```python
import numpy as np
import lib
import time
import random

def gen_params(parameters):
    p = np.zeros(len(parameters), dtype=np.float64)
    for i, par in enumerate(parameters):
        mini = par["min"]
        maxi = par["max"]
        p[i] = np.random.uniform(mini, maxi)
    return p


def a(costf=None, parameters=None, N=100, max_time=-1, debug=False):
    if costf is None:
        raise Exception("costf is a required kwarg")
    if parameters is None:
        raise Exception("parameters is a required kwarg")
    best_params = None
    best_cost = None
    it_best_costs = []
    it_best_params = []
    it_params = []
    start_time = time.perf_counter()
    times = []
    it = 0
    if max_time > 0:
        N = -1
    current_time = 0
    while (it < N or N < 0) and (current_time < max_time or max_time < 0):
        it += 1
        ps = gen_params(parameters)
        cost = costf(ps)
        if best_cost is None or np.isnan(best_cost) or cost < best_cost:
            best_params = ps
            best_cost = cost
        it_best_costs.append(best_cost)
        it_best_params.append(best_params)
        it_params.append(ps)
        current_time = time.perf_counter() - start_time
        times.append(current_time)
        if debug:
            print("parameters:", ps, end="\t")
            print("cost:", cost, end="\t")
            print("best cost:", best_cost)
    return {
        "results": {
            "best_params": best_params.tolist(),
            "best_cost": best_cost,
        },
        "stats": {
            "it_best_costs": it_best_costs,
            "it_best_params": list(map(lambda x: x.tolist(), it_best_params)),
            "it_params": list(map(lambda x: x.tolist(), it_params)),
            "time": times,
        }
    }

def best_m(params, costs, M=10, unzip=True):
    bests = sorted(zip(params, costs), key=lambda x: x[1])
    best_M = bests[:M]
    if unzip:
        return list(zip(*best_M))
    return best_M

def bests2parameters(bests):
    params = bests[0]
    p1 = params[0]
    ps = []
    for i in range(len(p1)):
        param_values = list(map(lambda x: x[i], params))
        ps.append({
            "min": min(param_values),
            "max": max(param_values),
            })
    return ps

def b_mod(costf=None, parameters=None, iterations=2, N=100, M=10, max_time=-1, debug=False):
    if costf is None:
        raise Exception("costf is a required kwarg")
    if parameters is None:
        raise Exception("parameters is a required kwarg")
    it_best_costs = []
    it_best_params = []
    it_params = []
    start_time = time.perf_counter()
    best_cost = None
    best_params = None
    times = []
    current_time = 0
    iteration_results = []
    for i in range(iterations):
        if debug:
            print("iteration: ", i)
        if max_time > 0 and current_time > max_time:
            break
        params = []
        costs = []
        it = 0
        while it < N:
            it += 1
            ps = gen_params(parameters)
            cost = costf(ps)
            params.append(ps)
            costs.append(cost)
            if best_cost is None or cost < best_cost:
                best_params = ps
                best_cost = cost
            it_best_costs.append(best_cost)
            it_best_params.append(best_params)
            it_params.append(ps)
            current_time = time.perf_counter() - start_time
            times.append(current_time)
            if debug:
                print("parameters:", ps, end="\t")
                print("cost:", cost, end="\t")
                print("best cost:", best_cost)
        bests = best_m(params, costs, M=M)
        parameters = bests2parameters(bests)
        iteration_results.append({
                "M": M,
                "best_m_params": list(map(lambda x: x.tolist(), bests[0])),
                "best_m_costs": bests[1],
                "best_params": best_params.tolist(),
                "best_cost": best_cost,
        })
    return {
        "results": {
            "best_params": best_params.tolist(),
            "best_cost": best_cost,
        },
        "stats": {
            "it_best_costs": it_best_costs,
            "it_best_params": list(map(lambda x: x.tolist(), it_best_params)),
            "it_params": list(map(lambda x: x.tolist(), it_params)),
            "time": times,
        },
        "iteration_results": iteration_results,
    }

def perturb(x, alpha=1.1):
    # generate random point in the unit hypersphere
    print(x, type(x))
    ndim = x.shape[0]
    random_point = np.random.normal(size=ndim)
    random_point /= np.linalg.norm(random_point)

    # scale and translate the point to fit the specified center and radius
    perturbed_point = x + alpha *  random_point

    return perturbed_point

def perturbn(x, alpha):
    """
    Randomly perturbs each element of x by adding noise from [-alpha, alpha].

    Args:
    - x (list or numpy array): The input array.
    - alpha (float): The range of noise to add. The noise is drawn from the interval [-alpha, alpha].

    Returns:
    - list: The perturbed array.
    """
    perturbed_x = [elem + random.uniform(-alpha, alpha) for elem in x]
    return perturbed_x

def perturb_percent(x, percent=0.1, ps=None):
    if ps is None:
        raise Exception("require parameters ps")
    out = np.zeros(x.shape)
    for i in range(len(x)):
        span = ps[i]['max'] - ps[i]['min']
        low = -span*percent
        high = span*percent
        r = np.random.uniform(low=low, high=high, size=1)
        out[i] = x[i] + r
        out[i] = max(ps[i]['min'], out[i])
        out[i] = min(ps[i]['max'], out[i])
    return out


def b(costf=None, parameters=None, perturb_pc=0.1, iterations=2, N=100, M=10, max_time=-1, debug=False):
    if costf is None:
        raise Exception("costf is a required kwarg")
    if parameters is None:
        raise Exception("parameters is a required kwarg")
    it_best_costs = []
    it_best_params = []
    it_params = []
    start_time = time.perf_counter()
    best_cost = None
    best_params = None
    times = []
    current_time = 0
    params = []
    costs = []
    it = 0
    while (it < N or N < 0) and (current_time < max_time or max_time < 0):
        it += 1
        ps = gen_params(parameters)
        cost = costf(ps)
        params.append(ps)
        costs.append(cost)
        if best_cost is None or cost < best_cost:
            best_params = ps
            best_cost = cost
        it_best_costs.append(best_cost)
        it_best_params.append(best_params)
        it_params.append(ps)
        current_time = time.perf_counter() - start_time
        times.append(current_time)
        if debug:
            print("parameters:", ps, end="\t")
            print("cost:", cost, end="\t")
            print("best cost:", best_cost)
    bests = best_m(params, costs, M=M)

    for i in range(iterations - 1):
        params = []
        costs = []
        it = 0
        while it < N and (current_time < max_time or max_time < 0):
            it += 1
            choice = random.choice(bests[0])
            new_params = perturb_percent(choice, percent=perturb_pc, ps=parameters)
            new_cost = costf(choice)
            params.append(new_params)
            it_params.append(new_params)
            costs.append(new_cost)
            if new_cost < best_cost:
                best_cost = new_cost
                best_params = new_params
            it_best_costs.append(best_cost)
            it_best_params.append(best_params)
            current_time = time.perf_counter() - start_time
            times.append(current_time)
        bests = best_m(params + list(bests[0]), costs + list(bests[1]), M=M)

    return {
        "results": {
            "best_params": best_params.tolist(),
            "best_cost": best_cost,
        },
        "stats": {
            "it_best_costs": it_best_costs,
            "it_best_params": list(map(lambda x: x.tolist(), it_best_params)),
            "it_params": list(map(lambda x: x.tolist(), it_params)),
            "time": times,
        }
    }


if __name__ == "__main__":
    # costf = lib.f_real
    # parameters=[{"min": 0, "max": 20},{"min": 0, "max": 20}]
    # N=1000
    # out = b(costf=costf, iterations=30, parameters=parameters, N=N, M=300, debug=False, alpha=5)
    # print(out['results']['best_params'])

    x = np.array([0, 0])
    print(x, perturb_percent(x, percent=0.5, ps=[{'min': 0, 'max': 20},{'min': 0, 'max': 20}]))
```