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 完整过程。

- 请绘制样例代码中 LM 阻尼因子 μ 随着迭代变化的曲线图
- 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。
- 如果有实现其他阻尼因子更新策略可加分（选做）。

解. 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.

a. 首先改写代码，保存 lambda 的值到 u_data.txt：

如下

```
ofstream out("u_data.txt");

if (!out.is_open()) //is_open()返回真 (1) , 代表打开成功
{
    cout << " file  don't exist" << endl;
}
```

打开文件

```
while (!stop && (iter < iterations))
{
    std::cout << "iter: " << iter << " , chi= " << currentChi_ << " , Lambda= " << currentLambda_
              << std::endl;
    bool oneStepSuccess = false;

    int false_cnt = 0;

    while (!oneStepSuccess) // 不断尝试 Lambda, 直到成功迭代一步
    {
        // setLambda
        AddLambdatoHessianLM();
        // 第四步, 解线性方程 H X = B
        SolveLinearSystem();
        //
        RemoveLambdaHessianLM();

        // 优化退出条件1:  delta_x_ 很小则退出
        if (delta_x_.squaredNorm() <= 1e-6 || false_cnt > 10)
        {
            stop = true;
            break;
        }

        // 更新状态量 X = X+ delta_x
        UpdateStates();
        // 判断当前步是否可行以及 LM 的 lambda 怎么更新
        oneStepSuccess = IsGoodStepInLM();

        out<<currentLambda_<<endl;
```

记录所有的lambda值

```
        // 后续处理,
        if (oneStepSuccess)
        {
            // 在新线性化点 构建 hessian
            MakeHessian();
            // TODO:: 这个判断条件可以丢掉, 条件 b_max <= 1e-12 很难达到, 这里的阈值条件不应该用绝对值, 而是相对值
            //double b_max = 0.0;
            //for (int i = 0; i < b_.size(); ++i) {
            //b_max = max(fabs(b_(i)), b_max);
            //}
            // 优化退出条件2:  如果残差 b_max 已经很小了, 那就退出
            // stop = (b_max <= 1e-12);
            false_cnt = 0;
        }
        else
        {
            false_cnt++;
            RollbackStates(); // 误差没下降, 回滚
        }
    }
    iter++;

    // 优化退出条件3:  currentChi_ 跟第一次的chi2相比, 下降了 1e6 倍则退出
    if (sqrt(currentChi_) <= stopThresholdLM_)
        stop = true;
}
std::cout << "problem solve cost: " << t_solve.toc() << " ms" << std::endl;
std::cout << "   makeHessian cost: " << t_hessian_cost_ << " ms" << std::endl;

out.close();
return true;
```

关闭文件

编译给的代码,进入项目目录：
编译并且运行可执行文件：
$mdkir build
$cd build
$cmake ..
$make
$ ./app/testCurveFitting

