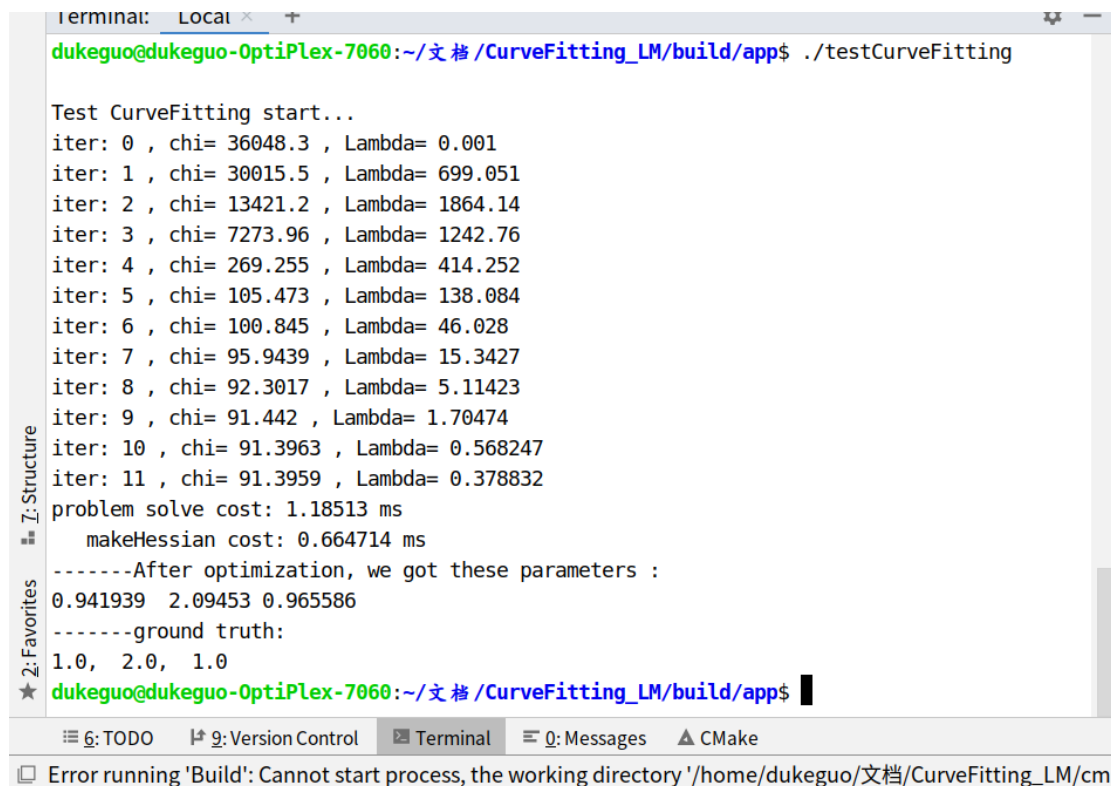


作业

- 1 样例代码给出了使用 LM 算法来估计曲线 $y = \exp(ax^2 + bx + c)$ 参数 a, b, c 的完整过程。
 - ① 请绘制样例代码中 LM 阻尼因子 μ 随着迭代变化的曲线图
 - ② 将曲线函数改成 $y = ax^2 + bx + c$ ，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。
 - ③ 实现其他更优秀的阻尼因子策略，并给出实验对比（选做，评优秀），策略可参考论文^a 4.1.1 节。

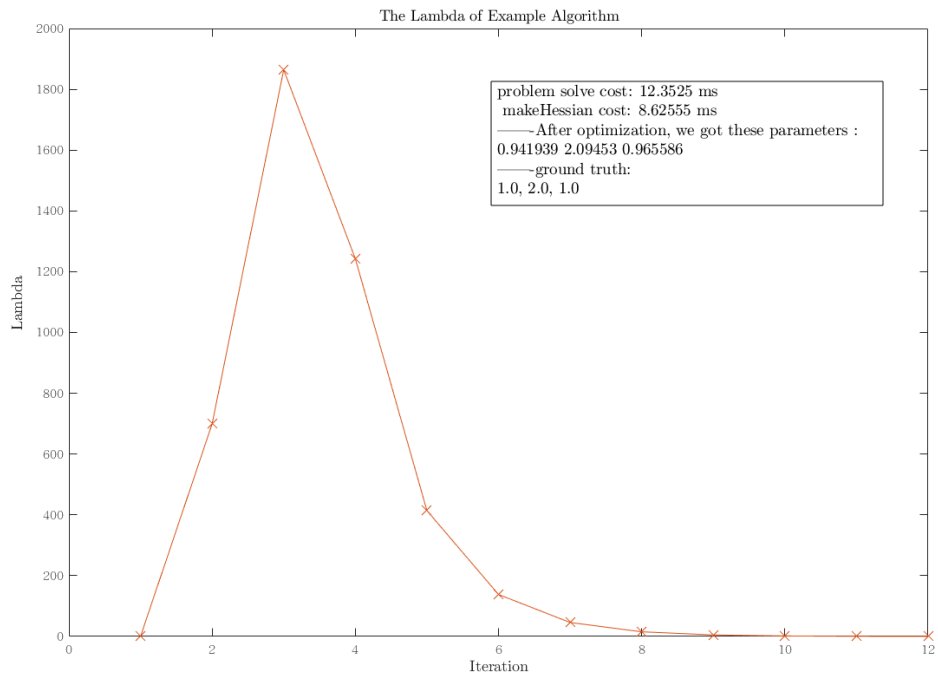
习题 1：直接使用编译器编译示例代码可以得到以下结果：迭代了 12 次，最终 $a=0.9419$ ， $b=2.0945$ ， $c=0.9656$ ，与真实值 121 接近



```
terminal: Local x +
dukeguo@dukeguo-OptiPlex-7060:~/文档/CurveFitting_LM/build/app$ ./testCurveFitting

Test CurveFitting start...
iter: 0 , chi= 36048.3 , Lambda= 0.001
iter: 1 , chi= 30015.5 , Lambda= 699.051
iter: 2 , chi= 13421.2 , Lambda= 1864.14
iter: 3 , chi= 7273.96 , Lambda= 1242.76
iter: 4 , chi= 269.255 , Lambda= 414.252
iter: 5 , chi= 105.473 , Lambda= 138.084
iter: 6 , chi= 100.845 , Lambda= 46.028
iter: 7 , chi= 95.9439 , Lambda= 15.3427
iter: 8 , chi= 92.3017 , Lambda= 5.11423
iter: 9 , chi= 91.442 , Lambda= 1.70474
iter: 10 , chi= 91.3963 , Lambda= 0.568247
iter: 11 , chi= 91.3959 , Lambda= 0.378832
problem solve cost: 1.18513 ms
makeHessian cost: 0.664714 ms
-----After optimization, we got these parameters :
0.941939 2.09453 0.965586
-----ground truth:
1.0, 2.0, 1.0
dukeguo@dukeguo-OptiPlex-7060:~/文档/CurveFitting_LM/build/app$
```

得到的 lambda 的分布如下图所示：



两次实验是不同时间做的，用时上有些区别。

2.把函数改成 $y=ax^2+bx+c$ ，修改 computeResidual 和 computeJacobian 函数，并将数据量提高到 1000，修正函数后的结果是更少的迭代数。

```

CurveFitting.cpp | problem.cc
jacobians_
39 // 计算残差对变量的雅克比
40 virtual void ComputeJacobians() override
41 {
42     Vec3 abc = vertices_[0]->Parameters();
43     double exp_y = std::exp( x: abc(index:0)*x_ + abc(index:1)*x_ + abc(index:2) );
44     Eigen::Matrix<double, 1, 3> jaco_abc; // 误差为1维，状态量 3 个，所以是 1x3 的雅克比矩阵
45     jaco_abc << x_ * x_ * exp_y, x_ * exp_y, 1 * exp_y;
46     jacobians_[0] = jaco_abc;
47
48     // dukeguo 20191104
49     Eigen::Matrix<double, 1, 3> jaco_abc; // 误差为1维，状态量 3 个，所以是 1x3 的雅克比矩阵
50     jaco_abc << x_ * x_ * exp_y, x_ * exp_y, 1 * exp_y;
51     jacobians_[0] = jaco_abc;
52 }
53
54 // 返回边的类型信息
55 virtual std::string TypeInfo() const override { return "CurveFittingEdge"; }
56 public:
57 double x_, y_; // x_ 值， y_ 值为 measurement
main
Run: testCurveFitting
/home/dukeguo/文档/CurveFitting_LM/cmake-build-debug/app/te

// 计算曲线模型误差
virtual void ComputeResidual() override
{
    // 示例代码
    Vec3 abc = vertices_[0]->Parameters(); // 估计的参数
    residual_(index:0) = std::exp( x: abc(index:0)*x_ + abc(index:1)*x_ + abc(index:2) ) - y_; // 构建残差

    // dukeguo 20191104
    Vec3 abc = vertices_[0]->Parameters(); // 估计的参数
    residual_(0) = abc(0) * x_ * x_ + abc(1) * x_ + abc(2) - y_; // y=ax^2+bx+c 构建残差
}

```

```
dukeguo@dukeguo-OptiPlex-7060: ~/文档/CurveFitting_LM/cmake-build-debug/app
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
dukeguo@dukeguo-OptiPlex-7060:~/文档/CurveFitting_LM/cmake-build-debug/app$ ./testCurveFitting

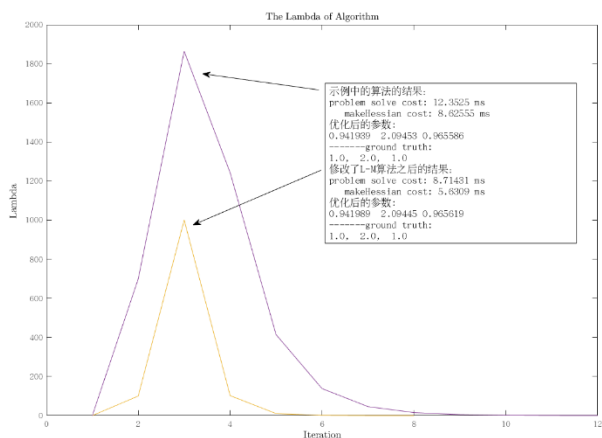
Test CurveFitting start...
iter: 0 , chi= 3.21386e+06 , Lambda= 19.95
iter: 1 , chi= 974.658 , Lambda= 6.65001
iter: 2 , chi= 973.881 , Lambda= 2.21667
iter: 3 , chi= 973.88 , Lambda= 1.47778
problem solve cost: 36.3369 ms
makeHessian cost: 29.9651 ms
-----After optimization, we got these parameters :
0.999588 2.0063 0.968786
-----ground truth:
1.0, 2.0, 1.0
dukeguo@dukeguo-OptiPlex-7060:~/文档/CurveFitting_LM/cmake-build-debug/app$
```

3.阅读了 LM 论文后，参考了一些资料，修改了 problem.cc 里面的 IsGoodStepInLM():

```
310 //recompute residuals after update state
311 //统计所有的残差
312 double tempChi = 0.0;
313 for (auto edge: edges) {
314     edge.second->ComputeResidual();
315     tempChi += edge.second->Chi2();
316 }
317
318 if (tempChi < currentChi && !isfinite(tempChi)){
319     //如果损失函数下降了，采用论文里第一种更新策略
320     Vec3 abc = vertices_[0]->Parameters();
321     Vec3 abc_trial = abc + delta_x_;
322     //论文里面的式子(10)
323     wip = abc_trial.dot(abc)/(abc.norm()*abc_trial.norm());
324     currentLambda *= pow(beta,wip);
325     currentChi_ = tempChi;
326     return true;
327 }else{
328     //如果损失函数上升了，执行第二种更新策略
329     MatXX D = Hessian;
330     //判断Hessian矩阵是否正定
331     bool is_D_positive_definite = !D.ldlt().vectorD().cwiseAbs().minCoeff() < 1e-6;
332     if (is_D_positive_definite){
333         //如果正定，执行论文的式 (12)
334         currentLambda /= beta;
335         return false;
336     }else{
337         //如果不正定就取Hessian矩阵每一列非对角线元素绝对值和的最小值作为新的lambda，参照论文式(12)
338         Eigen::VectorXd d_i_j(D.rows());
339         for(int i=0;i<D.rows();i++){
340             for(int j=0;j<D.cols();j++){
341                 if(i!=j){
342                     d_i_j(i)+=abs(D(i,j));
343                 }
344             }
345         }
346         currentLambda_ = d_i_j.minCoeff();
347         return false;
348     }
349 }
```

```
Run: testCurveFitting
/home/dukeguo/文档/CurveFitting_LM/cmake-build-debug/app/tes

Test CurveFitting start...
iter: 0 , chi= 36048.3 , Lambda= 0.001
iter: 1 , chi= 14218.6 , Lambda= 100
iter: 2 , chi= 707.856 , Lambda= 1000.48
iter: 3 , chi= 103.978 , Lambda= 102.075
iter: 4 , chi= 101.56 , Lambda= 10.3188
iter: 5 , chi= 92.6208 , Lambda= 1.06747
iter: 6 , chi= 91.4023 , Lambda= 0.107681
iter: 7 , chi= 91.3959 , Lambda= 0.0107687
problem solve cost: 8.68086 ms
makeHessian cost: 5.63403 ms
-----After optimization, we got these parameters :
0.941989 2.09445 0.965619
-----ground truth:
1.0, 2.0, 1.0
Process finished with exit code 0
```



习题 2:

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:

$$\mathbf{f}_{15} = \frac{\partial \alpha_{b_i b_{k+1}}}{\partial \delta \mathbf{b}_k^g} = -\frac{1}{4} (\mathbf{R}_{b_i b_{k+1}} [(\mathbf{a}^{b_{k+1}} - \mathbf{b}_k^a)] \times \delta t^2) (-\delta t)$$

$$\mathbf{g}_{12} = \frac{\partial \alpha_{b_i b_{k+1}}}{\partial \mathbf{n}_k^g} = -\frac{1}{4} (\mathbf{R}_{b_i b_{k+1}} [(\mathbf{a}^{b_{k+1}} - \mathbf{b}_k^a)] \times \delta t^2) (\frac{1}{2} \delta t)$$

$$\begin{aligned} f_{15} &= \frac{\partial \alpha_{b_i b_{k+1}}}{\partial \delta b_k^g} \\ &= \frac{\partial \frac{1}{4} q_{b_i b_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2} \omega \delta t \end{bmatrix} \otimes \begin{bmatrix} 1 \\ -\frac{1}{2} \delta b_k^g \delta t \end{bmatrix} (a^{b_{k+1}} - b_k^a) \delta t^2}{\partial \delta b_k^g} \\ &= \frac{1}{4} \frac{\partial R_{b_i b_{k+1}} \exp([- \delta b_k^g \delta t] \times) (a_{b_{k+1}} - b_k^a) \delta t^2}{\partial \delta b_k^g} \\ &= \frac{1}{4} \frac{\partial R_{b_i b_{k+1}} (I + [- \delta b_k^g \delta t] \times) (a_{b_{k+1}} - b_k^a) \delta t^2}{\partial \delta b_k^g} \\ &= -\frac{1}{4} \frac{\partial R_{b_i b_{k+1}} ([(a_{b_{k+1}} - b_k^a) \delta t^2] \times) (-\delta b_k^g \delta t)}{\partial \delta b_k^g} \\ &= -\frac{1}{4} (R_{b_i b_{k+1}} [(a_{b_{k+1}} - b_k^a)] \times \delta t^2) (-\delta t) \end{aligned}$$

$$\begin{aligned} g_{12} &= \frac{\partial \alpha_{b_i b_{k+1}}}{\partial n_k^g} \\ &= \frac{\partial \frac{1}{4} q_{b_i b_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2} \omega \delta t \end{bmatrix} \otimes \begin{bmatrix} 1 \\ \frac{1}{2} (\frac{1}{2} \delta n_k^g \delta t) \end{bmatrix} (a^{b_{k+1}} - b_k^a) \delta t^2}{\partial \delta n_k^g} \\ &= \frac{1}{4} \frac{\partial R_{b_i b_{k+1}} \exp([\frac{1}{2} \delta n_k^g \delta t] \times) (a_{b_{k+1}} - b_k^a) \delta t^2}{\partial \delta n_k^g} \\ &= \frac{1}{4} \frac{\partial R_{b_i b_{k+1}} (I + [\frac{1}{2} \delta n_k^g \delta t] \times) (a_{b_{k+1}} - b_k^a) \delta t^2}{\partial \delta n_k^g} \\ &= -\frac{1}{4} \frac{\partial R_{b_i b_{k+1}} ([(a_{b_{k+1}} - b_k^a) \delta t^2] \times) (\frac{1}{2} \delta n_k^g \delta t)}{\partial \delta n_k^g} \\ &= -\frac{1}{4} (R_{b_i b_{k+1}} [(a_{b_{k+1}} - b_k^a)] \times \delta t^2) (\frac{1}{2} \delta t) \end{aligned}$$

习题 3:

3 证明式(9)。 $\Delta x_{lm} = - \sum_{j=1}^n \frac{v_j^T F'^T v_j}{\lambda_j + \mu} v_j$

证明:
$$\Delta x_{lm} = - \sum_{j=1}^n \frac{v_j^T F'^T v_j}{\lambda_j + \mu} v_j$$

$\because J^T J$ 半正定, \forall 非零向量 x 存在 $x^T J^T J x \geq 0$ 恒成立.

$$(J^T J + \mu I) \Delta x_{lm} = -J^T f$$

令 $J^T J + \mu I$ 为 A , 式为 $A \Delta x_{lm} = -J^T f$

$$\begin{aligned} \therefore x^T A x &= x^T (J^T J + \mu I) x \\ &= x^T J^T J x + x^T \mu I x \\ &= x^T J^T J x + \mu x^T x > 0 \end{aligned}$$

$\therefore x^T A x > 0$, A 正定, 可特征值分解, 对 $J^T J$ 进行特征值分解 $V \Lambda V^T$, 即 $\{\lambda_j\}$ 和 $\{v_j\}$ 分别是特征值和特征向量. 就有

$$J^T J v_j = \lambda_j v_j$$

$$\begin{aligned} A v_j &= (J^T J + \mu I) v_j \\ &= J^T J v_j + \mu v_j \\ &= (\lambda_j + \mu) v_j \end{aligned}$$

$\therefore (\lambda_j + \mu)$ 和 v_j 分别是 A 的特征值和特征向量

$$A = V \Sigma V^T$$

其中 $\Sigma = \text{diag}(\lambda_1 + \mu, \lambda_2 + \mu, \dots, \lambda_n + \mu)$

$$A \Delta x_{lm} = -J^T f$$

$$V \Sigma V^T \Delta x_{lm} = -J^T f$$

$$\Delta x_{lm} = -(V \Sigma V^T)^{-1} J^T f$$

$$\Delta x_{lm} = -V \Sigma^{-1} V^T J^T f$$

其中 $\Sigma^{-1} = \text{diag}(\frac{1}{\lambda_1 + \mu}, \frac{1}{\lambda_2 + \mu}, \dots, \frac{1}{\lambda_n + \mu})$

$$\therefore \Delta x_{lm} = - \sum_{j=1}^n \frac{v_j^T J^T f v_j}{\lambda_j + \mu} v_j = - \sum_{j=1}^n \frac{v_j^T F'^T v_j}{\lambda_j + \mu} v_j$$