```
import sympy as sp
import numpy as np
import functools

x, y = sp.symbols('x y', real=True)
f = 3 * (x - 5)**4 + (10 * ((y - 9)**2))
g = sp.Max(x - 5, 0) + (10 * sp.Abs(y - 9))
relu = sp.Max(x,0)

def f_real(xv):
    return 3 * (xv[0] - 5)**4 + 10 * (xv[1] - 9)**2

f_diff_x = f.diff(x)
f_diff_y = f.diff(y)
def f_grad(xv):
    return np.array([
        f_diff_x.subs(x, xv[0]).subs(y, xv[1]),
        f_diff_y.subs(x, xv[0]).subs(y, xv[1]),
    ])

g_diff_x = f.diff(x)
g_diff_y = f.diff(y)
def g_grad(xv):
    return np.array([
        g_diff_x.subs(x, xv[0]).subs(y, xv[1]),
        g_diff_y.subs(x, xv[0]).subs(y, xv[1]),
    ])

def g_real(xv):
    return np.maximum(xv[0] - 5, 0) + 10 * np.abs(xv[1] - 9)


def apply_sym(x, f):
    for x_sym, x_val in zip(f.free_symbols, x):
        f = f.subs(x_sym, x_val)
    return f

config = {
    "f": {
        "sym": f,
        "real": f_real,
        "name": "f",
    },
    "g": {
        "sym": g,
        "real": g_real,
        "name": "g",
    },
    "relu": {
        "sym": relu,
        "real": lambda x: max(x, 0),
        "name": "relu",
    }
}

class GradientDescent():
    def __init__(self):
        self._max_iter = 1000
        self._debug = False
        self._converged = lambda x1, x2: False
        self._epsilon = 0.0001
        self._dimension = None
        self._beta = 0
        self._algorithm = None
        self._iteration = None
        self._function = None
        self._sum = None
        self._x_value = None
        self._step_coeff = None
        self._converged_value = None
        self._grad_value = None
        self._m = None
        self._v = None
        self._adam_grad = None
        self._beta = None
        self._beta2 = None
        self._step_size = None
        self._z = None
        self._f_star = None

    def step_size(self, a):
        self._step_size = a
        return self

    def beta(self, b):
        self._beta = b
        return self

    def beta2(self, b):
        self._beta2 = b
        return self

    def epsilon(self, e):
        self._epsilon = e
        return self

    def function(self, f, function_name=None, dimension=None):
        self._function = f
        self.function_name = function_name
        self._dimension = dimension
```

```
101:         return self
102:
103:     def sym_function(self, function, function_name=None):
104:         self.function_name = function_name
105:         self._dimension = len(function.free_symbols)
106:         def fn(x):
107:             return apply_sym(x, function)
108:
109:         diffs = [function.diff(var) for var in function.free_symbols]
110:
111:         def grad(x):
112:             return np.array([
113:                 apply_sym(x, diff) for diff in diffs])
114:
115:         self._function = fn
116:         self._gradient = grad
117:         return self
118:
119:     def gradient(self, g):
120:         self._gradient = g
121:         return self
122:
123:     def max_iter(self, m):
124:         self._max_iter = m
125:         return self
126:
127:     def start(self, s):
128:         self._start = s
129:         return self
130:
131:     def debug(self, d):
132:         self._debug = d
133:         return self
134:
135:     def converged(self, c):
136:         self._converged = c
137:         return self
138:
139:     def set_iterate(self, f):
140:         self.iterate = functools.partial(f, self)
141:         return self
142:
143:     def algorithm(self, alg):
144:         self._algorithm = alg
145:         if self._algorithm == "rmsprop":
146:             import rmsprop
147:             self.set_iterate(rmsprop.iterate)
148:         elif self._algorithm == "adam":
149:             import adam
150:             self.set_iterate(adam.iterate)
151:         elif self._algorithm == "heavy_ball":
152:             import heavy_ball
153:             self.set_iterate(heavy_ball.iterate)
154:         else:
155:             raise Exception("Unknown algorithm:" + alg)
156:         return self
157:
158:     def state_dict(self):
159:         print(self._function(self._x_value))
160:         return {
161:             "alg": self._algorithm,
162:             "function_name": self.function_name,
163:             "iteration": self._iteration,
164:             "step_coeff": self._step_coeff,
165:             "adam_grad": self._adam_grad,
166:             "f(x)": self._function(self._x_value),
167:             "epsilon": self._epsilon,
168:             "converged": self._converged_value,
169:             "gradient": self._grad_value,
170:             "m": self._m,
171:             "v": self._v,
172:             "beta1": self._beta,
173:             "beta2": self._beta2,
174:             "alpha": self._step_size,
175:             "sum": self._sum,
176:             "z": self._z,
177:             **{"x" + str(i): self._x_value[i] for i in range(len(self._x_value))},
178:         }
179:
180:     def run2csv(self, fname, summarise=True):
181:         import pandas as pd
182:         iterations = list(self.iterate())
183:         df = pd.DataFrame(iterations)
184:         df.to_csv(fname)
185:         if(summarise):
186:             with open(fname + ".summary", "w") as f:
187:                 print(f"iterations: {len(df)}", file=f)
188:                 print(f"start: {df['x0'][0]} {df['x1'][0]}", file=f)
189:                 print(f"final: {df['x0'][len(df) - 1]} {df['x1'][len(df) - 1]}", file=f)
190:
191:
192: if __name__ == "__main__":
193:     print(f.diff(x), f.diff(y))
194:     print(g.diff(x), g.diff(y))
```