```python
import numpy as np

# Objective function (to minimize)
def objective_function(x):
    return np.sum(x**2)

# Grey Wolf Optimizer (GWO) implementation
class GreyWolfOptimizer:
    def __init__(self, objective_function, n_wolves, max_iter, dim, lower_bound, upper_bound):
        self.obj_func = objective_function
        self.n_wolves = n_wolves
        self.max_iter = max_iter
        self.dim = dim
        self.lower_bound = lower_bound
        self.upper_bound = upper_bound
        self.alpha_position = np.zeros(dim)
        self.beta_position = np.zeros(dim)
        self.delta_position = np.zeros(dim)
        self.alpha_score = float("inf")
        self.beta_score = float("inf")
        self.delta_score = float("inf")
        self.wolves_position = np.random.uniform(lower_bound, upper_bound, (n_wolves, dim))

    def update_position(self, A, C, wolf_position, best_position):
        # Update the position of each wolf based on alpha, beta, or delta wolves
        D = np.abs(C * best_position - wolf_position)
        return best_position - A * D

    def optimize(self):
        for t in range(self.max_iter):
            for i in range(self.n_wolves):
                fitness = self.obj_func(self.wolves_position[i])

                # Update alpha, beta, and delta wolves
                if fitness < self.alpha_score:
                    self.alpha_score = fitness
                    self.alpha_position = self.wolves_position[i].copy()

                elif fitness < self.beta_score and fitness > self.alpha_score:
                    self.beta_score = fitness
                    self.beta_position = self.wolves_position[i].copy()

                elif fitness < self.delta_score and fitness > self.beta_score:
                    self.delta_score = fitness
                    self.delta_position = self.wolves_position[i].copy()
```

```
            # Calculate A and C coefficients
            a = 2 - t * (2 / self.max_iter)  # Linearly decreases from 2 to 0
            r1, r2, r3 = np.random.rand(3, self.n_wolves, self.dim)  # Random values for each wolf and dimension
            A1 = 2 * a * r1 - a
            A2 = 2 * a * r2 - a
            A3 = 2 * a * r3 - a
            C1 = 2 * r1
            C2 = 2 * r2
            C3 = 2 * r3

            # Update positions of all wolves
            for i in range(self.n_wolves):
                # Update the position of each wolf using alpha, beta, delta
                self.wolves_position[i] = self.update_position(A1[i], C1[i], self.wolves_position[i], self.alpha_position)
                self.wolves_position[i] = self.update_position(A2[i], C2[i], self.wolves_position[i], self.beta_position)
                self.wolves_position[i] = self.update_position(A3[i], C3[i], self.wolves_position[i], self.delta_position)

                # Ensure the wolves stay within the bounds
                self.wolves_position[i] = np.clip(self.wolves_position[i], self.lower_bound, self.upper_bound)

        return self.alpha_position, self.alpha_score

# Parameters for the optimizer
n_wolves = 50
max_iter = 100
dim = 2
lower_bound = -10
upper_bound = 10

# Instantiate the GWO
gwo = GreyWolfOptimizer(objective_function, n_wolves, max_iter, dim, lower_bound, upper_bound)

# Run the optimization
best_position, best_score = gwo.optimize()

# Output the results
print("Best solution found: ", best_position)
print("Objective function value: ", best_score)
```

```
Best solution found:  [-4.03126745e-19  4.16286660e-19]
Objective function value:  3.358057558818742e-37
```