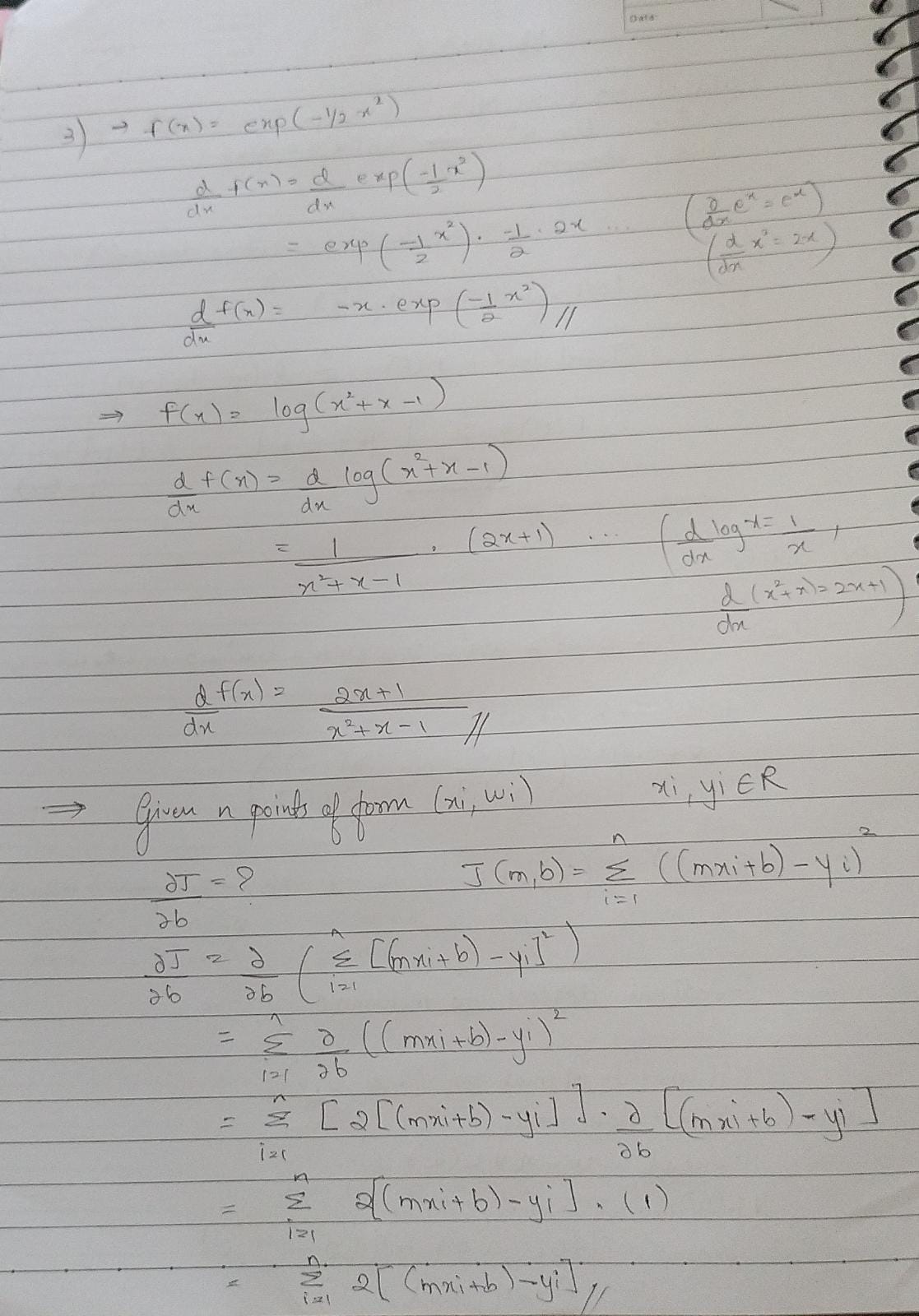
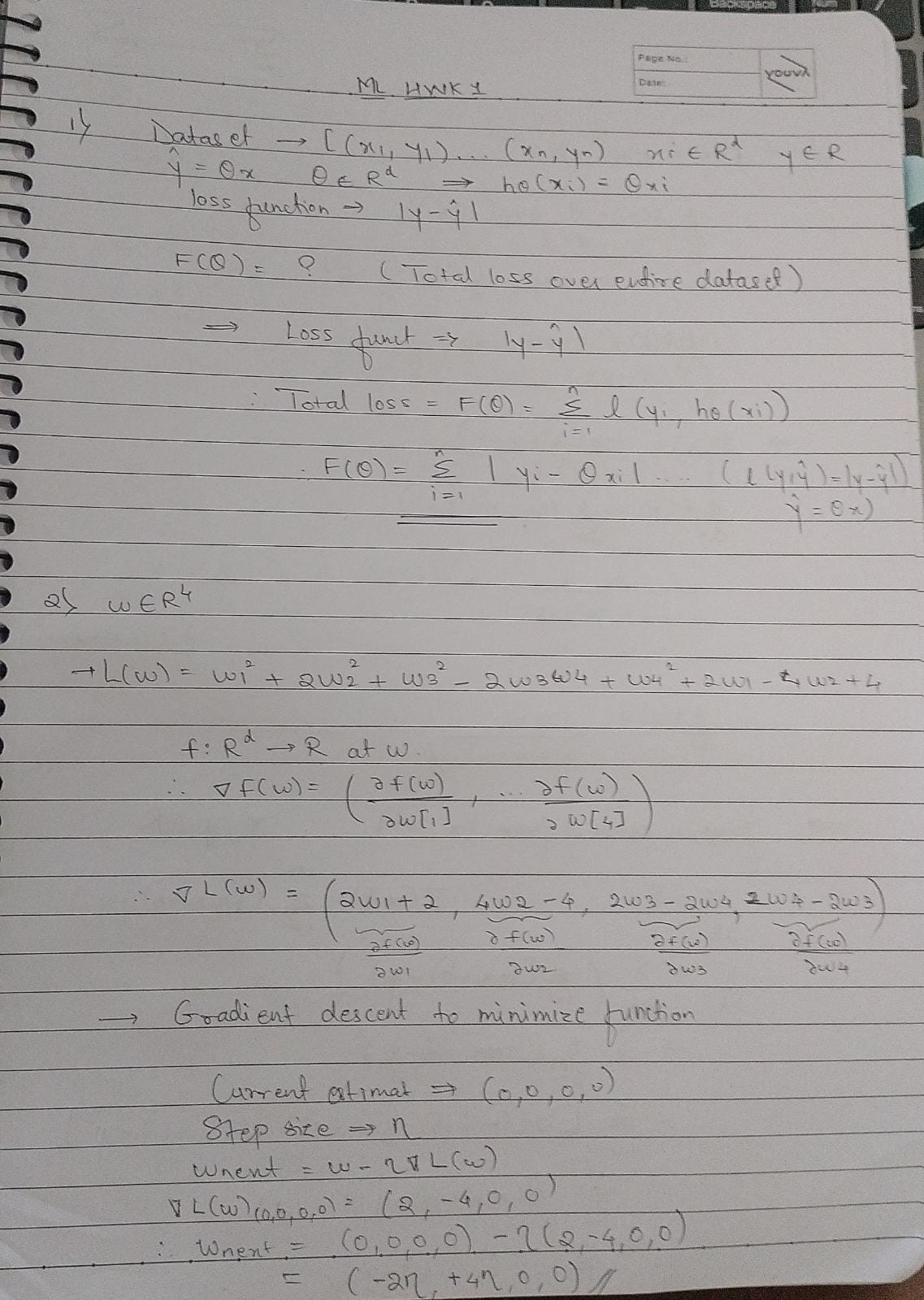
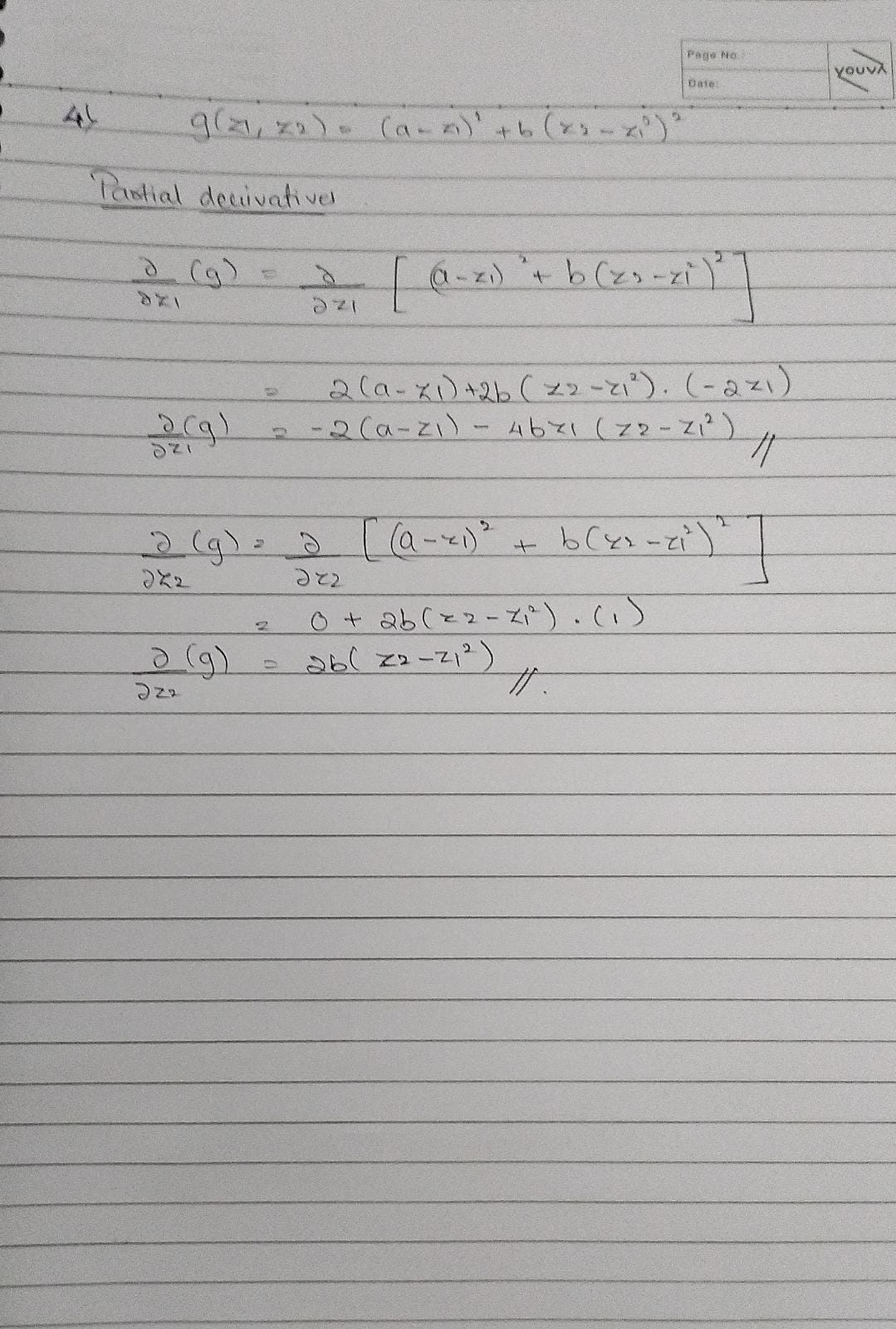
Bhargavi Poyekar- CH33454

