Question 1

Question 1-a

In [64]:
```
import random
from sklearn import datasets
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import scipy.linalg as la
import math
from scipy.fftpack import fft,fftfreq
from scipy.linalg import toeplitz


def derivative(theta):
    return 2*theta
def cost(theta):
    return theta**2


theta = 5
learning_rate=0.01
old=theta
for i in range(1000):
    theta = theta - (learning_rate * derivative(theta))
    if abs(old-theta)==0:
        break
    old=theta
print('theta value for learning rate of 0.01 and iterations 1
000: theta= '+str(theta))


theta = 5
learning_rate=0.1
old=theta
for i in range(100):
    theta = theta - (learning_rate * derivative(theta))
    if abs(old-theta)==0:
        break
    old=theta
print("theta value for learning rate of 0.1 and iterations 10
0: theta= "+str(theta))
```

```
theta value for learning rate of 0.01 and iterations 1000:
theta= 8.414836786079803e-09
theta value for learning rate of 0.1 and iterations 100: th
eta= 1.0185179881672439e-09
```

Question 1-b

In [65]:
```
def derivative(theta):
    return 2*theta
def cost(theta):
    return theta[0]**2+theta[1]**2

#Question 1-b
theta = [5,7]
learning_rate=0.01
old=theta.copy()

for i in range(1000):
    theta[0] = old[0] - (learning_rate * derivative(old[0]))
    theta[1] = old[1] - (learning_rate * derivative(old[1]))

    if abs(old[0]-theta[0])==0 and abs(old[1]-theta[1])==0:
        break
    old=theta.copy()
print("theta values for learning rate =0.01 and iterations=10
00 : theta_0=" +str(theta[0])+"  theta_1="+str(theta[1]))

theta = [5,7]
learning_rate=0.1
old=theta.copy()

for i in range(100):
    theta[0] = old[0] - (learning_rate * derivative(old[0]))
    theta[1] = old[1] - (learning_rate * derivative(old[1]))

    if abs(old[0]-theta[0])==0 and abs(old[1]-theta[1])==0:
        break
    old=theta.copy()
print("theta values for learning rate =0.1 and iterations=100
: theta_0=" +str(theta[0])+"  theta_1="+str(theta[1]))
```

```
theta values for learning rate =0.01 and iterations=1000 :
theta_0=8.414836786079803e-09  theta_1=1.1780771500511704e-
08
theta values for learning rate =0.1 and iterations=100 : th
eta_0=1.0185179881672439e-09  theta_1=1.4259251834341386e-0
9
```

Question 1-c

In [66]:
```
def derivative(theta):
    return 2*(theta-1)
def cost(theta):
    return (theta-1)**2

#Question 1-c
theta = 7
learning_rate=0.01
old=theta

for i in range(1000):
    theta = theta - (learning_rate * derivative(theta))
    if abs(old-theta)==0:
        break
    old=theta
print('theta value for learning rate of 0.01 and iterations 1
000: theta= '+str(theta))
theta = 7
learning_rate=0.1
old=theta

for i in range(100):
    theta = theta - (learning_rate * derivative(theta))
    if abs(old-theta)==0:
        break
    old=theta
print('theta value for learning rate of 0.1 and iterations 10
0: theta= '+str(theta))
```

```
theta value for learning rate of 0.01 and iterations 1000:
theta= 1.0000000100978041
theta value for learning rate of 0.1 and iterations 100: th
eta= 1.0000000012222217
```

