import numpy as np

from scipy.special import gamma

# Objective Function (for example, Sphere function)

def objective(x):

return np.sum(x\*\*2)

# Lévy flight function

def levy\_flight(beta, dim):

sigma = (gamma(1+beta)\*np.sin(np.pi\*beta/2) /

(gamma((1+beta)/2)\*beta\*np.power(2, (beta-1)/2)))\*\*(1/beta)

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

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

return u / np.abs(v)\*\*(1/beta)

# Cuckoo Search algorithm

def cuckoo\_search(obj\_func, dim, bounds, N=20, pa=0.25, max\_iter=100):

nests = np.random.uniform(bounds[0], bounds[1], (N, dim))

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

best\_nest = nests[np.argmin(fitness)]

best\_fitness = np.min(fitness)

for \_ in range(max\_iter):

# Generate new solutions using Lévy flights

new\_nests = np.copy(nests)

for i in range(N):

step = levy\_flight(1.5, dim) # Lévy exponent 1.5

new\_nests[i] = nests[i] + 0.01 \* step

new\_nests[i] = np.clip(new\_nests[i], bounds[0], bounds[1]) # Bound the new nest position

# Evaluate new solutions

new\_fitness = np.array([obj\_func(nest) for nest in new\_nests])

# Abandon worst nests and replace with new ones

for i in range(N):

if np.random.rand() < pa and new\_fitness[i] < fitness[i]:

nests[i] = new\_nests[i]

fitness[i] = new\_fitness[i]

# Update the best solution

best\_nest\_idx = np.argmin(fitness)

best\_nest = nests[best\_nest\_idx]

best\_fitness = fitness[best\_nest\_idx]

return best\_nest, best\_fitness

# Parameters

dim = 10

bounds = [-5, 5]

best\_nest, best\_fitness = cuckoo\_search(objective, dim, bounds)

print(f"Best Nest: {best\_nest}")

print(f"Best Fitness: {best\_fitness}")