BIS LAB

LAB 5

Cuckoo Search (CS):

Cuckoo Search (CS) is a nature-inspired optimization algorithm based on the brood parasitism of some cuckoo species. This behavior involves laying eggs in the nests of other birds, leading to the optimization of survival strategies. CS uses Lévy flights to generate new solutions, promoting global search capabilities and avoiding local minima. The algorithm is widely used for solving continuous optimization problems and has applications in various domains, including engineering design, machine learning, and data mining.

PYTHON CODE:

Below is a **complete, clean Python implementation** of the **Cuckoo Search Algorithm** to **find the maximum value** of a mathematical function (you can easily adapt it for minimization).

We'll use:

f(x)=xsin⁡(10πx)+1.0,x∈[0,1]f(x) = x \sin(10\pi x) + 1.0, \quad x \in [0, 1]f(x)=xsin(10πx)+1.0,x∈[0,1]

```
import numpy as np
import random
import math

# 1. Define the Problem (Objective Function)
def objective_function(x):
    """Example function to maximize."""
    return x * math.sin(10 * math.pi * x) + 1.0


# 2. Initialize Parameters
num_nests = 20            # Number of nests (population)
pa = 0.25                 # Discovery probability (fraction of nests
abandoned)
max_iter = 50             # Number of iterations
x_min, x_max = 0, 1       # Search space boundaries

# Lévy flight parameters
beta = 1.5   # Lévy exponent

# 3. Initialize Population
nests = np.random.uniform(x_min, x_max, num_nests)

# 4. Evaluate Fitness
```

```
fitness = np.array([objective_function(x) for x in nests])
best_nest = nests[np.argmax(fitness)]
best_fitness = max(fitness)

# Helper: Lévy flight step
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)
    v = np.random.normal(0, 1)
    step = u / abs(v) ** (1 / Lambda)
    return step

# 5-7. Main Loop
for iteration in range(max_iter):
    # Generate new solutions via Lévy flights
    new_nests = np.copy(nests)
    for i in range(num_nests):
        step_size = 0.01 * levy_flight(beta)
        new_nests[i] = nests[i] + step_size * np.random.randn()
        new_nests[i] = np.clip(new_nests[i], x_min, x_max)

    # Evaluate fitness of new solutions
    new_fitness = np.array([objective_function(x) for x in new_nests])

    # Replace nests if new solutions are better
    for i in range(num_nests):
        if new_fitness[i] > fitness[i]:
            fitness[i] = new_fitness[i]
            nests[i] = new_nests[i]

    # Abandon worst nests with probability pa
    abandon = np.random.rand(num_nests) < pa
    nests[abandon] = np.random.uniform(x_min, x_max, np.sum(abandon))
    fitness[abandon] = [objective_function(x) for x in nests[abandon]]

    # Update global best
    current_best = nests[np.argmax(fitness)]
    current_best_fit = max(fitness)
    if current_best_fit > best_fitness:
        best_fitness = current_best_fit
        best_nest = current_best

    print(f"Iteration {iteration + 1} | Best Fitness:
{best_fitness:.5f}")

# 8. Output the Best Solution
```

```
print("\n Best Solution Found:")
print(f"x = {best_nest:.5f}")
print(f"Fitness = {best_fitness:.5f}")
```

```
Iteration 1 | Best Fitness: 1.84947
Iteration 2 | Best Fitness: 1.85056
Iteration 3 | Best Fitness: 1.85056
Iteration 4 | Best Fitness: 1.85059
Iteration 5 | Best Fitness: 1.85059
Iteration 6 | Best Fitness: 1.85059
Iteration 7 | Best Fitness: 1.85059
Iteration 8 | Best Fitness: 1.85059
Iteration 9 | Best Fitness: 1.85059
Iteration 10 | Best Fitness: 1.85059
Iteration 11 | Best Fitness: 1.85059
Iteration 12 | Best Fitness: 1.85059
Iteration 13 | Best Fitness: 1.85059
Iteration 14 | Best Fitness: 1.85059
Iteration 15 | Best Fitness: 1.85059
Iteration 16 | Best Fitness: 1.85059
Iteration 17 | Best Fitness: 1.85059
Iteration 18 | Best Fitness: 1.85059
Iteration 19 | Best Fitness: 1.85059
Iteration 20 | Best Fitness: 1.85059
Iteration 21 | Best Fitness: 1.85059
Iteration 22 | Best Fitness: 1.85059
Iteration 23 | Best Fitness: 1.85059
Iteration 24 | Best Fitness: 1.85059
Iteration 25 | Best Fitness: 1.85059
Iteration 26 | Best Fitness: 1.85059
Iteration 27 | Best Fitness: 1.85059
Iteration 28 | Best Fitness: 1.85059
Iteration 29 | Best Fitness: 1.85059
Iteration 30 | Best Fitness: 1.85059
Iteration 31 | Best Fitness: 1.85059
Iteration 32 | Best Fitness: 1.85059
Iteration 33 | Best Fitness: 1.85059
Iteration 34 | Best Fitness: 1.85059
Iteration 35 | Best Fitness: 1.85059
Iteration 36 | Best Fitness: 1.85059
Iteration 37 | Best Fitness: 1.85059
Iteration 38 | Best Fitness: 1.85059
Iteration 39 | Best Fitness: 1.85059
Iteration 40 | Best Fitness: 1.85059
Iteration 41 | Best Fitness: 1.85059
Iteration 42 | Best Fitness: 1.85059
Iteration 43 | Best Fitness: 1.85059
Iteration 44 | Best Fitness: 1.85059
Iteration 45 | Best Fitness: 1.85059
Iteration 46 | Best Fitness: 1.85059
Iteration 47 | Best Fitness: 1.85059
Iteration 48 | Best Fitness: 1.85059
Iteration 49 | Best Fitness: 1.85059
Iteration 50 | Best Fitness: 1.85059

🥚 Best Solution Found:
x = 0.85113
Fitness = 1.85059
```