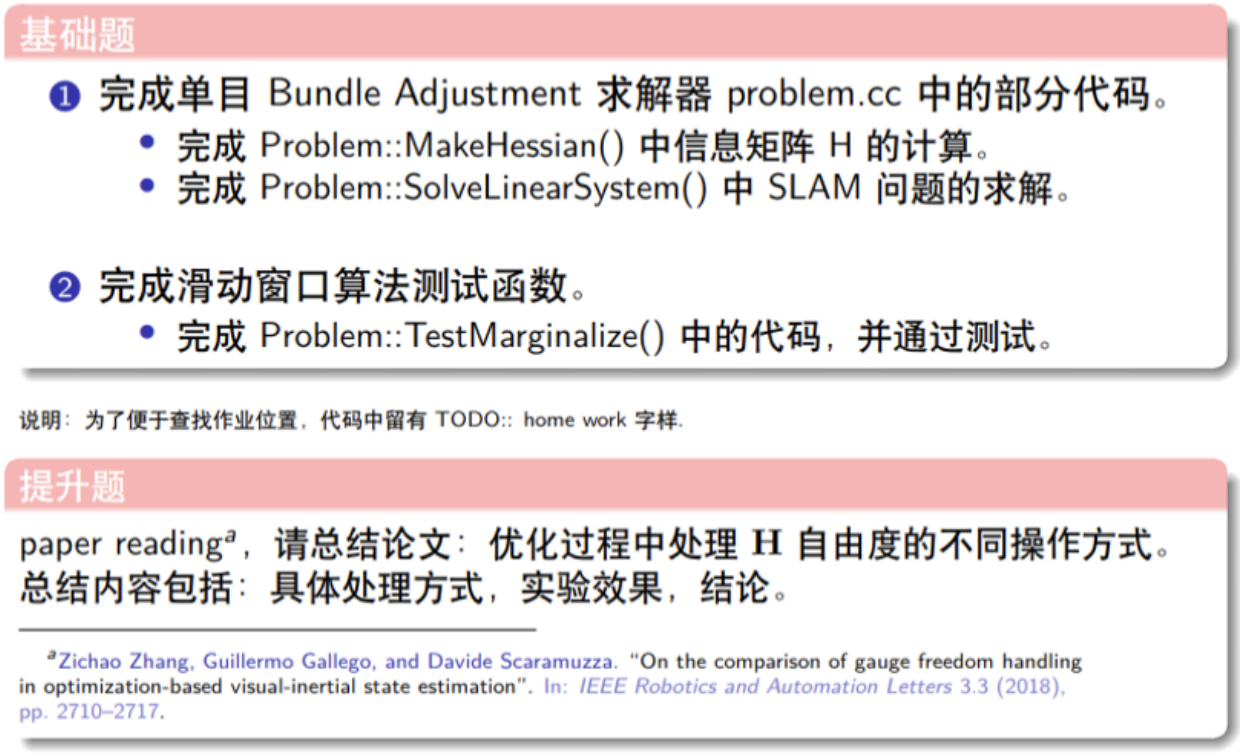
vio_hw4

Created by	Ⓢ Shijie Xie
Created time	@March 20, 2024 9:37 AM
Status	To Do
Tags	

第五章作业：



1. BA求解器
    a. MakeHessian()：根据排好序的状态量，通过两个loop，计算各个状态量之间的JTJ并叠加

```
    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;
    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. SolveLinearSystem()：通过舒尔补的方式构建上三角矩阵，先求解位姿的更新量，再求解landmark的更新量。因为landmard矩阵块是对角阵，所以可以加速嘛？

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

// step2: solve Hpp * delta_x = bpp
VecX delta_x_pp(VecX::Zero(reserve_size));
// PCG Solver
// add lambda
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);
// delta_x_ll = PCGSolver(Hmm, bmm - Hmp * delta_x_pp, 100);
delta_x_ll = Hmm_inv * (bmm - Hmp * delta_x_pp);
delta_x_.tail(marg_size) = delta_x_ll;
```

c. 结论：

MonoBA优化结果如下：

```
 ordered_landmark_vertices_ size : 20
iter: 0 , chi= 3.89166 , Lambda= 0.00350962
iter: 1 , chi= 0.0191522 , Lambda= 0.00116987
iter: 2 , chi= 0.000102713 , Lambda= 0.000389958
problem solve cost: 0.403336 ms
   makeHessian cost: 0.209792 ms

Compare MonoBA results after opt...
after opt, point 0 : gt 0.205093 ,noise 0.210355 ,opt 0.205036
after opt, point 1 : gt 0.129504 ,noise 0.150718 ,opt 0.129333
after opt, point 2 : gt 0.1261 ,noise 0.116083 ,opt 0.126211
after opt, point 3 : gt 0.235724 ,noise 0.315749 ,opt 0.236052
after opt, point 4 : gt 0.238076 ,noise 0.236205 ,opt 0.238639
after opt, point 5 : gt 0.135429 ,noise 0.12302 ,opt 0.135573
after opt, point 6 : gt 0.132224 ,noise 0.131639 ,opt 0.132361
after opt, point 7 : gt 0.151326 ,noise 0.16443 ,opt 0.151526
after opt, point 8 : gt 0.208375 ,noise 0.190058 ,opt 0.208622
after opt, point 9 : gt 0.13258 ,noise 0.129144 ,opt 0.132373
after opt, point 10 : gt 0.134346 ,noise 0.104514 ,opt 0.134149
after opt, point 11 : gt 0.165078 ,noise 0.155534 ,opt 0.165205
after opt, point 12 : gt 0.130437 ,noise 0.115108 ,opt 0.130476
after opt, point 13 : gt 0.24978 ,noise 0.208902 ,opt 0.250042
after opt, point 14 : gt 0.196972 ,noise 0.20206 ,opt 0.197066
after opt, point 15 : gt 0.139294 ,noise 0.14615 ,opt 0.139396
after opt, point 16 : gt 0.215204 ,noise 0.252551 ,opt 0.215621
after opt, point 17 : gt 0.180137 ,noise 0.145778 ,opt 0.180044
after opt, point 18 : gt 0.186037 ,noise 0.225254 ,opt 0.186076
after opt, point 19 : gt 0.142718 ,noise 0.230708 ,opt 0.142899
------------ pose translation ----------------
translation after opt: 0 : 0.00180639  0.00329685 8.27791e-05 || gt: 0 0 0
translation after opt: 1 :-1.07294  3.99476 0.865725 || gt:  -1.0718         4 0.866025
translation after opt: 2 :-4.00114  6.93014 0.865833 || gt:       -4   6.9282 0.866025
```

由于BA问题存在7个自由度的不确定性，导致初始位置，不在（0，0，0）处，因为优化器并不知道初始位姿是固定的，所以我们可以固定初始位姿的状态量，避免被优化。

d. 当固定前两个pose的位姿时，优化结果为：

```
Compare MonoBA results after opt...
after opt, point 0 : gt 0.205093 ,noise 0.210355 ,opt 0.205139
after opt, point 1 : gt 0.129504 ,noise 0.150718 ,opt 0.129391
after opt, point 2 : gt 0.1261 ,noise 0.116083 ,opt 0.126094
after opt, point 3 : gt 0.235724 ,noise 0.315749 ,opt 0.236038
after opt, point 4 : gt 0.238076 ,noise 0.236205 ,opt 0.238029
after opt, point 5 : gt 0.135429 ,noise 0.12302 ,opt 0.135585
after opt, point 6 : gt 0.132224 ,noise 0.131639 ,opt 0.132296
after opt, point 7 : gt 0.151326 ,noise 0.16443 ,opt 0.151594
after opt, point 8 : gt 0.208375 ,noise 0.190058 ,opt 0.208452
after opt, point 9 : gt 0.13258 ,noise 0.129144 ,opt 0.13245
after opt, point 10 : gt 0.134346 ,noise 0.104514 ,opt 0.134155
after opt, point 11 : gt 0.165078 ,noise 0.155534 ,opt 0.164893
after opt, point 12 : gt 0.130437 ,noise 0.115108 ,opt 0.130419
after opt, point 13 : gt 0.24978 ,noise 0.208902 ,opt 0.250086
after opt, point 14 : gt 0.196972 ,noise 0.20206 ,opt 0.196966
after opt, point 15 : gt 0.139294 ,noise 0.14615 ,opt 0.139453
after opt, point 16 : gt 0.215204 ,noise 0.252551 ,opt 0.215368
after opt, point 17 : gt 0.180137 ,noise 0.145778 ,opt 0.180019
after opt, point 18 : gt 0.186037 ,noise 0.225254 ,opt 0.185979
after opt, point 19 : gt 0.142718 ,noise 0.230708 ,opt 0.142759
------------ pose translation ----------------
translation after opt: 0 :0 0 0 || gt: 0 0 0
translation after opt: 1 : -1.0718        4 0.866025 || gt:  -1.0718         4 0.866025
translation after opt: 2 :-3.99834  6.92859  0.85604 || gt:       -4    6.9282 0.866025
```

可以看见第一帧的微小偏移不存在了。

2. 滑动窗口算法测试函数： 大致分为两个步骤，移动marginalization的对象，完成舒尔补的计算。

```cpp
// 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;
```

3. 总结处理H自由度的不同操作方式：

   Gauge fixation方法通过固定状态量，将对应的Jacobian设成0，使得优化问题有唯一解。

   Gauge prior方法通过给需要固定的状态量添加一个先验的方式，使得目标方程满秩，从而解决零空间的存在，Gauge fixation本质上和Gauge prior一直，Gauge fixation添加了一个无穷大的先验，而Gauge prior的先验需要调整，性能会随着prior而变化，但当prior足够大时，性能会随之稳定。当在prior较小的情况下，迭代次数和收敛时间都会比较大，这是因为优化工程中会在前两帧以提升prior作为代价来实现最小化重投影误差，所以需要更多的迭代次数和收敛时间。

   Free Gauge则是通过对hessian取伪逆的方式，添加了额外的约束（parameter updates with smallest norm），从而得到唯一的解。（这个方法具有通用性，不需要知道哪些状态量需要被约束）。

   通过accuracy和computational cost来比较不同方法的性能，三者的准确性基本一致。gauge prior方法需要选择合适的prior，否则算力消耗会更大。在合适的prior

下，gauge fixation和gauge prior的效率相同，free gauge会更快，因为优化距离最短。