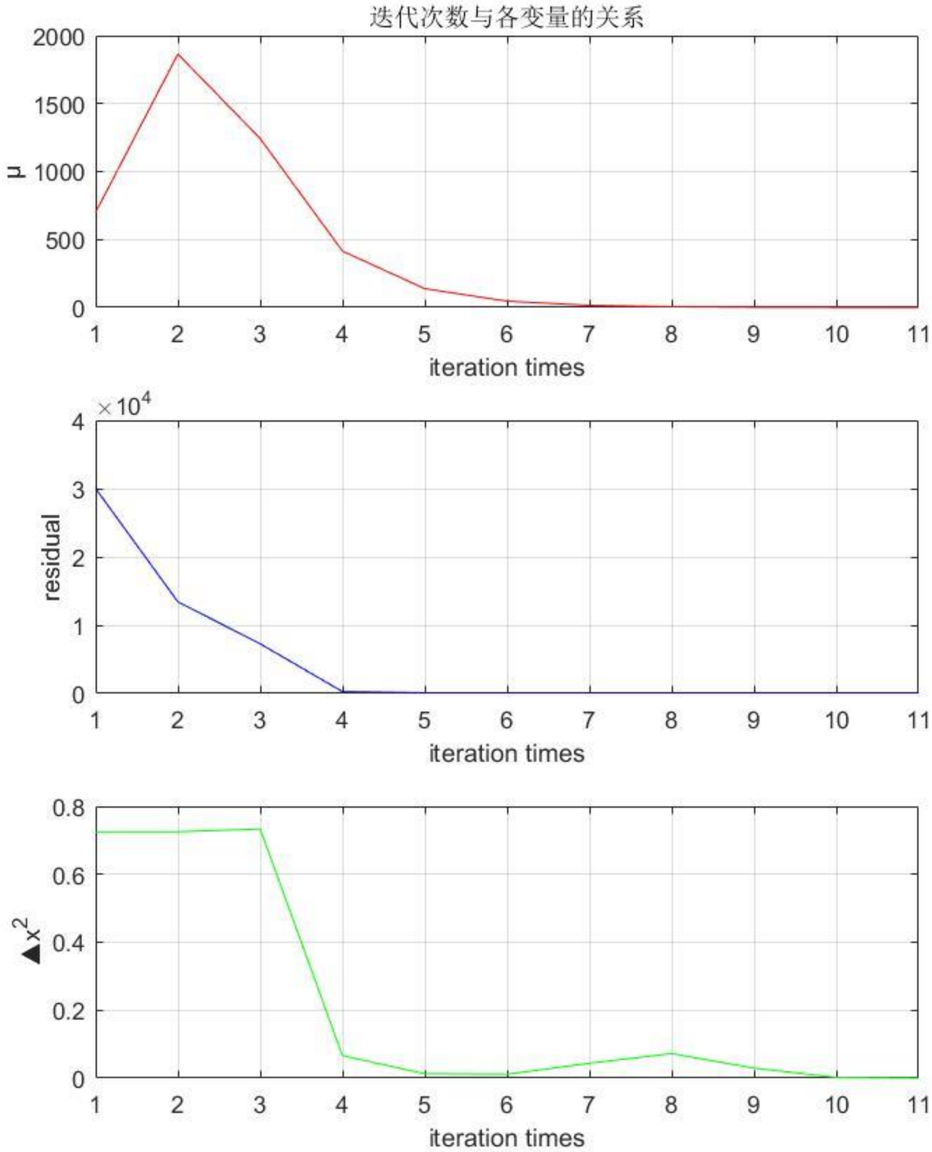
必做与选做习题

孙正茂

1.1 绘制样例代码中 LM 阻尼因子 μ 随着迭代变化的曲线图



从上图可以看出，一开始近似并不理想，所以μ开始增大，直到出现使残差下降的▲x 后，此时 LM 接近最速下降法，方向是最优的，只是步长小了许多；随着迭代次数增多，残差在逐渐减小，证明方向是好的，因此μ在第一次增加后开始减小，此时 LM 接近 GN 法，使▲x 有轻微的增加，随后开始接近局部极小值，▲x 开始减小，直到迭代结束。

1.2 将曲线函数改成 y=ax2+bx+c，请修改样例代码中残差计算，雅克比计算等函数，完成曲线参数估计。

按照样例代码只更改残差和雅可比，得：

```
residual_(0) = abc[0] * x_ * x_ + abc[1] * x_ + abc[2] - y_;//残差

jacobian_abc << x_ * x_ , x_ , 1;//雅可比
```