Question 1-d

In [67]:
```
def derivative(theta):
    return 2*(theta-1)
def cost(theta):
    return (theta[0]-1)**2+(theta[1]-1)**2

#Question 1-d
theta = [5,7]
learning_rate=0.01
old=theta.copy()

for i in range(1000):
    theta[0] = old[0] - (learning_rate * derivative(old[0]))
    theta[1] = old[1] - (learning_rate * derivative(old[1]))

    if abs(old[0]-theta[0])==0 and abs(old[1]-theta[1])==0:
        break
    old=theta.copy()
print("theta values for learning rate =0.01 and iterations=10
00 : theta_0=" +str(theta[0])+"  theta_1="+str(theta[1]))

theta = [5,7]
learning_rate=0.1
old=theta.copy()

for i in range(100):
    theta[0] = old[0] - (learning_rate * derivative(old[0]))
    theta[1] = old[1] - (learning_rate * derivative(old[1]))

    if abs(old[0]-theta[0])==0 and abs(old[1]-theta[1])==0:
        break
    old=theta.copy()
print("theta values for learning rate =0.1 and iterations=100
: theta_0=" +str(theta[0])+"  theta_1="+str(theta[1]))
```

```
theta values for learning rate =0.01 and iterations=1000 :
theta_0=1.000000006731869  theta_1=1.0000000100978041
theta values for learning rate =0.1 and iterations=100 : th
eta_0=1.0000000008148144  theta_1=1.0000000012222217
```

Question 1-e

In [68]:

```
def cost_derivative(thetain,X,Y,alpha):
    y_hat=np.dot(X,thetain)
    error=y_hat-Y
    cost=np.dot(error.T,error)
    thetaout=thetain-(alpha*2*np.dot(X.T,error))
    return thetaout,cost
#Question 1-e
df = pd.read_csv('assign2.csv')
X=df.iloc[:,0:1]
Y=df.iloc[:,1:2]
X1=X.copy()
meanx=X1.mean()
stdx=X1.std()
X1=np.array(X1)
X1=X1.reshape(len(X1),)
X = (X - X.mean()) / X.std()
X=np.c_[ np.ones(X.shape[0]),X ]
alpha=0.01
Y=np.array(Y)
Y=Y.reshape(len(Y),)
np.random.seed(143)
theta=np.random.rand(2)
old=theta.copy()
oldc=0
for i in range(100):
    theta,cost=cost_derivative(old,X,Y,alpha)
    if i!=0 and oldc<cost:
        print("learning rate is too high")
        break
    if i!=0 and oldc-cost==0:
        break
    oldc=cost
    old=theta.copy()

theta[0]=theta[0]-theta[1]*meanx/stdx
theta[1]=theta[1]/stdx
print("theta values for learning rate =0.01 and iterations=10
0: theta_0=" +str(theta[0])+"  theta_1="+str(theta[1]))
print("\n")
alpha=0.1
theta=np.random.rand(2)
old=theta.copy()
oldc=0
for i in range(10):
    theta,cost=cost_derivative(old,X,Y,alpha)
    if i!=0 and oldc<cost:
        print("learning rate is too high for learning rate=
"+str(alpha))
        break
```

```
    if i!=0 and oldc-cost==0:
        break
    oldc=cost
    old=theta.copy()

theta[0]=theta[0]-theta[1]*meanx/stdx
theta[1]=theta[1]/stdx
print("theta values for learning rate =0.1 and iterations=10
0: theta_0=" +str(theta[0])+"  theta_1="+str(theta[1]))
```

```
theta values for learning rate =0.01 and iterations=100: th
eta_0=49.237557841005625  theta_1=-0.008611932237692523
```

```
learning rate is too high for learning rate= 0.1
theta values for learning rate =0.1 and iterations=100: the
ta_0=-15707.637988354076  theta_1=2.822830534256178
```

Observation:

Here alpha1 is 0.01 & alpha2 is 0.1. From 1a to 1d , it is clearly visible that there will be certain global minimum for given functions & one can also easily find out the optimal parameters(which can be used for best fit to get nearly accuarete prediction & absolute correct prediction in case of cost =0). Also since these functions are bound to converge the greater value of alpha(i.e. alpha2=0.1) takes significant less number of iterations. Now, in case of 1e we even don't know that the given dataset can be fit by two parameters or not. so I take random value of theta0 & theta1 for initialization & apply gradient descent for 1000 iterations. It turns out that alpha = 0.1 is quite big for this dataset. Cost value will monotnically decreases with each iterations while with alpha = 0.01 . We can safely state that theta is nearly at its optimal value.

