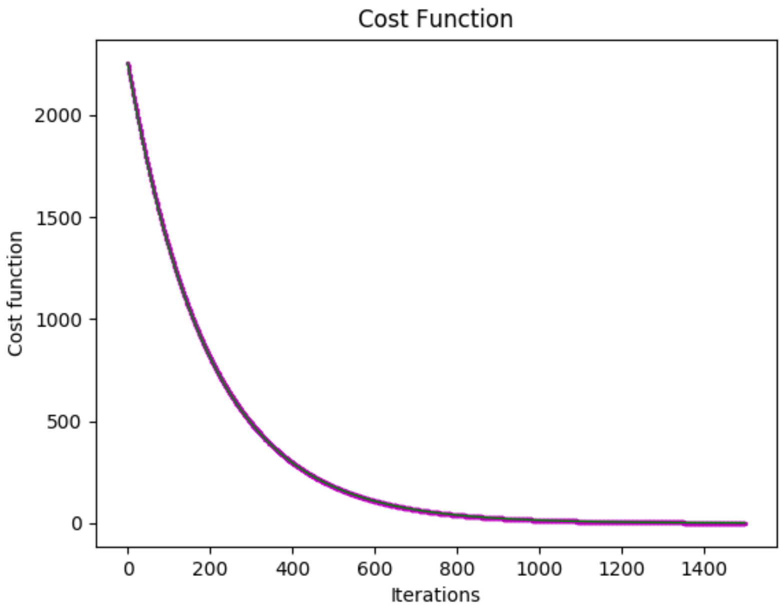
```
Alpha: 0.000684
-----------Initial Parameters-----------
theta-0: 0
theta-1: 0
theta-2: 0

-----------Final Parameters without Scaling-----------
theta-0: 0.625412906145
theta-1: 3.98201277142
theta-2: 0.977893882797

y = 0.625412906145 + 3.98201277142*x1 + 0.977893882797*x2
```


Multiple Linear Regression without Scaling
y - axis
x - axis
X0
X1

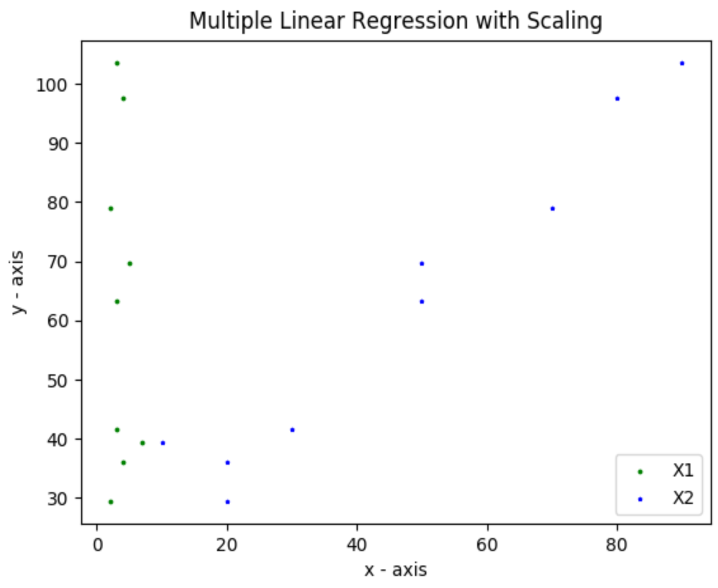

Cost Function
Cost function
Theta-0

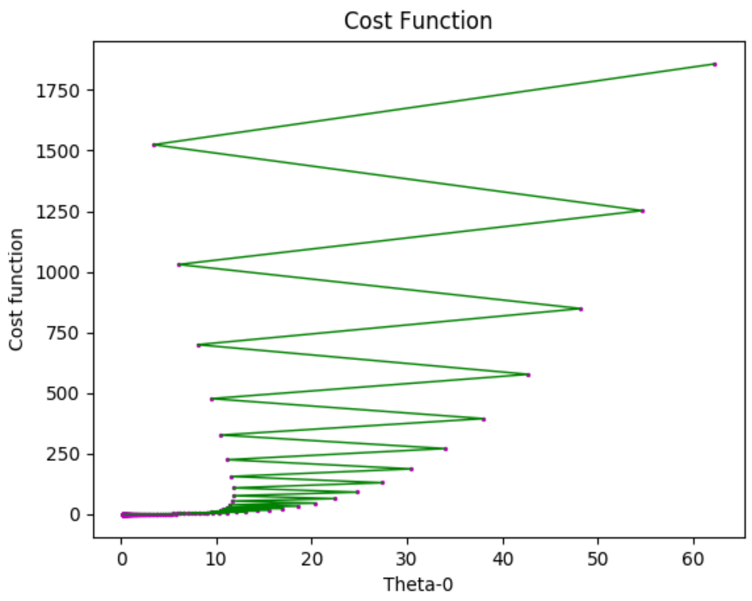

Cost Function
Cost function
Theta-1

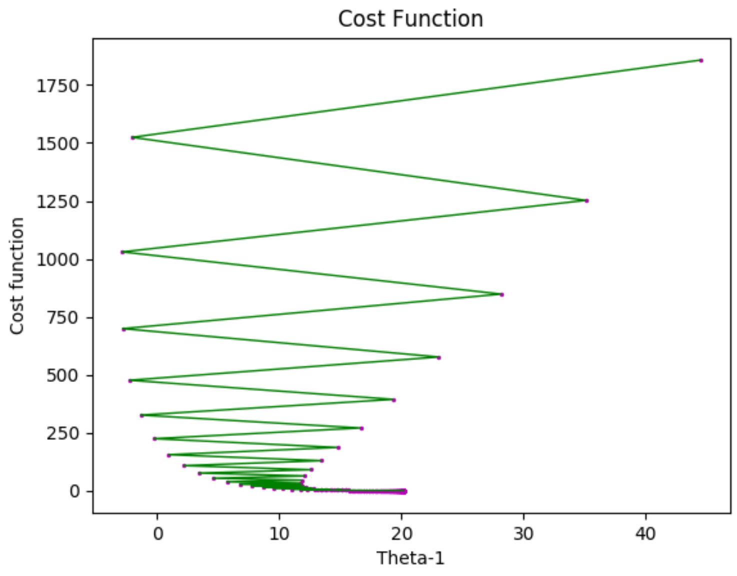

Cost Function
Cost function
Theta-2

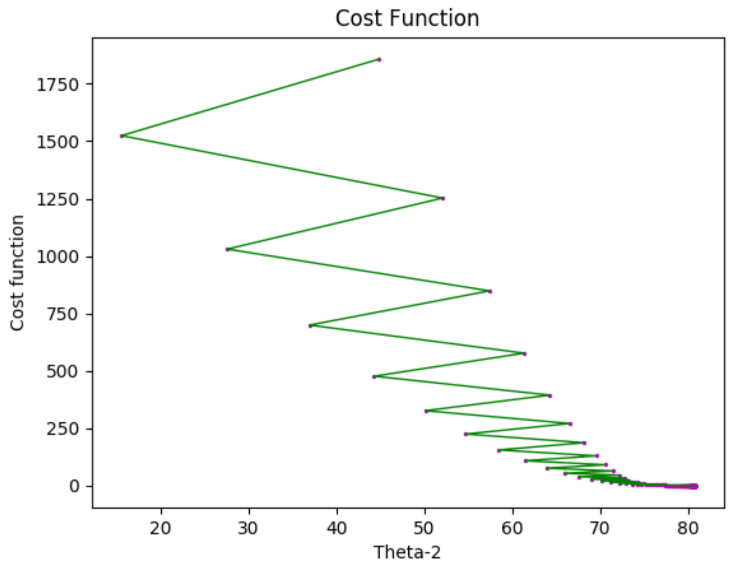

Cost Function
Cost function
Iterations

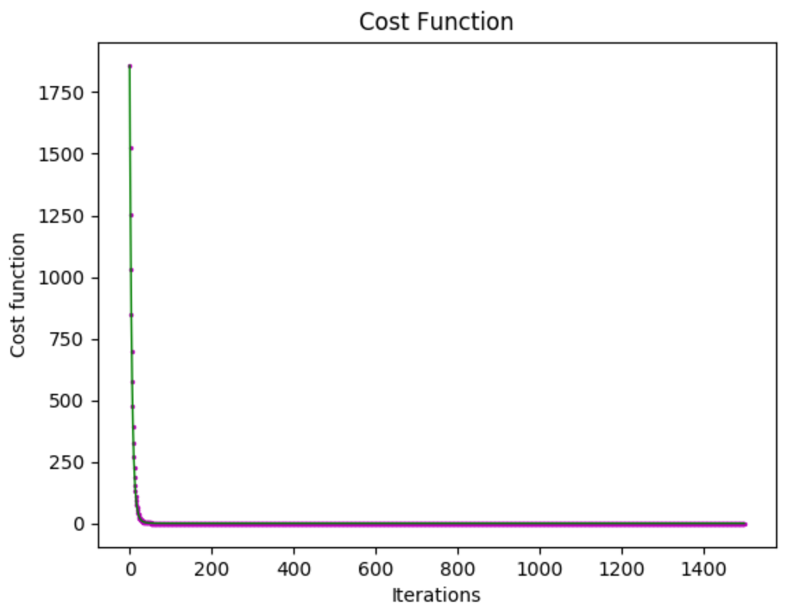
```
Alpha: 1.0
-----------Initial Parameters-----------
theta-0: 0
theta-1: 0
theta-2: 0

-----------Final Parameters with Scaling-----------
theta-0: 0.299768317405
theta-1: 4.03808282653
theta-2: 1.00867940921

y = 0.299768317405 + 4.03808282653*x1 + 1.00867940921*x2
```











Observation:
For the given training samples for multiple linear regression: low value of alpha (~0.0001) was required to reach convergence with out using feature scaling. Whereas when feature scaling was used on predictor variables (divide by range), large value of alpha (1.0) was able to reach convergence in less no. of iterations. Varying initial assumed values of the parameters do not effect the convergence.

Conclusion:
The learning rate (alpha) and the feature scaling are the key factors in obtaining convergence over no. of iterations. For multiple linear regression, when all the features are not similar (roughly) in values, feature scaling is useful for gradient algorithm to converge in less no. of iterations.