Homework 1

Question 1 Answer:

$$\frac{\partial(MSE(\theta_j))}{\partial\theta_j} = \frac{1}{m}\sum_{i=0}^{m} 2(\theta^T \cdot x^i - y^i) \cdot x^i$$

$$x^i = 1$$

$$\frac{\partial(MSE(\theta_j))}{\partial\theta_j} = \frac{1}{m}\sum_{i=0}^{m} 2(\theta^T \cdot x^i - y^i)$$

Question 2 Answer:

$$\frac{\partial(MSE(\theta_0))}{\partial\theta_0} = \frac{1}{m}\sum_{i=0}^{m} 2(\theta^T \cdot x^i - y^i) \cdot x^i$$

$$\frac{\partial(MSE(\theta_1))}{\partial\theta_1} = \frac{1}{m}\sum_{i=0}^{m} 2(\theta^T \cdot x^i - y^i) \cdot x^i$$

$$\frac{\partial(MSE(\theta_n))}{\partial\theta_n} = \frac{1}{m}\sum_{i=0}^{m} 2(\theta^T \cdot x^i - y^i) \cdot x^i$$

$$\frac{\partial(MSE(\theta_n))}{\partial\theta_n} = \frac{2}{m}X^T(\theta^T \cdot X - y)$$

.

.

$$\nabla_\theta MSE(\theta) \begin{bmatrix} \frac{\partial}{\partial\theta_0}MSE(\theta_0) \\ \\ \frac{\partial}{\partial\theta_0}MSE(\theta_0) \\ \\ \vdots \\ \frac{\partial}{\partial\theta_0}MSE(\theta_0) \end{bmatrix} = \frac{2}{m}X^T(\theta^T \cdot X - y)$$

Implementation part (Explained):

Libraries

At this part we load the python needed libraries,
You can find and understand each library functionality in google yet here is a breif explination

Numpy: supporting multi-dimensional arrays and matrices and mathematical functions on those arrays.

Pandas: data manipulation and analysis library

matplotlib:plotting library.

sklearn:machine learning library.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
```

Data and describtion

Load the needed data.

```python
from sklearn.datasets import load_boston
boston = load_boston()
```

check its describtion to know and understand how to use it.

```python
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usu

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie Mellon Universit

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regressio
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the
```

Show and analyse the data

At this part we used the panda library to get the data in table and show it based on the features.

```
data = pd.DataFrame(boston.data, columns=boston["feature_names"])
print(data.head())
```

```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT
0     15.3  396.90   4.98
1     17.8  396.90   9.14
2     17.8  392.83   4.03
3     18.7  394.63   2.94
4     18.7  396.90   5.33
```

Split the data to train and test.

At this part we used the sklearn library to split and the data to train,test data sets and check them

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test =  train_test_split(data,  boston.target , test_size = 0.33, ra
```

Show The splitted data

```
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(339, 13)
(167, 13)
(339,)
(167,)
```

