**p1**

d=-

**p2**

**(a)**Hessian =

**(b)**yes

    prove:

        assume that H=

        let =

        then

        so that matrix H is positive semi-definite,thus the function is convex

**p3**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

**#(1)**

d = [100,100,100,127,127,127,152,152,152,178,178,178]

t\_squre = [0.36,0.38,0.46,0.46,0.49,0.51,0.50,0.53,0.56,0.55,0.58,0.61]

plt.scatter(d, t\_squre, color='blue', label='Data Points')

plt.xlabel('distance/cm')

plt.ylabel('time^2/s^2')

plt.title('Distance vs Time^2')

plt.legend()

plt.show()

**#(2)**

my\_linear\_model = LinearRegression()

my\_linear\_model.fit(np.array(d).reshape(-1, 1), np.array(t\_squre).reshape(-1, 1)) # training model

print(f'Coefficients: {my\_linear\_model.coef\_[0][0]:.4f}') # priting trained/optimal coeffs of polynomial terms

print(f'Intercept: {my\_linear\_model.intercept\_[0]:.4f}') # trained/optimal worker bias

**#(3)**

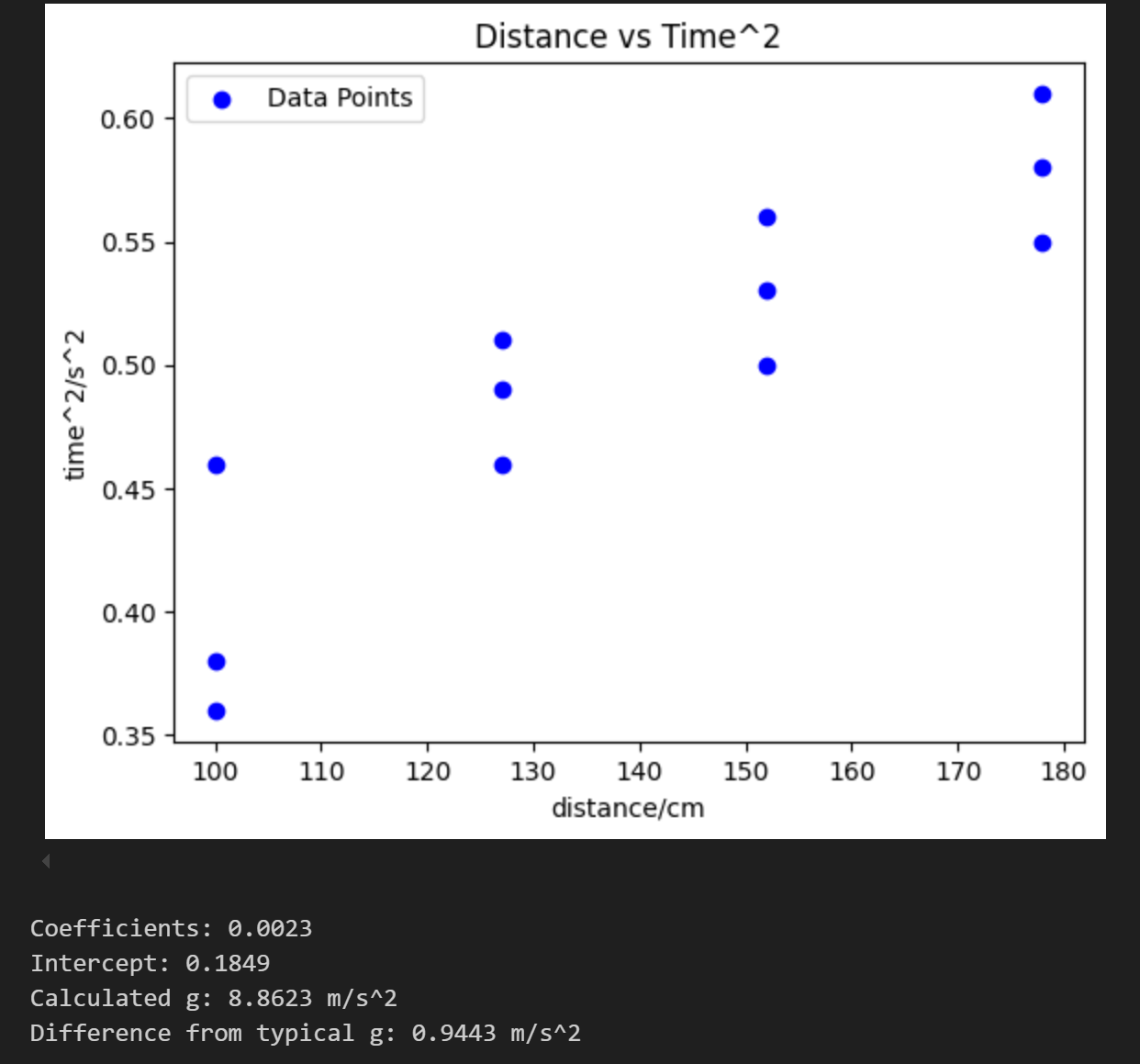
g\_cal = 2/(my\_linear\_model.coef\_[0][0]\*100) # calculating g from coeffs

g\_typical= 9.80665 # typical value of g

delta\_g = abs(g\_cal - g\_typical) # calculating delta\_g

print(f'Calculated g: {g\_cal:.4f} m/s^2')

print(f'Difference from typical g: {delta\_g:.4f} m/s^2')



**P4**

**(1)**

import numpy as np

import pandas as pd

from sklearn.metrics import mean\_squared\_error

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

def model\_D(x,D):

    poly = PolynomialFeatures(degree=D, include\_bias=True)

    X\_poly = poly.fit\_transform(x)

    if D == 2:

        w = np.array([[5,1.5,0.03]])

    elif D == 3:

        w = np.array([[2,0.5,0.01,-0.001]])

    return X\_poly @ w.T

x\_validation = np.array([[4], [8], [9],[14]])

y\_validation = np.array([[10], [20], [25], [35]])

y\_pred\_2 = model\_D(x\_validation, 2)

y\_pred\_3 = model\_D(x\_validation, 3)

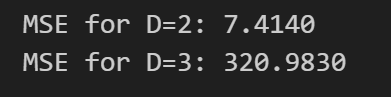
mse\_2 = mean\_squared\_error(y\_validation, y\_pred\_2)

mse\_3 = mean\_squared\_error(y\_validation, y\_pred\_3)

print(f'MSE for D=2: {mse\_2:.4f}')

print(f'MSE for D=3: {mse\_3:.4f}')

output:



**(2)**MSE for D=2 is much smaller than MSE for D=3,thus D=2 is more approperiate

**P5**

**(1)**

import numpy as np

import matplotlib.pyplot as plt

#function and its derivative

f = lambda v: v\*\*4-5\*v\*\*2-3\*v

f\_derivative = lambda w: 4\*w\*\*3 - 10\*w - 3

max\_no\_iterations = 3

alpha = 0.1

w = np.zeros(max\_no\_iterations)

w0 = -2.0

w[0] = w0

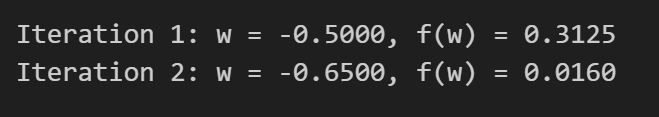
#gradient descent

for i in range(1,max\_no\_iterations):

    w[i] = w[i-1] - alpha \* f\_derivative(w[i-1])

    print(f'Iteration {i}: w = {w[i]:.4f}, f(w) = {f(w[i]):.4f}')

output:



**(2.a)**

import numpy as np

import matplotlib.pyplot as plt

#function and its derivative

f = lambda v: v\*\*4-5\*v\*\*2-3\*v

f\_derivative = lambda w: 4\*w\*\*3 - 10\*w - 3

max\_no\_iterations = 10000

alphas = [0.2,0.1,0.01,0.001]

epsilon = 0.001

w = np.zeros(max\_no\_iterations)

w0 = -2.0

w[0] = w0

#gradient descent

for alpha in alphas:

    print(f'Using alpha = {alpha}')

    w[0] = w0  # Reset w[0] for each alpha

     #gradient descent

    for i in range(1,max\_no\_iterations):

        w[i] = w[i-1] - alpha \* f\_derivative(w[i-1])

        if abs(f\_derivative(w[i])) < epsilon:

