视觉SLAM进阶：从零开始手写VIO

第三章：基于优化的 IMU 与视觉信息融合

第三章作业

深蓝学院 shenlanxueyuan.com

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。

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：

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

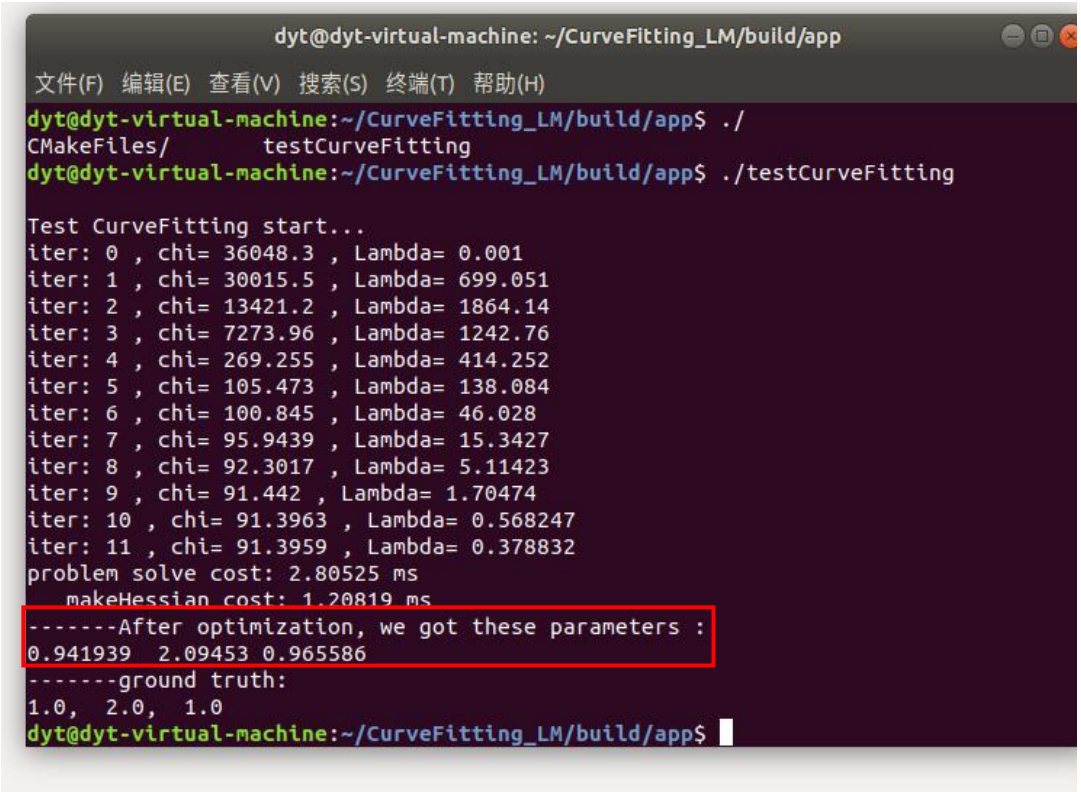
$$\mathbf{g}_{12} = \frac{\partial \boldsymbol{\alpha}_{b_i b_{k+1}}}{\partial \mathbf{n}_k^g} = -\frac{1}{4}(\mathbf{R}_{b_i b_{k+1}}[(\mathbf{a}^{b_{k+1}} - \mathbf{b}_k^a)]_\times \delta t^2)(\frac{1}{2}\delta t)$$

3 证明式(9)。

---

[a] Henri Gavin. "The Levenberg-Marquardt method for nonlinear least squares curve-fitting problems". In: *Department of Civil and Environmental Engineering, Duke University* (2011), pp. 1–15.

1.1 绘制样例代码中 LM 阻尼因子μ随着迭代变化的曲线图



1.1 绘制样例代码中 LM 阻尼因子μ随着迭代变化的曲线图

- 在backend的problem.cc中添加存储数据代码

```
// LM 算法迭代求解
bool stop = false;
int iter = 0;
while (!stop && (iter < iterations)) {
    std::cout << "iter: " << iter << " , chi= " << currentChi_ << " , Lambda= " << currentLambda_
              << std::endl;

    ofstream foutC("/home/dyt/CurveFitting_LM/output/result.txt", ios::app);
    foutC.setf(ios::fixed, ios::floatfield);
    foutC.precision(5);
    foutC << currentChi_ << " "
          << currentLambda_
          << endl;
```

- 得到对应的输出文本

```
36048.34457 0.00100
30015.51967 699.05067
13421.18650 1864.13511
7273.96333 1242.75674
269.25534 414.25225
105.47260 138.08408
100.84485 46.02803
95.94388 15.34268
92.30167 5.11423
91.44200 1.70474
91.39628 0.56825
91.39587 0.37883
```

1.1 绘制样例代码中 LM 阻尼因子μ随着迭代变化的曲线图

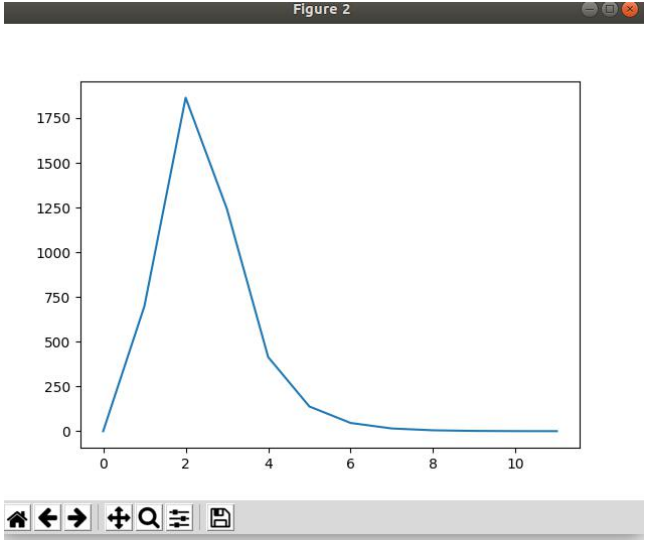
深蓝学院 shenlanxueyuan.com

- Python绘图

features2d.hpp | CurveFitting.cpp | problem.cc | display.

```
import matplotlib.pyplot as plt
import numpy as np;
import matplotlib
import os

Chi_ = []
Lambda_ = []


for line in open('result.txt', 'r'):
        values = [float(s) for s in line.split()]
        Chi_.append( values[0] )
        Lambda_.append( values[1] )

plt.figure(1)
plt.plot(Chi_)

plt.figure(2)
plt.plot(Lambda_)
plt.show()
```

- 得到如下结果



1.2 变换函数完成曲线参数估计

深蓝学院 shenlanxueyuan.com

- 修改观测y

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

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

        // 每个观测对应的残差函数
        shared_ptr< CurveFittingEdge > edge(new CurveFittingEdg
```

- 修改残差

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

- 修改雅克比矩阵

```
// 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;
}
```

1.2 变换函数完成曲线参数估计

- 得到结果

```
Test CurveFitting start...
iter: 0 , chi= 719.475 , Lambda= 0.001
iter: 1 , chi= 91.395 , Lambda= 0.000333333
problem solve cost: 2.48159 ms
   makeHessian cost: 0.115346 ms
-------After optimization, we got these parameters :
 1.61039  1.61853 0.995178
-------ground truth:
1.0,  2.0,  1.0
```

- 增大数据点数到1000

```
dyt@dyt-virtual-machine:~/CurveFitting_LM/build/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: 9.83355 ms
   makeHessian cost: 2.00736 ms
-------After optimization, we got these parameters :
0.999588   2.0063 0.968786
-------ground truth:
1.0,  2.0,  1.0
```

- 减小噪声均方差到0.1

