2 熟悉 Eigen 矩阵运算 (3 分,约 2 小时)

设线性方程 Ax = b,在 A 为方阵的前提下,请回答以下问题:

1. 在什么条件下,x 有解且唯一?

R(A)=n，n 为未知数个数。
即矩阵的秩等于未知数个数。

2. 高斯消元法的原理是什么?

高斯消元是逐步的消除线性方程组中的未知数，来构造一个行阶梯矩阵（上/下三角），然后回代解出每一个未知数的办法。举例来说,有三个等式

$$x + y + z = 6$$
$$3x + 2y + z = 14$$
$$2x - y + z = 5$$

构造增广矩阵

$$\left|\begin{matrix}1 & 1 & 1\\3 & 2 & 1\\2 & -1 & 1\end{matrix}\right|\begin{matrix}6\\14\\5\end{matrix}\right|$$

这个矩阵$\mathrm{Row}_2 - 3 * Row_1, Row_3 - 2 * Row_1$得到(消除一个未知数)

$$\left|\begin{matrix}1 & 1 & 1\\0 & -1 & -2\\0 & -3 & -1\end{matrix}\right|\begin{matrix}6\\-4\\-7\end{matrix}\right|$$

再$\mathrm{Row}_3 + 3 * Row_2$得到上三角矩阵（再消除一个未知数）

$$\left|\begin{matrix}1 & 1 & 1\\0 & -1 & -2\\0 & 0 & 5\end{matrix}\right|\begin{matrix}6\\-4\\5\end{matrix}\right|$$

也就得到了三个等式

$$x + y + z = 6$$
$$-y - 2z = -4$$
$$5z = 5$$

从下往上代入解出的未知数，得到 z=1, y=2, x=3。

3. QR 分解的原理是什么?

QR 分解是把一个满秩矩阵 A 分解成 A=QR 的式子。其中 Q 是一个正交矩阵，R 是一个上三角矩阵。这样 Ax=b 就可以转化为 QRx=b，左右代入Q^{-1}得到$\mathrm{Q}^{-1}QRx = Q^{-1}b 即 Rx = Q^{T}b$.
Q 很容易计算，R 是上三角矩阵，最后就跟上题高斯消元一样回代解出结果。
其实它是把 A 矩阵（举例 3x3 矩阵）的列向量组$\vec{a}_1, \vec{a}_2, \vec{a}_3$进行 gram-schmidt 正交化，得到一组正交的基$\vec{q}_1, \vec{q}_2, \vec{q}_3$，A 矩阵等于一组常数和正交基组的乘积，比如

$$\mathrm{A}_{n,1} = r_{11}q_1$$
$$A_{n,2} = r_{1,2}q_1 + r_{2,2}q_2$$
$$A_{n,3} = r_{1,3}q_1 + r_{2,3}q_2 + r_{3,3}q_3$$

即

$$\begin{bmatrix}A_{n,1} & A_{n,2} & A_{n,3}\end{bmatrix} = \begin{bmatrix}q_{n,1} & q_{n,2} & q_{n,3}\end{bmatrix} * \begin{bmatrix}r_{11} & r_{12} & r_{13}\\0 & r_{22} & r_{23}\\0 & 0 & r_{33}\end{bmatrix}$$

我们知道 A，我们知道 q，则 R 可知。
这种分解可以简化计算，主要是避免算A^{-1}.

4. Cholesky 分解的原理是什么?

同样有 Ax=B，如果 A 对称正定，则存在$A = LL^T$，L 为对角元为正数的下三角矩阵。（Cholesky 分解即是 LU 分解的特殊情况）

$$即A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{bmatrix} * \begin{bmatrix} L_{11} & L_{21} & L_{31} \\ 0 & L_{22} & L_{32} \\ 0 & 0 & L_{33} \end{bmatrix}$$

$$然后a_{11} = L_{11} * L_{11}，L_{11} = sqrt(a_{11})$$

$$a_{21} = L_{21}L_{11}, \quad L_{21} = \frac{a_{21}}{L_{11}}$$

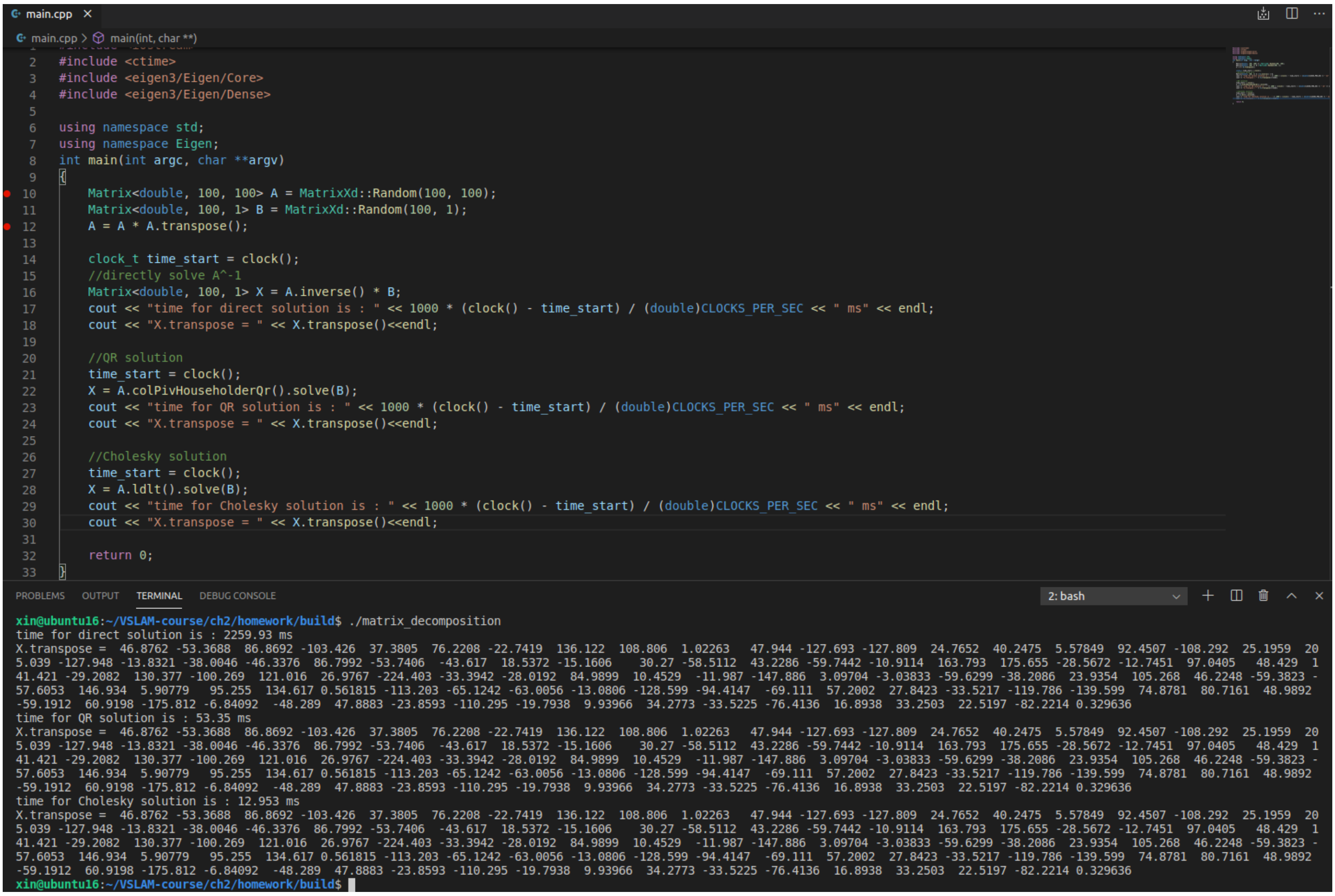
$$a_{31} = L_{31}L_{11}, \quad L_{31} = \frac{a_{31}}{L_{11}}$$

$$总结出来：L_{:,1} = \frac{a_{:,1}}{L_{11}}$$

另外易知对角元素$a_{kk} = \sum_i^k L_{ki}^2$ 则可以得到L_{kk}(比如$a_{22} = L_{21}^2 + L_{22}^2, L_{21}已知，可知L_{22}$)
还有$a_{ik} = \sum_j^{k-1} L_{ij}L_{kj} + L_{ik}L_{kk}$ $(i = k+1, \dots, n)$则可以得到L_{ik}(比如$a_{32} = L_{31}L_{21} + L_{32}L_{22}$)
可以以此计算每个元素得到整个 L 矩阵

5. 编程实现 A 为 100 × 100 随机矩阵时,用 QR 和 Cholesky 分解求 x 的程序。

直接法用时 2259.93ms； QR 分解 53.35ms； Cholesky 分解 12.953ms



```cpp
#include <ctime>
#include <eigen3/Eigen/Core>
#include <eigen3/Eigen/Dense>

using namespace std;
using namespace Eigen;
int main(int argc, char **argv)
{
    Matrix<double, 100, 100> A = MatrixXd::Random(100, 100);
    Matrix<double, 100, 1> B = MatrixXd::Random(100, 1);
    A = A * A.transpose();

    clock_t time_start = clock();
    //directly solve A^-1
    Matrix<double, 100, 1> X = A.inverse() * B;
    cout << "time for direct solution is : " << 1000 * (clock() - time_start) / (double)CLOCKS_PER_SEC << " ms" << endl;
    cout << "X.transpose = " << X.transpose()<<endl;

    //QR solution
    time_start = clock();
    X = A.colPivHouseholderQr().solve(B);
    cout << "time for QR solution is : " << 1000 * (clock() - time_start) / (double)CLOCKS_PER_SEC << " ms" << endl;
    cout << "X.transpose = " << X.transpose()<<endl;

    //Cholesky solution
    time_start = clock();
    X = A.ldlt().solve(B);
    cout << "time for Cholesky solution is : " << 1000 * (clock() - time_start) / (double)CLOCKS_PER_SEC << " ms" << endl;
    cout << "X.transpose = " << X.transpose()<<endl;

    return 0;
}
```

```
xin@ubuntu16:~/VSLAM-course/ch2/homework/build$ ./matrix_decomposition
time for direct solution is : 2259.93 ms
X.transpose =  46.8762 -53.3688  86.8692 -103.426  37.3805  76.2208 -22.7419  136.122  108.806  1.02263   47.944 -127.693 -127.809  24.7652  40.2475  5.57849  92.4507 -108.292  25.1959  20
5.039 -127.948 -13.8321 -38.0046 -46.3376  86.7992 -53.7406  -43.617  18.5372 -15.1606    30.27 -58.5112  43.2286 -59.7442 -10.9114  163.793  175.655 -28.5672 -12.7451  97.0405   48.429  1
41.421 -29.2082  130.377 -100.269  121.016  26.9767 -224.403 -33.3942 -28.0192  84.9899  10.4529  -11.987 -147.886  3.09704 -3.03833 -59.6299 -38.2086  23.9354  105.268  46.2248 -59.3823 -
57.6053  146.934  5.90779   95.255  134.617 0.561815 -113.203 -65.1242 -63.0056 -13.0806 -128.599 -94.4147  -69.111  57.2002  27.8423 -33.5217 -119.786 -139.599  74.8781  80.7161  48.9892
-59.1912  60.9198 -175.812 -6.84092  -48.289  47.8883 -23.8593 -110.295 -19.7938  9.93966  34.2773 -33.5225 -76.4136  16.8938  33.2503  22.5197 -82.2214 0.329636
time for QR solution is : 53.35 ms
X.transpose =  46.8762 -53.3688  86.8692 -103.426  37.3805  76.2208 -22.7419  136.122  108.806  1.02263   47.944 -127.693 -127.809  24.7652  40.2475  5.57849  92.4507 -108.292  25.1959  20
5.039 -127.948 -13.8321 -38.0046 -46.3376  86.7992 -53.7406  -43.617  18.5372 -15.1606    30.27 -58.5112  43.2286 -59.7442 -10.9114  163.793  175.655 -28.5672 -12.7451  97.0405   48.429  1
41.421 -29.2082  130.377 -100.269  121.016  26.9767 -224.403 -33.3942 -28.0192  84.9899  10.4529  -11.987 -147.886  3.09704 -3.03833 -59.6299 -38.2086  23.9354  105.268  46.2248 -59.3823 -
57.6053  146.934  5.90779   95.255  134.617 0.561815 -113.203 -65.1242 -63.0056 -13.0806 -128.599 -94.4147  -69.111  57.2002  27.8423 -33.5217 -119.786 -139.599  74.8781  80.7161  48.9892
-59.1912  60.9198 -175.812 -6.84092  -48.289  47.8883 -23.8593 -110.295 -19.7938  9.93966  34.2773 -33.5225 -76.4136  16.8938  33.2503  22.5197 -82.2214 0.329636
time for Cholesky solution is : 12.953 ms
X.transpose =  46.8762 -53.3688  86.8692 -103.426  37.3805  76.2208 -22.7419  136.122  108.806  1.02263   47.944 -127.693 -127.809  24.7652  40.2475  5.57849  92.4507 -108.292  25.1959  20
5.039 -127.948 -13.8321 -38.0046 -46.3376  86.7992 -53.7406  -43.617  18.5372 -15.1606    30.27 -58.5112  43.2286 -59.7442 -10.9114  163.793  175.655 -28.5672 -12.7451  97.0405   48.429  1
41.421 -29.2082  130.377 -100.269  121.016  26.9767 -224.403 -33.3942 -28.0192  84.9899  10.4529  -11.987 -147.886  3.09704 -3.03833 -59.6299 -38.2086  23.9354  105.268  46.2248 -59.3823 -
57.6053  146.934  5.90779   95.255  134.617 0.561815 -113.203 -65.1242 -63.0056 -13.0806 -128.599 -94.4147  -69.111  57.2002  27.8423 -33.5217 -119.786 -139.599  74.8781  80.7161  48.9892
-59.1912  60.9198 -175.812 -6.84092  -48.289  47.8883 -23.8593 -110.295 -19.7938  9.93966  34.2773 -33.5225 -76.4136  16.8938  33.2503  22.5197 -82.2214 0.329636
xin@ubuntu16:~/VSLAM-course/ch2/homework/build$
```

3 几何运算练习 (2 分,约 1 小时)

请编程实现此事,并提交你的程序。

```cpp
#include <iostream>
#include <ctime>
#include <eigen3/Eigen/Core>
#include <eigen3/Eigen/Dense>

using namespace std;
using namespace Eigen;
int main(int argc, char **argv)
{
    //T_c1w
    Quaterniond q1 = Quaterniond(0.55, 0.3, 0.2, 0.2);
    q1.normalize();
    Vector3d t1 = Vector3d(0.7, 1.1, 0.2);
    Matrix4d T_c1w = Matrix4d::Identity();
    T_c1w.block(0, 0, 3, 3) = q1.matrix();
    T_c1w.block(0, 3, 3, 1) = t1;
    //T_c2w
    Quaterniond q2 = Quaterniond(-0.1, 0.3, -0.7, 0.2);
    q2.normalize();
    Vector3d t2 = Vector3d(-0.1, 0.4, 0.8);
    Matrix4d T_c2w = Matrix4d::Identity();
    T_c2w.block(0, 0, 3, 3) = q2.matrix();
    T_c2w.block(0, 3, 3, 1) = t2;
    //V_c1
    Vector4d p1 = Vector4d(0.5, -0.1, 0.2, 1);
    //V_w = T_c1w^-1 * V_c1
    Vector4d V_w = T_c1w.inverse() * p1;
    //V_c2 = T_c2w * V_w
    Vector4d V_c2 = T_c2w * V_w;
    cout <<"V_c2"<<endl<< V_c2 << endl;

    return 0;
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
xin@ubuntu16:~/VSLAM-course/ch2/homework/build$ ./robot
V_c2
 1.08228
0.663509
0.686957
       1
```

4 旋转的表达 (2 分,约 1 小时)

1. 设有旋转矩阵 R,证明$R^T R = I$且 det R = +1^2

假设有一组正交基$[e_1, e_2, e_3]$构成坐标系 1，经过一次旋转变成$[e_1', e_2', e_3']$构成坐标系 2,对于同一个向量 a，他在两个坐标系下的坐标分别为$[a_1, a_2, a_3]^T$和$[a_1', a_2', a_3']^T$.

把这两个向量用基和系数的乘积来表示，则

$$[e_1, e_2, e_3]\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = [e_1', e_2', e_3']\begin{bmatrix} a_1' \\ a_2' \\ a_3' \end{bmatrix}$$

两边同时左乘以$[e_1, e_2, e_3]^T$,得到旋转矩阵R_2^1定义如下

$$I * \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} e_1^T e_1' & e_1^T e_2' & e_1^T e_3' \\ e_2^T e_1' & e_2^T e_2' & e_2^T e_3' \\ e_3^T e_1' & e_3^T e_2' & e_3^T e_3' \end{bmatrix}\begin{bmatrix} a_1' \\ a_2' \\ a_3' \end{bmatrix} = R_2^1 * \begin{bmatrix} a_1' \\ a_2' \\ a_3' \end{bmatrix}$$

那我们反过来

$$[e_1', e_2', e_3']\begin{bmatrix} a_1' \\ a_2' \\ a_3' \end{bmatrix} = [e_1, e_2, e_3]\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

两边同时乘以$[e_1', e_2', e_3']^T$,得到旋转矩阵${R_2^1}^{-1}$定义如下

$$I * \begin{bmatrix} a_1' \\ a_2' \\ a_3' \end{bmatrix} = \begin{bmatrix} {e_1'}^T e_1 & {e_1'}^T e_2 & {e_1'}^T e_3 \\ {e_2'}^T e_1 & {e_2'}^T e_2 & {e_2'}^T e_3 \\ {e_3'}^T e_1 & {e_3'}^T e_2 & {e_3'}^T e_3 \end{bmatrix}\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = R_1^2 * \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

可见${R_2^1}^{-1} = R_1^2 = {R_2^1}^T$，旋转矩阵的逆等于它的转置,则有$R^T R = R^{-1} R = I$。

同时证明旋转矩阵各列两两正交，举例来说,设$e_1 = (x, 0,0)^T, e_2 = (0, y, 0)^T, e_3 = (0,0, z)^T, e_1' = (x', 0,0)^T, e_2' = (0, y', 0)^T, e_3' = (0,0, z')^T$。

以R_2^1*第一列第二列为例,第一列* $= (xx', 0,0)^T$, *第二列* $= (0, yy', 0)^T$。显然这两列的点积为 0

则旋转矩阵是正交矩阵，其行列式等于±1。

2. 设有四元数 q，我们把虚部记为 ε，实部记为 η，那么 q = (ε,η)。请说明 ε 和 η 的维度。

实部 1 维，虚部 3 维

3. 请证明对任意单位四元数 q1, q2，四元数乘法可写成矩阵乘法:

正常向量情况的表达式$q_1 q_2 = [\eta_a \varepsilon_b + \eta_b \varepsilon_a + \varepsilon_a^{\times} \varepsilon_b, \eta_a \eta_b - \varepsilon_a^T \varepsilon_b]^T$

定义的$q_1^{+} q_2 = \begin{bmatrix} \eta_a I + \varepsilon_a^{\times} & \varepsilon_a \\ -\varepsilon_a^T & \eta_a \end{bmatrix}\begin{bmatrix} \varepsilon_b \\ \eta_b \end{bmatrix}$

$$= \begin{bmatrix} \eta_a I \varepsilon_b + \varepsilon_a^{\times} \varepsilon_b + \varepsilon_a \eta_b \\ -\varepsilon_a^T \varepsilon_b + \eta_a \eta_b \end{bmatrix}$$

矩阵表达式结果跟上面向量表达式一样。

定义的$q_2^{\oplus} \quad q_1 = \begin{bmatrix} \eta_b I - \varepsilon_b^{\times} & \varepsilon_b \\ -\varepsilon_b^T & \eta_b \end{bmatrix}\begin{bmatrix} \varepsilon_a \\ \eta_a \end{bmatrix}$

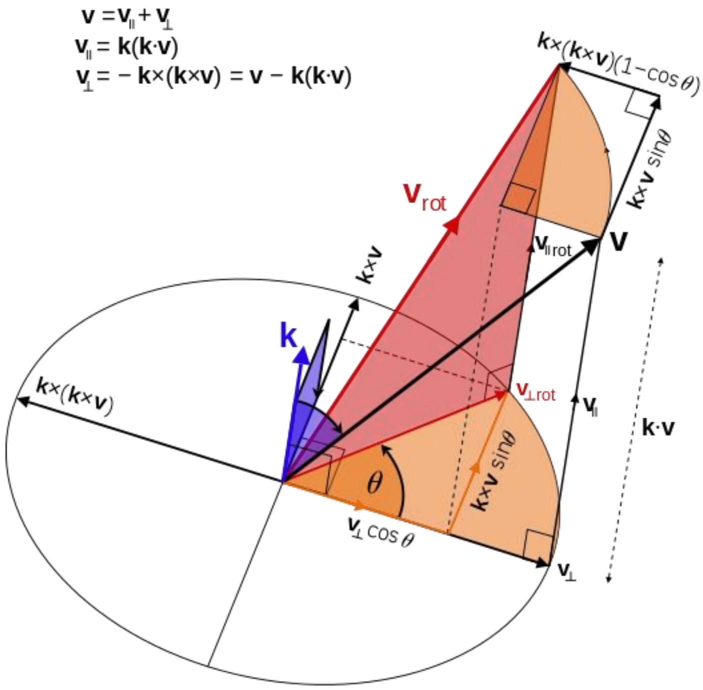
$$= \begin{bmatrix} \eta_b I \varepsilon_a - \varepsilon_b^{\times} \varepsilon_a + \varepsilon_b \eta_a \\ -\varepsilon_b^T \varepsilon_a + \eta_b \eta_a \end{bmatrix}$$

我们只需关注$-\varepsilon_b^{\times}\varepsilon_a=\begin{bmatrix}0 & z_b & -y_b\\ -z_b & 0 & x_b\\ y_b & -x_b & 0\end{bmatrix}\begin{bmatrix}x_a\\ y_a\\ z_a\end{bmatrix}=\begin{bmatrix}z_by_a-y_bz_a\\ -z_bx_a+x_bz_a\\ y_bx_a-x_by_a\end{bmatrix}$

$$\varepsilon_a^{\times}\varepsilon_b=\begin{bmatrix}0 & -z_a & y_a\\ z_a & 0 & -x_a\\ -y_a & x_a & 0\end{bmatrix}\begin{bmatrix}x_b\\ y_b\\ z_b\end{bmatrix}=\begin{bmatrix}-z_ay_b+y_az_b\\ z_ax_b-x_az_b\\ -y_ax_b+x_ay_b\end{bmatrix}$$

即叉积的负交换律$-\varepsilon_b^{\times}\varepsilon_a=\varepsilon_a^{\times}\varepsilon_b$，则第一行部分和向量表达式第一行也一样。

5 罗德里格斯公式的证明 (2 分，约 1 小时)



以上图为例，定义向量V和绕 k 旋转θ之后得到向量V_{rot}，$V_{\perp}$为V垂直于k的分量，$V_{\parallel}$为V平行于k的分量。$k\times V$为垂直于$k,V_{\perp}$平面的向量，也可以理解为$V_{\perp}$围绕k旋转 90° 之后的结果，那$k\times(k\times V)$就是$V_{\perp}$围绕k旋转 180° 的结果，有$V_{\perp}=-k\times(k\times V)$。

推导如下:

V_{rot}等于其垂直于 k 的分量 + 其平行于 k 的分量: $V_{rot}=V_{rot\perp}+V_{rot\parallel}$

旋转不会改变 V 在 k 上的分量，则$V_{rot\parallel}=V_{\parallel}$

根据向量投影公式有$V_{\parallel}=(v*k)k$

另外从极坐标的表示方式来说，把图中的圆看作xoy坐标系，则有向量$\boldsymbol{r}=rcos\theta e_x+rsin\theta e_y$。(r 为标量，半径。$e_xe_y$为 xy 方向单位向量)

则$V_{rot\perp}=cos\theta V_{\perp}+sin\theta k\times V$

$$\text{则}V_{rot}=V_{rot\perp}+V_{rot\parallel}$$
$$=cos\theta V_{\perp}+sin\theta k\times V+V_{\parallel}$$
$$=cos\theta(V-V_{\parallel})+sin\theta k\times V+V_{\parallel}$$
$$=cos\theta V-cos\theta V_{\parallel}+V_{\parallel}+sin\theta k\times V$$
$$V_{rot}=cos\theta V+(1-cos\theta)(k*V)k+\ \mathrm{sin\theta}*k\times V$$
$$V_{rot}=(cos\theta+(1-cos\theta)kk^T+\ \mathrm{sin\theta}*k^{\wedge})V$$
$$\boldsymbol{R}=\ \boldsymbol{cos\theta}+(\boldsymbol{1}-\boldsymbol{cos\theta})\boldsymbol{kk^T}+\ \mathbf{sin\theta}*\boldsymbol{k}^{\wedge}$$

6 四元数运算性质的验证 (1 分，约 1 小时)

1. 此时 p′ 必定为虚四元数（实部为零）。请你验证上述说法

假设有点$p=[0,x_p,y_p,z_p]^T$，四元数$q_a=[s_a,x_a,y_a,z_a]^T$，令$p'=q_apq_a^{-1}$

q_ap代入四元数乘法向量形式得到$q_ap=[0s_a-v_p^Tv_a,0v_a+s_av_p+v_p\times v_a]^T$

$$q_ap=[-v_p^Tv_a,s_av_p+v_p\times v_a]^T$$

定义$q_ap=q_{former}=[s_f,x_f,y_f,z_f]^T=[-v_p^Tv_a,s_av_p+v_p\times v_a]^T$

$$\text{则实部}\mathbf{s_f}=-(\mathbf{x_px_a}+\mathbf{y_p}\boldsymbol{y_a}+\boldsymbol{z_pz_a})$$

$$\text{而虚部}\mathbf{v_f}=[\mathbf{s_ax_p}-\mathbf{z_py_a}+\mathbf{y_pz_a},\quad \mathbf{s_ay_p}+\mathbf{z_px_a}-\mathbf{x_pz_a},\quad \mathbf{s_az_p}-\mathbf{y_px_a}+\mathbf{x_py_a}]^{\mathbf{T}}$$

剩下的$q_a^{-1}=\frac{q_a^*}{\|q_a\|^2}=\frac{[s_a,-v_a]^T}{s_a^2+x_a^2+y_a^2+z_a^2}$

定义$q_a^{-1}=q_{later}=[s_l,x_l,y_l,z_l]^T=\frac{[s_a,-v_a]^T}{s_a^2+x_a^2+y_a^2+z_a^2}$

$$\text{则实部}\boldsymbol{s_l}=\frac{\boldsymbol{s_a}}{\boldsymbol{s_a^2+x_a^2+y_a^2+z_a^2}}$$

$$\text{而虚部}\boldsymbol{v_l}=\frac{-[\mathbf{x_a},\boldsymbol{y_a},\boldsymbol{z_a}]^{\mathbf{T}}}{\boldsymbol{s_a^2+x_a^2+y_a^2+z_a^2}}$$

q_{former}，q_{later}代入乘法向量形式有$q_{former}*q_{later}=[s_fs_l-v_f^Tv_l,\text{虚部略}]^T$

现在只需要验证$s_fs_l-v_f^Tv_l$是否等于 0，代入上面得到的$\boldsymbol{s_f}$，$\boldsymbol{s_l}$，$\boldsymbol{v_f},\boldsymbol{v_l}$

$$s_fs_l=\frac{-(x_px_a+y_py_a+z_pz_a)*s_a}{s_a^2+x_a^2+y_a^2+z_a^2}$$

$$-v_f^Tv_l=-[s_ax_p-z_py_a+y_pz_a\quad s_ay_p+z_px_a-x_pz_a\quad s_az_p-y_px_a+x_py_a]*\begin{matrix}x_a\\y_a\\z_a\end{matrix}*\frac{-1}{s_a^2+x_a^2+y_a^2+z_a^2}$$

我们忽略分母$s_a^2+x_a^2+y_a^2+z_a^2$(另外，$-v_f^Tv_l$里俩-1 抵消了)简化一下

$$(s_fs_l-v_f^Tv_l)_{\text{分子}}=(-x_px_as_a-y_py_as_a-z_pz_as_a)+s_ax_px_a-z_py_ax_a+y_pz_ax_a+s_ay_py_a+z_px_ay_a-x_pz_ay_a+s_az_pz_a-y_px_az_a+x_py_az_a=0$$

2. 上式亦可写成矩阵运算：p′ = Qp，请根据你的推导，给出矩阵 Q。

$$q^+=\begin{bmatrix}\eta I+\varepsilon^\times & \varepsilon\\ -\varepsilon^T & \eta\end{bmatrix},q^\oplus=\begin{bmatrix}\eta I-\varepsilon^\times & \varepsilon\\ -\varepsilon^T & \eta\end{bmatrix}$$

$$p'=q^+q^{-1^\oplus}p$$

$$p'=\begin{bmatrix}\eta I+\varepsilon^\times & \varepsilon\\ -\varepsilon^T & \eta\end{bmatrix}\begin{bmatrix}\eta I+\varepsilon^\times & -\varepsilon\\ \varepsilon^T & \eta\end{bmatrix}p$$

$$p'=\begin{bmatrix}\eta I^2+2\eta I\varepsilon^\times+\varepsilon^{\times 2}+\varepsilon\varepsilon^T & -\eta I\varepsilon-\varepsilon^T\varepsilon^\times+\varepsilon\eta\\ -\varepsilon^T\eta I-\varepsilon^T\varepsilon^\times+\eta\varepsilon^T & \varepsilon^T\varepsilon+\eta^2\end{bmatrix}p$$

$$p'=\begin{bmatrix}\varepsilon\varepsilon^T+\eta I^2+2\eta I\varepsilon^\times+\varepsilon^{\times 2} & 0\\ 0 & 1\end{bmatrix}p$$

(注意，Q 左上角是旋转矩阵 R)

7 * 熟悉 C++11 (2 分，约 1 小时)

非静态数据成员初始化 int index = 0;
C++11 允许在定义的时候对非 static 非 const 成员初始化。

拓展初始化列表 vector<A> avec{a1,a2,a3};
以前只有数组可以使用初始化列表，现在其他容器可以用{}实现。

lambda 表达式 [](const A&a1, const A&a2){return a1.index<a2.index;}
创建匿名的函数对象，简化工作
[]代表函数可以获得的全局变量,（）形参，->返回类型（此处没有定义）,{}函数体

自动类型 auto & a:avec
auto 从初始化表达式中推断数据类型，比如 auto i = 1.0;
编译时进行推导，不影响运行效率。
也不影响编译速度，因为本来也要判断右侧类型跟左侧是否匹配。

range-base for loop for (auto & a:avec)
有 iterator 的序列可以用这种简化的 for 循环