

```
1: import global_random_search
2: import lib
3: import numpy as np
4: import sgd
5: import matplotlib.pyplot as plt
6: from matplotlib.lines import Line2D
7: import pandas as pd
8: import time
9: import json
10:
11: f = {
12:     "function": lib.f_real,
13:     "gradient": lib.f_grad,
14:     "dname": "$f(x)$",
15:     "name": "f",
16:     "alpha": 0.0065,
17: }
18:
19: g = {
20:     "function": lib.g_real,
21:     "gradient": lib.g_grad,
22:     "dname": "$g(x)$",
23:     "name": "g",
24:     "alpha": 0.003,
25: }
26:
27:
28: def gradient_descent_constant(step_size=0.0065, start=[0, 0], funcs=f, max_iter=20000):
29:     start = np.array(start)
30:     g = sgd.StochasticGradientDescent()
31:     g.step_size(step_size)
32:     g.start(start)
33:     def function_generator():
34:         while True:
35:             yield funcs["function"], funcs["gradient"]
36:     g.function_generator(function_generator())
37:     g.debug(True)
38:     g.alg("constant")
39:     it = 0
40:     while it < max_iter:
41:         it += 1
42:         g.step()
43:         yield {
44:             "x": g._x_value,
45:         }
46:
47:
48: custom_lines = [
49:     Line2D([0], [0], color='purple', lw=2),
50:     Line2D([0], [0], color='blue', lw=2),
51:     Line2D([0], [0], color='orange', lw=2),
52:     Line2D([0], [0], color='black', lw=2),
53: ]
54: custom_labels = ['rnd search b_mod', 'rnd search b', 'rnd search a', 'gradient descent']
55:
56: def vis_results(results):
57:     def f(x, y):
58:         return 3 * (x - 5)**4 + 10 * (y - 9)**2
59:     def g(x, y):
60:         return np.maximum(x - 5, 0) + 10 * np.abs(y - 9)
61:
62:     x = np.linspace(0, 10, 400)
63:     y = np.linspace(0, 18, 400)
64:     X, Y = np.meshgrid(x, y)
65:     Z_f = f(X, Y)
66:     Z_g = g(X, Y)
67:
68:     fig = plt.figure(figsize=(12, 6))
69:
70:     axf = fig.add_subplot(1, 2, 1)
71:     axf.contourf(X, Y, Z_f, levels=30, cmap='viridis')
72:     axf.set_title('$f(x, y)$')
73:     axf.set_xlabel('$x$')
74:     axf.set_ylabel('$y$')
75:
76:     axg = fig.add_subplot(1, 2, 2)
77:     axg.contourf(X, Y, Z_g, levels=30, cmap='viridis')
78:     axg.set_title('$g(x, y)$')
79:     axg.set_xlabel('$x$')
80:     axg.set_ylabel('$y$')
81:
82:     for b_results in results['f']['b']:
83:         x_coords = [point[0] for point in b_results['stats']['it_best_params']]
84:         y_coords = [point[1] for point in b_results['stats']['it_best_params']]
85:         axf.plot(x_coords, y_coords, linestyle='--', label="rndsearch b", color='orange')
86:     for b_results in results['g']['b']:
87:         x_coords = [point[0] for point in b_results['stats']['it_best_params']]
88:         y_coords = [point[1] for point in b_results['stats']['it_best_params']]
89:         axg.plot(x_coords, y_coords, linestyle='--', label="rndsearch b", color='orange')
90:     for a_results in results['f']['a']:
91:         x_coords = [point[0] for point in a_results['stats']['it_best_params']]
92:         y_coords = [point[1] for point in a_results['stats']['it_best_params']]
93:         axf.plot(x_coords, y_coords, linestyle='--', label="rndsearch a", color='blue')
94:     for a_results in results['g']['a']:
95:         x_coords = [point[0] for point in a_results['stats']['it_best_params']]
96:         y_coords = [point[1] for point in a_results['stats']['it_best_params']]
97:         axg.plot(x_coords, y_coords, linestyle='--', label="rndsearch a", color='blue')
98:
99:     axf.legend()
100:     axg.legend()
```

```
101:     plt.tight_layout()
102:     plt.savefig("fig/bii-contours.pdf")
103:
104:     return axf, axg
105:
106: max_time=1
107: if __name__ == "__main__":
108:     all_results = {}
109:     for funcs in f, g:
110:         res = list(gradient_descent_constant(funcs=funcs, step_size=funcs["alpha"]))
111:
112:
113:     plt.figure()
114:     results = {
115:         "b_mod": [],
116:         "b": [],
117:         "a": [],
118:     }
119:     res = pd.DataFrame(res)
120:     res["f(x)"] = res["x"].apply(funcs["function"])
121:
122:     for i in range(5):
123:         # ps = [{"min": 0, "max": 10}, {"min": 0, "max": 18}]
124:         # grs = global_random_search.b_mod(
125:         #     costf=funcs["function"], iterations=100, parameters=ps, N=1000, M=100, max_time=max_time)
126:         # costs = grs['stats']['it_best_costs']
127:         # plt.plot(grs['stats']['time'], costs, label="rnd search b_mod")
128:         # print(funcs["name"], "total iterations global random search b_mod: ", len(grs['stats']['time']))
129:         ps = [{"min": 0, "max": 10}, {"min": 0, "max": 18}]
130:         grs = global_random_search.b_mod(
131:             costf=funcs["function"], iterations=100, parameters=ps, N=20, M=10)
132:         costs = grs['stats']['it_best_costs']
133:         plt.plot(list(range(len(costs))), costs, label="rnd search b_mod", color="purple")
134:         results["b_mod"].append(grs)
135:         print(funcs["name"], "total iterations global random search b_mod: ", len(grs['stats']['time']))
136:
137:         ps = [{"min": 0, "max": 10}, {"min": 0, "max": 18}]
138:         grs = global_random_search.b(
139:             costf=funcs["function"], iterations=250, parameters=ps, perturb_pc=0.0001, N=400, M=100)
140:         costs = grs['stats']['it_best_costs']
141:         plt.plot(list(range(len(costs))), costs, label="rnd search b", color="blue")
142:         results["b"].append(grs)
143:         print(funcs["name"], "total iterations global random search b: ", len(grs['stats']['time']))
144:
145:         ps = [{"min": 0, "max": 10}, {"min": 0, "max": 18}]
146:         grs = global_random_search.a(
147:             costf=funcs["function"], parameters=ps, N=100000)
148:         costs = grs['stats']['it_best_costs']
149:         plt.plot(list(range(len(costs))), costs, label="rnd search a", color="orange")
150:         results["a"].append(grs)
151:         print(funcs["name"], "total iterations global random search a: ", len(grs['stats']['time']))
152:
153:
154:     plt.plot(list(range(len(res["f(x)"]))), res["f(x)"], label="gradient descent", color="black")
155:     plt.title(f"Global Random Search vs Gradient Descent on {funcs['dname']}")
156:     plt.legend(custom_lines, custom_labels, loc='lower right')
157:     plt.yscale('log')
158:     plt.xlabel("function/gradient evals")
159:     plt.ylabel(funcs['dname'])
160:     plt.tight_layout()
161:     plt.savefig(f"fig/bii-evals-{funcs['name']}.pdf")
162:     print(funcs["name"], "total iterations gradient descent: ", len(res))
163:
164:     all_results[funcs['name']] = results
```