Question 2

In [23]:
```
import random


from sklearn import datasets
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import scipy.linalg as la
import math
from scipy.fftpack import fft,fftfreq
from scipy.linalg import toeplitz
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import animation
from IPython.display import HTML, Image
import matplotlib
from matplotlib import rc
rc('animation', html='jshtml')
matplotlib.rcParams['animation.embed_limit'] = 2**128

def csot(X1,Y1,th0,th1,m):
    theta=np.ones(2,)
    theta[0]=th0
    theta[1]=th1
    y_hat=np.dot(X1,theta)
    error=y_hat-Y1
    cost=(1/(2*m))*np.dot(error.T,error)
    return cost

X=pd.core.frame.DataFrame([1,3,6])
Y=pd.core.frame.DataFrame([6,10,16])
X1=X.copy()
meanx=X1.mean()
stdx=X1.std()
X1=np.array(X1)
X1=X1.reshape(len(X1),)

X=np.c_[ np.ones(X.shape[0]),X ]
alpha=0.1
iterations=101
Y=np.array(Y)
Y=Y.reshape(len(Y),)
m = len(Y)
np.random.seed(89)
theta=np.random.rand(2)




oldc=[]
```

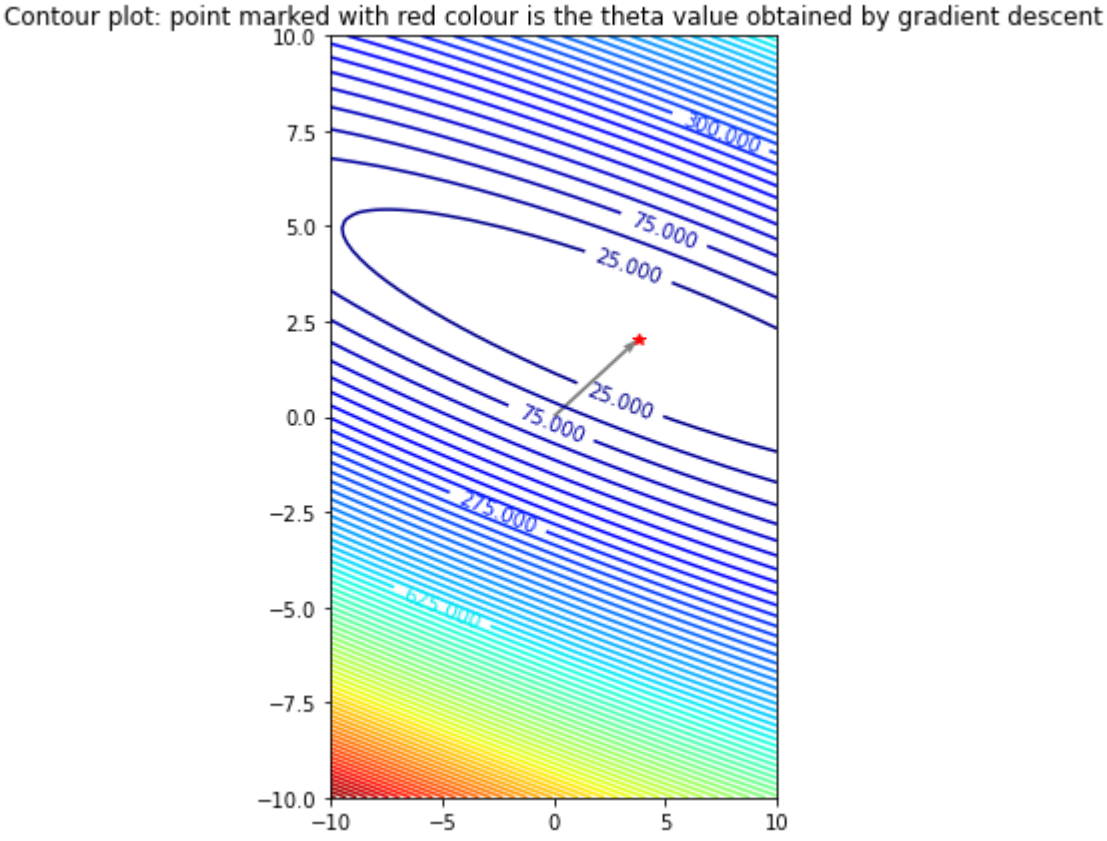
```
y_hat=np.dot(X,theta)
error=y_hat-Y
cost=(1/(2*m))*np.dot(error.T,error)
oldc.append(cost)
oldtheta=[theta]
for i in range(iterations):
    y_hat=np.dot(X,theta)
    error=y_hat-Y
    cost=(1/(2*m))*np.dot(error.T,error)
    oldc.append(cost)
    theta=theta-(alpha*(1/m)*np.dot(X.T,error))
    oldtheta.append(theta)

theta=oldtheta[-1]

print("theta_0= "+str(theta[0])+"   theta_1= "+str(theta[1]))

oldtheta=np.array(oldtheta)

th0=oldtheta[:,0]
th1=oldtheta[:,1]
anglesx = th0[1:] - th0[:-1]
anglesy = th1[1:] - th1[:-1]


th=np.linspace(-10,10,100)
