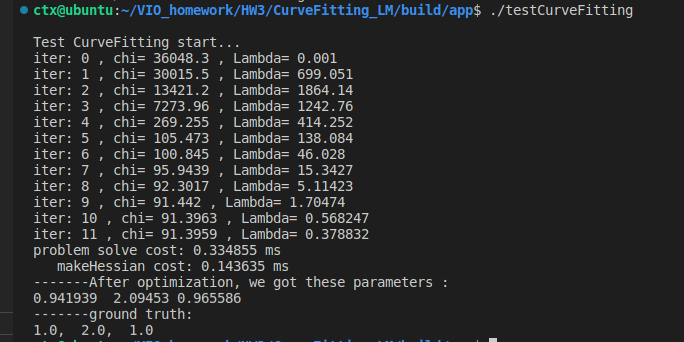
**1、**

**（1）绘制阻尼因子随迭代变化的曲线图**

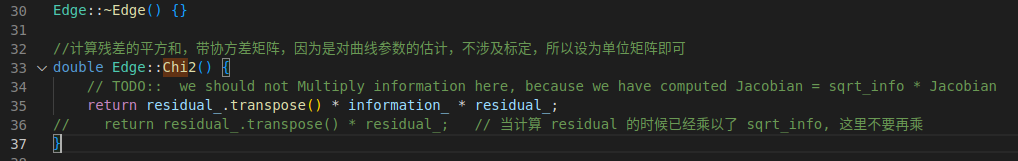
首先跑通样例代码，运行结果如下：



其中，iter是迭代次数，Lambda是阻尼因子

**chi是整个系统残差的平方和**

因为edge类中计算残差的函数里计算的是J^T\*I\*J

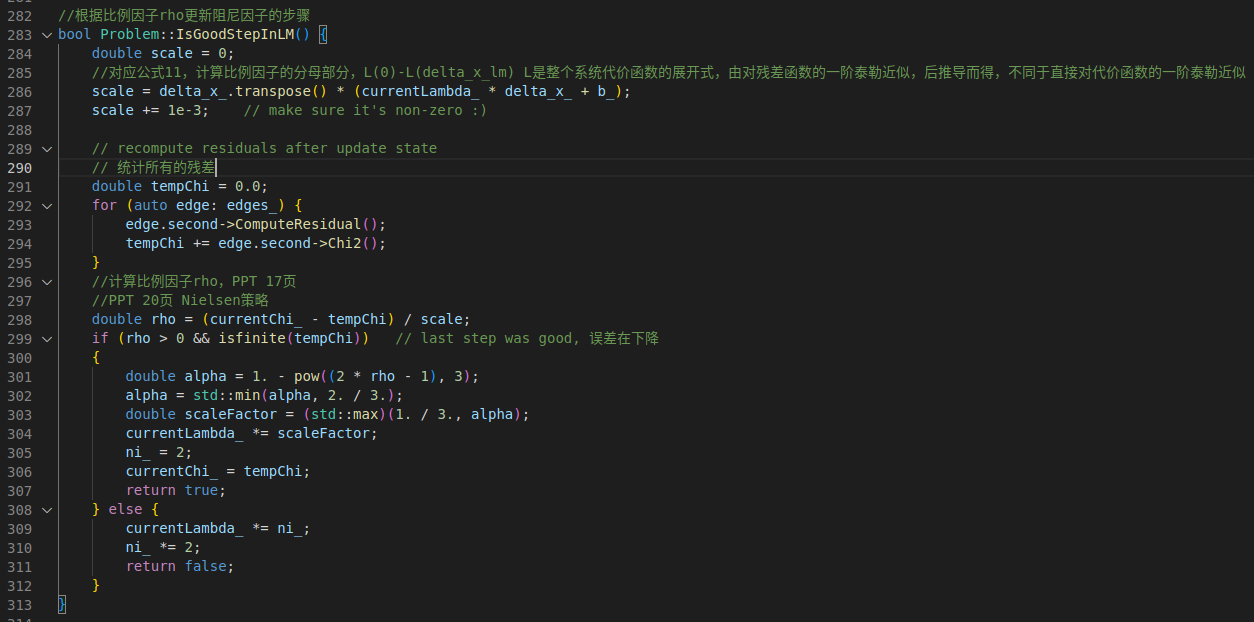


LM算法初始化函数

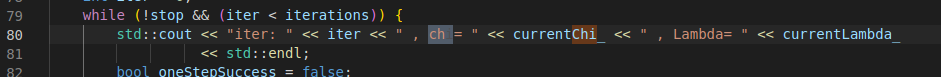
**屏幕上有字

描述已自动生成**

