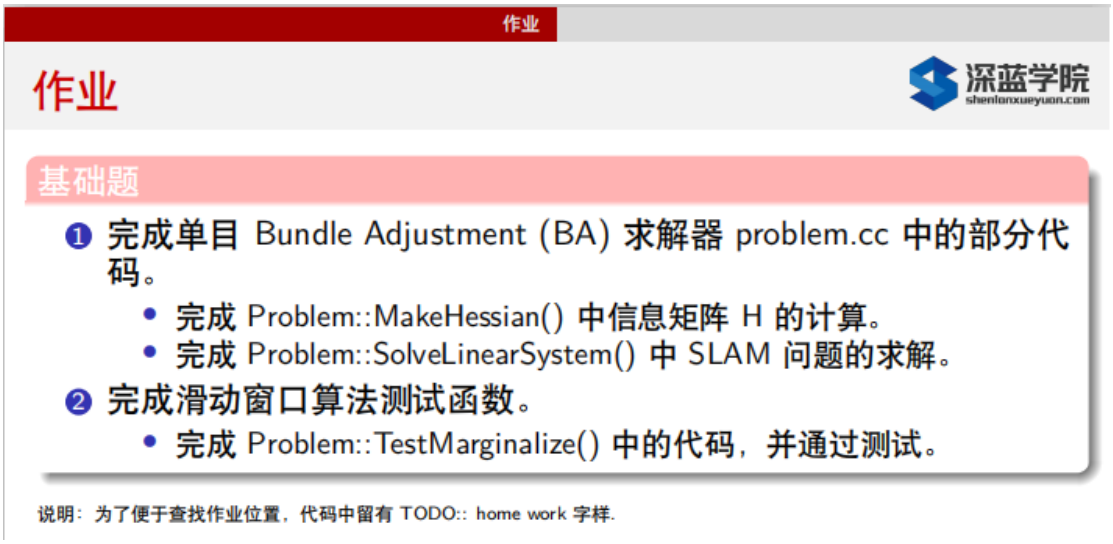


完成 MakeHessian()

```
    // TODO:: home work. 完成 H index 的填写.
    // noalias混淆问题, Eigen库的 matA=(matB*matA).eval()可以解决
    H.block(index_i, index_j, dim_i, dim_j).noalias() += hessian;
    // H.block(?,?, ?, ?).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();
        // H.block(?,?, ?, ?).noalias() += hessian.transpose();
    }
}
b.segment(index_i, dim_i).noalias() -= JtW * edge.second->Residual();
```

完成 SolveLinearSystem()

```
// 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);//左上角
MatXX Hpm = Hessian_.block(0 , reserve_size, reserve_size, marg_size);//左下角
MatXX Hmp = Hessian_.block(reserve_size, 0, marg_size, reserve_size);//右上角

VecX bpp = b_.segment(0,reserve_size);
VecX bmm = b_.segment(reserve_size,marg_size);


// 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 * Hpm.transpose();
b_pp_schur_ = bpp - tempH * bmm;



// TODO:: home work. step3: solve landmark
VecX delta_x_ll(marg_size);
// delta_x_ll = ???;
delta_x_ll = Hmm_inv * (bmm - Hpm.transpose() * delta_x_pp);//注意b的正负号
delta_x_.tail(marg_size) = delta_x_ll;
```

完成 Problem::TestMarginalize()

```
// 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(?,?,?,?) = temp_botRows;
// H_marg.block(?,?,?,?) = temp_rows;
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;
```

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

```
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: 1.02112 ms
   makeHessian cost: 0.381486 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 :-0.000478002   0.00115905  0.000366505 || 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**

- **请总结论文[a]：优化过程中处理 H 自由度的不同操作方式。内容包括：具体处理方式，实验效果，结论。(加分题，评选良好)**

论文中关于优化过程中处理 H 自由度的不同操作方式共有三种方法 gauge fixation approach、gauge prior approach 和 free gauge approach，具体的分析如下。

1、 具体的处理方式：

### A. Gauge Prior: Choosing the Appropriate Prior Weight

Before comparing the three approaches from Section III, we need to choose the prior covariance $\Sigma_0^P$ in the gauge prior approach. A common choice is $\Sigma_0^P = \sigma_0^2 \mathrm{I}$, for which the prior (11) becomes $\|\mathbf{r}_0^P\|_{\Sigma_0^P}^2 = w^P \|\mathbf{r}_0^P\|^2$, with $w^P = 1/\sigma_0^2$. We tested a wide range of the prior weight $w^P$ on different configurations and the results were similar. Therefore, we will look at one configuration in detail. Note that $w^P = 0$ is essentially the free gauge approach, whereas $w^P \to \infty$ is the gauge fixation approach. 两种方法的初始化假设

重要结论

- gauge fixation approach 通过将雅克比矩阵设为 0 来固定第一帧的优化参数。
- free gauge approach 对于奇异 H 矩阵使用伪逆或者 LM 的方法进行求解。
- gauge prior approach 通过在误差函数中加入一个 prior error，即第一帧位置初值与优化值的误差，并分配权重 wp，当 wp=0 时，相当于 free gauge approach，当 wp 趋于无穷大时，相当于 gauge fixation approach。

2、 处理效果:

The average RMSEs of 50 trials are listed in Table II. We omit the results for the gauge prior approach because they are identical to the ones from the gauge fixation approach up to around 8 digits after the decimal. It can be seen that there are only small differences between the free gauge approach and the gauge fixation approach, and neither of them has a better accuracy in all simulated configurations.

The convergence time and number of iterations are plotted in Fig. 7. The computational cost of the gauge prior approach and the gauge fixation approach are almost identical. The free gauge approach is slightly faster than the other two. Specifically, except for the *sine* trajectory with random 3D points, the free gauge approach takes fewer iterations and less time to converge. Note that the gauge fixation approach takes the least time per iteration due to the smaller number of variables in the optimization (see Table I).

三种方法有相似的性能，

- free gauge approach 的计算效率更快，迭代次数少，易收敛。
- 而且 gauge fixation approach 由于优化参数更少，每次迭代时间更短。
- Gauge Prior approach 中，权重大于一定阈值后，RMSE 稳定，迭代次数和收敛时间稳定。不同的先验权重对于算法的精度影响较小，对计算成本有影响，选择合适的先验权重可以加快计算速度。

### C. Discussion

Based on the results in this section, we conclude that:

- The three approaches have almost the same accuracy.
- In the gauge prior approach, one needs to select the proper prior weight to avoid increasing the computational cost.
- With a proper weight, the gauge prior approach has almost the same performance (accuracy and computational cost) as the gauge fixation approach.
- The free gauge approach is slightly faster than the others, because it takes fewer iterations to converge (cf. [14]).