x,y=np.meshgrid(th,th)
zs = np.array(  [csot(X, Y,t0,t1,m)
                     for t0, t1 in zip(np.ravel(x), np.ravel
(y)) ] )
zs = zs.reshape(x.shape)

fig, (ax1, ax) = plt.subplots(1, 2)
ax1.contour(x, y, zs, 100, cmap = 'jet')



line1, = ax1.plot([], [], 'r', label = 'Gradient descent', lw
= 1.5)
point1, = ax1.plot([], [], '.', color = 'red', markersize =
4)
value_display1 = ax1.text(0.02, 0.02, '', transform=ax1.trans
Axes)
value_display1.set_animated(True)
ax.set_title('X vs Y')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.scatter(X1, Y, color='red')
line, = ax.plot([], [], lw=2)
annotation = ax.text(0.1, 0.1, '')
```

```
annotation.set_animated(True)
def init_1():
    line1.set_data([], [])
    point1.set_data([], [])
    value_display1.set_text('')
    line.set_data([], [])
    annotation.set_text('')

    return  line1, point1, value_display1,line, annotation

def animate_1(i):

    line1.set_data(th0[:i], th1[:i])


    point1.set_data(th0[i], th1[i])


    value_display1.set_text('cost = %.2f iteration= '%oldc
[i]+str(i))
    xq = np.linspace(np.min(X1)-2,np.max(X1)+1,500)
    yq = oldtheta[i][1]*xq + oldtheta[i][0]
    line.set_data(xq, yq)
    annotation.set_text('Cost = %.2f' % (oldc[i]))

    return line1, point1, value_display1,line, annotation



ax1.legend(loc = 1)

anim1 = animation.FuncAnimation(fig, animate_1, init_func=ini
t_1,
                                frames=len(oldc), interval=30,
                                  blit=True)
plt.close(fig)
```

```
theta_0= 3.798250105062286   theta_1= 2.044623271812862
```

