Grey Wolf Optimizer

```
import numpy as np


# Objective function (Example: Sphere function)
def objective_function(x):
    return np.sum(x**2)


# Grey Wolf Optimization Algorithm
class GreyWolfOptimizer:
    def __init__(self, obj_func, dim, n_wolves, max_iter, lb, ub):
        self.obj_func = obj_func  # Objective function
        self.dim = dim            # Dimensionality of the problem
        self.n_wolves = n_wolves # Number of wolves
        self.max_iter = max_iter # Maximum number of iterations
        self.lb = lb             # Lower bound of the search space
        self.ub = ub             # Upper bound of the search space
        self.alpha_pos = np.zeros(dim)  # Position of alpha wolf
        self.alpha_score = float("inf") # Fitness of alpha wolf
        self.beta_pos = np.zeros(dim)   # Position of beta wolf
        self.beta_score = float("inf")  # Fitness of beta wolf
        self.delta_pos = np.zeros(dim)  # Position of delta wolf
        self.delta_score = float("inf") # Fitness of delta wolf
        self.positions = np.random.rand(n_wolves, dim) * (ub - lb) + lb  # Initial positions of wolves
        self.scores = np.zeros(n_wolves)  # Fitness scores


    def optimize(self):
        # Main optimization loop
        for t in range(self.max_iter):
            a = 2 - t * (2 / self.max_iter)  # Decreases linearly from 2 to 0
            for i in range(self.n_wolves):
                # Evaluate fitness of each wolf
```

```
        self.scores[i] = self.obj_func(self.positions[i])

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

    # Update the positions of the wolves
    for i in range(self.n_wolves):
        # Calculate random values for A and C
        r1 = np.random.rand(self.dim)
        r2 = np.random.rand(self.dim)
        A = 2 * a * r1 - a
        C = 2 * r2

        # Update the position of the wolf
        D_alpha = np.abs(C * self.alpha_pos - self.positions[i])
        D_beta = np.abs(C * self.beta_pos - self.positions[i])
        D_delta = np.abs(C * self.delta_pos - self.positions[i])

        X1 = self.alpha_pos - A * D_alpha
        X2 = self.beta_pos - A * D_beta
        X3 = self.delta_pos - A * D_delta

        # Calculate new position for the wolf
```

```
            self.positions[i] = (X1 + X2 + X3) / 3

            # Apply boundary constraints (if any)
            self.positions[i] = np.clip(self.positions[i], self.lb, self.ub)

        # Optionally, print the best solution found so far
        #print(f"Iteration {t+1}/{self.max_iter} - Best Score: {self.alpha_score}")

    # Return the best solution found
    return self.alpha_pos, self.alpha_score

# Hyperparameters
dim = 30            # Number of dimensions (variables)
n_wolves = 50       # Number of wolves (population size)
max_iter = 1000     # Maximum number of iterations
lb = -10            # Lower bound of search space
ub = 10             # Upper bound of search space

# Instantiate the optimizer
optimizer = GreyWolfOptimizer(obj_func=objective_function, dim=dim, n_wolves=n_wolves,
max_iter=max_iter, lb=lb, ub=ub)

# Perform optimization
best_position, best_score = optimizer.optimize()

# Output the best solution found
print("\nBest Position: ", best_position)
print("Best Score: ", best_score)
```

```
Best Position:  [ 3.31543532e-28 -2.57971219e-28  2.90350626e-28 -3.27713250e-28
  3.52185014e-28  2.80085911e-28 -3.38381673e-28 -2.97466794e-28
 -2.31745008e-28  3.13252393e-28 -2.87816050e-28  1.79119454e-28
 -2.84588645e-28  2.90763602e-28 -3.38953643e-28 -3.35192731e-28
 -2.62987429e-28 -3.10876600e-28  3.13119841e-28  3.25839295e-28
 -2.77855075e-28  3.09139060e-28  2.99660816e-28  2.85167667e-28
 -2.75530248e-28 -3.56770417e-28  2.01980511e-28  3.23116555e-28
 -3.70571356e-28  3.36635177e-28]
Best Score:  2.744736165706468e-54
```