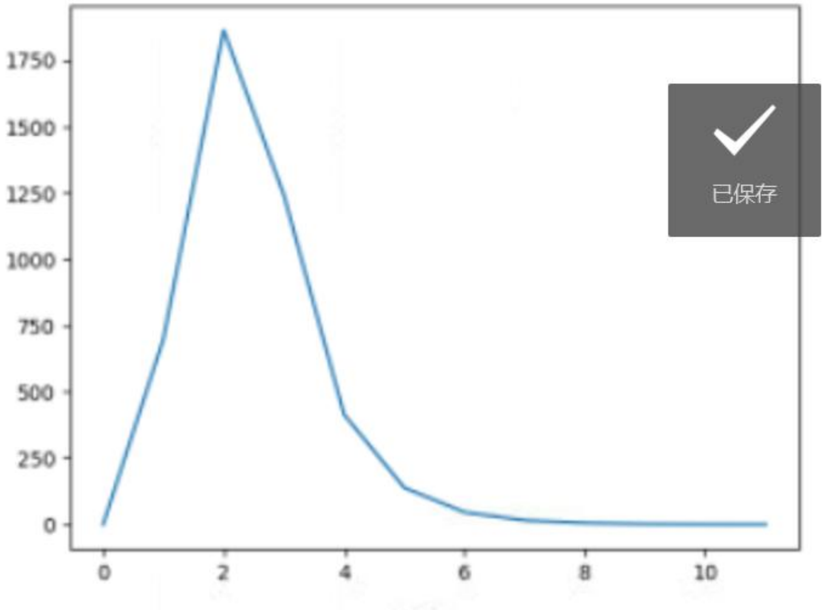
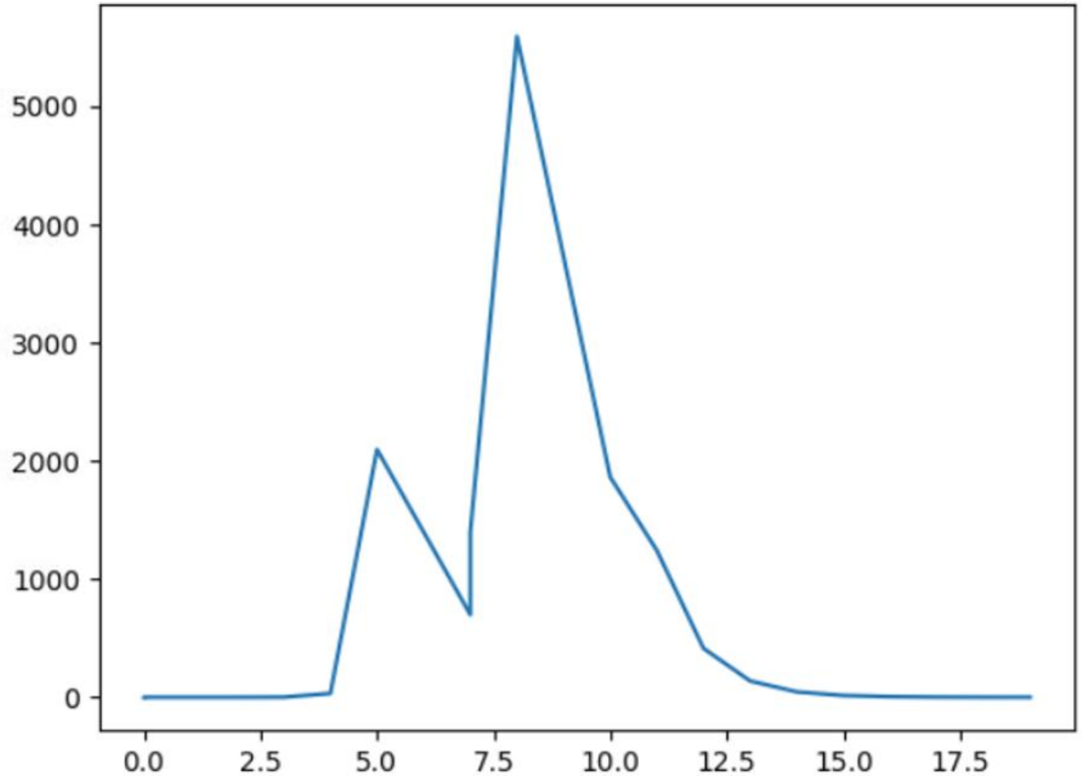
指数函数拟合的lambda变化


```cpp
    ofstream outfile;
    outfile.open("/home/w/Desktop/VIO_DIR/class3/a/CurveFitting_LM/utils/data.txt");

    while (!stop && (iter < iterations))
    {
        outfile << i2 << " " << currentLambda_ << endl;
        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
        {
            AddLambdatoHessianLM();
            SolveLinearSystem(); //解线性方程 H X = b
            RemoveLambdaHessianLM();

            if (delta_x_.squaredNorm() <= 1e-6 || false_cnt > 10)
            {
                stop = true;
                break;
            }
            UpdateStates();

            oneStepSuccess = IsGoodStepInLM(); // 判断当前步是否可行以及 LM 的 lambda 怎么更新

            if (oneStepSuccess)
            {
                MakeHessian(); // 在新线性化点 构建 hessian
                false_cnt = 0;
            }
            else
            {
                outfile << i2 << " " << currentLambda_ << endl;
                false_cnt++;
                i2++;
                RollbackStates(); // 误差没下降, 回滚
```

