컴퓨터와 선형대수학
-LU 분해

1. LU 분해

연립방정식을 풀 때, 직접 종이에 적으면서 풀기는 쉽지만, 그것을 컴퓨터가 풀도록 알고리즘으로 구현하는 것은 어렵다. 이때 연립방정식을 행렬을 이용하여 풀면 컴퓨터로도 연립방정식을 풀 수 있다. 단순한 연립방정식을 푸는 경우 고급 수학1 수업 시간에 가우스소거법을 이용한 방법을 배웠다.

컴퓨터에서 행렬을 계산할 때, 작은 행렬의 경우 문제없이 빠르게 계산해낼 수 있지만, 행렬의 크기가 커질수록 계산량이 기하급수적으로 커진다는 문제점이 있다. 컴퓨터 과학에서 알고리즘에서 입력값의 변화에 따라 연산을 실행할 때, 연산 횟수에 비해 시간이 얼마만큼 걸리는지 나타내는 방법인 Big-O 표기법을 이용하였을 때, 두 행렬의 곱의 시간복잡도를 나타내면 단 O(n^3)의 시간복잡도를 가진다. 이는 입력값의 세제곱만큼의 계산을 해야 한다는 것을 의미한다. n^3의 시간복잡도를 가지므로 입력값인 n의 값이 커지면 계산량도 그 세제곱만큼 많아져 컴퓨터의 입장에서도 너무 많은 계산을 해야 한다. 이러한 문제를 해결하기 위해 나타난 것이 LU 분해이다. LU 분해를 이용하면 계산을 최대한 간단히 하면서 답을 찾을 수 있다.

LU 분해를 하는 방법은 다음과 같다:

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= y_1 \\
a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= y_2 \\
a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= y_3
\end{aligned}
$$

위와 같은 연립방정식이 있다고 해보면, 위 연립방정식을 행렬로 표현할 수 있다.

$$
\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}
\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} =
\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}
$$
$$
\Longrightarrow Ax = B
$$

위 A 행렬을 아래와 같이 하삼각행렬과 상삼각행렬의 형태로 바꿔야 한다.

$$
\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} =
\begin{pmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{pmatrix}
\begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}
$$
$$
\Longrightarrow A = LU
$$

이렇게 A를 LU 분해하면 연립방정식을 풀 수 있다. LU 분해한 행렬을 이용한 연립방정식 풀이의 예시는 다음과 같다:

$$\begin{pmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{pmatrix}$$
$A = LU$

2행 − 1행 × 2
$$\begin{pmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ -2 & 7 & 2 \end{pmatrix}$$

3행 − 1행 × (−1)
$$\begin{pmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix}\begin{pmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 8 & 3 \end{pmatrix}$$

3행 − 2행 × (−1)
$$\begin{pmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1 & 1 \end{pmatrix}\begin{pmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 0 & 1 \end{pmatrix}$$
이렇게 하면 A 행렬을 LU 분해할 수 있다. 이제 연립방정식을 풀어보자.
$$\begin{pmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 5 \\ -2 \\ 9 \end{pmatrix}$$
$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1 & 1 \end{pmatrix}\begin{pmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 5 \\ -2 \\ 9 \end{pmatrix}$$

$LUx = b$에서 Ux를 z로 치환하면,
$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1 & 1 \end{pmatrix}\begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} 5 \\ -2 \\ 9 \end{pmatrix}$$
$$\begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} 5 \\ -12 \\ 2 \end{pmatrix}$$
$$Lx = \begin{pmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 5 \\ -12 \\ 2 \end{pmatrix}$$

$$\therefore \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}$$

이렇게 하면 연립방정식을 풀 수 있다. 하지만 여기서 드는 의문점은 왜 LU 분해를 사용하는가이다. LU 분해를 이용해 연립방정식을 푸는 과정을 보면 가우스소거법으로 한 번에 끝낼 수 있는 계산을 두 번에 나눠서 하는 것 같다. 이렇게 보면 효율적이지 않아 보인다. LU 분해를 이용한 연립방정식의 풀이는 “Ax=b에서 A는 그대로 있고, 여러 b에 대한 x를 구해야 하는 경우” 강점을 드러낸다. 고급 수학1 시간에 배운 가우스소거법을 사용하면 첨가 행렬을 만들어 b도 변형을 시켜야 한다. 그러나 LU 분해를 사용하면 Ax=b의 b 부분은 건드리지 않고 행렬을 계산하여 마지막에 b를 구한다는 것을 알 수 있다. 이를 통해 여러 다른 b에 대한 연립방정식의 해를 구할 수 있다. 이러한 LU 분해는 특정 분야에 대해서만 사용 가능한 것이 아니라 방정식을 컴퓨터로 풀어 문제를 풀어야 하는 경우 전반에 걸쳐서 활용할 수 있다.

