Course3_homework_LM_error_grad

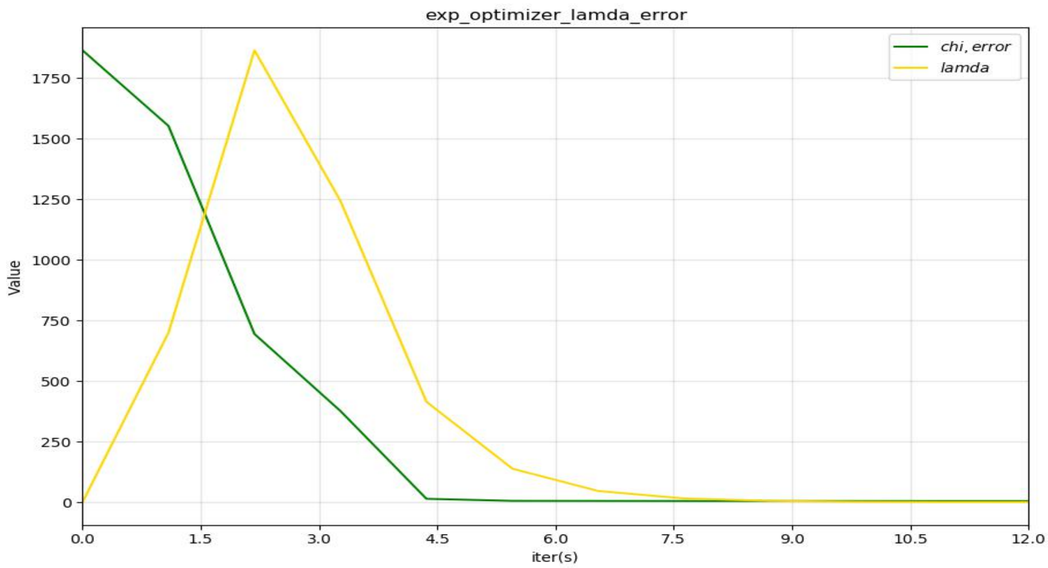
0. Summary

（1）对于$y = ax^2 + bx + c$,发现对于样本比较敏感；N=1000 时计算正常；而 N=100 和 10000 时计算有问题，说明不一定存在一套参数适应各种样本；

