```python
import global_random_search
import lib
import numpy as np
import sgd
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
import pandas as pd
import time
import json

f = {
    "function": lib.f_real,
    "gradient": lib.f_grad,
    "dname": "$f(x)$",
    "name": "f",
    "alpha": 0.0065,
}

g = {
    "function": lib.g_real,
    "gradient": lib.g_grad,
    "dname": "$g(x)$",
    "name": "g",
    "alpha": 0.003,
}


def gradient_descent_constant(step_size=0.0065, start=[0, 0], funcs=f, max_iter=20000):
    start = np.array(start)
    g = sgd.StochasticGradientDescent()
    g.step_size(step_size)
    g.start(start)
    def function_generator():
        while True:
            yield funcs["function"], funcs["gradient"]
    g.function_generator(function_generator())
    g.debug(True)
    g.alg("constant")
    it = 0
    while it < max_iter:
        it += 1
        g.step()
        yield {
                "x": g._x_value,
        }


custom_lines = [
        Line2D([0], [0], color='purple', lw=2),
        Line2D([0], [0], color='blue', lw=2),
        Line2D([0], [0], color='orange', lw=2),
        Line2D([0], [0], color='black', lw=2),
        ]
custom_labels = ['rnd search b_mod', 'rnd search b', 'rnd search a', 'gradient descent']

def vis_results(results):
    def f(x, y):
        return 3 * (x - 5)**4 + 10 * (y - 9)**2
    def g(x, y):
        return np.maximum(x - 5, 0) + 10 * np.abs(y - 9)

    x = np.linspace(0, 10, 400)
    y = np.linspace(0, 18, 400)
    X, Y = np.meshgrid(x, y)
    Z_f = f(X, Y)
    Z_g = g(X, Y)

    fig = plt.figure(figsize=(12, 6))

    axf = fig.add_subplot(1, 2, 1)
    axf.contour(X, Y, Z_f, levels=30, cmap='viridis')
    axf.set_title('$f(x, y)$')
    axf.set_xlabel('$x$')
    axf.set_ylabel('$y$')

    axg = fig.add_subplot(1, 2, 2)
    axg.contour(X, Y, Z_g, levels=30, cmap='viridis')
    axg.set_title('$g(x, y)$')
    axg.set_xlabel('$x$')
    axg.set_ylabel('$y$')

    for b_results in results['f']['b']:
        x_coords = [point[0] for point in b_results['stats']['it_best_params']]
        y_coords = [point[1] for point in b_results['stats']['it_best_params']]
        axf.plot(x_coords, y_coords, linestyle='-', label="rndsearch b", color='orange')
    for b_results in results['g']['b']:
        x_coords = [point[0] for point in b_results['stats']['it_best_params']]
        y_coords = [point[1] for point in b_results['stats']['it_best_params']]
        axg.plot(x_coords, y_coords, linestyle='-', label="rndsearch b", color='orange')
    for a_results in results['f']['a']:
        x_coords = [point[0] for point in a_results['stats']['it_best_params']]
        y_coords = [point[1] for point in a_results['stats']['it_best_params']]
        axf.plot(x_coords, y_coords, linestyle='-', label="rndsearch a", color='blue')
    for a_results in results['g']['a']:
        x_coords = [point[0] for point in a_results['stats']['it_best_params']]
        y_coords = [point[1] for point in a_results['stats']['it_best_params']]
        axg.plot(x_coords, y_coords, linestyle='-', label="rndsearch a", color='blue')

    axf.legend()
    axg.legend()
    plt.tight_layout()
    plt.savefig("fig/bii-contours.pdf")

    return axf, axg

max_time=1
if __name__ == "__main__":
    all_results = {}
    for funcs in f, g:
        res = list(gradient_descent_constant(funcs=funcs, step_size=funcs["alpha"]))


        plt.figure()
        results = {
                "b_mod": [],
                "b": [],
                "a": [],
        }
        res = pd.DataFrame(res)
        res["f(x)"] = res["x"].apply(funcs["function"])

        for i in range(5):
            # ps = [{"min": 0, "max": 10}, {"min": 0, "max": 18}]
            # grs = global_random_search.b_mod(
            #     costf=funcs["function"], iterations=100, parameters=ps, N=1000, M=100, max_time=max_time)
            # costs = grs['stats']['it_best_costs']
            # plt.plot(grs['stats']['time'], costs, label="rnd search b_mod")
            # print(funcs["name"], "total iterations global random search b_mod: ", len(grs['stats']['time']))
            ps = [{"min": 0, "max": 10}, {"min": 0, "max": 18}]
            grs = global_random_search.b_mod(
                costf=funcs["function"], iterations=100, parameters=ps, N=20, M=10)
            costs = grs['stats']['it_best_costs']
            plt.plot(list(range(len(costs))), costs, label="rnd search b_mod", color="purple")
            results["b_mod"].append(grs)
            print(funcs["name"], "total iterations global random search b_mod: ", len(grs['stats']['time']))

            ps = [{"min": 0, "max": 10}, {"min": 0, "max": 18}]
            grs = global_random_search.b(
                costf=funcs["function"], iterations=250, parameters=ps, perturb_pc=0.0001, N=400, M=100)
            costs = grs['stats']['it_best_costs']
            plt.plot(list(range(len(costs))), costs, label="rnd search b", color="blue")
            results["b"].append(grs)
            print(funcs["name"], "total iterations global random search b: ", len(grs['stats']['time']))

            ps = [{"min": 0, "max": 10}, {"min": 0, "max": 18}]
            grs = global_random_search.a(
                costf=funcs["function"], parameters=ps, N=100000)
            costs = grs['stats']['it_best_costs']
            plt.plot(list(range(len(costs))), costs, label="rnd search a", color="orange")
            results["a"].append(grs)
            print(funcs["name"], "total iterations global random search a: ", len(grs['stats']['time']))


        plt.plot(list(range(len(res["f(x)"]))), res["f(x)"], label="gradient descent", color="black")
        plt.title(f"Global Random Search vs Gradient Descent on {funcs['dname']}")
        plt.legend(custom_lines, custom_labels, loc='lower right')
        plt.yscale('log')
        plt.xlabel("function/gradient evals")
        plt.ylabel(funcs['dname'])
        plt.tight_layout()
        plt.savefig(f"fig/bii-evals-{funcs['name']}.pdf")
        print(funcs["name"], "total iterations gradient descent: ", len(res))

        all_results[funcs['name']] = results
```