```
dyt@dyt-virtual-machine:~/CurveFitting_LM/build/app$ ./testCurveFitting

Test CurveFitting start...
iter: 0 , chi= 614.937 , Lambda= 0.001
iter: 1 , chi= 0.913952 , Lambda= 0.000333333
iter: 2 , chi= 0.91395 , Lambda= 0.000222222
problem solve cost: 1.5669 ms
   makeHessian cost: 0.105625 ms
-------After optimization, we got these parameters :
 1.06107  1.96183 0.999517
-------ground truth:
1.0,  2.0,  1.0
```

1.3 实现其他阻尼因子策略

4 The Levenberg-Marquardt Method

The Levenberg-Marquardt algorithm adaptively varies the parameter updates between the gradient descent update and the Gauss-Newton update,

$$\left[\boldsymbol{J}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{J} + \lambda\boldsymbol{I}\right]\boldsymbol{h}_{\mathsf{lm}} = \boldsymbol{J}^{\mathsf{T}}\boldsymbol{W}(\boldsymbol{y}-\hat{\boldsymbol{y}}) \;, \tag{12}$$

where small values of the *damping parameter* λ result in a Gauss-Newton update and large values of λ result in a gradient descent update. The damping parameter λ is initialized to be large so that first updates are small steps in the steepest-descent direction. If any iteration happens to result in a worse approximation ($\chi^2(\boldsymbol{p}+\boldsymbol{h}_{\mathsf{lm}}) > \chi^2(\boldsymbol{p})$), then λ is increased. Otherwise, as the solution improves, λ is decreased, the Levenberg-Marquardt method approaches the Gauss-Newton method, and the solution typically accelerates to the local minimum [6, 7, 8].

In Marquardt's update relationship [8], the values of λ are normalized to the values of $\boldsymbol{J}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{J}$.

$$\left[\boldsymbol{J}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{J} + \lambda\,\mathsf{diag}(\boldsymbol{J}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{J})\right]\boldsymbol{h}_{\mathsf{lm}} = \boldsymbol{J}^{\mathsf{T}}\boldsymbol{W}(\boldsymbol{y}-\hat{\boldsymbol{y}}) \;, \tag{13}$$

4.1 Numerical Implementation

Many variations of the Levenberg-Marquardt have been published in papers and in code, e.g., [4, 6, 10, 11]. This document borrows from some of these. In iteration i, the step $\boldsymbol{h}$ is evaluated by comparing $\chi^2(\boldsymbol{p})$ to $\chi^2(\boldsymbol{p}+\boldsymbol{h})$. The step is accepted if the metric ρ_i [9] is greater than a user-specified threshold, $\epsilon_4 > 0$. This metric is a measure of the actual improvement in χ^2 as compared to the improvement of an LM update assuming the approximation (8) were exact.

$$\begin{aligned}
\rho_i(\boldsymbol{h}_{\mathsf{lm}}) &= \frac{\chi^2(\boldsymbol{p}) - \chi^2(\boldsymbol{p}+\boldsymbol{h}_{\mathsf{lm}})}{(\boldsymbol{y}-\hat{\boldsymbol{y}})^{\mathsf{T}}\boldsymbol{W}(\boldsymbol{y}-\hat{\boldsymbol{y}}) - (\boldsymbol{y}-\hat{\boldsymbol{y}}-\boldsymbol{J}\boldsymbol{h}_{\mathsf{lm}})^{\mathsf{T}}\boldsymbol{W}(\boldsymbol{y}-\hat{\boldsymbol{y}}-\boldsymbol{J}\boldsymbol{h}_{\mathsf{lm}})} \qquad (14)\\
&= \frac{\chi^2(\boldsymbol{p}) - \chi^2(\boldsymbol{p}+\boldsymbol{h}_{\mathsf{lm}})}{\boldsymbol{h}_{\mathsf{lm}}^{\mathsf{T}}\left(\lambda_i\boldsymbol{h}_{\mathsf{lm}} + \boldsymbol{J}^{\mathsf{T}}\boldsymbol{W}\left(\boldsymbol{y}-\hat{\boldsymbol{y}}(\boldsymbol{p})\right)\right)} \qquad \text{if using eq'n (12) for } \boldsymbol{h}_{\mathsf{lm}} \text{ (15)}\\
&= \frac{\chi^2(\boldsymbol{p}) - \chi^2(\boldsymbol{p}+\boldsymbol{h}_{\mathsf{lm}})}{\boldsymbol{h}_{\mathsf{lm}}^{\mathsf{T}}\left(\lambda_i\mathsf{diag}(\boldsymbol{J}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{J})\boldsymbol{h}_{\mathsf{lm}} + \boldsymbol{J}^{\mathsf{T}}\boldsymbol{W}\left(\boldsymbol{y}-\hat{\boldsymbol{y}}(\boldsymbol{p})\right)\right)} \qquad \text{if using eq'n (13) for } \boldsymbol{h}_{\mathsf{lm}} \text{(16)}
\end{aligned}$$

4.1.1 *Initialization and update of the L-M parameter, λ, and the parameters* $\mathbf{p}$

In `lm.m` users may select one of three methods for initializing and updating λ and $\boldsymbol{p}$.

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 $\rho_i(\boldsymbol{h}) > \epsilon_4$: $\boldsymbol{p} \leftarrow \boldsymbol{p}+\boldsymbol{h}$; $\lambda_{i+1} = \lambda_i \max\left[1/3, 1-(2\rho_i-1)^3\right]$; $\nu_i = 2$;
   otherwise: $\lambda_{i+1} = \lambda_i\nu_i$; $\quad \nu_{i+1} = 2\nu_i$;

For the examples in section 4.4, method 1 [8] with $L_{\uparrow} \approx 11$ and $L_{\downarrow} \approx 9$ exhibits good convergence properties.

1.3 实现其他阻尼因子策略（策略一）

深蓝学院 shenlanxueyuan.com

- 修改IsGoodStepInLM()函数

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

    ulong size = Hessian_.cols();
    MatXX diag_hessian(MatXX::Zero(size, size));
    for (ulong i = 0; i < size; ++i) {
        diag_hessian(i, i) = Hessian_(i, i);
    }

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

    double rho = (currentChi_ - tempChi) / scale;
//    std::cout << "rho = " << rho << std::endl;

    double L_down = 9.0;
    double L_up = 11.0;

    if (rho > 0 && isfinite(tempChi))   // last step was good, 误差在下降
    {
        currentLambda_ = std::max(currentLambda_ / L_down, 1e-7);
        currentChi_ = tempChi;
        return true;
    } else {
        currentLambda_ = std::min(currentLambda_ * L_up, 1e7);
        return false;
    }
```