2. 프로그램 제작

LU 분해 탐구 내용을 바탕으로 C++ 프로그래밍 언어를 이용하여 3x3 형태의 행렬을 입력받아 L과 U 행렬로 분해한 후, 연립방정식의 해를 계산하는 프로그램을 제작하였다. 이차원배열과 이중반복문 개념을 주로 적용하였다.

```
#include <stdio.h>
#include <stdlib.h>

int LU(double A_input[3][3]);  // LU 분해 함수 선언

double A[3][3];  // A(U) 행렬 선언
double L[3][3];  // L 행렬 선언

int main(){
        int i,j;
        double b[3];
        double z[3];
        double x[3];

        // A 행렬 입력
        printf("A행렬 입력\n");
        for(i=0;i<3;i++){
                printf("%d 번째 식의 계수 입력 : ", i+1);
                for(j=0;j<3;j++){
                scanf("%lf",&A[j][i]);
                }
        }
        // b 행렬 입력
        printf("\nb 행렬 입력 : ");
        for(i=0;i<3;i++){
```

```
            scanf("%lf", &b[i]);
        }
        // (1, 1) 항에 0이 있는지 판단
        if(A[0][0]==0){
            printf("\na(0, 0)이 0입니다.");
            exit;
        }else{
            // L 행렬의 (1, 1), (2, 2), (3, 3) 항에 1 대입
            for(i=0;i<3;i++){
                L[i][i] = 1;
            }

            LU(A);  // LU 분해 함수 호출

            // L 행렬 출력
            printf("\nL:");
            for(i=0;i<3;i++){
                printf("\n");
                for(j=0;j<3;j++){
                    printf("%.1lf ", L[j][i]);
                }
            }
            // U 행렬 출력
            printf("\n\nU:");
            for(i=0;i<3;i++){
                printf("\n");
                for(j=0;j<3;j++){
                    printf("%.1lf ", A[j][i]);
                }
            }
        }

        // Ux=z 로 치환 후, Lz=b 계산
        z[0] = b[0];
        z[1] = b[1]-L[0][1]*z[0];
        z[2] = b[2]-L[0][2]*z[0]-L[1][2]*z[1];
        // Ux=z 계산
        x[2] = z[2]/A[2][2];
        x[1] = (z[1]-A[2][1]*x[2])/A[1][1];
        x[0] = (z[0]-A[1][0]*x[1]-A[2][0]*x[2])/A[0][0];
```

```
        printf("\n\nx1 : %.1lf\nx2 : %.1lf\nx3 : %.1lf", x[0], x[1], x[2]);   //결과
}

// LU 분해 함수
int LU(double A_input[3][3]){
        double a;
        int i;

        // (1, 2)항 소거
        a = A_input[0][1]/A_input[0][0];
        L[0][1] = a;
        for(i=0;i<3;i++){
                A_input[i][1] = A_input[i][1]-A_input[i][0]*a;
                A[i][1] = A_input[i][1];
        }
        // (1, 3)항 소거
        a = A_input[0][2]/A_input[0][0];
        L[0][2] = a;
        for(i=0;i<3;i++){
                A_input[i][2] = A_input[i][2]-A_input[i][0]*a;
                A[i][2] = A_input[i][2];
        }
        // (2, 3) 항 소거
        a = A_input[1][2]/A_input[1][1];
        L[1][2] = a;
        for(i=0;i<3;i++){
                A_input[i][2] = A_input[i][2]-A_input[i][1]*a;
                A[i][2] = A_input[i][2];
        }
}
```

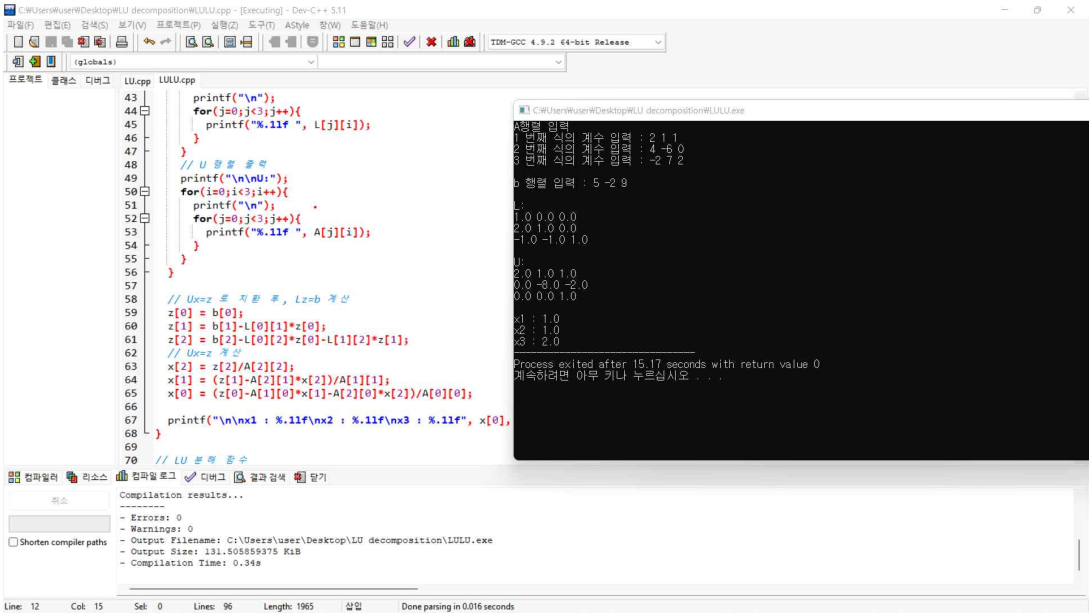

그림 1 실행 모습

입력
A행렬 입력 1 번째 식의 계수 입력 : 2 1 1 2 번째 식의 계수 입력 : 4 -6 0 3 번째 식의 계수 입력 : -2 7 2 b 행렬 입력 : 5 -2 9
출력
L: 1.0 0.0 0.0 2.0 1.0 0.0 -1.0 -1.0 1.0 U: 2.0 1.0 1.0 0.0 -8.0 -2.0 0.0 0.0 1.0 x1 : 1.0 x2 : 1.0 x3 : 2.0

3. 느낀점

LU 분해를 조사하면서 고급 수학1 과목을 수강하면서 배운 행렬에 대한 전반적인 지식을 활용하여 행렬의 내용을 되짚어보며 복습하고 실제 계산에 적용해볼 수 있었다. 또한 LU 분해에서 수업 시간에 잠깐 짚고 넘어갔던 기본 행렬의 개념이 필요해 자료를 조사하고 추가로 공부하면서 가우스소거법이 행렬과 기본 행렬의 곱으로 표현될 수 있다는 사실을 알았다.

LU 분해와 컴퓨터 연산의 관계를 조사하면서 컴퓨터로 방정식을 풀이하는데 행렬을 사용하는 이유에 대해 알 수 있었다. 행렬의 연산을 탐구하면서 1학년 때 가감법을 이용해 미지수가 두 개인 연립일차방정식의 근을 구하는 컴퓨터 프로그램을 제작했던 일을 떠올렸고, 내가 알게 모르게 행렬의 원리를 이미 사용해봤다는 것을 깨달았다.

처음 LU 분해를 접했을 때 “가우스소거법만으로도 풀리는 문제를 왜 두 번에 걸쳐서 계산하는 걸까?” 하는 의문을 가져, LU 분해의 유용성을 조사하면서 가우스소거법만 사용하는 것과 차별되는 LU 분해 방식의 이점을 알아보았다. Ax=b 방정식에서 A를 분해하는 것만으로 b 행렬을 건드리지 않고 해를 구할 수 있다는 부분에서 LU 분해의 특별함에 대해 감탄하였고 LU 분해가 어째서 컴퓨터의 연립방정식 풀이에 많이 사용되는지 이해할 수 있었다.

C++ 언어를 이용해 3x3 행렬의 LU 분해가 진행되는 과정을 알고리즘으로 정리하고 구현함으로써 LU 분해의 과정을 더욱 잘 이해할 수 있었다. LU 분해를 식으로만 접하였을 때는 화면에 있는 식이 어떤 의미인지는 알았으나 깊이있게 이해하지는 못하였다. 하지만 프로그램을 제작하면서 LU 분해 과정을 알고리즘으로 구현하기 위해 깊이 고민하고, 끝없이 손으로 써보면서 LU 분해와 연립방정식의 해가 구해지는 과정을 논리적으로 깊이 이해할 수 있었다. 1학년 때 가감법을 이용해서 미지수가 2개인 연립일차방정식을 푼 것에서 발전하여, 수학의 행렬과 비슷한 개념인 2차원 배열과 이중반복문을 적절히 활용하여 행렬의 연산을 구현하였고, 미지수가 3개 이상일 때도 적용할 수 있는 연립일차방정식 풀이 프로그램을 제작할 수 있었다.