作业四： Levenberg-Marquardt

实验目的： 实现 Levenberg-Marquardt(LM)方法。

实验过程：

LM方法是使用最广泛的非线性最小二乘算法，对函数关系x=f(p)，给定f(·)与含噪声的观测向量x，估计使得函数值最小的参数向量p。LM方法是介于牛顿法与梯度下降法之间的一种非线性优化方法，对于过参数化问题不敏感，能有效处理冗余参数问题，使代价函数陷入局部极小值的机会大大减小。

计算步骤如下图所示：

步骤 1 取初始点 p_0，终止控制常数 ε，计算 $\varepsilon_0=\|x-f(p_0)\|$，$k:=0$，$\lambda_0=10^{-3}$，$\nu=10$（也可以是其他大于 1 的数）。

步骤 2 计算 Jacobi 矩阵 J_k，计算 $\bar{N}_k=J_k^{\mathrm{T}}J_k+\lambda_k I$，构造增量正规方程 $\bar{N}_k\cdot\delta_k=J_k^{\mathrm{T}}\varepsilon_k$。

步骤 3 求解增量正规方程得到 δ_k

（1）如果 $\|x-f(p_k+\delta_k)\|<\varepsilon_k$，则令 $p_{k+1}=p_k+\delta_k$，若 $\|\delta_k\|<\varepsilon$，停止迭代，输出结果；否则令 $\lambda_{k+1}=\lambda_k/\nu$，转到步骤 2。

（2）如果 $\|x-f(p_k+\delta_k)\|\geq\varepsilon_k$，则令 $\lambda_{k+1}=\nu\cdot\lambda_k$，重新解正规方程得到 δ_k，返回步骤（1）。

在具体的实现过程中初始定义循环50次，但在迭代过程中，若参数已经不再发生改变，则停止迭代，完成算法步骤。实验代码如下所示：

```
# -*- coding: utf-8 -*-
"""
Created on Mon May 30 13:15:26 2016

@author: Dell
"""

import numpy as np

#拟合用数据。
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]
```

```
#LM算法
#初始猜测s
a0 = 10
b0 = 0.5
data = []
for w in x:
    data = data + [-b0*w]
y_init = a0*np.exp(data)

#数据个数
datanum=len(y)
# 参数维数
paramdim=2
# 迭代最大次数
iterations=50
# LM算法的阻尼系数初值
lamda=0.01
# step1: 变量赋值
updateJ=1
a_tmp=a0
b_tmp=b0

#step2: 迭代
for i in range(iterations):
    if updateJ==1:
        # 根据当前估计值，计算雅克比矩阵
        J=np.zeros(datanum*paramdim).reshape(datanum,paramdim)
        for j in range(len(x)):
            J[j,:]=[np.exp(-b_tmp*x[j]),-a_tmp*x[j]*np.exp(-b_tmp*x[j])]
        # 根据当前参数，得到函数值
        tmp = []
        for w in x:
            tmp = tmp + [-b_tmp*w]
        y_tmp = a_tmp*np.exp(tmp)
        # 计算误差
        difference = y-y_tmp;
        # 计算（拟）海塞矩阵
        H = np.dot(J.T,J)
        # 若是第一次迭代，计算误差
        if i == 0:
            e=np.dot(difference,difference)

    # 根据阻尼系数lamda混合得到H矩阵
```

```
        H_lm = H+(lamda*np.eye(paramdim,paramdim));
        # 计算步长dp，并根据步长计算新的可能的\参数估计值
        dp = np.dot(np.mat(H_lm).I,np.dot(J.T,difference[:]))
        g = np.dot(J.T,difference[:])
        a_lm=a_tmp+dp[0,0]
        b_lm=b_tmp+dp[0,1]
        data2=[]
        for  w in x:
            data2 = data2 + [-b_lm*w]
        y_tmp_lm=a_lm*np.exp(data2)
        # 计算新的可能估计值对应的y和计算残差e
        difference_lm = y-y_tmp_lm
        e_lm=np.dot(difference_lm,difference_lm)

        if a_tmp == a_lm and b_tmp == b_lm:
            break

        # 根据误差，决定如何更新参数和阻尼系数
        if e_lm < e:
            lamda = lamda/10
            a_tmp = a_lm
            b_tmp = b_lm
            e = e_lm
            updateJ=1
        else:
            updateJ=0
            lamda=lamda*10
        print '第',i,'次迭代'
        print 'a=',a_tmp,'b=',b_tmp,'lamda=',lamda,'e=',e
```