1.3 实现其他阻尼因子策略（策略一）

深蓝学院 shenlanxueyuan.com

- 修改AddLambdatoHessianLM()

```
void Problem::AddLambdatoHessianLM() {
    ulong size = Hessian_.cols();
    assert(Hessian_.rows() == Hessian_.cols() && "Hessian is not square");
    for (ulong i = 0; i < size; ++i) {
        Hessian_(i, i) *= (1.+currentLambda_);
    }
}
```

- 修改RemoveLambdaHessianLM()

```
void Problem::RemoveLambdaHessianLM() {
    ulong size = Hessian_.cols();
    assert(Hessian_.rows() == Hessian_.cols() && "Hessian is not square");
    // TODO:: 这里不应该减去一个，数值的反复加减容易造成数值精度出问题？而应该保存叠加lambda前的值，在这里直接赋值
    for (ulong i = 0; i < size; ++i) {
        Hessian_(i, i) /= (1.+currentLambda_);
    }
}
```

1.3 实现其他阻尼因子策略（策略一）

深蓝学院 shenlanxueyuan.com

```
dyt@dyt-virtual-machine:~/CurveFitting_LM/build/app$ ./testCurveFitting

Test CurveFitting start...
iter: 0 , chi= 3.21386e+06 , Lambda= 19.95
iter: 1 , chi= 2.5285e+06 , Lambda= 2.21667
iter: 2 , chi= 626415 , Lambda= 0.246297
iter: 3 , chi= 41816.4 , Lambda= 0.0273663
iter: 4 , chi= 2648.89 , Lambda= 0.0030407
iter: 5 , chi= 1053.8 , Lambda= 0.000337856
iter: 6 , chi= 973.973 , Lambda= 3.75395e-05
iter: 7 , chi= 973.88 , Lambda= 4.17106e-06
problem solve cost: 7.05877 ms
   makeHessian cost: 2.83975 ms
-------After optimization, we got these parameters :
0.999584  2.00634 0.968725
-------ground truth:
1.0,  2.0,  1.0
```

1.3 实现其他阻尼因子策略（策略二）

深蓝学院 shenlanxueyuan.com

- 修改IsGoodStepInLM()函数

```
bool Problem::IsGoodStepInLM() {
    // 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 Numerator = b_.transpose() * delta_x_;
    double alpha = Numerator / ((currentChi_ - tempChi)/2. + 2.*Numerator);
    alpha = std::max(alpha, 1e-1);

    RollbackStates();
    delta_x_ *=alpha;
    UpdateStates();

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

    double scale = 0;
    scale = delta_x_.transpose() * (currentLambda_ * delta_x_ + b_);
    scale += 1e-3;    // make sure it's non-zero :)
    double rho = (currentChi_ - tempChi) / scale;
```

```
    double scale = 0;
    scale = delta_x_.transpose() * (currentLambda_ * delta_x_ + b_);
    scale += 1e-3;    // make sure it's non-zero :)
    double rho = (currentChi_ - tempChi) / scale;
//    std::cout << "rho = " << rho << std::endl;

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

1.3 实现其他阻尼因子策略（策略二）

深蓝学院 shenlanxueyuan.com

- AddLambdatoHessianLM()

```
void Problem::AddLambdatoHessianLM() {
    ulong size = Hessian_.cols();
    assert(Hessian_.rows() == Hessian_.cols() && "Hessian is not square");
    for (ulong i = 0; i < size; ++i) {
        Hessian_(i, i) *= (1.+currentLambda_);
    }
}
```

- RemoveLambdaHessianLM()

```
void Problem::RemoveLambdaHessianLM() {
    ulong size = Hessian_.cols();
    assert(Hessian_.rows() == Hessian_.cols() && "Hessian is not square");
    // TODO:: 这里不应该减去一个，数值的反复加减容易造成数值精度出问题？而应该保存叠加lambda前的值，在这里直接赋值
    for (ulong i = 0; i < size; ++i) {
        Hessian_(i, i) /= (1.+currentLambda_);
    }
}
```

1.3 实现其他阻尼因子策略（策略二）

深蓝学院 shenlanxueyuan.com

- 策略二结果

```
dyt@dyt-virtual-machine:~/CurveFitting_LM/build/app$ ./testCurveFitting

Test CurveFitting start...
iter: 0 , chi= 3.21386e+06 , Lambda= 19.95
iter: 1 , chi= 2.97101e+06 , Lambda= 14.887
iter: 2 , chi= 2.6832e+06 , Lambda= 11.0924
iter: 3 , chi= 2.35387e+06 , Lambda= 8.24987
iter: 4 , chi= 1.99321e+06 , Lambda= 6.12204
iter: 5 , chi= 1.61894e+06 , Lambda= 4.53104
iter: 6 , chi= 1.25478e+06 , Lambda= 3.34343
iter: 7 , chi= 925795 , Lambda= 2.45903
iter: 8 , chi= 651940 , Lambda= 1.80252
iter: 9 , chi= 442354 , Lambda= 1.31717
iter: 10 , chi= 293884 , Lambda= 0.960137
iter: 11 , chi= 194575 , Lambda= 0.698958
iter: 12 , chi= 129715 , Lambda= 0.508867
iter: 13 , chi= 86813.8 , Lambda= 0.370872
iter: 14 , chi= 57561.7 , Lambda= 0.270551
iter: 15 , chi= 37282.3 , Lambda= 0.197335
iter: 16 , chi= 23446.8 , Lambda= 0.143729
iter: 17 , chi= 14439.4 , Lambda= 0.104457
iter: 18 , chi= 8937.81 , Lambda= 0.0757406
iter: 19 , chi= 5781.26 , Lambda= 0.0548245
iter: 20 , chi= 4038.1 , Lambda= 0.0396753
iter: 21 , chi= 3061.13 , Lambda= 0.0287743
iter: 22 , chi= 2464.48 , Lambda= 0.0209598
iter: 23 , chi= 2051.09 , Lambda= 0.0153331
iter: 24 , chi= 1735.77 , Lambda= 0.011239
iter: 25 , chi= 1488.41 , Lambda= 0.00823319
iter: 26 , chi= 1300.27 , Lambda= 0.00601777
iter: 27 , chi= 1166.53 , Lambda= 0.00438505
iter: 28 , chi= 1079.31 , Lambda= 0.0031848
iter: 29 , chi= 1027.46 , Lambda= 0.00230573
problem solve cost: 53.0669 ms
   makeHessian cost: 22.9402 ms
-------After optimization, we got these parameters :
0.978324  2.22483 0.585006
-------ground truth:
1.0,  2.0,  1.0
```

1.3 实现其他阻尼因子策略（策略三）

深蓝学院 shenlanxueyuan.com

- 策略三

```
double rho = (currentChi_ - tempChi) / scale;
if (rho > 0 && isfinite(tempChi))   // last step was good, 误差在下降
{
    double alpha = 1. - pow((2 * rho - 1), 3);
    alpha = std::min(alpha, 2. / 3.);
    double scaleFactor = (std::max)(1. / 3., alpha);
    currentLambda_ *= scaleFactor;
    ni_ = 2;
    currentChi_ = tempChi;
    return true;
} else {
    currentLambda_ *= ni_;
    ni_ *= 2;
    return false;
}
```

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：

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

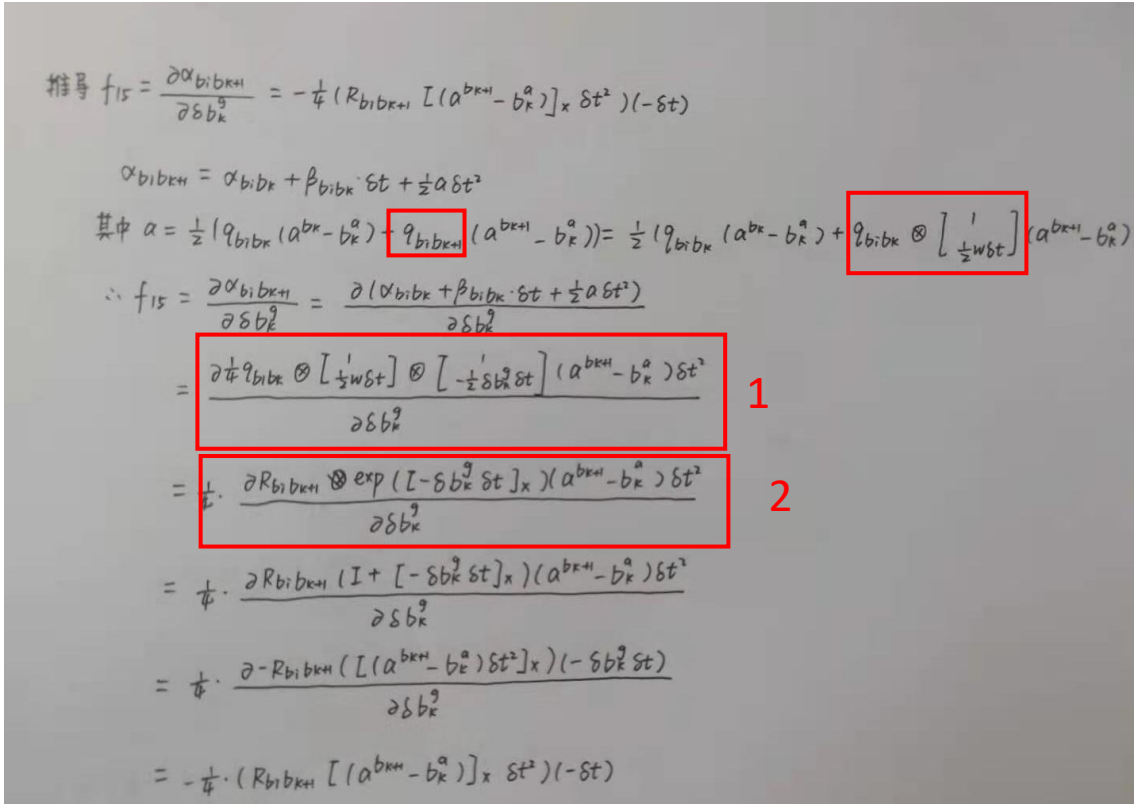
$$\mathbf{g}_{12} = \frac{\partial \boldsymbol{\alpha}_{b_i b_{k+1}}}{\partial \mathbf{n}_k^g} = -\frac{1}{4}(\mathbf{R}_{b_i b_{k+1}}[(\mathbf{a}^{b_{k+1}} - \mathbf{b}_k^a)]_\times \delta t^2)(\frac{1}{2}\delta t)$$

$$\mathbf{F} = \begin{bmatrix} \mathbf{I} & \mathbf{f}_{12} & \mathbf{I}\delta t & -\frac{1}{4}(\mathbf{q}_{b_i b_k} + \mathbf{q}_{b_i b_{k+1}})\delta t^2 & \mathbf{f}_{15} \\ \mathbf{0} & \mathbf{I} - [\boldsymbol{\omega}]_\times \delta t & \mathbf{0} & \mathbf{0} & -\mathbf{I}\delta t \\ \mathbf{0} & \mathbf{f}_{32} & \mathbf{I} & -\frac{1}{2}(\mathbf{q}_{b_i b_k} + \mathbf{q}_{b_i b_{k+1}})\delta t & \mathbf{f}_{35} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}$$

$$\mathbf{G} = \begin{bmatrix} \frac{1}{4}\mathbf{q}_{b_i b_k}\delta t^2 & \mathbf{g}_{12} & \frac{1}{4}\mathbf{q}_{b_i b_{k+1}}\delta t^2 & \mathbf{g}_{14} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \frac{1}{2}\mathbf{I}\delta t & \mathbf{0} & \frac{1}{2}\mathbf{I}\delta t & \mathbf{0} & \mathbf{0} \\ \frac{1}{2}\mathbf{q}_{b_i b_k}\delta t & \mathbf{g}_{32} & \frac{1}{2}\mathbf{q}_{b_i b_{k+1}}\delta t & \mathbf{g}_{34} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}\delta t & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}\delta t \end{bmatrix}$$

2 公式推导

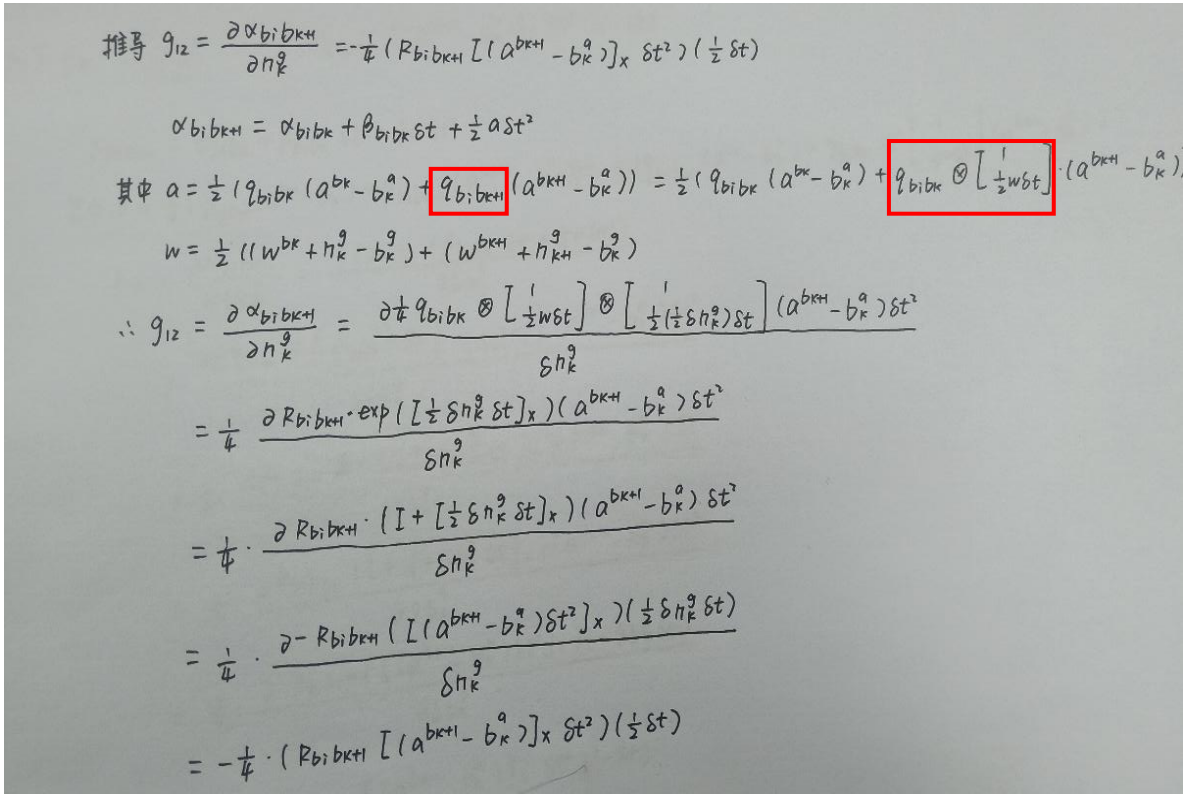
$$\mathbf{f}_{15} = \frac{\partial \boldsymbol{\alpha}_{b_i b_{k+1}}}{\partial \delta \mathbf{b}_k^g} = -\frac{1}{4}(\mathbf{R}_{b_i b_{k+1}}[(\mathbf{a}^{b_{k+1}} - \mathbf{b}_k^a)]_\times \delta t^2)(-\delta t)$$



$$\boldsymbol{\alpha}_{b_i b_{k+1}} = \boldsymbol{\alpha}_{b_i b_k} + \boldsymbol{\beta}_{b_i b_k}\delta t + \frac{1}{2}\mathbf{a}\delta t^2$$

2 公式推导

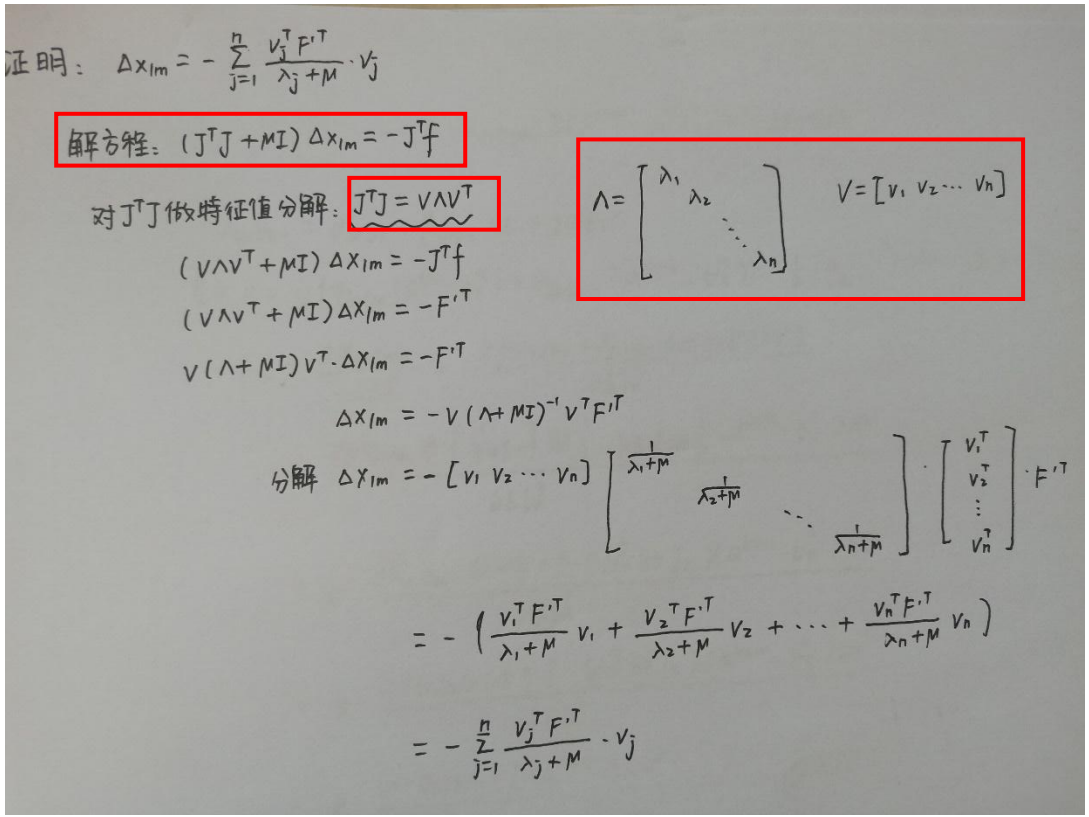
$$\mathbf{g}_{12} = \frac{\partial \boldsymbol{\alpha}_{b_i b_{k+1}}}{\partial \mathbf{n}_k^g} = -\frac{1}{4}(\mathbf{R}_{b_i b_{k+1}}[(\mathbf{a}^{b_{k+1}} - \mathbf{b}_k^a)]_\times \delta t^2)(\frac{1}{2}\delta t)$$



$$\boldsymbol{\alpha}_{b_i b_{k+1}} = \boldsymbol{\alpha}_{b_i b_k} + \boldsymbol{\beta}_{b_i b_k}\delta t + \frac{1}{2}\mathbf{a}\delta t^2$$

3 证明题

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



深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening