

```
1: import numpy as np
2: import lib
3: import time
4: import random
5:
6: def gen_params(parameters):
7:     p = np.zeros(len(parameters), dtype=np.float64)
8:     for i, par in enumerate(parameters):
9:         mini = par["min"]
10:        maxi = par["max"]
11:        p[i] = np.random.uniform(mini, maxi)
12:    return p
13:
14:
15: def a(costf=None, parameters=None, N=100, max_time=-1, debug=False):
16:     if costf is None:
17:         raise Exception("costf is a required kwarg")
18:     if parameters is None:
19:         raise Exception("parameters is a required kwarg")
20:     best_params = None
21:     best_cost = None
22:     it_best_costs = []
23:     it_best_params = []
24:     it_params = []
25:     start_time = time.perf_counter()
26:     times = []
27:     it = 0
28:     if max_time > 0:
29:         N = -1
30:     current_time = 0
31:     while (it < N or N < 0) and (current_time < max_time or max_time < 0):
32:         it += 1
33:         ps = gen_params(parameters)
34:         cost = costf(ps)
35:         if best_cost is None or np.isnan(best_cost) or cost < best_cost:
36:             best_params = ps
37:             best_cost = cost
38:             it_best_costs.append(best_cost)
39:             it_best_params.append(best_params)
40:             it_params.append(ps)
41:             current_time = time.perf_counter() - start_time
42:             times.append(current_time)
43:             if debug:
44:                 print("parameters:", ps, end="\t")
45:                 print("cost:", cost, end="\t")
46:                 print("best cost:", best_cost)
47:     return {
48:         "results": {
49:             "best_params": best_params.tolist(),
50:             "best_cost": best_cost,
51:         },
52:         "stats": {
53:             "it_best_costs": it_best_costs,
54:             "it_best_params": list(map(lambda x: x.tolist(), it_best_params)),
55:             "it_params": list(map(lambda x: x.tolist(), it_params)),
56:             "time": times,
57:         }
58:     }
59:
60: def best_m(params, costs, M=10, unzip=True):
61:     bests = sorted(zip(params, costs), key=lambda x: x[1])
62:     best_M = bests[:M]
63:     if unzip:
64:         return list(zip(*best_M))
65:     return best_M
66:
67: def bests2parameters(bests):
68:     params = bests[0]
69:     p1 = params[0]
70:     ps = []
71:     for i in range(len(p1)):
72:         param_values = list(map(lambda x: x[i], params))
73:         ps.append({
74:             "min": min(param_values),
75:             "max": max(param_values),
76:         })
77:     return ps
78:
79: def b_mod(costf=None, parameters=None, iterations=2, N=100, M=10, max_time=-1, debug=False):
80:     if costf is None:
81:         raise Exception("costf is a required kwarg")
82:     if parameters is None:
83:         raise Exception("parameters is a required kwarg")
84:     it_best_costs = []
85:     it_best_params = []
86:     it_params = []
87:     start_time = time.perf_counter()
88:     best_cost = None
89:     best_params = None
90:     times = []
91:     current_time = 0
92:     iteration_results = []
93:     for i in range(iterations):
94:         if debug:
95:             print("iteration: ", i)
96:         if max_time > 0 and current_time > max_time:
97:             break
98:         params = []
99:         costs = []
100:        it = 0
```

```

101:         while it < N:
102:             it += 1
103:             ps = gen_params(parameters)
104:             cost = costf(ps)
105:             params.append(ps)
106:             costs.append(cost)
107:             if best_cost is None or cost < best_cost:
108:                 best_params = ps
109:                 best_cost = cost
110:             it_best_costs.append(best_cost)
111:             it_best_params.append(best_params)
112:             it_params.append(ps)
113:             current_time = time.perf_counter() - start_time
114:             times.append(current_time)
115:             if debug:
116:                 print("parameters:", ps, end="\t")
117:                 print("cost:", cost, end="\t")
118:                 print("best cost:", best_cost)
119:             bests = best_m(params, costs, M=M)
120:             parameters = bests2parameters(bests)
121:             iteration_results.append({
122:                 "M": M,
123:                 "best_m_params": list(map(lambda x: x.tolist(), bests[0])),
124:                 "best_m_costs": bests[1],
125:                 "best_params": best_params.tolist(),
126:                 "best_cost": best_cost,
127:             })
128:         return {
129:             "results": {
130:                 "best_params": best_params.tolist(),
131:                 "best_cost": best_cost,
132:             },
133:             "stats": {
134:                 "it_best_costs": it_best_costs,
135:                 "it_best_params": list(map(lambda x: x.tolist(), it_best_params)),
136:                 "it_params": list(map(lambda x: x.tolist(), it_params)),
137:                 "time": times,
138:             },
139:             "iteration_results": iteration_results,
140:         }
141:
142: def perturb(x, alpha=1.1):
143:     # generate random point in the unit hypersphere
144:     print(x, type(x))
145:     ndim = x.shape[0]
146:     random_point = np.random.normal(size=ndim)
147:     random_point /= np.linalg.norm(random_point)
148:
149:     # scale and translate the point to fit the specified center and radius
150:     perturbed_point = x + alpha * random_point
151:
152:     return perturbed_point
153:
154: def perturbn(x, alpha):
155:     """
156:     Randomly perturbs each element of x by adding noise from [-alpha, alpha].
157:
158:     Args:
159:     - x (list or numpy array): The input array.
160:     - alpha (float): The range of noise to add. The noise is drawn from the interval [-alpha, alpha].
161:
162:     Returns:
163:     - list: The perturbed array.
164:     """
165:     perturbed_x = [elem + random.uniform(-alpha, alpha) for elem in x]
166:     return perturbed_x
167:
168: def perturb_percent(x, percent=0.1, ps=None):
169:     if ps is None:
170:         raise Exception("require parameters ps")
171:     out = np.zeros(x.shape)
172:     for i in range(len(x)):
173:         span = ps[i]['max'] - ps[i]['min']
174:         low = -span*percent
175:         high = span*percent
176:         r = np.random.uniform(low=low, high=high, size=1)
177:         out[i] = x[i] + r
178:         out[i] = max(ps[i]['min'], out[i])
179:         out[i] = min(ps[i]['max'], out[i])
180:     return out
181:
182:
183: def b(costf=None, parameters=None, perturb_pc=0.1, iterations=2, N=100, M=10, max_time=-1, debug=False):
184:     if costf is None:
185:         raise Exception("costf is a required kwarg")
186:     if parameters is None:
187:         raise Exception("parameters is a required kwarg")
188:     it_best_costs = []
189:     it_best_params = []
190:     it_params = []
191:     start_time = time.perf_counter()
192:     best_cost = None
193:     best_params = None
194:     times = []
195:     current_time = 0
196:     params = []
197:     costs = []
198:     it = 0
199:     while (it < N or N < 0) and (current_time < max_time or max_time < 0):
200:         it += 1

```

```
201:         ps = gen_params(parameters)
202:         cost = costf(ps)
203:         params.append(ps)
204:         costs.append(cost)
205:         if best_cost is None or cost < best_cost:
206:             best_params = ps
207:             best_cost = cost
208:         it_best_costs.append(best_cost)
209:         it_best_params.append(best_params)
210:         it_params.append(ps)
211:         current_time = time.perf_counter() - start_time
212:         times.append(current_time)
213:         if debug:
214:             print("parameters:", ps, end="\t")
215:             print("cost:", cost, end="\t")
216:             print("best cost:", best_cost)
217:     bests = best_m(params, costs, M=M)
218:
219:     for i in range(iterations - 1):
220:         params = []
221:         costs = []
222:         it = 0
223:         while it < N and (current_time < max_time or max_time < 0):
224:             it += 1
225:             choice = random.choice(bests[0])
226:             new_params = perturb_percent(choice, percent=perturb_pc, ps=parameters)
227:             new_cost = costf(choice)
228:             params.append(new_params)
229:             it_params.append(new_params)
230:             costs.append(new_cost)
231:             if new_cost < best_cost:
232:                 best_cost = new_cost
233:                 best_params = new_params
234:             it_best_costs.append(best_cost)
235:             it_best_params.append(best_params)
236:             current_time = time.perf_counter() - start_time
237:             times.append(current_time)
238:         bests = best_m(params + list(bests[0]), costs + list(bests[1]), M=M)
239:
240:     return {
241:         "results": {
242:             "best_params": best_params.tolist(),
243:             "best_cost": best_cost,
244:         },
245:         "stats": {
246:             "it_best_costs": it_best_costs,
247:             "it_best_params": list(map(lambda x: x.tolist(), it_best_params)),
248:             "it_params": list(map(lambda x: x.tolist(), it_params)),
249:             "time": times,
250:         }
251:     }
252:
253:
254: if __name__ == "__main__":
255:     # costf = lib.f_real
256:     # parameters=[{"min": 0, "max": 20}, {"min": 0, "max": 20}]
257:     # N=1000
258:     # out = b(costf=costf, iterations=30, parameters=parameters, N=N, M=300, debug=False, alpha=5)
259:     # print(out['results']['best_params'])
260:
261:     x = np.array([0, 0])
262:     print(x, perturb_percent(x, percent=0.5, ps=[{'min': 0, 'max': 20}, {'min': 0, 'max': 20}])))
```