LU 분해법

김기택
국민대학교 소프트웨어학과

LU 분해법 개요

- 정사각형 행렬 A 는 하부 삼각행렬 L 과 상부 삼각행렬 U 의 곱으로 표현될 수 있다.

$$\mathbf{A} = \mathbf{L}\,\mathbf{U}$$

- 주어진 **A** 에 대해 **L** 과 **U** 를 계산하는 과정을 LU 분해법 (decomposition) 또는 LU 인수분해 (factorization) 라고 한다.
  - 특정한 조건이 **L** 또는 **U** 에 부여되지 않는 한 LU 분해는 고유하지 않다. (**L** 과 **U** 의 조합은 무한)
- 일반적으로 사용하는 방법 3가지는 다음과 같다.

이름	제약 조건
Doolittle 분해	$L_{ii} = 1,\ i = 1, 2, \ldots, n$
Crout 분해	$U_{ii} = 1,\ i = 1, 2, \ldots, n$
Choleski 분해	$\mathbf{L} = \mathbf{U}^T$

- **A** 를 분해한 후에 $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ 는 $\mathbf{L}\mathbf{U}\,\mathbf{x} = \mathbf{b}$ 로 표현된다. 이중 $\mathbf{U}\,\mathbf{x} = \mathbf{y}$ 라 하고 이는 전진 대입에 의해 푼다. 그러면 최종식은 $\mathbf{L}\,\mathbf{y} = \mathbf{b}$ 가 되며 이는 후진 대입에 의해 푼다.
- LU 분해법은 **외부조건 벡터, b 가 변화해도 계수 행렬을 소거할 필요 없이 한번 만 분해를 하면 계속 사용**할 수 있어 **계산 시간이 크게 단축**되는 장점이 있다.

Doolittle 분해법 (분해 단계)

- 가우스 소거와 밀접한 관련이 있다. 예를 들어 3×3 행렬 $\mathbf{A}$와 $\mathbf{A} = \mathbf{L}\,\mathbf{U}$ 와 같은 삼각행렬이 있다고 하자.

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{bmatrix} \qquad \mathbf{U} = \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix}$$

- 두 행렬을 곱하면,

$$\mathbf{A} = \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ U_{11}L_{21} & U_{12}L_{21} + U_{22} & U_{13}L_{21} + U_{23} \\ U_{11}L_{31} & U_{12}L_{31} + U_{22}L_{32} & U_{13}L_{31} + U_{23}L_{32} + U_{33} \end{bmatrix}$$

- 여기에 가우스 소거를 적용하여 진행하면,

첫번째 단계

$$\mathbf{A}' = \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & U_{22}L_{32} & U_{23}L_{32} + U_{33} \end{bmatrix}$$

$\lambda_2 = \frac{U_{11}L_{21}}{U_{11}} = L_{21}$

$\lambda_3 = \frac{U_{11}L_{31}}{U_{11}} = L_{31}$

두번째 단계

$$\mathbf{A}'' = \mathbf{U} = \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix}$$

$\lambda_3 = \frac{U_{22}L_{32}}{U_{22}} = L_{32}$

Doolittle 분해법의 특징

- 앞의 예에서 다음과 같은 중요한 2가지 특징이 있음을 알 수 있다.
  - 행렬 **U** 는 가우스 소거로 인해 발생하는 상삼각 행렬과 동일하다.
  - **L** 의 대각선이 아닌 요소는 가우스 소거 중에 사용되는 **피벗 방정식의 승수(multiplier)**가 된다. 즉, L_{ij} 는 A_{ij} 를 소거하는 승수이다.

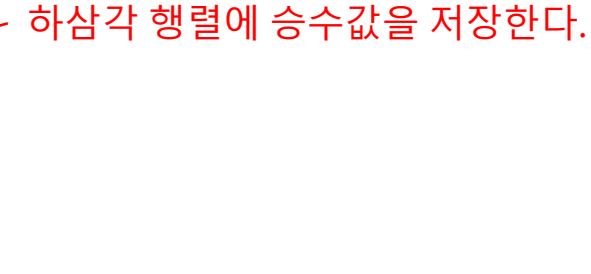
- 계수 행렬의 하위 삼각형 부분에 승수를 저장하고 계수를 소거할 때 계수를 대체하는 것이 일반적이다. (L_{ij} 가 A_{ij} 를 대체함)
  - **L** 의 대각선 요소는 1 (unity) 이기 때문에 저장할 필요가 없다.
  - 최종 분해된 계수 행렬의 형태는 **L** 과 **U** 가 혼합된 형태이다. 분해 과정은 가우스 소거 과정과 동일하다.

$$[\mathbf{L} \setminus \mathbf{U}] = \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ L_{21} & U_{22} & U_{23} \\ L_{31} & L_{32} & U_{33} \end{bmatrix}$$

분해 단계 알고리즘

- 분해 단계를 프로그램으로 나타내면 다음과 같다.

```
for k in range(0, n-1):
    for i in range(k+1, n):
        if a[i, k] != 0.0:
            lam = a[i, k] / a[k, k]
            a[i, k+1:n] = a[i, k+1:n] – lam*a[k, k+1:n]
            a[i, k] = lam
```



Doolittle 분해법 (해를 구하는 단계)

- 전진 대입에 의해 $\mathbf{L}\,\mathbf{y} = \mathbf{b}$ 의 해를 구하는 절차를 보자. 스칼라 방정식은 다음과 같다.

$$
\begin{aligned}
y_1 &= b_1 \\
L_{21}\,y_1 + y_2 &= b_2 \\
&\vdots \\
L_{k1}\,y_1 + L_{k2}\,y_2 + \ldots + L_{k,k\text{-}1}\,y_{k\text{-}1} + y_k &= b_k \\
&\vdots
\end{aligned}
$$

- y_k 에 대한 k 번째 방정식을 풀면,

$$y_k = b_k - \sum_{j=1}^{k-1} L_{kj} y_j \quad , \qquad k = 2, 3, \ldots, n$$

- 전진 대입 알고리즘은 다음과 같고, 후진 대입 알고리즘은 가우스 소거법과 동일하다.

```
y[0] = b[0]
for k in range(1, n):
    y[k] = b[k] – np.dot(a[k, 0:k], y[0: k])
```

첫번째 행은 자동으로 해가 구해진다.

두번째 행부터 n-1 행까지 반복

Ludecomp 프로그램

```
## module LUdecomp
''' a = LUdecomp(a)
LUdecomposition: [L][U] = [a]
x = LUsolve(a,b)
Solution phase: solves [L][U]{x} = {b}
'''
import numpy as np
def LUdecomp(a):
    n = len(a)
    for k in range(0,n-1):
        for i in range(k+1,n):
            if a[i,k] != 0.0:
                lam = a [i,k]/a[k,k]
                a[i,k+1:n] = a[i,k+1:n] - lam*a[k,k+1:n]
                a[i,k] = lam
    return a
def LUsolve(a,b):
    n = len(a)
    for k in range(1,n):
        b[k] = b[k] - np.dot(a[k,0:k],b[0:k])
    b[n-1] = b[n-1]/a[n-1,n-1]
    for k in range(n-2,-1,-1):
        b[k] = (b[k] - np.dot(a[k,k+1:n],b[k+1:n]))/a[k,k]
    return b
```

모듈에 대한 설명

LU분해과정

해를 구하는 과정

해가 b 에 저장됨

결과 b 를 반환

Choleski 분해법

- Choleski 분해는 $\mathbf{A} = \mathbf{LU} = \mathbf{LL}^T$ ($\mathbf{U} = \mathbf{L}^T$) 의 관계를 가지도록 분해하는 것이다. 다음 2가지 한계가 있다.
  - $\mathbf{LL}^T$ 는 항상 대칭 행렬이므로 Choleski 분해를 위해서는 **A** 가 대칭이어야 한다. (그리고 양의 정부호 행렬 – positive definite)
  - 분해 과정에는 **A** 원소들의 특정 조합의 제곱근을 취하는 것이 포함되는데, 음수의 제곱근을 피하기 위해서는 **A** 가 양의 값이어야 한다.
- 장점
  - Choleski 분해는 약 $n^3/6$ 의 긴(long) 연산과 n 제곱근 계산이 포함된다. 이는 LU 분해에 필요한 연산 수의 절반 정도이다. 따라서 상대적으로 효율이 좋은 것은 대칭의 활용 때문이다.

Positive definite matrix (양정 행렬)

- 정의
  - 0 이 아닌 모든 벡터에 대해 $\mathbf{x}^T \mathbf{M}\, \mathbf{x} > 0$ 인 조건을 만족하는 행렬, $\mathbf{M}$

- 예

$$\mathbf{M} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$\mathbf{x}^T \mathbf{M}\, \mathbf{x} = [x_1\ x_2] \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 2x_1^2 + 2x_1 x_2 + 2x_2^2$$
$$= (x_1 + x_2)^2 + x_1^2 + x_2^2 > 0$$

  - 위 행렬은 양의 정부호 행렬이다.

Choleski 분해 (1)

- 예로서 3×3 행렬의 분해를 살펴 보자.