1. LM

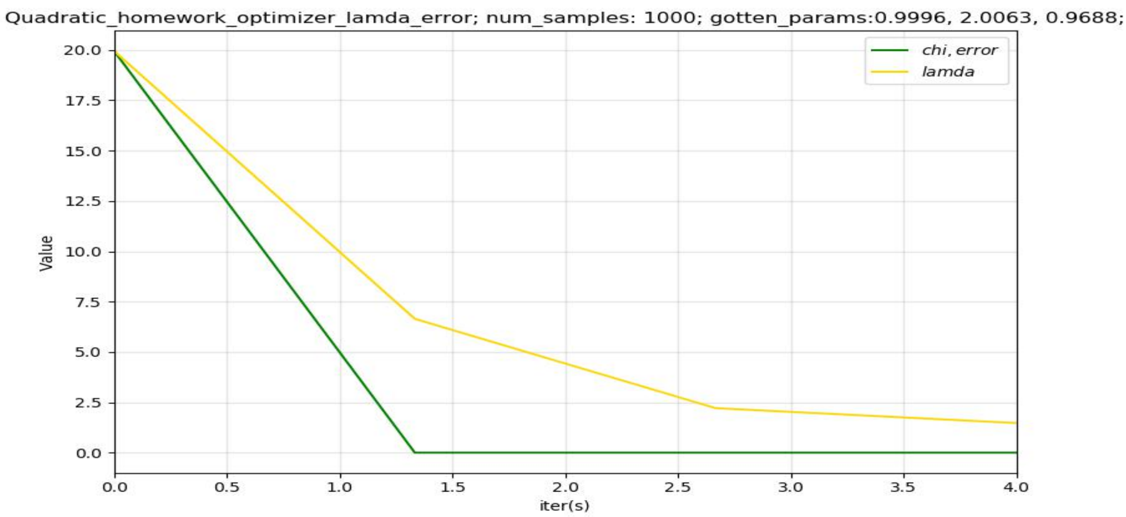
（1） ❶ 请绘制样例代码中 LM 阻尼因子 μ 随着迭代变化的曲线图

下图种，将 chi_error 的量纲转换为和 lamda 相同（即转换为相同的 max），这样可以显示在同一个图中；



（2） ❷ 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。

程序中的样本点数目为 1000，下图中也是将 chi_error 的量纲换算为和 lamda 相同；
计算的 params 是正确的；



修改的 code 如下：

```
// 误差模型 模板参数：观测值维度, 类型, 连接顶点类型
class CurveFittingEdge: public Edge
{
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
    CurveFittingEdge( double x, double y ): Edge(1,1, std::vector<std::string>{"abc"}) {
        x_ = x;
        y_ = y;
    }
    // 计算曲线模型误差
    virtual void ComputeResidual() override
    {
        Vec3 abc = verticies_[0]->Parameters();  // 估计的参数
        //residual_(0) = std::exp( abc(0)*x_*x_ + abc(1)*x_ + abc(2) ) - y_;  // 构建残差,orig;
        residual_(0) = 1.0*( abc(0)*x_*x_ + abc(1)*x_ + abc(2) ) - y_;  // 构建残差,homework!!;
    }

    // 计算残差对变量的雅克比
    virtual void ComputeJacobians() override
    {
        Vec3 abc = verticies_[0]->Parameters();
        //double exp_y = std::exp( abc(0)*x_*x_ + abc(1)*x_ + abc(2) );//orig;

        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
        //jaco_abc << x_ * x_ * exp_y, x_ * exp_y , 1 * exp_y;//orig
        jaco_abc << x_ * x_ , x_  , 1.0 ;//homework!!;
        jacobians_[0] = jaco_abc;
    }

int main()
{
    double a=1.0, b=2.0, c=1.0;         // 真实参数值
    int N =1000;  // 100;                       // 数据点;orig:100; homework:1000;
    double w_sigma= 1.;                 // 噪声Sigma值

    std::default_random_engine generator;
    std::normal_distribution<double> noise(0.,w_sigma);

    // 构建 problem
    Problem problem(Problem::ProblemType::GENERIC_PROBLEM);
    shared_ptr< CurveFittingVertex > vertex(new CurveFittingVertex());

    // 设定待估计参数 a, b, c初始值
    vertex->SetParameters(Eigen::Vector3d (0.,0.,0.));
    // 将待估计的参数加入最小二乘问题
    problem.AddVertex(vertex);


    std::cout<<"sample num: "<<N<<std::endl;

    // 构造 N 次观测
    for (int i = 0; i < N; ++i) {//orig;
    //for (int i = 1; i < N; ++i) {//homework;

        double x = i/100.;
        double n = noise(generator);
        // 观测 y
        //double y = std::exp( a*x*x + b*x + c ) + n;//orig
//          double y = std::exp( a*x*x + b*x + c );
        double y = a*x*x + b*x + c  + n;//homework;
```

程序测试结果：

[a] 若 N=100，则输出如下：

sample num: 100

Test CurveFitting start...
iter: 0 , chi= 719.475 , Lambda= 0.001
iter: 1 , chi= 91.395 , Lambda= 0.000333333
delta_x_.squaredNorm() 1.19984e-07 //**lamda** 很小，**delta_x** 也不再变化，说明陷入局部最优；
problem solve cost: 1.65108 ms
makeHessian cost: 0.345635 ms
-------After optimization, we got these parameters :
1.61039 1.61853 0.995178
-------ground truth:
1.0, 2.0, 1.0

[b] 若 N=10000，则输出如下：
sample num: 10000

Test CurveFitting start...
iter: 0 , chi= 2.10148e+11 , Lambda= 1.9995e+06
sqrt(currentChi_)1435.1 //**lamda** 很大，接近最速下降法，**error** 下降很快，达到程序的中止标准；
problem solve cost: 33.1489 ms
makeHessian cost: 23.8273 ms
-------After optimization, we got these parameters :
1.01207 1.04577 0.042027
-------ground truth:
1.0, 2.0, 1.0

（3）其他的阻尼因子策略

❸ 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。

4.1 Numerical Implementation

The Jacobian ($\mathbf{J} \in \mathcal{R}^{m\times n}$) is numerically approximated using backwards differences,

$$J_{ij} = \frac{\partial \hat{y}_i}{\partial p_j} = \frac{\hat{y}(t_i; \mathbf{p} + \delta\mathbf{p}_j) - \hat{y}(t_i; \mathbf{p})}{||\delta\mathbf{p}_j||} , \tag{14}$$

where the j-th element of $\delta\mathbf{p}_j$ is the only non-zero element and is set to $\epsilon_2(1 + |p_j|)$. If in an iteration $\chi^2(\mathbf{p}) - \chi^2(\mathbf{p}+\mathbf{h}) > \epsilon_3 \mathbf{h}^T(\lambda\mathbf{h} + \mathbf{J}^T\mathbf{W}(\mathbf{y} - \hat{\mathbf{y}}))$ then $\mathbf{p}+\mathbf{h}$ is sufficiently better than $\mathbf{p}$, $\mathbf{p}$ is replaced by $\mathbf{p}+\mathbf{h}$, and λ is reduced by a factor of ten. Otherwise λ is increased by a factor of ten, and the algorithm proceeds to the next iteration. Convergence is achieved if $\max(|h_i/p_i|) < \epsilon_2$, $\chi^2/m < \epsilon_3$, or $\max(|\mathbf{J}^T\mathbf{W}(\mathbf{y} - \hat{\mathbf{y}})|) < \epsilon_1$. Otherwise, iterations terminate when the iteration number exceeds a pre-specified limit.

Code 如下；停止策略未变；

problem.cc edge.cc vertex.cc CurveFitting.cpp

```cpp

bool Problem::IsGoodStepInLM() {
    double scale = 0;
    scale = delta_x_.transpose() * (currentLambda_ * delta_x_ + b_);
    scale += 1e-3;    // make sure it's non-zero :)

    // recompute residuals after update state
    // 统计所有的残差
    double tempChi = 0.0;
    for (auto edge: edges_) {
        edge.second->ComputeResidual();
        tempChi += edge.second->Chi2();
    }

    double rho = (currentChi_ - tempChi) / scale;

    const float e3=0.001;
    if (rho > e3 && isfinite(tempChi))   // last step was good, 误差在下降
    {
        currentLambda_ *= 0.1;
        currentChi_ = tempChi;
        return true;

    } else {
        currentLambda_ *= 10;
        return false;

    }
```

运行结果如下：

Nielson method

```
(base) ep@ep-VirtualBox:/media/sf_vslam_vio/lesson_doc/course3_hw_CurveFitting_LM/CurveFitting_LM$ ./cmake-build-debug/app/testCurveFitting
sample num: 1000

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
delta_x_.squaredNorm() 1.42592e-09
problem solve cost: 115.154 ms
   makeHessian cost: 88.9684 ms
-------After optimization, we got these parameters :
0.999588    2.0063 0.968786
-------ground truth:
1.0,  2.0,  1.0
```

paper《The L-M method for nonlinear》 method

```
(base) ep@ep-VirtualBox:/media/sf_vslam_vio/lesson_doc/course3_hw_CurveFitting_LM/CurveFitting_LM$ ./cmake-build-debug/app/testCurveFitting
sample num: 1000

Test CurveFitting start...
iter: 0 , chi= 3.21386e+06 , Lambda= 19.95
iter: 1 , chi= 974.658 , Lambda= 1.995
iter: 2 , chi= 973.88 , Lambda= 0.1995
delta_x_.squaredNorm() 2.8089e-07
problem solve cost: 80.5968 ms
   makeHessian cost: 63.2194 ms
-------After optimization, we got these parameters :
0.999573  2.00648 0.968326
-------ground truth:
1.0,  2.0,  1.0
```

