CUCKOO SEARCH ALGORITH BY ENGINEERING DESIGN

import numpy as np

# Define the objective function: A simplified "drag function" that we aim to minimize

def drag\_function(x):

    # x[0]: curvature, x[1]: width, x[2]: slope

    # A hypothetical drag equation (for demonstration purposes)

    return x[0]\*\*2 + 2 \* x[1]\*\*2 + 3 \* x[2]\*\*2 + 4 \* x[0] \* x[1] - 2 \* x[1] \* x[2]

# Lévy flight function using numpy for Gamma and other computations

def gamma\_function(x):

    if x == 0.5:

        return np.sqrt(np.pi)  # Special case for gamma(1/2)

    elif x == 1:

        return 1  # Special case for gamma(1)

    elif x == 2:

        return 1  # Special case for gamma(2)

    else:

        return np.math.factorial(int(x) - 1) if x.is\_integer() else np.inf

def levy\_flight(Lambda):

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

             (gamma\_function((1 + Lambda) / 2) \* Lambda \* 2 \*\* ((Lambda - 1) / 2))) \*\* (1 / Lambda)

    u = np.random.randn() \* sigma

    v = np.random.randn()

    step = u / abs(v) \*\* (1 / Lambda)

    return step

# Cuckoo Search Algorithm

def cuckoo\_search(n, iterations, pa, lower\_bound, upper\_bound):

    # Initialize nests randomly

    dim = 3  # Number of design parameters

    nests = np.random.uniform(lower\_bound, upper\_bound, (n, dim))

    # Evaluate fitness of initial nests

    fitness = np.array([drag\_function(nest) for nest in nests])

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

    best\_fitness = min(fitness)

    # Cuckoo Search main loop

    for \_ in range(iterations):

        for i in range(n):

            # Generate a new solution by Lévy flight

            step\_size = levy\_flight(1.5)

            new\_nest = nests[i] + step\_size \* np.random.uniform(-1, 1, dim)

            new\_nest = np.clip(new\_nest, lower\_bound, upper\_bound)  # Ensure within bounds

            new\_fitness = drag\_function(new\_nest)

            # Replace nest if the new solution is better

            if new\_fitness < fitness[i]:

                nests[i] = new\_nest

                fitness[i] = new\_fitness

        # Abandon a fraction of the worst nests and create new ones

        for i in range(int(pa \* n)):

            nests[-(i + 1)] = np.random.uniform(lower\_bound, upper\_bound, dim)

            fitness[-(i + 1)] = drag\_function(nests[-(i + 1)])

        # Update the best nest

        if min(fitness) < best\_fitness:

            best\_fitness = min(fitness)

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

    return best\_nest, best\_fitness

# Gather user input for the algorithm

print("Welcome to the Aerodynamics Optimization using Cuckoo Search!")

n = int(input("Enter the number of nests (population size): "))

iterations = int(input("Enter the number of iterations: "))

pa = float(input("Enter the probability of abandonment (between 0 and 1): "))

lower\_bound = float(input("Enter the lower bound for the design parameters: "))

upper\_bound = float(input("Enter the upper bound for the design parameters: "))

# Run the Cuckoo Search algorithm

best\_solution, best\_drag\_value = cuckoo\_search(n, iterations, pa, lower\_bound, upper\_bound)

# Display the result

print("\nOptimization Results:")

print("Best Solution (Design Parameters):", best\_solution)

print("Best Drag Value:", best\_drag\_value)

OUTPUT:

