import numpy as np

# 1. DEFINE THE "MOUNTAIN RANGE" (FITNESS FUNCTION)

# The peak of this mountain is at (2, 3) with an altitude of 100.

def get\_altitude(coords):

x, y = coords[0], coords[1]

peak\_x, peak\_y = 2, 3

# An inverted paraboloid: 100 - ((x-2)^2 + (y-3)^2)

return 100 - np.power(x - peak\_x, 2) - np.power(y - peak\_y, 2)

print("--- Hiker Simulation Setup ---")

try:

n\_hikers = int(input("Enter the number of hikers in the group (default: 20): "))

except ValueError:

print("Invalid input. Using 20 hikers.")

n\_hikers = 20

try:

n\_iterations = int(input("Enter the number of search iterations (default: 50): "))

except ValueError:

print("Invalid input. Using 50 iterations.")

n\_iterations = 50

print("\n--- Starting The Hike ---")

# --- PSO ALGORITHM PARAMETERS ---

w = 0.5 # Inertia: tendency to keep walking in the same direction

c1 = 0.8 # Cognitive: trust in one's own memory

c2 = 0.9 # Social: trust in the group's best-found spot

bounds = [(-10, 10), (-10, 10)] # The boundaries of the map

# --- INITIALIZE THE HIKERS (PARTICLES) ---

# Each hiker's current coordinates (position)

positions = np.random.rand(n\_hikers, 2) \* 20 - 10

# Each hiker's current direction and speed (velocity)

velocities = np.random.rand(n\_hikers, 2) \* 0.1

# Each hiker remembers their personal best spot and its altitude

pbest\_positions = np.copy(positions)

pbest\_altitudes = np.array([get\_altitude(p) for p in positions])

# The group knows the overall best spot found by any hiker

gbest\_index = np.argmax(pbest\_altitudes) # Note: argmax for maximization

gbest\_position = pbest\_positions[gbest\_index]

gbest\_altitude = pbest\_altitudes[gbest\_index]

# --- SETUP AND RUN THE SIMULATION ---

# Print a nice header for the output table

print(f"\n{'Iteration':<12} {'Highest Point Found':<25} {'Max Altitude':<15}")

print("-" \* 55)

# Main loop where the search happens

for i in range(n\_iterations):

for j in range(n\_hikers):

# UPDATE EACH HIKER'S DIRECTION AND SPEED (VELOCITY)

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

cognitive\_pull = c1 \* r1 \* (pbest\_positions[j] - positions[j])

social\_pull = c2 \* r2 \* (gbest\_position - positions[j])

velocities[j] = w \* velocities[j] + cognitive\_pull + social\_pull

# HIKER MOVES TO A NEW POSITION

positions[j] = positions[j] + velocities[j]

# Ensure hikers don't walk off the map

positions[j] = np.clip(positions[j], bounds[0][0], bounds[0][1])

# HIKER CHECKS THE ALTITUDE AT THE NEW SPOT

current\_altitude = get\_altitude(positions[j])

if current\_altitude > pbest\_altitudes[j]: # Note: > for maximization

pbest\_positions[j] = positions[j]

pbest\_altitudes[j] = current\_altitude

# UPDATE GROUP'S BEST IF THIS HIKER FOUND A NEW HIGHEST POINT

if current\_altitude > gbest\_altitude: # Note: > for maximization

gbest\_position = positions[j]

gbest\_altitude = current\_altitude

# --- SHOW PROPER OUTPUT FOR THIS ITERATION ---

pos\_str = f"[{gbest\_position[0]:.4f}, {gbest\_position[1]:.4f}]"

print(f"{i+1:<12} {pos\_str:<25} {gbest\_altitude:<15.4f}")

# --- FINAL OUTPUT ---

print("\n" + "="\*55)

print("Search Complete!")

print(f"The highest peak found by the hikers is at: {gbest\_position}")

print(f"Maximum altitude reached: {gbest\_altitude}")

print(f"(The actual peak is at [2.0, 3.0] with an altitude of 100.0)")

Output:

