COMP 543, Tools and Models for Data Science

Chengyin Liu, cl93

Homework #2, Gradient descent algorithm and Newton's method

```
import numpy as np
import math

def f(x, y):
    return math.sin(x + y) + (x - y) ** 2 - 1.5 * x + 2.5 * y + 1

#Gradient descent algorithm
def dfdx(x, y):
    return math.cos(x + y) + 2 * (x - y) - 1.5

def dfdy(x, y):
    return math.cos(x + y) - 2 * (x - y) + 2.5
#df = lambda x, y: np.array([[math.cos(x + y) + 2 * (x - y) - 1.5], [math.cos(x + y) - 2 * (x - y) +
2.5]])

def gd_optimize(a):
    x = a
    lr = 1
    E = 10e-20
    f_last = 0
    f_cur = f(x[0], x[1])
    e = float("inf")
    while e > E:
        x[0] = x[0] - lr * dfdx(x[0], x[1])
        x[1] = x[1] - lr * dfdy(x[0], x[1])
        f_last = f_cur
        f_cur = f(x[0], x[1])
        print f_cur
        e = abs(f_cur - f_last)
        if f_cur > f_last:
            lr *= 0.5
        elif f_cur < f_last:
            lr *= 1.1
    print x


gd_optimize (np.array([-0.2, -1.0]))
'''
-0.299618028565
-1.24020432371
-1.28271337878
-1.56898405697
-1.74876107958
```

```
-1.69678830301
-1.88549468525
-1.91274889866
-1.91317888424
-1.91321966004
-1.91322203521
-1.91322237308
-1.91322254094
-1.91322248845
-1.91322288211
-1.91322295476
-1.91322295494
-1.91322295498
-1.91322295498
-1.91322295498
-1.91322295498
-1.91322295498
-1.91322295498
-1.91322295498
-1.91322295498
-1.91322295498
-1.91322295498
-1.91322295498
-1.91322295498
-1.91322295498
-1.91322295498
-1.91322295498
-1.91322295498
-1.91322295498
-1.91322295498
-1.91322295498
[-0.54719755 -1.54719755]
'''

gd_optimize (np.array([-0.5, -1.5]))
'''
-1.89300412135
-1.90631138843
-1.91072608172
-1.91213679222
-1.91235958337
-1.91224104229
-1.91319950058
-1.91322288322
-1.91322293645
-1.91322295225
-1.91322295451
-1.91322295474
-1.91322295479
-1.91322295476
-1.91322295496
```

```
-1.91322295498
-1.91322295498
-1.91322295498
-1.91322295498
-1.91322295498
-1.91322295498
[-0.54719755 -1.54719755]
'''

#Newton's method
def dfdxdx(x, y):
    return -math.sin(x + y) + 2

def dfdxdy(x, y):
    return -math.sin(x + y) - 2

def dfdydy(x, y):
    return -math.sin(x + y) + 2

def hessian(x, y):
    return np.array([[dfdxdx(x, y), dfdxdy(x, y)], [dfdxdy(x, y), dfdydy(x, y)]])

def nm_optimize(a):
    x = a
    E = 10e-20
    f_last = 0
    f_cur = f(x[0], x[1])
    e = float("inf")
    while e > E:
        dL = np.array([dfdx(x[0], x[1]), dfdy(x[0], x[1])])
        x = x - np.dot(np.linalg.inv(hessian(x[0], x[1])), dL)
        f_last = f_cur
        f_cur = f(x[0], x[1])
        print f_cur
        e = abs(f_cur - f_last)
    print x


nm_optimize (np.array ([-0.2, -1.0]))
'''
-1.91281352075
-1.91322291866
-1.91322295498
-1.91322295498
[-0.54719755 -1.54719755]
'''

nm_optimize (np.array ([-0.5, -1.5]))
'''
-1.91322090085
-1.91322295498
```

```
-1.91322295498
-1.91322295498
-1.91322295498
[-0.54719755 -1.54719755]
'''
```