BIS LAB

LAB 7

Parallel Cellular Algorithm for Function Optimization:

Use a Parallel Cellular Algorithm to optimize a function efficiently by evolving solutions in a distributed and parallel manner.

PYTHON CODE:

```
import numpy as np

# 1️⃣ Define the Objective Function (example)
def objective_function(x):
    # Example: Sphere function (minimize)
    return np.sum(x ** 2)

# 2️⃣ Cellular Automata Optimization
def cellular_automata_optimization(f, grid_size=10, max_iterations=50,
lb=-10, ub=10, dim=2, neighborhood=1):
    num_cells = grid_size * grid_size

    # Initialize grid of solutions (each cell = vector of dimension
`dim`)
    grid = np.random.uniform(lb, ub, (grid_size, grid_size, dim))

    # Evaluate fitness for each cell
    fitness = np.zeros((grid_size, grid_size))
    for i in range(grid_size):
        for j in range(grid_size):
            fitness[i, j] = f(grid[i, j])

    # Find best initial solution
    best_idx = np.unravel_index(np.argmin(fitness), fitness.shape)
    best_solution = grid[best_idx]
    best_fitness = fitness[best_idx]

    # Main loop
    for iteration in range(max_iterations):
        new_grid = np.copy(grid)

        # For each cell in grid
        for i in range(grid_size):
            for j in range(grid_size):
                # Find neighborhood indices
```

```
                i_min = max(0, i - neighborhood)
                i_max = min(grid_size, i + neighborhood + 1)
                j_min = max(0, j - neighborhood)
                j_max = min(grid_size, j + neighborhood + 1)

                # Extract neighbors and their fitness
                neighbors = grid[i_min:i_max, j_min:j_max, :]
                neighbor_fitness = fitness[i_min:i_max, j_min:j_max]

                # Best neighbor
                idx_best =
np.unravel_index(np.argmin(neighbor_fitness), neighbor_fitness.shape)
                best_neighbor = neighbors[idx_best]

                # Update rule: average with best neighbor
                new_grid[i, j] = (grid[i, j] + best_neighbor) / 2.0

                # Boundary enforcement
                new_grid[i, j] = np.clip(new_grid[i, j], lb, ub)

        # Update grid and recompute fitness
        grid = new_grid
        for i in range(grid_size):
            for j in range(grid_size):
                fitness[i, j] = f(grid[i, j])

        # Update global best
        best_idx = np.unravel_index(np.argmin(fitness), fitness.shape)
        if fitness[best_idx] < best_fitness:
            best_solution = grid[best_idx]
            best_fitness = fitness[best_idx]

        print(f"Iteration {iteration+1} | Best Fitness =
{best_fitness:.6f}")

    # Return best solution
    return best_solution, best_fitness


# 3️⃣ Run the algorithm
best_sol, best_fit = cellular_automata_optimization(objective_function,
grid_size=10, max_iterations=50, lb=-5, ub=5, dim=2)

print("\n✅ Best Solution Found:")
print("x =", best_sol)
print("Fitness =", best_fit)
```

Output:

```
Iteration 1 | Best Fitness = 0.022848
Iteration 2 | Best Fitness = 0.022848
Iteration 3 | Best Fitness = 0.008335
Iteration 4 | Best Fitness = 0.001871
Iteration 5 | Best Fitness = 0.000076
Iteration 6 | Best Fitness = 0.000076
Iteration 7 | Best Fitness = 0.000076
Iteration 8 | Best Fitness = 0.000004
Iteration 9 | Best Fitness = 0.000004
Iteration 10 | Best Fitness = 0.000000
Iteration 11 | Best Fitness = 0.000000
Iteration 12 | Best Fitness = 0.000000
Iteration 13 | Best Fitness = 0.000000
Iteration 14 | Best Fitness = 0.000000
Iteration 15 | Best Fitness = 0.000000
Iteration 16 | Best Fitness = 0.000000
Iteration 17 | Best Fitness = 0.000000
Iteration 18 | Best Fitness = 0.000000
Iteration 19 | Best Fitness = 0.000000
Iteration 20 | Best Fitness = 0.000000
Iteration 21 | Best Fitness = 0.000000
Iteration 22 | Best Fitness = 0.000000
Iteration 23 | Best Fitness = 0.000000
Iteration 24 | Best Fitness = 0.000000
Iteration 25 | Best Fitness = 0.000000
Iteration 26 | Best Fitness = 0.000000
Iteration 27 | Best Fitness = 0.000000
Iteration 28 | Best Fitness = 0.000000
Iteration 29 | Best Fitness = 0.000000
Iteration 30 | Best Fitness = 0.000000
Iteration 31 | Best Fitness = 0.000000
Iteration 32 | Best Fitness = 0.000000
Iteration 33 | Best Fitness = 0.000000
Iteration 34 | Best Fitness = 0.000000
Iteration 35 | Best Fitness = 0.000000
Iteration 36 | Best Fitness = 0.000000
Iteration 37 | Best Fitness = 0.000000
Iteration 38 | Best Fitness = 0.000000
Iteration 39 | Best Fitness = 0.000000
Iteration 40 | Best Fitness = 0.000000
Iteration 41 | Best Fitness = 0.000000
Iteration 42 | Best Fitness = 0.000000
Iteration 43 | Best Fitness = 0.000000
Iteration 44 | Best Fitness = 0.000000
Iteration 45 | Best Fitness = 0.000000
Iteration 46 | Best Fitness = 0.000000
Iteration 47 | Best Fitness = 0.000000
Iteration 48 | Best Fitness = 0.000000
Iteration 49 | Best Fitness = 0.000000
Iteration 50 | Best Fitness = 0.000000

✅ Best Solution Found:
x = [ 3.4797040e-12 -1.8043457e-12]
Fitness = 1.5364003308026177e-23
```