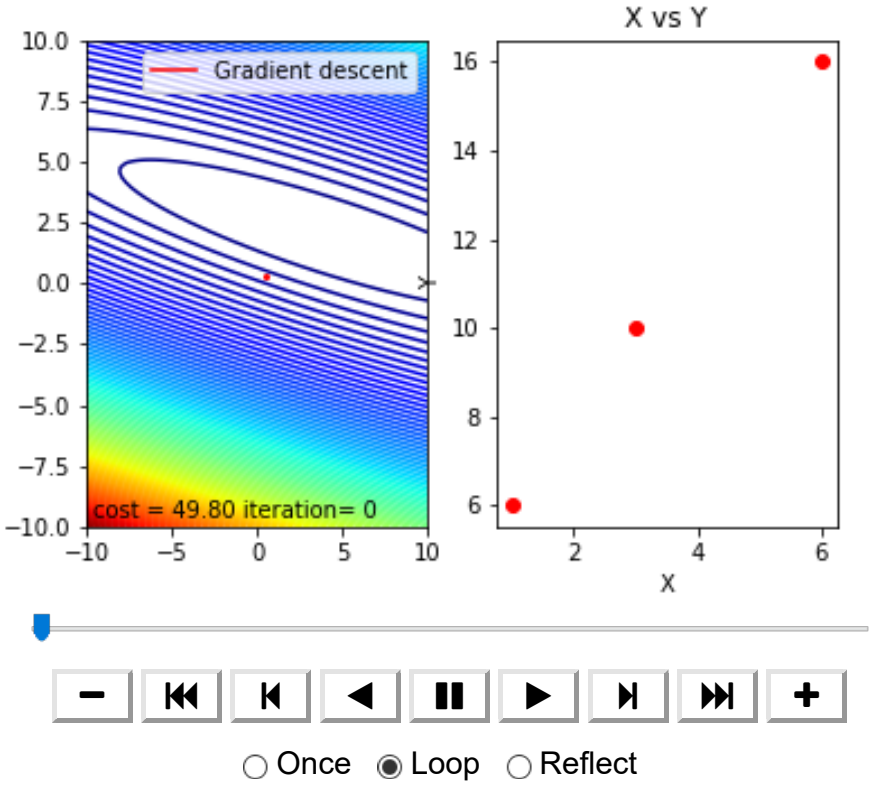
In [24]:
```
fig1 = plt.figure(figsize = (4,7))
ax2 = plt.axes()
ax2.set_title("Contour plot: point marked with red colour is
the theta value obtained by gradient descent")
manual_locations = [(0, 0), (2.5, 2.5),
                    (-2.5, -2.5), (5,5),
                    (-5, -5), (7.5, 7.5),(4,2)]
c=ax2.contour(x, y, zs, 70, cmap = 'jet')
plt.clabel(c, inline=True, fontsize=10, manual=manual_locatio
ns)
ax2.quiver(oldtheta[-1][0], oldtheta[-1][1],  scale_units = '
xy', angles = 'xy', scale = 1, color = 'black', alpha = .5)
plt.plot(oldtheta[-1][0], oldtheta[-1][1], 'r*')
```

Out[24]:
```
[<matplotlib.lines.Line2D at 0x7fa2c8602be0>]
```



In [25]:
```
anim1
```

Out[25]:



Observation:

This problem is theoratically pretty straight forward. The data set has only 3 values & input(x) has only one value. So we will ocnside rtwo parameters theta0 & theta1.Considered alpha = 0.1(alpha2 in code). I have initialized theta vector with two random values & for 105 iterations I have kept note of each theta & corresponding cost. Each iteration is lead by the updated value of theta & corresponding cost is plotted in contour plot. It is seen that both values of theta0 & theta1 are increased from initialized value & reach the optimal value where cost is 0.0047. Here plotted value is sum of residual errors(i.e. cost).By arrow the final destination is been pointed out. In the animation part, we can actually see that how the value of theta is changing with each iteration & also the value of cost is decreasing. It may seem that cost is fluctuating in contour plot but in each successive iteration it is coming inwards (moving to lesser value) only. At last after 105 iterations, the cost is nearly equals to zero & from left subplot we can see that prediction almost excatly fits the given data.

Question 3

In [39]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import scipy.linalg as la
import math
from scipy.fftpack import fft,fftfreq
from scipy.linalg import toeplitz
from matplotlib import animation

def cost_1(thetain,X,Y):
    m=len(Y)
    y_hat=np.dot(X,thetain)
    error=y_hat-Y
    cost=np.dot(error.T,error)
    return cost

