import numpy as np

import math # Import the standard math module

def levy\_flight(Lambda):

sigma = (math.gamma(1 + Lambda) \* math.sin(math.pi \* Lambda / 2) /

(math.gamma((1 + Lambda) / 2) \* Lambda \* 2 \*\* ((Lambda - 1) / 2))) \*\* (1 / Lambda)

u = np.random.normal(0, sigma, 1)

v = np.random.normal(0, 1, 1)

step = u / abs(v) \*\* (1 / Lambda)

return step

def cuckoo\_search(obj\_function, bounds, n=25, pa=0.25, max\_iter=100):

# Initialize nests

dim = len(bounds)

nests = np.random.rand(n, dim)

for i in range(dim):

nests[:, i] = nests[:, i] \* (bounds[i][1] - bounds[i][0]) + bounds[i][0]

fitness = np.array([obj\_function(nest) for nest in nests])

# Start optimization

for \_ in range(max\_iter):

for i in range(n):

# Generate a new solution via Levy flight

new\_nest = nests[i] + levy\_flight(1.5) \* np.random.randn(dim)

# Apply bounds

new\_nest = np.clip(new\_nest, [b[0] for b in bounds], [b[1] for b in bounds])

new\_fitness = obj\_function(new\_nest)

# Update if new solution is better

if new\_fitness < fitness[i]:

nests[i] = new\_nest

fitness[i] = new\_fitness

# Abandon some nests and create new ones

abandon\_idx = np.random.rand(n) < pa

for i in np.where(abandon\_idx)[0]:

nests[i] = np.random.rand(dim) \* (np.array([b[1] for b in bounds]) - np.array([b[0] for b in bounds])) + np.array([b[0] for b in bounds])

fitness[i] = obj\_function(nests[i])

# Return the best solution

best\_idx = np.argmin(fitness)

return nests[best\_idx], fitness[best\_idx]

# Example usage: Minimize f(x) = x^2

def objective(x):

return sum(xi\*\*2 for xi in x)

bounds = [(-10, 10), (-10, 10)] # 2D problem

best\_solution, best\_fitness = cuckoo\_search(objective, bounds)

print("Best Solution:", best\_solution)

print("Best Fitness:", best\_fitness)

OUTPUT:

