LAB 4

Cuckoo Search (CS)

Application:

Portfolio Optimization using the Cuckoo Search algorithm. Aim is to find the optimal allocation of capital among different stocks to maximize return while minimizing risk. The objective is to balance both the expected returns and the risk (standard deviation) of the portfolio.

Code:

```
import numpy as np
import matplotlib.pyplot as plt

# Cuckoo Search Algorithm
class CuckooSearch:
    def __init__(self, n_cuckoos, n_iterations, alpha=1, beta=1, lambda_=0.1):
        self.n_cuckoos = n_cuckoos
        self.n_iterations = n_iterations
        self.alpha = alpha
        self.beta = beta
        self.lambda_ = lambda_

    def fitness(self, portfolio, returns, cov_matrix):
        portfolio_return = np.sum(portfolio * returns)
        portfolio_risk = np.sqrt(np.dot(portfolio.T, np.dot(cov_matrix, portfolio)))
        return portfolio_return - portfolio_risk

    def levy_flight(self, n):
        return np.random.normal(0, 1, n) * np.random.normal(0, 1, n)

    def optimize(self, returns, cov_matrix, bounds):
        n_assets = len(returns)
        population = np.random.rand(self.n_cuckoos, n_assets)
        population = population / population.sum(axis=1)[:, np.newaxis]

        best_fitness = -np.inf
        best_solution = None
        all_fitness = []
        all_risk = []
        all_return = []

        for _ in range(self.n_iterations):
            for i in range(self.n_cuckoos):
                fitness_value = self.fitness(population[i], returns, cov_matrix)
                all_fitness.append(fitness_value)
                portfolio_return = np.sum(population[i] * returns)
                portfolio_risk = np.sqrt(np.dot(population[i].T, np.dot(cov_matrix,
population[i])))
                all_risk.append(portfolio_risk)
                all_return.append(portfolio_return)

                if fitness_value > best_fitness:
```

```
                    best_fitness = fitness_value
                    best_solution = population[i]

            for i in range(self.n_cuckoos):
                new_solution = population[i] + self.alpha * self.levy_flight(n_assets)
                new_solution = np.clip(new_solution, 0, 1)
                new_solution = new_solution / new_solution.sum()
                new_fitness = self.fitness(new_solution, returns, cov_matrix)

                if new_fitness > self.fitness(population[i], returns, cov_matrix):
                    population[i] = new_solution

        return best_solution, best_fitness, all_fitness, all_risk, all_return

# User Input for Portfolio Optimization
print("Portfolio Optimization using Cuckoo Search")
n_assets = int(input("Enter the number of assets: "))

# Get returns for each asset
returns = []
print("\nEnter the expected returns for each asset:")
for i in range(n_assets):
    ret = float(input(f"Asset {i + 1} return: "))
    returns.append(ret)
returns = np.array(returns)

# Get covariance matrix
print("\nEnter the covariance matrix row by row:")
cov_matrix = []
for i in range(n_assets):
    row = list(map(float, input(f"Row {i + 1} (space-separated): ").split()))
    cov_matrix.append(row)
cov_matrix = np.array(cov_matrix)

# Cuckoo Search Parameters
n_cuckoos = int(input("\nEnter the number of cuckoos (population size): "))
n_iterations = int(input("Enter the number of iterations: "))
alpha = float(input("Enter the step size (alpha): "))

# Initialize and optimize using Cuckoo Search
cuckoo_search = CuckooSearch(n_cuckoos=n_cuckoos, n_iterations=n_iterations, alpha=alpha)
best_portfolio, best_fitness, all_fitness, all_risk, all_return =
cuckoo_search.optimize(returns, cov_matrix, bounds=None)

# Output the results
print("\nOptimal Portfolio Allocation:", best_portfolio)
print("Best Fitness (Return - Risk):", best_fitness)

# Plot the results
plt.figure(figsize=(8, 6))
plt.scatter(all_risk, all_return, c=all_fitness, cmap='viridis', label="Fitness of
Solutions")
plt.colorbar(label="Fitness (Return - Risk)")
plt.xlabel("Risk (Standard Deviation)")
plt.ylabel("Return")
plt.title("Portfolio Optimization: Risk vs Return")
```

```
plt.legend()
plt.grid(True)
plt.show()
```

Output:

```
Portfolio Optimization using Cuckoo Search
Enter the number of assets: 3

Enter the expected returns for each asset:
Asset 1 return: 0.12
Asset 2 return: 0.18
Asset 3 return: 0.15

Enter the covariance matrix row by row:
Row 1 (space-separated): 0.1 0.03 0.05
Row 2 (space-separated): 0.03 0.12  0.06
Row 3 (space-separated): 0.05 0.06 0.15

Enter the number of cuckoos (population size): 30
Enter the number of iterations: 100
Enter the step size (alpha): 0.01

Optimal Portfolio Allocation: [0.3955319  0.46829795 0.13617014]
Best Fitness (Return - Risk): -0.11029369554813326
```