Sanghamitra R
1BM22CS237

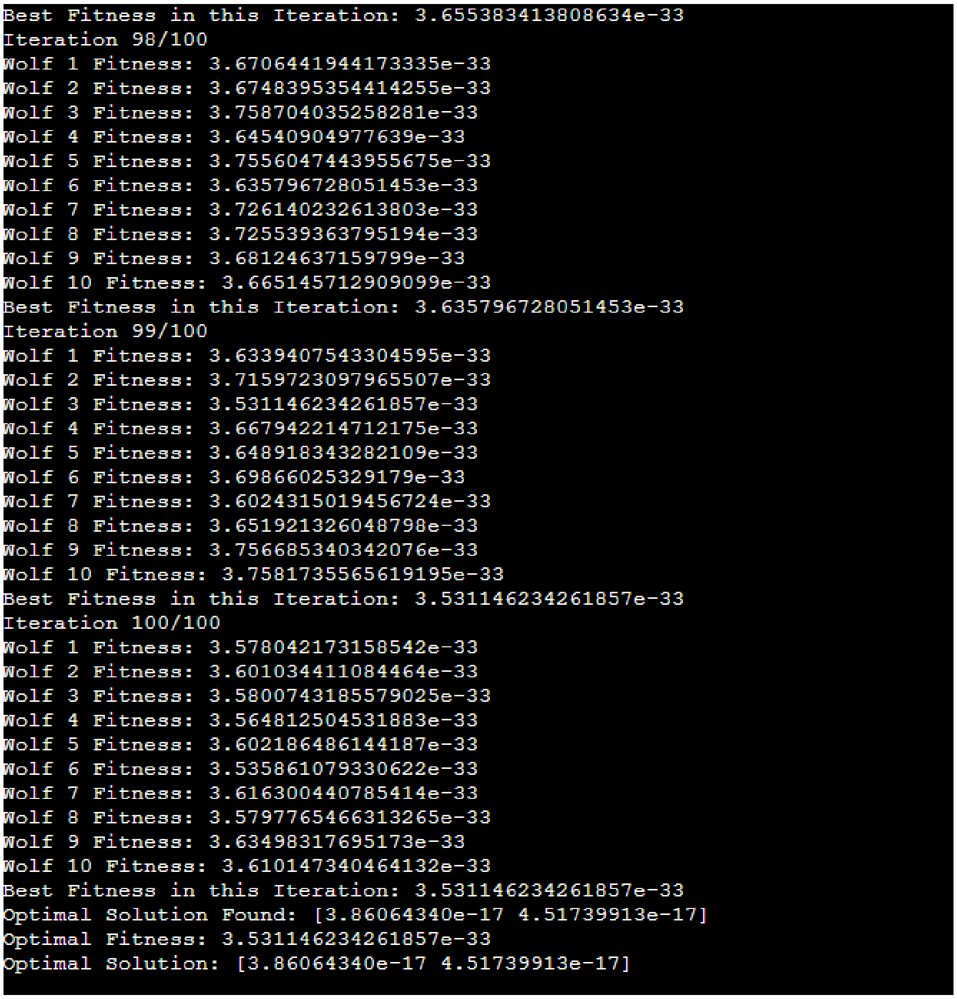
Grey wolf Optimiser

Code:

```
import numpy as np

def initialize_wolves(search_space, num_wolves):
    dimensions = len(search_space)
    wolves = np.zeros((num_wolves, dimensions))
    for i in range(num_wolves):
        wolves[i] = np.random.uniform(search_space[:, 0], search_space[:, 1])
    return wolves

def fitness_function(x):
    return np.sum(x**2)  # Example: Sphere function (minimize sum of squares)

def gwo_algorithm(search_space, num_wolves, max_iterations):
    dimensions = len(search_space)
    wolves = initialize_wolves(search_space, num_wolves)
    alpha_wolf = np.zeros(dimensions)
    beta_wolf = np.zeros(dimensions)
    gamma_wolf = np.zeros(dimensions)
    alpha_fitness = float('inf')
    beta_fitness = float('inf')
    gamma_fitness = float('inf')
    best_fitness = float('inf')

    for iteration in range(max_iterations):
        a = 2 - (iteration / max_iterations) * 2  # Linearly decreasing 'a'

        for i in range(num_wolves):
            fitness = fitness_function(wolves[i])
            if fitness < alpha_fitness:
                gamma_wolf = beta_wolf.copy()
                gamma_fitness = beta_fitness
                beta_wolf = alpha_wolf.copy()
                beta_fitness = alpha_fitness
                alpha_wolf = wolves[i].copy()
                alpha_fitness = fitness
            elif fitness < beta_fitness:
                gamma_wolf = beta_wolf.copy()
                gamma_fitness = beta_fitness
                beta_wolf = wolves[i].copy()
                beta_fitness = fitness
            elif fitness < gamma_fitness:
                gamma_wolf = wolves[i].copy()
                gamma_fitness = fitness

        for i in range(num_wolves):
            for j in range(dimensions):
                r1, r2 = np.random.random(), np.random.random()
                A1, C1 = 2 * a * r1 - a, 2 * r2
                D_alpha = abs(C1 * alpha_wolf[j] - wolves[i, j])
                X1 = alpha_wolf[j] - A1 * D_alpha

                r1, r2 = np.random.random(), np.random.random()
                A2, C2 = 2 * a * r1 - a, 2 * r2
                D_beta = abs(C2 * beta_wolf[j] - wolves[i, j])
                X2 = beta_wolf[j] - A2 * D_beta

                r1, r2 = np.random.random(), np.random.random()
                A3, C3 = 2 * a * r1 - a, 2 * r2
                D_gamma = abs(C3 * gamma_wolf[j] - wolves[i, j])
                X3 = gamma_wolf[j] - A3 * D_gamma

                wolves[i, j] = (X1 + X2 + X3) / 3
                wolves[i, j] = np.clip(wolves[i, j], search_space[j, 0], search_space[j, 1])

    return alpha_wolf  # Return the best solution found

# Example usage
search_space = np.array([[-5, 5], [-5, 5]])  # Define the search space bounds
num_wolves = 10  # Number of wolves
max_iterations = 100  # Maximum number of iterations

optimal_solution = gwo_algorithm(search_space, num_wolves, max_iterations)
print("Optimal Solution:", optimal_solution)

import numpy as np
```

```
def initialize_wolves(search_space, num_wolves):
    dimensions = len(search_space)
    wolves = np.zeros((num_wolves, dimensions))
    for i in range(num_wolves):
        wolves[i] = np.random.uniform(search_space[:, 0], search_space[:, 1])
    return wolves

def fitness_function(x):
    return np.sum(x**2)  # Example: Sphere function (minimize sum of squares)

def gwo_algorithm(search_space, num_wolves, max_iterations):
    dimensions = len(search_space)
    wolves = initialize_wolves(search_space, num_wolves)

    alpha_wolf = np.zeros(dimensions)
    beta_wolf = np.zeros(dimensions)
    gamma_wolf = np.zeros(dimensions)

    alpha_fitness = float('inf')
    beta_fitness = float('inf')
    gamma_fitness = float('inf')
    best_fitness = float('inf')

    for iteration in range(max_iterations):
        a = 2 - (iteration / max_iterations) * 2  # Parameter 'a' decreases linearly from 2 to
0
        print(f"Iteration {iteration + 1}/{max_iterations}")

        for i in range(num_wolves):
            fitness = fitness_function(wolves[i])
            print(f"Wolf {i+1} Fitness: {fitness}")

            if fitness < alpha_fitness:
                gamma_wolf = beta_wolf.copy()
                gamma_fitness = beta_fitness
                beta_wolf = alpha_wolf.copy()
                beta_fitness = alpha_fitness
                alpha_wolf = wolves[i].copy()
                alpha_fitness = fitness
            elif fitness < beta_fitness:
                gamma_wolf = beta_wolf.copy()
                gamma_fitness = beta_fitness
                beta_wolf = wolves[i].copy()
                beta_fitness = fitness
            elif fitness < gamma_fitness:
                gamma_wolf = wolves[i].copy()
                gamma_fitness = fitness

        print(f"Best Fitness in this Iteration: {alpha_fitness}")

        if alpha_fitness < best_fitness:
            best_fitness = alpha_fitness

        for i in range(num_wolves):
            for j in range(dimensions):
                r1 = np.random.random()
                r2 = np.random.random()

                A1 = 2 * a * r1 - a
                C1 = 2 * r2
                D_alpha = np.abs(C1 * alpha_wolf[j] - wolves[i, j])
                X1 = alpha_wolf[j] - A1 * D_alpha

                r1 = np.random.random()
                r2 = np.random.random()
                A2 = 2 * a * r1 - a
                C2 = 2 * r2
                D_beta = np.abs(C2 * beta_wolf[j] - wolves[i, j])
                X2 = beta_wolf[j] - A2 * D_beta

                r1 = np.random.random()
                r2 = np.random.random()
                A3 = 2 * a * r1 - a
                C3 = 2 * r2
                D_gamma = np.abs(C3 * gamma_wolf[j] - wolves[i, j])
                X3 = gamma_wolf[j] - A3 * D_gamma

                wolves[i, j] = (X1 + X2 + X3) / 3
                wolves[i, j] = np.clip(wolves[i, j], search_space[j, 0], search_space[j, 1])
            print(f"Optimal Solution Found: {alpha_wolf}")
            print(f"Optimal Fitness: {best_fitness}")
            return alpha_wolf

        # Example usage
        search_space = np.array([[-5, 5], [-5, 5]])  # Define the search space
        num_wolves = 10  # Number of wolves in the pack
        max_iterations = 100  # Maximum number of iterations

        # Run the GWO algorithm
        optimal_solution = gwo_algorithm(search_space, num_wolves, max_iterations)
        print("Optimal Solution:", optimal_solution)
```

Output:



```
Best Fitness in this Iteration: 3.655383413808634e-33
Iteration 98/100
Wolf 1 Fitness: 3.6706441944173335e-33
Wolf 2 Fitness: 3.6748395354414255e-33
Wolf 3 Fitness: 3.758704035258281e-33
Wolf 4 Fitness: 3.64540904977639e-33
Wolf 5 Fitness: 3.7556047443955675e-33
Wolf 6 Fitness: 3.635796728051453e-33
Wolf 7 Fitness: 3.726140232613803e-33
Wolf 8 Fitness: 3.725539363795194e-33
Wolf 9 Fitness: 3.68124637159799e-33
Wolf 10 Fitness: 3.665145712909099e-33
Best Fitness in this Iteration: 3.635796728051453e-33
Iteration 99/100
Wolf 1 Fitness: 3.6339407543304595e-33
Wolf 2 Fitness: 3.7159723097965507e-33
Wolf 3 Fitness: 3.531146234261857e-33
Wolf 4 Fitness: 3.667942214712175e-33
Wolf 5 Fitness: 3.648918343282109e-33
Wolf 6 Fitness: 3.69866025329179e-33
Wolf 7 Fitness: 3.6024315019456724e-33
Wolf 8 Fitness: 3.651921326048798e-33
Wolf 9 Fitness: 3.756685340342076e-33
Wolf 10 Fitness: 3.7581735565619195e-33
Best Fitness in this Iteration: 3.531146234261857e-33
Iteration 100/100
Wolf 1 Fitness: 3.578042173158542e-33
Wolf 2 Fitness: 3.601034411084464e-33
Wolf 3 Fitness: 3.5800743185579025e-33
Wolf 4 Fitness: 3.564812504531883e-33
Wolf 5 Fitness: 3.602186486144187e-33
Wolf 6 Fitness: 3.535861079330622e-33
Wolf 7 Fitness: 3.616300440785414e-33
Wolf 8 Fitness: 3.5797765466313265e-33
Wolf 9 Fitness: 3.63498317695173e-33
Wolf 10 Fitness: 3.610147340464132e-33
Best Fitness in this Iteration: 3.531146234261857e-33
Optimal Solution Found: [3.86064340e-17 4.51739913e-17]
Optimal Fitness: 3.531146234261857e-33
Optimal Solution: [3.86064340e-17 4.51739913e-17]
```