Homework 04

一、 实验目标：

实现 Levenberg-Marquardt 算法

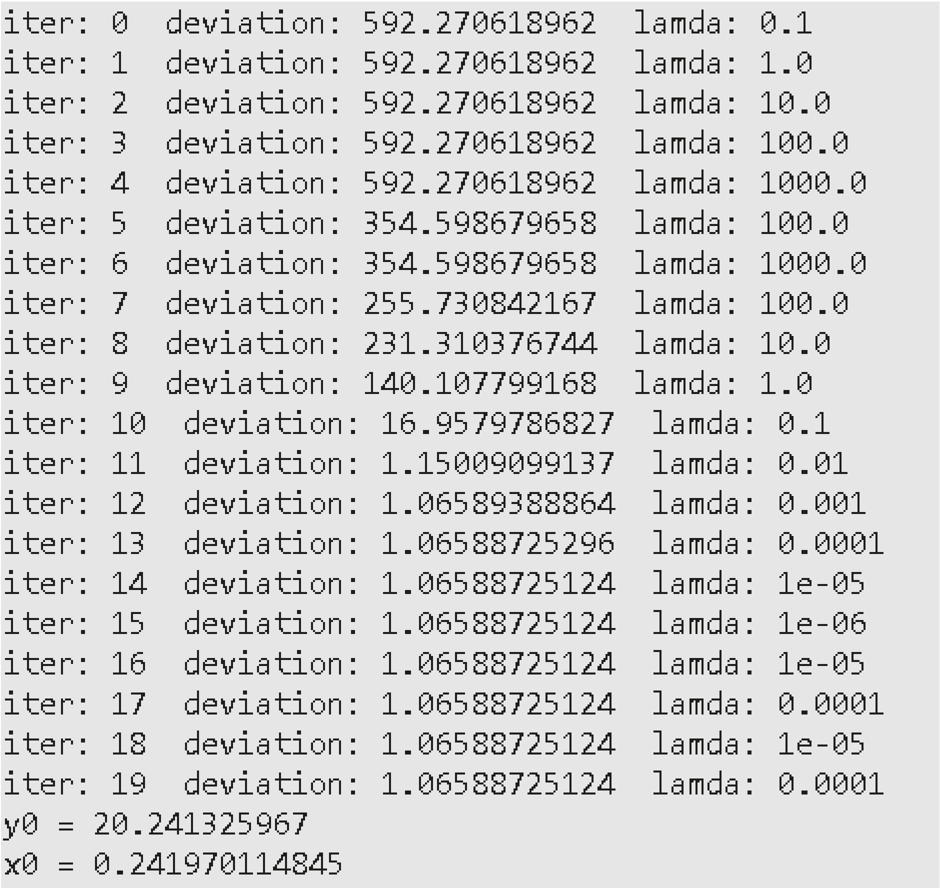
二、 实验过程：

第一步，创建拟合用数据，对于数据集，猜测初始 x0，y0 值。将 LM 算法的阻尼系数初始设置为 0.01

第二步，开始迭代，第一次迭代强制采用牛顿法，通过雅各布矩阵，得到新的 y_fit，并计算出误差。根据阻尼系数计算出步长后，x0, y0 移动步长，更新误差。根据误差，决定参数和阻尼系数的更新。

第三步，输出迭代结果。

三、 实验结果：



```
iter: 0  deviation: 592.270618962  lamda: 0.1
iter: 1  deviation: 592.270618962  lamda: 1.0
iter: 2  deviation: 592.270618962  lamda: 10.0
iter: 3  deviation: 592.270618962  lamda: 100.0
iter: 4  deviation: 592.270618962  lamda: 1000.0
iter: 5  deviation: 354.598679658  lamda: 100.0
iter: 6  deviation: 354.598679658  lamda: 1000.0
iter: 7  deviation: 255.730842167  lamda: 100.0
iter: 8  deviation: 231.310376744  lamda: 10.0
iter: 9  deviation: 140.107799168  lamda: 1.0
iter: 10  deviation: 16.9579786827  lamda: 0.1
iter: 11  deviation: 1.15009099137  lamda: 0.01
iter: 12  deviation: 1.06589388864  lamda: 0.001
iter: 13  deviation: 1.06588725296  lamda: 0.0001
iter: 14  deviation: 1.06588725124  lamda: 1e-05
iter: 15  deviation: 1.06588725124  lamda: 1e-06
iter: 16  deviation: 1.06588725124  lamda: 1e-05
iter: 17  deviation: 1.06588725124  lamda: 0.0001
iter: 18  deviation: 1.06588725124  lamda: 1e-05
iter: 19  deviation: 1.06588725124  lamda: 0.0001
y0 = 20.241325967
x0 = 0.241970114845
```

从迭代过程可以开出，当阻尼因子较小时，根据 H_lm = H + (lamda * np.identity(dim))，步长主要由 Hessian matrix 得到，x0，y0 移动步长后，发现误差变大，不采用该次移动，lamda = lamda * 10。 从第 6 次迭代到第 14 次迭代，采用牛顿法，误差大幅减小的同时，不断减小阻尼系数，直到最后趋于稳定。

四、 实验代码及注释：

```
import numpy as np

#fitting data, from 'Mathematical Experiment'
x = [0.25, 0.5, 1, 1.5, 2, 3, 4, 6, 8]
y = [19.21, 18.15, 15.36, 14.10, 12.89, 9.32, 7.45, 5.24, 3.01]

#assumption
x0 = 0.5
y0 = 10
y_init = y0 * np.exp([-x0 * w for w in x])
```

```
num = len(x)
dim = 2
iterationMax = 20
#initial lamda for L-M
lamda = 0.01

#assignment
update = 1

#iteration
for i in range(iterationMax):
    if update == 1:
        #calculate Jacobi matrix
        JacobiM = np.zeros(num* dim).reshape(num, dim)
        for j in range(num):
            JacobiM[j, :] = [np.exp(-x0 * x[j]), -y0 * x[j] * np.exp(-x0 * x[j])]
        #calculate new y(y_fit)
        y_fit = y0 * np.exp([-x0 * w for w in x])
        #calculate distance
        dis = y - y_fit
        #Hessian matrix
        H = np.dot(JacobiM.T, JacobiM)
        #calculate deviation
        if i==0:
            deviation = np.dot(dis, dis)
    H_lm = H + (lamda * np.identity(dim))
    #calculate step length
    step = np.dot(np.mat(H_lm).I, np.dot(JacobiM.T, dis[:]))
    g = np.dot(JacobiM.T, dis[:])
    #try to move x0, y0
    y_lm = y0 + step[0, 0]
    x_lm = x0 + step[0, 1]
    y_fit_lm = y_lm * np.exp([-x_lm * w for w in x])
    #update deviation
    dis_lm = y - y_fit_lm
    deviation_lm = np.dot(dis_lm, dis_lm)

    #update parameters and lamda depending on deviation_lm
    if deviation_lm < deviation:
        lamda = lamda / 10
        y0 = y_lm
        x0 = x_lm
        deviation = deviation_lm
        update = 1
    else:
        lamda = lamda * 10
        update = 0
    print "iter: " + str(i) +"    deviation: " + str(deviation) + "    lamda: " + str(lamda)
print "y0 = " + str(y0)
print "x0 = " + str(x0)
```