

# **Numerical Analysis: homework 02**

Due on Tuesday, March 14, 2017

**102061149 Fu-En Wang**

# 1 Introduction

To solve such an equation:

$$Ax = b \quad (1)$$

where  $A$  is  $n \times n$  given nonsingular matrix and  $b$  is a given vector, while  $x$  is a vector to solve. Although Gaussian Elimination is an easy method for this problem, everytime when we adjust  $b$ , we have to re-calculate it again, which will be time-consuming. As a result, LU Decomposition provides a more flexible method to solve this problem.

## 1.1 LU Decomposition

Given a  $n \times n$  matrix  $A$ , we can split it by:

$$A = L \times U \quad (2)$$

where  $L$  is lower-triangular matrix and  $U$  is upper-triangular matrix. And we can express  $L$  and  $U$  by this format:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ u_{21} & 1 & 0 \\ u_{31} & u_{32} & 1 \end{bmatrix} \quad (3)$$

$$U = \begin{bmatrix} l_{11} & l_{12} & l_{13} \\ 0 & l_{22} & l_{23} \\ 0 & 0 & l_{33} \end{bmatrix} \quad (4)$$

So

$$A = \begin{bmatrix} 1 & 0 & 0 \\ u_{21} & 1 & 0 \\ u_{31} & u_{32} & 1 \end{bmatrix} \times \begin{bmatrix} l_{11} & l_{12} & l_{13} \\ 0 & l_{22} & l_{23} \\ 0 & 0 & l_{33} \end{bmatrix}$$

After we find out  $L$  and  $U$ , we can express problem by

$$LUx = b$$

Let  $Y = Ux$ , then

$$LY = b$$

$Y$  can be easily calculated, this is called **Forward Substitution**. After getting  $Y$ , the equation will be  $Ux = Y$ .  $x$  can be also easily calculated, this is called **Backward substitution**.

## 1.2 Error Calculation

Since the accuracy of our algorithm is quite important, I use **mean square error** to evaluate the performance:

$$Error = \sqrt{\frac{\sum_{i=1}^n (b'[i] - b[i])^2}{n}} \quad (5)$$

where  $b$  is the given vector and  $b'$  is the multiplication of  $A$  and  $x'$  which is calculated by our algorithm.

## 2 C++ Implementation

### 2.1 LU Decomposition

```

MAT &luFact(MAT &m1){
    for(int i=0; i<m1.m; i++){
        for(int j=i+1; j < m1.m; j++){
            m1[j][i] /= m1[i][i];
5           }
        for(int j=i+1; j < m1.m; j++){
            for(int k=i+1; k < m1.m; k++){
                m1[j][k] -= m1[j][i] * m1[i][k];
10            }
        }
    }
    return m1;
}

```

### 2.2 Forward Substitution

```

VEC fwdSubs(MAT &m1, VEC b){
    VEC Y(b);
    for(int i=0; i < m1.m; i++){
        for(int j=0; j < i; j++){
5            Y[i] -= m1[i][j] * Y[j];
        }
    }
    return Y;
}

```

### 2.3 Backward Substitution

```

VEC bckSubs(MAT &m1, VEC b){
    VEC X(b);
    for(int i=m1.m-1; i >= 0; i--){
        for(int j=m1.m-1; j > i; j--){
5            X[i] -= m1[i][j] * X[j];
        }
        X[i] /= m1[i][i];
    }
10    return X;
}

```

## 3 Complexity

### 3.1 LU Decomposition

In the outer double for-loop, we do  $\frac{n(n+1)}{2}$  times of one additional for-loop, so this is a  $O(n^3)$  problem.

### 3.2 Forward/Backward substitution

The double for-loop do  $\frac{n(n+1)}{2}$  times of calculation, so this is a  $O(n^2)$  problem.

## 4 Discussion

### 4.1 Performance Evaluation

In this project, I use 5 numbers to indicate the accuracy and efficiency of our method.

1. **Error**(mean square error of the calculated result and answer)
2. **Runtime**(total execution time of program)
3. **LU**(total runtime consumed by LU Decomposition)
4. **FWD**(total runtime consumed by Forward Substitution)
5. **BCK**(total runtime consumed by Backward Substitution)

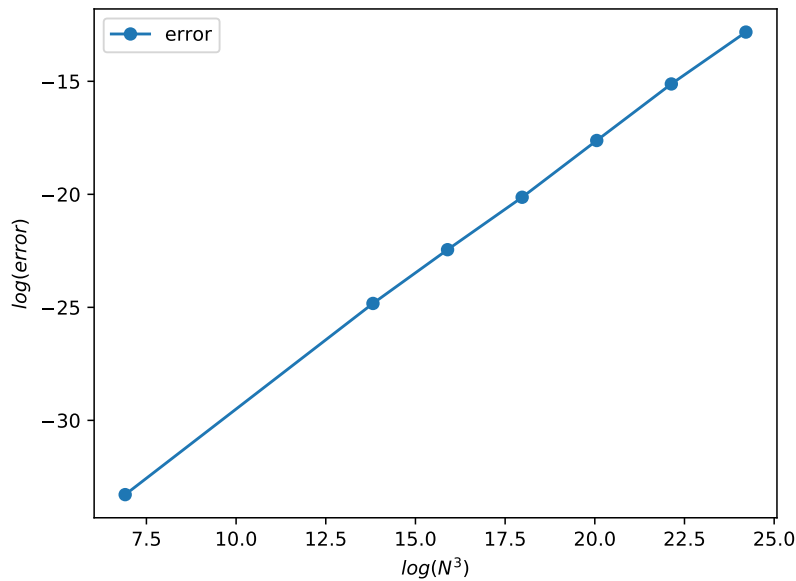
Table 1 shows detailed result of m3 to m10 execution result.

Table 1: Execution result of all matrix

	m3.dat	m4.dat	m5.dat	m6.dat	m7.dat	m8.dat	m9.dat	m10.dat
N	3	10	100	200	400	800	1600	3200
Error	0	3.50E-15	1.65E-11	1.78E-10	1.81E-09	2.23E-08	2.72E-07	2.70E-06
Runtime(s)	0.005	0.004	0.007	0.026	0.162	1.137	9.343	67.129
LU(s)	0	2.00E-06	0.00187	0.015341	0.130141	1.02409	8.895	65.4746
FWD(s)	1.00E-06	1.00E-06	4.10E-05	0.000121	0.000467	0.002353	0.007206	0.028792
BCK(s)	0	1.00E-06	2.90E-05	0.00012	0.000479	0.001957	0.007655	0.03202

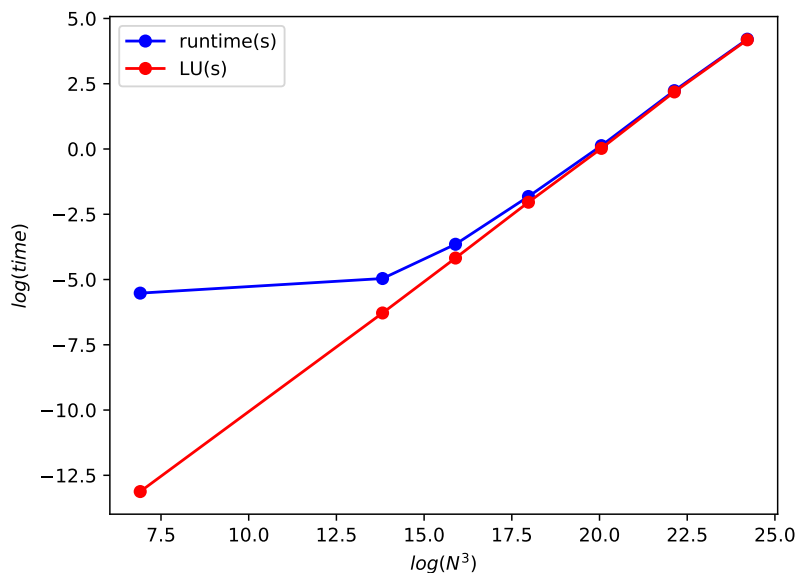
### 4.2 Error vs $N^3$

Figure 1 shows that the mean square error will grow up as dimension increases.

Figure 1: Error vs  $N^3$ 

### 4.3 Runtime/LU vs $N^3$

As mentioned in Section 3, the complexity of the whole process of LU Decomposition is  $O(n^3)$ . As a result, when plotting Runtime/LU to  $N^3$ , it should be a straight line. Figure 2 shows the result of Runtime and LU, respectively. Both of them match my expectation.

Figure 2: Runtime/LU vs  $N^3$ 

### 4.4 Forward/Backward Substitution

As mentioned in Section 3, the complexity of Forward/Backward Substitution is  $O(n^2)$ . So the relation between  $n^2$  and FWD/BCK will be a straight line. Figure 3 and Figure 4 show the experiment result, which

satisfy my opinion.

Figure 3: FWD vs  $N^2$

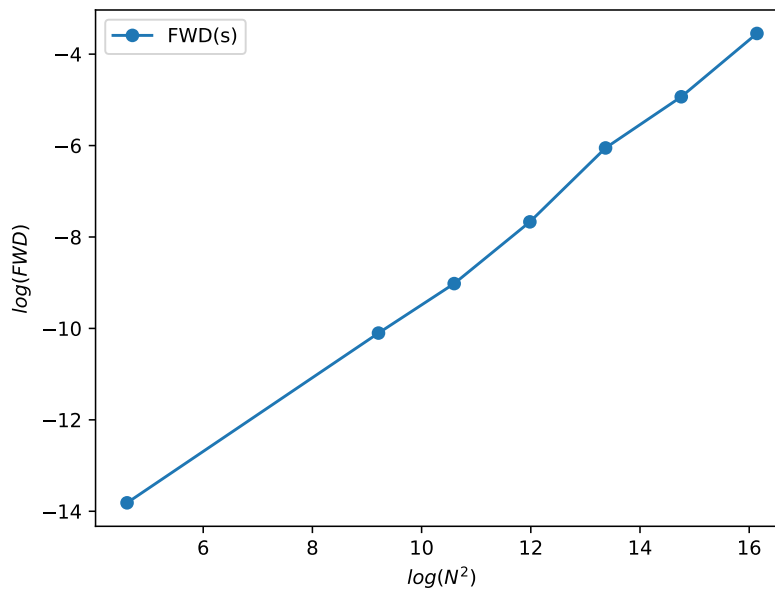
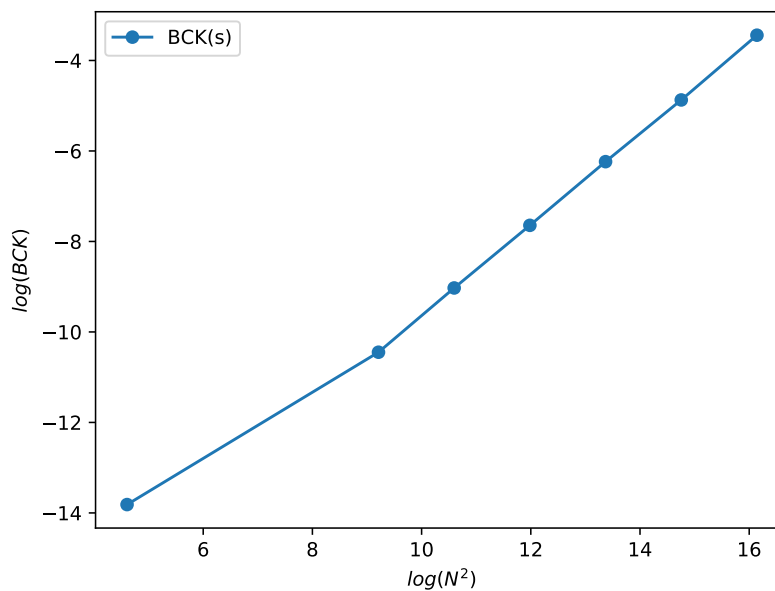


Figure 4: BCK vs  $N^2$



## 5 Advantage of LU Decomposition

In Section 3, we have already known the complexity of LU Decomposition is  $O(n^3)$ , which is same as Gaussian Elimination. But when solving  $Ax = b$  with a lot of  $b$ , we just need to do LU Decomposition once and then do Forward/Backward Substitution for different  $b$ , which complexity is  $O(n^2)$ . As a result, LU Decomposition will be much faster in such case.