2. error_grad 推导；f15;g12;

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

$$\boldsymbol{\omega}=\frac{1}{2}((\boldsymbol{\omega}^{b_k}-\mathbf{b}_k^g)+(\boldsymbol{\omega}^{b_{k+1}}-\mathbf{b}_k^g))$$

$$\mathbf{q}_{b_ib_{k+1}}=\mathbf{q}_{b_ib_k}\otimes\begin{bmatrix}1\\ \frac{1}{2}\boldsymbol{\omega}\delta t\end{bmatrix}$$

$$\mathbf{a}=\frac{1}{2}(\mathbf{q}_{b_ib_k}(\mathbf{a}^{b_k}-\mathbf{b}_k^a)+\mathbf{q}_{b_ib_{k+1}}(\mathbf{a}^{b_{k+1}}-\mathbf{b}_k^a))$$

$$\mathbf{a}=\frac{1}{2}(\mathbf{q}_{b_ib_k}(\bar{\mathbf{a}}^{b_k}-\mathbf{b}_k^a)+\mathbf{q}_{b_ib_{k+1}}(\bar{\mathbf{a}}^{b_{k+1}}-\mathbf{b}_k^a))$$

$$\begin{aligned}\boldsymbol{\alpha}_{b_ib_{k+1}}&=\boldsymbol{\alpha}_{b_ib_k}+\boldsymbol{\beta}_{b_ib_k}\delta t+\frac{1}{2}\mathbf{a}\delta t^2\\&=\boldsymbol{\alpha}_{b_ib_k}+\boldsymbol{\beta}_{b_ib_k}\delta t+\frac{1}{2}\left[\ \frac{1}{2}(\mathbf{q}_{b_ib_k}(\bar{\mathbf{a}}^{b_k}-\mathbf{b}_k^a)+\mathbf{q}_{b_ib_k}\otimes\begin{bmatrix}1\\ \frac{1}{2}\boldsymbol{\omega}\delta t\end{bmatrix}(\bar{\mathbf{a}}^{b_{k+1}}-\mathbf{b}_k^a))\ \right]\delta t^2\end{aligned}$$

$$\begin{aligned}\frac{\partial\boldsymbol{\alpha}_{b_ib_{k+1}}}{\partial\delta\mathbf{b}_k^g}&=\frac{\frac{1}{4}\partial\ \mathbf{q}_{b_ib_k}\otimes\begin{bmatrix}1\\ \frac{1}{2}(\boldsymbol{\omega}-\delta\mathbf{b}_k^g)\delta t\end{bmatrix}(\bar{\mathbf{a}}^{b_{k+1}}-\mathbf{b}_k^a))\ \delta t^2}{\partial\delta\mathbf{b}_k^g}\\&=\frac{1}{4}\frac{\partial\mathbf{R}_{b_ib_k}\exp([(\boldsymbol{\omega}-\delta\mathbf{b}_k^g)\delta t]_\times)(\mathbf{a}^{b_{k+1}}-\mathbf{b}_k^a)\ \delta t^2}{\partial\delta\mathbf{b}_k^g}\\&=\frac{1}{4}\frac{\partial\mathbf{R}_{b_ib_k}\exp([\boldsymbol{\omega}\delta t]_\times)\exp\left([-J_r(\boldsymbol{\omega}\delta t)\delta\mathbf{b}_k^g\delta t]_\times\right)(\mathbf{a}^{b_{k+1}}-\mathbf{b}_k^a)\delta t^2}{\partial\delta\mathbf{b}_k^g}\\&=\frac{1}{4}\frac{\partial-\mathbf{R}_{b_ib_{k+1}}([(\mathbf{a}^{b_{k+1}}-\mathbf{b}_k^a)\delta t^2]_\times)(-J_r(\boldsymbol{\omega}\delta t)\delta\mathbf{b}_k^g\delta t)}{\partial\delta\mathbf{b}_k^g}\\&=-\frac{1}{4}(\mathbf{R}_{b_ib_{k+1}}[(\mathbf{a}^{b_{k+1}}-\mathbf{b}_k^a)]_\times\delta t^2)(-J_r(\boldsymbol{\omega}\delta t)\delta t)\\&\approx-\frac{1}{4}(\mathbf{R}_{b_ib_{k+1}}[(\mathbf{a}^{b_{k+1}}-\mathbf{b}_k^a)]_\times\delta t^2)(-\delta t)\end{aligned}$$

（2） $$\mathbf{g}_{12}=\frac{\partial\delta\boldsymbol{\alpha}_{b_{k+1}}}{\partial\mathbf{n}_k^g}=\mathbf{g}_{14}=\frac{\partial\delta\boldsymbol{\alpha}_{b_{k+1}}}{\partial\mathbf{n}_{k+1}^g}=-\frac{1}{4}(\mathbf{R}_{b_ib_{k+1}}[(\mathbf{a}^{b_{k+1}}-\mathbf{b}_k^a)]_\times\delta t^2)(\frac{1}{2}\delta t)$$

$$\boldsymbol{\omega}=\frac{1}{2}((\bar{\boldsymbol{\omega}}^{b_k}+\mathbf{n}_k^g-\mathbf{b}_k^g)+(\bar{\boldsymbol{\omega}}^{b_{k+1}}+\mathbf{n}_{k+1}^g-\mathbf{b}_k^g))$$

和（1）的主要差别在于$\frac{1}{2}\mathbf{n}_k^g$；

$$\frac{\partial \delta \boldsymbol{\alpha}_{b_{k+1}}}{\partial \mathbf{n}_k^g} = \frac{\frac{1}{4}\partial\ \mathbf{q}_{b_i b_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\left(\boldsymbol{\omega} + \frac{1}{2}\mathbf{n}_k^g\right)\delta t \end{bmatrix} (\bar{\mathbf{a}}^{b_{k+1}} - \mathbf{b}_k^a))\ \delta t^2}{\partial \mathbf{n}_k^g}$$

$$= \frac{1}{4}\frac{\partial \mathbf{R}_{b_i b_k}\exp\left(\left[\left(\boldsymbol{\omega} + \frac{1}{2}\mathbf{n}_k^g\right)\delta t\right]_\times\right)(\mathbf{a}^{b_{k+1}} - \mathbf{b}_k^a)\ \delta t^2}{\partial \mathbf{n}_k^g}$$

$$= \frac{1}{4}\frac{\partial \mathbf{R}_{b_i b_k}\exp\left([\boldsymbol{\omega}\delta t]_\times\right)\exp\left(\left[\ J_r\left(\boldsymbol{\omega}\delta t\right)\frac{1}{2}\mathbf{n}_k^g\,\delta t\ \right]_\times\right)(\mathbf{a}^{b_{k+1}} - \mathbf{b}_k^a)\,\delta t^2}{\partial \mathbf{n}_k^g}$$

$$= \frac{1}{4}\frac{\partial - \mathbf{R}_{b_i b_{k+1}}\left(\left[(\mathbf{a}^{b_{k+1}} - \mathbf{b}_k^a)\delta t^2\right]_\times\right)\left(\ J_r\left(\boldsymbol{\omega}\delta t\right)\frac{1}{2}\mathbf{n}_k^g\,\delta t\ \right)}{\partial \mathbf{n}_k^g}$$