**CMSC 678 Machine Learning: Homework Assignment**

****

****

**CODE:**

''' Objective Function -> g(z1, z2) = (a - z1)^2 + b \* (z2 - z1^2)^2

# Aim: To minimize the objective function

# Partial Derivatives:

    --> d/dz1(g) = -2\*(a - z1) - 4 \* b \* z1(z2 - z1^2)

    --> d/dz2(g) = 2b (z2 - z1^2)

'''

# partial derivative w.r.t z1

def partial\_z1(a,b, z1,z2):

    partial = (-2\*(a-z1))-(4\*b\*z1\*(z2-(z1\*\*2)))

    return partial

# partial derivative w.r.t z2

def partial\_z2(a,b, z1,z2):

    partial = 2\*b\*(z2-z1\*\*2)

    return partial

# Objective function value

def g\_z1\_z2(a,b, z1,z2):

    g=(a-z1)\*\*2 + b\*(z2-(z1\*\*2))\*\*2

    return g

# Update z1, z2 values

def update\_z1\_z2(z1,z2, lr, gz1, gz2):

    z1\_upd=z1-(lr\*gz1)

    z2\_upd=z2-(lr\*gz2)

    return z1\_upd,z2\_upd

# y = g(z1, z2)

y\_values=[]

# initial a and b

a=1

b=100

# initial z1, z2

z1=1

z2=3

z1\_vals=[z1]

z2\_vals=[z2]

# learning rate

lr\_vals=[0.001]

# Calculate g(z1,z2) for initial values

y\_values.append(g\_z1\_z2(a,b,z1,z2))

# Calculate Partial derivatives for initial values

partial\_g\_z1=partial\_z1(a,b,z1,z2)

partial\_g\_z2=partial\_z2(a,b,z1,z2)

partialz1=[partial\_g\_z1]

partialz2=[partial\_g\_z2]

for i in range(1,50):

    # Update z1 and z2 values for each iteration

    z1,z2=update\_z1\_z2(z1,z2, lr\_vals[0], partial\_g\_z1,partial\_g\_z2)

    # Storing those values for finding optimal values

    z1\_vals.append(z1)

    z2\_vals.append(z2)

    # Calculate g(z1,z2)

    y\_values.append(g\_z1\_z2(a,b,z1,z2))

    #Calculate partial derivatives

    partial\_g\_z1=partial\_z1(a,b,z1,z2)

    partial\_g\_z2=partial\_z2(a,b,z1,z2)

    partialz1.append(partial\_g\_z1)

    partialz2.append(partial\_g\_z2)

# Finding minimum value of g(z1,z2)

min\_y=min(y\_values)

index\_min\_y=y\_values.index(min\_y)

# Finding optimal value of z1, z2

z1\_opt=z1\_vals[index\_min\_y]

z2\_opt=z2\_vals[index\_min\_y]

# Table for displaying all the values

from prettytable import PrettyTable

columns = ["Iter", "z1", "z2", "Partial z1", "Partial z2", "g(z1,z2)"]

myTable = PrettyTable()

# Add Columns

myTable.add\_column(columns[0], range(50))

myTable.add\_column(columns[1], z1\_vals)

myTable.add\_column(columns[2], z2\_vals)

myTable.add\_column(columns[3], partialz1)

myTable.add\_column(columns[4], partialz2)

myTable.add\_column(columns[5], y\_values)

print(myTable)

# Plotting g(z1,z2) for each iteration

import matplotlib.pyplot as plt

import numpy as np

xpoints=range(50)

ypoints=y\_values

plt.title(f"g(z1,z2) with learning rate={lr\_vals[0]}\n (a,b)=({a,b}) \n initial (z1,z2)=({z1\_vals[0],z2\_vals[0]}\n Optimal (z1\*,z2\*)=({z1\_opt,z2\_opt})")

plt.plot(xpoints,ypoints)

plt.xlabel("Iterations")

plt.ylabel("g(z1,z2)")

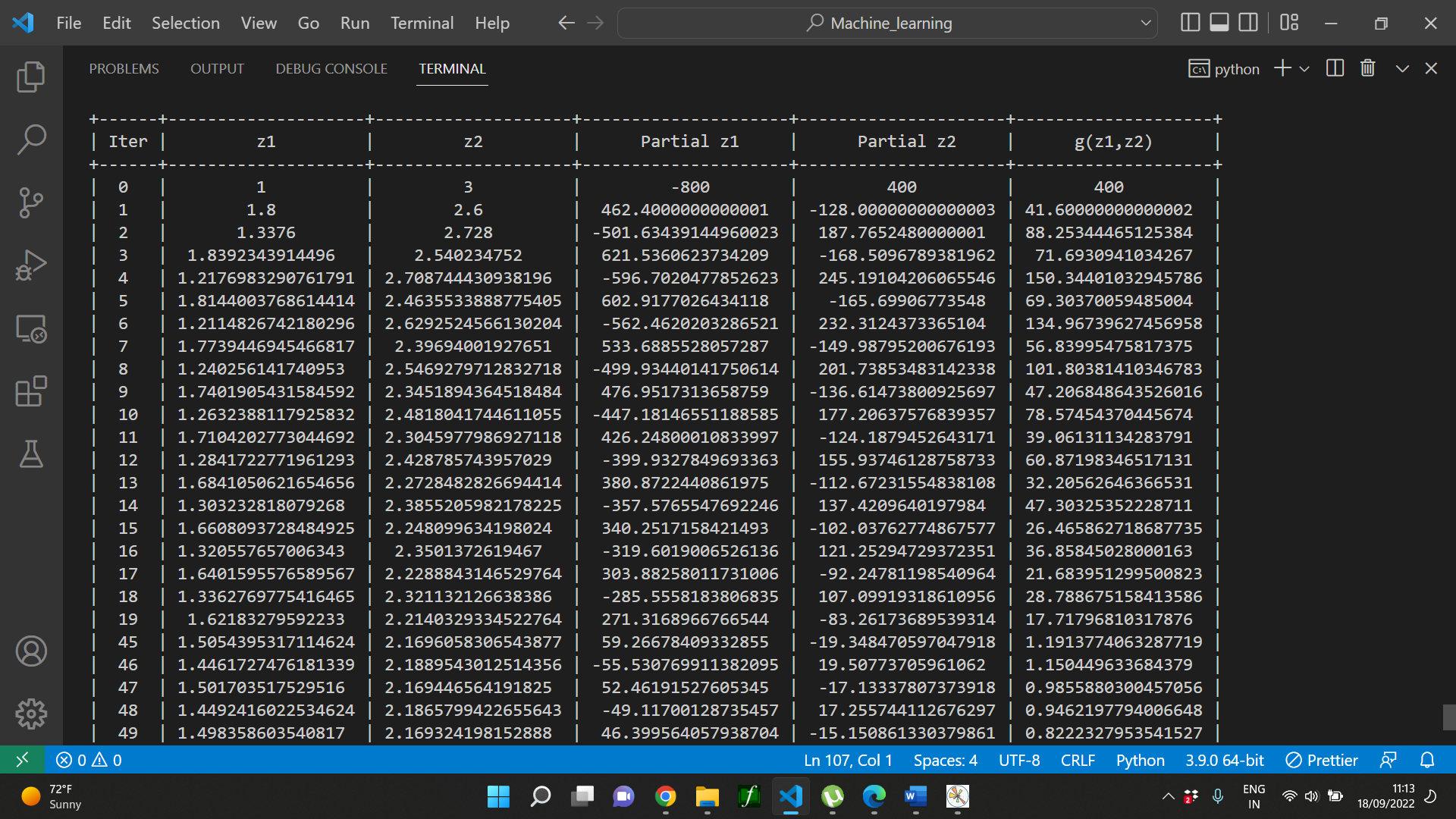
plt.show()

**OUTPUT:**

**(a, b) = (1, 100)**

**(z1, z2) = (1, 3)**

**Learning Rate: 0.001**

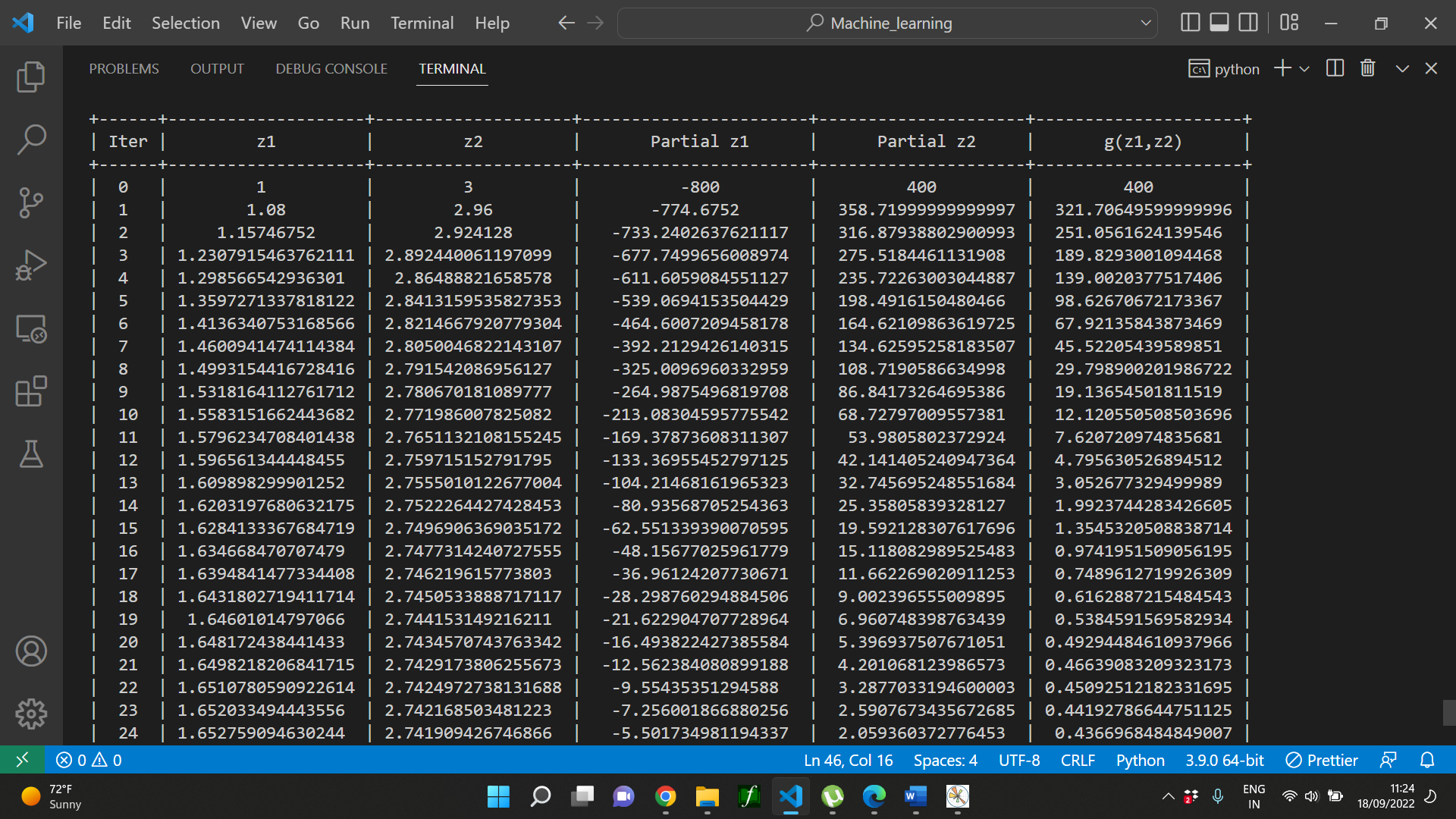


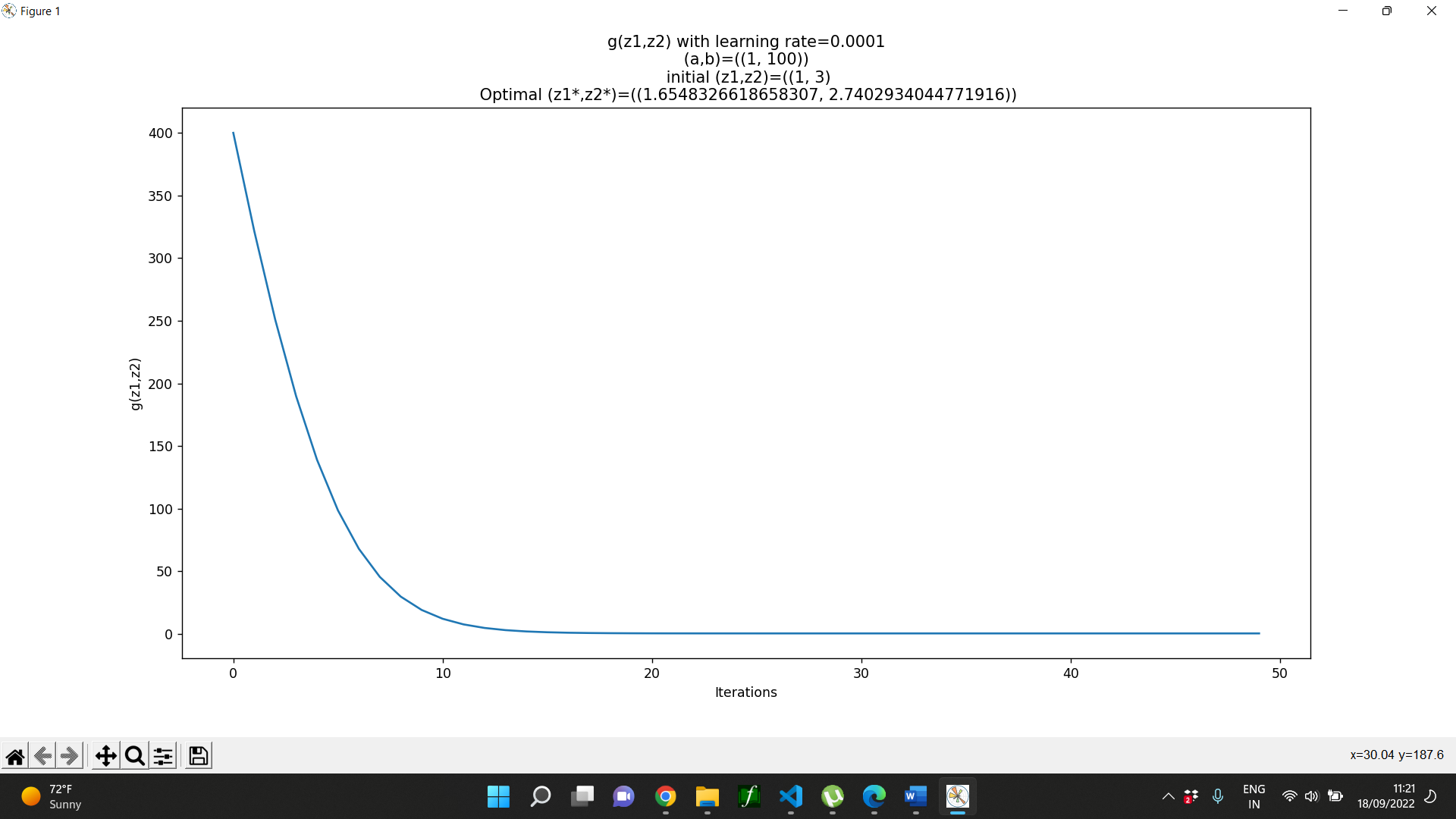


**(a, b) = (1, 100)**

**(z1, z2) = (1, 3)**

**Learning Rate: 0.0001**





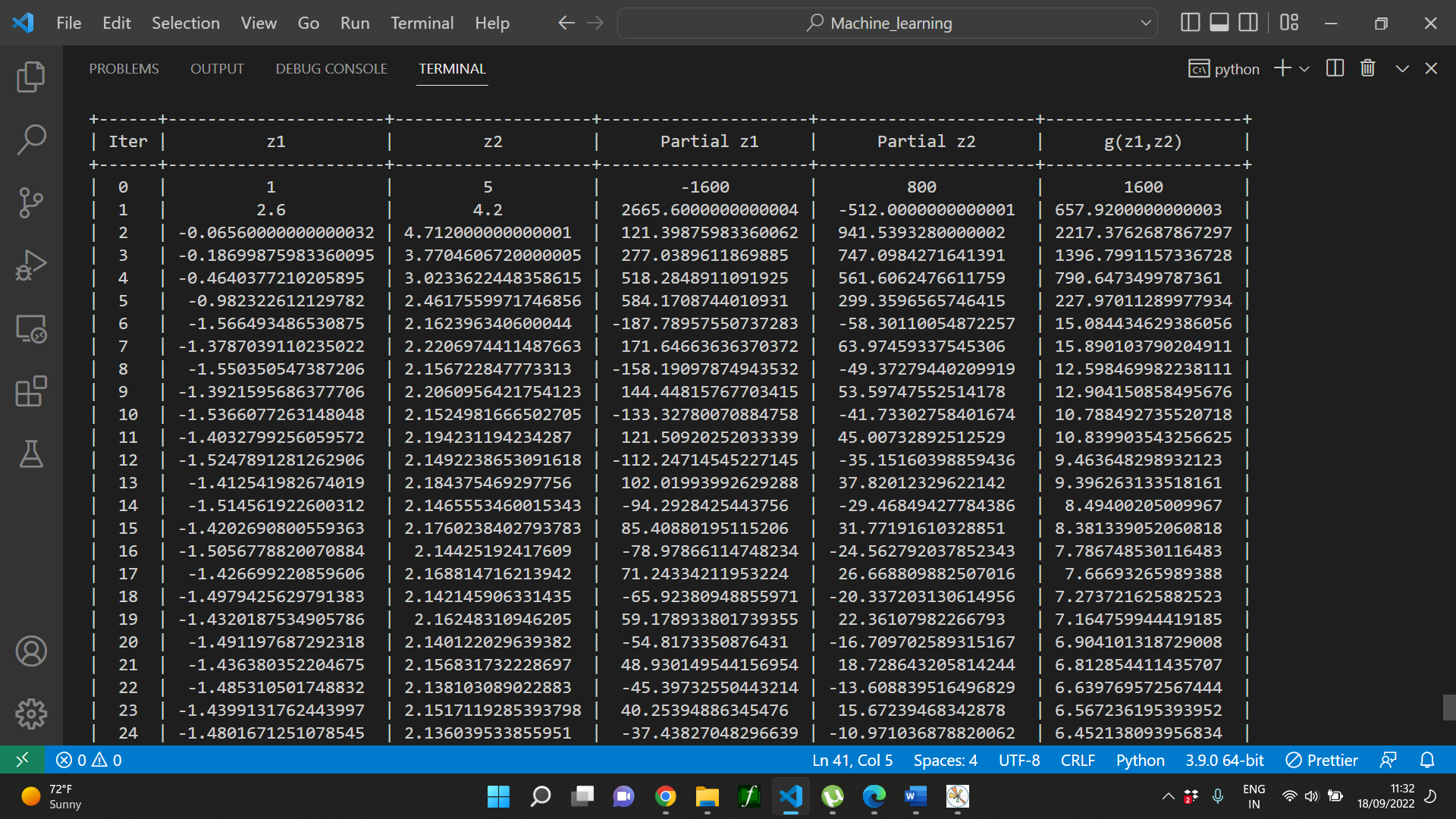
With the same initial values of a, b and z1, z2, after changing the learning rate, I got smooth curve for 0.0001.

