4 Cuckoo search algorithm.

Algorithm.

Input:
n: Number of host nests (population)
Pa: Fraction of worst nests to be abandoned
MaxIterations: Maximum number of iterations
f(x): Objective function to minimize
Dimension: dimensionality of the problem
Bounds: Lower & upper limits of the search space.

Initialize:
1) Generate an initial population of n random host nests x_i (for $i=1, 2, \ldots, n$)
2) Evaluate the fitness $f(x_i)$ for each nest
3) Determine the current best solution x^* with the best fitness $f(x^*)$

while the current iteration $t <$ MaxIterations:
step 1: perform levy flight for randomly selected nest
select a random nest x_i
Generate a new solution x' using levy flight:

$$x' = x_i + \alpha L(x_i - x^*)$$

where α is the levy flight step, α is the step size.
Clip x' within bounds, if necessary

Step 2 Evaluate the fitness $f(x')$ of the new solution.
If $f(x') < f(x_j)$ replace a randomly chosen nest x_j with x'.
Step 3: Abandon worse nests.
Identify P_a worst nests & replace them with new random solutions.
Step 4: Update the current best solution.
Identify & retain the nest with best fitness

code:

```
import numpy as np

import math

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] - bound[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 sol via Levy flight
        new_nest = nest[i] + levy_flight(15) *
            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 nest & create new
```

```
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 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)]
best_solution, best_fitness = cuckoo_search
(objective, bounds)
print("Best solution:", best_solution)
print("Best Fitness:", best_fitness)
```

→ output:

Best Solution: [-1.05365458 -1.65091143]

Best Fitness: 2.514629684039968e-15