从零开始手写VIO 第五课作业

边城量子 2019.07.20

基础题

1. 完成单目 $BundleAdjustment$ 求解器 $problem.cc$ 中的部分代码.

- ① 完成 $Problem :: MakeHessian()$ 中信息矩阵 H 的计算
- ② 完成 $Problem :: SolveLinearSystem()$ 中 $SLAM$ 问题的求解

2. 完成滑动窗口算法测试函数

- ✍ 完成 $Problem :: TestMarginalize()$ 中的代码, 并通过测试.

以上两个问题的回答:

- 1. 修改 `problem.cc` 中的 `Problem::MakeHessian()` 函数:
    - 新增代码片段如下:

```
// TODO:: home work. 完成 H index 的填写.
H.block(index_i, index_j, dim_i, dim_j).noalias() += hessian;   //
放到矩阵的哪一维度, 维度是多少, 类似下面的b.segment
if (j != i) {
    // 对称的下三角
    // TODO:: home work. 完成 H index 的填写.
    H.block(index_j, index_i, dim_j, dim_i).noalias() +=
hessian.transpose();
}
```

    - 完整的 `Problem::MakeHessian()` 函数如下:

```
void Problem::MakeHessian() {
    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));

    for (auto &edge: edges_) {

        // 遍历所有边
        edge.second->ComputeResidual();
        edge.second->ComputeJacobians();

        auto jacobians = edge.second->Jacobians();
        auto verticies = edge.second->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();   // 这个顶点是几维的

            MatXX JtW = jacobian_i.transpose() * edge.second->Information();
            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

                // TODO:: home work. 完成 H index 的填写.
                H.block(index_i, index_j, dim_i, dim_j).noalias() += hessian;   // 放到矩阵的哪一维度，维度是多少，类似下面的b.segment
                if (j != i) {
                    // 对称的下三角
                    // TODO:: home work. 完成 H index 的填写.
                    H.block(index_j, index_i, dim_j, dim_i).noalias() += hessian.transpose();
                }
            }
            b.segment(index_i, dim_i).noalias() -= JtW * edge.second->Residual();
        }

    }

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


//    Eigen::JacobiSVD<Eigen::MatrixXd> svd(H, Eigen::ComputeThinU | Eigen::ComputeThinV);
//    std::cout << svd.singularValues() <<std::endl;

    if (err_prior_.rows() > 0) {
        b_prior_ -= H_prior_ * delta_x_.head(ordering_poses_);   // update the error_prior
    }
    Hessian_.topLeftCorner(ordering_poses_, ordering_poses_) += H_prior_;
    b_.head(ordering_poses_) += b_prior_;

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

}
```

- 2. 修改 `problem.cc` 中的 `Problem::SolveLinearSystem()` 函数:
  - 修改的后的 `SolveLinearSystem()` 代码如下:

```
/*
 * Solve Hx = b, we can use PCG iterative method or use sparse
Cholesky
 */
void Problem::SolveLinearSystem() {

    if (problemType_ == ProblemType::GENERIC_PROBLEM) {

        // 非 SLAM 问题直接求解
        // PCG solver
        MatXX H = Hessian_;
        for (ulong i = 0; i < Hessian_.cols(); ++i) {
            H(i, i) += currentLambda_;
        }
//          delta_x_ = PCGSolver(H, b_, H.rows() * 2);
        delta_x_ = Hessian_.inverse() * b_;

    }
    else {

        // SLAM 问题采用舒尔补的计算方式
        // step1: schur marginalization --> Hpp, bpp
        int reserve_size = ordering_poses_;
        int marg_size = ordering_landmarks_;

        // TODO:: home work. 完成矩阵块取值，Hmm，Hpm，Hmp，bpp，bmm (取
出对应维度)
        MatXX Hmm = Hessian_.block(reserve_size, reserve_size,
marg_size, marg_size);
        MatXX Hpm = Hessian_.block(0, reserve_size, reserve_size,
marg_size);
        MatXX Hmp = Hessian_.block(reserve_size, 0, marg_size,
reserve_size);
        VecX bpp = b_.segment(0, reserve_size);
        VecX bmm = b_.segment(reserve_size, marg_size);

        // Hmm 是对角线矩阵，它的求逆可以直接为对角线块分别求逆，如果是逆深
度，对角线块为1维的，则直接为对角线的倒数，这里可以加速
        MatXX Hmm_inv(MatXX::Zero(marg_size, marg_size));
        for (auto landmarkVertex : idx_landmark_vertices_) {
            int idx = landmarkVertex.second->OrderingId() -
reserve_size;
            int size = landmarkVertex.second->LocalDimension();
            Hmm_inv.block(idx, idx, size, size) = Hmm.block(idx,
idx, size, size).inverse();
        }

        // TODO:: home work. 完成舒尔补 Hpp, bpp 代码
        // 计算b_pp_schur和H_pp_schu 时都需要用到的中间变量 Hpm *
Hmm_inv
        MatXX Hpm_Hmm = Hpm * Hmm_inv;
```

```
        //  计算 Hpm * Hmm_inv * Hmp
        H_pp_schur_ = Hessian_.block(0,0,reserve_size,
reserve_size) - Hpm_Hmm * Hmp;
        //  计算 Hpm * Hmm_inv * bmm
        b_pp_schur_ = bpp - Hpm_Hmm * bmm;

        // step2: solve Hpp * delta_x = bpp
        VecX delta_x_pp(VecX::Zero(reserve_size));
        // PCG Solver
        for (ulong i = 0; i < ordering_poses_; ++i) {
            H_pp_schur_(i, i) += currentLambda_;
        }

        int n = H_pp_schur_.rows() * 2;                       // 迭
代次数
        delta_x_pp = PCGSolver(H_pp_schur_, b_pp_schur_, n);  // 哈
哈，小规模问题，搞 pcg 花里胡哨
        delta_x_.head(reserve_size) = delta_x_pp;
        //        std::cout << delta_x_pp.transpose() << std::endl;

        // TODO:: home work. step3: solve landmark
        VecX delta_x_ll(marg_size);
        // 注意: 此处和课件的公式(6)不同, bmm前没有负号, 是因为构造三角阵的
时候, 已经给b设定了负号,如下:
        //       b.segment(index_i, dim_i).noalias() -= JtW *
edge.second->Residual();
        delta_x_ll = Hmm_inv * ( bmm -  Hmp * delta_x_pp);
        delta_x_.tail(marg_size) = delta_x_ll;
    }
}
```

- 3. 修改 `problem.cc` 中的 `Problem::TestMarginalize()` 函数:
    - 修改的后的 `TestMarginalize()` 代码如下:

```
void Problem::TestMarginalize() {

    // Add marg test
    int idx = 1;            // marg 中间那个变量
    int dim = 1;            // marg 变量的维度
    int reserve_size = 3;   // 总共变量的维度
    double delta1 = 0.1 * 0.1;
    double delta2 = 0.2 * 0.2;
    double delta3 = 0.3 * 0.3;

    int cols = 3;
    MatXX H_marg(MatXX::Zero(cols, cols));
    H_marg << 1./delta1, -1./delta1, 0,
            -1./delta1, 1./delta1 + 1./delta2 + 1./delta3,
-1./delta3,
            0.,  -1./delta3, 1/delta3;
    std::cout << "---------- TEST Marg: before marg------------"<<
std::endl;
    std::cout << H_marg << std::endl;

    // TODO:: home work. 将变量移动到右下角
    /// 准备工作:  move the marg pose to the Hmm bottown right
```

```
    // 将 row i 移动矩阵最下面
    Eigen::MatrixXd temp_rows = H_marg.block(idx, 0, dim,
reserve_size);
    Eigen::MatrixXd temp_botRows = H_marg.block(idx + dim, 0,
reserve_size - idx - dim, reserve_size);
    H_marg.block(idx, 0, reserve_size - idx - dim, reserve_size) =
temp_botRows;
    H_marg.block(reserve_size - dim, 0, dim, reserve_size) =
temp_rows;

    // 将 col i 移动矩阵最右边
    Eigen::MatrixXd temp_cols = H_marg.block(0, idx, reserve_size,
dim);
    Eigen::MatrixXd temp_rightCols = H_marg.block(0, idx + dim,
reserve_size, reserve_size - idx - dim);
    H_marg.block(0, idx, reserve_size, reserve_size - idx - dim) =
temp_rightCols;
    H_marg.block(0, reserve_size - dim, reserve_size, dim) =
temp_cols;

    std::cout << "---------- TEST Marg: 将变量移动到右下角------------
"<< std::endl;
    std::cout<< H_marg <<std::endl;

    /// 开始 marg ： schur
    double eps = 1e-8;
    int m2 = dim;
    int n2 = reserve_size - dim;   // 剩余变量的维度
    Eigen::MatrixXd Amm = 0.5 * (H_marg.block(n2, n2, m2, m2) +
H_marg.block(n2, n2, m2, m2).transpose());

    Eigen::SelfAdjointEigenSolver<Eigen::MatrixXd> saes(Amm);
    Eigen::MatrixXd Amm_inv = saes.eigenvectors() *
Eigen::VectorXd(
            (saes.eigenvalues().array() >
eps).select(saes.eigenvalues().array().inverse(), 0)).asDiagonal()
*
                              saes.eigenvectors().transpose();

    // TODO:: home work. 完成舒尔补操作
    Eigen::MatrixXd Arm = H_marg.block(0,n2,n2,m2);
    Eigen::MatrixXd Amr = H_marg.block(n2,0,m2,n2);
    Eigen::MatrixXd Arr = H_marg.block(0,0,n2,n2);

    Eigen::MatrixXd tempB = Arm * Amm_inv;
    Eigen::MatrixXd H_prior = Arr - tempB * Amr;

    std::cout << "---------- TEST Marg: after marg------------"<<
std::endl;
    std::cout << H_prior << std::endl;
}
```

- 4.编译并执行 ./testMonoBA, 执行结果如下:

```
0 order: 0
1 order: 6
2 order: 12
```

```
 ordered_landmark_vertices_ size : 20
iter: 0 , chi= 5.35099 , Lambda= 0.00597396
iter: 1 , chi= 0.0289048 , Lambda= 0.00199132
iter: 2 , chi= 0.000109162 , Lambda= 0.000663774
problem solve cost: 1.07113 ms
  makeHessian cost: 0.557268 ms

Compare MonoBA results after opt...
after opt, point 0 : gt 0.220938 ,noise 0.227057 ,opt 0.220992
after opt, point 1 : gt 0.234336 ,noise 0.314411 ,opt 0.234854
after opt, point 2 : gt 0.142336 ,noise 0.129703 ,opt 0.142666
after opt, point 3 : gt 0.214315 ,noise 0.278486 ,opt 0.214502
after opt, point 4 : gt 0.130629 ,noise 0.130064 ,opt 0.130562
after opt, point 5 : gt 0.191377 ,noise 0.167501 ,opt 0.191892
after opt, point 6 : gt 0.166836 ,noise 0.165906 ,opt 0.167247
after opt, point 7 : gt 0.201627 ,noise 0.225581 ,opt 0.202172
after opt, point 8 : gt 0.167953 ,noise 0.155846 ,opt 0.168029
after opt, point 9 : gt 0.21891 ,noise 0.209697 ,opt 0.219314
after opt, point 10 : gt 0.205719 ,noise 0.14315 ,opt 0.205995
after opt, point 11 : gt 0.127916 ,noise 0.122109 ,opt 0.127908
after opt, point 12 : gt 0.167904 ,noise 0.143334 ,opt 0.168228
after opt, point 13 : gt 0.216712 ,noise 0.18526 ,opt 0.216866
after opt, point 14 : gt 0.180009 ,noise 0.184249 ,opt 0.180036
after opt, point 15 : gt 0.226935 ,noise 0.245716 ,opt 0.227491
after opt, point 16 : gt 0.157432 ,noise 0.176529 ,opt 0.157589
after opt, point 17 : gt 0.182452 ,noise 0.14729 ,opt 0.182444
after opt, point 18 : gt 0.155701 ,noise 0.182258 ,opt 0.155769
after opt, point 19 : gt 0.14646 ,noise 0.240649 ,opt 0.14677
------------ pose translation ----------------
translation after opt: 0 :-0.000477994   0.00115908  0.000366504 || gt: 0 0
0
translation after opt: 1 :-1.06959  4.00018 0.863877 || gt:  -1.0718
 4 0.866025
translation after opt: 2 :-4.00232  6.92678 0.867244 || gt:       -4
6.9282 0.866025
---------- TEST Marg: before marg------------
     100     -100        0
    -100  136.111 -11.1111
       0 -11.1111  11.1111
---------- TEST Marg: 将变量移动到右下角------------
     100        0     -100
       0  11.1111 -11.1111
    -100 -11.1111  136.111
---------- TEST Marg: after marg------------
 26.5306 -8.16327
-8.16327  10.2041
```

提升题

paper reading, 请总结论文: 优化过程中处理 H 自由度的不同操作方式. 总结内容包括: 具体处理方式, 实现效果, 结论.

```
Zichao Zhang, Guillermo Gallego, and Davide Scaramuzza. "On the comparison of
gauge freedom handling in optimization-based visual-inertial state estimation".
In: IEEE Rototics and Automation Letters 3.3 (2018)
```

回答

待完善...