第二节课习题

崔华坤

二、熟悉 Eigen 矩阵运算

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？

答：系数矩阵 A 非奇异，等价于如下说法：

1) A 可逆
2) A 的行列式 det(A)或|A|不为 0
3) A 的秩 rank(A)等于 n，A 满秩（假设 A 为 n 阶方阵）
4) A 的转置矩阵可逆
5) 存在一个 n 阶方阵 B，使得 $BA=I_n$

2. 高斯消元法的原理是什么？

答：步骤为：

1) 从第 1 列中挑选一个绝对值最大的元素，作为主元，将其所在行换到第 1 行；
2) 然后用第 1 行乘上（$-a_{k1}/a_{11}$），依次加到第 k 行，使第 1 列在对角线以下的元素变为 0，如下所示：

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ & a_{22} & \cdots & a_{2n} & b_2 \\ & & \ddots & \vdots & \vdots \\ & & & a_{nn} & b_n \end{array}\right) \quad \left(\prod_{k=1}^{n} a_{kk} \neq 0\right)$$

3) 对方程组进行回代，从后往前逐个解出 x。
4) 对于 n 阶线性方程组，Gauss 消元法的计算量大约是 $2n^3/3+O(n^2)$

3. QR 分解的原理是什么？

答：QR 分解是一种将矩阵分解的方式，这种方式可以把矩阵分解成一个正交矩阵 Q 和一个上三角矩阵 R 相乘，QR 分解经常用来解线性最小二乘法问题。

正交矩阵：$Q^T=Q^{-1}$

4. Cholesky 分解的原理是什么？

答：Cholesky 分解（楚列斯基分解）是指将一个正定的 Hermite 矩阵分解成一个

下三角矩阵与其共轭转置之乘积。Cholesky 分解在求解线性方程组中的效率约两倍于 LU 分解。

正定：一个实对称矩阵 M 是正定的，当且仅当对于所有的非零实系数向量 z，都有 $z^TMz>0$。M 的所有特征值都是正的。

Hermite 矩阵：又称埃尔米特矩阵，也称自伴随矩阵，是共轭对称的方阵。矩阵中第 i 行第 j 列的元素都与第 j 行第 i 列的元素复共轭。$A=A^H$，如下所示，Hermite 矩阵的主对角线上的元素都是实数，其特征值也是实数。实对称矩阵是 Hermite 矩阵的特例。

$$\begin{bmatrix} 3 & 2+i \\ 2-i & 1 \end{bmatrix}$$

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。

答：

stevencui@ubuntu: ~/Project/SLAMCourse/l2/build

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

#define SIZE 100
int main()
{
cout<<"useEigen..."<<endl;
Eigen::Matrix<double, SIZE, SIZE>  A;
A = Eigen::MatrixXd::Random(SIZE, SIZE);
//A = A.transpose() * A;

Eigen::Matrix<double, SIZE, 1> b;
b = Eigen::MatrixXd::Random(SIZE,1);


//inverse
Eigen::Matrix<double, SIZE, 1> x1;
x1  = A.inverse() * b;
cout<<"Direct Inverse Result:\n"<<x1.matrix().transpose()<<endl;

//QR
Eigen::Matrix<double, SIZE, 1> x2;
x2 = A.fullPivHouseholderQr().solve(b);
cout<<"QR Result:\n"<<x2.matrix().transpose()<<endl;

//Cholesky
Eigen::Matrix<double, SIZE, 1> x3;
x3 = A.ldlt().solve(b);
cout<<"Cholesky LDLT Result:\n"<<x3.matrix().transpose()<<endl;

return 1;
}
```

```
stevencui@ubuntu:~/Project/SLAMCourse/l2/build$ make
Scanning dependencies of target useEigen
[ 50%] Building CXX object CMakeFiles/useEigen.dir/useEigen.cpp.o
./u     [100%] Linking CXX executable useEigen
[100%] Built target useEigen
stevencui@ubuntu:~/Project/SLAMCourse/l2/build$ ./useEigen
useEigen...
Direct Inverse Result:
  -3.20275    2.00917   -1.42388    4.75724   -3.37883   -4.58979   -4.06869
905    -3.28092    2.65787    1.13721   -1.14728    1.81384   -3.01321  -0.6864
1.30139  0.0514718  -0.781168  -0.140511    0.236243   0.528249  -0.636057    1
    -3.9308     1.9129   -0.585467    3.13969    0.47735   -2.50282   -1.59472
2112     1.17004    0.506682    1.68861    0.900261    3.51295   -2.74951    3.17
 2.09269    -0.90607   -1.77557    0.582746     2.3128 -0.0564984   -1.32888
QR Result:
  -3.20275    2.00917   -1.42388    4.75724   -3.37883   -4.58979   -4.06869
905    -3.28092    2.65787    1.13721   -1.14728    1.81384   -3.01321  -0.6864
1.30139  0.0514718  -0.781168  -0.140511    0.236243   0.528249  -0.636057    1
    -3.9308     1.9129   -0.585467    3.13969    0.47735   -2.50282   -1.59472
2112     1.17004    0.506682    1.68861    0.900261    3.51295   -2.74951    3.17
 2.09269    -0.90607   -1.77557    0.582746     2.3128 -0.0564984   -1.32888
Cholesky LDLT Result:
  -2.06358    1.33299    2.75057   -3.05575    1.71225     1.3388    1.95178
344    0.978025    2.35266     1.6936     1.71308   -2.45652    0.286457    1.546
1.20645  0.0560538  -0.949366   -1.40453    0.22717    1.04404    1.73222   -1
  -0.197127   0.381682    1.30146   -1.50703    0.109369   -0.96286  -0.315061
4817     2.87572   -2.87962  -0.493882     1.49178     1.60802  -0.261867   -1.05
-3.15716   -1.32534    3.11585   -3.79036    1.53567   0.566368   0.402963
stevencui@ubuntu:~/Project/SLAMCourse/l2/build$
```

三、几何运算练习

```
stevencui@ubuntu: ~/Project/SLAMCourse/l2/useGeometry/build
#include <iostream>
#include <Eigen/Core>
#include <Eigen/Geometry>

using namespace std;
using namespace Eigen;

int main()
{
cout<<"START..."<<endl;
Isometry3d T_C1_W = Isometry3d::Identity();
Quaterniond q_C1_W(0.55,0.3,0.2,0.2);
cout<<"q_C1_W:"<<q_C1_W.coeffs().transpose()<<endl;
T_C1_W.rotate(q_C1_W.normalized().toRotationMatrix());
T_C1_W.pretranslate(Vector3d(0.7,1.1,0.2));
cout<<"T_C1_W:\n"<<T_C1_W.matrix()<<endl;

Isometry3d T_C2_W = Isometry3d::Identity();
Quaterniond q_C2_W(-0.1,0.3,-0.7,0.2);
T_C2_W.rotate(q_C2_W.normalized().toRotationMatrix());
T_C2_W.pretranslate(Vector3d(-0.1,0.4,0.8));

Vector3d p_C1(0.5,-0.1,0.2);
Vector3d p_C2 = T_C2_W * T_C1_W.inverse() * p_C1;
cout<<"p_C2: "<<p_C2.transpose()<<endl;

return 1;

}
```

```
stevencui@ubuntu:~/Project/SLAMCourse/l2/useGeometry/build$ make
Scanning dependencies of target useGeometry
[ 50%] Building CXX object CMakeFiles/useGeometry.dir/useGeometry.cpp.o
[100%] Linking CXX executable useGeometry
[100%] Built target useGeometry
stevencui@ubuntu:~/Project/SLAMCourse/l2/useGeometry/build$ ./useGeometry
START...
q_C1_W: 0.3  0.2  0.2 0.55
T_C1_W:
 0.661376  -0.21164  0.719577       0.7
 0.719577  0.449735 -0.529101       1.1
 -0.21164  0.867725  0.449735       0.2
        0         0         0         1
p_C2:  1.08228 0.663509 0.686957
```

四、旋转的表达

1. 设有旋转矩阵 R，证明 $R^TR = I$ 且 $detR = +1$。

答：设两个坐标系的坐标基分别为$[e_1,e_2,e_3]^T$，$[e_1',e_2',e_3']^T$，推导如下：

$$\mathrm{R} = \begin{bmatrix} e_1^T \\ e_2^T \\ e_3^T \end{bmatrix} \begin{bmatrix} e_1' & e_2' & e_3' \end{bmatrix} \tag{1}$$

$$R^TR = \begin{bmatrix} e_1{'}^T \\ e_2{'}^T \\ e_3{'}^T \end{bmatrix} \begin{bmatrix} e_1 & e_2 & e_3 \end{bmatrix} \begin{bmatrix} e_1^T \\ e_2^T \\ e_3^T \end{bmatrix} \begin{bmatrix} e_1' & e_2' & e_3' \end{bmatrix} \tag{2}$$

下面分析中间部分：

$$\begin{bmatrix} e_1 & e_2 & e_3 \end{bmatrix} \begin{bmatrix} e_1^T \\ e_2^T \\ e_3^T \end{bmatrix} \tag{3}$$

为了方便计算，这里以二维为例，令：

$$e_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, e_2 = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$$

则根据单位坐标基性质，有以下关系满足：

$$\begin{aligned} |e_1|^2 &= x_1^2 + y_1^2 = 1 \\ |e_2|^2 &= x_2^2 + y_2^2 = 1 \\ e_1 \cdot e_2 &= x_1x_2 + y_1y_2 = 0 \end{aligned} \tag{4}$$

将式（3）展开可得：

$$\begin{bmatrix} e_1 & e_2 \end{bmatrix} \begin{bmatrix} e_1^T \\ e_2^T \end{bmatrix} = \begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix} = \begin{bmatrix} x_1^2 + x_2^2 & x_1y_1 + x_2y_2 \\ x_1y_1 + x_2y_2 & y_1^2 + y_2^2 \end{bmatrix} \tag{5}$$

由式(4)可得：

$$x_1x_2 + y_1y_2 = 0$$
$$\frac{x_1}{y_1} = -\frac{y_2}{x_2}$$
$$\frac{y_1^2 + x_1^2}{y_1^2} = \frac{x_2^2 + y_2^2}{x_2^2}$$
$$y_1^2 = x_2^2$$

同理可得：$y_2^2 = x_1^2$

由式(4)亦可得：

$$\begin{aligned}(x_1x_2 + y_1y_2)^2 = x_1^2x_2^2 + y_1^2y_2^2 + 2x_1x_2y_1y_2 = 0 \\ = x_1^2y_1^2 + x_2^2y_2^2 + 2x_1x_2y_1y_2 \\ = (x_1y_1 + x_2y_2)^2 = 0 \\ \Leftrightarrow x_1y_1 + x_2y_2 = 0\end{aligned}$$

故，式(5)可化简为：

$$[e_1 \quad e_2]\begin{bmatrix} e_1^T \\ e_2^T \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I$$

最后有：

$$R^TR = \begin{bmatrix} {e_1}'^T \\ {e_2}'^T \\ {e_3}'^T \end{bmatrix}[e_1 \quad e_2 \quad e_3]\begin{bmatrix} e_1^T \\ e_2^T \\ e_3^T \end{bmatrix}[e_1' \quad e_2' \quad e_3'] = \begin{bmatrix} {e_1}'^T \\ {e_2}'^T \\ {e_3}'^T \end{bmatrix}[e_1' \quad e_2' \quad e_3'] = I$$

行列式证明如下：

$$|R^TR| = |R^T||R| = |R||R| = 1$$

2. **答：四元数的虚部包含三个分量，$\boldsymbol{\varepsilon} = \left(\boldsymbol{n}_x\sin\frac{\boldsymbol{\theta}}{2}, \boldsymbol{n}_y\sin\frac{\boldsymbol{\theta}}{2}, \boldsymbol{n}_z\sin\frac{\boldsymbol{\theta}}{2}\right)$，实部包含一个分量：**$\eta = \cos\frac{\theta}{2}$

3. 证明：

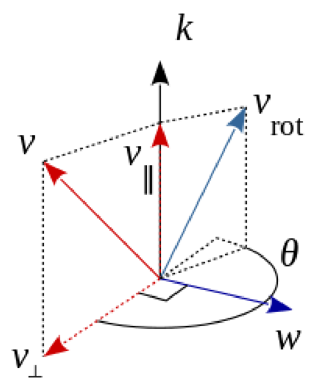
设$q_1 = \begin{bmatrix} \varepsilon_1 \\ \eta_1 \end{bmatrix}$，$q_2 = \begin{bmatrix} \varepsilon_2 \\ \eta_2 \end{bmatrix}$，则根据四元数乘法运算可得：

$$q_1q_2 = \begin{bmatrix} \varepsilon_1\eta_2 + \varepsilon_2\eta_1 + \varepsilon_1^\times\varepsilon_2 \\ \eta_1\eta_2 - \varepsilon_1^T\varepsilon_2 \end{bmatrix} = \begin{bmatrix} \eta_1 I + \varepsilon_1^\times & \varepsilon_1 \\ -\varepsilon_1^T & \eta_1 \end{bmatrix}\begin{bmatrix} \varepsilon_2 \\ \eta_2 \end{bmatrix} = q_1^\oplus q_2$$

亦有：

$$q_1q_2 = \begin{bmatrix} \varepsilon_1\eta_2 + \varepsilon_2\eta_1 - \varepsilon_2^\times\varepsilon_1 \\ \eta_1\eta_2 - \varepsilon_1^T\varepsilon_2 \end{bmatrix} = \begin{bmatrix} \eta_2 I - \varepsilon_2^\times & \varepsilon_2 \\ -\varepsilon_2^T & \eta_2 \end{bmatrix}\begin{bmatrix} \varepsilon_1 \\ \eta_1 \end{bmatrix} = q_2^+ q_1$$

五 罗德里格斯公式的证明



旋转前，根据矩阵投影公式可得：

$$v_{\parallel} = kk^T v$$

$$v_{\perp} = v - v_{\parallel} = (1 - kk^T)v$$

则旋转后矢量为：

$$\begin{aligned} v_{rot} &= v_{\parallel rot} + v_{\perp rot} = v_{\parallel} + (\cos\theta\, v_{\perp} + \sin\theta\, k \times v) \\ &= kk^T v + \cos\theta\,(1 - kk^T)v + \sin\theta\, k \times v \\ &= [\cos\theta + (1 - \cos\theta)kk^T + \sin\theta\, k^\wedge]v \end{aligned}$$

则旋转矩阵为：

$$R = \cos\theta + (1 - \cos\theta)kk^T + \sin\theta\, k^\wedge$$

6 四元数运算性质的验证

答：

设：

$$\mathrm{q} = [\eta \quad \varepsilon]$$

$$\mathrm{p} = [0 \quad \zeta]$$

则根据四元数乘法规则，可得为：

$$\mathrm{qp} = [-\varepsilon^T\zeta \quad \eta\zeta + \varepsilon^\wedge\zeta]$$

又有：

$$\mathrm{q}^{-1} = [\eta \quad -\varepsilon]$$

则qpq^{-1}的实部可写成：

$$\begin{aligned} &-\varepsilon^T\zeta\eta + (\eta\zeta + \varepsilon^\wedge\zeta)^T\varepsilon \\ &= -\varepsilon^T\zeta\eta + \eta\zeta^T\varepsilon + (\varepsilon^\wedge\zeta)^T\varepsilon \\ &= 0 \end{aligned}$$

qpq^{-1}的虚部可写成：

$$
\begin{aligned}
&\varepsilon\varepsilon^T\zeta + \eta(\eta\zeta + \varepsilon^\wedge\zeta) - (\eta\zeta + \varepsilon^\wedge\zeta)^\wedge\varepsilon \\
&= \varepsilon\varepsilon^T\zeta + \eta(\eta\zeta + \varepsilon^\wedge\zeta) + \varepsilon^\wedge(\eta\zeta + \varepsilon^\wedge\zeta) \\
&= (\varepsilon\varepsilon^T + \eta^2 + 2\eta\varepsilon^\wedge + \varepsilon^\wedge\varepsilon^\wedge)\zeta
\end{aligned}
$$

则 R 可写成：

$$
R = \begin{bmatrix} 0 & 0_{1\times3} \\ 0_{3\times1} & \varepsilon\varepsilon^T + \eta^2 + 2\eta\varepsilon^\wedge + \varepsilon^\wedge\varepsilon^\wedge \end{bmatrix}
$$

其中左下角变为罗德里格斯公式的三维形式。

7 * 熟悉 C++11

答：

lambda 函数

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

class A {
public:
    A(const int& i ) : index(i) {}
    int index = 0;
};

int main() {
    A a1(3), a2(5), a3(9);
    vector<A> avec{a1, a2, a3};
    std::sort(avec.begin(), avec.end(), [](const A&a1, const A&a2) {return a1.index<a2.index;});
    for ( auto& a: avec ) cout<<a.index<<" ";
    cout<<endl;
    return 0;
}
```

for 区间迭代

auto 自动推导类型