```
src/lib.py
                   Sun Mar 24 15:59:08 2024
    1: import sympy as sp
    2: import numpy as np
    3: import functools
    4:
    5: x, y = sp.symbols('x y', real=True)
6: f = 3 * (x - 5)**4 + (10 * ((y - 9)**2))
7: g = sp.Max(x - 5, 0) + (10 * sp.Abs(y - 9))
    8: relu = sp.Max(x,0)
    9:
   10: def f_real(x, y):
11: return 3 * (x - 5)**4 + 10 * (y - 9)**2
   12:
   13:
   14: def g_real(x, y):
   15:
            return np.maximum(x - 5, 0) + 10 * np.abs(y - 9)
   17: def relu_real(x):
   18:
            return np.maximum(x,0)
   19:
   20:
   21: def apply_sym(x, f):
   22:
            for x_sym, x_val in zip(f.free_symbols, x):
                f = f.subs(x_sym, x_val)
   23:
   24:
            return f
   25:
   26: config = {
            "f": {
   27:
                "sym": f,
   28:
   29:
                "real": f_real,
                "name": "f",
   30:
   31:
            "g": {
   32:
                "sym": g,
   33:
                "real": g_real,
"name": "g",
   34:
   35:
   36:
            "relu": {
   37:
   38:
                "sym": relu,
   39:
                "real": lambda x: max(x, 0),
   40:
                "name": "relu",
   41:
            }
   42: }
   44: class GradientDescent():
   45:
           def __init__(self):
                self._max_iter = 1000
   47:
                self._debug = False
   48:
                self._converged = lambda x1, x2: False
                self.\_epsilon = 0.0001
   49:
                self._dimension = None
   50:
   51:
                self.\_beta = 0
   52:
                self._algorithm = None
   53:
                self._iteration = None
   54:
                self._function = None
   55:
                self.\_sum = None
   56:
                self._x_value = None
   57:
                self._step_coeff = None
   58:
                self._converged_value = None
   59:
                self._grad_value = None
                self._m = None
   60:
                self._v = None
   62:
                self._adam_grad = None
   63:
                self._beta = None
                self._beta2 = None
   65:
                self._step_size = None
                self._z = None
   66:
   67:
                self._f_star = None
   68:
   69:
            def step_size(self, a):
   70:
                self._step_size = a
                return self
   71:
   72:
   73:
            def beta(self, b):
   74:
                self.\_beta = b
   75:
                return self
   76:
   77:
            def beta2(self, b):
                self._beta2 = b
   78:
                return self
   79:
   80:
   81:
            def epsilon(self, e):
   82:
                self.\_epsilon = e
                return self
   83:
   84:
   85:
            def function(self, f, function_name=None, dimension=None):
   86:
                self.\_function = f
   87:
                self.function_name = function_name
   88:
                self. dimension = dimension
   89:
                return self
   90:
   91:
            def sym_function(self, function, function_name=None):
   92:
                self.function_name = function_name
   93:
                self._dimension = len(function.free_symbols)
   94:
                def fn(x):
   95:
                     return apply_sym(x, function)
   96:
                diffs = [function.diff(var) for var in function.free_symbols]
   97:
   98:
   99:
                def grad(x):
  100:
                     return np.array([
```

```
101:
                      apply_sym(x, diff) for diff in diffs])
102:
103:
             self.\_function = fn
104:
             self._gradient = grad
105:
             return self
106:
107:
         def gradient(self, g):
108:
             self._gradient = g
109:
             return self
110:
111:
         def max_iter(self, m):
112:
             self._max_iter = m
113:
             return self
114:
115:
         def start(self, s):
116:
             self.\_start = s
117:
             self._x_value = s
118:
             return self
119:
120:
         def debug(self, d):
121:
             self.\_debug = d
122:
             return self
123:
124:
         def converged(self, c):
125:
             self._converged = c
126:
             return self
127:
128:
         def set_iterate(self, f):
             self.iterate = functools.partial(f, self)
129:
130:
             return self
131:
132:
         def algorithm(self, alg):
133:
             self._algorithm = alg
             if self._algorithm == "rmsprop":
134:
135:
                  import rmsprop
136:
                  self.set_iterate(rmsprop.iterate)
137:
             elif self._algorithm == "adam":
138:
                 import adam
139:
                 self.set_iterate(adam.iterate)
140:
             elif self._algorithm == "heavy_ball":
141:
                 import heavy_ball
142:
                 self.set_iterate(heavy_ball.iterate)
143:
144:
                 raise Exception("Unknown algorithm:" + alg)
145:
             return self
146:
147:
         def state_dict(self):
148:
             print (self._function(self._x_value))
149:
             return {
150:
                 "alg": self._algorithm,
                 "function_name": self.function_name,
151:
                  "iteration": self._iteration,
152:
                 "step_coeff": self._step_coeff,
153:
154:
                 "adam_grad": self._adam_grad,
155:
                 "f(x)": self._function(self._x_value),
                 "epsilon": self._epsilon,
156:
157:
                 "converged": self._converged_value,
                 "gradient": self._grad_value,
158:
                 "m": self._m,
159:
                 "v": self._v,
160:
                 "beta1": self._beta,
161:
                 "beta2": self._beta2,
162:
                 "alpha": self._step_size,
163:
                 "sum": self._sum,
164:
                 "z": self._z,
165:
                  **{"x" + str(i): self._x_value[i] for i in range(len(self._x_value))},
166:
167:
             }
168:
169:
         def run2csv(self, fname, summarise=True):
170:
             import pandas as pd
171:
             iterations = list(self.iterate())
172:
             df = pd.DataFrame(iterations)
173:
             df.to_csv(fname)
174:
             if(summarise):
175:
                 with open(fname + ".summary", "w") as f:
                      print(f"iterations: {len(df)}", file=f)
176:
                      print(f"start: {df['x0'][0]} {df['x1'][0]}", file=f)
177:
                      print(f"final: {df['x0'][len(df) - 1]} {df['x1'][len(df) - 1]}", file=f)
178:
179:
180:
                == "__main_
181: if ___name_
182:
        print(f.diff(x), f.diff(y))
183:
        print(q.diff(x), q.diff(y))
```

src/lib.py

Sun Mar 24 15:59:08 2024