

**Linear least Squares Method:**

Implementation of the linear least squares method: The weights that minimize the given cost function is computed and a linear fit to the given points is plotted.

import numpy as np

import random as rand

import matplotlib.pyplot as plt

x=np.arange(1,51).reshape(1,50) #Setting x from 1 to 50

one=np.ones([1,50])

X=np.array([x,one]).reshape(2,50)

y=x+np.random.uniform(-1,1,size=[1,50]) #Setting y to be x+arbitrary real number

P=np.matmul(np.transpose(X),np.linalg.inv(np.matmul(X,np.transpose(X)))) #Calculating pseudo inverse

w=np.matmul(y,P) #Obtaining the weights

plt.scatter(x,y)

z=(w[0][0]\*x)+w[0][1]

plt.plot(x,z,'r.')

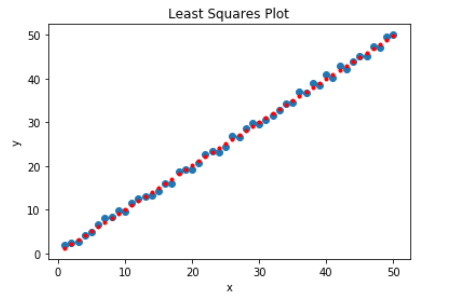
plt.xlabel('x')

plt.ylabel('y')

plt.title('Linear Least Squares Plot')

plt.show()

Output:



# Gradient Descent Method:

Implementing the Gradient Descent algorithm to observe the minimizer for the given cost function.

w=np.array([[0],[1]]) #Initial points choosen

E=[ ]

Epoch=[ ]

epoch=0

learning\_rate=0.005 #Setting learning rate

threshold=0.08 #Setting threshold value

i=0

cd=np.zeros([1,2])

cd=ab[:,i].reshape(1,2)

while(np.absolute((y[0,i]-np.matmul(cd,m)))<threshold): #Condition to check if absolute value of error is below a particular threshold

g=(np.matmul(cd,m)-y[0,i])\*m #Calculating gradient

w=w-(learning\_rate\*g) #updating weights

i=i+1

plt.scatter(x,y)

z=(w[1]\*x)+w[0]

plt.plot(x,z,'r.')

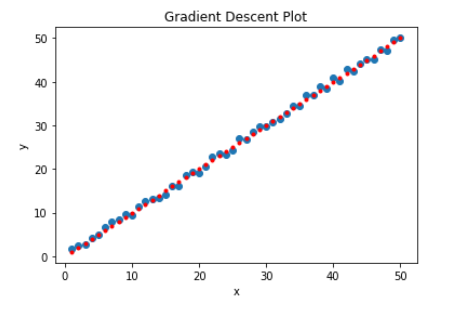
plt.xlabel('x')

plt.ylabel('y')

plt.title('Gradient Descent Plot')

plt.show()

Output:



The weights obtained using the gradient descent algorithm inclines closer to the optimal weights computed in (c) than the one computed using the Linear Least squares method.