LM算法迭代更新步骤，将实时的残差平方和累加起来，赋值给currentChi\_

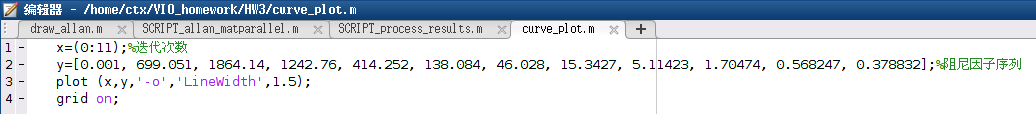
****

**输出的是currentChi\_成员变量**



在MATLAB中画图，横坐标x为迭代次数，纵坐标y为阻尼因子在各个迭代次数时的值

代码及运行结果如下：

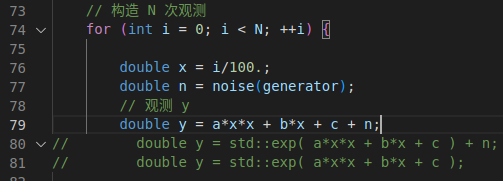


图表, 折线图

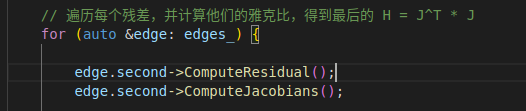
描述已自动生成

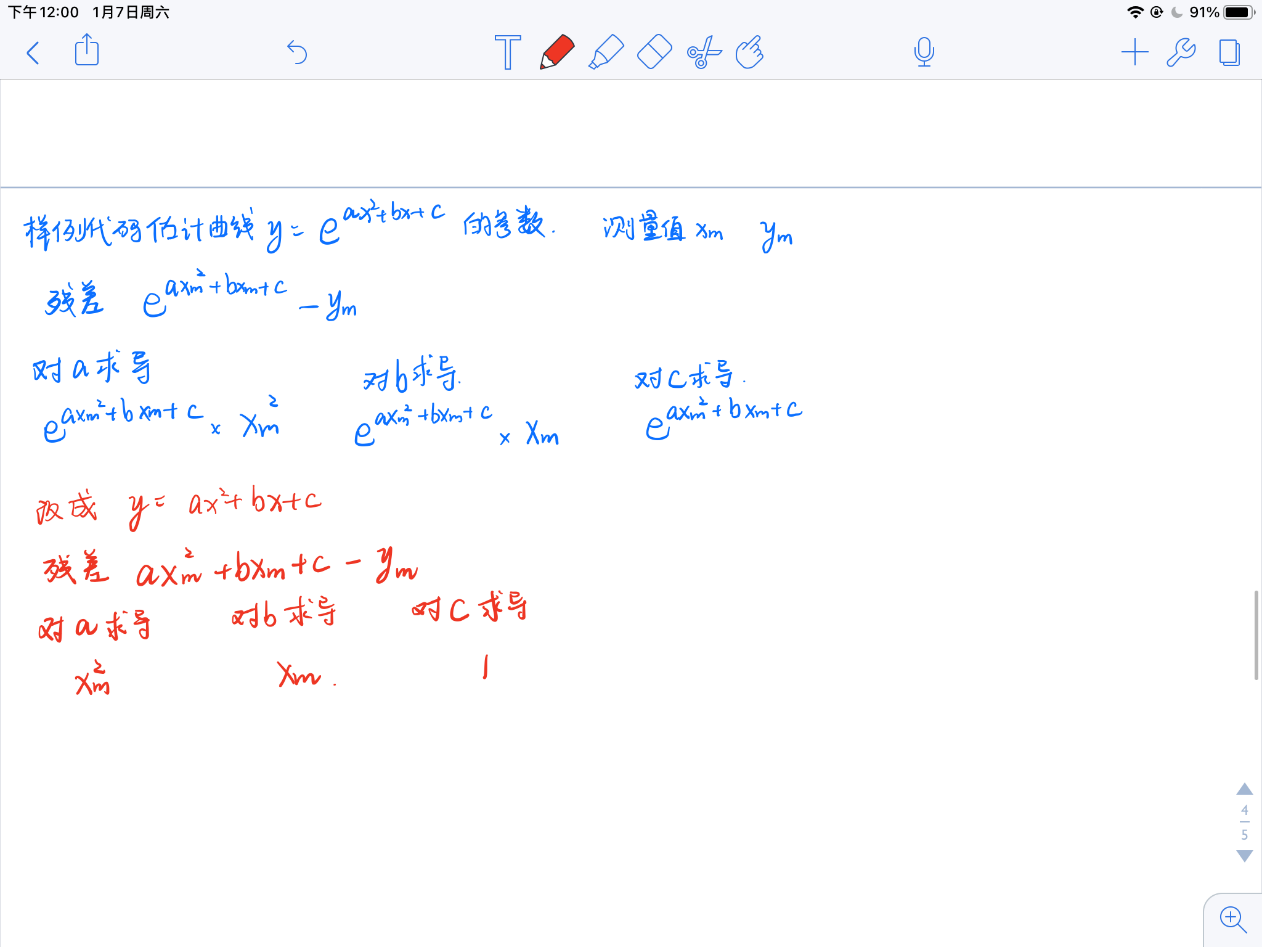
**（2）修改曲线函数，完成曲线参数估计**

首先修改生成测量值的函数：

