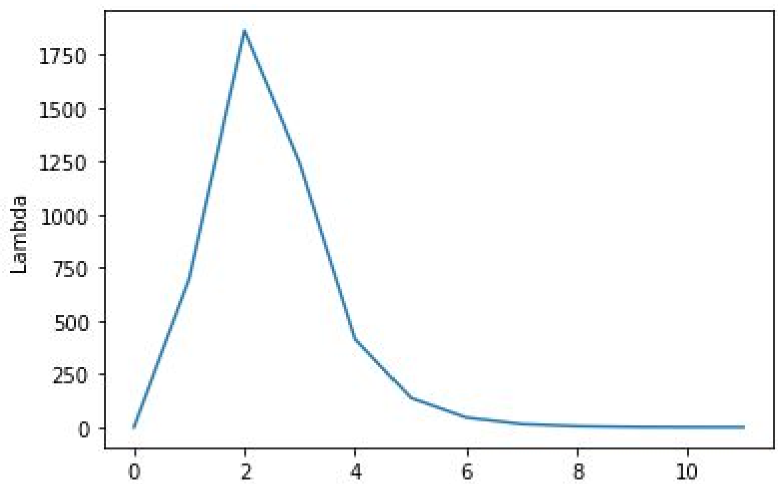
Week 3 作业

基于优化的IMU预积分与视觉信息融合

by WeihengXia

1. 代码修改

1.1. 绘制LM阻尼因子的变化图



1.2. 曲线参数估计

a) 代码修改: 残差和Jacobian

```
    // 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;
    }
```

b) 曲线估计：

```
❯ ./testCurveFitting

Test CurveFitting start...
iter: 0 , chi= 719.475 , Lambda= 0.001
iter: 1 , chi= 91.395 , Lambda= 0.000333333
problem solve cost: 0.164072 ms
   makeHessian cost: 0.101826 ms
-------After optimization, we got these parameters :
 1.61039  1.61853 0.995178
-------ground truth:
1.0,  2.0,  1.0
```

通过对比分析看出，此次曲线估计的结果不是很好。

原因有以下几点：

1) 数据量N只有100，可以将其适当增加到1000，再次计算结果如下：

```
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: 1.20385 ms
   makeHessian cost: 0.959858 ms
-------After optimization, we got these parameters :
0.999588   2.0063 0.968786
-------ground truth:
1.0,  2.0,  1.0
```

2) 初始化的a,b,c参数为0，可以适当进行估计，优化初始参数。

1.3. 其他阻尼因子策略

首先，通过阅读文献我发现有3种马夸尔特-阻尼因子策略，如下图所示：

4.1.1 Initialization and update of the L-M parameter, λ, and the parameters **p**

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

1. $\lambda_0 = \lambda_o$; λ_o is user-specified [8].
   use eq'n (13) for $\boldsymbol{h}_{\text{lm}}$ 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. $\lambda_0 = \lambda_o \max\left[\mathsf{diag}[\boldsymbol{J}^\mathsf{T}\boldsymbol{W}\boldsymbol{J}]\right]$; λ_o is user-specified.
   use eq'n (12) for $\boldsymbol{h}_{\text{lm}}$ 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. $\lambda_0 = \lambda_o \max\left[\mathsf{diag}[\boldsymbol{J}^\mathsf{T}\boldsymbol{W}\boldsymbol{J}]\right]$; λ_o is user-specified [9].
   use eq'n (12) for $\boldsymbol{h}_{\text{lm}}$ 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: $\lambda_{i+1} = \lambda_i \nu_i$; $\quad \nu_{i+1} = 2\nu_i$;

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

其次，在阅读三种策略后，我发现第三种策略Nielsen策略已经被示范代码实现了。其具体算法如下图所示：

$$
\begin{aligned}
&\text{if } \rho > 0 \\
&\qquad \mu := \mu * \max\left\{\tfrac{1}{3}, 1-(2\rho-1)^3\right\}; \quad \nu := 2 \\
&\text{else} \\
&\qquad \mu := \mu * \nu; \quad \nu := 2 * \nu
\end{aligned}
$$

算法3. Nielsen策略

为进行对比实验，我决定实现第一种策略：

1. $\lambda_0 = \lambda_o$; 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: $\lambda_{i+1} = \min[\lambda_i L_\uparrow, 10^7]$;

算法1. L策略（自己取的名字）

（Tip: 文章有注释，L_up=11和L_down=9对于曲线拟合的收敛有好的效果）

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

策略一中提到的公式(13)和(16)如下图所示：

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

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

第一种策略的实现代码如下：

```
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_);
    }
}

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_);
    }
}
```

```cpp
bool Problem::IsGoodStepInLM() {
    double scale = 0;
    // scale = delta_x_.transpose() * (currentLambda_ * delta_x_ + b_);
    scale = delta_x_.transpose() * (currentLambda_ * Hessian_.diagonal() * delta_x_ + b_); //参考论文公式(16)
    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;
    double L_down = 9.0;
    double L_up = 11.0;

    if (rho > 0 && isfinite(tempChi))   // last step was good, 误差在下降
    {
        currentChi_ = tempChi;
        currentLambda_ = std::max(currentLambda_/L_down, 1e-7);

        return true;
    } else {
        currentLambda_ = std::min(currentLambda_*L_up, 1e7);
        return false;
    }
```

第一种策略的二次函数拟合结果如下：

```
Test CurveFitting start...
iter: 0 , chi= 36048.3 , Lambda= 0.001
iter: 1 , chi= 34760.2 , Lambda= 17.8946
iter: 2 , chi= 8020.58 , Lambda= 1.98828
iter: 3 , chi= 779.997 , Lambda= 0.22092
iter: 4 , chi= 348.805 , Lambda= 0.0245467
iter: 5 , chi= 145.33 , Lambda= 0.00272741
iter: 6 , chi= 101 , Lambda= 0.000303046
iter: 7 , chi= 92.3181 , Lambda= 3.36718e-05
iter: 8 , chi= 91.3999 , Lambda= 3.74131e-06
iter: 9 , chi= 91.3959 , Lambda= 4.15701e-07
problem solve cost: 0.529022 ms
   makeHessian cost: 0.273964 ms
-------After optimization, we got these parameters :
0.941955   2.0945   0.9656
-------ground truth:
1.0,  2.0,  1.0
```

第一种策略的exp函数拟合结果如下：

```
Test CurveFitting start...
iter: 0 , chi= 36048.3 , Lambda= 0.001
iter: 1 , chi= 34760.2 , Lambda= 17.8946
iter: 2 , chi= 8020.58 , Lambda= 1.98828
iter: 3 , chi= 779.997 , Lambda= 0.22092
iter: 4 , chi= 348.805 , Lambda= 0.0245467
iter: 5 , chi= 145.33 , Lambda= 0.00272741
iter: 6 , chi= 101 , Lambda= 0.000303046
iter: 7 , chi= 92.3181 , Lambda= 3.36718e-05
iter: 8 , chi= 91.3999 , Lambda= 3.74131e-06
iter: 9 , chi= 91.3959 , Lambda= 4.15701e-07
problem solve cost: 1.58468 ms
   makeHessian cost: 1.02852 ms
-------After optimization, we got these parameters :
0.941955   2.0945   0.9656
-------ground truth:
1.0,  2.0,  1.0
```

2. 公式推导

（推不动了，下周再推）

3. 公式证明

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

证:

步骤1 - LM公式对高斯牛顿法改进，加入了阻尼因子，公式如下:

$$\left(\mathbf{J}^{\top}\mathbf{J} + \mu \mathbf{I}\right) \Delta \mathbf{x}_{\mathrm{lm}} = -\mathbf{J}^{\top}\mathbf{f} \quad \text{with} \quad \mu \geq 0$$

步骤2 - 对JT*J进行特征值分解，得到:

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

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

$$\Delta \mathbf{x}_{\mathrm{lm}} = -\mathbf{V}\left(\boldsymbol{\Lambda} + \mu \mathbf{I}\right)^{-1}\mathbf{V}^{\top}\mathbf{F}'^{\top}$$

步骤3 - 根据正规矩阵的谱分解得到:

$$\begin{aligned}
\Delta \mathbf{x}_{\mathrm{lm}} &= -\mathbf{V}(\boldsymbol{\Lambda} + \mu \mathbf{I})^{-1}\mathbf{V}^{\top}\mathbf{F}' \\
&= -\left[\begin{array}{llll} \mathbf{v}_1 \mathbf{v}_2 & \cdots & \mathbf{v}_3 \end{array}\right]
\left[\begin{array}{llll} \frac{1}{\lambda_1+\mu} & & & \\ & \frac{1}{\lambda_2+\mu} & & \\ & & \ddots & \\ & & & \frac{1}{\lambda_n+\mu} \end{array}\right]
\left[\begin{array}{c} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_n^T \end{array}\right] \mathbf{F}'^T \\
&= -\left[\begin{array}{llll} \mathbf{v}_1 \mathbf{v}_2 & \cdots & \mathbf{v}_3 \end{array}\right]
\left[\begin{array}{c} \frac{\mathbf{v}_1^T \mathbf{F}'^T}{\lambda_1+\mu} \\ \frac{\mathbf{v}_2^T \mathbf{F}'^T}{\lambda_2+\mu} \\ \vdots \\ \frac{\mathbf{v}_n^T \mathbf{F}'^T}{\lambda_n+\mu} \end{array}\right] \\
&= -\left(\frac{\mathbf{v}_1^T \mathbf{F}'^T}{\lambda_1+\mu}\mathbf{v}_1 + \frac{\mathbf{v}_2^T \mathbf{F}'^T}{\lambda_2+\mu}\mathbf{v}_2 + \cdots + \frac{\mathbf{v}_n^T \mathbf{F}'^T}{\lambda_n+\mu}\mathbf{v}_n\right) = -\sum_{j=1}^{n} \frac{\mathbf{v}_j^T \mathbf{F}'^T}{\lambda_j+\mu}\mathbf{v}_j
\end{aligned}$$