Comp 543 HW3

Kai-Po Lin

【Description】Implement Gradient Descent and Newton Method to minimum the function of the given point

【Code】

```
import math
import numpy as np

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

def dfx(x, y):
    return math.cos(x+y) + 2 * (x-y) - 1.5

def dfy(x, y):
    return math.cos(x+y) - 2 * (x-y) + 2.5

def dfx2(x, y):
    return -math.sin(x+y) + 2

def dfy2(x, y):
    return -math.sin(x+y) + 2

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

def gd_optimize(a):
    learningRate = 1.0
    lossPrev = abs(f(a[0], a[1]))
    lossDiff = 1.0
    resultCur = np.zeros(2)
    resultPrev = a

    while(abs(lossDiff) > 10e-20):
        resultCur = resultPrev - learningRate * np.array([dfx(resultPrev[0], resultPrev[1]),
dfy(resultPrev[0], resultPrev[1])])

        lossCur = f(resultCur[0], resultCur[1])
```

```
        print(lossCur)

        lossDiff = lossCur - lossPrev
        # If the value of the loss increases in an iteration, then we half the learning rate
        if(lossDiff > 0):
            learningRate *= 0.5
        # If the value of the loss decreases in an iteration, then we multiply 1.1 on the learning rate
        else:
            learningRate *= 1.1

        lossPrev = lossCur
        resultPrev = resultCur
    print(resultCur)

def procedurenm_optimize(a):
    lossPrev = abs(f(a[0], a[1]))
    lossDiff = 1.0
    resultCur = np.zeros(2)
    resultPrev = np.transpose(a)

    while(abs(lossDiff) > 10e-20):
        x = resultPrev[0]
        y = resultPrev[1]
        resultCur = resultPrev - np.dot(np.linalg.inv([[dfx2(x, y), dfxfy(x, y)], [dfxfy(x, y), dfy2(x, y)]]),
                                        np.transpose([dfx(x, y), dfy(x, y)]))
        lossCur = f(resultCur[0], resultCur[1])
        print(lossCur)

        lossDiff = lossCur - lossPrev
        lossPrev = lossCur
        resultPrev = resultCur
    print(resultCur)

if __name__ == '__main__':
    print(">>> gd_optimize(np.array([-0.2, -1.0]))")
    gd_optimize(np.array([-0.2, -1.0]))
    print("\n>>> gd_optimize(np.array([-0.5, -1.5]))")
    gd_optimize(np.array([-0.5, -1.5]))
    print("\n\n>>> procedurenm_optimize(np.array([-0.2, -1.0]))")
    procedurenm_optimize(np.array([-0.2, -1.0]))
    print("\n>>> procedurenm_optimize(np.array([-0.5, -1.5]))")
    procedurenm_optimize(np.array([-0.5, -1.5]))
```

【Result】

```
> python .\HW3.py
>>> gd_optimize(np.array([-0.2, -1.0]))
-1.3175387318156826
2.2697639236744944
4.079481528079684
-1.8532957824166747
-1.9105801424619937
-1.9129334036957086
-1.9131605893134562
-1.913199710099247
-1.9132091171905095
-1.913210500085209
-1.9132066665260998
-1.913222871023228
-1.9132229522917301
-1.913222954744021
-1.9132229549379107
-1.9132229549670634
-1.9132229549736688
-1.913222954975085
-1.9132229549739836
-1.9132229549810225
-1.9132229549810362
-1.9132229549810367
-1.9132229549810362
-1.9132229549810367
-1.9132229549810362
-1.9132229549810367
-1.9132229549810367
[-0.54719755 -1.54719755]

>>> gd_optimize(np.array([-0.5, -1.5]))
-1.9109295805761808
-1.9114681674883558
-1.9110297007042236
-1.9132215281704674
-1.9132229214706045
-1.913222952576786
-1.9132229546063524
-1.9132229548741102
```

```
-1.9132229549304762
-1.9132229549439543
-1.9132229549407707
-1.9132229549810185
-1.9132229549810362
-1.9132229549810358
-1.9132229549810367
-1.9132229549810362
-1.9132229549810367
-1.9132229549810367
[-0.54719755 -1.54719755]


>>> procedurenm_optimize(np.array([-0.2, -1.0]))
-1.9128135207487107
-1.9132229186591214
-1.9132229549810362
-1.9132229549810362
[-0.54719755 -1.54719755]

>>> procedurenm_optimize(np.array([-0.5, -1.5]))
-1.9132209008539096
-1.913222954980231
-1.9132229549810362
-1.9132229549810367
-1.9132229549810367
[-0.54719755 -1.54719755]
```