指数函数拟合的lambda变化

请绘制样例代码中 LM 阻尼因子 μ 随着迭代变化的曲线图：





初始值tau对lambda的影响

代码中使用的是LM论文中第三种策略：

Strategy:

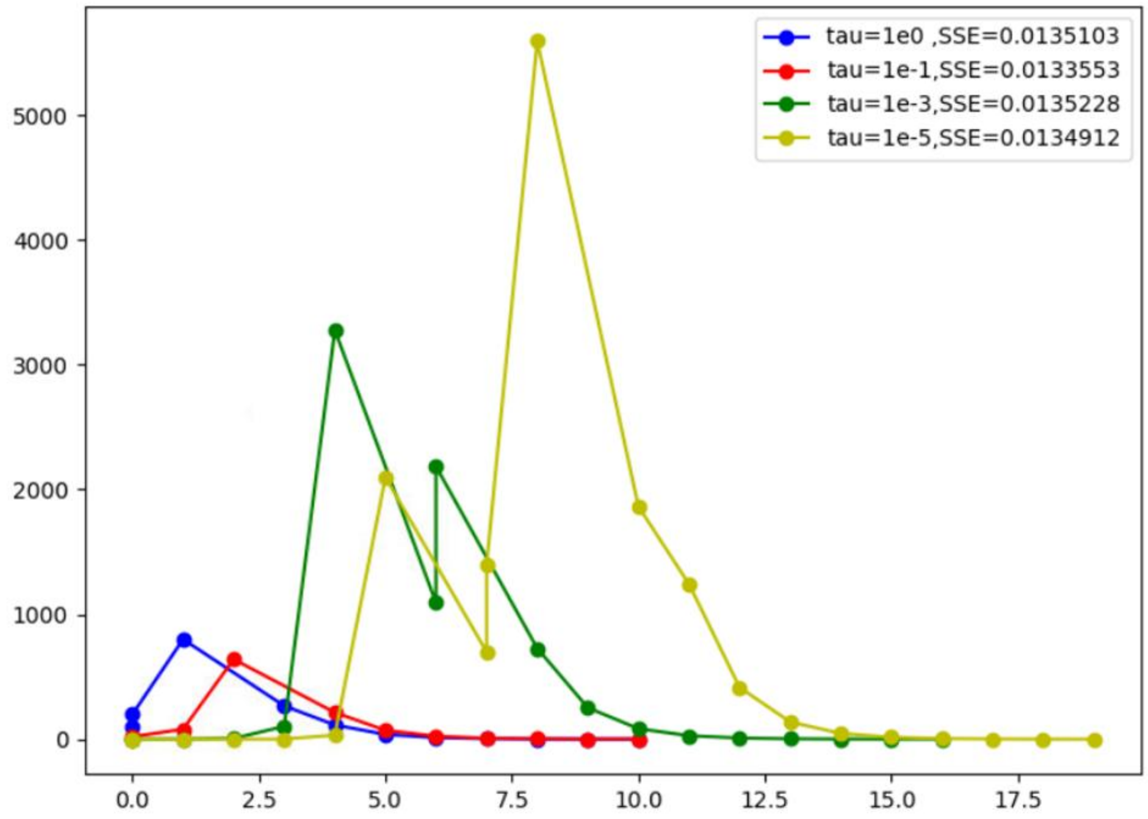
$\lambda_0 = \lambda_0 max[diag[J^T WJ]]$; λ_0 is user-specified

