1. 验证旋转矩阵是正交矩阵。

，展开范数的平方：

，代入得：

=,对任意向量都成立，所以：

.

1. \* 寻找罗德里格斯公式的推导过程并加以理解。

向量沿单位旋转轴向量旋转角度后得到的旋转向量的罗德里格斯公式：

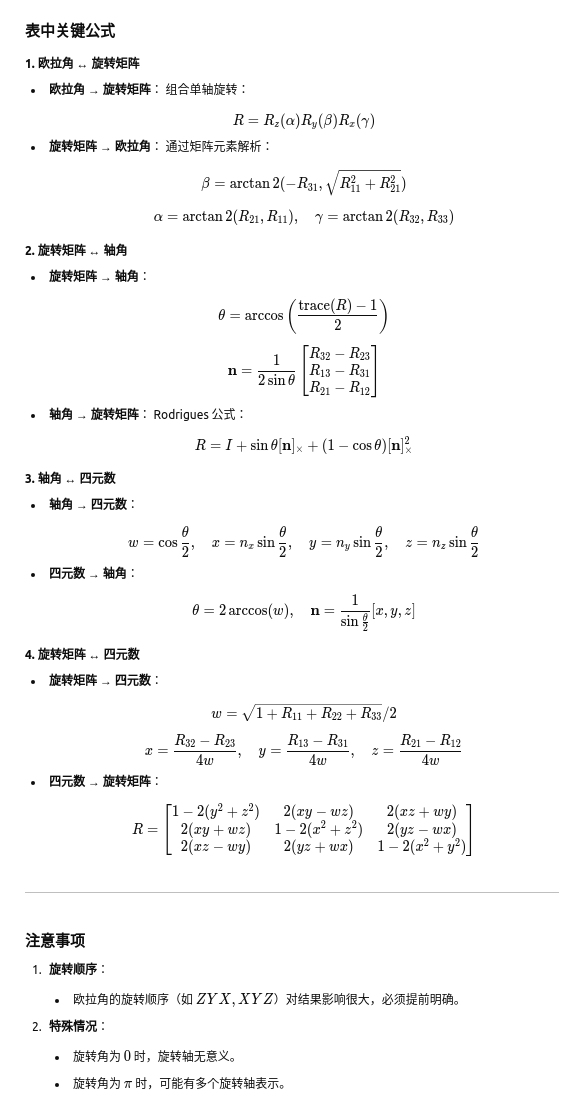
，推导：

由...,

...

1. 验证四元数旋转某个点后，结果是一个虚四元数（实部为零），所以仍然对应到一个三维空间点，见式（3.33）。
2. 画表总结旋转矩阵、轴角、欧拉角、四元数的转换关系。





1. 假设有一个大的Eigen矩阵，想把它的左上角3✖3的块取出来，然后赋值为。请编程实现。

#include <iostream>  
#include <Eigen/Dense>  
  
using namespace std;  
using namespace Eigen;  
  
#define MATRIX\_SIZE 5  
  
int main(int argc, char \*\*argv) {  
 Matrix<double, MATRIX\_SIZE, MATRIX\_SIZE> bigMatrix  
 = MatrixXd::*Random*(MATRIX\_SIZE, MATRIX\_SIZE);  
  
 cout << "The big matrix: \n" << bigMatrix << endl;  
  
 Matrix3d extractedBlock = bigMatrix.block<3,3>(0,0);  
  
 cout << "The extracted matrix block: \n" << extractedBlock << endl;  
  
 extractedBlock.setIdentity();  
  
 cout << "The assigned matrix block: \n" << extractedBlock << endl;  
  
 return 0;  
}

1. \* 一般线性方程有几种做法？你能在Eigen中实现吗？

# include <iostream>  
# include <Eigen/Dense>  
# include <Eigen/Core>  
#include <random> // 使用 C++11 随机数库  
  
  
using namespace std;  
using namespace Eigen;  
  
// 自定义高斯消元法函数  
VectorXd gaussianElimination(const MatrixXd& A, const VectorXd& b) {  
 const Index n = A.rows(); // 矩阵的行数（即方程个数）  
  
 // 将 A 和 b 合并为增广矩阵 [A | b]  
 MatrixXd augmented(n, n + 1);  
 augmented << A, b;  
  
 // 消元阶段：将矩阵 A 转化为上三角矩阵  
 for (Index k = 0; k < n; ++k) {  
 // 1. 选主元：确保当前对角线元素非零  
 for (Index i = k + 1; i < n; ++i) {  
 if (augmented(k, k) == 0) {  
 cerr << "Zero pivot encountered. Matrix is singular!" << endl;  
 exit(EXIT\_FAILURE);  
 }  
 // 2. 消去第 k 列的第 i 行元素  
 const double factor = augmented(i, k) / augmented(k, k);  
 for (Index j = k; j <= n; ++j) {  
 augmented(i, j) -= factor \* augmented(k, j);  
 }  
 }  
 }  
  
 // 回代阶段：从最后一行开始，求解未知数  
 VectorXd x(n);  
 for (Index i = n - 1; i >= 0; --i) {  
 x(i) = augmented(i, n);  
 for (Index j = i + 1; j < n; ++j) {  
 x(i) -= augmented(i, j) \* x(j);  
 }  
 x(i) /= augmented(i, i);  
 }  
  
 return x;  
}  
  
int main() {  
 cout.precision(3); // 限制精确度  
  
 std::random\_device rd; // 获取一个高质量的随机种子  
 std::default\_random\_engine generator(rd()); // 初始化随机数生成器  
 std::uniform\_real\_distribution<double> distribution(-1.0, 1.0); // 均匀分布 [-1, 1]  
  
 // 初始化 Eigen 矩阵  
 MatrixXd B1(3, 3); // 生成一个 rows x cols 的矩阵  
  
 for (int i = 0; i < 3; ++i) {  
 for (int j = 0; j < 3; ++j) {  
 B1(i, j) = distribution(generator); // 填充每个元素  
 }  
 }  
  
 EigenSolver<Matrix3d> const solver1(B1);  
 MatrixXd A1 = B1.transpose() \* B1 + 0.1 \* MatrixXd::*Identity*(3, 3);  
  
 const VectorXcd eigenValueA = solver1.eigenvalues();  
  
 cout << "eigenValue of A = \n" << eigenValueA << endl;  
  
 const VectorXd b1 = VectorXd::*Random*(3,1);  
  
 // 1.1 调用自定义高斯消元法（Gaussian Elimination）函数  
 const VectorXd gex = gaussianElimination(A1, b1);  
  
 // 输出解  
 cout << "Solution of Gaussian Elimination: x = \n" << gex << endl;  
  
 // 1.2 使用Eigen解决Ax=b  
 const VectorXd ex = A1.partialPivLu().solve(b1);  
 cout << "Solution of Eigen: x = \n" << ex << endl;  
  
 // 2. 使用LU分解法求解线性方程Ax=b  
 const Matrix<double, 3, 1> lux = A1.lu().solve(b1); // A.lu().solve(b) 的 lu() 分解适用于方阵（n×n）  
  
 cout << "Solution of lu decomposition: x = \n" << lux << endl;  
  
 // 3. 使用LLT分解法求解线性方程Ax=b，要求A是对称正定矩阵  
 LLT<Matrix<double, 3, 3>> const llt(A1); // LLT 分解  
 if (llt.info() == Success) {  
 const Matrix<double, 3, 1> lltx = llt.solve(b1);  
 cout << "Solution of LLT decomposition: x = \n" << lltx << endl;  
 } else {  
 cout << "Matrix A is not positive definite!" << endl;  
 }  
  
 // 4. 使用QR分解法  
 // QR 分解  
 HouseholderQR<MatrixXd> const qr(A1);  
  
 // 求解 Ax = b  
 VectorXd qrx = qr.solve(b1);  
  
 // 获取 Q 和 R  
 MatrixXd Q = qr.householderQ();  
 MatrixXd R = qr.matrixQR().triangularView<Upper>();  
  
 cout << "Solution of QR decomposition: x = \n" << qrx << endl;  
 cout << "Q = \n" << Q << "\nR = \n" << R << endl;  
  
 // 5. 奇异值分解 (SVD) 求解  
 // 对矩阵 A 进行奇异值分解  
 JacobiSVD<MatrixXd> const svd(A1, ComputeThinU | ComputeThinV);  
  
 // 使用 SVD 求解 Ax = b  
 const VectorXd svdx = svd.solve(b1);  
  
 cout << "Solution of SVD: x = \n" << svdx << endl;  
  
 // 输出奇异值  
 cout << "Singular values of A:\n" << svd.singularValues() << endl;  
  
 // 使用特征值分解求解  
 // 使用 EigenSolver 进行特征值分解  
 EigenSolver<Matrix3d> solver2(A1);  
  
 // 获取特征向量矩阵 V 和特征值对角矩阵 Lambda  
 Matrix3d V = solver2.eigenvectors().real(); // 提取实部特征向量  
 Vector3d Lambda = solver2.eigenvalues().real(); // 提取实部特征值  
  
 // 计算中间变量 y = V.inverse() \* b  
 Vector3d y = V.inverse() \* b1;  
  
 // 通过 Lambda 求解 y，并还原 x = V \* y  
 for (int i = 0; i < 3; ++i) {  
 if (Lambda(i) != 0) { // 避免除以 0  
 y(i) /= Lambda(i);  
 } else {  
 cerr << "Error: Zero eigenvalue encountered!" << endl;  
 return -1;  
 }  
 }  
  
 Vector3d evdx = V \* y;  
  
 cout << "Solution of Eigen Value Decomposition: x = \n" << evdx << endl;  
  
 return 0;  
}