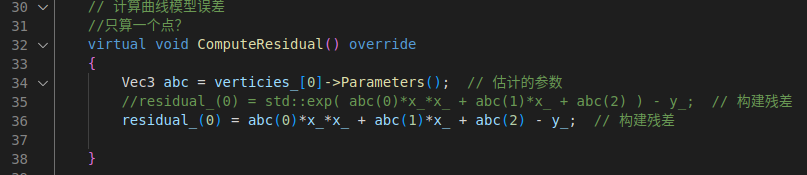


Main函数中调用Problem::Solve函数，Solve函数调用MakeHessian函数，MakeHessian函数中调用计算残差和雅可比的函数，对这两个函数进行修改





修改残差计算：



修改雅可比计算：

文本

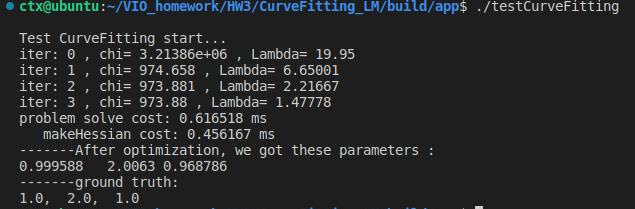
描述已自动生成

运行结果：

文本

描述已自动生成

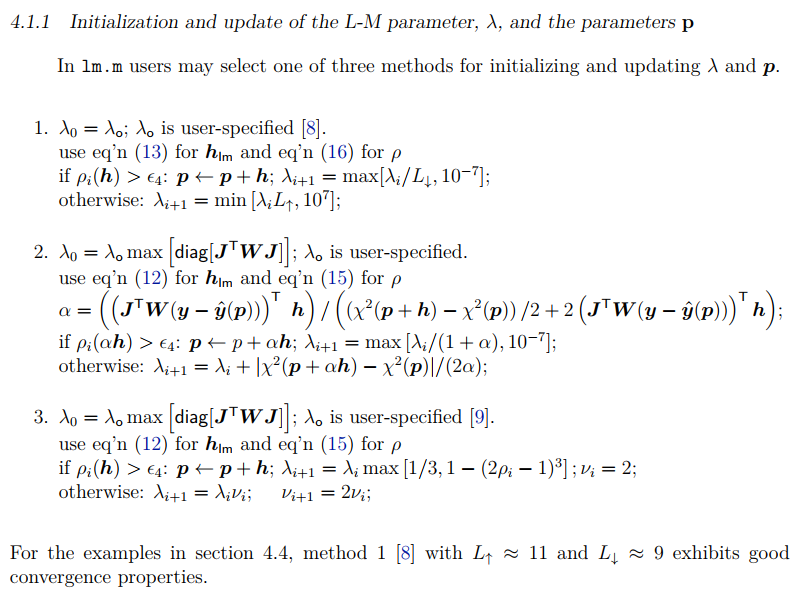
对a和b的估计结果不太理想，而且只迭代了两次，所以增加观测点数量，运行结果如下：



