Lab-5-Grey Wolf Optimizer (GWO)

Code:

```
import numpy as np

# Define the Grey Wolf Optimizer (GWO) algorithm
class GreyWolfOptimizer:
    def __init__(self, func, dim, num_wolves, max_iter, lb, ub):
        """
        :param func: The objective function to optimize (should take a numpy array as input)
        :param dim: The dimension of the search space
        :param num_wolves: The number of wolves (population size)
        :param max_iter: The maximum number of iterations
        :param lb: Lower bound of the search space
        :param ub: Upper bound of the search space
        """
        self.func = func
        self.dim = dim
        self.num_wolves = num_wolves
        self.max_iter = max_iter
        self.lb = lb
        self.ub = ub

        self.alpha_pos = np.zeros(dim)
        self.beta_pos = np.zeros(dim)
        self.delta_pos = np.zeros(dim)
        self.alpha_score = float("inf")
        self.beta_score = float("inf")
        self.delta_score = float("inf")

        # Initialize the positions of the wolves randomly
        self.positions = np.random.rand(self.num_wolves, self.dim) * (self.ub - self.lb) + self.lb

    def fitness(self, position):
        """Calculate the fitness value of a wolf"""
        return self.func(position)

    def update_position(self, wolf, alpha_pos, beta_pos, delta_pos, a, A, C):
        """Update the position of a wolf based on the positions of alpha, beta, and delta wolves"""
```

```
        # Update for alpha, beta, and delta wolves, using the respective A and C values
        D_alpha = np.abs(C[0] * alpha_pos - wolf)
        D_beta = np.abs(C[1] * beta_pos - wolf)
        D_delta = np.abs(C[2] * delta_pos - wolf)

        # Updated positions
        X1 = alpha_pos - A[0] * D_alpha
        X2 = beta_pos - A[1] * D_beta
        X3 = delta_pos - A[2] * D_delta

        # New position is the average of the three updated positions
        new_position = (X1 + X2 + X3) / 3
        return new_position

    def optimize(self):
        """Run the optimization process"""
        for t in range(self.max_iter):
            a = 2 - t * (2 / self.max_iter)  # Declining a over iterations
            A = 2 * a * np.random.rand(self.num_wolves, self.dim) - a  # Random vector for wolves' A values
            C = 2 * np.random.rand(self.num_wolves, self.dim)  # Random vector for wolves' C values

            for i in range(self.num_wolves):
                # Evaluate fitness of each wolf
                fitness_value = self.fitness(self.positions[i])

                # Update alpha, beta, and delta wolves
                if fitness_value < self.alpha_score:
                    self.alpha_score = fitness_value
                    self.alpha_pos = self.positions[i]
                elif fitness_value < self.beta_score:
                    self.beta_score = fitness_value
                    self.beta_pos = self.positions[i]
                elif fitness_value < self.delta_score:
                    self.delta_score = fitness_value
                    self.delta_pos = self.positions[i]

            # Update the positions of all wolves
            for i in range(self.num_wolves):
```

```
                self.positions[i] = self.update_position(self.positions[i], self.alpha_pos, self.beta_pos,
self.delta_pos, a, A[i], C[i])

            # Optionally, print progress
            if t % 1 == 0:  # Adjust the interval for printing
                print(f"Iteration {t}/{self.max_iter}, Best fitness: {self.alpha_score}")

        return self.alpha_pos, self.alpha_score

# Example usage:
def objective_function(x):
    """A simple objective function: Sphere function"""
    return np.sum(x**2)

# User input for parameters
dim = int(input("Enter the number of dimensions: "))
num_wolves = int(input("Enter the number of wolves: "))
max_iter = int(input("Enter the maximum number of iterations: "))
lb = float(input("Enter the lower bound of the search space: "))
ub = float(input("Enter the upper bound of the search space: "))

# Initialize and run the GWO algorithm
optimizer = GreyWolfOptimizer(func=objective_function, dim=dim, num_wolves=num_wolves,
max_iter=max_iter, lb=lb, ub=ub)
best_position, best_score = optimizer.optimize()

print(f"\nBest Position: {best_position}")
print(f"Best Score: {best_score}")
```

Output:

```
Enter the number of dimensions: 3
Enter the number of wolves: 35
Enter the maximum number of iterations: 9
Enter the lower bound of the search space: -5
Enter the upper bound of the search space: 5
Iteration 0/9, Best fitness: 5.1261148298595725
Iteration 1/9, Best fitness: 1.7348203391018997
Iteration 2/9, Best fitness: 1.350210015340434
Iteration 3/9, Best fitness: 0.30549540443394696
Iteration 4/9, Best fitness: 0.19136125125293585
Iteration 5/9, Best fitness: 0.15841368969575084
Iteration 6/9, Best fitness: 0.11811841118705839
Iteration 7/9, Best fitness: 0.10128658372111207
Iteration 8/9, Best fitness: 0.091277213497343

Best Position: [-0.21950171  0.16166378  0.14070943]
Best Score: 0.091277213497343
```