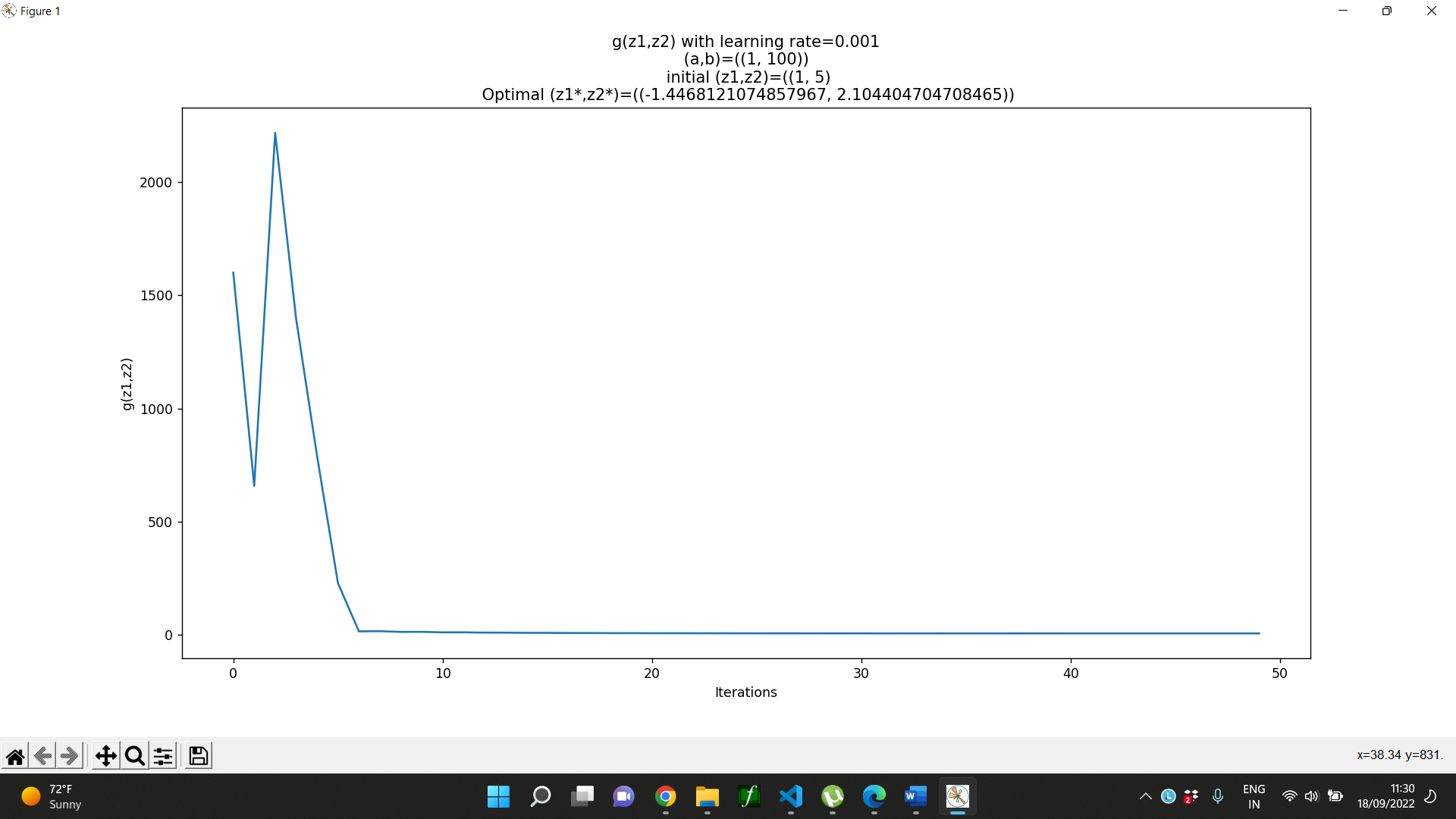
Changing values for z1, z2:

**(a, b) = (1, 100)**

**(z1, z2) = (1, 5)**

**Learning Rate: 0.001**

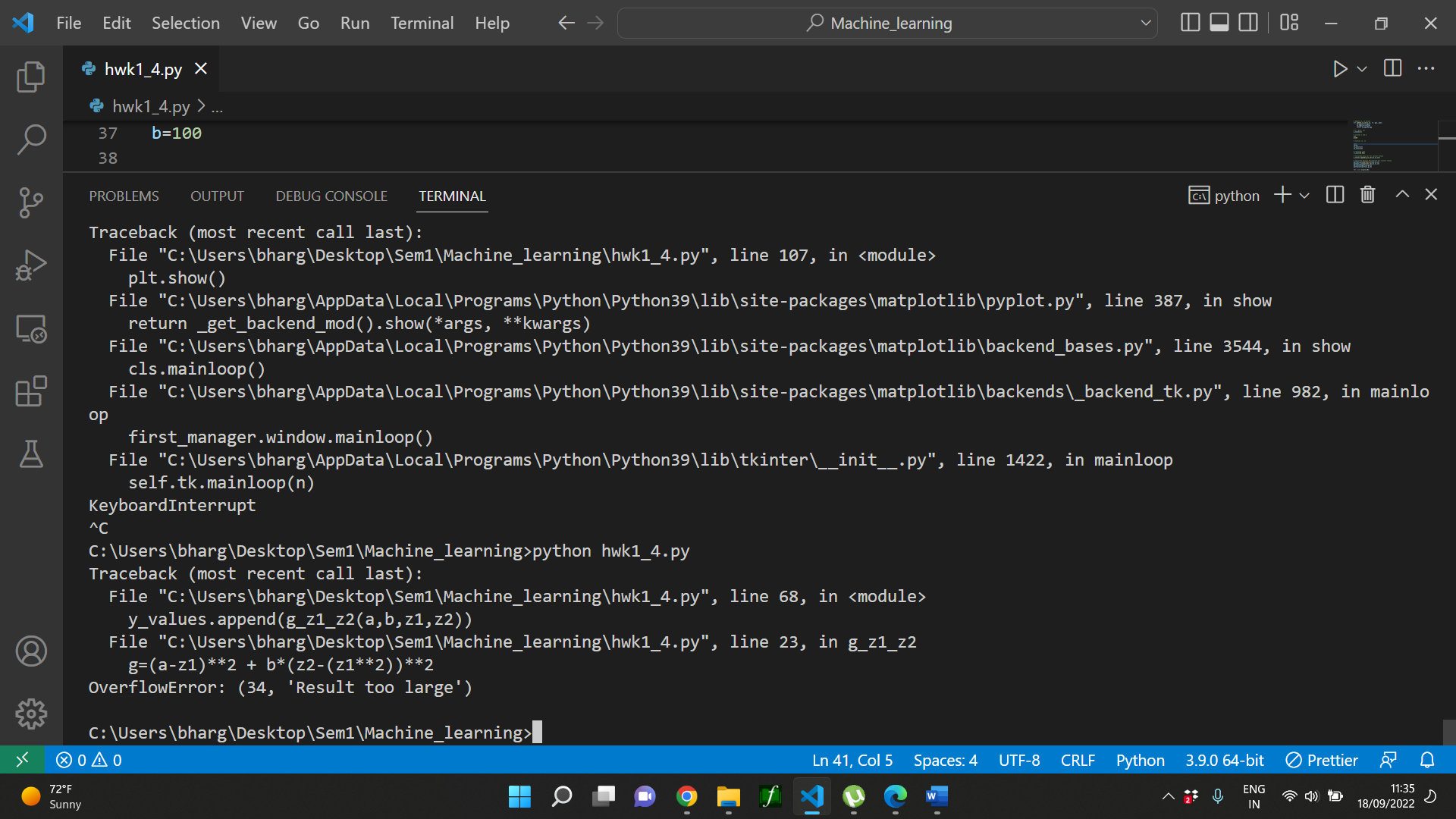




**(a, b) = (1,100)**

**(z1, z2) = (1, 10)**

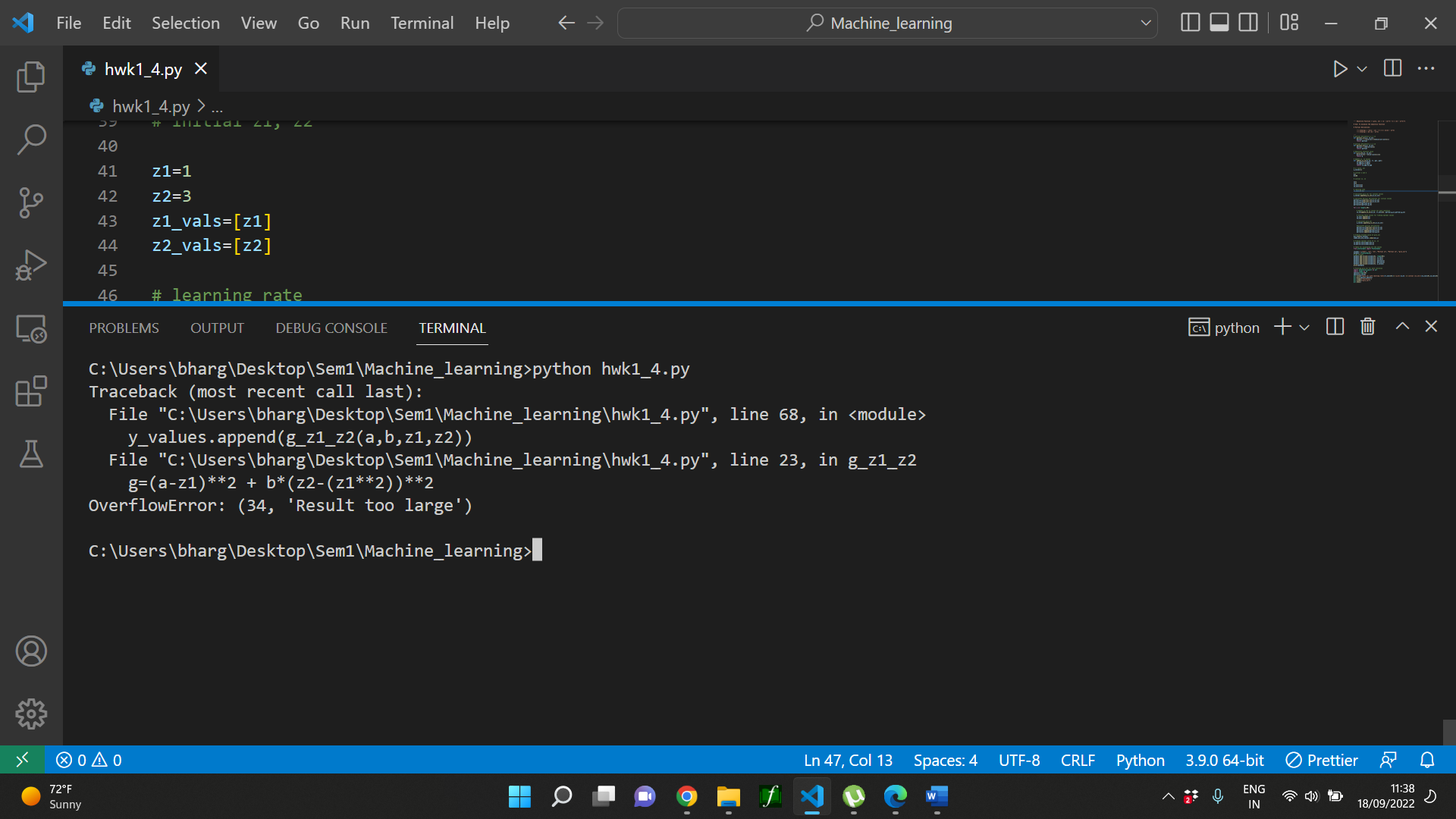
**Learning Rate: 0.001**



**(a, b) = (1, 100)**

**(z1, z2) = (1, 3)**

**Learning Rate: 0.01**



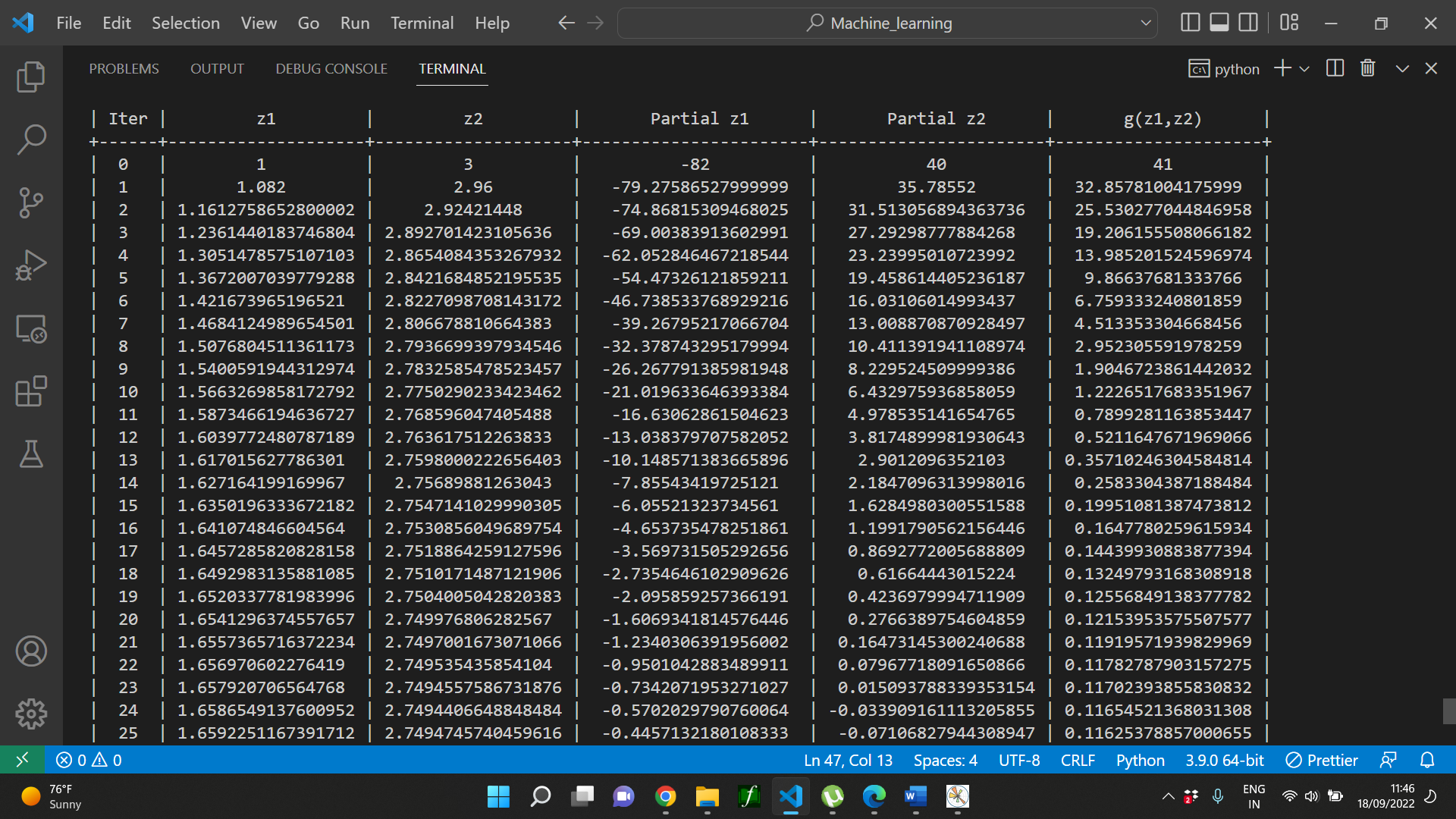
The g value increased exponentially for greater value of z2 and smaller learning rate, because b value is large, so loss is not decreasing, instead it is increasing.

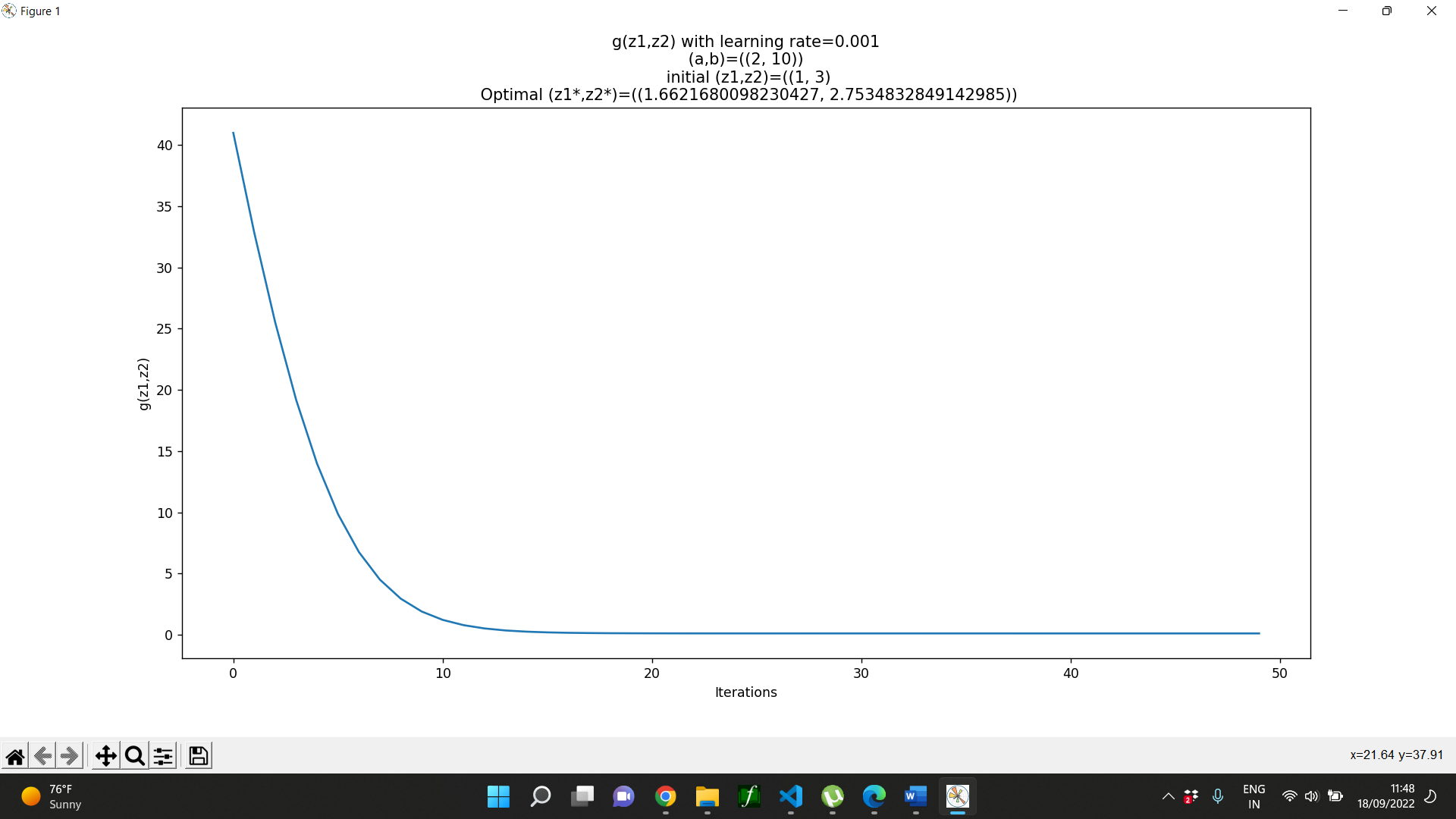
**Different set of a and b:**

**(a, b) = (2, 10)**

**(z1, z2) = (1, 3)**

**Learning Rate: 0.001**



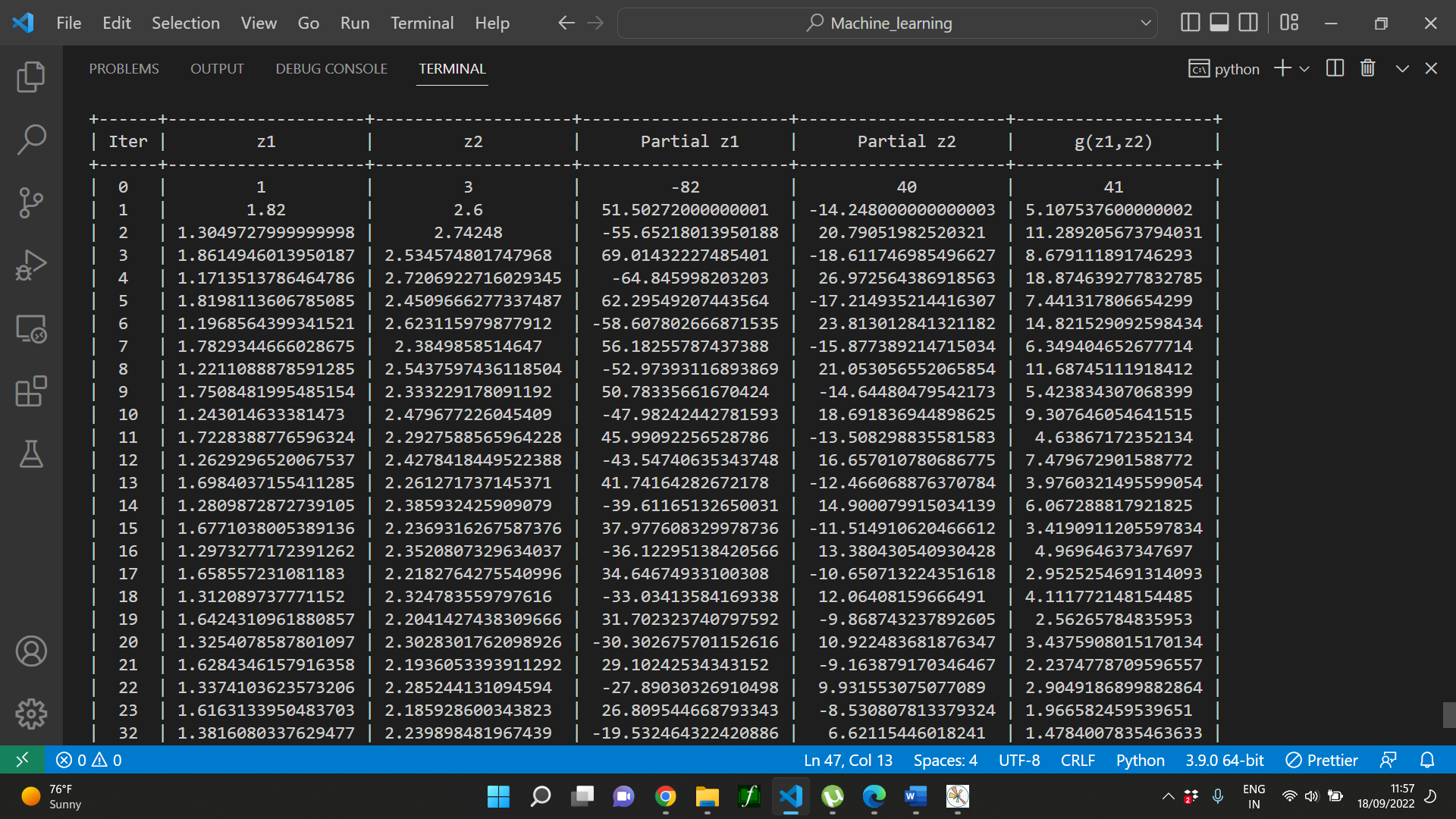


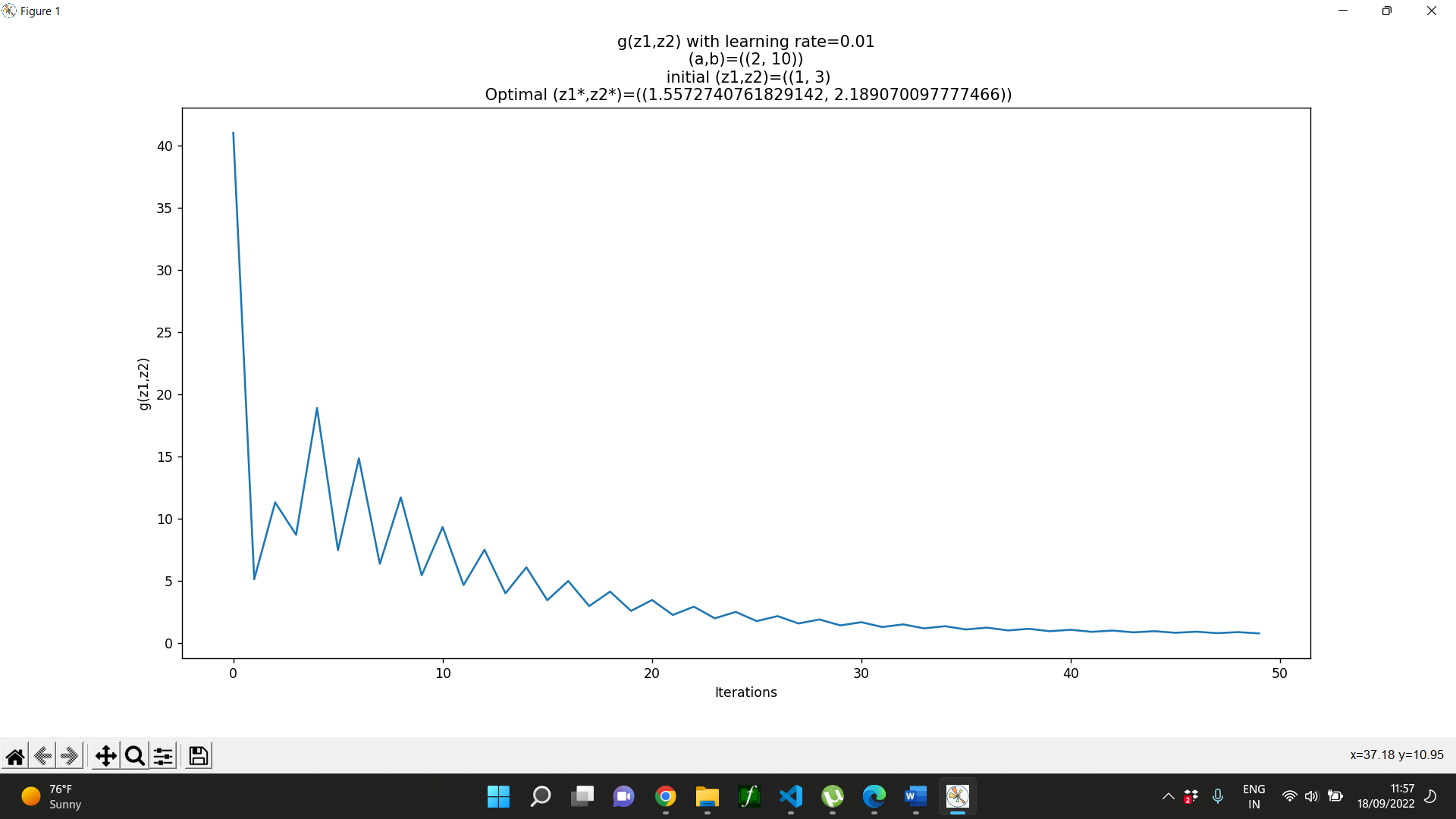
When I decreased value of b, I found a smooth curve, for the learning rate 0.001.

**(a, b) = (2, 10)**

**(z1, z2) = (1, 3)**

**Learning Rate: 0.01**



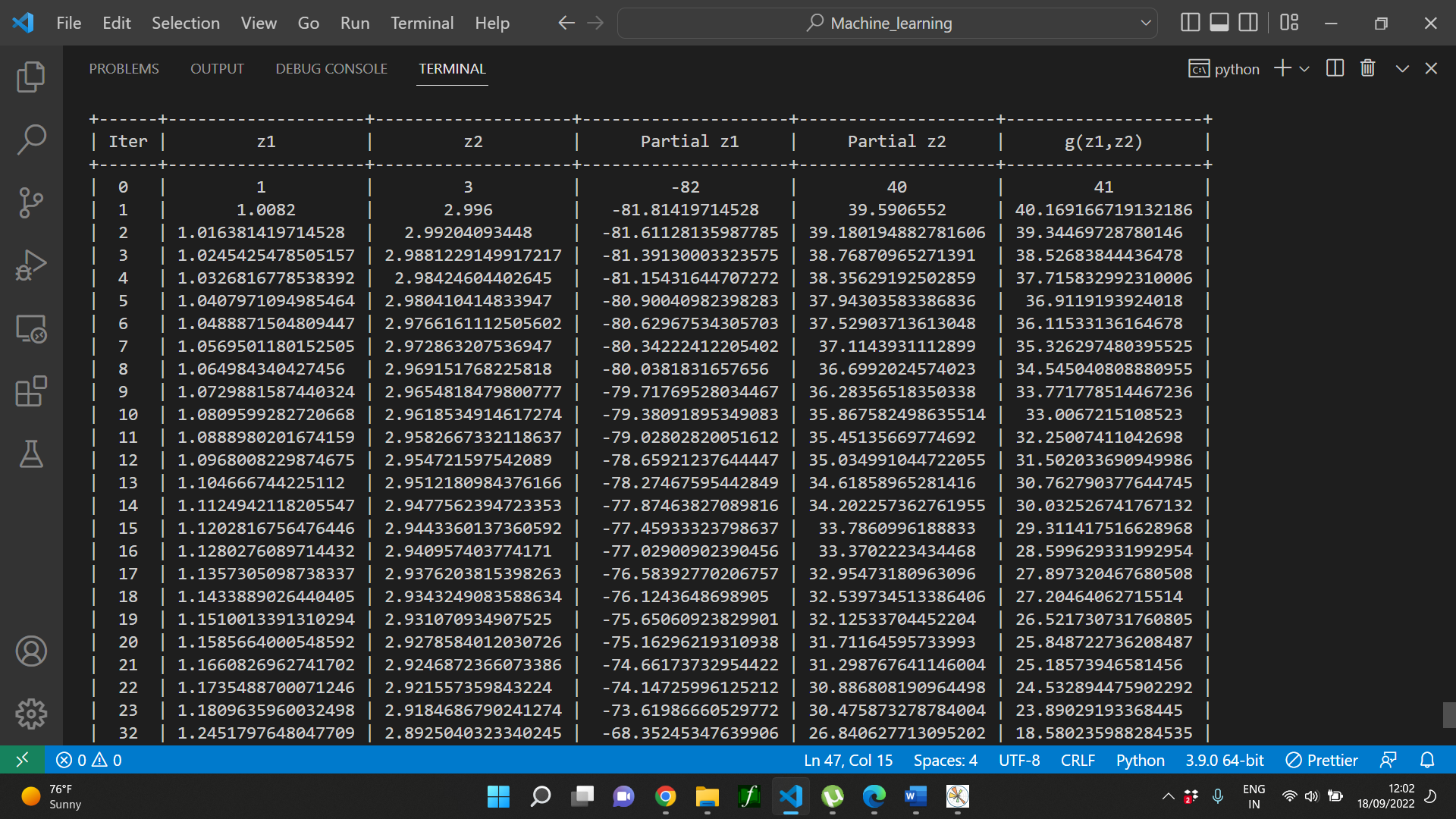


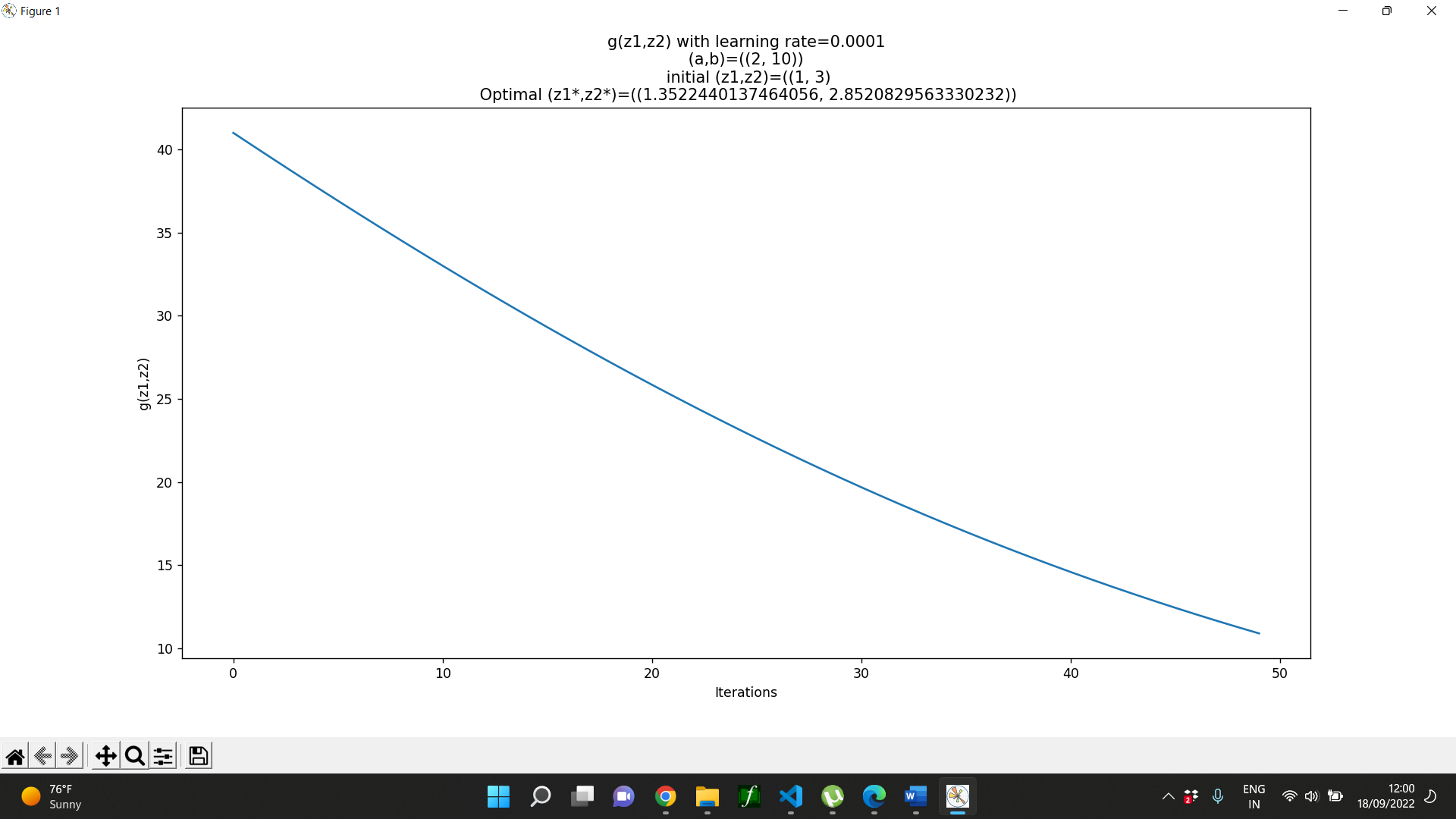
The learning rate is high, so model is trying to learn fast, but in that process it is updating in larger amount, so I can see both increasing and decreasing values for g in the start.

**(a, b) = (2, 10)**

**(z1, z2) = (1, 3)**

**Learning Rate: 0.0001**



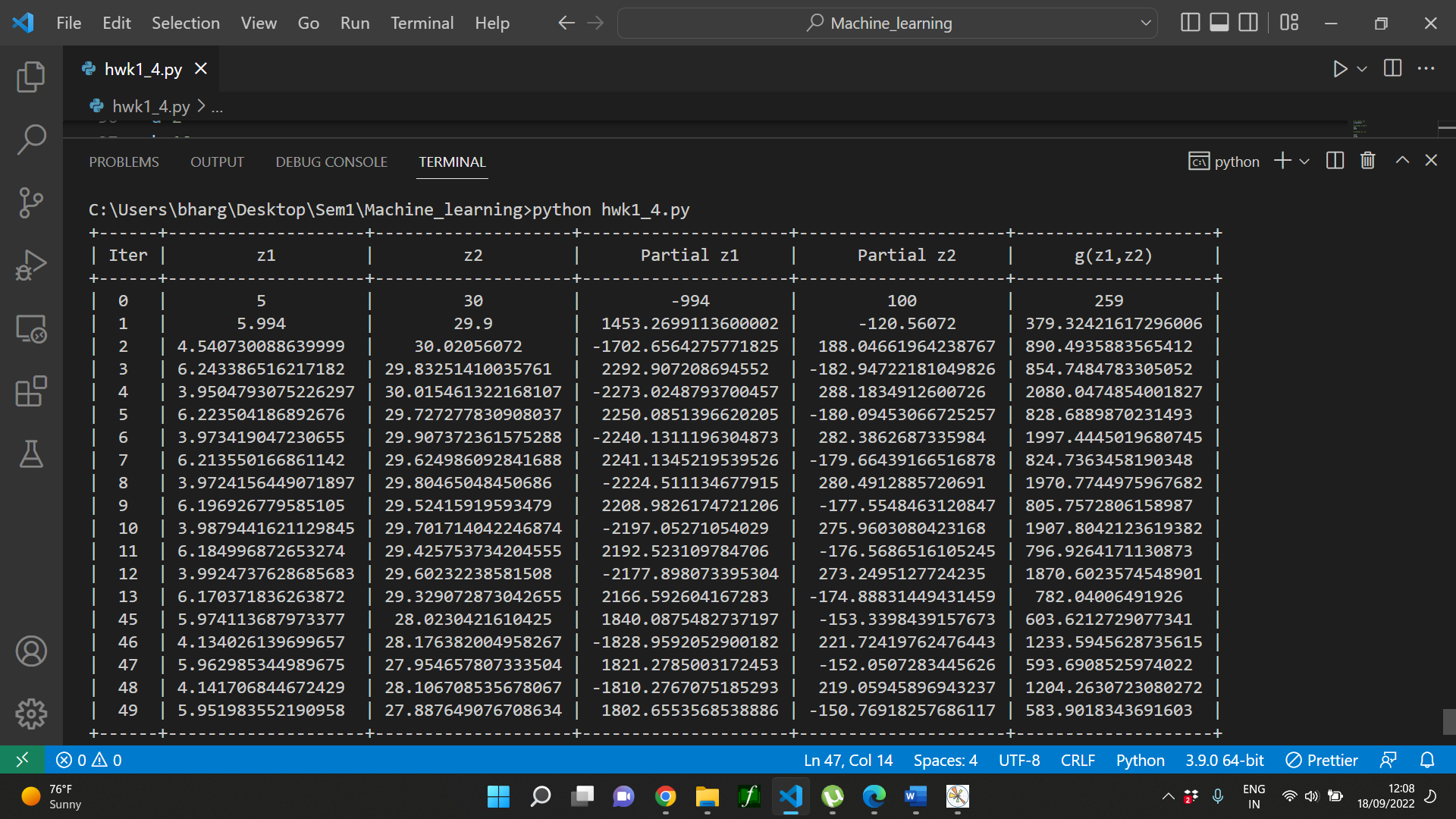


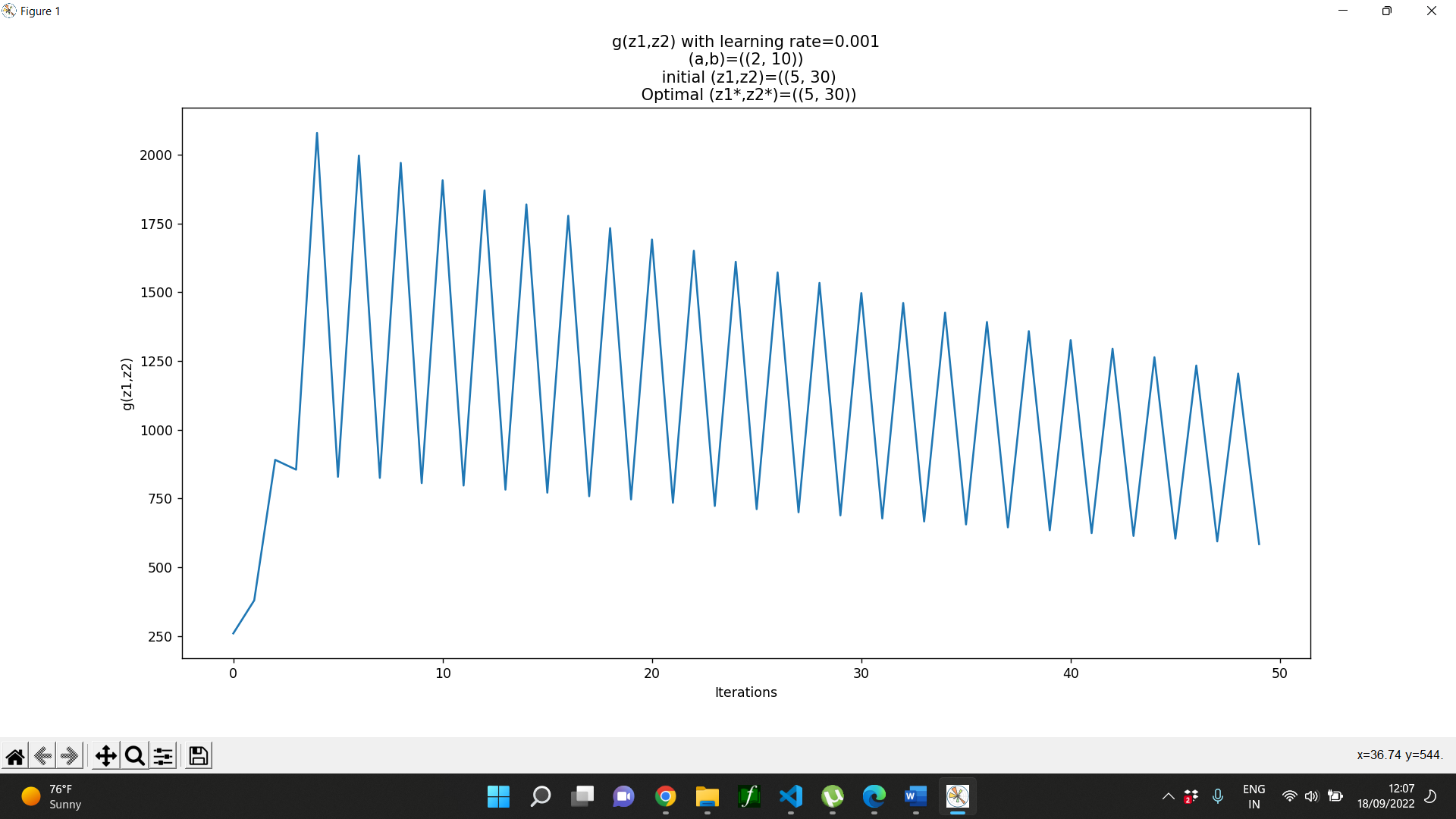
Model is learning too slow, hence the function value decreased very slowly.

**(a, b) = (2, 10)**

**(z1, z2) = (5, 30)**

**Learning Rate: 0.001**



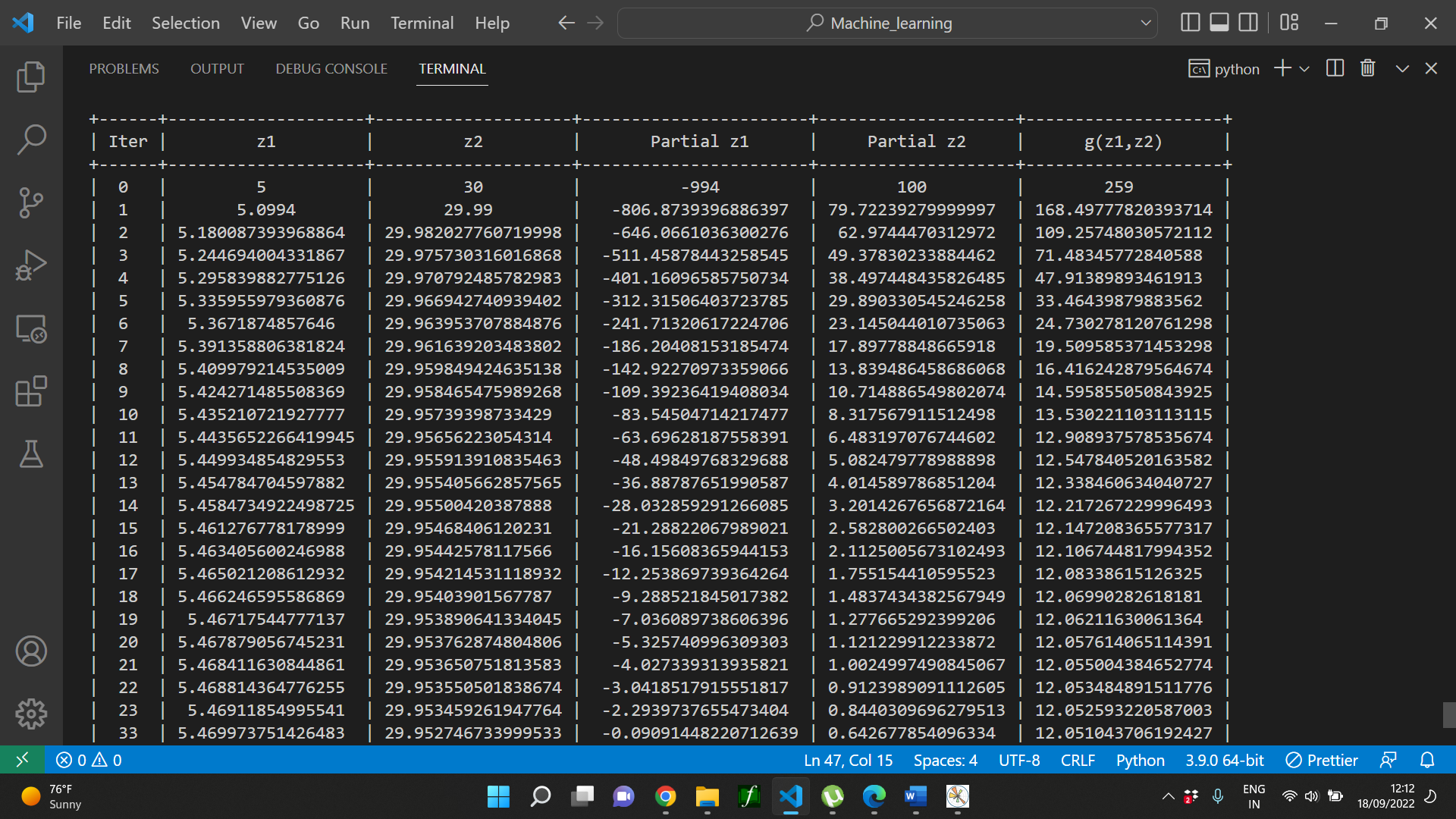


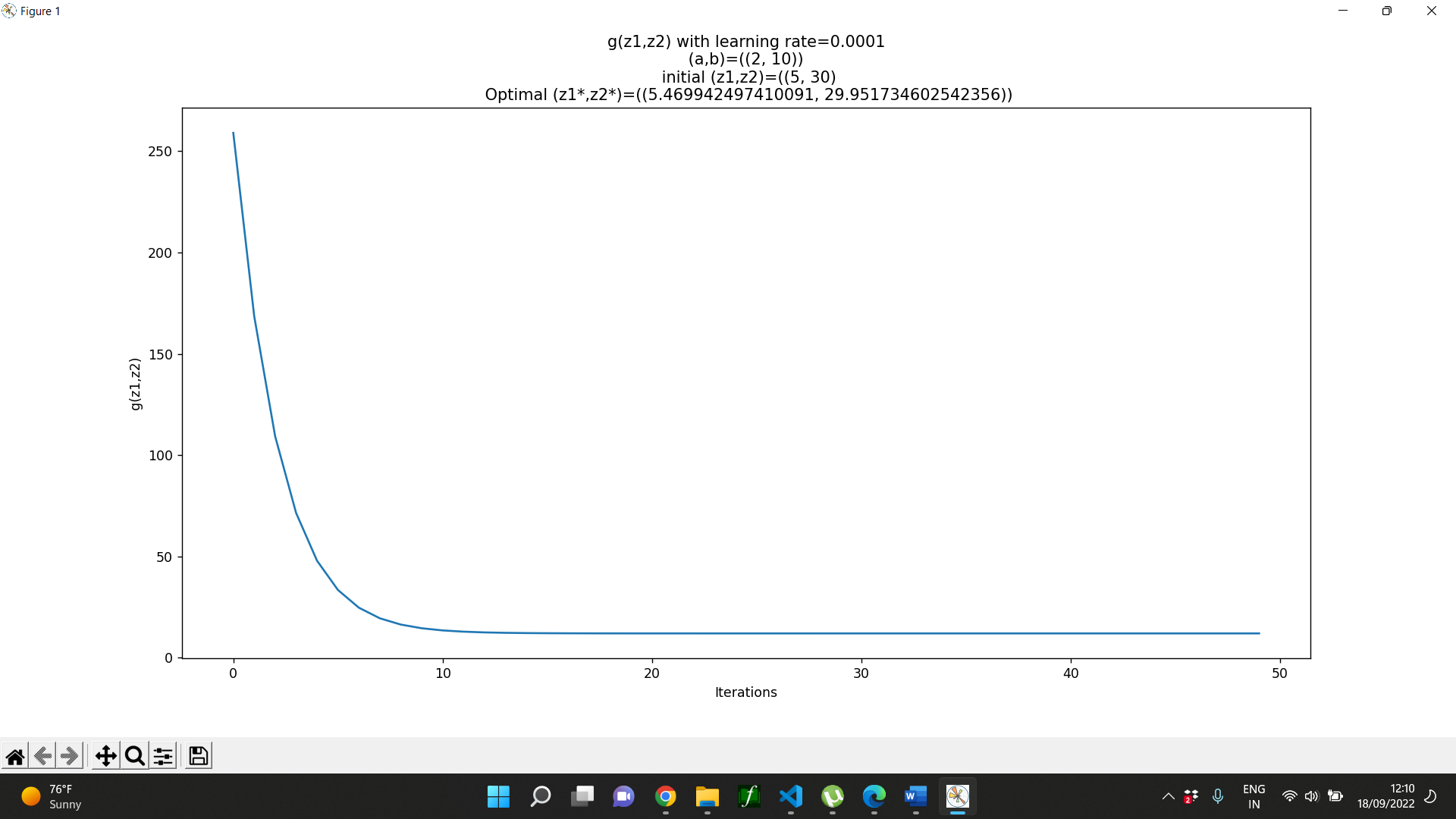
For the higher values of z1 and z2, model is trying to learn too fast, and the values are increasing and decreasing sometimes.

**(a, b) = (2, 10)**

**(z1, z2) = (5, 30)**

**Learning Rate: 0.0001**





For the same values of a, b, z1 and z2, after decreasing the learning rate, model learns in a smooth manner with smaller updates.

**Conclusion:**

1. The perfect selection of learning rate depends on the value of gradient.
2. For big gradient, we need smaller