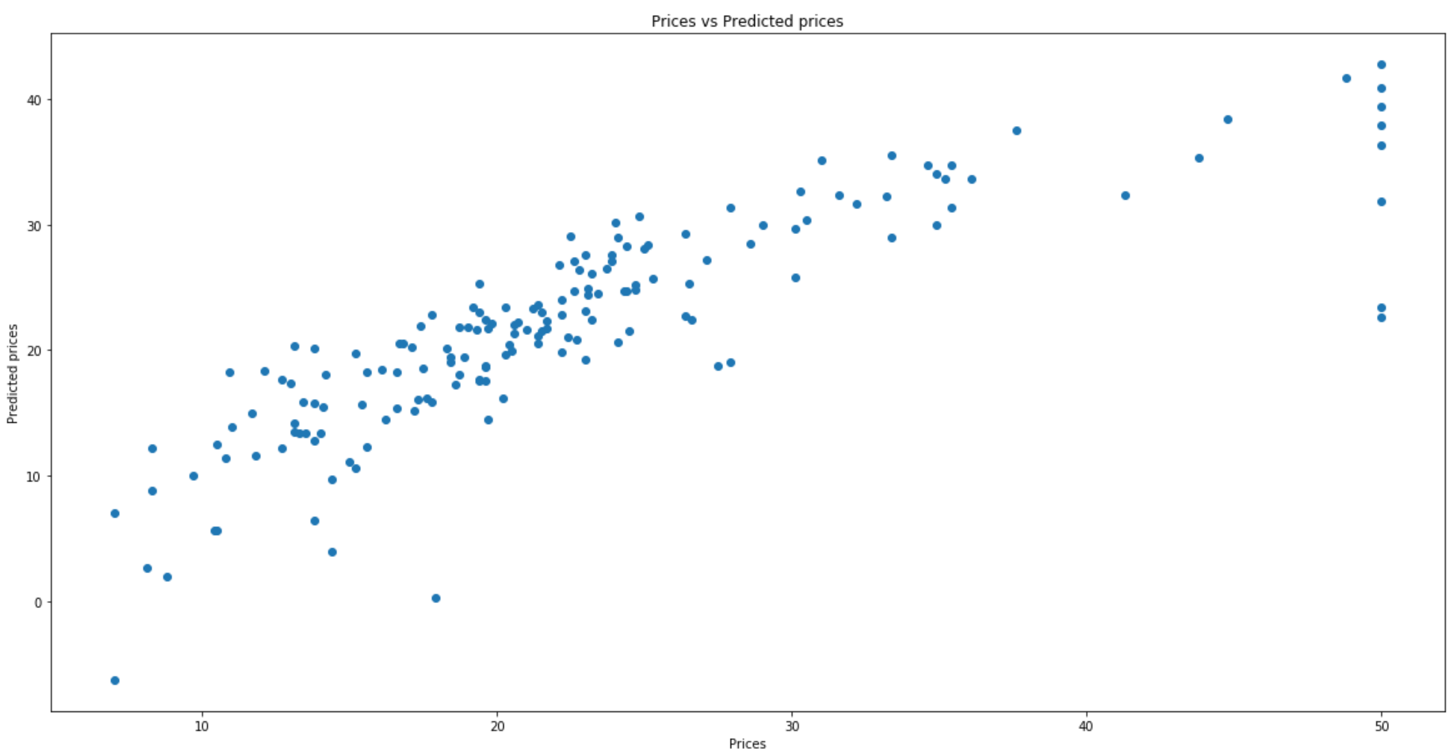
Use linearRegression method.

We used the Sklear leinear regression mathod with the normalize turned on, so it will do the normlisation automatically without an effort from our sides.

```
from sklearn.linear_model import LinearRegression

lrm = LinearRegression(normalize=True)
lrm.fit(X_train, Y_train)

Y_pred = lrm.predict(X_test)

plt.figure(figsize=(20,10))
plt.scatter(Y_test, Y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
```

```
Text(0.5, 1.0, 'Prices vs Predicted prices')
```

The results is printed here and showed as comparison between the predicted and trained



Show the trained and test data.

Just to check the reality of our linear regression.

```
df = pd.DataFrame({'Actual': Y_test, 'Predicted': Y_pred})
print(df)
print('Mean Squared Error:', sklearn.metrics.mean_squared_error(Y_test, Y_pred))
```

```
     Actual  Predicted
0      37.6  37.467236
1      27.9  31.391547
2      22.6  27.120196
3      13.8   6.468433
4      35.2  33.629667
..      ...        ...
162    14.4   9.718369
163    35.4  34.705200
164    25.3  25.704102
165    18.3  20.154309
166    16.6  15.394658

[167 rows x 2 columns]
Mean Squared Error: 28.530458765974647
```

The mean Squared error.

```
mse = sklearn.metrics.mean_squared_error(Y_test, Y_pred)
print(mse)
```

```
28.530458765974647
```

The Gradient Descent Mean Squared error Algorithm.

Function parameters:
test: tested data
pred: Predicted Data.
learningRate: learning rate
iteration: number of iterations (default= 100)
theta: our theta (default to 100)

The function start with initlising the history array, needed for the plots, which it takes the number of our iterations.

First theta, has been initlised without any computing, then we had iterated it over and calculated it ever each iteration based on

$\theta_n = \theta - \eta \nabla_\theta MSE(\theta)$

η = Learning Rate

MSE(θ) = Mean squared error

The first theta is random then we caluclate it by substracting the previuos one from our MSE multiplied by the learning rate to get the results.

then the function plot/draw the figure that we expexted with the results.

```
def GDMSE(test,pred,learningRate,iteration=100,theta=100):
    history1 = [x for x in range(iteration)]
    history1[0]=theta;
    for i in range(100):
        theta = theta - learningRate * sklearn.metrics.mean_squared_error(test, pred)
        history1[i]=theta;
        print(i,theta)
#     plt.set_xlim()

    plt.figure(figsize=(20,10))
    plt.scatter(history1, [x for x in range(iteration)])
#     ax = plt.gca()
#     ax.set_ylim([0,100])
#     ax.set_xlim([0,100])
# #     for i in range(iteration):
#         x = history1[i]
#         y = i
#         plt.text(x+0.3, y+0.3, i, fontsize=9)
#     plt.show()
```

Different Learning rates

Learning rate = 0.5

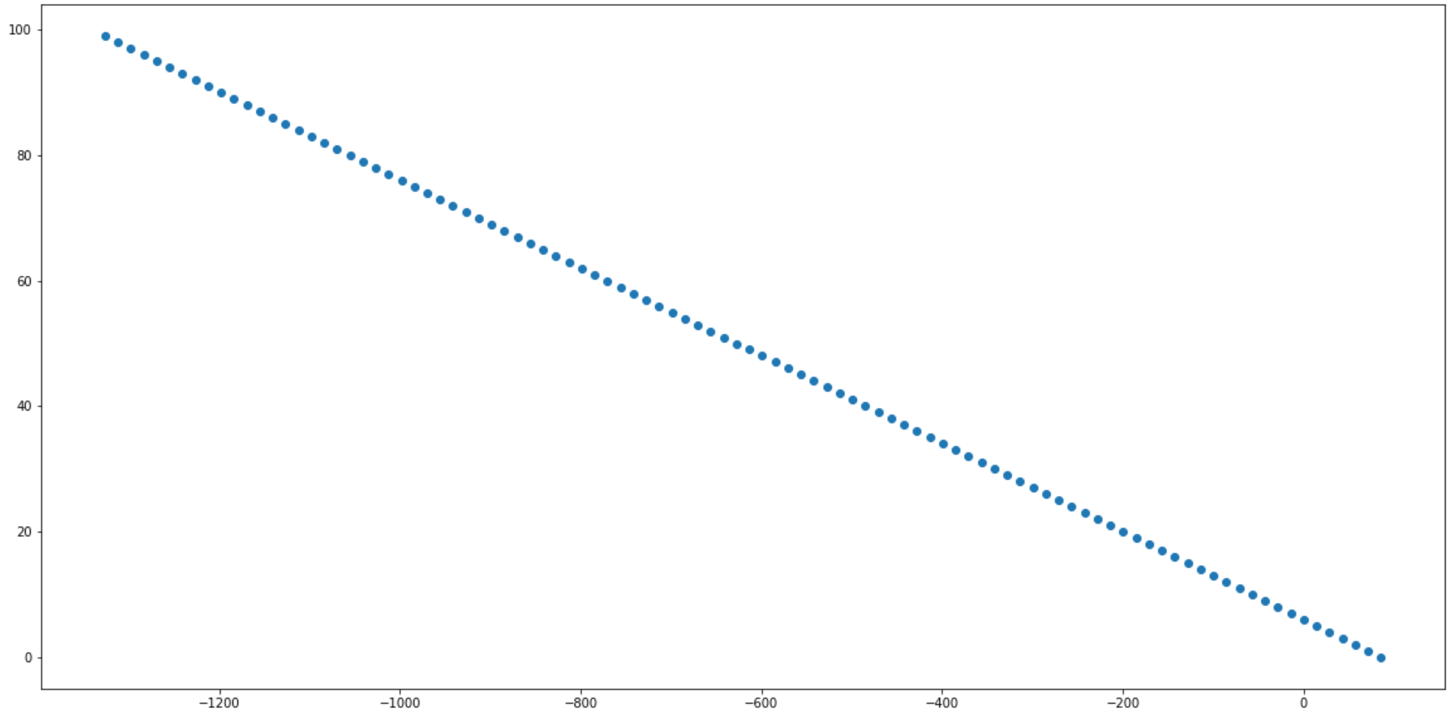
```
GDMSE(Y_test,Y_pred,0.5)
```

Results

0 85.73477061701267
1 71.46954123402534
2 57.20431185103802
3 42.9390824680507
4 28.673853085063378
5 14.408623702076055
6 0.14339431908873124
7 -14.121835063898592
8 -28.387064446885915
9 -42.65229382987324
10 -56.91752321286056
11 -71.18275259584789
12 -85.44798197883522
13 -99.71321136182254
14 -113.97844074480987
15 -128.2436701277972
16 -142.50889951078452
17 -156.77412889377183
18 -171.03935827675915
19 -185.30458765974646
20 -199.56981704273377
21 -213.8350464257211
22 -228.1002758087084
23 -242.36550519169572
24 -256.63073457468306
25 -270.8959639576704
26 -285.16119334065775
