蒋超，PA2

一、

1. 当$\mathrm{r}(\mathrm{A}) = r(B) = n$，x 有解且唯一，r 为 A、B 的秩，n 为维度。

2. 高斯消元法，也就是我们中学学过的消元法，将方程组中一方程的未知数用含有另一未知数的代数式表示，并将其带入到另一方程中，这样就消去了一个元，得到一个解。高斯消元法主要用于二元一次方程组的求解。

3. QR 分解（正交三角）

定义：对于 n 阶方阵 A，若存在正交矩阵 Q 和上三角矩阵 R，使得 A = QR，则该式称为矩阵 A 的完全 QR 分解或正交三角分解。

过程：其中，正交矩阵 Q 来自 Gram-Schmidt 正交化方法，R 来自$\mathrm{R} = Q^T A$。

作用：是目前求一般矩阵全部**特征值和特征向量**最有效且广泛应用的方法。

4. Cholesky 分解（平方根法）

定理：若 $\mathrm{A} \in R^{n\times n}$ 对称正定，则存在一个对角元为正数的下三角矩阵 $\mathrm{L} \in R^{n\times n}$，使得 $\mathrm{A} = \mathrm{L}L^T$ 成立。

过程：即利用该等式，开方、递推计算。

作用：当线性方程组 $\mathrm{Ax} = \mathrm{B}$，其中 A 为正定对称矩阵时，有特定解法。

5.

代码：

```
#include <iostream>
using namespace std;
#include <ctime>
#include <Eigen/Core>
#include <Eigen/Dense>

#define MATRIX_SIZE 100

int main() {

    // 如果不确定矩阵大小，可以使用动态大小的矩阵
    Eigen::Matrix< double, Eigen::Dynamic, Eigen::Dynamic > matrix_NN;

    // 我们求解 matrix_NN * x = v_Nd 这个方程
    // Eigen::Matrix< double, MATRIX_SIZE, MATRIX_SIZE > matrix_NN;
    matrix_NN = Eigen::MatrixXd::Random( MATRIX_SIZE, MATRIX_SIZE );
    Eigen::Matrix< double, MATRIX_SIZE,    1> v_Nd;
    v_Nd = Eigen::MatrixXd::Random( MATRIX_SIZE,1 );

    clock_t time_stt = clock(); // 计时
    // 直接求逆
    Eigen::Matrix<double,MATRIX_SIZE,1> x = matrix_NN.inverse()*v_Nd;
    cout<<"time use in normal inverse is " << 1000* (clock() - time_stt)/(double)CLOCKS_PER_SEC << "ms"<<
endl;
```

```
    // 通常用矩阵分解来求，例如 QR 分解，速度会快很多
    time_stt = clock();
    x = matrix_NN.colPivHouseholderQr().solve(v_Nd);
    cout <<"time use in Qr decomposition is " <<1000* (clock() - time_stt)/(double)CLOCKS_PER_SEC <<"ms" << endl;

    // Cholesky  分解
    time_stt = clock();
    x = matrix_NN.llt().solve(v_Nd);
    cout<<"time use in Cholesky decomposition is "<<1000* (clock() - time_stt)/(double)CLOCKS_PER_SEC <<"ms" << endl;

    return 0;
}
```

运行结果：

```
Eigen_1
/home/jc/mySLAM/PA2/Eigen_1/cmake-build-debug/Eigen_1
time use in normal inverse is 0.604ms
time use in Qr decomposition is 0.317ms
time use in Cholesky decomposition is 0.018ms

Process finished with exit code 0
```

二、

代码：

```
#include <iostream>
#include <Eigen/Core>
#include <Eigen/Geometry>

int main() {
    Eigen::Quaterniond q1(0.55, 0.3, 0.2, 0.2);
    Eigen::Vector3d t1(0.7, 1.1, 0.2);
    Eigen::Quaterniond q2(-0.1, 0.3, -0.7, 0.2);
    Eigen::Vector3d t2(-0.1, 0.4, 0.8);
    Eigen::Vector3d p1 = Eigen::Vector3d(0.5, -0.1, 0.2);
    Eigen::Vector3d p;
    Eigen::Vector3d p2;

    // Tcw1
    Eigen::Isometry3d Tcw1 = Eigen::Isometry3d::Identity();
    Tcw1.rotate(q1.normalized());
    Tcw1.pretranslate(t1);
    // Tcw * p = p1
    p = Tcw1.inverse() * p1;
    std::cout<<"p = "<<std::endl;
```

```
    std::cout<<p.transpose()<<std::endl;
    // Tcw2
    Eigen::Isometry3d Tcw2 = Eigen::Isometry3d::Identity();
    Tcw2.rotate(q2.normalized());
    Tcw2.pretranslate(t2);
    // Tcw2 * p = p2
    p2 = Tcw2 * p;
    std::cout<<"p2 = "<<std::endl;
    std::cout<<p2.transpose()<<std::endl;
    return 0;
}
```

运行结果：

```
Eigen_2
/home/jc/mySLAM/PA2/Eigen_2/cmake-build-debug/Eigen_2
p =
-0.995767 -0.497354  0.491005
p2 =
 1.08228 0.663509 0.686957

Process finished with exit code 0
```

三、

1. 由 $\mathrm{Ra} = a' \ \rightarrow \ \ a = R^{-1}a'$;

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{e}_1^T\boldsymbol{e}_1^{'} & \boldsymbol{e}_1^T\boldsymbol{e}_2^{'} & \boldsymbol{e}_1^T\boldsymbol{e}_3^{'} \\ \boldsymbol{e}_2^T\boldsymbol{e}_1^{'} & \boldsymbol{e}_2^T\boldsymbol{e}_2^{'} & \boldsymbol{e}_2^T\boldsymbol{e}_3^{'} \\ \boldsymbol{e}_3^T\boldsymbol{e}_1^{'} & \boldsymbol{e}_3^T\boldsymbol{e}_2^{'} & \boldsymbol{e}_3^T\boldsymbol{e}_3^{'} \end{bmatrix} \begin{bmatrix} a_1^{'} \\ a_2^{'} \\ a_3^{'} \end{bmatrix} \triangleq \boldsymbol{R}\boldsymbol{a}^{'}.$$

中，矩阵可改写为$\begin{bmatrix} e_1^T \\ e_2^T \\ e_3^t \end{bmatrix}$，即$R^T$，可以的出$R^{-1} = R^T$；即$R^TR = I$;

$|R| = |R^T|,\ |R^T||R| = 1 \ \rightarrow \ |R| = 1,$

2. ε 维度为 3， η 维度为 1

3. 设: $q_1 = (s_1,\ x_1,\ y_1,\ z_1)$，$q_2 = (s_2,\ x_2,\ y_2,\ z_2)$

则：$q_1^{+}q_2 = \begin{bmatrix} s_1 & -z_1 & y_1 & x_1 \\ z_1 & s_1 & -x_1 & y_1 \\ -y_1 & x_1 & s_1 & z_1 \\ -x_1 & -y_1 & -x_3 & s_1 \end{bmatrix} \begin{bmatrix} s_2 \\ x_2 \\ y_2 \\ z_2 \end{bmatrix}$

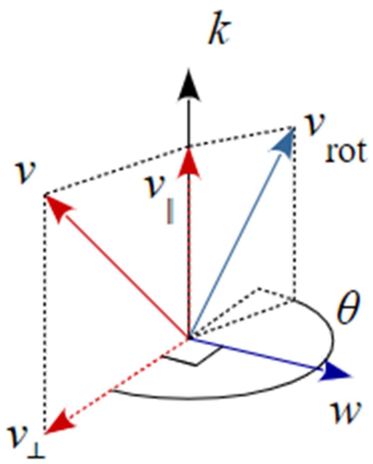
根据矩阵乘法规则与四元数运算法则，可以计算出该结果等于q_1q_2。

同理: $q_1q_2 = q_2^{\oplus}q_1$

本题意义在与将四元数乘法转化为矩阵乘法。

有疑问：$q_1^{+}q_2$ 中q_2是什么形式？ 是矩阵还是四元数？难道可以推出$q_1q_2 = q_2^{\oplus}q_1^{+}$，是否正确？？

四、



证明：

先考虑对一个向量作旋转，其中v是原向量，三维的单位向量$\mathrm{k} = [k_x\ k_y\ k_z]^T$是旋转轴，θ是旋转角度，$v_{rot}$是旋转后的向量。

先通过点积得到 v 在 k 方向的平行分量$v_{\|}$。

$$v_{\|} = (v \cdot k)k$$

在通过叉乘得到与 k 正交的两个向量$v_{\perp}$和w。

$$v_{\perp} = v - v_{\|} = v - (v \cdot k)k = -k \times (k \times v) \cdots\cdots (1)$$

$$w = k \times v$$

这样就得到了 3 个相互正交的向量，如图所示，根据叉乘定义可以得出（此处为难点）：

$$|w| = |v_{\perp}| = |v| \sin\alpha \qquad (\alpha\text{ 为 v 与 k 夹角})$$

继而得出：

$$v_{rot} = v_{\|} + \cos(\theta)\, v_{\perp} + \sin(\theta)\, w \cdots\cdots (2)$$

再引入叉积矩阵的概念；记 K 为向量 $\mathrm{k} = [k_x \quad k_y \quad k_z]^T$ 的叉积矩阵，则 K 为一个反对称矩阵：

$$\mathrm{K} = \begin{bmatrix} 0 & -k_z & k_y \\ k_z & 0 & -k_x \\ -k_y & k_x & 0 \end{bmatrix}$$

其有如下性质：

$$k \times v = Kv$$

为了利用该性质，需要将v_{rot}替换为v与k的叉积关系，先根据(1)式做代换：

$$v_{\|} = v + k \times (k \times v) \quad \cdots\cdots (3)$$

根据(1)(2)(3)得到：

$$v_{rot} = v + k \times (k \times v) - \cos(\theta)\ k \times (k \times v) + \sin(\theta)\ k \times v$$

根据叉积矩阵的性质：

$$v_{rot} = v + (1 - \cos(\theta))K^2 v + \sin(\theta)\, Kv$$

$$v_{rot} = (I + \big((1 - \cos(\theta))K^2 + \sin(\theta)\, K\big)v$$

假设原坐标系基向量矩阵为 B，旋转后的坐标系基向量矩阵为 C，那么：

$$\mathrm{BR} = \mathrm{C}$$

$$\mathrm{B}C^{-1} = R$$

将v，v_{rot}替换为 B、C：

$$\mathrm{B} = \left(I + \left((1 - \cos(\theta))K^2 + \sin(\theta)\,K\right)C\right.$$

同时右乘C^{-1}，即为罗德里格斯公式的标准形式。

$$\mathrm{R} = \mathrm{I} + (1 - \cos(\theta))K^2 + \sin(\theta)\,K$$

证明完毕。

感想：并不复杂，但要清楚理解叉积及叉积矩阵的含义。

五、

1.根据第 4 提结果可得提示中的(6)式，设$\mathrm{q} = (\mathrm{s}, \mathrm{x}, \mathrm{y}, \mathrm{z})$

则q^{-1} = (s, -x, -y, -z) / $(s^2 + x^2 + y^2 + z^2)$

$$q^{+} = \begin{bmatrix} s & -z & y & x \\ z & s & -x & y \\ -y & x & s & z \\ -x & -y & -z & s \end{bmatrix}$$

$$q^{-1\oplus} = \begin{bmatrix} s & -z & y & -x \\ z & s & -x & -y \\ -y & x & s & -z \\ x & y & z & s \end{bmatrix}$$

经计算$q^{+}q^{-1\oplus}p$ 的第一行为 0，即p'实部为 0

2. 由(6)式易得：$\mathrm{Q} = q^{+}q^{-1\oplus}$

六、

1.代码段 17 行，使用了范围 for 循环，实现了类似 foreach 功能，对 vector 容器中每一个 A 对象 a，都打印了 a.index。

2.代码段 17 行，使用了自动类型推导关键词 auto，编译器会根据上下文情况，确定 auto 变量的真正类型，此处类型为 A。

3.代码段 16 行，sort 函数第三个参数为 lambda 表达式，这里定义了可调用对象，接受两个 A 类型变量的常量引用，返回是否第一个的 index 属性值小于第二个，由于函数体只有一个 return 语句且没有指定返回类型，表达式将自动推断返回类型为 bool。

4.代码段 15 行，使用了花括号对容器对象进行列表初始化，更加优雅。

5.代码段 16 行，使用了 vector 对象的容器迭代器 begin()和 end()，用来遍历所有元素。