Lab-5

Grey Wolf Optimizer (GWO)

```
import numpy as np

def objective_function(x):
    """Example objective function: Sphere function."""
    return sum(x**2)

def initialize_population(dim, n_wolves, bounds):
    """Initialize the positions of the wolves randomly 
within the given bounds."""
    return np.random.uniform(bounds[0], bounds[1], 
(n_wolves, dim))

def gwo(objective_function, bounds, dim, n_wolves, 
n_iterations):
    """
    Grey Wolf Optimizer (GWO) algorithm.

    Parameters:
    - objective_function: The function to optimize.
    - bounds: Tuple of (lower_bound, upper_bound) for 
each dimension.
    - dim: Number of dimensions.
    - n_wolves: Number of wolves in the pack.
    - n_iterations: Number of iterations.

    Returns:
    - best_solution: The best solution found.
    - best_score: The best objective function value.
    """
    # Initialize population
    wolves = initialize_population(dim, n_wolves, bounds)
    fitness = np.apply_along_axis(objective_function, 1, 
wolves)

    # Initialize alpha, beta, and delta
    alpha, beta, delta = np.argsort(fitness)[:3]
    alpha_pos, alpha_score = wolves[alpha], 
fitness[alpha]
    beta_pos, beta_score = wolves[beta], fitness[beta]
```

```
    delta_pos, delta_score = wolves[delta], 
fitness[delta]

    # Main optimization loop
    for iteration in range(n_iterations):
        a = 2 - 2 * (iteration / n_iterations)  # 
Linearly decreasing a

        for i in range(n_wolves):
            for j in range(dim):
                # Update each wolf's position
                r1, r2 = np.random.rand(), 
np.random.rand()
                A1, C1 = 2 * a * r1 - a, 2 * r2
                D_alpha = abs(C1 * alpha_pos[j] - 
wolves[i, j])
                X1 = alpha_pos[j] - A1 * D_alpha

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

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

                # Average position update
                wolves[i, j] = (X1 + X2 + X3) / 3.0

            # Enforce bounds
            wolves[i, :] = np.clip(wolves[i, :], 
bounds[0], bounds[1])

        # Evaluate fitness and update alpha, beta, delta
        fitness = np.apply_along_axis(objective_function, 
1, wolves)
```

```
        sorted_indices = np.argsort(fitness)
        alpha, beta, delta = sorted_indices[:3]
        alpha_pos, alpha_score = wolves[alpha], 
fitness[alpha]
        beta_pos, beta_score = wolves[beta], 
fitness[beta]
        delta_pos, delta_score = wolves[delta], 
fitness[delta]

    return alpha_pos, alpha_score

# Example usage
dim = 5  # Number of dimensions
bounds = (-10, 10)  # Search space bounds
n_wolves = 30  # Number of wolves
n_iterations = 100  # Number of iterations

best_solution, best_score = gwo(objective_function, 
bounds, dim, n_wolves, n_iterations)
print(f"Best solution: {best_solution}")
print(f"Best score: {best_score}")
```

Output:

```
Best solution: [-1.52807921e-11 -1.39104785e-11  1.29132014e-11 -1.85709387e-11
 -1.49726055e-11]
Best score: 1.16281346713889e-21
```