USN : 1BM22CS253 LAB-

4 : Cuckoo Search (CS):

CODE:

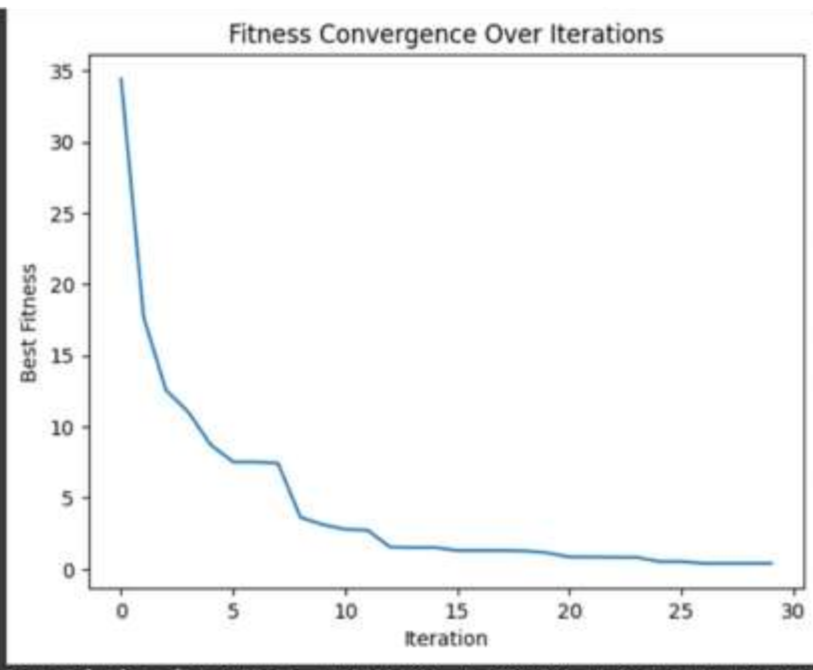
```
#cuckoo search import numpy
as np import random import
math import matplotlib.pyplot
as plt


# Define a sample func on to op mize (Sphere func on in this case) def
objec ve_func on(x):

    return np.sum(x ** 2)


# Lévy flight func on
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_itera ons=100, discovery_rate=0.25, dim=5,
lower_bound=10, upper_bound=10):

    # Ini alize nests     nests = np.random.uniform(lower_bound, upper_bound,
(num_nests, dim))     fitness = np.array([objec ve_func on(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 itera on

    for itera on in range(num_itera ons):         #
Generate new solu ons via Lévy flight        for
i in range(num_nests):

            step_size = levy_flight(Lambda)              new_solu on = nests[i] +
step_size * (nests[i] - best_nest)              new_solu on = np.clip(new_solu
on, lower_bound, upper_bound)                   new_fitness = objec ve_func
on(new_solu on)

            # Replace nest if new solu on is be er
if new_fitness < fitness[i]:                nests[i] =
new_solu on               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] = objec ve_func on(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()

        fitness_history.append(best_fitness)         print(f"Itera on {itera
on+1}/{num_itera ons}, Best Fitness: {best_fitness}")


    plt.plot(fitness_history)           plt.  tle('Fitness
Convergence Over Itera ons')      plt.xlabel('Itera
on')        plt.ylabel('Best  Fitness')        plt.show()
return best_nest, best_fitness


best_nest, best_fitness = cuckoo_search(num_nests=30, num_itera ons=100, dim=10,
lower_bound=-5, upper_bound=5) print("Best Solu on:", best_nest) print("Best
Fitness:", best_fitness)
```

OUTPUT:

```
Iteration 1/30, Best Fitness: 34.421347350368414
Iteration 2/30, Best Fitness: 17.701267864864427
Iteration 3/30, Best Fitness: 12.572246094152595
Iteration 4/30, Best Fitness: 11.025968548544025
Iteration 5/30, Best Fitness: 8.713786692960158
Iteration 6/30, Best Fitness: 7.5206125475077785
Iteration 7/30, Best Fitness: 7.5206125475077785
Iteration 8/30, Best Fitness: 7.426062303628502
Iteration 9/30, Best Fitness: 3.6305424687807872
Iteration 10/30, Best Fitness: 3.122312407680085
Iteration 11/30, Best Fitness: 2.7935374916676268
Iteration 12/30, Best Fitness: 2.7258275326189683
Iteration 13/30, Best Fitness: 1.5451154817432429
Iteration 14/30, Best Fitness: 1.5138101828809285
Iteration 15/30, Best Fitness: 1.5138101828809285
Iteration 16/30, Best Fitness: 1.300269684490209
Iteration 17/30, Best Fitness: 1.300269684490209
Iteration 18/30, Best Fitness: 1.300269684490209
Iteration 19/30, Best Fitness: 1.2738498249584989
Iteration 20/30, Best Fitness: 1.1445834652176474
Iteration 21/30, Best Fitness: 0.8487556087655604
Iteration 22/30, Best Fitness: 0.8487556087655604
Iteration 23/30, Best Fitness: 0.8289231635578032
Iteration 24/30, Best Fitness: 0.8242402471719793
Iteration 25/30, Best Fitness: 0.5258270013075049
Iteration 26/30, Best Fitness: 0.5258270013075049
Iteration 27/30, Best Fitness: 0.3996236442626478
Iteration 28/30, Best Fitness: 0.3996236442626478
Iteration 29/30, Best Fitness: 0.3996236442626478
Iteration 30/30, Best Fitness: 0.3996236442626478
```




```
Best Solution: [ 0.1187463  -0.27226584  0.30789726 -0.04631337 -0.41395998  0.0566571
 -0.05240855 -0.11176247  0.15695263 -0.00202369]
Best Fitness: 0.3996236442626478
```