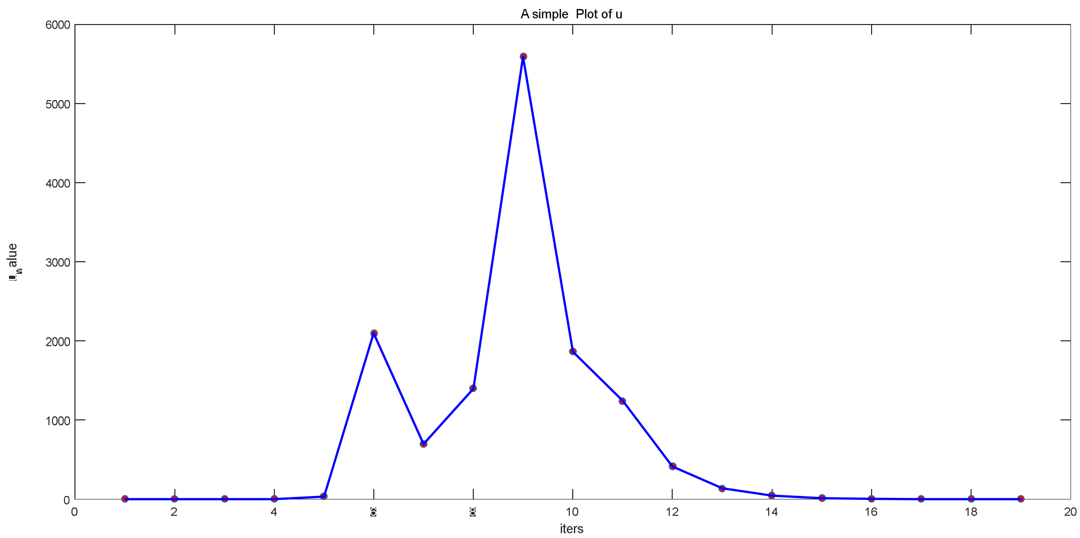
b.绘图脚本 draw.py

```python
import numpy as np
import matplotlib.pyplot as plt

data = []

for line in open("~/Downloads/CurveFitting_LM/build/app/u_data.txt","r"):
        data.append(line)
plt.plot(data,'b-',lw = 1.5)
plt.plot(data,'ro')
plt.grid(True)
plt.axis('tight')
plt.xlabel('iters')
plt.ylabel('u_value')
plt.title('A simple Plot of u')
plt.show()
```

c.最后画图曲线为：



u 值的图形

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。

代码修改如下：

```
    // 计算曲线模型误差
    virtual void ComputeResidual() override
    {
        //y = exp(ax2 + bx + c)
        // 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

        //y = exp(ax2 + bx + c)
        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
    {
        //y = exp(ax2 + bx + c)
        // 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_ * exp_y, x_ * exp_y , 1 * exp_y;
        // jacobians_[0] = jaco_abc;

        //y = exp(ax2 + bx + c)
        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;
    }
```

```
    double a=1.0, b=2.0, c=1.0;         // 真实参数值
    int N = 700;                          // 数据点
    double w_sigma= 1.;                 // 噪声Sigma值
```


```
    double a=1.0, b=2.0, c=1.0;         // 真实参数值
    int N = 700;                          // 数据点
    double w_sigma= 1.;                 // 噪声Sigma值
```


和 1 题的编译方法一样：

$ ./app/testCurveFitting

终端截图如下：

```
vslam@vslam:~/VIO_Tutorial/3.基于优化的IMU预积分与视觉信息融合/CurveFitting_LM/build$ ./app/testCurveFitting

Test CurveFitting start...
iter: 0 , chi= 653482 , Lambda= 3.34941
iter: 1 , chi= 696.818 , Lambda= 1.11647
iter: 2 , chi= 696.767 , Lambda= 0.372156
problem solve cost: 42.4039 ms
   makeHessian cost: 1.17797 ms
-------After optimization, we got these parameters :
 1.00192  1.98863 0.989354
-------ground truth:
1.0,  2.0,  1.0
vslam@vslam:~/VIO_Tutorial/3.基于优化的IMU预积分与视觉信息融合/CurveFitting_LM/build$
```

3.如果有实现其他阻尼因子更新策略可加分（选做）。

暂时未作

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：

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

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

解：

中值积分 a，和 w 为：

$$a=\frac{1}{2}\left(q_{b_ib_k}\left(a^{b_k}-b_k^a\right)+q_{b_ib_{k+1}}\left(a^{b_{k-1}}-b_k^a\right)\right)$$

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

求解 f_{15}

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

$$=\alpha_{b_ib_k}+\beta_{b_ib_k}\delta t+\frac{1}{4}\left(q_{b_ib_k}\left(a^{b_k}-b_k^a\right)+q_{b_ib_{k+1}}\begin{bmatrix}1\\-\frac{1}{2}\delta b_k^g\delta t\end{bmatrix}\left(a^{b_{k+1}}-b_k^a\right)\right)\delta t^2$$

$$f_{15}=\frac{\partial\alpha_{b_ib_{k+1}}}{\partial\delta b_k^g}=\frac{\partial\frac{1}{4}\left(q_{b_ib_k}\left(a^{b_k}-b_k^a\right)+q_{b_ib_{k+1}}\begin{bmatrix}1\\-\frac{1}{2}\delta b_k^g\delta t\end{bmatrix}\left(a^{b_{k+1}}-b_k^a\right)\right)\delta t^2}{\partial\delta b_k^g}$$

$$=\frac{\partial\frac{1}{4}\left(q_{b_ib_{k+1}}\begin{bmatrix}1\\-\frac{1}{2}\delta b_k^g\delta t\end{bmatrix}\left(a^{b_{k+1}}-b_k^a\right)\right)\delta t^2}{\partial\delta b_k^g}$$

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

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

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

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

求解 g_{12}

$$\alpha_{b_ib_{k+1}} = \alpha_{b_ib_k} + \beta_{b_ib_k}\delta t + \frac{1}{2}a\delta t^2$$
$$= \alpha_{b_ib_k} + \beta_{b_ib_k}\delta t + \frac{1}{4}\left(q_{b_ib_k}\left(a^{b_k} - b_k^a\right) + q_{b_ib_{k+1}}\begin{bmatrix}1 \\ \frac{1}{4}n_k^g\delta t\end{bmatrix}\left(a^{b_{k+1}} - b_k^a\right)\right)\delta t^2$$

$$g_{12} = \frac{\partial \alpha_{b_ib_{k+1}}}{\partial n_k^g} = \frac{\partial \frac{1}{4}\left(q_{b_ib_k}\left(a^{b_k} - b_k^a\right) + q_{b_ib_{k+1}}\begin{bmatrix}1 \\ \frac{1}{4}n_k^g\delta t\end{bmatrix}\left(a^{b_{k+1}} - b_k^a\right)\right)\delta t^2}{\partial n_k^g}$$
$$= \frac{\partial \frac{1}{4}\left(q_{b_ib_{k+1}}\begin{bmatrix}1 \\ \frac{1}{4}n_k^g\delta t\end{bmatrix}\left(a^{b_{k+1}} - b_k^a\right)\right)\delta t^2}{\partial n_k^g}$$
$$= \frac{\partial \frac{1}{4}\left(R_{b_ib_{k+1}} \exp\left(\frac{1}{2}n_k^g\delta t\right)_\times \left(a^{b_{k+1}} - b_k^a\right)\right)\delta t^2}{\partial n_k^g}$$
$$= \frac{\partial \frac{1}{4}\left(R_{b_ib_{k+1}\times} \left(I + \frac{1}{2}n_k^g\delta t\right)_\times \left(a^{b_{k+1}} - b_k^a\right)\right)\delta t^2}{\partial n_k^g}$$
$$= \frac{\partial \frac{1}{4}\left(R_{b_ib_{k+1}\times} \left(\frac{1}{2}n_k^g\delta t\right)_\times \left(a^{b_{k+1}} - b_k^a\right)\right)\delta t^2}{\partial n_k^g}$$
$$= \frac{\partial -\frac{1}{4} R_{b_ib_{k+1}} \left(a^{b_{k+1}} - b_k^a\right)\delta t^2{}_\times \left(\frac{1}{2}n_k^g\delta t\right)}{\partial n_k^g}$$
$$= -\frac{1}{4} R_{b_ib_{k+1}} \left(a^{b_{k+1}} - b_k^a\right)\delta t^2{}_\times \left(\frac{1}{2}\delta t\right)$$

3 证明式(9)。

$$\Delta \mathbf{x}_{\mathrm{lm}} = -\sum_{j=1}^{n} \frac{\mathbf{v}_j^{\top} \mathbf{F}'^{\top}}{\lambda_j + \mu} \mathbf{v}_j \qquad (9)$$

解：

阻尼因子 μ 大小是相对于 $\mathbf{J}^T\mathbf{J}$ 的元素而言的。半正定的信息矩阵 $\mathbf{J}^T\mathbf{J}$ 特征值 $\{\lambda_j\}$ 和对应的特征向量为 $\{\boldsymbol{v}_j\}$。对 $\mathbf{J}^T\mathbf{J}$ 做特征值分解分解后有： $\mathbf{J}^T\mathbf{J} = \mathbf{V\Lambda V}^T$：

