```
import numpy as np


def objective_function(x):
    return np.sum(x**2)


class GreyWolfOptimizer:
    def __init__(self, n_wolves, n_dimensions, max_iter, lower_bound, upper_bound, objective_function, convergence_threshold=1e-6):
        self.n_wolves = n_wolves  # Number of wolves
        self.n_dimensions = n_dimensions  # Number of dimensions
        self.max_iter = max_iter  # Maximum iterations
        self.lower_bound = lower_bound  # Lower bound of the search space
        self.upper_bound = upper_bound  # Upper bound of the search space
        self.objective_function = objective_function  # Objective function to optimize
        self.convergence_threshold = convergence_threshold  # Convergence threshold for stopping early

        # Initialize the wolves' positions randomly within the bounds
        self.positions = np.random.uniform(self.lower_bound, self.upper_bound, (self.n_wolves, self.n_dimensions))
        self.fitness = np.array([self.objective_function(self.positions[i]) for i in range(self.n_wolves)])

        # Initialize alpha, beta, and delta wolves
        self.alpha_position = np.zeros(self.n_dimensions)
        self.beta_position = np.zeros(self.n_dimensions)
        self.delta_position = np.zeros(self.n_dimensions)
        self.alpha_score = float('inf')
        self.beta_score = float('inf')
        self.delta_score = float('inf')

    def update_wolves(self, a):
        for i in range(self.n_wolves):
            # Calculate the distance of the current wolf from the alpha, beta, and delta wolves
            A1 = 2 * a * np.random.rand(self.n_dimensions) - a
            C1 = 2 * np.random.rand(self.n_dimensions)
            D_alpha = np.abs(C1 * self.alpha_position - self.positions[i])
            X1 = self.alpha_position - A1 * D_alpha

            A2 = 2 * a * np.random.rand(self.n_dimensions) - a
            C2 = 2 * np.random.rand(self.n_dimensions)
            D_beta = np.abs(C2 * self.beta_position - self.positions[i])
            X2 = self.beta_position - A2 * D_beta

            A3 = 2 * a * np.random.rand(self.n_dimensions) - a
            C3 = 2 * np.random.rand(self.n_dimensions)
            D_delta = np.abs(C3 * self.delta_position - self.positions[i])
            X3 = self.delta_position - A3 * D_delta

            # Update the position of the wolf
            self.positions[i] = (X1 + X2 + X3) / 3

            # Apply bounds to the new position
            self.positions[i] = np.clip(self.positions[i], self.lower_bound, self.upper_bound)

            # Calculate fitness for the new position
            fitness = self.objective_function(self.positions[i])

            # Update alpha, beta, and delta if necessary
            if fitness < self.alpha_score:
                self.alpha_score = fitness
                self.alpha_position = self.positions[i]
            elif fitness < self.beta_score:
                self.beta_score = fitness
                self.beta_position = self.positions[i]
            elif fitness < self.delta_score:
                self.delta_score = fitness
                self.delta_position = self.positions[i]

    def optimize(self):
        # Iterate for the given number of iterations
        for t in range(self.max_iter):
            # Calculate the coefficient a
            a = 2 - t * (2 / self.max_iter)

            # Update the wolves' positions
            self.update_wolves(a)

            # Check for convergence based on the alpha_score (best fitness)
            if self.alpha_score < self.convergence_threshold:
                print(f"Converged early at iteration {t+1} with best fitness = {self.alpha_score}")
                break  # Exit early if the convergence threshold is reached
```

```
            print(f"Iteration {t+1}/{self.max_iter}: Best Fitness = {self.alpha_score}")

        # Return the best position and the corresponding fitness score
        return self.alpha_position, self.alpha_score

# Parameters for optimization
n_wolves = 30  # Number of wolves
n_dimensions = 2  # Number of dimensions (problem size)
max_iter = 100  # Maximum number of iterations
lower_bound = -10  # Lower bound of search space
upper_bound = 10  # Upper bound of search space

# Initialize and run the Grey Wolf Optimization
gwo = GreyWolfOptimizer(n_wolves, n_dimensions, max_iter, lower_bound, upper_bound, objective_function)
best_position, best_fitness = gwo.optimize()

print(f"Best Solution: {best_position}")
print(f"Best Fitness: {best_fitness}")
```

```
Iteration 1/100: Best Fitness = 11.445334546946041
Iteration 2/100: Best Fitness = 0.20456898464499124
Iteration 3/100: Best Fitness = 0.11100067443115716
Iteration 4/100: Best Fitness = 0.03246062571756299
Iteration 5/100: Best Fitness = 0.02633145150915702
Iteration 6/100: Best Fitness = 0.0010657909801555366
Iteration 7/100: Best Fitness = 0.0007974787349212369
Iteration 8/100: Best Fitness = 2.4891278605268483e-05
Iteration 9/100: Best Fitness = 2.3822376131733303e-06
Converged early at iteration 10 with best fitness = 1.0274802206744994e-07
Best Solution: [-2.46266618e-05 -3.19595916e-04]
Best Fitness: 1.0274802206744994e-07
```