def cost_derivative(thetain,X,Y,alpha):
    number_of_rows = X.shape[0]
    random_indices = np.random.choice(number_of_rows, size=2,
replace=False)
    X_i = X[random_indices, :]
    y_i = Y[random_indices]
    prediction = np.dot(X_i,thetain)
    thetaout = thetain -(2)*alpha*( X_i.T.dot((prediction - y
_i)))
    error=prediction-y_i
    cost=error**2
    return thetaout,cost
df = pd.read_csv('assign2.csv')
X=df.iloc[:,0:1]
Y=df.iloc[:,1:2]
X1=X.copy()
meanx=X1.mean()
stdx=X1.std()
X1=np.array(X1)
X1=X1.reshape(len(X1),)
X = (X - X.mean()) / X.std()
X=np.c_[ np.ones(X.shape[0]),X ]
alpha=0.01
iterations=200
Y=np.array(Y)
Y=Y.reshape(len(Y),)
theta=np.random.rand(2)
old=theta.copy()
oldc=[]
for i in range(iterations):
    theta,cost11=cost_derivative(old,X,Y,alpha)
    oldc.append(cost_1(theta,X,Y))
    old=theta.copy()
```

```
theta[0]=theta[0]-theta[1]*meanx/stdx
theta[1]=theta[1]/stdx
print("minimum obtained value for the given function is %.2
f"%np.min(oldc))
```

```
minimum obtained value for the given function is 1573.33
```

Observation:

This question has same approach as Q1.(e) but this time we have to apply stochastic GD.Here difference is that for SGD we used 2 random samples instead of whole batch for each iterations.It is indeed a fast algorithm as it gives values near to original observations by taking into account few observations compared to whole dataset.

Question 4

In [55]:
```
#Question 4-a
def derivative(theta):
    return theta
def cost(theta):
    return theta**2


theta = 5
old=theta
counter=0
for i in range(100):
    theta = theta - (derivative(theta))
    if abs(old-theta)==0:# if previous theta and current thet
a are same then break
        break
    old=theta
    counter=counter+1
print('theta value after iterations='+str(counter)+' for whic
h we will obtain minimum of the loss function : theta= '+str
(theta))
```

```
theta value after iterations=1 for which we will obtain min
imum of the loss function : theta= 0
```

In [56]:
```
#Question 4-b
def derivative(theta):
    return theta
def cost(theta):
    return theta[0]**2+theta[1]**2


theta = [5,7]

old=theta.copy()
counter=0
for i in range(100):
    theta[0] = old[0] - derivative(old[0])
    theta[1] = old[1] - derivative(old[1])

    if abs(old[0]-theta[0])==0 and abs(old[1]-theta[1])==0:
        break
    old=theta.copy()
    counter=counter+1
print("theta values after iterations="+str(counter)+"  for wh
ich we will obtain minimum of the loss function :theta_0=" +s
tr(theta[0])+"  theta_1="+str(theta[1]))
```

```
theta values after iterations=1  for which we will obtain m
inimum of the loss function :theta_0=0  theta_1=0
```

In [57]:
```
#Question 4-c
def derivative(theta):
    return theta-1
def cost(theta):
    return (theta-1)**2


theta = 5
old=theta
counter=0
for i in range(100):
    theta = theta - (derivative(theta))
    if abs(old-theta)==0:# if previous theta and current thet
a are same then break
        break
    old=theta
    counter=counter+1
print('theta value after iterations='+str(counter)+' for whic
h we will obtain minimum of the loss function : theta= '+str
(theta))
```

```
theta value after iterations=1 for which we will obtain min
imum of the loss function : theta= 1
```

In [58]:
```
#Question 4-d
def derivative(theta):
    return theta-1
def cost(theta):
    return ((theta[0]-1)**2)+((theta[1]-1)**2)

theta = [5,7]

old=theta.copy()
counter=0
for i in range(100):
    theta[0] = old[0] - derivative(old[0])
    theta[1] = old[1] - derivative(old[1])

    if abs(old[0]-theta[0])==0 and abs(old[1]-theta[1])==0:
        break
    old=theta.copy()
    counter=counter+1
print("theta values after iterations="+str(counter)+"  for wh
ich we will obtain minimum of the loss function :theta_0=" +s
tr(theta[0])+"  theta_1="+str(theta[1]))
```

```
theta values after iterations=1  for which we will obtain m
inimum of the loss function :theta_0=1  theta_1=1
```

In [69]:
```
#Question 4-e


def cost_derivative(thetain,X,Y,H):
    y_hat=np.dot(X,thetain)
    error=y_hat-Y
    cost=np.dot(error.T,error)
    thetaout=thetain-np.dot(H,2*np.dot(X.T,error))
    return thetaout,cost
df = pd.read_csv('assign2.csv')
X=df.iloc[:,0:1]
Y=df.iloc[:,1:2]
X1=X.copy()
meanx=X1.mean()
stdx=X1.std()
X1=np.array(X1)
X1=X1.reshape(len(X1),)
X = (X - X.mean()) / X.std()
X=np.c_[ np.ones(X.shape[0]),X ]
alpha=0.1
Y=np.array(Y)
Y=Y.reshape(len(Y),)
theta=np.random.rand(2)
old=theta.copy()
oldc=0
n=len(theta)
z=np.zeros((n,n))
for i in range(n):
  for j in range(n):
    z[i][j]=np.dot(X[:,i].T,X[:,j])
z=np.linalg.inv(2*z)
counter=0
for i in range(100):
    theta,cost=cost_derivative(old,X,Y,z)
    counter=counter+1
    if oldc<cost:
      break
    if i!=0 and oldc-cost==0:
      break
    oldc=cost
    old=theta.copy()

theta[0]=theta[0]-theta[1]*meanx/stdx
theta[1]=theta[1]/stdx
print("theta values after iterations="+str(counter)+"  for wh
ich we will obtain minimum of the loss function :theta_0=" +s
tr(theta[0])+"  theta_1="+str(theta[1]))
```

```
theta values after iterations=1  for which we will obtain m
inimum of the loss function :theta_0=49.23762989433488  the
ta_1=-0.008611934783475305
```

Observation:

a)Here loss function has only one parameter theta,so its Hessian matrix will be of dimensions 1x1.Its derivative and double derivative is calculated and by newtons method, we get L'(theta)/L" (theta)=theta.In this case, after one iteration the theta will become zero and after further iterations theta=0 will remain constant.Theta=0 will be the theta value for which loss function will be minimum.

b)Here loss function has two parameters theta0 and theta1,let theta=[theta0 theta1],so its Hessian matrix will be of dimensions 2x2.Its derivative and double derivative is calculated and by newtons method, we get H^-1*L'(theta)=theta.In this case, after one iteration the theta will become zero vector and after further iterations theta=0 will remain constant.Theta=[0 0] will be the theta value for which loss function will be minimum.

c)Here loss function has only one parameter theta,so its Hessian matrix will be of dimensions 1x1.Its derivative and double derivative is calculated and by newtons method, we get f'(x)/f" (x)=theta-1.In this case, after one iteration the theta will become one and after further iterations theta=1 will remain constant.Theta=1 will be the theta value for which loss function will be minimum.

d)Here loss function has two parameters theta0 and theta1,let theta=[theta0 theta1],so its Hessian matrix will be of dimensions 2x2.Its derivative and double derivative is calculated and by newtons method, we get H^-1*L'(theta)=theta.In this case, after one iteration the theta will become [1 1] vector and after further iterations theta=[1 1] will remain constant.Theta=[1 1] will be the theta value for which loss function will be minimum.

e)Here loss function has two parameters theta0 and theta1,let theta=[theta0 theta1],so its Hessian matrix will be of dimensions 2x2.Its derivative and double derivative is calculated , we get Hij=2(X[:,i]^T*X[:,j]) where X[:,i] is the column vector ,here H will become constant for every theta.Here we get the exact theta for which the loss function will be minimum just after single iteration because here after differentiation of order more than 2 the function will become 0 so, here newtons method will directly gives us the theta same as the theta obtained by pseudo inverse method.If the function has nonzero derivative for orders more than 2 then it will take more than 1 iteration but here the order of theta 0 and theta1 was only 2.