则 $V = \left[v_1, v_2, v_3, \ldots, v_j\right]_{m*j}$ $\qquad$ $V^T = \begin{bmatrix} v_1^T \\ v_2^T \\ v_3^T \\ .. \\ v_j^T \end{bmatrix}_{j*m}$ $\qquad$ $\Lambda_{j*j} = diag(\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_j)$

故：

$$J^T J = V\Lambda V^T = \left[v_1, v_2, v_3, \ldots, v_j\right]_{m*j} diag(\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_j) \begin{bmatrix} v_1^T \\ v_2^T \\ v_3^T \\ \ldots \\ v_j^T \end{bmatrix}_{j*m} \qquad (1)$$

$$VV^T = \left[v_1, v_2, v_3, \ldots, v_j\right]_{m*j} \begin{bmatrix} v_1^T \\ v_2^T \\ v_3^T \\ \ldots \\ v_j^T \end{bmatrix}_{j*m} = I \qquad (2)$$

解如下方程：$\left(J^T J + \mu I\right)\Delta x = -J^T f = -F'^T$

由公式 1，2 解如下：

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

$$\left(V\Lambda V^T + V diag(\mu, \mu, \mu, .., \mu) V^T\right)\Delta x = -F'^T$$

$$\left(V\left(diag(\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_j) + diag(\mu, \mu, \mu, .., \mu)\right)V^T\right)\Delta x = -F'^T$$

$$\left(V diag(\lambda_1 + \mu, \lambda_2 + \mu, \lambda_3 + \mu, \ldots, \lambda_j + \mu) V^T\right)\Delta x = -F'^T$$

两边 $V diag(\lambda_1 + \mu, \lambda_2 + \mu, \lambda_3 + \mu, \ldots, \lambda_j + \mu)\ V^T$ 的逆矩阵：

$$\left(V diag(\lambda_1 + \mu, \lambda_2 + \mu, \lambda_3 + \mu, \ldots, \lambda_j + \mu)\ V^T\right)^{-1} = V diag(\lambda_1 + \mu, \lambda_2 + \mu, \lambda_3 + \mu, \ldots, \lambda_j + \mu)^{-1} V^T$$

$$\Delta x = -\left(V diag(\lambda_1+\mu, \lambda_2+\mu, \lambda_3+\mu, ..., \lambda_j+\mu) V^T \right)^{-1} F'^T$$

$$\Delta x = -V diag(\lambda_1+\mu, \lambda_2+\mu, \lambda_3+\mu, ..., \lambda_j+\mu)^{-1} V^T F'^T$$

$$\Delta x = -\left[v_1, v_2, v_3, \ldots, v_j \right]_{m*j} diag(\lambda_1+\mu, \lambda_2+\mu, \lambda_3+\mu, ..., \lambda_j+\mu)^{-1} \begin{bmatrix} v_1^T \\ v_2^T \\ v_3^T \\ ... \\ v_j^T \end{bmatrix}_{j*m} F'^T$$

$$= -\left[v_1, v_2, v_3, \ldots, v_j \right]_{m*j} diag(\lambda_1+\mu, \lambda_2+\mu, \lambda_3+\mu, ..., \lambda_j+\mu)^{-1} \begin{bmatrix} v_1^T \\ v_2^T \\ v_3^T \\ ... \\ v_j^T \end{bmatrix}_{j*m} F'^T$$

$$= -\left[v_1, v_2, v_3, \ldots, v_j \right]_{m*j} diag(\lambda_1+\mu, \lambda_2+\mu, \lambda_3+\mu, ..., \lambda_j+\mu)^{-1} \begin{bmatrix} v_1^T F'^T \\ v_2^T F'^T \\ v_3^T F'^T \\ ... \\ v_j^T F'^T \end{bmatrix}_{j*m}$$

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

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

注明：$v_j^T F'^T$为1×1标 量