参考链接

舒尔补(schur complement)

通过舒尔补操作，我们可以将一个矩阵

$$H=\begin{bmatrix}A & C^T\\ C & D\end{bmatrix}=\begin{bmatrix}I & 0\\ CA^{-1} & I\end{bmatrix}\begin{bmatrix}A & 0\\ 0 & \Delta_A\end{bmatrix}\begin{bmatrix}I & A^{-1}C^T\\ 0 & I\end{bmatrix}$$

$$H^{-1}=\begin{bmatrix}A & C^T\\ C & D\end{bmatrix}^{-1}=\begin{bmatrix}I & -A^{-1}C^T\\ 0 & I\end{bmatrix}\begin{bmatrix}A^{-1} & 0\\ 0 & \Delta_A^{-1}\end{bmatrix}\begin{bmatrix}I & 0\\ -CA^{-1} & I\end{bmatrix}$$

$$\Delta_A=D-CA^{-1}C^T,\text{也就是}A\text{关于矩阵}M\text{的舒尔补}$$

用这个技巧可以求解$HX=g$形式的方程

$$\begin{bmatrix}A & 0\\ 0 & \Delta_A\end{bmatrix}\begin{bmatrix}I & A^{-1}C^T\\ 0 & I\end{bmatrix}\begin{bmatrix}X_a\\ X_b\end{bmatrix}=\begin{bmatrix}I & 0\\ -CA^{-1} & I\end{bmatrix}\begin{bmatrix}g_a\\ g_b\end{bmatrix}$$

$$\begin{bmatrix}A & C^T\\ 0 & \Delta_A\end{bmatrix}\begin{bmatrix}X_a\\ X_b\end{bmatrix}=\begin{bmatrix}g_a\\ -CA^{-1}g_a+g_b\end{bmatrix}$$

注意到，X_b此时的解和X_a没有关系，也就是说

- 我们可以通过比较少的计算量先求得X_b，然后再将X_b带进去求X_a,这就加速了计算的过程
- 另一个就是关于边缘化的东西了，下面内容都是讲这个的。

边缘化

假设有

$$x=\begin{bmatrix}\alpha\\ \beta\end{bmatrix}\sim N(0,\begin{bmatrix}A & C^T\\ C & D\end{bmatrix})$$

那么，我们有公式如下:

$$\begin{aligned}P(\alpha,\beta)=&P(\alpha)P(\beta|\alpha)\\ \propto&exp\left(-\frac{1}{2}\begin{bmatrix}\alpha\\ \beta\end{bmatrix}^T\begin{bmatrix}A & C^T\\ C & D\end{bmatrix}^{-1}\begin{bmatrix}\alpha\\ \beta\end{bmatrix}\right)\\ =&exp\left(-\frac{1}{2}\begin{bmatrix}\alpha\\ \beta\end{bmatrix}^T\begin{bmatrix}I & -A^{-1}C^T\\ 0 & I\end{bmatrix}\begin{bmatrix}A^{-1} & 0\\ 0 & \Delta_A^{-1}\end{bmatrix}\begin{bmatrix}I & 0\\ -CA^{-1} & I\end{bmatrix}\begin{bmatrix}\alpha\\ \beta\end{bmatrix}\right)\\ =&\text{省略一些过程,直接暴力从左算到右就可以得到一下形式}\\ =&exp\left(-\frac{1}{2}\left(\alpha^TA^{-1}\alpha+(\beta-CA^{-1}\alpha)^T\Delta_A^{-1}(\beta-CA^{-1}\alpha)\right)\right)\\ =&exp\left(-\frac{1}{2}\alpha^TA^{-1}\alpha\right)exp\left(-\frac{1}{2}(\beta-CA^{-1}\alpha)^T\Delta_A^{-1}(\beta-CA^{-1}\alpha)\right)\end{aligned}$$

可以对应起来，即

$$P(\alpha)\propto exp\left(-\frac{1}{2}\alpha^TA^{-1}\alpha\right)$$

$$P(\beta|\alpha)\propto exp\left(-\frac{1}{2}(\beta-CA^{-1}\alpha)^T\Delta_A^{-1}(\beta-CA^{-1}\alpha)\right)$$

按照我的理解

边缘化所做的事情，就是从$P(\alpha,\beta)$里面假设α状态为确定值了，所以可以用$P(\beta|\alpha)$来表示新的$P(\beta)$

也就是从原本的

$$x = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \sim N(0, \begin{bmatrix} A & C^T \\ C & D \end{bmatrix})$$

变成了

$$x = [\beta - CA^{-1}\alpha] \sim N(0, \Delta_A)$$

这样似乎就是成功的减少掉了α这一个变量，同时协方差也得到了更新，均值也得到了更新。

在vins中应用

我们有很多的残差块。

IMU残差

$$r_1 = r_{imu} = f_1(x)$$

相机残差

$$r_2 = r_{cam} = f_2(x)$$

上一次边缘化的结果(如果有的话)

$$r_3 = r_m = f_3(x)$$

我们将至放在了一起做优化，其实相当于在解一个很大的方程

$$\begin{aligned} J &= J_1 + J_2 + J_3 \\ J^T J X &= J^T r \end{aligned} \tag{1}$$

我们这个(1)理论上应该已经包含了所有约束信息，此时，我们要从滑动窗口里面边缘化掉一部分的变量，在这之后这个方程理论上也要发生一些改变。即利用前面提到的舒尔补，对这个方程进行一些操作。

$$\begin{bmatrix} H_{mm} & H_{mr} \\ 0 & \Delta_{mm} \end{bmatrix} \begin{bmatrix} X_m \\ X_r \end{bmatrix} = \begin{bmatrix} g_m \\ -H_{rm}A^{-1}g_m + g_r \end{bmatrix}$$

$$\Delta_{mm} = H_{rr} - H_{rm}H_{mm}^{-1}H_{mr}$$

```
Eigen::VectorXd bmm = b.segment(0, m);      //边缘化的b
Eigen::MatrixXd Amr = A.block(0, m, m, n); //左下角
Eigen::MatrixXd Arm = A.block(m, 0, n, m); //右上角
Eigen::MatrixXd Arr = A.block(m, m, n, n); //右下角
Eigen::VectorXd brr = b.segment(m, n);      //保留的b
A = Arr - Arm * Amm_inv * Amr;              //舒尔补
b = brr - Arm * Amm_inv * bmm;              //舒尔补
```

我们同时可以从这里面分解出雅克比矩阵和残差

> 先证明一下Δ_{mm}为对称矩阵
>
> $$\begin{aligned} \Delta_{mm}^T =& (H_{rr} - H_{rm}H_{mm}^{-1}H_{mr})^T \\ =& H_{rr}^T - H_{mr}^T H_{mm}^{-T} H_{rm}^T \\ =& H_{rr} - H_{rm}H_{mm}^{-1}H_{mr} \\ =& \Delta_{mm} \end{aligned}$$

先对Δ_{mm}做特征分解

$$\Delta_{mm} = VDV^T$$

将对角矩阵D做分解,可以得到

$$D = S^T S = S^T S = SS$$

S为对角矩阵

我们把Δ_{mm}当做新的H矩阵，又$H = J^T J$,所以

$$J^T J = \Delta_{mm} = VDV^T = VSS^T V^T = (S^T V^T)^T S^T V^T$$

即

$$J = S^T V^T = SV^T$$

我们将$b = -H_{rm} A^{-1} g_m + g_r$当成新的$J^T r$

那么

$$J^T r = -H_{rm} A^{-1} g_m + g_r = b$$

那么

$$r = J^{-T} b = (VS^T)^{-1} b = S^{-1} V^T b$$

```
Eigen::SelfAdjointEigenSolver<Eigen::MatrixXd> saes2(A);
Eigen::VectorXd S = Eigen::VectorXd((saes2.eigenvalues().array() >
eps).select(saes2.eigenvalues().array(), 0));
Eigen::VectorXd S_inv = Eigen::VectorXd((saes2.eigenvalues().array() >
eps).select(saes2.eigenvalues().array().inverse(), 0));

Eigen::VectorXd S_sqrt = S.cwiseSqrt();
Eigen::VectorXd S_inv_sqrt = S_inv.cwiseSqrt();

linearized_jacobians = S_sqrt.asDiagonal() * saes2.eigenvectors().transpose();
linearized_residuals = S_inv_sqrt.asDiagonal() *
saes2.eigenvectors().transpose() * b;
```

也就是说，我们通过$r_1 r_2 r_3$,得到了新的r_3

这一项新的r_3也就是边缘化的项，它保存了被边缘化掉的一些历史信息，我们将这一项一起加到系统中做优化。

既然是把这一项加入到系统中做优化，那必不可少的求误差和雅克比矩阵。

r_3的残差

r_3在一开始生成的时候就已经代表的是这一项的残差了，但是系统是会更新的。假设原本的求$r_3(X_0)$是在X_0处求得的，现在的X更新了，也就是产生了新的增量$dx = X - X_0$,用一阶泰勒展开去更新即可

$$r_3 \leftarrow r_3 + Jdx$$

雅克比则假设不更新了，就是原来的J

```
Eigen::Map<Eigen::VectorXd>(residuals, n) = marginalization_info->linearized_residuals + marginalization_info->linearized_jacobians * dx;
//....
jacobian.leftCols(local_size) = marginalization_info->linearized_jacobians.middleCols(idx, local_size);
```

First Estimate Jacobian(FEJ)

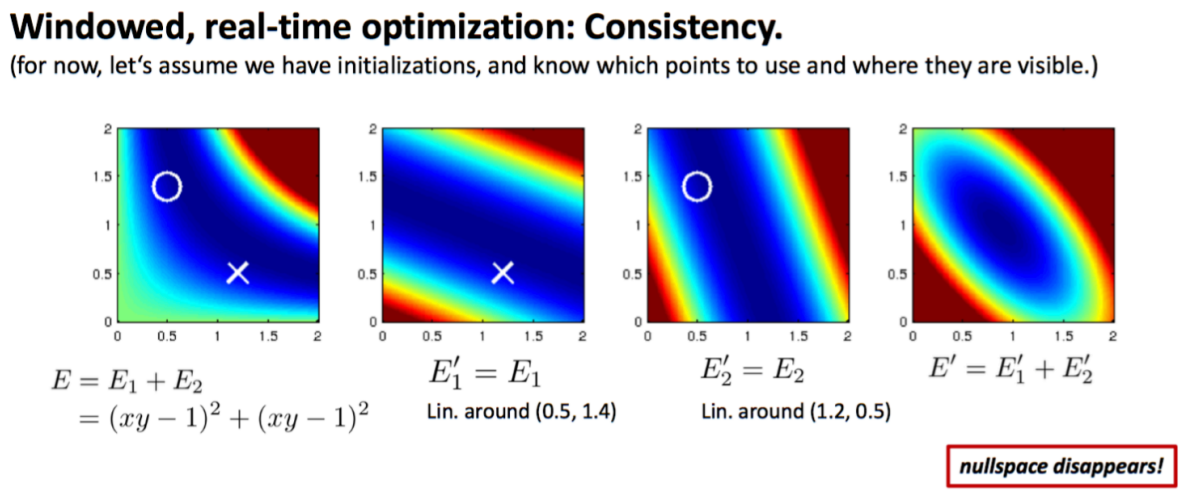
vins中并没有使用这个东西，原因是说加了效果反而不是很好，不加也没有观测到一些异常。

主要就是在不同的地方线性化导致了一些问题

r_3这一项是由$r_1 r_2$ 旧的r_3组成的，其中，r_1 r_2是每次都是在最新的估计点附近展开的，而旧的r_3则包含了部分信息是在旧的历史点上所展开的，这就导致了线性点展开不太一致的问题.

> 我粗略的理解是在求舒尔补的时候，所使用到的矩阵是由那一时刻的J所决定的,显然，那一时刻的X_0和最新时刻的X不一样，也就是说，这个J矩阵，有些地方的元素是在X_0求得的，有些是在X时刻求得的。

这就会导致



- 第一张图，零空间是一条直线,$xy = 1$
- 第二张图，如果E_1在(0.5,1.4)展开，那么零空间是一条直线，虽然我不知道怎么展开
- 第三张图，如果E_2在(1.2,0.5)展开，那么零空间是一条直线，虽然我不知道怎么展开
- 第四张图，把两个函数合起来，原问题的零空间消失了，也就是说无解。