深蓝学院 VIO 课程大作业

温焕宇

2019.8.26

1 更优的优化策略

1.1 a. 选用更优的 LM 策略，使得 VINS-sys-code 在 MH-05 数据集上收敛速度更快或精度更高

【使用电脑配置：i3-8100,16G】

The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems 论文中第一种更新 Lambda 方式，但本文稍作修改，λ 初值依然是海森矩阵对角线元素最大值，将 λ 更新策略改为第一种。其中根据经验设置 $L_{\uparrow} \approx 11, L_{\downarrow} \approx 9$

主要算法如下：

$$
\begin{aligned}
&\lambda_0 = \lambda_o \max[diag[\mathbf{H}]]; \lambda_o = 10^{-5} \\
&\rho = \frac{\left(F(x) - F(x + \mathbf{h}_{lm})\right)}{\mathbf{h}_{lm}^T (F(x) \max[diag[H]] * \mathbf{h}_{lm} + b)} \\
&\textbf{if} \quad \rho > 0 \\
&\quad \lambda = \max[\lambda / L_{\downarrow}, 10^{-7}] \\
&\textbf{else} \\
&\quad \lambda = \min[\lambda / L_{\uparrow}, 10^{7}]
\end{aligned}
\tag{1}
$$

修改代码如下：

```cpp
double maxDiagonal = 0;
ulong size = Hessian_.cols();
assert(Hessian_.rows() == Hessian_.cols() && "Hessian is not square");
for (ulong i = 0; i < size; ++i) {
    maxDiagonal = std::max(fabs(Hessian_(i, i)), maxDiagonal); //Hessian矩阵对角元素最大值
}
scale = delta_x_.transpose() * (currentLambda_ * maxDiagonal * delta_x_ + b_);
scale += 1e-3;
double rho = (currentChi_ - tempChi) / scale;
if(rho > 0 && isfinite(tempChi))
{
    double scaleFactor = (std::max)(currentLambda_/9, 10e-7);
    currentLambda_ = scaleFactor;
    currentChi_ = tempChi;
    return true;
} else{
    double scaleFactor = (std::min)(currentLambda_/11, 10e+7);
    currentLambda_ = scaleFactor;
    return false;
}
```

使用 evo 工具评估精度，并且以 txt 文档记录每次迭代 Solve 时间和 MakeHessian 时间，然后计算平均值，结果如下：

name	max/m	mean/m	min/m	rmse/m	轨迹总长/m	SolveTime/s	MakeHessian/s
原始 LM 策略	0.483173	0.225505	0.232007	0.2423021	98.316	140.3868	94.1249
更新 LM 策略	**0.435452**	**0.225190**	**0.229332**	**0.239122**	98.260	**139.5707**	**91.9516**

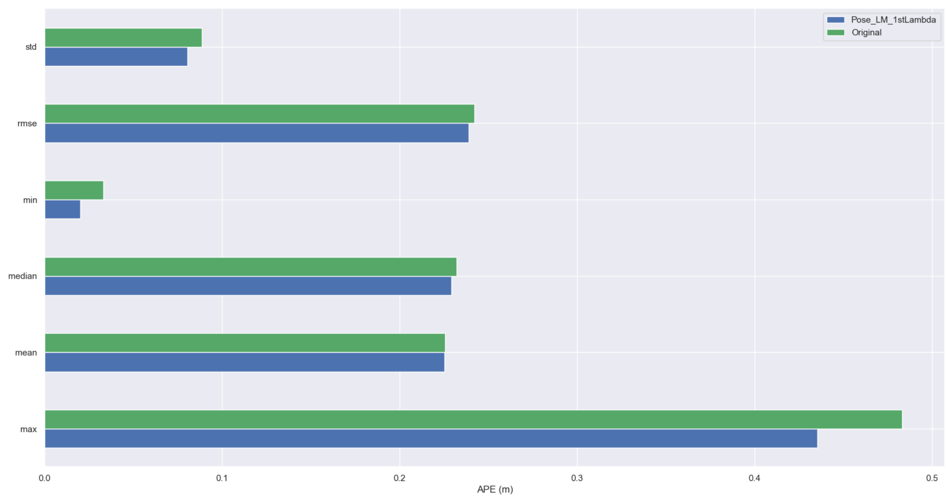

图 1: 重新设计后的更新策略与原始更新策略对比，蓝色表示 our，绿色表示原始策略

结果分析：在更新 LM 策略之后，精度提升 0.0032m。收敛速度提升 1-3s。

1.2 b. 实现 dog-leg 算法替换 LM 算法，并测试替换后的 VINS-sys-code 在 MH-05 的精度

正如 LM 方法，DogLeg 也是联合高斯牛顿法和最速下降法，但是 DogLeg 是利用信赖域半径来控制。LM 和 Dogleg 都是基于信赖域算法，其一个重要特点是其鲁棒性，而且信赖域子问题一定有解[1]。

本文采用的 DogLeg 算法[2]如下：给定 $f:\mathbf{R}^n \mapsto \mathbf{R}^m$。当前 x 及高斯牛顿步 $\mathbf{h}_{gn}$ 由下式写出：

$$\mathbf{J}(x)\mathbf{h} \simeq -\mathbf{f}(x) \tag{2}$$

对两边同时乘以 $\mathbf{J}^T$ 得：

$$\left(\mathbf{J}(x)^T\mathbf{J}(x)\right)\mathbf{h}_{gn} = -\mathbf{J}(x)^T\mathbf{f}(x) \tag{3}$$

上式也可以简写成：

$$\mathbf{H}\mathbf{h}_{gn} = \mathbf{b} \tag{4}$$

其中 $\mathbf{H} = \mathbf{J}(x)^T\mathbf{J}(x)$，$\mathbf{b} = -\mathbf{J}(x)^T\mathbf{f}(x)$。

最速下降方向的解由下式给出：

$$\alpha = \frac{\|\mathbf{b}\|^2}{\|\mathbf{b}^T\mathbf{H}\mathbf{b}\|} \tag{5}$$

$$\mathbf{A} = \mathbf{h}_{sd} = \alpha\mathbf{b} \tag{6}$$

[1] 数值最优化算法与理论

[2] 《Meehods For Non-Linear Lesast Squares Problems》及 g2o 库源码

高斯牛顿方向的解由下式：

$$\mathbf{B} = \mathbf{h}_{gn} = \mathbf{H}^{-1}\mathbf{b} \tag{7}$$

Dogleg 策略是根据信赖域半径选择步长，如图 1(图中 a、b 与上式中 A、B 对应，)。

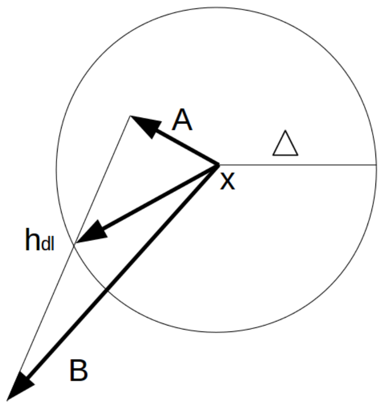

图 2: 信赖域和 DogLeg 步

选择步长 $\mathbf{h}_{dl}$ 策略如下：

$$\begin{aligned} \textbf{if} \quad & \|\mathbf{h}_{gn}\| \leqslant \triangle \\ & \mathbf{h}_{dl} := \mathbf{B} \\ \textbf{elseif} \quad & \|\mathbf{a}\| \geqslant \triangle \\ & \mathbf{h}_{dl} := (\triangle/\|\mathbf{A}\|)\mathbf{A} \\ \textbf{else} \quad & \\ & \mathbf{h}_{dl} := \mathbf{A} + \beta(\mathbf{B} - \mathbf{A}) \end{aligned} \tag{8}$$

其中 △ 为信赖域半径，β 定义如下：
$c = \mathbf{A}^T(\mathbf{B} - \mathbf{A})$

$$\begin{aligned} \textbf{if} \quad & \mathbf{c} \leqslant 0 \\ & \beta := \Big(-c + \sqrt{c^2 + \|\mathbf{B} - \mathbf{A}\|^2(\triangle^2 - \|\mathbf{A}\|^2)} \Big) / \|\mathbf{B} - \mathbf{A}\|^2 \\ \textbf{else} \quad & \\ & \beta := (\triangle^2 - \|\mathbf{A}\|^2) / \Big(c + \sqrt{c + \|\mathbf{B} - \mathbf{A}\|^2(\triangle^2 - \|\mathbf{A}\|^2)} \Big) \end{aligned} \tag{9}$$

计算下降比 ρ 及更新信赖域半径 △ 策略如下：

$$\rho = \frac{F(x) - F(x + h_{dl})}{L(0) - L(h_{dl})} \tag{10}$$

其中

$$L(0) - L(h_{dl}) = \begin{cases} F(\mathbf{x}) & \text{if} \quad \mathbf{h}_{dl} = \mathbf{h}_{gn} \\ \frac{\triangle(2\|\alpha\mathbf{b}\| - \triangle)}{2\alpha} & \text{if} \quad \mathbf{h}_{dl} = \frac{-\triangle}{\|\mathbf{b}\|}\mathbf{b} \\ \frac{1}{2}\alpha(1-\beta)^2\|\mathbf{b}\|^2 + \beta(2-\beta)F(\mathbf{x}) & \text{otherwise} \end{cases}$$

ρ **更新策略：**

$$
\begin{aligned}
\textbf{if} \quad & \rho > 0.75 \\
& \triangle := max\{\triangle, 3 * \|\mathbf{h}_{dl}\|\} \\
\textbf{else} \quad & \rho < 0.25 \\
& \mathbf{h}_{dl} := \triangle / 4
\end{aligned}
\tag{11}
$$

总体算法参考以下流程（具体参数设置以代码实际测试结果为准）：

Algorithm 3.21. Dog Leg Method

begin
  $k := 0; \quad \mathbf{x} := \mathbf{x}_0; \quad \Delta := \Delta_0; \quad \mathbf{g} := \mathbf{J}(\mathbf{x})^\top \mathbf{f}(\mathbf{x})$ $\{1^\circ\}$
  found $:= (\|\mathbf{f}(\mathbf{x})\|_\infty \le \varepsilon_3)$ **or** $(\|\mathbf{g}\|_\infty \le \varepsilon_1)$ $\{2^\circ\}$
  while (**not** *found*) **and** $(k < k_{\max})$
    $k := k+1$; Compute α by (3.19)
    $\mathbf{h}_{\mathrm{sd}} := -\alpha \mathbf{g}$; Solve $\mathbf{J}(\mathbf{x})\mathbf{h}_{\mathrm{gn}} \simeq -\mathbf{f}(\mathbf{x})$ $\{3^\circ\}$
    Compute $\mathbf{h}_{\mathrm{dl}}$ by (3.20)
    if $\|\mathbf{h}_{\mathrm{dl}}\| \le \varepsilon_2(\|\mathbf{x}\| + \varepsilon_2)$
      found := **true**
    else
      $\mathbf{x}_{\mathrm{new}} := \mathbf{x} + \mathbf{h}_{\mathrm{dl}}$
      $\varrho := (F(\mathbf{x}) - F(\mathbf{x}_{\mathrm{new}}))/(L(\mathbf{0}) - L(\mathbf{h}_{\mathrm{dl}}))$ $\{4^\circ\}$
      if $\varrho > 0$
        $\mathbf{x} := \mathbf{x}_{\mathrm{new}}; \quad \mathbf{g} := \mathbf{J}(\mathbf{x})^\top \mathbf{f}(\mathbf{x})$
        found $:= (\|\mathbf{f}(\mathbf{x})\|_\infty \le \varepsilon_3)$ **or** $(\|\mathbf{g}\|_\infty \le \varepsilon_1)$
      if $\varrho > 0.75$ $\{5^\circ\}$
        $\Delta := \max\{\Delta, 3{*}\|\mathbf{h}_{\mathrm{dl}}\|\}$
      elseif $\varrho < 0.25$
        $\Delta := \Delta/2$; *found* $:= (\Delta \le \varepsilon_2(\|\mathbf{x}\| + \varepsilon_2))$ $\{6^\circ\}$
end

【注】主要代码参考附录 A。

本次对比三种方法结果（原始 LM，改进后 LM 及本文修改的 DogLeg），使用 evo 分析绝对轨迹误差 APE 结果及所需处理平均时间数据如下：

name	max/m	mean/m	min/m	rmse/m	轨迹总长/m	SolveTime/s	MakeHessian/s
原始 LM	0.483173	0.225505	0.232007	0.242302	98.316	140.3868	94.1249
改进 LM	**0.435452**	0.225190	**0.229332**	0.239122	98.260	139.5707	91.9516
DogLeg	0.442087	**0.220603**	0.234866	**0.235843**	98.181	**99.6962**	**71.9366**

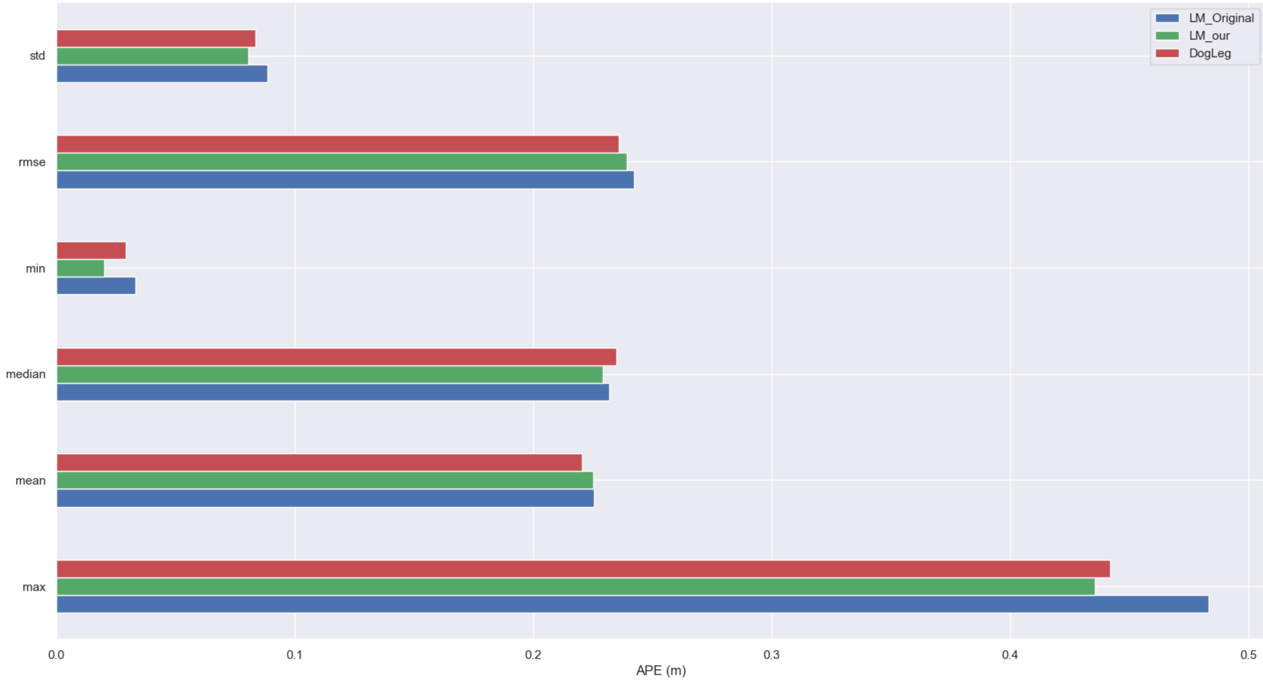

图 3: 三种结果对比: 红色表 DogLeg、绿色表改进后的 LM、蓝色表示原始 LM

结果分析：DogLeg 相对原始 LM 的 RMSE 提升 0.007m, 比改进后的提升 0.002m，稍有提升。但是 DogLeg 速度相对于 LM 提升 28% 左右

2 更快的 MakeHessian 矩阵

2.0.1 本文使用 OpenMP 加速

OpenMP 是一种用于共享内存并行系统的多线程程序设计方案，支持的编程语言包括 C、C++ 和 Fortran。OpenMP 提供了对并行算法的高层抽象描述，特别适合在多核 CPU 机器上的并行程序设计。编译器根据程序中添加的 pragma 指令，自动将程序并行处理，使用 OpenMP 降低了并行编程的难度和复杂度。当编译器不支持 OpenMP 时，程序会退化成普通（串行）程序。程序中已有的 OpenMP 指令不会影响程序的正常编译运行。

openMP 指令以 #pragm omp 开始，后边跟具体功能指令，格式如：#pragma omp 指令 [子句 [, 子句]...]。[3]

本次使用电脑配置：i3-8100,16G，四核四线程。

经过实际测试代码测试，指定 12 线程时间最短：

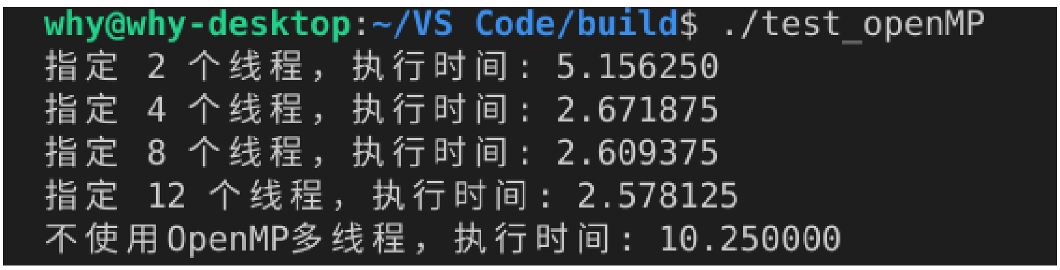

图 4: 线程数所耗时间对比

[3] 详细参考 https://blog.csdn.net/dcrmg/article/details/53862448

故修改代码如附录 B。

本次运行结果分析，以 txt 文档记录每次迭代 Solve 时间和 MakeHessian 时间，然后计算平均值进行结果对比如下：

name	RMS	SolveTime/s	MakeHessian/s	improve%
LM	0.239123	140.3868	94.1249	
LM-openMP	0.239123	132.2827	87.2298	7%
DogLeg	0.235843	99.6962	71.9366	
DogLeg-openMP	0.235843	**94.2062**	**67.9905**	6%

结果分析：针对 LM 和 DogLeg 两种方法加速提高约为 6% - 7%, 有较小幅度提升。精度基本没有任何变化。

3 总结

本次作业，更加深入的了解了 LM 优化方式，对于不同的 λ 更新策略具有不同的效果，参考[4]。更新 Dogleg 算法后相比于 LM 具有更高的精度和迭代速度，得根据具体代码仔细调参。最后了解了多线程加速 openMP，在保证精度不变的基础上计算速度得到一定程度提升。

[4] The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems

4 附录 A: 修改 DogLeg 算法

主要算法部分：

```
bool Problem::DLSolve(int iterations)
{
    // 1、设置参数： 初始值，信赖域上界，信赖域半径，\mu
    // 2、寻找最优解： 首先确定方向，再确定步长
    if (edges_.size() == 0 || verticies_.size() == 0) {
        std::cerr << "\nCannot solve problem without edges or verticies" << std::endl;
        return false;
    }
    TicToc t_solve;
    // 统计优化变量的维数，为构建 H 矩阵做准备
    SetOrdering();
    // 遍历edge，构建 H 矩阵。里面有delta_x_初值
    MakeHessian();

    // CumputeChiInitAndLambdaInit();
    ComputeLambdaInitLM();
    radius_ = 1e4;

    bool found = false;
    int iter = 0;
    const int numIterationsMax = 10;
    double last_chi_ = 1e20;
    while (!found && (iter < numIterationsMax)) {
        std::cout << "iter: " << iter << " , chi= " << currentChi_ << " , radius= " << radius_ << std::endl
;
        bool oneStepSuccess = false;
        int false_cnt = 0;
        while (!oneStepSuccess && false_cnt < 10)  // 不断尝试 Lambda, 直到成功迭代一步
        {
            // 计算alpha 和 h_gn
            alpha_ = b_.squaredNorm() / (Hessian_ * b_).dot(b_);
            h_sd_ = alpha_ * b_; // 相当于论文中a
            h_gn_ = Hessian_.ldlt().solve(b_);

            // 计算h_dl 步长
            if (h_gn_.norm() <= radius_)
                h_dl_ = h_gn_;
            else if ( h_sd_.norm() >= radius_ )
                h_dl_ = ( radius_ / h_sd_.norm() ) * h_sd_;
            else
            {
                // 计算beta用于更新步长
                VecX a = h_sd_;
                VecX b = h_gn_;
                double c = h_sd_.dot( b - a );

                if (c <= 0)
                    beta_ = ( -c + sqrt(c*c + (b-a).squaredNorm() * (radius_*radius_ - a.squaredNorm())) )
 / (b - a).squaredNorm();
                else
                    beta_ = (radius_*radius_ - a.squaredNorm()) / (c + sqrt(c*c + (b-a).squaredNorm() * (
radius_*radius_ - a.squaredNorm())));
                assert(beta_ > 0.0 && beta_ < 1.0 && "Error while computing beta");
                h_dl_ = h_sd_ + beta_ * (h_gn_ - h_sd_);
                assert(h_dl_.norm() < radius_ + 1e-5 && "Computed step does not corredpond to the trust
region");
            }
            delta_x_ = h_dl_; // 步长即delta_x_
```

```
            UpdateStates();
            oneStepSuccess = IsGoodStepDL();
            if(oneStepSuccess)
            {
                MakeHessian();
                false_cnt = 0;
            }
            else
            {
                false_cnt++;
                RollbackStates();
            }
        }
        iter++;
        if(last_chi_ - currentChi_ < 1e-5 || b_.norm() < 1e-5)
        {
            std::cout << "sqrt(currentChi_) <= stopThresholdLM_" << std::endl;
            found = true;
        }
        last_chi_ = currentChi_;
    }
    std::cout << "problem solve cost: " << t_solve.toc() << " ms" << std::endl;
    std::cout << "   makeHessian cost: " << t_hessian_cost_ << " ms" << std::endl;
    t_hessian_cost_ = 0.;
    return true;
}
```

判断是否好的一次迭代

```
bool Problem::IsGoodStepDL()
{
    double rho;
    double tempChi = 0.0;
    for (auto edge: edges_) {
        edge.second->ComputeResidual();
        tempChi += edge.second->RobustChi2();
    }
    if (err_prior_.size() > 0)
        tempChi += err_prior_.norm();
    tempChi *= 0.5;          // 1/2 * err^2

    // 计算更新rho
    double scale = 0;
    if(h_dl_ == h_gn_)
        scale = currentChi_;
    else if(h_dl_ == radius_ * b_ / b_.squaredNorm())
        scale = radius_ * (2 * (alpha_ * b_).norm() - radius_) / (2 * alpha_);
    else
        scale = 0.5 * alpha_ * pow((1 - beta_), 2) * b_.squaredNorm() + beta_ * (2 - beta_) * currentChi_;
    rho = (currentChi_ - tempChi) / scale;
  // 更新信赖域半径
    if(rho > 0.75)
    {
        radius_ = std::max(radius_, 3*h_dl_.norm());
    }
    else if(rho < 0.25)
    {
        // radius_ = radius_ / 2;
        radius_ = std::max(radius_ * 0.5, 1e-7);
    }

```

```
    if(rho > 0 && isfinite(tempChi))
    {
        // delta_x_ = delta_x_ + h_dl_;
        currentChi_ = tempChi;
        return true;
    }
    else
    {
        return false;
    }
}
```

5 附录 B: 修改 openMP 加速

```
#ifdef USE_OPENMP
#pragma omp declare reduction (+: VecX: omp_out=omp_out+omp_in)\
     initializer(omp_priv=VecX::Zero(omp_orig.size()))
#pragma omp declare reduction (+: MatXX: omp_out=omp_out+omp_in)\
     initializer(omp_priv=MatXX::Zero(omp_orig.rows(), omp_orig.cols()))
#endif
void Problem::MakeHessianNormal(){
    TicToc t_h;
    // 直接构造大的 H 矩阵
    ulong size = ordering_generic_;
    MatXX H(MatXX::Zero(size, size));
    VecX b(VecX::Zero(size));

    vector< shared_ptr<myslam::backend::Edge> > vec_edge_;
    int edgesSize = edges_.size();
    vec_edge_.reserve(edges_.size());
    for (auto edge: edges_) {
        vec_edge_.push_back(edge.second);
    }

    #ifdef USE_OPENMP
    omp_set_num_threads(12);
    Eigen::setNbThreads(1);
    #pragma omp parallel for reduction(+: H) reduction(+: b)
    #endif

    // for (auto &edge: edges_) {
    for (int i = 0; i < edgesSize; i++) {

        auto edge = vec_edge_[i];
        // edge.second->ComputeResidual();
        // edge.second->ComputeJacobians();
        edge->ComputeResidual();
        edge->ComputeJacobians();

        // TODO:: robust cost
        // auto jacobians = edge.second->Jacobians();
        // auto verticies = edge.second->Verticies();
        auto jacobians = edge->Jacobians();
        auto verticies = edge->Verticies();
        // assert(jacobians.size() == verticies.size());

        for (size_t i = 0; i < verticies.size(); ++i) {
            auto v_i = verticies[i];
```

```
            if (v_i->IsFixed()) continue;    // Hessian 里不需要添加它的信息，也就是它的雅克比为 0

            auto jacobian_i = jacobians[i];
            ulong index_i = v_i->OrderingId();
            ulong dim_i = v_i->LocalDimension();

            // 鲁棒核函数会修改残差和信息矩阵，如果没有设置 robust cost function，就会返回原来的
            double drho;
            MatXX robustInfo(edge->Information().rows(),edge->Information().cols());
            edge->RobustInfo(drho,robustInfo);

            MatXX JtW = jacobian_i.transpose() * robustInfo;
            for (size_t j = i; j < verticies.size(); ++j) {
                auto v_j = verticies[j];

                if (v_j->IsFixed()) continue;

                auto jacobian_j = jacobians[j];
                ulong index_j = v_j->OrderingId();
                ulong dim_j = v_j->LocalDimension();

                assert(v_j->OrderingId() != -1);
                MatXX hessian = JtW * jacobian_j;

                // 所有的信息矩阵叠加起来
                H.block(index_i, index_j, dim_i, dim_j).noalias() += hessian;
                if (j != i) {
                    // 对称的下三角
                    H.block(index_j, index_i, dim_j, dim_i).noalias() += hessian.transpose();

                }
            }
            b.segment(index_i, dim_i).noalias() -= drho * jacobian_i.transpose()* edge->Information() * edge->Residual();
        }

    }
    Hessian_ = H;
    b_ = b;
    t_hessian_cost_ += t_h.toc();

    if(H_prior_.rows() > 0)
    {
        MatXX H_prior_tmp = H_prior_;
        VecX b_prior_tmp = b_prior_;

        /// 遍历所有 POSE 顶点，然后设置相应的先验维度为 0 .  fix 外参数, SET PRIOR TO ZERO
        /// landmark 没有先验
        for (auto vertex: verticies_) {
            if (IsPoseVertex(vertex.second) && vertex.second->IsFixed() ) {
                int idx = vertex.second->OrderingId();
                int dim = vertex.second->LocalDimension();
                H_prior_tmp.block(idx,0, dim, H_prior_tmp.cols()).setZero();
                H_prior_tmp.block(0,idx, H_prior_tmp.rows(), dim).setZero();
                b_prior_tmp.segment(idx,dim).setZero();
//                std::cout << " fixed prior, set the Hprior and bprior part to zero, idx: "<<idx <<" dim: "<<dim<<std::endl;
            }
        }
        Hessian_.topLeftCorner(ordering_poses_, ordering_poses_) += H_prior_tmp;
        b_.head(ordering_poses_) += b_prior_tmp;
    }
```

```
    delta_x_ = VecX::Zero(size);   // initial delta_x = 0_n;
}
```