If $\rho_i(h) > \varepsilon$: $p \rightarrow p + h$; $\lambda_{i+1} = \lambda_i max\left[\frac{1}{3}, 1 - (2\rho_i - 1)^3\right]$; $\nu_i = 2$;

Otherwise: $\lambda_{i+1} = \lambda_i \nu_i$; $\nu_{i+1} = 2\nu_i$;

```
void Problem::ComputeLambdaInitLM()
{
    ni_ = 2.;
    currentLambda_ = -1.;
    currentChi_ = 0.0;

    for (auto edge : edges_)
    {
        currentChi_ += edge.second->Chi2();
    }
    if (err_prior_.rows() > 0)
        currentChi_ += err_prior_.norm();

    stopThresholdLM_ = 1e-6 * currentChi_; // 迭代条件为 误差下降 1e-6

    double maxDiagonal = 0;
    ulong size = Hessian_.cols();//3

    for (ulong i = 0; i < size; ++i)
    {
        maxDiagonal = std::max(fabs(Hessian_(i, i)), maxDiagonal);/
    }
    double tau = 1e-5;
    currentLambda_ = tau * maxDiagonal;
}
```

初始值tau对lambda的影响



二次函数拟合实验

深蓝学院
shenlanxueyuan.com

- 修改残差函数的计算、雅可比矩阵的计算、函数表达式生成数据

```
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 y=exp()...
    residual_(0) = abc(0) * x_ * x_ + abc(1) * x_ + abc(2) - y_;          //作业 y=axx+bx+c的残差
}
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_ * exp_y, x_ * exp_y, 1 * exp_y;  //y=exp()对a,b,c的导数
    jaco_abc <<x_* x_, x_,1;                  //  作业 y=axx+bx+c对a,b,c的导数.
    jacobians_[0] = jaco_abc;
}
```

```
        //double y = std::exp(a * x * x + b * x + c) + n; //指数
        double y = a * x * x + b * x + c + n; //作业,二次函数
```

二次函数拟合实验

原来的代码设置，当数据量N=100时,

当N=100,则x在区间[0,1]内,y在区间[1,4]内,在这个区间内比较扁,拟合的结果比较差：

a=1.61039 b=1.61853 c=0.995178

将w_sigma=0.1变小,a=1.305 b=1.81 c=0.997

将数据量N=1000加大,a=0.999 b=2.01,c=0.967

其他初始化策略：

1. $\lambda_0 = \lambda_{\mathsf{o}}$; λ_{o} is user-specified [8].
   use eq'n (13) for $\boldsymbol{h}_{\mathsf{lm}}$ and eq'n (16) for ρ
   if $\rho_i(\boldsymbol{h}) > \epsilon_4$: $\boldsymbol{p} \leftarrow \boldsymbol{p} + \boldsymbol{h}$; $\lambda_{i+1} = \max[\lambda_i / L_{\downarrow}, 10^{-7}]$;
   otherwise: $\lambda_{i+1} = \min[\lambda_i L_{\uparrow}, 10^7]$;

