# Week #4 Assignment – Satish Ramachandran

# Problem 1

'''

Week4 - Assignment problem #1

'''

import math

import matplotlib.pyplot as plt

x = 5

y = 5

learning\_rate = 0.01

epsilon = 0.000001

iteration = 0

'''

Function is:

f(x,y) = z = -1 \* math.sqrt(25 - (x - 2) \*\* 2 - (y - 3) \*\*2)

'''

def func\_z(x, y):

return (-1 \* math.sqrt(25 - (x - 2) \*\* 2 - (y - 3) \*\*2))

'''

Partial derivative w.r.t x

'''

def dz\_dx(x, y):

return ((2 \* (x - 2)) / (math.sqrt((25 - (x - 2) \*\* 2 - (y - 3) \*\* 2))))

'''

Partial derivative w.r.t y

'''

def dz\_dy(x, y):

return ((2 \* (y - 3)) / (math.sqrt((25 - (x - 2) \*\* 2 - (y - 3) \*\* 2))))

while True:

iteration = iteration + 1

plt.plot(x, y, 'o')

new\_x = x - learning\_rate \* dz\_dx(x,y)

new\_y = y - learning\_rate \* dz\_dy(x,y)

if (abs(x - new\_x) < epsilon) and (abs(y - new\_y) < epsilon):

print("Solution reached..")

x = new\_x

y = new\_y

plt.plot(x, y, 'o')

plt.waitforbuttonpress()

print('New x and y less than epsilon')

break

# More improvements could be made

x = new\_x

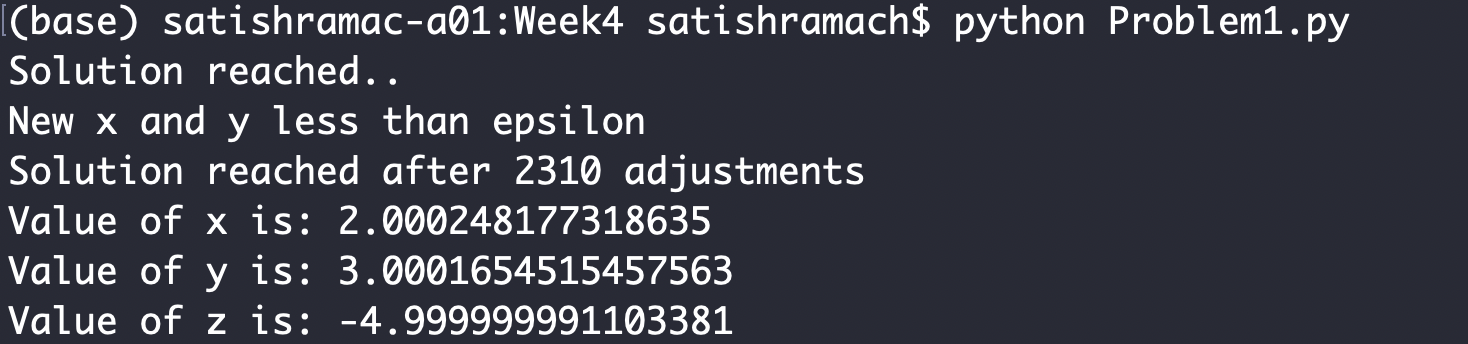
y = new\_y

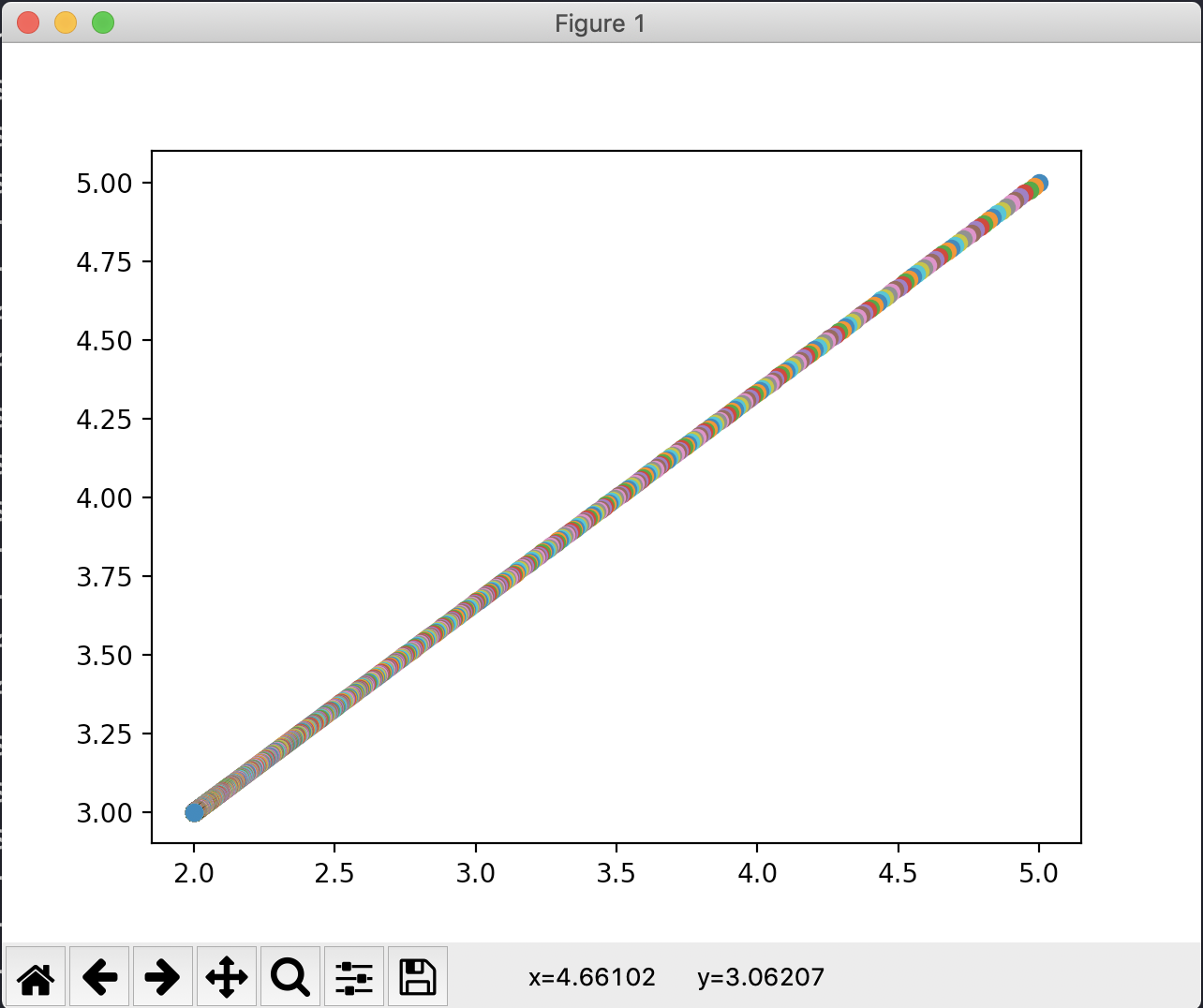
print('Solution reached after ' + str(iteration) + ' adjustments')

print('Value of x is: ' + str(x))

print('Value of y is: ' + str(y))

print('Value of z is: ' + str(func\_z(x, y)))





# Problem 2

'''

Week 4 - Problem 2

Solution using Scikit-learn and hand-coded Gradient descent method

'''

import numpy as np

import matplotlib.pyplot as plt

from sklearn import linear\_model

from sklearn import preprocessing

### Generate the data ###

def generate\_data(random\_seed, n\_samples):

train\_x = np.linspace(0,20,n\_samples)

train\_y = 3.7 \* train\_x + 14 + 4 \* np.random.randn(n\_samples)

print("X data")

print("------")

print("Size: " + str(np.shape(train\_x)))

print(train\_x)

print("Y data")

print("------")

print("Size: " + str(np.shape(train\_y)))

print(train\_y)

return(train\_x, train\_y)

### SciKit Learn method

def scikit\_method(x\_data, y\_data):

print("Using SciKit learn..")

linear\_reg = linear\_model.LinearRegression()

print("Dimensions: X: " + str(x\_data.ndim) + ", Y: " + str(y\_data.ndim))

# IMPORTANT: LinearRegression expects a 2-D array. So, add a dimension using

# reshape()

linear\_reg.fit(x\_data.reshape(-1,1), y\_data.reshape(-1,1))

print("Slope : " + str(linear\_reg.coef\_))

print("Intercept: " + str(linear\_reg.intercept\_))

return (linear\_reg.coef\_, linear\_reg.intercept\_)

### Handcoded Gradient descent method

'''

Partial differentiation w.r.t slope

'''

def dy\_dslope(slope, intercept, x\_data, y\_data):

return (-2 \* sum((y\_data - slope \* x\_data - intercept) \* x\_data))

'''

Partial differentiation w.r.t intercept

'''

def dy\_dintercept(slope, intercept, x\_data, y\_data):

return (-2 \* sum((y\_data - slope \* x\_data - intercept)))

'''

Gradient Descent method

'''

def gradient\_descent(x\_data, y\_data, learn\_rate, epochs):

print("Gradient Descent method..")

slope = 0

intercept = 0

for iteration in range(epochs):

#print(iteration)

slope = slope - learn\_rate \* dy\_dslope(slope, intercept, x\_data, y\_data)

intercept = intercept - learn\_rate \* dy\_dintercept(slope, intercept, x\_data, y\_data)

return (slope, intercept)

'''

Function to descale a min\_max scaled data

'''

def deScale\_y(y\_orig, scaled\_y):

result\_y = []

min\_y = min(y\_orig)

max\_y = max(y\_orig)

for each\_scaled\_y in scaled\_y:

result\_y.append(each\_scaled\_y \* (max\_y - min\_y) + min\_y)

return result\_y

train\_x\_actual, train\_y\_actual = generate\_data(42, 30)

train\_x = preprocessing.minmax\_scale(train\_x\_actual)

train\_y = preprocessing.minmax\_scale(train\_y\_actual)

s\_slope, s\_intercept = scikit\_method(train\_x, train\_y)

gd\_slope, gd\_intercept = gradient\_descent(train\_x, train\_y, 0.001, 4000)

print("RESULTS" + "\n" + "-------")

print("SciKit Learn : slope: " + str(s\_slope) + " intercept: " + str(s\_intercept))

sci\_slope = s\_slope[0][0]

sci\_intercept = s\_intercept[0]

result\_scaled\_y = train\_x \* sci\_slope + sci\_intercept

sci\_descaled\_computed\_y = deScale\_y(train\_y\_actual, result\_scaled\_y)

print("Gradient Descent : slope: " + str(gd\_slope) + " intercept: " + str(gd\_intercept))

result\_scaled\_y = train\_x \* gd\_slope + gd\_intercept

gd\_descaled\_computed\_y = deScale\_y(train\_y\_actual, result\_scaled\_y)

# Plot the results

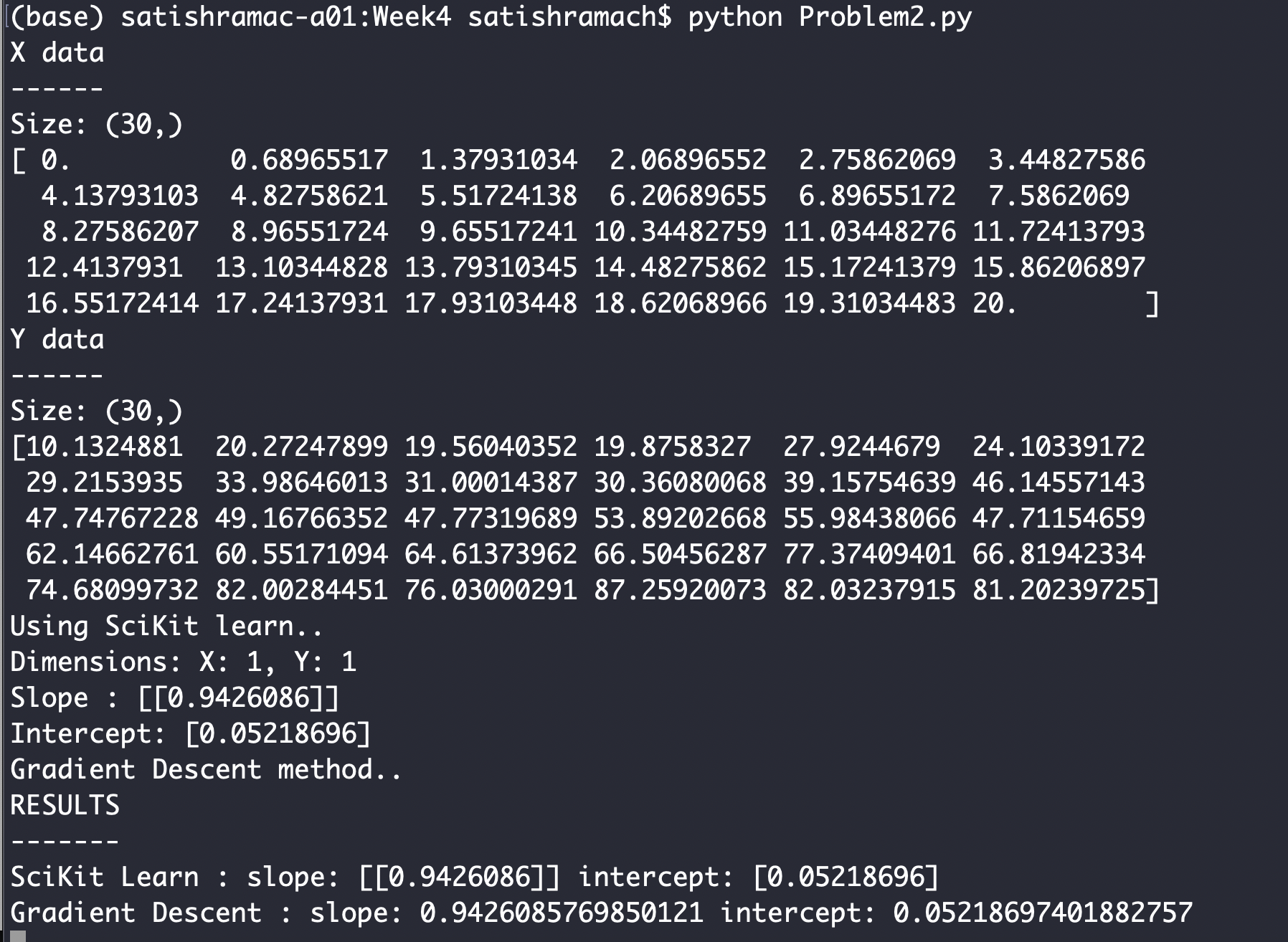
plt.plot(train\_x\_actual, train\_y\_actual, 'o')

plt.plot(train\_x\_actual, sci\_descaled\_computed\_y, '-o')

plt.plot(train\_x\_actual, gd\_descaled\_computed\_y, '-o')

plt.show()

plt.waitforbuttonpress()



As can be seen, the SciKit derived values and Gradient Descent values are almost the same.

This is the plot of the deScaled values of Y based on the computed slope and intercept. Since both the values are extremely close, the plot line seems like just one.

