Lab 4 – Cuckoo Search:

```
import numpy as np

def objective_function(x):
    """Example objective function: Sphere function."""
    return sum(x_i**2 for x_i in x)

def levy_flight(Lambda):
    """Generate step sizes based on Lévy distribution."""
    sigma = (np.math.gamma(1 + Lambda) * np.sin(np.pi * Lambda / 2) /
             (np.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_func, bounds, n_nests=25, p_a=0.25, iterations=100):
    """Cuckoo Search Optimization Algorithm."""
    num_dimensions = len(bounds)

    # Initialize nests with random positions
    nests = np.random.uniform(bounds[:, 0], bounds[:, 1], size=(n_nests, num_dimensions))
    fitness = np.array([obj_func(nest) for nest in nests])

    best_nest = nests[np.argmin(fitness)]
    best_fitness = fitness.min()

    for _ in range(iterations):
        # Generate new solutions via Lévy flights
        new_nests = np.copy(nests)
        for i in range(n_nests):
```

```
            step = levy_flight(1.5)  # Lévy flight with λ=1.5
            new_nests[i] += step * (nests[i] - best_nest)
            # Enforce boundary conditions
            new_nests[i] = np.clip(new_nests[i], bounds[:, 0], bounds[:, 1])

        # Evaluate the new solutions
        new_fitness = np.array([obj_func(nest) for nest in new_nests])

        # Replace nests if the new solution is better
        for i in range(n_nests):
            if new_fitness[i] < fitness[i]:
                nests[i] = new_nests[i]
                fitness[i] = new_fitness[i]

        # Abandon a fraction of the worst nests and replace them
        for i in range(n_nests):
            if np.random.rand() < p_a:
                nests[i] = np.random.uniform(bounds[:, 0], bounds[:, 1], num_dimensions)
                fitness[i] = obj_func(nests[i])

        # Update the best nest
        current_best = np.argmin(fitness)
        if fitness[current_best] < best_fitness:
            best_nest = nests[current_best]
            best_fitness = fitness[current_best]

    return best_nest, best_fitness

# Example usage
if __name__ == "__main__":
    # Define the problem
```

```
bounds = np.array([[-10, 10], [-10, 10]])  # Search space bounds
best_solution, best_value = cuckoo_search(objective_function, bounds)
print("Best Solution:", best_solution)
print("Best Value:", best_value)
```

OUTPUT:

```
Best Solution: [-6.11142439  1.35222608]
Best Value: 0.0030363553011617 82
```