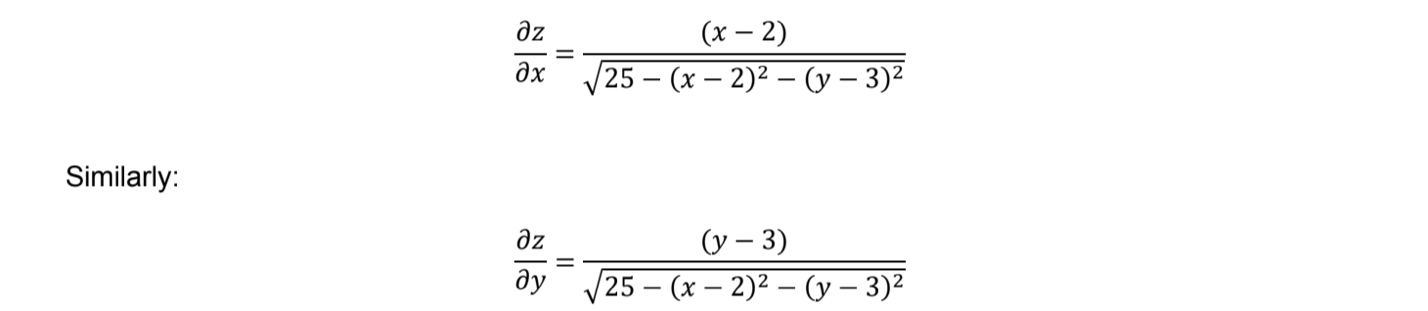
Week#5 – Satish Ramachandran

# 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:



'''

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):

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)

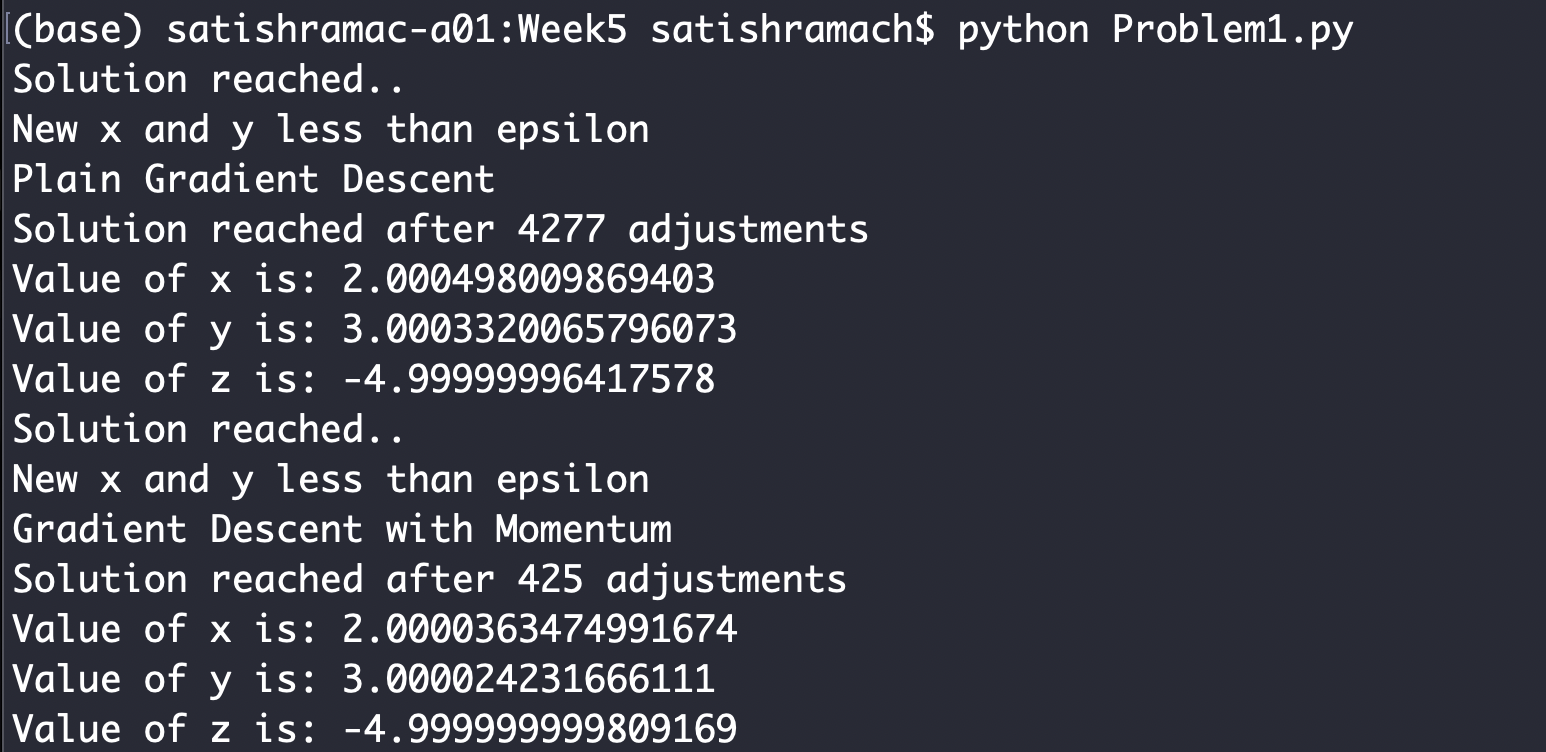
print('Gradient Descent with Momentum')

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)))

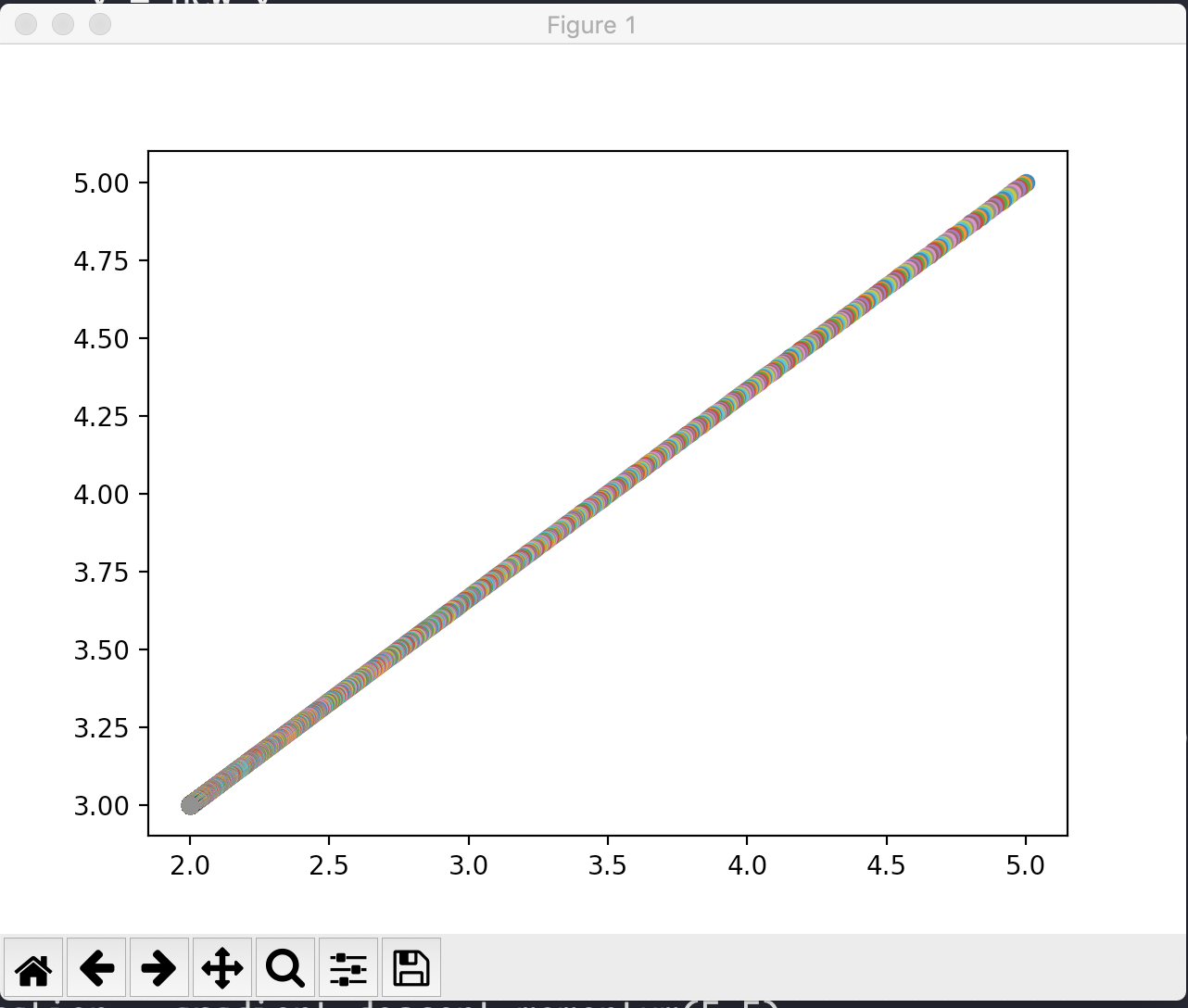


# Problem#2

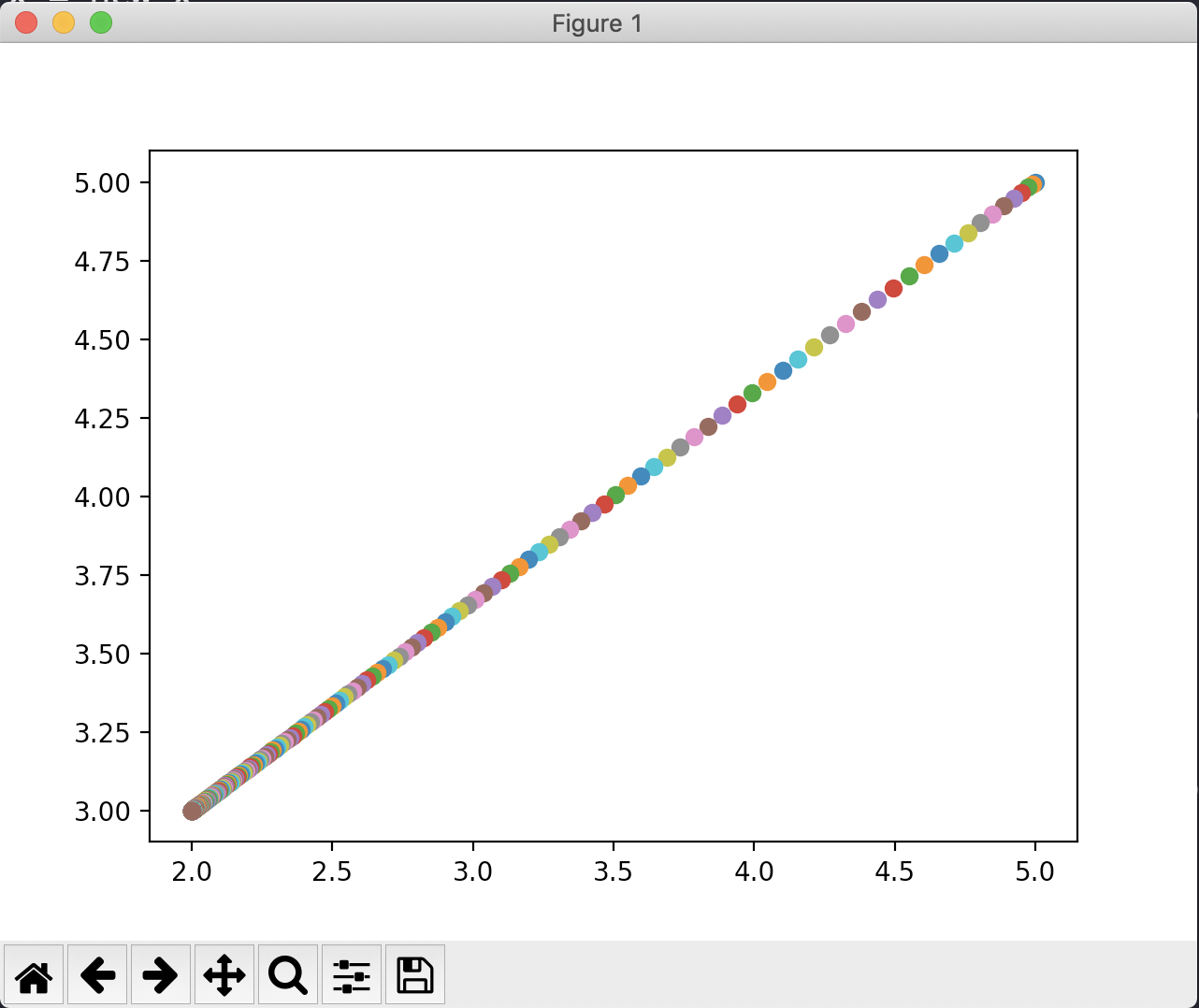
Clearly, GD with momentum converges faster than the plain Gradient Descent algorithm. The number of iterations is 4277 vs 425, which means, it took the momentum based algorithm only 10% of the number of iterations.

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.