```
# Grey Wolf

import numpy as np

def obj_fn(x):
    """Objective function to minimize."""
    return np.sum(x**2)  # Example: Sphere function

def gwo(obj_fn, dim, wolves, iters, lb, ub):
    """Grey Wolf Optimizer (GWO) implementation."""
    # Initialize wolf positions
    pos = np.random.uniform(low=lb, high=ub, size=(wolves, dim))
    a_pos, b_pos, d_pos = np.zeros(dim), np.zeros(dim), np.zeros(dim)
    a_score, b_score, d_score = float("inf"), float("inf"), float("inf")

    for t in range(iters):
        for i in range(wolves):
            fit = obj_fn(pos[i])
            # Update Alpha, Beta, Delta
            if fit < a_score:
                d_score, d_pos = b_score, b_pos.copy()
                b_score, b_pos = a_score, a_pos.copy()
                a_score, a_pos = fit, pos[i].copy()
            elif fit < b_score:
                d_score, d_pos = b_score, b_pos.copy()
                b_score, b_pos = fit, pos[i].copy()
            elif fit < d_score:
                d_score, d_pos = fit, pos[i].copy()

        # Update wolf positions
        a = 2 - t * (2 / iters)  # Linearly decreasing factor
        for i in range(wolves):
            for j in range(dim):
                r1, r2 = np.random.rand(), np.random.rand()
                A1, C1 = 2 * a * r1 - a, 2 * r2
                D_a = abs(C1 * a_pos[j] - pos[i, j])
                X1 = a_pos[j] - A1 * D_a

                r1, r2 = np.random.rand(), np.random.rand()
                A2, C2 = 2 * a * r1 - a, 2 * r2
                D_b = abs(C2 * b_pos[j] - pos[i, j])
                X2 = b_pos[j] - A2 * D_b

                r1, r2 = np.random.rand(), np.random.rand()
                A3, C3 = 2 * a * r1 - a, 2 * r2
                D_d = abs(C3 * d_pos[j] - pos[i, j])
                X3 = d_pos[j] - A3 * D_d

                # Update position
                pos[i, j] = (X1 + X2 + X3) / 3

            # Keep wolves within bounds
            pos[i] = np.clip(pos[i], lb, ub)

        # Print progress
        print(f"Iter {t+1}/{iters}, Best Score: {a_score}, Best Pos: {a_pos}")

    return a_score, a_pos

# Parameters
dim = 5       # Problem dimension
wolves = 20   # Number of wolves
iters = 50   # Number of iterations
lb = -10      # Lower bound
ub = 10       # Upper bound

# Run GWO
best_score, best_pos = gwo(obj_fn, dim, wolves, iters, lb, ub)
print("\nFinal Best Score:", best_score)
print("Final Best Pos:", best_pos)
```

```
 -9.33495824e-05]
Iter 29/50, Best Score: 4.8612940167762056e-08, Best Pos: [ 8.73799545e-05  6.90619292e-05 -1.03958694e-04  8.78444381e-05
 -1.32981496e-04]
Iter 30/50, Best Score: 4.465014099864875e-08, Best Pos: [ 8.79493032e-05  9.19031929e-05 -9.18147734e-05  1.03737943e-04
 -9.63190046e-05]
Iter 31/50, Best Score: 3.376436040408867e-08, Best Pos: [ 5.99788742e-05  6.32230283e-05 -7.27721367e-05  5.96035430e-05
 -1.31610704e-04]
Iter 32/50, Best Score: 3.244011607715192e-08, Best Pos: [ 5.22935300e-05  6.95153585e-05 -8.46519734e-05  9.09855499e-05
 -9.71019611e-05]
Iter 33/50, Best Score: 2.3542582236164313e-08, Best Pos: [ 3.68869824e-05  6.79391776e-05 -8.32565410e-05  5.66527003e-05
 -8.61685606e-05]
Iter 34/50, Best Score: 1.7068040435965883e-08, Best Pos: [ 4.71329736e-05  4.83255705e-05 -6.94119808e-05  5.73526092e-05
 -6.63612659e-05]
Iter 35/50, Best Score: 1.536107962393052e-08, Best Pos: [ 4.90772166e-05  4.60937557e-05 -6.68101468e-05  4.67733069e-05
 -6.46261106e-05]
Iter 36/50, Best Score: 1.2807750790739046e-08, Best Pos: [ 5.14103722e-05  5.25440899e-05 -3.70818629e-05  5.06945114e-05
 -5.88119460e-05]
Iter 37/50, Best Score: 1.028277927282104e-08, Best Pos: [ 4.44082431e-05  4.08383928e-05 -5.49306115e-05  4.33896783e-05
 -4.17477740e-05]
Iter 38/50, Best Score: 8.82859250073914e-09, Best Pos: [ 4.10002458e-05  3.59081636e-05 -4.58068630e-05  4.28765058e-05
 -4.38350623e-05]
Iter 39/50, Best Score: 7.187578241412738e-09, Best Pos: [ 3.75139915e-05  3.31563221e-05 -3.61636530e-05  3.73221820e-05
 -4.44992351e-05]
Iter 40/50, Best Score: 6.5525208121222395e-09, Best Pos: [ 3.65984240e-05  3.39879024e-05 -3.73587571e-05  3.66725005e-05
 -3.62953115e-05]
Iter 41/50, Best Score: 5.631151900085803e-09, Best Pos: [ 3.12248163e-05  3.14920148e-05 -3.46760718e-05  3.81062260e-05
 -3.17789449e-05]
Iter 42/50, Best Score: 5.193113818898217e-09, Best Pos: [ 3.42860149e-05  3.27507523e-05 -2.93862094e-05  3.50354036e-05
 -2.92222933e-05]
Iter 43/50, Best Score: 4.477534815881126e-09, Best Pos: [ 2.72499669e-05  3.38947883e-05 -2.91677902e-05  3.35127196e-05
 -2.47437887e-05]
Iter 44/50, Best Score: 4.1542671221332185e-09, Best Pos: [ 2.60864946e-05  3.06555372e-05 -3.00784053e-05  3.04082311e-05
 -2.65448484e-05]
Iter 45/50, Best Score: 3.8614837517460874e-09, Best Pos: [ 2.52420327e-05  2.98192445e-05 -3.07223507e-05  2.67003755e-05
 -2.60454085e-05]
Iter 46/50, Best Score: 3.550695374205194e-09, Best Pos: [ 2.57683344e-05  2.80457103e-05 -2.69830007e-05  2.85071093e-05
 -2.36514025e-05]
Iter 47/50, Best Score: 3.464538016730741e-09, Best Pos: [ 2.51266126e-05  2.86058209e-05 -2.59044695e-05  2.72851806e-05
 -2.44821515e-05]
Iter 48/50, Best Score: 3.3692584538908164e-09, Best Pos: [ 2.42463680e-05  2.74184818e-05 -2.65870380e-05  2.75119174e-05
 -2.37870292e-05]
Iter 49/50, Best Score: 3.28967835050078e-09, Best Pos: [ 2.52370904e-05  2.70586475e-05 -2.68253827e-05  2.70011014e-05
 -2.17241013e-05]
Iter 50/50, Best Score: 3.1862735571213157e-09, Best Pos: [ 2.39704899e-05  2.74038207e-05 -2.60185465e-05  2.61912051e-05
 -2.23108897e-05]

Final Best Score: 3.1862735571213157e-09
Final Best Pos: [ 2.39704899e-05  2.74038207e-05 -2.60185465e-05  2.61912051e-05
 -2.23108897e-05]
```