## **Lab-4-**Cuckoo Search (CS)

# Code:

```
import numpy as np
import random

# Step 1: Define the Problem (Optimization Function)
def objective_function(x):
    # Example: Sphere function (minimization problem)
    return sum(x**2)

# Step 2: Initialize Parameters
def cuckoo_search(num_nests, max_iter, pa, dim, lower_bound, upper_bound):
    # Initialize nests randomly within the bounds
    nests = np.random.uniform(low=lower_bound, high=upper_bound, size=(num_nests, dim))

    # Evaluate fitness of each nest
    fitness = np.apply_along_axis(objective_function, 1, nests)

    # Track the best solution found
    best_fitness = np.min(fitness)
    best_nest = nests[np.argmin(fitness)]

    # Step 3: Iterate the process
    for iter in range(max_iter):
        # Generate new solutions using Lévy flights
        new_nests = np.copy(nests)
        for i in range(num_nests):
            # Lévy flight (random walk)
            step_size = np.random.normal(0, 1, dim) * (np.abs(np.random.normal(0, 1, dim)) **
(1/2))
            new_nests[i] = nests[i] + step_size

            # Boundary check (Keep the nest within bounds)
            new_nests[i] = np.clip(new_nests[i], lower_bound, upper_bound)

        # Evaluate fitness of new nests
        new_fitness = np.apply_along_axis(objective_function, 1, new_nests)
```

```
        # Step 4: Abandon Worst Nests and Replace
        for i in range(num_nests):
            if new_fitness[i] < fitness[i]:  # If new nest is better
                nests[i] = new_nests[i]
                fitness[i] = new_fitness[i]

        # Discovering worst nests and abandon them with probability pa
        if random.random() < pa:
            abandon_indices = np.argsort(fitness)[:int(num_nests * 0.25)]  # abandon worst 25% nests
            nests[abandon_indices] = np.random.uniform(low=lower_bound, high=upper_bound, size=(len(abandon_indices), dim))
            fitness[abandon_indices] = np.apply_along_axis(objective_function, 1, nests[abandon_indices])

        # Track best solution so far
        current_best_fitness = np.min(fitness)
        current_best_nest = nests[np.argmin(fitness)]

        if current_best_fitness < best_fitness:
            best_fitness = current_best_fitness
            best_nest = current_best_nest

        # Output the current best solution (optional)
        print(f"Iteration {iter+1}/{max_iter}, Best Fitness: {best_fitness}")

    return best_nest, best_fitness

# Step 5: User Input for Parameters
if __name__ == "__main__":
    # User input for parameters
    num_nests = int(input("Enter number of nests: "))
    max_iter = int(input("Enter number of iterations: "))
    pa = float(input("Enter probability of discovery (pa) [0, 1]: "))
    dim = int(input("Enter the number of dimensions: "))
    lower_bound = float(input("Enter lower bound for search space: "))
    upper_bound = float(input("Enter upper bound for search space: "))

    # Run the Cuckoo Search Algorithm
```

```
best_solution, best_fitness = cuckoo_search(num_nests, max_iter, pa, dim, lower_bound, upper_bound)

# Output the best solution found
print(f"\nBest Solution: {best_solution}")
print(f"Best Fitness: {best_fitness}")
```

## Output:

```
Enter number of nests: 20
Enter number of iterations: 8
Enter probability of discovery (pa) [0, 1]: 0.25
Enter the number of dimensions: 2
Enter lower bound for search space: -5
Enter upper bound for search space: 5
Iteration 1/8, Best Fitness: 0.027458587410925876
Iteration 2/8, Best Fitness: 0.027458587410925876
Iteration 3/8, Best Fitness: 0.027458587410925876
Iteration 4/8, Best Fitness: 0.027458587410925876
Iteration 5/8, Best Fitness: 0.027458587410925876
Iteration 6/8, Best Fitness: 0.027458587410925876
Iteration 7/8, Best Fitness: 0.027458587410925876
Iteration 8/8, Best Fitness: 0.027458587410925876

Best Solution: [ 1.85421997 -3.93906311]
Best Fitness: 0.027458587410925876
```