PARTICLE SWARM OPTIMIZATION FOR FUNCTION OPTIMIZATION

import numpy as np

# -------------------------------

# Function Definition

# -------------------------------

def f(position):

x, y = position

return x\*\*2 + y\*\*2

def fitness(position):

return f(position)

# -------------------------------

# PSO Parameters

# -------------------------------

num\_particles = 30

max\_iter = 100 # small number of iterations

w = float(input("Enter inertia weight (w): "))

c1 = float(input("Enter cognitive coefficient (c1): "))

c2 = float(input("Enter social coefficient (c2): "))

# Search space

x\_min, x\_max = -10, 10

y\_min, y\_max = -10, 10

# Stopping thresholds

fitness\_threshold = 1e-6

velocity\_threshold = 1e-6

# -------------------------------

# Initialize particles (2D)

# -------------------------------

positions = np.random.uniform([x\_min, y\_min], [x\_max, y\_max], (num\_particles, 2))

velocities = np.random.uniform(-1, 1, (num\_particles, 2))

pbest\_positions = positions.copy()

pbest\_scores = np.array([fitness(p) for p in positions])

gbest\_position = pbest\_positions[np.argmin(pbest\_scores)]

gbest\_velocity = velocities[np.argmin(pbest\_scores)].copy()

gbest\_score = np.min(pbest\_scores)

# -------------------------------

# PSO Main Loop

# -------------------------------

iteration = 0

while (gbest\_score > fitness\_threshold and

np.linalg.norm(gbest\_velocity) > velocity\_threshold and

iteration < max\_iter):

iteration += 1

for i in range(num\_particles):

r1, r2 = np.random.rand(2), np.random.rand(2)

# Velocity update

velocities[i] = (w \* velocities[i] +

c1 \* r1 \* (pbest\_positions[i] - positions[i]) +

c2 \* r2 \* (gbest\_position - positions[i]))

# Position update

positions[i] += velocities[i]

# Keep positions inside search space

positions[i][0] = np.clip(positions[i][0], x\_min, x\_max)

positions[i][1] = np.clip(positions[i][1], y\_min, y\_max)

# Evaluate fitness

score = fitness(positions[i])

# Update personal best

if score < pbest\_scores[i]:

pbest\_scores[i] = score

pbest\_positions[i] = positions[i].copy()

# Update global best

best\_idx = np.argmin(pbest\_scores)

if pbest\_scores[best\_idx] < gbest\_score:

gbest\_score = pbest\_scores[best\_idx]

gbest\_position = pbest\_positions[best\_idx].copy()

gbest\_velocity = velocities[best\_idx].copy()

# Print only every 10 iterations

if iteration % 10 == 0 or gbest\_score < fitness\_threshold:

print(f"Iteration {iteration} | Best Position = {gbest\_position}, f(x,y)={gbest\_score:.6f}")

# -------------------------------

# Final Result

# -------------------------------

print("\nApproximate Solution Found by PSO:")

print(f"x ≈ {gbest\_position[0]:.6f}, y ≈ {gbest\_position[1]:.6f}")

print(f"f(x,y) ≈ {gbest\_score:.6e}")

print(f"Iterations taken: {iteration}")

