Grey wolf optimization

```
import numpy as np

def grey_wolf_optimization(
    objective,
    bounds,
    n_wolves=25,
    n_iter=250,
    seed=None,
    return_history=True,
):
    """
    Grey Wolf Optimizer (GWO) for bounded minimization.

    Parameters
    ----------
    objective : callable
        f(x) -> float. Accepts a 1D numpy array of shape (d,), returns
scalar.
    bounds : list[tuple[float,float]] | np.ndarray
        Variable bounds [(min, max), ...] of length d.
    n_wolves : int
        Population size.
    n_iter : int
        Number of iterations.
    seed : int | None
        RNG seed for reproducibility.
    return_history : bool
        If True, also return array of best fitness per iteration.

    Returns
    -------
    best_x : np.ndarray, shape (d,)
    best_f : float
    history : np.ndarray, shape (n_iter,)  (only if return_history=True)
    """
    rng = np.random.default_rng(seed)

    bounds = np.array(bounds, dtype=float)
    lb, ub = bounds[:, 0], bounds[:, 1]
    assert np.all(ub > lb), "Each upper bound must be greater than lower
bound."
    d = len(bounds)

    # Initialize wolves uniformly within bounds
    X = rng.uniform(lb, ub, size=(n_wolves, d))
```

```
    # Evaluate
    fitness = np.apply_along_axis(objective, 1, X)

    # Track alpha (best), beta (2nd), delta (3rd)
    def top3(X, fitness):
        idx = np.argsort(fitness)
        return X[idx[0]], fitness[idx[0]], X[idx[1]], fitness[idx[1]],
X[idx[2]], fitness[idx[2]]

    X_alpha, f_alpha, X_beta, f_beta, X_delta, f_delta = top3(X, fitness)

    history = np.empty(n_iter, dtype=float)

    for t in range(n_iter):
        # Linearly decrease 'a' from 2 to 0
        a = 2.0 - 2.0 * (t / (n_iter - 1 if n_iter > 1 else 1))

        # Random coefficients
        r1 = rng.random((n_wolves, d))
        r2 = rng.random((n_wolves, d))

        A1 = 2 * a * r1 - a
        C1 = 2 * r2

        r1 = rng.random((n_wolves, d))
        r2 = rng.random((n_wolves, d))
        A2 = 2 * a * r1 - a
        C2 = 2 * r2

        r1 = rng.random((n_wolves, d))
        r2 = rng.random((n_wolves, d))
        A3 = 2 * a * r1 - a
        C3 = 2 * r2

        # Distance vectors
        D_alpha = np.abs(C1 * X_alpha - X)
        D_beta  = np.abs(C2 * X_beta  - X)
        D_delta = np.abs(C3 * X_delta - X)

        # Candidate positions from alpha/beta/delta
        X1 = X_alpha - A1 * D_alpha
        X2 = X_beta  - A2 * D_beta
        X3 = X_delta - A3 * D_delta

        # New position is the mean of the three
        X_new = (X1 + X2 + X3) / 3.0
```

```
        # Enforce bounds (simple clipping)
        X_new = np.clip(X_new, lb, ub)

        # Evaluate and update pack
        f_new = np.apply_along_axis(objective, 1, X_new)

        # Greedy replacement
        replace = f_new < fitness
        X[replace] = X_new[replace]
        fitness[replace] = f_new[replace]

        # Update leaders
        X_alpha, f_alpha, X_beta, f_beta, X_delta, f_delta = top3(X,
fitness)

        if return_history:
            history[t] = f_alpha

    return (X_alpha, f_alpha, history) if return_history else (X_alpha,
f_alpha)


# --------------------------
# Example usage
# --------------------------
if __name__ == "__main__":
    # Rastrigin function (global min at x=0 with f=0). Harder than Sphere.
    def rastrigin(x):
        A = 10.0
        return A * x.size + np.sum(x**2 - A * np.cos(2 * np.pi * x))

    dim = 10
    bounds = [(-5.12, 5.12)] * dim

    best_x, best_f, hist = grey_wolf_optimization(
        rastrigin,
        bounds,
        n_wolves=30,
        n_iter=500,
        seed=42,
        return_history=True,
    )

    print("Best f:", best_f)
    print("Best x (first 5 dims):", np.round(best_x[:5], 4))
```

Output:

```
Best f: 5.325315880262934
Best x (first 5 dims): [ 0.9956 -0.0277  0.9949  0.9948 -0.0094]
```