实验结果：

```
第 0 次迭代
a= 10 b= 0.5 lamda= 0.1 e= 592.270618962
第 1 次迭代
a= 10 b= 0.5 lamda= 1.0 e= 592.270618962
第 2 次迭代
a= 10 b= 0.5 lamda= 10.0 e= 592.270618962
第 3 次迭代
a= 10 b= 0.5 lamda= 100.0 e= 592.270618962
第 4 次迭代
a= 10 b= 0.5 lamda= 1000.0 e= 592.270618962
第 5 次迭代
```

a= 10.0284872989 b= 0.217476988452 lamda= 100.0 e= 354.598679658

第 6 次迭代

a= 10.0284872989 b= 0.217476988452 lamda= 1000.0 e= 354.598679658

第 7 次迭代

a= 10.0553015165 b= 0.0247412124609 lamda= 100.0 e= 255.730842167

第 8 次迭代

a= 10.3292915848 b= 0.0644088764675 lamda= 10.0 e= 231.310376744

第 9 次迭代

a= 12.3383388159 b= 0.106022610769 lamda= 1.0 e= 140.107799168

第 10 次迭代

a= 17.5178391241 b= 0.207201685337 lamda= 0.1 e= 16.9579786827

第 11 次迭代

a= 20.0535124161 b= 0.240358796973 lamda= 0.01 e= 1.15009099137

第 12 次迭代

a= 20.2394193651 b= 0.241926359465 lamda= 0.001 e= 1.06589388864

第 13 次迭代

a= 20.2413033483 b= 0.241969240255 lamda= 0.0001 e= 1.06588725296

第 14 次迭代

a= 20.241325536 b= 0.241970097775 lamda= 1e-05 e= 1.06588725124

第 15 次迭代

a= 20.2413259588 b= 0.241970114518 lamda= 1e-06 e= 1.06588725124

第 16 次迭代

a= 20.2413259588 b= 0.241970114518 lamda= 1e-05 e= 1.06588725124

第 17 次迭代

a= 20.2413259588 b= 0.241970114518 lamda= 0.0001 e= 1.06588725124

第 18 次迭代

a= 20.241325967 b= 0.241970114845 lamda= 1e-05 e= 1.06588725124

第 19 次迭代

a= 20.241325967 b= 0.241970114845 lamda= 0.0001 e= 1.06588725124

第 20 次迭代

a= 20.241325967 b= 0.241970114845 lamda= 0.001 e= 1.06588725124

第 21 次迭代

a= 20.241325967 b= 0.241970114845 lamda= 0.01 e= 1.06588725124

第 22 次迭代

a= 20.241325967 b= 0.241970114845 lamda= 0.1 e= 1.06588725124

第 23 次迭代

a= 20.241325967 b= 0.241970114845 lamda= 1.0 e= 1.06588725124

第 24 次迭代

```
a= 20.241325967 b= 0.241970114845 lamda= 10.0 e= 1.06588725124
第 25 次迭代
a= 20.241325967 b= 0.241970114845 lamda= 100.0 e= 1.06588725124
第 26 次迭代
a= 20.241325967 b= 0.241970114845 lamda= 1000.0 e= 1.06588725124
第 27 次迭代
a= 20.241325967 b= 0.241970114845 lamda= 10000.0 e= 1.06588725124
第 28 次迭代
a= 20.241325967 b= 0.241970114845 lamda= 100000.0 e= 1.06588725124
第 29 次迭代
a= 20.241325967 b= 0.241970114845 lamda= 1000000.0 e= 1.06588725124
第 30 次迭代
a= 20.241325967 b= 0.241970114845 lamda= 10000000.0 e= 1.06588725124
第 31 次迭代
a= 20.241325967 b= 0.241970114845 lamda= 100000000.0 e= 1.06588725124
第 32 次迭代
a= 20.241325967 b= 0.241970114845 lamda= 1000000000.0 e= 1.06588725124
第 33 次迭代
a= 20.241325967 b= 0.241970114845 lamda= 100000000.0 e= 1.06588725124
第 34 次迭代
a= 20.241325967 b= 0.241970114845 lamda= 1000000000.0 e= 1.06588725124
第 35 次迭代
a= 20.241325967 b= 0.241970114845 lamda= 10000000000.0 e= 1.06588725124
```

实验小结：LM 算法的重要之处在于它为大参数化问题提供了快速收敛的正则化方法；对于过参数问题，LM 算法的性能也很优越。LM 算法同时具有梯度法和牛顿法的优点，在 LM 算法中，每次迭代是寻找一个合适的阻尼因子 λ，当 λ 很小时，算法就变成了 GAuss-Newton 法的最优步长计算式，λ 很大时，蜕化为梯度下降法的最优步长计算式。