Week#5 – Satish Ramachandran

Problem#1

NOTE: I'm fixing the derivative to remove the 2 from the numerator. Therefore, the new derivative functions that I'm using are:

$$\frac{\partial z}{\partial x} = \frac{(x-2)}{\sqrt{25-(x-2)^2-(y-3)^2}}$$

Similarly:

$$\frac{\partial z}{\partial y} = \frac{(y-3)}{\sqrt{25-(x-2)^2-(y-3)^2}}$$

```
'''
Week5 - Assignment problem #1
Solving Week4's problem using Gradient Descent with Momentum
'''

import math
import matplotlib.pyplot as plt

learning_rate = 0.01
epsilon = 0.000001

'''
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
NOTE: Fixing the derivative from Homework4
'''
def dz_dx(x, y):
   return ((x - 2) / (math.sqrt((25 - (x - 2) ** 2 - (y - 3) ** 2))))

'''
Partial derivative w.r.t y
NOTE: Fixing the derivative from Homework4
'''
def dz_dy(x, y):
```

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

'''
Plain Gradient Descent
'''
def plain_gradient_descent(x, y):
    iteration = 0
    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()
            plt.close()
            print('New x and y less than epsilon')
            return (x, y, iteration)
        # More improvements could be made
        x = new_x
        y = new_y

'''
Gradient Descent with momentum
'''
def gradient_descent_momentum(x, y):
    update_x = 0
    update_y = 0
    gamma = 0.9
    iteration = 0
    while True:
        iteration = iteration + 1
        plt.plot(x, y, 'o')
        update_x = (gamma * update_x) + (learning_rate * dz_dx(x,y))
        update_y = (gamma * update_y) + (learning_rate * dz_dy(x,y))

        new_x = x - update_x
        new_y = y - update_y
        if (abs(x - new_x) < epsilon) and (abs(y - new_y) < epsilon):
            print("Solution reached..")
            x = new_x
```

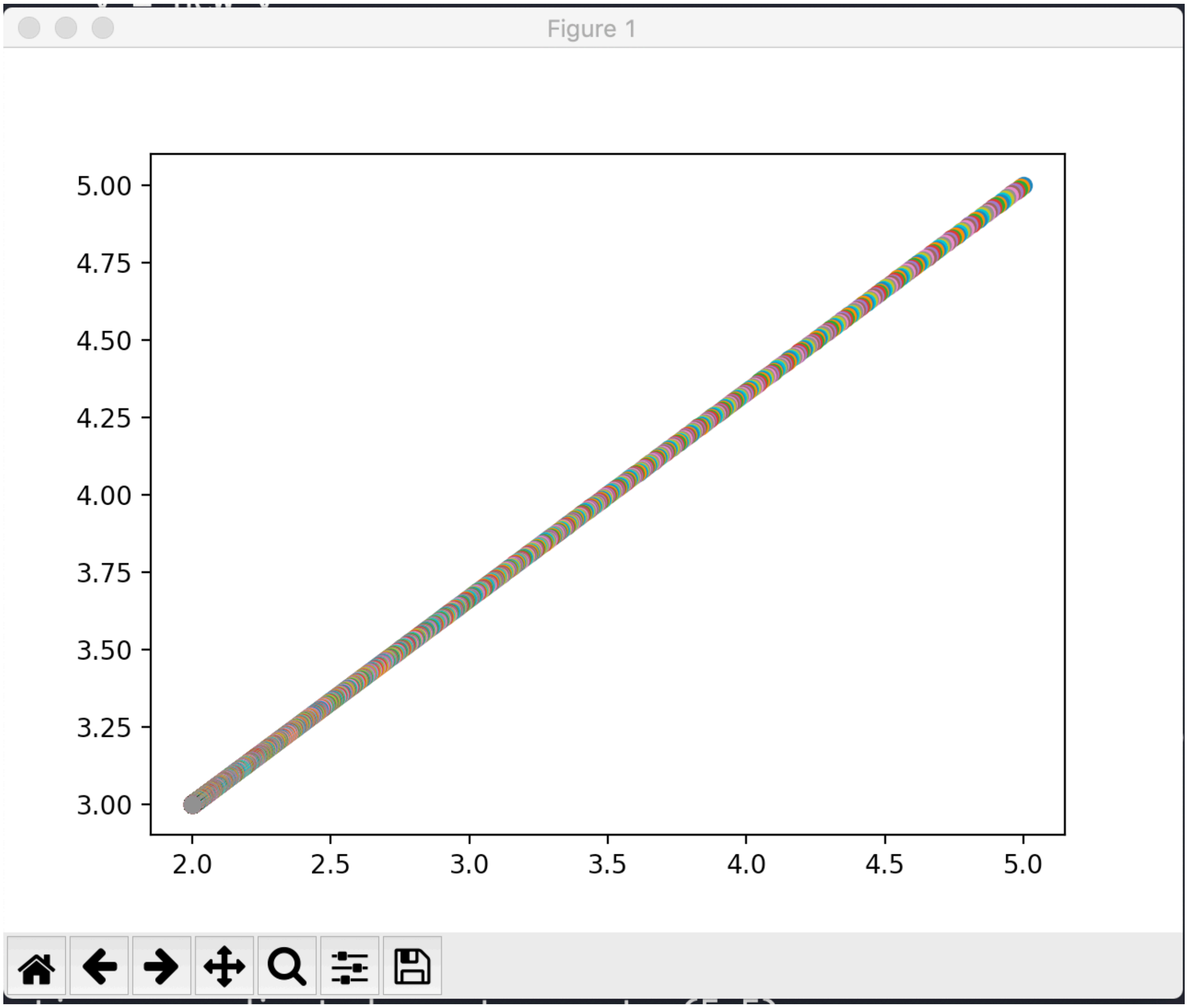
```
            y = new_y
            plt.plot(x, y, 'o')
            plt.waitforbuttonpress()
            plt.close()
            print('New x and y less than epsilon')
            return (x, y, iteration)
        # More improvements could be made
        x = new_x
        y = new_y



#First, try using Gradient Descent
x,y,iteration = plain_gradient_descent(5,5)

print('Plain Gradient Descent')
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)))

#Try the GD, with momentum
x,y,iteration = gradient_descent_momentum(5,5)
print('Gradient Descent with Momentum')
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)))
```

```
(base) satishramac-a01:Week5 satishramach$ python Problem1.py
Solution reached..
New x and y less than epsilon
Plain Gradient Descent
Solution reached after 4277 adjustments
Value of x is: 2.000498009869403
Value of y is: 3.0003320065796073
Value of z is: -4.99999996417578
Solution reached..
New x and y less than epsilon
Gradient Descent with Momentum
Solution reached after 425 adjustments
Value of x is: 2.0000363474991674
Value of y is: 3.000024231666111
Value of z is: -4.999999999809169
```

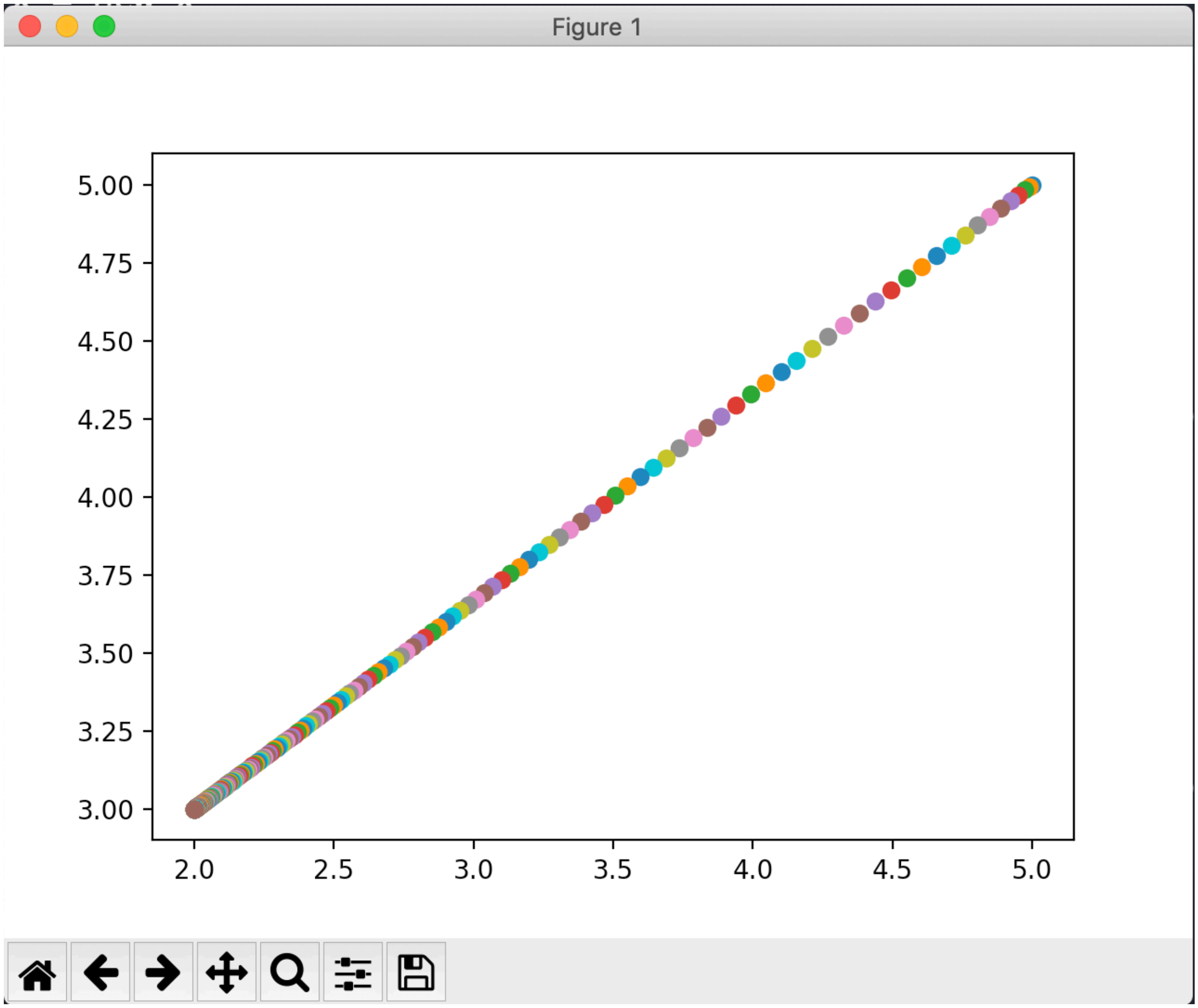
Problem#2

Clearly, GD with momentum converges faster than the plain Gradient Descent algorithm. The number of iterations is 4277 vs 425, which means, it took the momentum based algorithm only 10% of the number of iterations.
The plot of the solutions also shows the same.

Without momentum:



With momentum:



It can be seen that the increments became smaller as the solution was converging.