M.Sc. (Five Year Integrated) in Computer Science
(Artificial Intelligence & Data Science)

Third Semester

21-805-0303 MATHEMATICS FOR MACHINE LEARNING

Submitted in partial fulfillment
of the requirements for the award of degree in
Master of Science (Five Year Integrated)
in Computer Science (Artificial Intelligence & Data Science) of
Cochin University of Science and Technology (CUSAT)
Kochi

Submitted by

LANA ANVAR
(80522012)

DEPARTMENT OF COMPUTER SCIENCE
COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY (CUSAT)
KOCHI-682022

JANUARY 2024

Contents

ABSTRACT

This C++ program focuses on the implementation of a Matrix Decomposer class using the Eigen library to decompose symmetric positive definite (SPD) matrices. The program employs three distinct decomposition techniques: QR decomposition, LU decomposition, and Cholesky decomposition. The Matrix Decomposer class includes methods for generating SPD matrices and measuring the average execution time of each decomposition technique over multiple iterations.

In the main function, the user is prompted to input the size of the matrix, and an instance of the Matrix Decomposer is created accordingly. The program then proceeds to perform the matrix decomposition, providing insights into the average computational time for QR, LU, and Cholesky methods. The output also includes ratios that highlight the comparative efficiency of these decomposition techniques.

This project serves as a valuable tool for understanding and comparing the performance of different matrix decomposition methods, offering insights into their computational complexities and practical applications. The Eigen library's capabilities enhance the program's efficiency, making it a useful resource for students and researchers exploring linear algebra and numerical methods.

QUESTION 1 :

A bank issues credit cards to customers.Based on the past data, the bank has found out that 75is selected at random from the current database, construct the Binomial Probability Distribution of accounts paying on time. Plot the distribution of n clients paying on time. Hint We have p=0.75 (pays on time), n=25 individuals, We need to calculate P(X=x) when x=0,1,2,3,4,5,. . . ,25 Try to change the number of individuals, what do you observe?

2. QR Decomposition:

QR decomposition is a method to break down a matrix A into two simpler matrices: an orthogonal matrix Q and an upper triangular matrix R

$$A = QR$$

To find Q and R, we use the Gram-Schmidt Process. Q, a square matrix of dimensions $m \times m$, contains orthonormal columns, while R, an upper triangular matrix of size $n \times n$, captures the transformed relationships between the original columns of A. Post-decomposition, we obtain Q and R such that $A = QR$.
To find the unknowns in the system, represented by vector x, we solve $Rx = Q^T b$, simplifying the process of obtaining solutions through QR decomposition

3. Cholesky Decomposition:

Cholesky decomposition is a specialized method tailored for symmetric positive definite matrices. This technique transforms a matrix A into the product of a lower triangular matrix L and its conjugate transpose ($A = LL^T$). Specifically designed for symmetric positive definite matrices of size $n \times n$, Cholesky decomposition seeks to find the lower triangular matrix L

We achieve the values for L by the following equations:

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2} \quad \text{for} \quad i = 1, 2, ..., n$$

$$l_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} l_{jk} l_{ik}}{l_{jj}} \quad \text{for} \quad i > j$$

$$l_{ij} = 0 \quad \text{for} \quad i < j$$

and then gets the value for x by computing the following.

$$L^T x = y$$
$$Ly = b$$

DECOMPOSITION TIME ANALYSIS

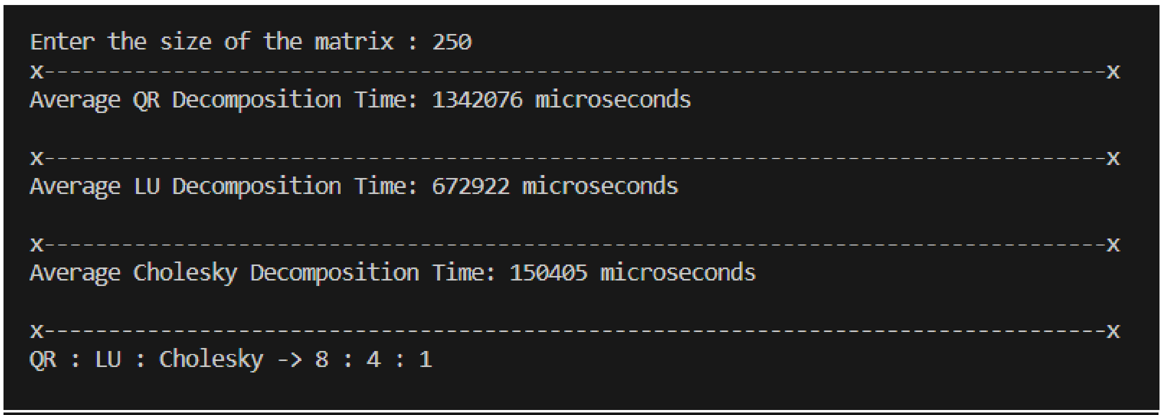
```
#include <iostream>
#include <Eigen/Dense>
#include <chrono>

using namespace Eigen;
using namespace std;
using namespace chrono;

class MatrixDecomposer
{

protected:
    MatrixXd symmetricMatrix;
    MatrixXd Q, R, L, U, L_cholesky;
    int size;

    void qrDecomposition()
    {
        HouseholderQR<MatrixXd> qr(symmetricMatrix);
        Q = qr.householderQ();
        R = qr.matrixQR().triangularView<Upper>();
    }

    void luDecomposition()
    {
        FullPivLU<MatrixXd> lu(symmetricMatrix);
        L = lu.matrixLU().triangularView<Lower>();
        U = lu.matrixLU().triangularView<Upper>();
    }

    void choleskyDecomposition()
    {
        LLT<MatrixXd> llt(symmetricMatrix);
        L_cholesky = llt.matrixL();
    }

public:
```

```
const MatrixXd &getSymmetricMatrix() const
{
    return symmetricMatrix;
}
MatrixDecomposer(int size) : size(size)
{
    generateSPDMatrix();
}

void generateSPDMatrix()
{
    MatrixXd L = MatrixXd::Random(size, size).array().abs();
    symmetricMatrix = L * L.transpose();
}

void decomposeAndMeasureTime()
{
    const int numIterations = 100;

    auto start = high_resolution_clock::now();
    for (int i = 0; i < numIterations; ++i)
    {
        qrDecomposition();
    }
    auto stop = high_resolution_clock::now();
    auto qr_duration = duration_cast<microseconds>(stop - start)
    / numIterations;
    cout << "x--------------------------------------------------x"
         << "\n";
    cout << "Average QR Decomposition Time: "
    << qr_duration.count() << " microseconds\n\n";

    start = high_resolution_clock::now();
    for (int i = 0; i < numIterations; ++i)
    {
        luDecomposition();
    }
    stop = high_resolution_clock::now();
```

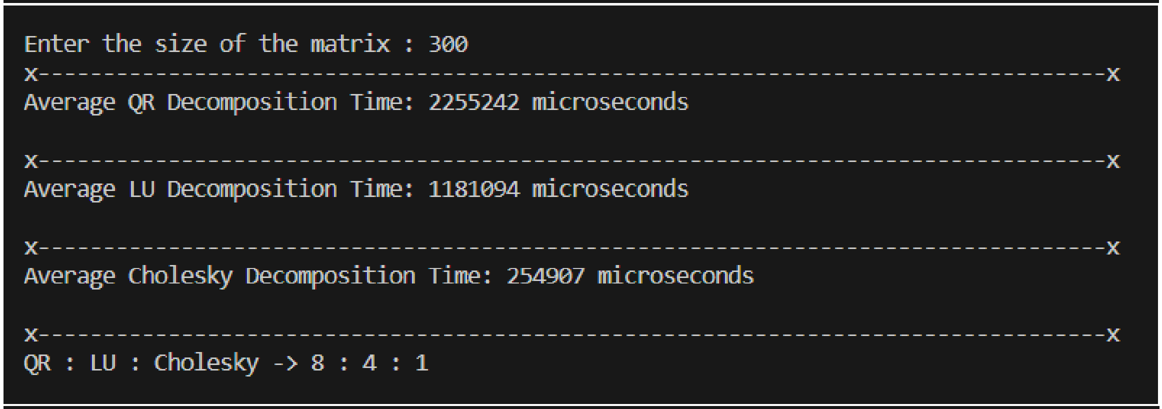
```
        auto lu_duration = duration_cast<microseconds>(stop - start)
        / numIterations;
         cout << "x--------------------------------------------------x"
             << "\n";
        cout << "Average LU Decomposition Time: "
        << lu_duration.count() << " microseconds\n\n";

        start = high_resolution_clock::now();
        for (int i = 0; i < numIterations; ++i)
        {
            choleskyDecomposition();
        }
        stop = high_resolution_clock::now();
        auto cholesky_duration = duration_cast<microseconds>
        (stop - start)
        / numIterations;
         cout << "x--------------------------------------------------x"
             << "\n";
        cout << "Average Cholesky Decomposition Time: " <<
        cholesky_duration.count() << " microseconds\n\n";

        double qr_cholesky_ratio = static_cast<double>
        (qr_duration.count()) / cholesky_duration.count();
        double lu_cholesky_ratio = static_cast<double>
        (lu_duration.count()) / cholesky_duration.count();
        double cholesky_cholesky_ratio = static_cast<double>
        (cholesky_duration.count()) / cholesky_duration.count();

         cout << "x--------------------------------------------------x"
             << "\n";

        cout << "QR : LU : Cholesky -> " << qr_cholesky_ratio
        << " : " << lu_cholesky_ratio << " : "
        << cholesky_cholesky_ratio
        << endl;
    }
};
```

```
int main()
{
    int matrixSize;
    cout << "\nEnter the size of the matrix : ";
    cin >> matrixSize;

    MatrixDecomposer decomposer(matrixSize);

    decomposer.decomposeAndMeasureTime();

    cout << "\n\n";

    return 0;
}
```

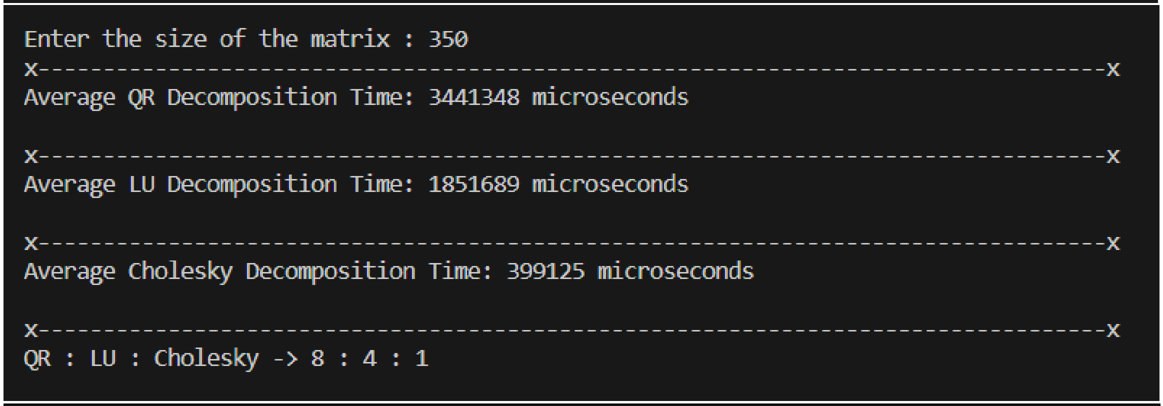
RESULT



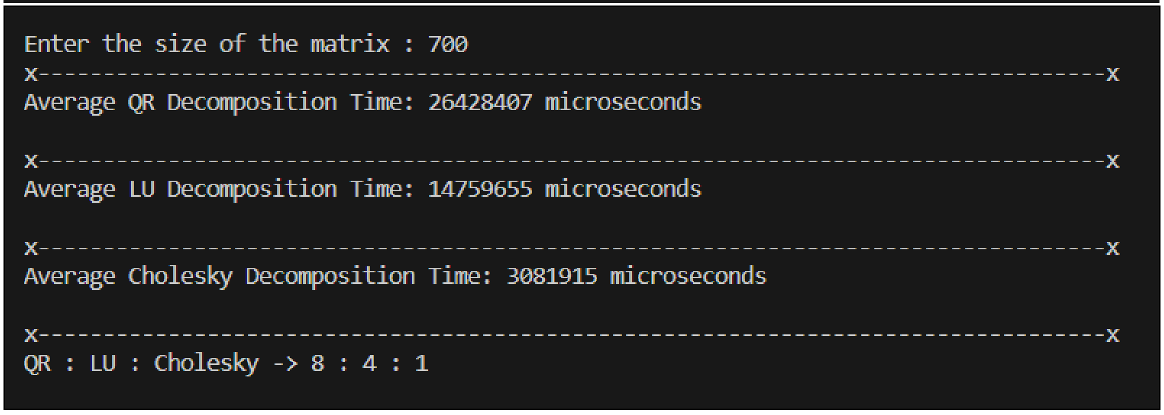
```
Enter the size of the matrix : 250
x---------------------------------------------------------------------------------x
Average QR Decomposition Time: 1342076 microseconds

x---------------------------------------------------------------------------------x
Average LU Decomposition Time: 672922 microseconds

x---------------------------------------------------------------------------------x
Average Cholesky Decomposition Time: 150405 microseconds

x---------------------------------------------------------------------------------x
QR : LU : Cholesky -> 8 : 4 : 1
```



```
Enter the size of the matrix : 300
x---------------------------------------------------------------------------------x
Average QR Decomposition Time: 2255242 microseconds

x---------------------------------------------------------------------------------x
Average LU Decomposition Time: 1181094 microseconds

x---------------------------------------------------------------------------------x
Average Cholesky Decomposition Time: 254907 microseconds

x---------------------------------------------------------------------------------x
QR : LU : Cholesky -> 8 : 4 : 1
```



```
Enter the size of the matrix : 350
x---------------------------------------------------------------------------------x
Average QR Decomposition Time: 3441348 microseconds

x---------------------------------------------------------------------------------x
Average LU Decomposition Time: 1851689 microseconds

x---------------------------------------------------------------------------------x
Average Cholesky Decomposition Time: 399125 microseconds

x---------------------------------------------------------------------------------x
QR : LU : Cholesky -> 8 : 4 : 1
```



```
Enter the size of the matrix : 700
x---------------------------------------------------------------------------------x
Average QR Decomposition Time: 26428407 microseconds

x---------------------------------------------------------------------------------x
Average LU Decomposition Time: 14759655 microseconds

x---------------------------------------------------------------------------------x
Average Cholesky Decomposition Time: 3081915 microseconds

x---------------------------------------------------------------------------------x
QR : LU : Cholesky -> 8 : 4 : 1
```

CONCLUSION

In conclusion, the analysis of the code output indicates that Cholesky decomposition is the fastest computational method for solving Ax=b, closely followed by LU and QR. Each decomposition technique has its distinct advantages and limitations.

Cholesky is the most efficient method but is limited to symmetric positive definite matrices. LU, on the other hand, proves to be versatile and applicable to any square matrix, while QR extends its reach to rectangular matrices.

It's essential to consider the specific characteristics of the matrices and the computational scenario, as the relative speeds of these methods may vary. Furthermore, QR decomposition has the additional advantage of deriving eigenvalues, enhancing its utility in certain scenarios.

In summary, Cholesky is the fastest, LU offers broad applicability to square matrices, and QR extends its reach to rectangular matrices. The choice of the most suitable method depends on the nature of the matrices and the specific requirements of the computation.