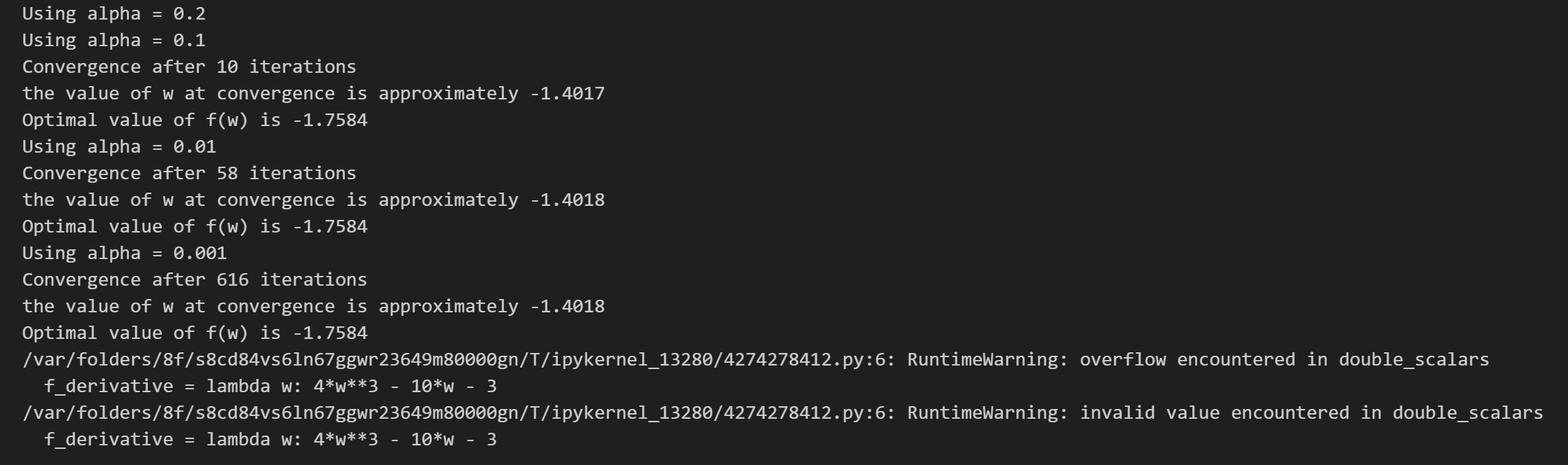
            print(f'Convergence after {i} iterations ')

            print(f'the value of w at convergence is approximately {w[i]:.4f}')

            print(f'Optimal value of f(w) is {f(w[i]):.4f}')

            break

output:



When alpha = 0.2,no optimal value is obtained

**(2.b)**

import numpy as np

import matplotlib.pyplot as plt

#function and its derivative

f = lambda v: v\*\*4-5\*v\*\*2-3\*v

f\_derivative = lambda w: 4\*w\*\*3 - 10\*w - 3

max\_no\_iterations = 10000

alpha = 0.01

epsilon = 0.001

w = np.zeros(max\_no\_iterations)

w0 = 0.5

#gradient descent

for i in range(1,max\_no\_iterations):

    w[i] = w[i-1] - alpha \* f\_derivative(w[i-1])

    if abs(f\_derivative(w[i])) < epsilon:

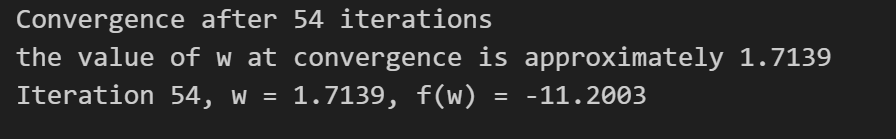
        print(f'Convergence after {i} iterations ')

        print(f'the value of w at convergence is approximately {w[i]:.4f}')

        break

print(f'Iteration {i}, w = {w[i]:.4f}, f(w) = {f(w[i]):.4f}')

output:



For my answer in a,I didn't get the global optimal value of w\_0.But if we choose the alpha = 0.01 and w\_0 = 0.5,we'll get the global optimal value of w\_0.