Cuckoo search algorithm

```
import numpy as np
import math

def _levy_flight(shape, beta=1.5, rng=None):
    """
    Generate Lévy flight steps using Mantegna's algorithm.
    Returns an array with given shape.
    """
    if rng is None:
        rng = np.random.default_rng()
    # Mantegna parameters (using math module instead of np.math)
    sigma_u = (
        math.gamma(1 + beta) * math.sin(math.pi * beta / 2)
        / (math.gamma((1 + beta) / 2) * beta * 2 ** ((beta - 1) / 2))
    ) ** (1 / beta)
    u = rng.normal(0, sigma_u, size=shape)
    v = rng.normal(0, 1, size=shape)
    step = u / (np.abs(v) ** (1 / beta))
    return step


def cuckoo_search(
    objective,
    bounds,
    n_nests=25,
    n_iter=250,
    pa=0.25,            # discovery/abandonment probability
    alpha=0.01,         # step size coefficient for Lévy flights
    beta=1.5,           # Lévy distribution exponent
    seed=None,
    return_history=True,
):
    """
    Cuckoo Search (CS) metaheuristic for bounded minimization.
    """
    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)
```

```
    def clip(X):
        return np.clip(X, lb, ub)

    # Initialize nests uniformly
    nests = rng.uniform(lb, ub, size=(n_nests, d))
    fitness = np.apply_along_axis(objective, 1, nests)

    best_idx = int(np.argmin(fitness))
    best = nests[best_idx].copy()
    best_f = float(fitness[best_idx])

    history = np.empty(n_iter, dtype=float) if return_history else
None
    scale = (ub - lb)
    step_scale = alpha * scale

    for t in range(n_iter):
        # Lévy flight step
        steps = _levy_flight((n_nests, d), beta=beta, rng=rng) *
step_scale
        cuckoos = nests + steps * (nests - best)
        cuckoos = clip(cuckoos)

        cuckoos_fit = np.apply_along_axis(objective, 1, cuckoos)

        # Replace some nests
        rand_idx = rng.integers(0, n_nests, size=n_nests)
        replace_mask = cuckoos_fit < fitness[rand_idx]
        nests[rand_idx[replace_mask]] = cuckoos[replace_mask]
        fitness[rand_idx[replace_mask]] = cuckoos_fit[replace_mask]

        # Abandon worst nests
        n_abandon = max(1, int(pa * n_nests))
        worst_idx = np.argsort(fitness)[-n_abandon:]
        i_idx = rng.integers(0, n_nests, size=n_abandon)
        j_idx = rng.integers(0, n_nests, size=n_abandon)
        eps = rng.random((n_abandon, d))
        new_nests = nests[worst_idx] + eps * (nests[i_idx] -
nests[j_idx])
        new_nests += 0.001 * rng.normal(size=new_nests.shape) * scale
        new_nests = clip(new_nests)

        new_fit = np.apply_along_axis(objective, 1, new_nests)
        better_mask = new_fit < fitness[worst_idx]
        nests[worst_idx[better_mask]] = new_nests[better_mask]
```

```
        fitness[worst_idx[better_mask]] = new_fit[better_mask]

        # Update best
        curr_idx = int(np.argmin(fitness))
        curr_best_f = float(fitness[curr_idx])
        if curr_best_f < best_f:
            best_f = curr_best_f
            best = nests[curr_idx].copy()

        if return_history:
            history[t] = best_f

    return (best, best_f, history) if return_history else (best,
best_f)


# --------------------------
# Example usage
# --------------------------
if __name__ == "__main__":
    # Rastrigin test function
    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 = cuckoo_search(
        rastrigin,
        bounds,
        n_nests=30,
        n_iter=500,
        pa=0.25,
        alpha=0.05,
        beta=1.5,
        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: 23.878956827787533
Best x (first 5 dims): [-0.995   2.9849 -2.9849  0.      0.995 ]
```