LAB-3

PARTICLE SWARM OPTIMIZATION

Code:

import numpy as np

class Particle:

def \_\_init\_\_(self, dim, bounds):

self.position = np.random.uniform(bounds[0], bounds[1], dim)

self.velocity = np.random.uniform(-1, 1, dim)

self.best\_position = self.position.copy()

self.best\_value = float('inf')

def update\_velocity(self, global\_best, w=0.5, c1=1.5, c2=1.5):

r1 = np.random.rand(len(self.position))

r2 = np.random.rand(len(self.position))

cognitive = c1 \* r1 \* (self.best\_position - self.position)

social = c2 \* r2 \* (global\_best - self.position)

self.velocity = w \* self.velocity + cognitive + social

def update\_position(self, bounds):

self.position += self.velocity

# Keep within bounds

self.position = np.clip(self.position, bounds[0], bounds[1])

def objective\_function(x):

# Example: Sphere function (minimize sum of squares)

return np.sum(x\*\*2)

def particle\_swarm\_optimization(dim, bounds, num\_particles=30, max\_iter=100):

swarm = [Particle(dim, bounds) for \_ in range(num\_particles)]

global\_best\_position = np.random.uniform(bounds[0], bounds[1], dim)

global\_best\_value = float('inf')

for iteration in range(max\_iter):

for particle in swarm:

fitness = objective\_function(particle.position)

# Update personal best

if fitness < particle.best\_value:

particle.best\_value = fitness

particle.best\_position = particle.position.copy()

# Update global best

if fitness < global\_best\_value:

global\_best\_value = fitness

global\_best\_position = particle.position.copy()

for particle in swarm:

particle.update\_velocity(global\_best\_position)

particle.update\_position(bounds)

print(f"Iteration {iteration+1}/{max\_iter}, Global Best Value: {global\_best\_value}")

return global\_best\_position, global\_best\_value

if \_\_name\_\_ == "\_\_main\_\_":

dim = 2

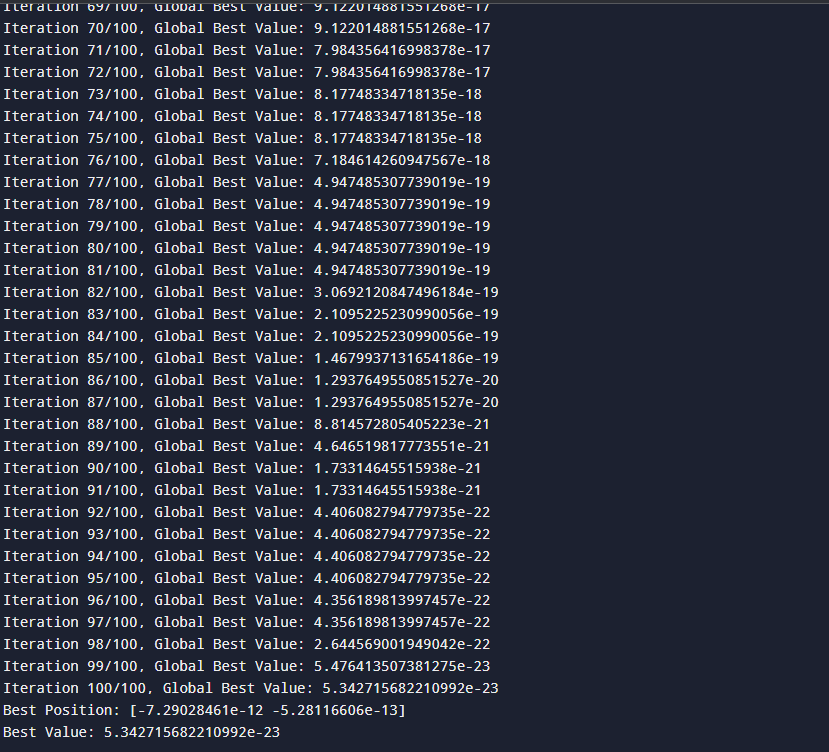
bounds = (-10, 10)

best\_pos, best\_val = particle\_swarm\_optimization(dim, bounds)

print(f"Best Position: {best\_pos}")

print(f"Best Value: {best\_val}")

Output:



ANT COLONY OPTIMIZATION

Code:

import numpy as np

import random

class AntColony:

def \_\_init\_\_(self, distances, n\_ants, n\_best, n\_iterations, decay, alpha=1, beta=1):

self.distances = distances

self.pheromone = np.ones(self.distances.shape) / len(distances)

self.all\_inds = range(len(distances))

self.n\_ants = n\_ants

self.n\_best = n\_best

self.n\_iterations = n\_iterations

self.decay = decay

self.alpha = alpha

self.beta = beta

def run(self):

shortest\_path = None

all\_time\_shortest\_path = ("placeholder", float('inf'))

for i in range(self.n\_iterations):

all\_paths = self.construct\_all\_paths()

self.spread\_pheromone(all\_paths, self.n\_best, shortest\_path=shortest\_path)

shortest\_path = min(all\_paths, key=lambda x: x[1])

if shortest\_path[1] < all\_time\_shortest\_path[1]:

all\_time\_shortest\_path = shortest\_path

self.pheromone \*= self.decay

print(f"Iteration {i+1}/{self.n\_iterations}, Best Path: {all\_time\_shortest\_path[1]}")

return all\_time\_shortest\_path

def construct\_all\_paths(self):

all\_paths = []

for \_ in range(self.n\_ants):

path = self.construct\_path(0) # start at node 0

all\_paths.append((path, self.path\_distance(path)))

return all\_paths

def construct\_path(self, start):

path = []

visited = set()

visited.add(start)

path.append(start)

prev = start

for \_ in range(len(self.distances) - 1):

move = self.pick\_move(self.pheromone[prev], self.distances[prev], visited)

path.append(move)

visited.add(move)

prev = move

return path

def pick\_move(self, pheromone, dist, visited):

pheromone = np.copy(pheromone)

pheromone[list(visited)] = 0

row = pheromone \*\* self.alpha \* ((1.0 / dist) \*\* self.beta)

norm\_row = row / row.sum()

move = np.random.choice(self.all\_inds, 1, p=norm\_row)[0]

return move

def path\_distance(self, path):

total = 0

for i in range(len(path)):

total += self.distances[path[i % len(path)]][path[(i + 1) % len(path)]]

return total

def spread\_pheromone(self, all\_paths, n\_best, shortest\_path):

sorted\_paths = sorted(all\_paths, key=lambda x: x[1])

for path, dist in sorted\_paths[:n\_best]:

for move in range(len(path)):

from\_city = path[move % len(path)]

to\_city = path[(move + 1) % len(path)]

self.pheromone[from\_city][to\_city] += 1.0 / self.distances[from\_city][to\_city]

self.pheromone[to\_city][from\_city] += 1.0 / self.distances[to\_city][from\_city]

if \_\_name\_\_ == "\_\_main\_\_":

distances = np.array([

[np.inf, 2, 2, 5, 7],

[2, np.inf, 4, 8, 2],

[2, 4, np.inf, 1, 3],

[5, 8, 1, np.inf, 2],

[7, 2, 3, 2, np.inf]

])

ant\_colony = AntColony(distances, n\_ants=10, n\_best=3, n\_iterations=100, decay=0.95, alpha=1, beta=2)

shortest\_path, distance = ant\_colony.run()

print("Best path found:", shortest\_path)

print("Path distance:", distance)

Output:

