Cuckoo Search  
import numpy as np  
import math  
  
# Objective Function (example: minimize Sphere function)  
def objective\_function(x):  
    return np.sum(x\*\*2)  
  
# Lévy flight function  
def levy\_flight(Lambda):  
    # Using math module instead of np.math  
    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.randn() \* sigma  
    v = np.random.randn()  
    step = u / abs(v) \*\* (1 / Lambda)  
    return step  
  
# Cuckoo Search Algorithm  
def cuckoo\_search(obj\_func, dim=2, n=15, pa=0.25, alpha=0.01, max\_iter=100):  
    # Initialize nests randomly  
    nests = np.random.uniform(-5, 5, size=(n, dim))  
    fitness = np.array([obj\_func(x) for x in nests])  
     
    # Find the current best nest  
    best\_idx = np.argmin(fitness)  
    best = nests[best\_idx].copy()  
     
    for t in range(max\_iter):  
        for i in range(n):  
            # Generate a new solution via Lévy flight  
            step\_size = alpha \* levy\_flight(1.5)  
            new\_nest = nests[i] + step\_size \* (nests[i] - best)  
            new\_nest = np.clip(new\_nest, -5, 5)  
             
            # Evaluate fitness  
            f\_new = obj\_func(new\_nest)  
             
            # Replace if the new solution is better  
            if f\_new < fitness[i]:  
                nests[i] = new\_nest  
                fitness[i] = f\_new  
                 
        # Abandon some nests (with probability pa)  
        for i in range(n):  
            if np.random.rand() < pa:  
                nests[i] = np.random.uniform(-5, 5, dim)  
                fitness[i] = obj\_func(nests[i])  
         
        # Update the best solution  
        best\_idx = np.argmin(fitness)  
        if fitness[best\_idx] < obj\_func(best):  
            best = nests[best\_idx].copy()  
         
        # Print progress  
        print(f"Iteration {t+1}: Best fitness = {obj\_func(best):.6f}")  
     
    return best, obj\_func(best)  
  
# Run the algorithm  
best\_solution, best\_value = cuckoo\_search(objective\_function, dim=2, n=20, max\_iter=50)  
print("\nBest Solution:", best\_solution)  
print("Best Fitness Value:", best\_value)

Output:

