深蓝学院
shenlanxueyuan.com

VIO第三章作业分享

主讲人 Jindong Shi

纲要

- **1. 使用LM算法估计曲线参数**
  - 1.1 绘制阻尼因子μ随着迭代变化的曲线图
  - 1.2 将曲线函数改成$y = ax^2 + bx + c$
  - 1.3 实现更优秀的阻尼因子策略
- **2. 公式推导雅可比F, G中的两项**
- **3. 证明式(9)**

1.使用LM算法估计曲线参数

1.1 绘制阻尼因子μ随着迭代变化的曲线图

这里使用Pangolin来绘制
损失函数currentChi 和
阻尼因子currentLambda
随着迭代步骤iter变化的曲线图。
我们在backend/problem.cc中
Problem类内写入draw_curve函数

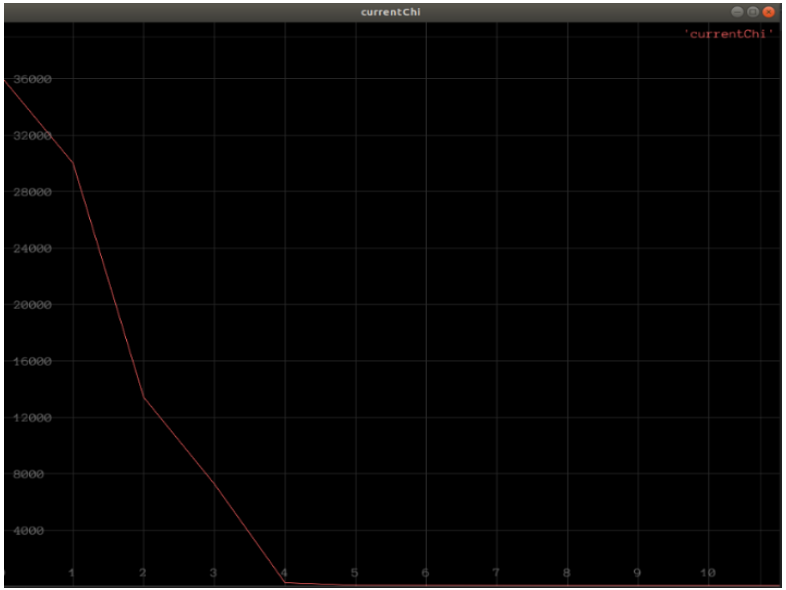
```cpp
void Problem::draw_curve(const vector<double, std::allocator<double>> &currentValue_vec, string name){
    // 参考 Pangolin Tutorial: https://github.com/yuntianli91/pangolin_tutorial/blob/master/task5/main.cpp
    // Create OpenGL window in single line
    pangolin::CreateWindowAndBind( window_title: name, w: 640, h: 480);
    // Data logger object
    pangolin::DataLog log;
    // Optionally add named labels
    std::vector<std::string> labels;
    labels.push_back(std::string(name));
    log.SetLabels(labels);
    // OpenGL 'view' of data. We might have many views of the same data.
    auto maxValue :iterator<...> = max_element( first: currentValue_vec.begin(), last: currentValue_vec.end());
    pangolin::Plotter plotter( default_log: &log, left: 0, right: currentValue_vec.size() - 1, bottom: 0, top: *maxValue, tickx: 1, ticky: *maxValue/20);
    plotter.SetBounds( bottom: 0.0, top: 1.0, left: 0.0, right: 1.0);
    // plotter.Track("$i");//坐标轴自动滚动
    pangolin::DisplayBase().AddDisplay( &: plotter);
    // Default hooks for exiting (Esc) and fullscreen (tab).
    unsigned iter = 0;
    while( !pangolin::ShouldQuit() )
    {
        glClear( mask: GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        if(iter < currentValue_vec.size()){
            log.Log( v: currentValue_vec[iter]);
        }
        iter++;

        // Render graph, Swap frames and Process Events
        pangolin::FinishFrame();
    }
}
```

1.使用LM算法估计曲线参数

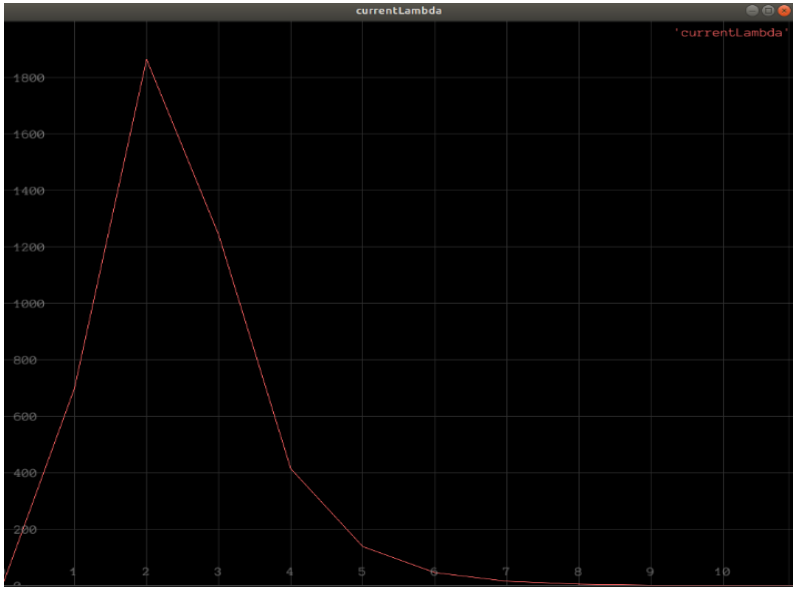
深蓝学院 shenlanxueyuan.com

1.1 绘制阻尼因子μ随着迭代变化的曲线图

损失函数变化:



阻尼因子变化:



1.使用LM算法估计曲线参数

1.2 将曲线函数改成$y = ax^2 + bx + c$

我们改变残差函数构建和残差对变量的雅可比:

```
// 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( residual_dimension: 1, num_vertices: 1,  vertices_types: std::vector<std::string>{"abc"}) {
        x_ = x;
        y_ = y;
    }
    // 计算曲线模型误差
    virtual void ComputeResidual() override
    {
        Vec3 abc = vertices_[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_( index: 0) = abc( index: 0)*x_*x_ + abc( index: 1)*x_ + abc( index: 2) - y_;  // 构建残差
    }

    // 计算残差对变量的雅克比
    virtual void ComputeJacobians() override
    {
        // Vec3 abc = vertices_[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;
        jaco_abc << x_ * x_, x_, 1;

        jacobians_[0] = jaco_abc;
    }
```

```
Test CurveFitting start...
iter: 0 , chi= 719.475 , Lambda= 0.001
iter: 1 , chi= 91.395 , Lambda= 0.000333333
problem solve cost: 2.68313 ms
   makeHessian cost: 2.0453 ms
-------After optimization, we got these parameters :
 1.61039  1.61853 0.995178
-------ground truth:
1.0,  2.0,  1.0
```

我们发现只有两步迭代，迭代结束后的残差仍然较大，得到的曲线参数离真实值误差较大。

1.使用LM算法估计曲线参数

1.3 实现更优秀的阻尼因子策略

新建变量update_strategy_用来存放用户想要输入的LM算法参数更新方法，可输入1, 2 或 3:

```
bool Problem::Solve(int iterations) {

    cout << "Please enter a LM parameter update strategy (1, 2 or 3):" << endl;
    cin >> update_strategy_;
```

首先是往Hessian矩阵中加入阻尼因子 λ 和去除 λ，根据参数更新方法不同，有不同的做法:

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

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

```
void Problem::AddLambdatoHessianLM() {
    ulong size = Hessian_.cols();
    assert(Hessian_.rows() == Hessian_.cols() && "Hessian is not square");
    for (ulong i = 0; i < size; ++i) {
        if(update_strategy_ == 1)
            // Hessian_(i, i) += currentLambda_ * Hessian_(i, i);
            Hessian_( row: i,  col: i) *= (1.+currentLambda_);
        else if(update_strategy_ == 2 || update_strategy_ == 3)
            Hessian_( row: i,  col: i) += currentLambda_;
        else{}
    }
}

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) {
        if(update_strategy_ ==1)
            // Hessian_(i, i) -= currentLambda_ * Hessian_(i, i);     // false remove
            Hessian_( row: i,  col: i) /= (1.+currentLambda_);
        else if(update_strategy_ == 2 || update_strategy_ == 3)
            Hessian_( row: i,  col: i) -= currentLambda_;
        else{}
    }
}
```

1.使用LM算法估计曲线参数

1.3 实现更优秀的阻尼因子策略

方法一:

1. $\lambda_0 = \lambda_o$; λ_o is user-specified [5].
   use eq'n (13) for $\mathbf{h}_{\text{lm}}$ and eq'n (16) for ρ
   if $\rho_i(\mathbf{h}) > \epsilon_4$: $\mathbf{p} \leftarrow \mathbf{p} + \mathbf{h}$; $\lambda_{i+1} = \max[\lambda_i / L_{\downarrow}, 10^{-7}]$;
   otherwise: $\lambda_{i+1} = \min[\lambda_i L_{\uparrow}, 10^{7}]$;

$$\rho_i(\mathbf{h}_{\text{lm}}) = \frac{\chi^2(\mathbf{p}) - \chi^2(\mathbf{p}+\mathbf{h}_{\text{lm}})}{(\mathbf{y}-\hat{\mathbf{y}})^{\mathsf{T}}(\mathbf{y}-\hat{\mathbf{y}}) - (\mathbf{y}-\hat{\mathbf{y}}-\mathbf{J}\mathbf{h}_{\text{lm}})^{\mathsf{T}}(\mathbf{y}-\hat{\mathbf{y}}-\mathbf{J}\mathbf{h}_{\text{lm}})} \tag{14}$$

$$= \frac{\chi^2(\mathbf{p}) - \chi^2(\mathbf{p}+\mathbf{h}_{\text{lm}})}{\mathbf{h}_{\text{lm}}^{\mathsf{T}}(\lambda_i \mathbf{h}_{\text{lm}} + \mathbf{J}^{\mathsf{T}}\mathbf{W}(\mathbf{y}-\hat{\mathbf{y}}(\mathbf{p})))} \quad \text{if using eq'n (12) for } \mathbf{h}_{\text{lm}} \tag{15}$$

$$= \frac{\chi^2(\mathbf{p}) - \chi^2(\mathbf{p}+\mathbf{h}_{\text{lm}})}{\mathbf{h}_{\text{lm}}^{\mathsf{T}}(\lambda_i \text{diag}(\mathbf{J}^{\mathsf{T}}\mathbf{W}\mathbf{J})\mathbf{h}_{\text{lm}} + \mathbf{J}^{\mathsf{T}}\mathbf{W}(\mathbf{y}-\hat{\mathbf{y}}(\mathbf{p})))} \quad \text{if using eq'n (13) for } \mathbf{h}_{\text{lm}} \tag{16}$$

其中参数 $L_{\uparrow}$ 和 $L_{\downarrow}$ 根据文献可设为$L_{\uparrow} = 11, L_{\downarrow} = 9.$

```cpp
bool Problem::IsGoodStepInLM() {
    double scale = 0;
    double rho = 0;

    // recompute residuals after update state
    // 统计所有的残差
    double tempChi = 0.0;
    for (auto edge :pair<...> : edges_) {
        edge.second->ComputeResidual();
        tempChi += edge.second->Chi2();
    }

    if(update_strategy_ ==1)
    {
        unsigned Hessian_size = Hessian_.rows();
        MatXX diag_Hessian = MatXX::Zero( rows: Hessian_size,  cols: Hessian_size);
        for (ulong i = 0; i < Hessian_size; ++i) {
            diag_Hessian( row: i,  col: i) = Hessian_( row: i,  col: i);
        }
        scale = delta_x_.transpose() * (currentLambda_ * diag_Hessian * delta_x_ + b_);
        scale += 1e-3;    // make sure it's non-zero :)
        rho = (currentChi_ - tempChi) / scale;

        double L_up = 11.0;
        double L_down = 9.0;
        if (rho > 0 && isfinite( x: 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.使用LM算法估计曲线参数

1.3 实现更优秀的阻尼因子策略

方法二:

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

方法二相较方法一更为复杂一些，我们需要先回滚Δx，然后再更新 α ·Δx 来计算残差和，然后根据计算出的rho和alpha来更新的阻尼因子 λ.

```
else if(update_strategy_ == 2)
{
    double alpha = double(b_.transpose() * delta_x_)/((tempChi - currentChi_)/2.0 + 2.0 * b_.transpose() * delta_x_);
    alpha = std::max(alpha, 1e-1);
    // cout << "alpha_factor = " << alpha_factor << endl;

    // 回滚delta_x_
    RollbackStates();
    // 回滚delta_x_之后，更新状态量 X = X + alpha * delta_x
    delta_x_ *= alpha;
    UpdateStates();

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

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

    rho = (currentChi_ - tempChi) / scale;

    if (rho > 0 && isfinite( x: 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( x: tempChi - currentChi_)/(2 * alpha);
        return false;
    }
}
```

1.使用LM算法估计曲线参数

深蓝学院 shenlanxueyuan.com

1.3 实现更优秀的阻尼因子策略

方法三:

3. $\lambda_0 = \lambda_o \max\left[\text{diag}[\mathbf{J}^\mathsf{T}\mathbf{W}\mathbf{J}]\right]$; λ_o is user-specified [6].
   use eq'n (12) for $\mathbf{h}_{\text{lm}}$ and eq'n (15) for ρ
   if $\rho_i(\mathbf{h}) > \epsilon_4$: $\mathbf{p} \leftarrow \mathbf{p} + \mathbf{h}$; $\lambda_{i+1} = \lambda_i \max\left[1/3, 1-(2\rho_i - 1)^3\right]$; $\nu_i = 2$;
   otherwise: $\lambda_{i+1} = \lambda_i \nu_i$; $\quad \nu_{i+1} = 2\nu_i$;

方法3是课上介绍的Nielsen策略，也是给出的原代码中实现的参数更新方法。

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

    if (rho > 0 && isfinite( x: tempChi))   // last step was good, 误差在下降
    {
        double alpha = 1. - pow( x: (2 * rho - 1),  y: 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;
    }
}
else{
    cout << "Can not find suitable LM update methods!" << endl;
    return false;
}
```

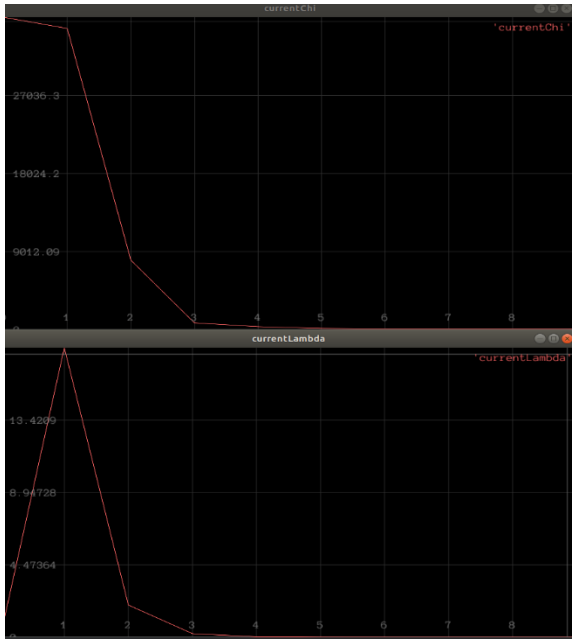
1.使用LM算法估计曲线参数

深蓝学院 shenlanxueyuan.com

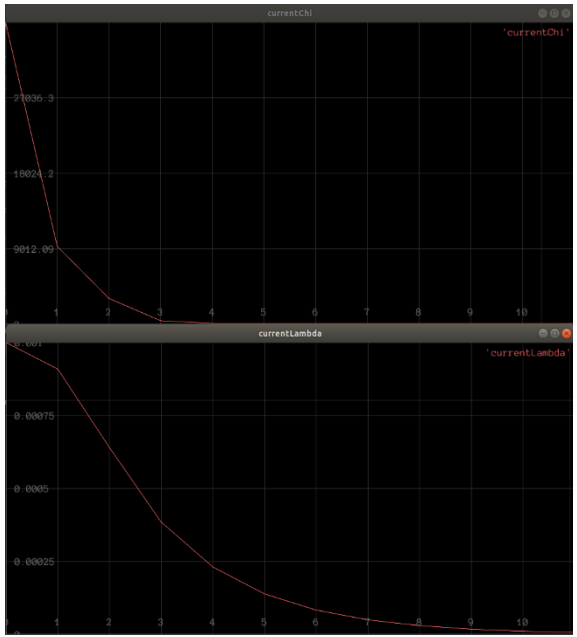
1.3 实现更优秀的阻尼因子策略

三种方法阻尼因子变化情况进行比较:

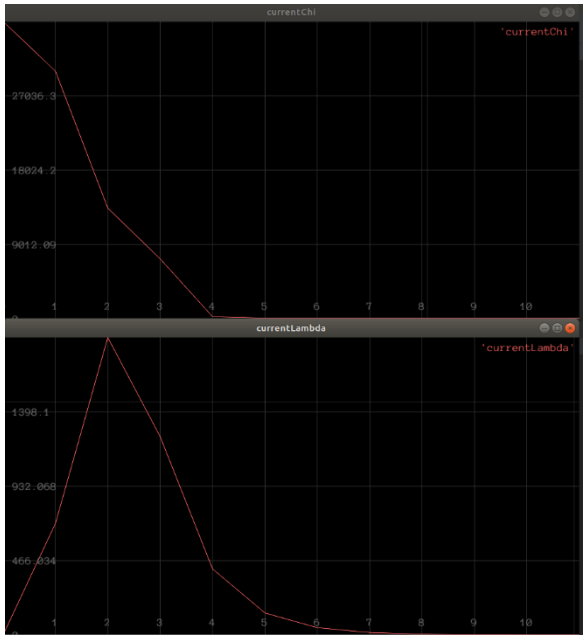
方法一



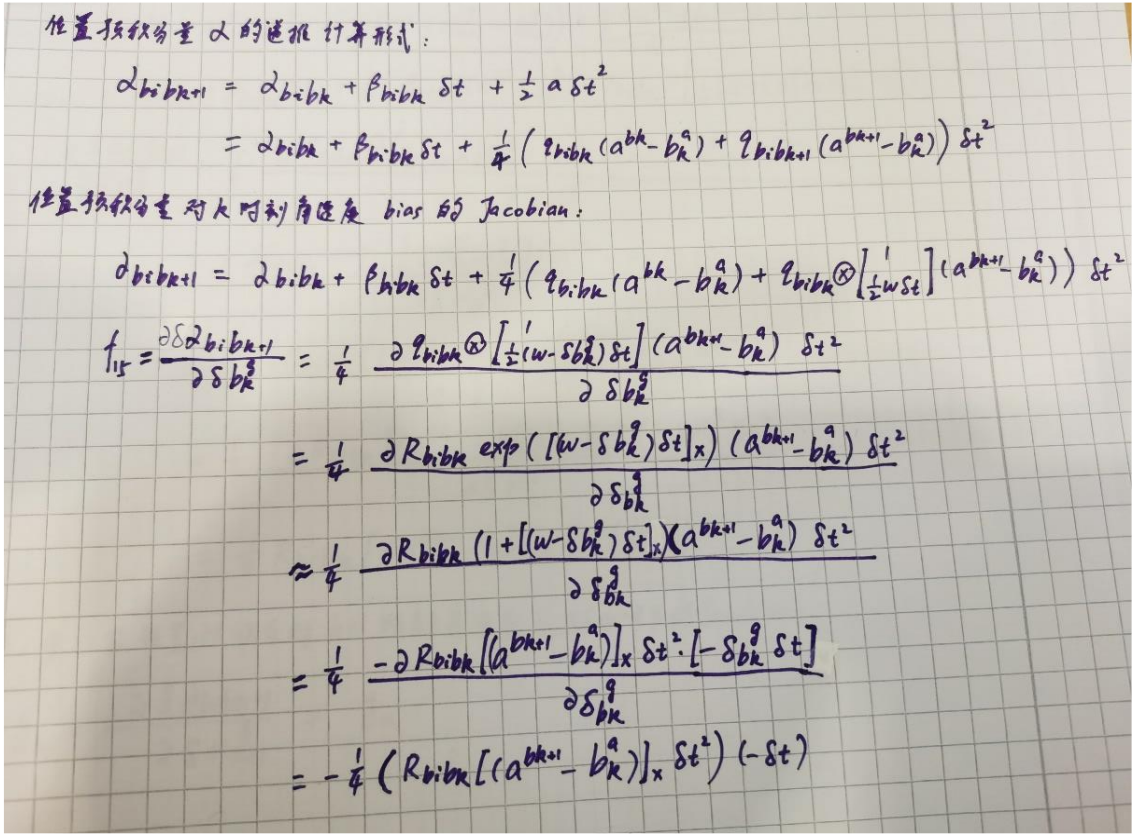
方法二



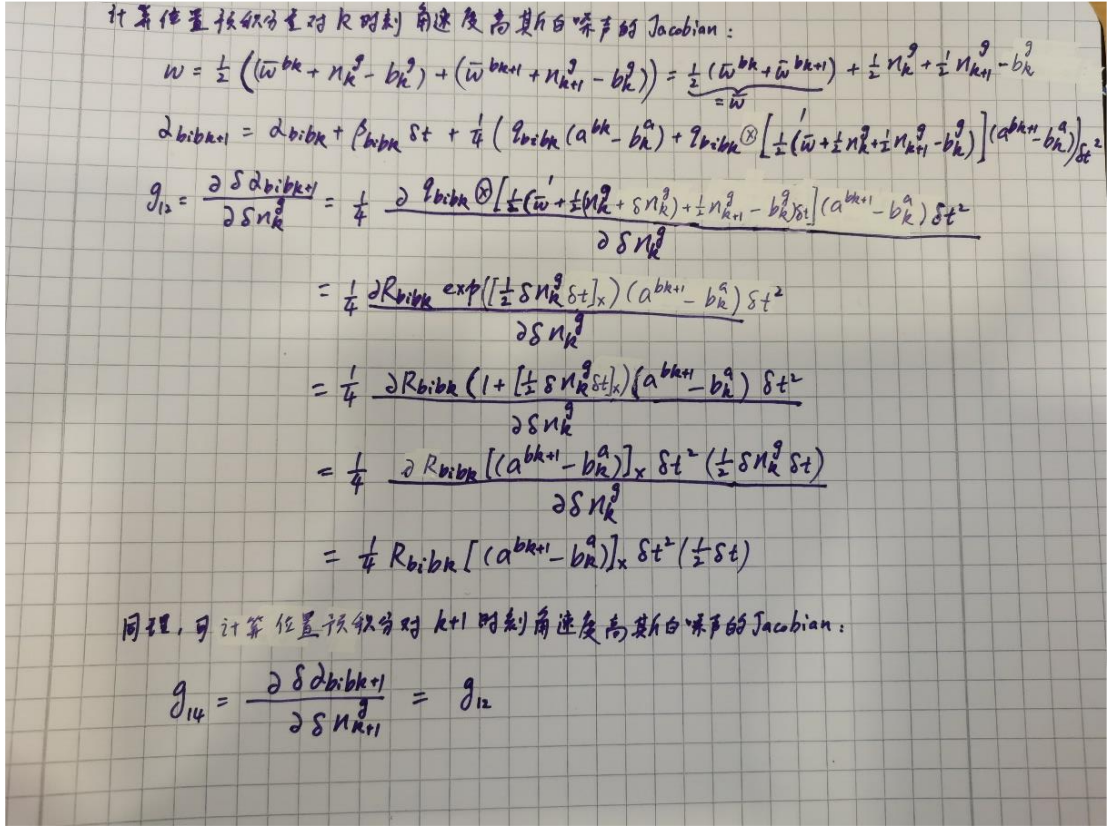
方法三



2. 公式推导雅可比F, G中的两项



2. 公式推导雅可比F, G中的两项



计算位置积分项对 k 时刻角速度高斯白噪声的 Jacobian：

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

$$\alpha_{bibk+1} = \alpha_{bibk} + \beta_{bibk}\delta t + \frac{1}{4}\left(q_{bibk}(a^{bk} - b_k^a) + q_{bibk} \otimes \left[\frac{1}{2}(\bar{w} + \frac{1}{2}n_k^g + \frac{1}{2}n_{k+1}^g - b_k^g)\right](a^{bk+1} - b_k^a)\right)\delta t^2$$

$$g_{12} = \frac{\partial \delta\alpha_{bibk+1}}{\partial \delta n_k^g} = \frac{1}{4}\frac{\partial\, q_{bibk} \otimes \left[\frac{1}{2}(\bar{w} + \frac{1}{2}(n_k^g + \delta n_k^g) + \frac{1}{2}n_{k+1}^g - b_k^g)\delta t\right](a^{bk+1} - b_k^a)\delta t^2}{\partial \delta n_k^g}$$

$$= \frac{1}{4}\frac{\partial R_{bibk}\exp([\frac{1}{2}\delta n_k^g \delta t]_\times)(a^{bk+1} - b_k^a)\delta t^2}{\partial \delta n_k^g}$$

$$= \frac{1}{4}\frac{\partial R_{bibk}(1 + [\frac{1}{2}\delta n_k^g \delta t]_\times)(a^{bk+1} - b_k^a)\delta t^2}{\partial \delta n_k^g}$$

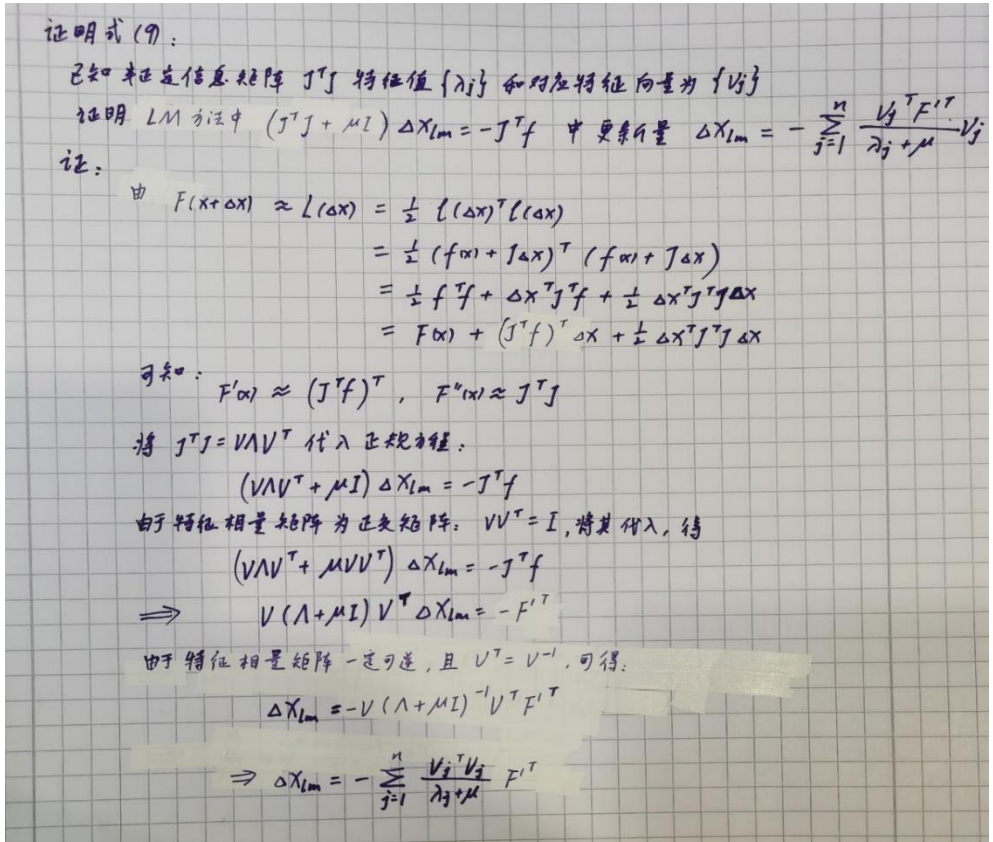
$$= \frac{1}{4}\frac{\partial R_{bibk}[(a^{bk+1} - b_k^a)]_\times \delta t^2 (\frac{1}{2}\delta n_k^g \delta t)}{\partial \delta n_k^g}$$

$$= \frac{1}{4} R_{bibk}[(a^{bk+1} - b_k^a)]_\times \delta t^2 (\frac{1}{2}\delta t)$$

同理，可计算位置积分对 k+1 时刻角速度高斯白噪声的 Jacobian：

$$g_{14} = \frac{\partial \delta\alpha_{bibk+1}}{\partial \delta n_{k+1}^g} = g_{12}$$

3. 证明式(9)



证明式(9)：

已知单位信息矩阵 J^TJ 特征值 $\{\lambda_j\}$ 和对应特征向量为 $\{v_j\}$

证明 LM 方法中 $(J^TJ+\mu I)\Delta x_{lm} = -J^Tf$ 中更新量 $\Delta x_{lm} = -\sum_{j=1}^{n} \frac{v_j^T F'^T}{\lambda_j+\mu} v_j$

证：

由 $F(x+\Delta x) \approx L(\Delta x) = \frac{1}{2}(\Delta x)^T(\Delta x)$

$= \frac{1}{2}(f(x)+J\Delta x)^T(f(x)+J\Delta x)$

$= \frac{1}{2}f^Tf + \Delta x^TJ^Tf + \frac{1}{2}\Delta x^TJ^TJ\Delta x$

$= F(x) + (J^Tf)^T\Delta x + \frac{1}{2}\Delta x^TJ^TJ\Delta x$

可知：$F'(x) \approx (J^Tf)^T$，$F''(x) \approx J^TJ$

将 $J^TJ = V\Lambda V^T$ 代入正规方程：

$(V\Lambda V^T+\mu I)\Delta x_{lm} = -J^Tf$

由于特征相量矩阵为正交矩阵：$VV^T = I$，将其代入，得

$(V\Lambda V^T+\mu VV^T)\Delta x_{lm} = -J^Tf$

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

由于特征相量矩阵一定可逆，且 $V^T = V^{-1}$，可得：

$\Delta x_{lm} = -V(\Lambda+\mu I)^{-1}V^TF'^T$

$\Rightarrow \Delta x_{lm} = -\sum_{j=1}^{n} \frac{v_j^Tv_j}{\lambda_j+\mu} F'^T$

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening