```python
import numpy as np
import random
import math
import matplotlib.pyplot as plt

# Define a sample function to optimize (Sphere function in this case)
def objective_function(x):
    return np.sum(x ** 2)

# Lévy flight function
def levy_flight(Lambda):
    sigma_u = (math.gamma(1 + Lambda) * np.sin(np.pi * Lambda / 2) /
               (math.gamma((1 + Lambda) / 2) * Lambda * 2 ** ((Lambda - 1) / 2))) ** (1 / Lambda)
    sigma_v = 1
    u = np.random.normal(0, sigma_u, size=1)
    v = np.random.normal(0, sigma_v, size=1)
    step = u / (abs(v) ** (1 / Lambda))
    return step

# Cuckoo Search algorithm
def cuckoo_search(num_nests=25, num_iterations=100, discovery_rate=0.25, dim=5, lower_bound=-10, upper_bound=10):
    # Initialize nests
    nests = np.random.uniform(lower_bound, upper_bound, (num_nests, dim))
    fitness = np.array([objective_function(nest) for nest in nests])

    # Get the current best nest
    best_nest_idx = np.argmin(fitness)
    best_nest = nests[best_nest_idx].copy()
    best_fitness = fitness[best_nest_idx]

    Lambda = 1.5  # Parameter for Lévy flights
    fitness_history = []  # To track fitness at each iteration

    for iteration in range(num_iterations):
        # Generate new solutions via Lévy flight
        for i in range(num_nests):
            step_size = levy_flight(Lambda)
            new_solution = nests[i] + step_size * (nests[i] - best_nest)
            new_solution = np.clip(new_solution, lower_bound, upper_bound)
            new_fitness = objective_function(new_solution)

            # Replace nest if new solution is better
            if new_fitness < fitness[i]:
                nests[i] = new_solution
                fitness[i] = new_fitness

        # Discover some nests with probability 'discovery_rate'
        random_nests = np.random.choice(num_nests, int(discovery_rate * num_nests), replace=False)
        for nest_idx in random_nests:
            nests[nest_idx] = np.random.uniform(lower_bound, upper_bound, dim)
            fitness[nest_idx] = objective_function(nests[nest_idx])

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

        # Store fitness for plotting
        fitness_history.append(best_fitness)

        # Print the best solution at each iteration (optional)
        print(f"Iteration {iteration+1}/{num_iterations}, Best Fitness: {best_fitness}")

    # Plot fitness convergence graph
    plt.plot(fitness_history)
    plt.title('Fitness Convergence Over Iterations')
```

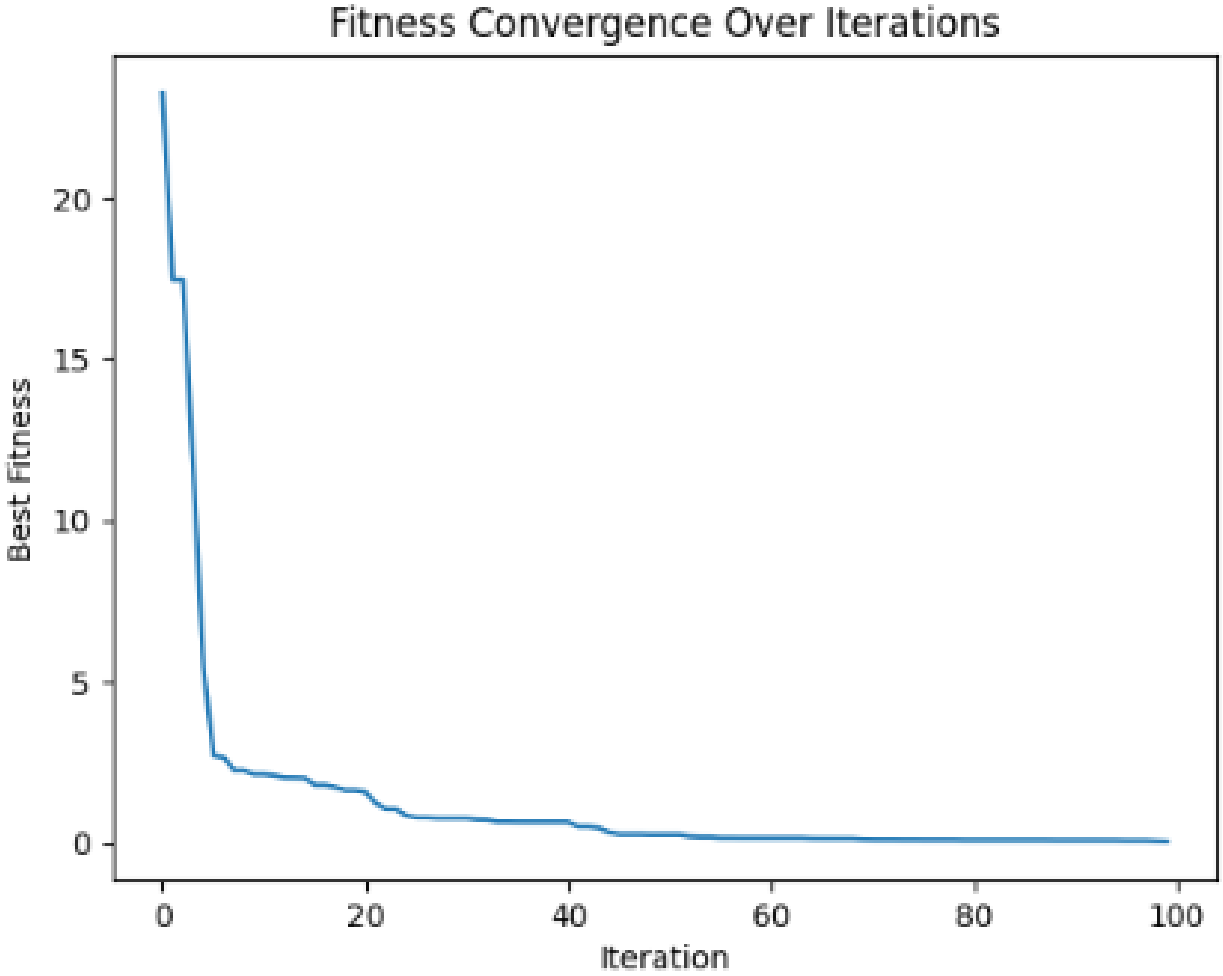
```
    plt.xlabel('Iteration')
    plt.ylabel('Best Fitness')
    plt.show()

    # Return the best solution found
    return best_nest, best_fitness

