```python
 1: import global_random_search
 2: import lib
 3: import numpy as np
 4: import sgd
 5: import matplotlib.pyplot as plt
 6: import pandas as pd
 7: import time
 8: import cifar_costf
 9: import json
10: import argparse
11: import c_vis
12: import even_samples
13: import math
14: import cps
15: ps = cps.ps
16:
17: ap = argparse.ArgumentParser()
18: # ap.add_argument("--exp", type=str, required=True)
19: ap.add_argument("--M", type=int, required=True)
20: ap.add_argument("--N", type=int, required=True)
21: ap.add_argument("--n", type=int, required=True)
22: ap.add_argument("--iterations", type=int, required=True)
23: args = ap.parse_args()
24:
25: f = {
26:     "function": lib.f_real,
27:     "gradient": lib.f_grad,
28:     "dname": "$f(x)$",
29:     "name": "f",
30:     "alpha": 0.0065,
31: }
32:
33: g = {
34:     "function": lib.g_real,
35:     "gradient": lib.g_grad,
36:     "dname": "$g(x)$",
37:     "name": "g",
38:     "alpha": 0.003,
39: }
40:
41:
42: def gradient_descent_constant(step_size=0.0065, start=[0, 0], funcs=f, max_time=1):
43:     start = np.array(start)
44:     g = sgd.StochasticGradientDescent()
45:     g.step_size(step_size)
46:     g.start(start)
47:     def function_generator():
48:         while True:
49:             yield funcs["function"], funcs["gradient"]
50:     g.function_generator(function_generator())
51:     g.debug(True)
52:     g.alg("constant")
53:     start_time = time.time()
54:     current_time = 0
55:     while current_time < max_time:
56:         current_time = time.time() - start_time
57:         g.step()
58:         yield {
59:                 "f(x)": g._function(g._x_value),
60:                 "x": g._x_value,
61:                 "time": time.time() - start_time,
62:         }
63:
64: if __name__ == "__main__":
65:
66:     train, test = even_samples.even_sample_categories(math.floor(args.n))
67:
68:     def costf(x):
69:         return cifar_costf.costf(x, train, test)
70:
71:     grs = global_random_search.b(
72:         debug=True,
73:         costf=costf, parameters=ps, N=args.N, M=args.M, iterations=args.iterations)
74:
75:     fname = f"data/c-b-N{args.N}-M{args.M}-n{args.n}-it{args.iterations}.json"
76:     save = {
77:         'results': grs,
78:         'param-limits': ps,
79:         'args': vars(args),
80:         'name': None,
81:     }
82:     with open(fname, "w") as f:
83:         json.dump(save, f)
```