27 -299.4264227236451
28 -313.69165210663243
29 -327.9568814896198
30 -342.2221108726071
31 -356.48734025559446
32 -370.7525696385818
33 -385.01779902156915
34 -399.2830284045565
35 -413.54825778754383
36 -427.8134871705312
37 -442.0787165535185
38 -456.34394593650586
39 -470.6091753194932
40 -484.87440470248055
41 -499.1396340854679
42 -513.4048634684552
43 -527.6700928514425
44 -541.9353222344298
45 -556.2005516174171
46 -570.4657810004044
47 -584.7310103833917
48 -598.996239766379
49 -613.2614691493662
50 -627.5266985323535

```
51 -641.7919279153408
52 -656.0571572983281
53 -670.3223866813154
54 -684.5876160643027
55 -698.85284544729
56 -713.1180748302772
57 -727.3833042132645
58 -741.6485335962518
59 -755.9137629792391
60 -770.1789923622264
61 -784.4442217452137
62 -798.709451128201
63 -812.9746805111882
64 -827.2399098941755
65 -841.5051392771628
66 -855.7703686601501
67 -870.0355980431374
68 -884.3008274261247
69 -898.566056809112
70 -912.8312861920992
71 -927.0965155750865
72 -941.3617449580738
73 -955.6269743410611
74 -969.8922037240484
75 -984.1574331070357
76 -998.422662490023
77 -1012.6878918730102
78 -1026.9531212559975
79 -1041.218350638985
80 -1055.4835800219723
81 -1069.7488094049597
82 -1084.0140387879471
83 -1098.2792681709345
84 -1112.544497553922
85 -1126.8097269369093
86 -1141.0749563198967
87 -1155.3401857028841
88 -1169.6054150858715
89 -1183.870644468859
90 -1198.1358738518463
91 -1212.4011032348337
92 -1226.6663326178211
93 -1240.9315620008085
94 -1255.196791383796
95 -1269.4620207667833
96 -1283.7272501497707
97 -1297.9924795327581
98 -1312.2577089157455
99 -1326.522938298733
```

Plot



discussion

Getting to the point (0,0) took the algorithm about 6 steps then it get over for the minus, we could stop the algorthm there since its accepted result.

Learning rate = 0.1

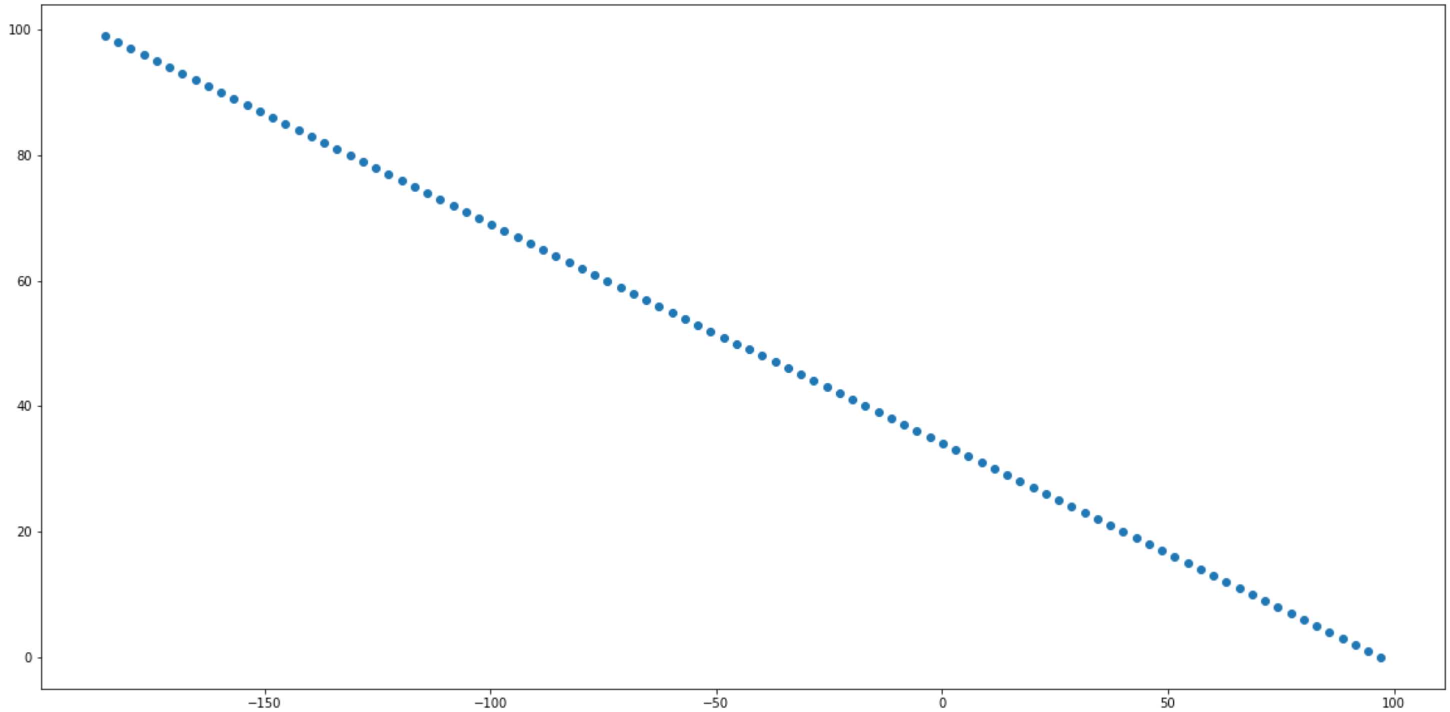
```
GDMSE(Y_test,Y_pred,0.1)
```

Results

```
0 97.14695412340254
1 94.29390824680507
2 91.44086237020761
3 88.58781649361015
4 85.73477061701269
5 82.88172474041522
6 80.02867886381776
7 77.1756329872203
8 74.32258711062283
9 71.46954123402537
10 68.61649535742791
11 65.76344948083045
12 62.91040360423298
13 60.05735772763552
14 57.20431185103806
15 54.351265974440594
16 51.49822009784313
17 48.64517422124567
18 45.792128344648205
19 42.93908246805074
20 40.08603659145328
21 37.232990714855816
22 34.37994483825835
23 31.52689896166089
24 28.673853085063428
25 25.820807208465965
26 22.967761331868502
27 20.11471545527104
28 17.261669578673576
29 14.408623702076111
30 11.555577825478647
31 8.702531948881182
32 5.849486072283717
33 2.9964401956862527
34 0.14339431908878808
35 -2.7096515575086766
36 -5.562697434106141
37 -8.415743310703606
38 -11.26878918730107
39 -14.121835063898535
40 -16.974880940496
41 -19.827926817093463
42 -22.680972693690926
43 -25.53401857028839
44 -28.38706444688585
45 -31.240110323483314
46 -34.09315620008078
47 -36.94620207667824
48 -39.7992479532757
49 -42.652293829873166
50 -45.50533970647063
```

```
51 -48.35838558306809
52 -51.211431459665555
53 -54.06447733626302
54 -56.91752321286048
55 -59.77056908945794
56 -62.623614966055406
57 -65.47666084265288
58 -68.32970671925034
59 -71.1827525958478
60 -74.03579847244526
61 -76.88884434904273
62 -79.74189022564019
63 -82.59493610223765
64 -85.44798197883512
65 -88.30102785543258
66 -91.15407373203004
67 -94.0071196086275
68 -96.86016548522497
69 -99.71321136182243
70 -102.5662572384199
71 -105.41930311501736
72 -108.27234899161482
73 -111.12539486821228
74 -113.97844074480975
75 -116.83148662140721
76 -119.68453249800467
77 -122.53757837460213
78 -125.3906242511996
79 -128.24367012779706
80 -131.09671600439452
81 -133.94976188099199
82 -136.80280775758945
83 -139.6558536341869
84 -142.50889951078437
85 -145.36194538738184
86 -148.2149912639793
87 -151.06803714057676
88 -153.92108301717423
89 -156.7741288937717
90 -159.62717477036915
91 -162.48022064696661
92 -165.33326652356408
93 -168.18631240016154
94 -171.039358276759
95 -173.89240415335647
96 -176.74545002995393
97 -179.5984959065514
98 -182.45154178314885
99 -185.30458765974632
```

Plot



Discussion

Getting to the point (0,0) took the algorithm about 34 steps then it get over for the minus, we could stop the algorthm there since its accepted result.

here we can see since we have decreased the learning rate it took more time for the algorthm to get the point that we are comparing to it.

Learning rate = 0.02

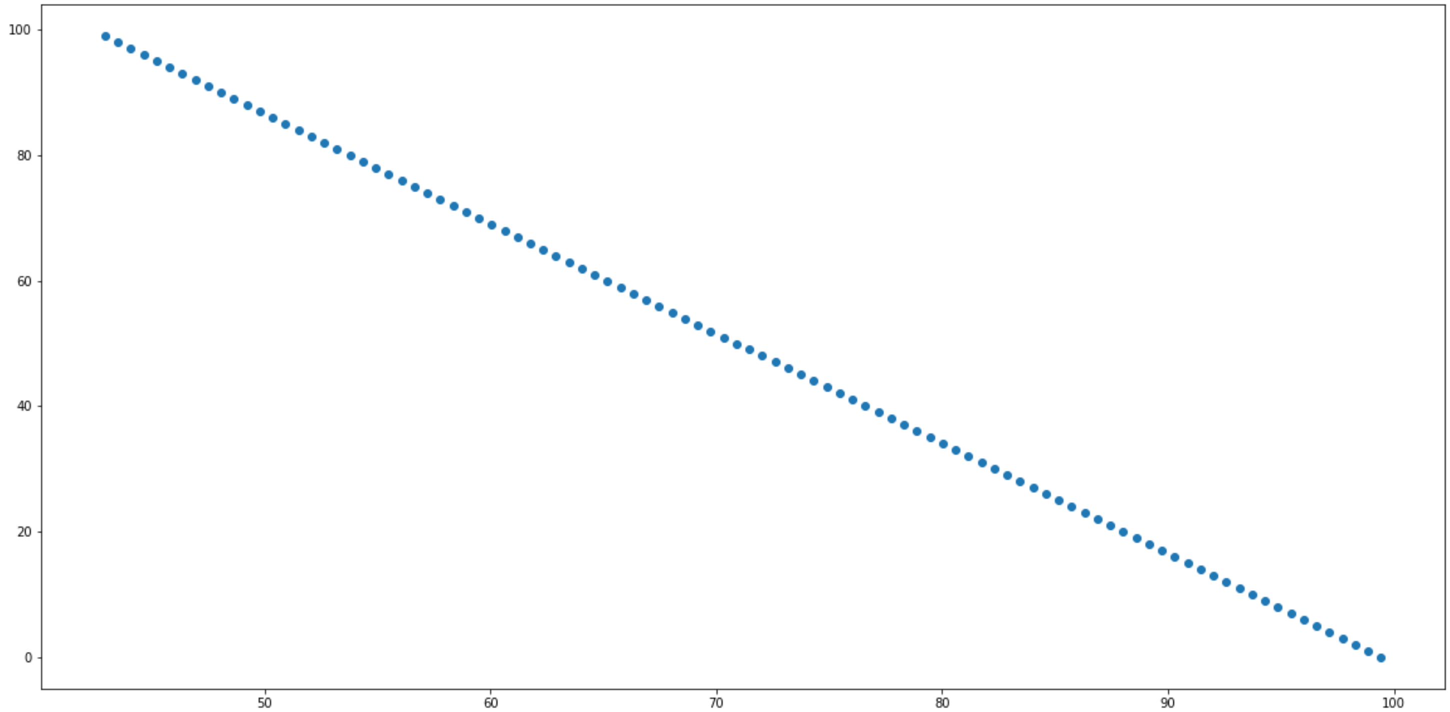
Results

```
GDMSE(Y_test,Y_pred,0.02)
```

0 99.42939082468051
1 98.85878164936102
2 98.28817247404153
3 97.71756329872204
4 97.14695412340255
5 96.57634494808306
6 96.00573577276357
7 95.43512659744408
8 94.86451742212459
9 94.2939082468051
10 93.72329907148561
11 93.15268989616612
12 92.58208072084663
13 92.01147154552714
14 91.44086237020765
15 90.87025319488816
16 90.29964401956867
17 89.72903484424918
18 89.1584256689297
19 88.5878164936102
20 88.01720731829072
21 87.44659814297123
22 86.87598896765174
23 86.30537979233225
24 85.73477061701276
25 85.16416144169327
26 84.59355226637378
27 84.02294309105429
28 83.4523339157348
29 82.88172474041531
30 82.31111556509582
31 81.74050638977633
32 81.16989721445684
33 80.59928803913735
34 80.02867886381786
35 79.45806968849837
36 78.88746051317888
37 78.31685133785939
38 77.7462421625399
39 77.17563298722041
40 76.60502381190092
41 76.03441463658143
42 75.46380546126194
43 74.89319628594245
44 74.32258711062296
45 73.75197793530347
46 73.18136875998398
47 72.61075958466449
48 72.040150409345
49 71.46954123402551
50 70.89893205870602

```
51 70.32832288338653
52 69.75771370806704
53 69.18710453274755
54 68.61649535742806
55 68.04588618210857
56 67.47527700678909
57 66.9046678314696
58 66.3340586561501
59 65.76344948083062
60 65.19284030551113
61 64.62223113019164
62 64.05162195487215
63 63.48101277955266
64 62.91040360423317
65 62.33979442891368
66 61.76918525359419
67 61.1985760782747
68 60.62796690295521
69 60.05735772763572
70 59.48674855231623
71 58.91613937699674
72 58.34553020167725
73 57.77492102635776
74 57.20431185103827
75 56.63370267571878
76 56.06309350039929
77 55.4924843250798
78 54.92187514976031
79 54.35126597444082
80 53.78065679912133
81 53.21004762380184
82 52.63943844848235
83 52.06882927316286
84 51.49822009784337
85 50.92761092252388
86 50.35700174720439
87 49.7863925718849
88 49.21578339656541
89 48.645174221245924
90 48.074565045926434
91 47.503955870606944
92 46.933346695287455
93 46.362737519967965
94 45.792128344648475
95 45.221519169328985
96 44.650909994009496
97 44.080300818690006
98 43.509691643370516
99 42.939082468051026
```

Plot



Discussion

As we see each step is taking about 0.4 point to move the algorthm which it will make the algorthm to take about 200 steps to reach the point (0,0).

here we can see that it will be too slow to get our point from this learning rate.

Results

For the learning rates and the iterations we have three possiblities which is:

Slow learning Rate: The algorithm will reach the desired results but it will take time for that since its gonna be too slow.

Optimal Learning Rate: The algorthm will reach the desired results in the lowest number of iterations.

High Learning Rate: the algorthm wont reach the desired results and will pass it, by jumbing over it and go to new results.

There is few ways to learn and know about the best learning rate, which it would be great to understand.

References:

andrew ng machine learning stanford

ethem alpaydın machine learning

https://scikit-learn.org/ docs

https://developers.google.com/machine-learning/crash-course/fitter/graph