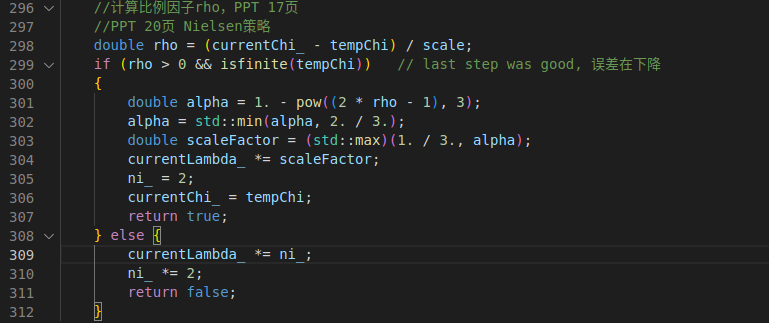
对参数的估计比较准确

**（3）其他阻尼因子策略**

提供的论文中4.1.1节内容：



**第三种是PPT20页中提到的Nielsen策略**



**修改阻尼因子策略，有四个函数需要注意：**

**一、LM算法初始化函数ComputeLambdaInitLM函数**

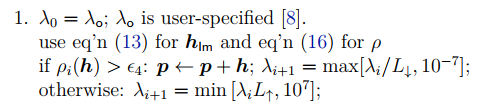
**二、LM算法迭代更新步骤函数IsGoodStepInLM函数**

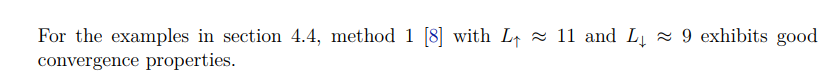
**三、将阻尼因子添加到J^T\*J上的函数：AddLambdatoHessianLM函数和RemoveLambdaHessianLM函数**

**为了进行不同阻尼因子策略的对比，估计曲线选用y=exp(ax^2+bx+c)，因为迭代次数比y=ax^2+bx+c多一些，数据点N设为100个，通过第一小问运行结果可以看出Nielsen策略中阻尼因子初始化值为0.001，所以我们这里的实现也初始化为0.001**

**第一种阻尼因子策略的实现：**

**论文中的h就是PPT中的Δx**

****

****

**公式13、16如下：**

**文本, Word

中度可信度描述已自动生成**

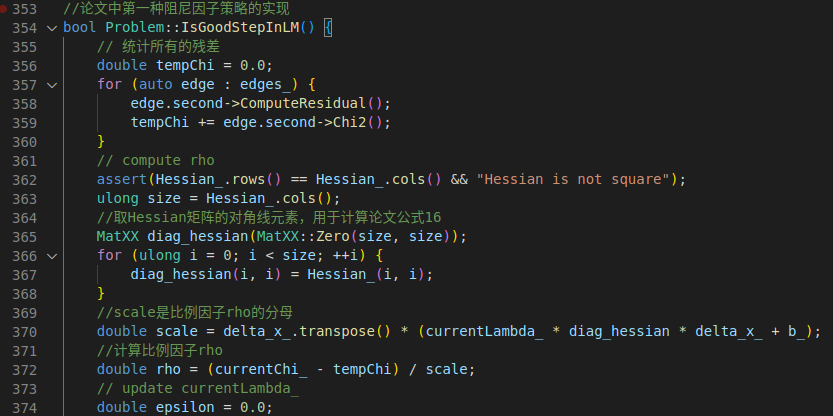
**图形用户界面, 文本, 应用程序, 表格

描述已自动生成**

修改ComputeLambdaInitLM函数：



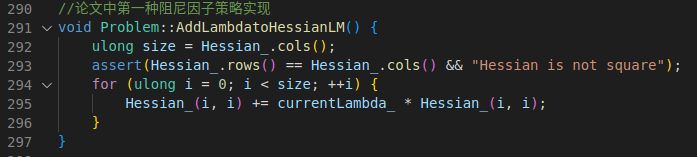
修改IsGoodStepInLM函数：



屏幕上有字

描述已自动生成

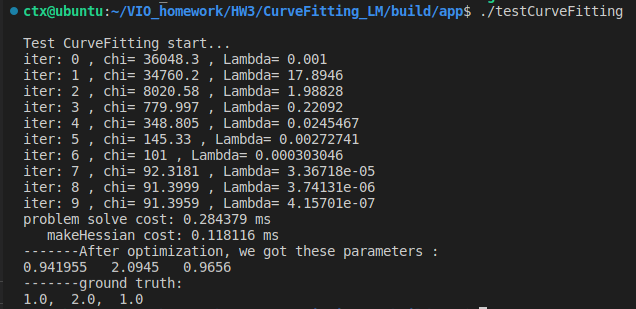
修改AddLambdatoHessianLM函数：



修改RemoveLambdaHessianLM函数：



**运行结果：**



和Nielsen策略对比迭代次数少两次，体现不出啥东西，残差函数平方和都从最初的36048.3找到了最小为91.3959，阻尼因子和Nielsen策略相比整体小很多，不过因为这里实现用到的论文公式13与Nielsen用到的公式12已经导致阻尼因子在问题求解中的地位不太一样了，不是直接加到原来Hessian矩阵（不是真正的Hessian矩阵，实际是J^T \* W \* J），而是先乘以对应Hessian矩阵中对应位置的对角线元素再加上去。

**第二题和第三题在手写笔记的后四页中**