

## Grey Wolf Algorithm

```
import numpy as np
```

```
# Objective function to minimize
```

```
def objective_function(x):
```

```
    return x**2 - 4*x + 4 # simple convex function, min at x = 2
```

```
# Parallel Cellular Algorithm
```

```
def parallel_cellular_algorithm(obj_func, grid_size=(10, 10), lb=-10, ub=10, max_iter=100):
```

```
    rows, cols = grid_size
```

```
    num_cells = rows * cols
```

```
# Initialize cells randomly in search space
```

```
cells = np.random.uniform(lb, ub, (rows, cols))
```

```
# Evaluate initial fitness
```

```
fitness = obj_func(cells)
```

```
# Define neighborhood (3x3 Moore neighborhood)
```

```
def get_neighbors(r, c):
```

```
    neighbors = []
```

```
    for i in range(r-1, r+2):
```

```
        for j in range(c-1, c+2):
```

```
            if (0 <= i < rows) and (0 <= j < cols) and not (i == r and j == c):
```

```
                neighbors.append((i, j))
```

```
    return neighbors
```

```
# Track best solution
```

```
best_value = np.min(fitness)
```

```
best_position = np.unravel_index(np.argmin(fitness), fitness.shape)
```

```
# Main iteration loop
```

```
for t in range(max_iter):
```

```
    new_cells = np.copy(cells)
```

```
    for r in range(rows):
```

```
        for c in range(cols):
```

```
            neighbors = get_neighbors(r, c)
```

```
            # Find the best neighbor
```

```
            neighbor_values = np.array([cells[i, j] for i, j in neighbors])
```

```
            neighbor_fitness = np.array([obj_func(cells[i, j]) for i, j in neighbors])
```

```
            best_neighbor_value = neighbor_values[np.argmin(neighbor_fitness)]
```

```

        # Update rule: average of current cell and best neighbor
        new_cells[r, c] = (cells[r, c] + best_neighbor_value) / 2

    # Apply boundaries
    new_cells = np.clip(new_cells, lb, ub)

    # Update cells and fitness
    cells = new_cells
    fitness = obj_func(cells)

    # Track global best
    current_best = np.min(fitness)
    if current_best < best_value:
        best_value = current_best
        best_position = np.unravel_index(np.argmin(fitness), fitness.shape)

    # Optional: print progress
    if (t + 1) % 10 == 0 or t == 0:
        print(f"Iteration {t+1}/{max_iter} → Best fitness: {best_value:.6f}")

print("\nBest solution found:")
print(f"x = {cells[best_position]:.6f}, fitness = {best_value:.6f}")
return cells[best_position], best_value

# Example run
best_x, best_fit = parallel_cellular_algorithm(
    objective_function,
    grid_size=(10, 10),
    lb=-10,
    ub=10,
    max_iter=50
)

```

Output:

---

Iteration 1/50 → Best fitness: 0.000303  
Iteration 10/50 → Best fitness: 0.000000  
Iteration 20/50 → Best fitness: 0.000000  
Iteration 30/50 → Best fitness: 0.000000  
Iteration 40/50 → Best fitness: 0.000000  
Iteration 50/50 → Best fitness: 0.000000

Best solution found:

$x = 2.000000$ , fitness = 0.000000