随后发现之迭代两次就完成了结束迭代，而且结果也不对，随即设定几种解决办法：

① ▲x<1e-6：将此阈值调小，发现▲x 特别小之后只是迭代次数增加了，x 加上特别小的▲x 后对 x 也没有影响，不可行；

②初始值μ调高：μ太大会导致▲x 变小，结果又回到原来的情况，不可行；

③提高噪声的标准差：发现结果更差了，考虑到可能和噪声将数据湮没的可能性；

④增加 Edge 的数量：发现迭代结果增加，结果比较好

⑤增大 x 的间隔：发现与第④种情况一样。

⑥增加鲁棒核函数：效果更差劲，数据已经漂飞了，当把二阶导去掉后效果有变好，至少比加上要强很多，但是也没有比原来更好，看来噪声过于大的话数据就不行了。下面分别是不加柯西核函数与加了的对比图，并增加了欧氏距离进行对比，使效果更直观。

a> Code:

```
// solve W and b_new
double s = edge.second->Residual().transpose() * edge.second->Residual();
double W = RobustKernel_p1(s) + 2 * RobustKernel_p2(s) * s * 0;//remove dd
hessian = W * hessian;//set new hessian
b.segment(index_i,dim_i).noalias()-
=RobustKernel_p1(s)*JtW*edge.second->Residual();//set new b

//sub function
double Problem::RobustKernel(double s)//rou
{    double rou = Cauchy_c * Cauchy_c * std::log(1 + s / Cauchy_c / Cauchy_c);
     return rou;}
```

```
double Problem::RobustKernel_p1(double s)//rou'
{     double rou_prime = 1 / (1 + s / (Cauchy_c * Cauchy_c));
      return rou_prime;}
double Problem::RobustKernel_p2(double s)//rou''
{     double rou_pp=(-1)*RobustKernel_p1(s)*RobustKernel_p1(s)
/(Cauchy_c*Cauchy_c);
      return rou_pp;}
```

b> 当噪声标准差比较小的时候：0.1

```
problem solve cost: 7.23838 ms
   makeHessian cost: 5.92294 ms
-------After optimization, we got these parameters :
 1.06202   1.9611 0.999621
-------ground truth:
1.0,  2.0,  1.0
加Cauthy核函数后与真实值的欧氏距离为: 0.0732118
```

```
problem solve cost: 6.78942 ms
   makeHessian cost: 5.45029 ms
-------After optimization, we got these parameters :
 1.06107  1.96183 0.999517
-------ground truth:
1.0,  2.0,  1.0
不加Cauthy核函数后与真实值的欧氏距离为: 0.0720172
```

c> 适当加大标准差：0.8

```
problem solve cost: 7.39699 ms
   makeHessian cost: 4.32392 ms
-------After optimization, we got these parameters :
1.66646 1.54789 1.01093
-------ground truth:
1.0,  2.0,  1.0
加Cauthy核函数后与真实值的欧氏距离为: 0.805412
```

```
problem solve cost: 2.60222 ms
   makeHessian cost: 2.03936 ms
-------After optimization, we got these parameters :
  1.4886  1.69449 0.996206
-------ground truth:
1.0,  2.0,  1.0
不加Cauthy核函数后与真实值的欧氏距离为: 0.576264
```

d> 将其中某些噪声放大十倍：

```
if((i % 10) == 0)
     n *= 10;
```

```
problem solve cost: 3.95113 ms
   makeHessian cost: 3.17123 ms
-------After optimization, we got these parameters :
 5.95538 -2.90201  1.82115
-------ground truth:
1.0,  2.0,  1.0
不加Cauthy核函数与真实值的欧氏距离为: 7.01853
```

```
problem solve cost: 9.72825 ms
   makeHessian cost: 6.40188 ms
-------After optimization, we got these parameters :
 1.16225  2.12403 0.889484
-------ground truth:
1.0,  2.0,  1.0
加Cauthy核函数与真实值的欧氏距离为: 0.232213
```

由上图对比知，当噪声是高斯白噪声的时候，柯西核函数的效果反而变差，但是差距不明显；但是当 outlier 变多的时候，核函数的作用显著增加。

总结：出现这种情况是因为噪声太大，湮没了数据，导致最终的优化值停留在噪声的局部极小值而无法出来；当存在一定量的噪声相对标准数据较小的值时，才能得到正确结果。

最终结果如下：

```
Test CurveFitting start...
iter: 0 , chi= 3.21386e+06 , Lambda= 19.95
iter: 1 , chi= 974.658 , Lambda= 13.3
iter: 2 , chi= 973.881 , Lambda= 8.86668
iter: 3 , chi= 973.88 , Lambda= 5.91112
iter: 4 , chi= 973.88 , Lambda= 3.94075
iter: 5 , chi= 973.88 , Lambda= 2.62716
iter: 6 , chi= 973.88 , Lambda= 1.75144
iter: 7 , chi= 973.88 , Lambda= 1.16763
problem solve cost: 78 ms
   makeHessian cost: 63.6479 ms
-------After optimization, we got these parameters :
0.999589  2.00628 0.968821
-------ground truth:
1.0,  2.0,  1.0
```

1.3（选做）

1.3.1 实现 Marquardt 更新策略

1、Code：

```
if(rho < 0.25)
{currentLambda_ *= 2;
return false;}
else
{
    if(rho > 0.75)
    {currentLambda_ = currentLambda_ / 3;
    return true;}
}
```

2、效果：

a>标准差为 0.5：

```
problem solve cost: 3.76201 ms
   makeHessian cost: 3.05017 ms
-------After optimization, we got these parameters :
 1.18321  1.88551 0.998551
-------ground truth:
1.0,  2.0,  1.0
Nielsen更新策略，与真实值的欧氏距离为: 0.216046
```

```
problem solve cost: 3.7544 ms
   makeHessian cost: 3.04945 ms
-------After optimization, we got these parameters :
 1.18321  1.88551 0.998551
-------ground truth:
1.0,  2.0,  1.0
Marquardt更新策略，与真实值的欧氏距离为: 0.216046
```

b>标准差为 0.8：

```
problem solve cost: 2.99819 ms
   makeHessian cost: 2.45638 ms
-------After optimization, we got these parameters :
  1.4886  1.69449 0.996206
-------ground truth:
1.0,  2.0,  1.0
Nielsen更新策略，与真实值的欧氏距离为: 0.576264
```

```
problem solve cost: 2.55234 ms
   makeHessian cost: 2.05544 ms
-------After optimization, we got these parameters :
  1.4886  1.69449 0.996206
-------ground truth:
1.0,  2.0,  1.0
Marquardt更新策略，与真实值的欧氏距离为: 0.576264
```

由上图知：无论标准差大或是小，两种迭代策略差异并不大，但是在一些特殊情况下（出现不必要的循环），Nielsen 表现要比 Marquardt 方法好.

1.3.2 new 更新策略

采用新的迭代方法进行实验，发现结果差不多，但是由于多了一些运算，使运行时间变长，参考的论文以及代码的修改以附件的形式放到另外两个文档里，此处只贴出对应参数的值与运行结果。

a>噪声标准差为 0.1：

```
problem solve cost: 5.06737 ms
   makeHessian cost: 4.09082 ms
-------After optimization, we got these parameters :
 1.06107  1.96184 0.999517
-------ground truth:
1.0,  2.0,  1.0
New方法与真实值的欧氏距离为: 0.0720149
```

```
problem solve cost: 3.86018 ms
   makeHessian cost: 3.05283 ms
-------After optimization, we got these parameters :
 1.06107  1.96183 0.999517
-------ground truth:
1.0,  2.0,  1.0
Nie方法与真实值的欧氏距离为: 0.0720172
```

b>噪声标准差为 0.3：

```
problem solve cost: 5.12229 ms
   makeHessian cost: 4.10354 ms
-------After optimization, we got these parameters :
 1.18323  1.88549 0.998555
-------ground truth:
1.0,  2.0,  1.0
New方法与真实值的欧氏距离为: 0.216075
```

```
problem solve cost: 3.9059 ms
   makeHessian cost: 3.08311 ms
-------After optimization, we got these parameters :
 1.18321  1.88551 0.998551
-------ground truth:
1.0,  2.0,  1.0
Nie方法与真实值的欧氏距离为: 0.216046
```

c>参数的选取：

```
double thita1 = 0.3;
double thita2 = 1-thita1;
double p0 = 0.73;
double p1 = 0.78;
double p2 = 0.93;
double p3 = 100;
double q1 = 2.43;
double q2 = 0.64;
double m  = 0.0003;
double a_new  = 1;
double b_new  = 0.5;
double rou_tilde = 0;
```

总结：由调试知：当噪声相对较大时（如 0.3），第一次的ρ接近 1，证明拟合的很好，然后ρ开始变大（到好几万），此时的实际值还在变，但是预测值已经进入局部最优值，即使 lambda 再变小，delta_x 也不会再变大了，直到结束迭代。