# Example usage
best_nest, best_fitness = cuckoo_search(num_nests=30, num_iterations=100, dim=10, lower_bound=-5, upper_bound=5)
print("Best Solution:", best_nest)
print("Best Fitness:", best_fitness)
```

```
Iteration 1/100, Best Fitness: 23.237909685786647
Iteration 2/100, Best Fitness: 17.44643763357293
Iteration 3/100, Best Fitness: 17.44643763357293
Iteration 4/100, Best Fitness: 11.620650773060179
Iteration 5/100, Best Fitness: 5.444003140912359
Iteration 6/100, Best Fitness: 2.7140863425513637
Iteration 7/100, Best Fitness: 2.6491997748622427
Iteration 8/100, Best Fitness: 2.2651205700923254
Iteration 9/100, Best Fitness: 2.2651205700923254
Iteration 10/100, Best Fitness: 2.1422032456182922
Iteration 11/100, Best Fitness: 2.1422032456182922
Iteration 12/100, Best Fitness: 2.1059528751434806
Iteration 13/100, Best Fitness: 2.0335240202843967
Iteration 14/100, Best Fitness: 2.0289344807734477
Iteration 15/100, Best Fitness: 2.0184359493168507
Iteration 16/100, Best Fitness: 1.8096424278417478
Iteration 17/100, Best Fitness: 1.8096424278417478
Iteration 18/100, Best Fitness: 1.7372918741983778
Iteration 19/100, Best Fitness: 1.647248420940563
Iteration 20/100, Best Fitness: 1.647248420940563
Iteration 21/100, Best Fitness: 1.596211943714965
Iteration 22/100, Best Fitness: 1.2610153234042785
Iteration 23/100, Best Fitness: 1.0708035345758535
Iteration 24/100, Best Fitness: 1.0670834679772039
Iteration 25/100, Best Fitness: 0.8629624035984536
Iteration 26/100, Best Fitness: 0.7931192407729026
Iteration 27/100, Best Fitness: 0.7931192407729026
Iteration 28/100, Best Fitness: 0.7716755790764325
Iteration 29/100, Best Fitness: 0.7716755790764325
Iteration 30/100, Best Fitness: 0.7716755790764325
Iteration 31/100, Best Fitness: 0.7716755790764325
Iteration 32/100, Best Fitness: 0.7347592138844211
Iteration 33/100, Best Fitness: 0.7265111048767167
Iteration 34/100, Best Fitness: 0.6564876013446722
Iteration 35/100, Best Fitness: 0.6564876013446722
Iteration 36/100, Best Fitness: 0.65125838830602
Iteration 37/100, Best Fitness: 0.6506764096850287
Iteration 38/100, Best Fitness: 0.6506764096850287
Iteration 39/100, Best Fitness: 0.6506764096850287
Iteration 40/100, Best Fitness: 0.6501119526352238
Iteration 41/100, Best Fitness: 0.6501119526352238
Iteration 42/100, Best Fitness: 0.503739775840174
Iteration 43/100, Best Fitness: 0.503739775840174
Iteration 44/100, Best Fitness: 0.4637919691373442
Iteration 45/100, Best Fitness: 0.32558045997256585
Iteration 46/100, Best Fitness: 0.2664508583902288
Iteration 47/100, Best Fitness: 0.2664508583902288
Iteration 48/100, Best Fitness: 0.2664508583902288
Iteration 49/100, Best Fitness: 0.2562059093121094
Iteration 50/100, Best Fitness: 0.2562059093121094
Iteration 51/100, Best Fitness: 0.2562059093121094
Iteration 52/100, Best Fitness: 0.2562059093121094
Iteration 53/100, Best Fitness: 0.1966670426778853
Iteration 54/100, Best Fitness: 0.17929468758238645
Iteration 55/100, Best Fitness: 0.17929468758238645
Iteration 56/100, Best Fitness: 0.15360284726695325
Iteration 57/100, Best Fitness: 0.15360284726695325
Iteration 58/100, Best Fitness: 0.15065623180446192
Iteration 59/100, Best Fitness: 0.15065623180446192
Iteration 60/100, Best Fitness: 0.1504898812236742
Iteration 61/100, Best Fitness: 0.1504898812236742
Iteration 62/100, Best Fitness: 0.1504898812236742
Iteration 63/100, Best Fitness: 0.1504898812236742
Iteration 64/100, Best Fitness: 0.1504898812236742
Iteration 65/100, Best Fitness: 0.13504209948389542
Iteration 66/100, Best Fitness: 0.13287979700203936
Iteration 67/100, Best Fitness: 0.13232153108311367
Iteration 68/100, Best Fitness: 0.13232153108311367
Iteration 69/100, Best Fitness: 0.13232153108311367
Iteration 70/100, Best Fitness: 0.10769776606356221
Iteration 71/100, Best Fitness: 0.10158913726968552
Iteration 72/100, Best Fitness: 0.09949026687883586
Iteration 73/100, Best Fitness: 0.09949026687883586
Iteration 74/100, Best Fitness: 0.0963335695261066
Iteration 75/100, Best Fitness: 0.0963335695261066
Iteration 76/100, Best Fitness: 0.09396161626495726
Iteration 77/100, Best Fitness: 0.09396161626495726
Iteration 78/100, Best Fitness: 0.09361523080569159
Iteration 79/100, Best Fitness: 0.09357203310092375
Iteration 80/100, Best Fitness: 0.08715794708315733
Iteration 81/100, Best Fitness: 0.08715794708315733
Iteration 82/100, Best Fitness: 0.08715794708315733
Iteration 83/100, Best Fitness: 0.08715794708315733
Iteration 84/100, Best Fitness: 0.08715794708315733
Iteration 85/100, Best Fitness: 0.08715794708315733
Iteration 86/100, Best Fitness: 0.08715794708315733
Iteration 87/100, Best Fitness: 0.08715794708315733
Iteration 88/100, Best Fitness: 0.08715794708315733
Iteration 89/100, Best Fitness: 0.082337778103759
Iteration 90/100, Best Fitness: 0.082337778103759
Iteration 91/100, Best Fitness: 0.082337778103759
Iteration 92/100, Best Fitness: 0.082337778103759
```

```
Iteration 93/100, Best Fitness: 0.082337778103759
Iteration 94/100, Best Fitness: 0.0808612356891892
Iteration 95/100, Best Fitness: 0.0808612356891892
Iteration 96/100, Best Fitness: 0.07345191398448075
Iteration 97/100, Best Fitness: 0.07345191398448075
Iteration 98/100, Best Fitness: 0.07345191398448075
Iteration 99/100, Best Fitness: 0.05734549573438352
Iteration 100/100, Best Fitness: 0.04483135325989235
```




```
Best Solution: [ 0.01266915  0.06289373  0.03687494 -0.05011045 -0.10876369  0.09418888
  0.01697123 -0.03496787  0.01475733  0.12006151]
Best Fitness: 0.04483135325989235
```