Lab 5 – Grey Wolf Optimizer

```
import numpy as np

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

def gwo(obj_func, bounds, n_wolves=20, iterations=100):
    """
    Grey Wolf Optimizer (GWO) implementation.

    Parameters:
        obj_func: The objective function to optimize.
        bounds: A 2D array of [min, max] for each dimension.
        n_wolves: Number of wolves in the population.
        iterations: Number of iterations.

    Returns:
        best_position: The best solution found.
        best_fitness: The fitness of the best solution.
    """
    num_dimensions = len(bounds)

    # Initialize the population of wolves randomly
    wolves = np.random.uniform(bounds[:, 0], bounds[:, 1], (n_wolves, num_dimensions))
    fitness = np.array([obj_func(wolf) for wolf in wolves])

    # Identify alpha, beta, and delta wolves
    alpha, beta, delta = None, None, None
    alpha_fitness, beta_fitness, delta_fitness = float('inf'), float('inf'), float('inf')
```

```
for i in range(n_wolves):
    if fitness[i] < alpha_fitness:
        alpha_fitness = fitness[i]
        alpha = wolves[i]
    elif fitness[i] < beta_fitness:
        beta_fitness = fitness[i]
        beta = wolves[i]
    elif fitness[i] < delta_fitness:
        delta_fitness = fitness[i]
        delta = wolves[i]

# Coefficient vectors
a = 2  # Linearly decreases from 2 to 0 during iterations

for t in range(iterations):
    for i in range(n_wolves):
        A1 = 2 * a * np.random.random(num_dimensions) - a
        C1 = 2 * np.random.random(num_dimensions)
        D_alpha = abs(C1 * alpha - wolves[i])
        X1 = alpha - A1 * D_alpha

        A2 = 2 * a * np.random.random(num_dimensions) - a
        C2 = 2 * np.random.random(num_dimensions)
        D_beta = abs(C2 * beta - wolves[i])
        X2 = beta - A2 * D_beta

        A3 = 2 * a * np.random.random(num_dimensions) - a
        C3 = 2 * np.random.random(num_dimensions)
        D_delta = abs(C3 * delta - wolves[i])
        X3 = delta - A3 * D_delta
```

```
            # Update the position
            wolves[i] = (X1 + X2 + X3) / 3

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

        # Evaluate new fitness
        fitness = np.array([obj_func(wolf) for wolf in wolves])

        # Update alpha, beta, and delta wolves
        for i in range(n_wolves):
            if fitness[i] < alpha_fitness:
                alpha_fitness = fitness[i]
                alpha = wolves[i]
            elif fitness[i] < beta_fitness:
                beta_fitness = fitness[i]
                beta = wolves[i]
            elif fitness[i] < delta_fitness:
                delta_fitness = fitness[i]
                delta = wolves[i]

        # Decrease the value of a linearly
        a -= 2 / iterations

    return alpha, alpha_fitness

# Example usage
if __name__ == "__main__":
    bounds = np.array([[-10, 10], [-10, 10]])  # Define bounds for each dimension
    best_position, best_fitness = gwo(objective_function, bounds)
```

```
print("Best Position:", best_position)
print("Best Fitness:", best_fitness)
```

OUTPUT:

```
Best Position: [1.10548923e-18 8.25766757e-19]
Best Fitness: 1.8932094590349692e-36
```