2、公式推导，根据课程知识，完成 F, G 中如下两项的推导过程：

$$g_{12}=\frac{1}{2}\frac{\partial\frac{1}{2}\left[q_{bibk}\otimes\begin{bmatrix}1\\ \frac{1}{2}w\delta t\end{bmatrix}\otimes\begin{bmatrix}1\\ \frac{1}{4}\delta n_k^g\delta t\end{bmatrix}(a^{b_{k+1}}-b_k^a)\delta t^2\right]}{\partial\delta n_k^g}$$

$$=\frac{1}{4}\frac{\partial\left[R_{bibk+1}\exp((\frac{1}{2}\delta n_k^g\delta t)_\times)(a^{b_{k+1}}-b_k^a)\delta t^2\right]}{\partial\delta n_k^g}$$

$$=\frac{1}{4}\frac{R_{bibk+1}(\frac{1}{2}\delta n_k^g\delta t)_\times(a^{b_{k+1}}-b_k^a)\delta t^2}{\delta n_k^g}$$

$$=-\frac{1}{4}\frac{R_{bibk+1}[(a^{b_{k+1}}-b_k^a)\delta t^2]_\times(\frac{1}{2}\delta n_k^g\delta t)}{\delta n_k^g}$$

$$=-\frac{1}{4}R_{bibk+1}[(a^{b_{k+1}}-b_k^a)\delta t^2]_\times(\frac{1}{2}\delta t)$$

$$f_{15}=\frac{1}{2}\frac{\partial\frac{1}{2}[q_{bibk}\otimes\begin{bmatrix}1\\ \frac{1}{2}w\delta t\end{bmatrix}\otimes\begin{bmatrix}1\\ -\frac{1}{2}\delta b_k^g\delta t\end{bmatrix}(a^{b_{k+1}}-b_k^a)\delta t^2]}{\partial\delta b_k^g}$$

$$=\frac{1}{4}\frac{\partial[R_{bibk+1}\exp((-\delta b_k^g\delta t)_\times)(a^{b_{k+1}}-b_k^a)\delta t^2]}{\partial\delta b_k^g}$$

$$=-\frac{1}{4}\frac{R_{bibk+1}[(-\delta b_k^g\delta t)_\times](a^{b_{k+1}}-b_k^a)\delta t^2}{\delta b_k^g}$$

$$=-\frac{1}{4}\frac{R_{bibk+1}((a^{b_{k+1}}-b_k^a)\delta t^2)_\times(-\delta b_k^g\delta t)}{\delta b_k^g}$$

$$=-\frac{1}{4}R_{bibk+1}(a^{b_{k+1}}-b_k^a)_\times\delta t^2(-\delta t)$$

3 证明式(9)

$$(J^TJ+\mu I)\Delta x = -(F')^T$$

$$\Rightarrow (V\Lambda V^T+\mu I)\Delta x = -F'^T$$

$$\Rightarrow V(\Lambda+\mu I)V^T\Delta x = -F'^T$$

$$\Rightarrow (\Lambda+\mu I)V^T\Delta x = -V^TF'^T$$

$V=[\vec{v}_1\ \vec{v}_2\ \cdots\ \vec{v}_n]$, $F'^T=J^Tf$

$n\times m\times m\times 1 = n\times 1$

$$V^T=\begin{bmatrix} v_1^T \\ \vdots \\ v_n^T \end{bmatrix}$$

$$\underset{n\times n}{\begin{bmatrix} \lambda_1+\mu & & & \\ & \lambda_2+\mu & & \\ & & \ddots & \\ & & & \lambda_n+\mu \end{bmatrix}}\begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_n^T \end{bmatrix}\Delta x = -\underset{n\times n}{\begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_n^T \end{bmatrix}}\underset{n\times 1}{F'^T} = -\underset{n\times 1}{\begin{bmatrix} v_1^TF'^T \\ v_2^TF'^T \\ \vdots \\ v_n^TF'^T \end{bmatrix}}$$

$$\therefore V^T\Delta x = -\begin{bmatrix} \frac{1}{\lambda_1+\mu} & & & \\ & \frac{1}{\lambda_2+\mu} & & \\ & & \ddots & \\ & & & \frac{1}{\lambda_n+\mu} \end{bmatrix}\begin{bmatrix} v_1^TF'^T \\ v_2^TF'^T \\ \vdots \\ v_n^TF'^T \end{bmatrix}$$

$$V^T\Delta x = -\begin{bmatrix} \frac{v_1^TF'^T}{\lambda_1+\mu} \\ \vdots \\ \frac{v_n^TF'^T}{\lambda_n+\mu} \end{bmatrix}$$

$$\therefore \Delta x = -[v_1\ v_2\ \cdots\ v_n]\begin{bmatrix} \frac{v_1^TF'^T}{\lambda_1+\mu} \\ \vdots \\ \frac{v_n^TF'^T}{\lambda_n+\mu} \end{bmatrix}$$

$$= -\sum_{j=1}^{n}\frac{v_j(v_j^TF'^T)}{\lambda_j+\mu}$$

$$= -\sum_{j=1}^{n}\frac{v_j^TF'^T}{\lambda_j+\mu}v_j$$ 得证.