2. $\lambda_0 = \lambda_{\mathsf{o}} \max\left[\mathsf{diag}[\boldsymbol{J}^{\mathsf{T}} \boldsymbol{W} \boldsymbol{J}]\right]$; λ_{o} is user-specified.
   use eq'n (12) for $\boldsymbol{h}_{\mathsf{lm}}$ and eq'n (15) for ρ
   $\alpha = \left(\left(\boldsymbol{J}^{\mathsf{T}} \boldsymbol{W} (\boldsymbol{y} - \hat{\boldsymbol{y}}(\boldsymbol{p}))\right)^{\mathsf{T}} \boldsymbol{h}\right) / \left((\chi^2(\boldsymbol{p}+\boldsymbol{h}) - \chi^2(\boldsymbol{p}))/2 + 2\left(\boldsymbol{J}^{\mathsf{T}} \boldsymbol{W} (\boldsymbol{y} - \hat{\boldsymbol{y}}(\boldsymbol{p}))\right)^{\mathsf{T}} \boldsymbol{h}\right)$;
   if $\rho_i(\alpha\boldsymbol{h}) > \epsilon_4$: $\boldsymbol{p} \leftarrow p + \alpha\boldsymbol{h}$; $\lambda_{i+1} = \max[\lambda_i/(1+\alpha), 10^{-7}]$;
   otherwise: $\lambda_{i+1} = \lambda_i + |\chi^2(\boldsymbol{p} + \alpha\boldsymbol{h}) - \chi^2(\boldsymbol{p})|/(2\alpha)$;

3. $\lambda_0 = \lambda_{\mathsf{o}} \max\left[\mathsf{diag}[\boldsymbol{J}^{\mathsf{T}} \boldsymbol{W} \boldsymbol{J}]\right]$; λ_{o} is user-specified [9].
   use eq'n (12) for $\boldsymbol{h}_{\mathsf{lm}}$ and eq'n (15) for ρ
   if $\rho_i(\boldsymbol{h}) > \epsilon_4$: $\boldsymbol{p} \leftarrow \boldsymbol{p} + \boldsymbol{h}$; $\lambda_{i+1} = \lambda_i \max[1/3, 1 - (2\rho_i - 1)^3]$; $\nu_i = 2$;
   otherwise: $\lambda_{i+1} = \lambda_i \nu_i$; $\quad \nu_{i+1} = 2\nu_i$;

策略1：

部分关键代码：

```
bool Problem::IsGoodStepInLM()
{
    double scale = 0;
    scale = delta_x_.transpose() * (currentLambda_ *Hessian_.diagonal() * delta_x_ + b_);//LM论文的公式(16)的分母
    scale += 1e-3; // make sure it's non-zero :)

    //重新计算残差的平方和.
    double tempChi = 0.0;
    for (auto edge : edges_)
    {
        edge.second->ComputeResidual();
        tempChi += edge.second->Chi2(); //误差的平方和.
    }

    double rho = (currentChi_ - tempChi) / scale;

    if (rho > 0 && isfinite(tempChi))
    {
        currentLambda_=(std::max)(currentLambda_/9. , 1e-7);
        currentChi_ = tempChi;
        return true;
    }
    else
    {
        currentLambda_ =(std::min)(currentLambda_*11,1e7);
        return false;
    }
}
```

策略1：

指数函数：

```
w@w:~/Desktop/VIO_DIR/class3/a/CurveFitting_LM/build$ make
Scanning dependencies of target slam_course_backend
[ 16%] Building CXX object backend/CMakeFiles/slam_course_backend.dir/problem_1.cc.o
[ 33%] Linking CXX static library libslam_course_backend.a
[ 66%] Built target slam_course_backend
[ 83%] Linking CXX executable testCurveFitting
[100%] Built target testCurveFitting
w@w:~/Desktop/VIO_DIR/class3/a/CurveFitting_LM/build$ ./app/testCurveFitting
Test CurveFitting start...
iter: 0, chi= 36048.3, Lambda= 1
iter: 1, chi= 34219.5, Lambda= 13.4444
iter: 2, chi= 1141.81, Lambda= 1.49383
iter: 3, chi= 531.043, Lambda= 0.165981
iter: 4, chi= 365.945, Lambda= 0.0184423
iter: 5, chi= 133.522, Lambda= 0.00204915
iter: 6, chi= 99.5329, Lambda= 0.000227683
iter: 7, chi= 91.9421, Lambda= 2.52981e-05
iter: 8, chi= 91.3974, Lambda= 2.8109e-06
iter: 9, chi= 91.3959, Lambda= 3.12322e-07
problem solve cost: 2.7422 ms
makeHessian cost: 1.70599 ms
-------After optimization, we got these parameters :
0.941903  2.09458 0.965571
-------ground truth:
1.0,  2.0,  1.0
```

策略1：

二次函数：

```
[100%] Built target testCurveFitting
w@w:~/Desktop/VIO_DIR/class3/a/CurveFitting_LM/build$ ./app/testCurveFitting
Test CurveFitting start...
iter: 0, chi= 3.21386e+06, Lambda= 1
iter: 1, chi= 343813, Lambda= 0.111111
iter: 2, chi= 14132.9, Lambda= 0.0123457
iter: 3, chi= 1713.77, Lambda= 0.00137174
iter: 4, chi= 985.199, Lambda= 0.000152416
iter: 5, chi= 973.883, Lambda= 1.69351e-05
iter: 6, chi= 973.88, Lambda= 1.88168e-06
problem solve cost: 2.84369 ms
makeHessian cost: 2.19292 ms
-------After optimization, we got these parameters :
0.999589  2.00629 0.968813
-------ground truth:
1.0,  2.0,  1.0
```

策略2：

1. $\lambda_0 = \lambda_{\mathsf{o}}$; λ_{o} is user-specified [8].
   use eq'n (13) for $\boldsymbol{h}_{\mathsf{lm}}$ and eq'n (16) for ρ
   if $\rho_i(\boldsymbol{h}) > \epsilon_4$: $\boldsymbol{p} \leftarrow \boldsymbol{p} + \boldsymbol{h}$; $\lambda_{i+1} = \max[\lambda_i / L_{\downarrow}, 10^{-7}]$;
   otherwise: $\lambda_{i+1} = \min[\lambda_i L_{\uparrow}, 10^{7}]$;

2. $\lambda_0 = \lambda_{\mathsf{o}} \max\left[\mathsf{diag}[\boldsymbol{J}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{J}]\right]$; λ_{o} is user-specified.
   use eq'n (12) for $\boldsymbol{h}_{\mathsf{lm}}$ and eq'n (15) for ρ
   $\alpha = \left(\left(\boldsymbol{J}^{\mathsf{T}}\boldsymbol{W}(\boldsymbol{y} - \hat{\boldsymbol{y}}(\boldsymbol{p}))\right)^{\mathsf{T}} \boldsymbol{h}\right) / \left(\left(\chi^2(\boldsymbol{p} + \boldsymbol{h}) - \chi^2(\boldsymbol{p})\right)/2 + 2\left(\boldsymbol{J}^{\mathsf{T}}\boldsymbol{W}(\boldsymbol{y} - \hat{\boldsymbol{y}}(\boldsymbol{p}))\right)^{\mathsf{T}} \boldsymbol{h}\right)$;
   if $\rho_i(\alpha\boldsymbol{h}) > \epsilon_4$: $\boldsymbol{p} \leftarrow p + \alpha\boldsymbol{h}$; $\lambda_{i+1} = \max[\lambda_i/(1+\alpha), 10^{-7}]$;
   otherwise: $\lambda_{i+1} = \lambda_i + |\chi^2(\boldsymbol{p} + \alpha\boldsymbol{h}) - \chi^2(\boldsymbol{p})|/(2\alpha)$;

3. $\lambda_0 = \lambda_{\mathsf{o}} \max\left[\mathsf{diag}[\boldsymbol{J}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{J}]\right]$; λ_{o} is user-specified [9].
   use eq'n (12) for $\boldsymbol{h}_{\mathsf{lm}}$ and eq'n (15) for ρ
   if $\rho_i(\boldsymbol{h}) > \epsilon_4$: $\boldsymbol{p} \leftarrow \boldsymbol{p} + \boldsymbol{h}$; $\lambda_{i+1} = \lambda_i \max[1/3, 1 - (2\rho_i - 1)^3]$; $\nu_i = 2$;
   otherwise: $\lambda_{i+1} = \lambda_i \nu_i$; $\quad \nu_{i+1} = 2\nu_i$;

策略2:


```cpp
bool Problem::IsGoodStepInLM()
{
    double tempChi = 0.0;
    for (auto edge : edges_)
    {
        edge.second->ComputeResidual();
        tempChi += edge.second->Chi2();
    }
    double scale = 0;
    Eigen::MatrixXd JWfh(1, 1);
    JWfh = b_.transpose() * delta_x_;
    double d_JWfh = JWfh(0, 0);
    double alpha = d_JWfh / ((currentChi_ - tempChi) / 2 + 2 * d_JWfh);
    delta_x_ *= alpha;
    scale = (alpha * delta_x_).transpose() * (currentLambda_ * delta_x_ * alpha + b_);
    scale += 1e-3; // make sure it's non-zero :)

    RollbackStates();
    UpdateStates();
    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_ = (std::max)(pow(10, -7), currentLambda_ / (1 + alpha));
        currentChi_ = tempChi;
        return true;
    }
    else
    {
        currentLambda_ += std::abs(tempChi - currentChi_) / (2 * alpha);
        RollbackStates();
        return false;
    }
```

策略2：


```
w@w:~/Desktop/VIO_DIR/class3/a/CurveFitting_LM/build$ make
Scanning dependencies of target slam_course_backend
[ 16%] Building CXX object backend/CMakeFiles/slam_course_backend.dir/problem_2.
cc.o
[ 33%] Linking CXX static library libslam_course_backend.a
[ 66%] Built target slam_course_backend
[ 83%] Linking CXX executable testCurveFitting
[100%] Built target testCurveFitting
w@w:~/Desktop/VIO_DIR/class3/a/CurveFitting_LM/build$ ./app/testCurveFitting
Test CurveFitting start...
iter: 0 , chi= 3.21386e+06 , Lambda= 19.95
iter: 1 , chi= 3.21386e+06 , Lambda= 13.3
iter: 2 , chi= 3.21386e+06 , Lambda= 9.35928
iter: 3 , chi= 3.21386e+06 , Lambda= 7.42248
iter: 4 , chi= 3.21386e+06 , Lambda= 6.56868
iter: 5 , chi= 3.21386e+06 , Lambda= 6.08372
iter: 6 , chi= 3.21386e+06 , Lambda= 5.72601
iter: 7 , chi= 3.21386e+06 , Lambda= 5.43387
iter: 8 , chi= 3.21386e+06 , Lambda= 5.18617
iter: 9 , chi= 3.21386e+06 , Lambda= 4.97185
iter: 10 , chi= 3.21386e+06 , Lambda= 4.78366
iter: 11 , chi= 3.21386e+06 , Lambda= 4.61649
iter: 12 , chi= 3.21386e+06 , Lambda= 4.46657
iter: 13 , chi= 3.21386e+06 , Lambda= 4.33102
iter: 14 , chi= 3.21386e+06 , Lambda= 4.20763
iter: 15 , chi= 3.21386e+06 , Lambda= 4.09464
iter: 16 , chi= 3.21386e+06 , Lambda= 3.99062
iter: 17 , chi= 3.21386e+06 , Lambda= 3.89442
iter: 18 , chi= 3.21386e+06 , Lambda= 3.80508
iter: 19 , chi= 3.21386e+06 , Lambda= 3.72182
iter: 20 , chi= 3.21386e+06 , Lambda= 3.64396
iter: 21 , chi= 3.21386e+06 , Lambda= 3.57093
iter: 22 , chi= 3.21386e+06 , Lambda= 3.50224
iter: 23 , chi= 3.21386e+06 , Lambda= 3.43747
iter: 24 , chi= 3.21386e+06 , Lambda= 3.37625
iter: 25 , chi= 3.21386e+06 , Lambda= 3.31828
iter: 26 , chi= 3.21386e+06 , Lambda= 3.26325
iter: 27 , chi= 3.21386e+06 , Lambda= 3.21094
iter: 28 , chi= 3.21386e+06 , Lambda= 3.16113
iter: 29 , chi= 3.21386e+06 , Lambda= 3.1136
problem solve cost: 40.5925 ms
   makeHessian cost: 32.5209 ms
-------After optimization, we got these parameters :
 0.91409  1.83397 0.881693
-------ground truth:
1.0,  2.0,  1.0
```

证明题：

深蓝学院
shenlanxueyuan.com

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

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

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

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

证明题f15：

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

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

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

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

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

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

证明题g12：

$$\alpha_{b_i b_{k+1}} = \alpha_{b_i b_k} + \beta_{b_i b_k}\delta t + \frac{1}{2}a\delta t^2$$

$$= \alpha_{b_i b_k} + \beta_{b_i b_k}\delta t + \frac{1}{2}\left(\frac{1}{2}\left(q_{b_i b_k}(a^{b_k} - b_k^a) + q_{b_i b_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\omega\delta t \end{bmatrix}(a^{b_{k+1}} - b_k^a)\right)\right)\delta t^2$$

$$g_{12} = \frac{\partial \alpha_{b_i b_{k+1}}}{\partial n_k^g} = \frac{\partial \frac{1}{4} q_{b_i b_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\omega\delta t \end{bmatrix} \otimes \begin{bmatrix} 1 \\ \frac{1}{4} n_k^g \delta t \end{bmatrix}(a^{b_{k+1}} - b_k^a)\delta t^2}{\partial n_k^g}$$

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

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

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

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

证明题：

$$(J^T J+\mu I)\Delta x_{lm}=(V\Lambda V^T+\mu I)\Delta x_{lm}=(V(\Lambda+\mu I)V^T)\Delta x_{lm}=-J^T f=-F'^T$$

所以：

$$\Delta x_{lm}=-V(\Lambda+\mu I)^{-1}V^T F'^T=-\begin{bmatrix}v_1\, v_2 & \cdots & v_3\end{bmatrix}\begin{bmatrix}\frac{1}{\lambda_1+\mu} & & \cdots & \\ & \frac{1}{\lambda_2+\mu} & \cdots & \\ & & \ddots & \\ & & & \frac{1}{\lambda_n+\mu}\end{bmatrix}\begin{bmatrix}v_1^T\\ v_2^T\\ \vdots\\ v_n^T\end{bmatrix}F'^T$$

$$=-\begin{bmatrix}v_1\, v_2 & \cdots & v_3\end{bmatrix}\begin{bmatrix}\frac{v_1^T F'^T}{\lambda_1+\mu}\\ \frac{v_2^T F'^T}{\lambda_2+\mu}\\ \vdots\\ \frac{v_n^T F'^T}{\lambda_n+\mu}\end{bmatrix}=-\left(\frac{v_1^T F'^T}{\lambda_1+\mu}v_1+\frac{v_2^T F'^T}{\lambda_2+\mu}v_2+\cdots+\frac{v_n^T F'^T}{\lambda_n+\mu}v_n\right)=-\sum_{j=1}^{n}\frac{v_j^T F'^T}{\lambda_j+\mu}v_j$$

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening