第二次作业

2 熟悉Eigen

1. Ax = b 在|a| != 0 的情况下有唯一解， 若|A| = 0, r(A) = r(A|b) 有无穷多组解，r(a) != r(A|b)方程组无解。
2. 高斯消元：若用初等行变换将增广矩阵 化为 ，则AX = B与CX = D是同解方程组。
   所以我们可以用初等行变换把增广矩阵转换为行阶梯阵，然后回代求出方程的解。
3. QR法：首先说明一下 QR分解法 A=QR，Q为一个正交矩阵，R为一个上三角矩阵，QR分解法能用来求解Ax=b问题的关键是在于，正交矩阵乘上另外一个任意矩阵不会改变矩阵的欧几里得范数(当然矩阵的大小要匹配)求解上面的方程等同于： min(||Ax-b||) || ||代表范数 , 这里是欧几里得范数.很好理解，让每个方程的误差的平方和最小。 ||Ax-b|| 给他里面乘上一个 正交矩阵 Q'，这里用到了正交矩阵的性质，所以不改变误差的大小||Q'Ax-Q'b|| 注意到A可以分解为QR， 所以转化为 || Rx-Q'b||，为了简便说明起见，我把一个固定的误差省略， 变成求解 min||Rx-Q'b|| ,平方和最小是0，R有是一个上三角矩阵 所以可以很方便的求解Rx=Q'b，利用回带就可以了， QR的分解的优点是具有数值的稳定性。
4. Cholesky 分解是把一个对称正定的矩阵表示成一个下三角矩阵L和其转置的乘积的分解。它要求矩阵的所有特征值必须大于零，故分解的下三角的对角元也是大于零的。Cholesky分解法又称平方根法，是当A为实对称正定矩阵时，LU三角分解法的变形。
5.

```
int main()
{

    Eigen::MatrixXd matrix_A;
    matrix_A = Eigen::MatrixXd::Random( 10, 10 );
    cout<<"matrix_A:\n"<<matrix_A<<endl;
    Eigen::MatrixXd matrix_b;
    matrix_b = Eigen::MatrixXd::Random( 10, 1 );
    cout<<"matrix_b:\n"<<matrix_b<<endl;
    Eigen::MatrixXd x;
    /******************时间比较*********************/
    clock_t time_stt = clock();
    x = matrix_A.inverse()*matrix_b;;  //利用求逆来解方程
    cout <<"time use  in normal inverse  is " <<1000*  (clock() - time_stt)/(double)CLOCKS_PER_SEC <<"ms" << endl;

    time_stt = clock();
    x = matrix_A.colPivHouseholderQr().solve(matrix_b);  //利用QR分解求解方程
    cout <<"time use in Qr decomposition is " <<1000*  (clock() - time_stt)/(double)CLOCKS_PER_SEC <<"ms" << endl;

    time_stt = clock();
    x = matrix_A.fullPivLu().solve(matrix_b);   //LU
    cout <<"time use  in fullPivLu  is " <<1000*  (clock() - time_stt)/(double)CLOCKS_PER_SEC <<"ms" << endl;

    time_stt = clock();
    x = matrix_A.llt().solve(matrix_b);   //llt Cholesky来解方程
    cout <<"time use  in llt(Cholesky)  is " <<1000*  (clock() - time_stt)/(double)CLOCKS_PER_SEC <<"ms" << endl;

    time_stt = clock();
    x = matrix_A.ldlt().solve(matrix_b);   //ldlt
    cout <<"time use  in ldlt is " <<1000*  (clock() - time_stt)/(double)CLOCKS_PER_SEC <<"ms"<< endl;

    return 0;
}
```

3 几何运算练习

```cpp
int main()
{
    //q1 = [0:35; 0:2; 0:3; 0:1] 第一项为实部
    Eigen::Quaterniond q1(0.35, 0.2, 0.3, 0.1);
    Eigen::Vector3d t1(0.3, 0.1, 0.1);
    cout<< "q1.coeffs \n"<<q1.coeffs()<<endl; //利用coffs输出，实部在最后一位
    Eigen::Quaterniond q2(-0.5, 0.4, -0.1, 0.2);
    Eigen::Vector3d t2(-0.1, 0.5, 0.3);
    Eigen::Vector3d p1(0.5, 0, 0.2);
    //归一化
    q1 = q1.normalized();
    cout<<"归一化 q1.coeffs() :\n"<<q1.coeffs()<<endl;
    cout<<"q1.inverse :\n"<<q1.inverse().coeffs()<<endl;
    q2 = q2.normalized();
    //way1
    // v *q1 + t1 = p1
    Eigen::Vector3d w = q1.inverse() * (p1 - t1);
    // v *q2 + t2 = p2
    Eigen::Vector3d p2 = q2 * w + t2;
    cout << "v2 = \n" << p2 << endl;
    //way2
    Eigen::Matrix3d R1 = Eigen::Matrix3d(q1); //转换为3*3旋转矩阵
    Eigen::Matrix3d R2 = Eigen::Matrix3d(q2);
    Eigen::Vector3d v_2 = R1.inverse()*(p1 - t1);
    Eigen::Vector3d v_2_2 = R2 * v_2 + t2;
    cout << "way2 v2= " << endl << v_2_2 << endl;
    return 0;
}
```

4 罗德里格斯公式的证明

设n为旋转轴向量，对任意一个绕n旋转θ后后的向量 v_{rot}，v 可以分解为两个平行于n和垂直于n的向量，v_n 和 $v_\perp$，有 $v = v_\parallel + v_\perp$

即 $v_\parallel = (v\cdot n)\cdot n$

$$v_\perp = v - v_\parallel = v - (v\cdot k)\cdot k = -k\times(k\times v)$$

$$v_\parallel^{rot} = v_\parallel$$

$$v_\perp^{rot} = \cos\theta\cdot v_\perp + \sin\theta\cdot(n\times v_\perp) = \cos\theta\cdot v_\perp + \sin\theta\cdot(n\times v)$$

$$\begin{aligned} v^{rot} &= v_n^{rot} + v_\perp^{rot} \\ &= v_\parallel + \cos\theta\cdot v_\perp + \sin\theta\cdot(n\times v) \\ &= v_n + \cos\theta\cdot(v - v_n) + \sin\theta(n\times v) \\ &= \cos\theta\cdot v + (1-\cos\theta)\cdot v_\parallel + \sin\theta\cdot(n\times v) \end{aligned}$$

即 $R\cdot v = \cos\theta\cdot v + (1-\cos\theta)\cdot n\cdot(n\cdot v) + \sin\theta\cdot(\hat{n}\cdot v)$

得到 $R = \cos\theta\cdot I + (I-\cos\theta)nn^T + \sin\theta + \hat{n}$

5 四元数运算性质的验证

假设 $p=[0, a, b, c]^T$ $q=[q_0, q_1, q_2, q_3]^T$. $p'=[0, a', b', c']^T$

$p' = q\cdot p\cdot q'$.

对于旋转矩阵有 $[a'\ b'\ c']^T = R\cdot[a, b, c]^T$

可知 $\begin{bmatrix} 1 & 0^T \\ 0 & R \end{bmatrix}\cdot p = p'$

则 $Q = \begin{bmatrix} 1 & 0^T \\ 0 & R \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1-2q_2^2-2q_3^2 & 2q_1q_2-2q_0q_3 & 2q_1q_3+2q_0q_2 \\ 0 & 2q_1q_2+2q_0q_3 & 1-2q_1^2-2q_3^2 & 2q_2q_3-2q_0q_1 \\ 0 & 2q_1q_3-2q_0q_2 & 2q_2q_3+2q_0q_1 & 1-2q_1^2-2q_2^2 \end{bmatrix}$

6 熟悉 C++11

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
class A
{
public:
    A(const int& i ) : index(i) {} //声明时实现构造
    int index = 0; //在声明成员变量时赋值
};
int main()
{
    A a1(3), a2(5), a3(9);
    vector<A> avec{a1, a2, a3}; // vector 声明元素

    std::sort(
        avec.begin(), avec.end(),
        [](const A&a1, const A&a2) {return a1.index<a2.index;});///使用lambda表达式
    for ( auto& a: avec ) //使用auto 数据类型
    cout<<a.index<<" ";
    cout<<endl;
    return 0;
}
```