$$= -\frac{1}{4}\left(\mathbf{R}_{b_i b_{k+1}}[(\mathbf{a}^{b_{k+1}} - \mathbf{b}_k^a)]_\times \delta t^2\right)\left(\ J_r\left(\boldsymbol{\omega}\delta t\right)\ \frac{1}{2}\delta t\right)$$

$$\approx\ -\frac{1}{4}(\mathbf{R}_{b_i b_{k+1}}[(\mathbf{a}^{b_{k+1}} - \mathbf{b}_k^a)]_\times \delta t^2)(\ \frac{1}{2}\delta t)$$

3.证明式（9）

$$\begin{aligned} F(\mathbf{x}+\Delta\mathbf{x}) \approx L(\Delta\mathbf{x}) &\equiv \frac{1}{2}\boldsymbol{\ell}(\Delta\mathbf{x})^\top\boldsymbol{\ell}(\Delta\mathbf{x}) \\ &= \frac{1}{2}\mathbf{f}^\top\mathbf{f} + \Delta\mathbf{x}^\top\mathbf{J}^\top\mathbf{f} + \frac{1}{2}\Delta\mathbf{x}^\top\mathbf{J}^\top\mathbf{J}\Delta\mathbf{x} \qquad (7) \\ &= F(\mathbf{x}) + \Delta\mathbf{x}^\top\mathbf{J}^\top\mathbf{f} + \frac{1}{2}\Delta\mathbf{x}^\top\mathbf{J}^\top\mathbf{J}\Delta\mathbf{x} \end{aligned}$$

这样损失函数就近似成了一个二次函数，并且如果雅克比是满秩的，
则 $\mathbf{J}^\top\mathbf{J}$ 正定，损失函数有最小值。
另外，易得：$F'(\mathbf{x}) = (\mathbf{J}^\top\mathbf{f})^\top$，以及 $F''(\mathbf{x}) \approx \mathbf{J}^\top\mathbf{J}$.

$$\mathbf{J}^\top\mathbf{J} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\top$$

$$\left(\mathbf{J}^\top\mathbf{J} + \mu\mathbf{I}\right)\Delta\mathbf{x}_{\mathrm{lm}} = -\mathbf{J}^\top\mathbf{f} \qquad \mathbf{J}^\top\mathbf{J} + \mu\mathbf{I} = \mathbf{V}\left(\boldsymbol{\Lambda} + \mu\mathbf{I}\right)\mathbf{V}^\top$$

$$\left(\mathbf{J}^\top\mathbf{J} + \mu\mathbf{I}\right)^{-1} = \mathbf{V}\left(\boldsymbol{\Lambda} + \mu\mathbf{I}\right)^{-1}\mathbf{V}^\top = \sum_{j=1}^{n}\frac{\overset{\text{列}}{\mathbf{v}_j}\ \overset{\text{行}}{\mathbf{v}_j}^\top}{\lambda_j + \mu}$$

$$\Delta\mathbf{x}_{\mathrm{lm}} = \sum_{j=1}^{n} \frac{\overset{列}{\mathbf{v}_j}\ \overset{行}{\mathbf{v}_j^{\top}}}{\lambda_j + \mu} \overset{列}{\left(-\mathbf{J}^{\top}\mathbf{f}\right)} = \sum_{j=1}^{n} \frac{\overset{行}{\mathbf{v}_j^{\top}} \overset{列}{\left(-\mathbf{J}^{\top}\mathbf{f}\right)} \overset{列}{\mathbf{v}_j}}{\lambda_j + \mu} = -\sum_{j=1}^{n} \frac{\mathbf{v}_j^{\top}\mathbf{F}'^{\top}}{\lambda_j + \mu}\mathbf{v}_j$$