```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_time=1):
    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")
    start_time = time.perf_counter()
    current_time = 0
    while current_time < max_time:
        current_time = time.perf_counter() - start_time
        g.step()
        yield {
                "f(x)": g._function(g._x_value),
                "x": g._x_value,
                "time": time.perf_counter() - start_time,
        }


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 thin(array, step = 1):
    return [array[i] for i in range(0, len(array), step)]

def vis_results(results, args):
    print("starting vis")
    params = thin(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) if args.function == "f" else g(X,Y)
    fig = plt.figure(figsize=(4, 4))

    axf = fig.add_subplot(1, 1, 1)
    axf.contourf(X, Y, Z_f, levels=30, cmap='viridis')
    axf.set_title(args.title)
    axf.set_xlabel('$x$')
    axf.set_ylabel('$y$')
    x_coords, y_coords = zip(*thin(params, step=args.thin))
    # y_coords = thin([point[1] for point in params], step=args.thin)
    cmap = plt.cm.Blues
    color = [cmap(i / len(x_coords)) for i in range(len(x_coords))]
    for x,y,c in zip(x_coords, y_coords, color):
        print(".", end="", flush=True)
        axf.scatter(x, y, s=3, color=c)
    plt.tight_layout()


if __name__ == "__main__":
    import argparse
    ap = argparse.ArgumentParser()
    ap.add_argument("-i", type=str)
    ap.add_argument("-o", type=str)
    ap.add_argument("--title", type=str)
    ap.add_argument("--function", type=str)
    ap.add_argument("--thin", type=int, default=20)
    args = ap.parse_args()
    results = None
    with open(args.i, "r") as f:
        results = json.load(f)
    vis_results(results, args)
    print("saving fig")
    plt.savefig(args.o)
```