import numpy as np

# Objective function for 1D (x^2)

def objective\_function\_1d(x):

return x[0]\*\*2 # x is a 1D array, even though we just care about the first element

# Lévy Flight to generate new solutions

def levy\_flight(num\_dim, beta=1.5):

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

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

u = np.random.normal(0, sigma\_u, num\_dim) # Lévy-distributed steps

v = np.random.normal(0, 1, num\_dim)

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

# Cuckoo Search Algorithm for 1D

def cuckoo\_search\_1d(num\_iterations, num\_nests, pa=0.25):

num\_dim = 1 # 1D problem

nests = np.random.rand(num\_nests, num\_dim) \* 10 - 5 # Random initialization within [-5, 5]

fitness = np.apply\_along\_axis(objective\_function\_1d, 1, nests) # Evaluate initial fitness

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

best\_fitness = np.min(fitness)

for \_ in range(num\_iterations):

for i in range(num\_nests):

new\_nest = nests[i] + levy\_flight(num\_dim) # Generate new solution using Lévy flight

new\_fitness = objective\_function\_1d(new\_nest)

if new\_fitness < fitness[i]: # Replace if new solution is better

nests[i] = new\_nest

fitness[i] = new\_fitness

# Abandon the worst nests

worst\_nests = np.argsort(fitness)[-int(pa \* num\_nests):]

for j in worst\_nests:

nests[j] = np.random.rand(num\_dim) \* 10 - 5 # Randomly initialize new nests

fitness[j] = objective\_function\_1d(nests[j])

# Update best solution found so far

current\_best\_idx = np.argmin(fitness)

current\_best\_fitness = fitness[current\_best\_idx]

if current\_best\_fitness < best\_fitness:

best\_fitness = current\_best\_fitness

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

return best\_nest, best\_fitness # Return the best solution and its fitness

# Run the cuckoo search on the 1D problem

best\_solution, best\_fitness = cuckoo\_search\_1d(num\_iterations=1000, num\_nests=25)

print(f"Best solution found: {best\_solution} with objective value: {best\_fitness}")

OUTPUT:

