深蓝学院 VIO 课程第三课作业

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. 如果有实现其他阻尼因子更新策略可加分（选做）。

2 公式推导，根据课程知识，完成 $\mathbf{F},\mathbf{G}$ 中如下两项的推导过程：

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

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

3 证明式(9)。

1.1 绘制 LM 阻尼因子随迭代变化的曲线图

修改 CurveFitting_LM 项目的 problem.h 和 problem.cc 源码，在 Problem::Solve()函数中添加记录阻尼因子μ的代码，执行后会写入一个 txt 文件中（见附件 1.1-lambdas.txt）。撰写一个脚本绘制阻尼因子μ随迭代次数变化的图，见图 1：

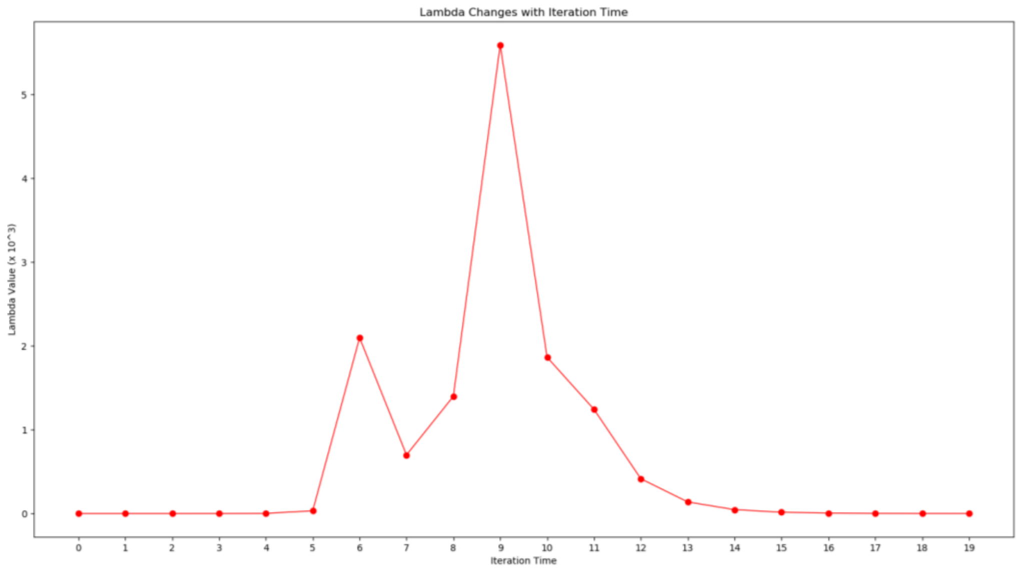

图 1 阻尼因子μ随迭代次数变化曲线图

1.2 修改曲线函数，完成代码中残差计算和曲线参数估计

新的曲线函数：$y = ax^2 + bx + c$

观测值受噪声影响：$y_m = ax^2 + bx + c + noise$

残差计算：$residual = ax^2 + bx + c - y_m$

残差对变量的雅克比：$\mathrm{J} = \begin{bmatrix} \frac{\partial r}{\partial a} & \frac{\partial r}{\partial b} & \frac{\partial r}{\partial c} \end{bmatrix} = \begin{bmatrix} x^2 & x & 1 \end{bmatrix}$

部分修改的代码如下图：

```
// 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) = abc(0)*x_*x_ + abc(1)*x_ + abc(2) - y_;  // 构建残差
    }

    // 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) );

        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_, x_, 1;
        jacobians_[0] = jaco_abc;
    }
    /// 返回边的类型信息
    virtual std::string TypeInfo() const override { return "CurveFittingEdge"; }
public:
    double x_,y_;  // x 值， y 值为 _measurement
};

void write2txt(const std::vector<double>& data) {
    ofstream ofs("lambdas2.txt");
    for (size_t i=0; i<data.size(); ++i) {
        ofs << data[i] << endl;
    }
    ofs.close();
}
```

图 2 新曲线模型的残差和雅克比计算代码

完整代码见附件 1.2-CurveFitting2.cpp，多次运行结果输出详情可见附件 1.2-runing_log.txt 文件，这里整理成表格：

表 1 不同观测点数下新曲线模型的参数估计结果

<table>
<tr><th>观测点数</th><th>迭代次数</th><th>耗时(ms)</th><th>计算值</th><th>真实值</th></tr>
<tr><td>100</td><td>2</td><td>1.913</td><td>[1.610, 1.618, 0.995]</td><td rowspan="4">[1.0 ,2.0, 1.0]</td></tr>
<tr><td>500</td><td>3</td><td>14.604</td><td>[1.014, 1.936, 1.025]</td></tr>
<tr><td>700</td><td>3</td><td>19.200</td><td>[1.001, 1.989, 0.989]</td></tr>
<tr><td>1000</td><td>4</td><td>36.508</td><td>[1.000, 2.006, 0.969]</td></tr>
</table>

1.3 其他阻尼因子的更新策略

查阅资料[1]，阻尼因子 μ 的更新策略除了 g2o 和 ceres 采用的 Nielsen 策略外，还有另外两种常用的更新策略：

第一种策略[2]：

$$\mu_{i+1} = \begin{cases} \max(\frac{\mu_i}{L_\downarrow}, 10^{-7}) & \rho > 0 \\ \min(\mu_i L_\uparrow, 10^{7}) & \rho \leq 0 \end{cases}$$

第二种策略较为复杂：

$$\mu_{i+1} = \begin{cases} \max(\frac{\mu_i}{1+\alpha}, 10^{-7}) & \rho(\alpha\Delta x) > 0 \\ \mu_i + |F(x+\Delta x) - F(x)|/(2\alpha) & \rho(\alpha\Delta x) \leq 0 \end{cases}$$

其中

$$\alpha = \frac{(J^T b)^T \Delta x}{\frac{F(x+\Delta x) - F(x)}{2} - 2(J^T b)^T \Delta x}$$

这里在 backend/problem.cc 文件中实现了另外的第一种更新策略，见下图。

```
bool Problem::IsGoodStepInLM2() {
    // 统计所有的残差
    double tempChi = 0.0;
    for (auto edge: edges_) {
        edge.second->ComputeResidual();
        tempChi += edge.second->Chi2();
    }

    double frac1 = delta_x_.transpose() * b_;
    double alpha = frac1 / ((tempChi-currentChi_)/2.0 + 2*frac1);
    RollbackStates();
    delta_x_ *= alpha;
    UpdateStates();

    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
    tempChi = 0.0;
    for (auto edge: edges_) {
        edge.second->ComputeResidual();
        tempChi += edge.second->Chi2();
    }

    double rho = (currentChi_ - tempChi) / scale;
    if (rho > 0 && isfinite(tempChi))   // last step was good, 误差在下降
    {
        currentLambda_ = max(currentLambda_/(1+alpha), 1.0e-7);
        currentChi_ = tempChi;
        return true;
    } else {
        currentLambda_ += abs(currentChi_ - tempChi)/(2*alpha);
        return false;
    }
}
```

图 3 新的阻尼因子 μ 更新策略源码

1 Henri P Gavin. The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems. Duke University, 2019.
2 D.W. Marquardt. “An algorithm for least-squares estimation of nonlinear parameters,” Journal of the Society for Industrial and Applied Mathematics, 11(2):431-441, 1963.

2. 证明公式f_{15}和g_{12}

f_{15}为位置预计分量（$\alpha_{b_ib_{k+1}}$）对 k 时刻角速度 bias（b_k^g）的 Jacobian，参考课件中f_{35}的推导公式和已知条件有：

$$\alpha_{b_ib_{k+1}} = \alpha_{b_ib_k} + \beta_{b_ib_k}\delta t + \frac{1}{2}a\delta t^2 \tag{1}$$

$$\omega = \frac{1}{2}\left(\left(\omega^{b_k} + n_k^g - b_k^g\right) + \left(\omega^{b_{k+1}} + n_{k+1}^g - b_k^g\right)\right) \tag{2}$$

其中

$$\begin{aligned} a &= \frac{1}{2}\left(q_{b_ib_k}(a^{b_k} + n_k^a - b_k^a) + q_{b_ib_{k+1}}(a^{b_{k+1}} + n_{k+1}^a - b_k^a)\right) \\ &= \frac{1}{2}\left(q_{b_ib_k}(a^{b_k} + n_k^a - b_k^a) + q_{b_ib_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\omega\delta t \end{bmatrix}(a^{b_{k+1}} + n_{k+1}^a - b_k^a)\right) \end{aligned} \tag{3}$$

可知f_{15}只与式(2)(3)中红色部分有关。由于噪声n_{k+1}^a无法测得，实际计算时用测量值代替，可以忽略，则有：

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

同理，对于g_{12}来说只与式(2)蓝色部分有关，则：

$$
\begin{aligned}
g_{12} &= \frac{\partial \alpha_{b_i b_{k+1}}}{\partial \delta n_k^g} = \frac{\partial \frac{1}{2} a \delta t^2}{\partial \delta n_k^g} \\
&= \frac{1}{4} \frac{\partial q_{b_i b_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\omega \delta t \end{bmatrix} \otimes \begin{bmatrix} 1 \\ \frac{1}{4}\delta n_k^g \delta t \end{bmatrix} (a^{b_{k+1}} - b_k^a)\delta t^2}{\partial \delta n_k^g} \\
&= \frac{1}{4} \frac{\partial R_{b_i b_{k+1}} \left(I + \left[\frac{1}{2}\delta n_k^g \delta t\right]_\times \right) (a^{b_{k+1}} - b_k^a)\delta t^2}{\partial \delta n_k^g} \\
&= -\frac{1}{4} \frac{\partial R_{b_i b_{k+1}} [(a^{b_{k+1}} - b_k^a)\delta t^2]_\times \left(\frac{1}{2}\delta n_k^g \delta t\right)}{\partial \delta n_k^g} = \\
&= -\frac{1}{4} R_{b_i b_{k+1}} ([(a^{b_{k+1}} - b_k^a)]_\times \delta t^2) \left(\frac{1}{2}\delta t\right)
\end{aligned} \tag{5}
$$

得证。

3. 证明下式

$$
\Delta \mathbf{x}_{\mathrm{lm}} = -\sum_{j=1}^{n} \frac{\mathbf{v}_j^\top \mathbf{F}'^\top}{\lambda_j + \mu} \mathbf{v}_j
$$

由课件公式已知：

$$
(\boldsymbol{J}^T \boldsymbol{J} + \mu \boldsymbol{I})\Delta \boldsymbol{x} = -\boldsymbol{J}^T f = -\boldsymbol{F}'^T \tag{6}
$$

$$
\boldsymbol{J}^T \boldsymbol{J} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T \tag{7}
$$

其中 $\boldsymbol{J}^T\boldsymbol{J}$ 半正定，其特征值和对应的特征向量为$\{\lambda_j\},\{v_j\}$。因为 V 正交，所以有：

$$
\boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I} \tag{8}
$$

则有：

$$
\begin{aligned}
\Delta \boldsymbol{x} &= -\left(\boldsymbol{J}^T\boldsymbol{J} + \mu \boldsymbol{I}\right)^{-\mathbf{1}} \boldsymbol{F}'^T \\
&= -(\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T + \mu \boldsymbol{V}\boldsymbol{V}^T)^{-\mathbf{1}} \boldsymbol{F}'^T \\
&= -(\boldsymbol{V}(\boldsymbol{\Lambda} + \mu \boldsymbol{I})\boldsymbol{V}^T)^{-\mathbf{1}} \boldsymbol{F}'^T \\
&= -\mathrm{V}(\boldsymbol{\Lambda} + \mu \boldsymbol{I})^{-\mathbf{1}} \boldsymbol{V}^T \boldsymbol{F}'^T
\end{aligned} \tag{9}
$$

$$
= -\begin{bmatrix} \boldsymbol{v_1} & \boldsymbol{v_2} & \cdots & \boldsymbol{v_n} \end{bmatrix} \begin{bmatrix} \frac{1}{\lambda_1+\mu} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \frac{1}{\lambda_2+\mu} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \frac{1}{\lambda_n+\mu} \end{bmatrix} \begin{bmatrix} \boldsymbol{v_1^T} \\ \boldsymbol{v_2^T} \\ \vdots \\ \boldsymbol{v_n^T} \end{bmatrix}
$$

$$
= -\sum_{i=1}^{n} \frac{\boldsymbol{v_i v_i^T}}{\lambda_i+\mu} \boldsymbol{F'}^T
$$

其中$\boldsymbol{v_i^T}$是 1×n 的矩阵，$\boldsymbol{F'}^T = \boldsymbol{J}^T f$ 则是 n×1 的矩阵，因此$\boldsymbol{v_i^T}\boldsymbol{F'}^T$的结果为一个数。根据矩阵乘法的结合律，有:

$$
\boldsymbol{v_i v_i^T F'^T} = \boldsymbol{v_i}\left(\boldsymbol{v_i^T F'^T}\right) = \left(\boldsymbol{v_i^T F'^T}\right)\boldsymbol{v_i}
$$

故

$$
\Delta \boldsymbol{x} = -\sum_{i=1}^{n} \frac{\boldsymbol{v_i v_i^T}}{\lambda_i+\mu} \boldsymbol{F'}^T = -\sum_{i=1}^{n} \frac{\boldsymbol{v_i^T F'^T}}{\lambda_i+\mu} \boldsymbol{v_i}
$$

证毕。