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T$$

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{bmatrix} \begin{bmatrix} L_{11} & L_{21} & L_{31} \\ 0 & L_{22} & L_{32} \\ 0 & 0 & L_{33} \end{bmatrix}$$

- 오른쪽에 있는 행렬 곱셈을 완료하면,

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} L_{11}^2 & L_{11}L_{21} & L_{11}L_{31} \\ L_{11}L_{21} & L_{21}^2 L_{22}^2 & L_{21}L_{31} + L_{22}L_{32} \\ L_{11}L_{31} & L_{21}L_{31} + L_{22}L_{32} & L_{31}^2 + L_{32}^2 + L_{33}^2 \end{bmatrix}$$

- 오른쪽은 대칭 행렬이므로 연산을 위한 요소는 총 6개 이다. 이 방정식들을 일정한 순서로 풀면 각 방정식에서 하나의 미지수만 가질 수 있다.

- 행렬의 하삼각 (또는 상삼각) 행렬을 살펴 보자. 첫번째 행부터 차례로 아래로 진행한다.

Choleski 분해 (2)

$$A_{11} = L_{11}^2 \qquad L_{11} = \sqrt{A_{11}}$$
$$A_{21} = L_{11}L_{21} \qquad L_{21} = A_{21}/L_{11}$$
$$A_{31} = L_{11}L_{31} \qquad L_{31} = A_{31}/L_{11}$$

- 두번째 열은 두번째 행으로 시작하여 L_{22} 와 L_{32} 를 산출한다.

$$A_{22} = L_{21}^2 + L_{22}^2 \qquad L_{22} = \sqrt{A_{22} - L_{21}^2}$$
$$A_{32} = L_{21}L_{31} + L_{22}L_{32} \qquad L_{32} = (A_{32} - L_{21}L_{31})/L_{22}$$

- 마지막 세번째 열, 세번째 행은 L_{33} 를 제공한다.

$$A_{33} = L_{31}^2 + L_{32}^2 + L_{33}^2 \qquad L_{33} = \sqrt{A_{33} - L_{31}^2 - L_{32}^2}$$

- 이제 $n \times n$ 행렬에 대한 결과를 추정할 수 있다. $\mathbf{L}\mathbf{L}^T$ 하삼각 행렬 부분에 있는 요소가

$$(\mathbf{L}\mathbf{L}^T)_{ij} = L_{i1}L_{j1} + L_{i2}L_{j2} + \cdots + L_{ij}L_{jj} = \sum_{k=1}^{j} L_{ik}L_{jk} \ , \qquad i \geq j$$

Choleski 분해 (3)

- 이 항을 A 의 원소로 표시하면,

$$A_{ij} = \sum_{k=1}^{j} L_{ik} L_{jk} \ , \qquad i = j, j+1, \dots, n, \qquad j = 1, 2, 3, \dots, n \qquad (2.17)$$

  - 표시된 인덱스의 범위는 하삼각 행렬의 요소로 제한한다.
- 첫번째 열 $(j = 1)$에 대해 앞의 식에서 다음을 얻을 수 있다.

$$L_{11} = \sqrt{A_{11}} \qquad L_{i1} = \frac{A_{i1}}{L_{11}} \ , \ i = 2, 3, \dots, n$$

- 다른 열로 진행하면 식 (2.17)에서 미지수는 L_{ij} 이며, 다른 요소들은 앞의 열에서 이미 계산되어 있음을 알 수 있다.
  - 3×3 행렬의 예에서 2번째 열의 경우 L_{22} 와 L_{32} 가 미지수 이며 다른 요소들은 앞 열에서 얻는다.
  - 식 (2.17)을 미지수를 별도로 표시하여 나타내면,

$$A_{ij} = \sum_{k=1}^{j-1} L_{ik} L_{jk} + L_{ij} L_{jj}$$

Choleski 분해(4)

- 만약, $i = j$ (대각선 항)인 경우 해는 다음과 같다.

$$L_{jj} = \sqrt{A_{jj} - \sum_{k=1}^{j-1} L_{jk}^2}, \qquad j = 2, 3, \ldots, n$$

- 비대각선 항목의 경우는 다음과 같다.

$$L_{ij} = \left(A_{ij} - \sum_{k=1}^{j-1} L_{ik} L_{jk} \right) / L_{jj}, \qquad j = 2, 3, \ldots, n-1, \qquad i = j+1, j+2, \ldots, n$$

Choleski 프로그램

```
## module choleski
''' L = choleski(a)
    Choleski decomposition: [L][L]transpose = [a]
    x = choleskiSol(L,b)
    Solution phase of Choleski's decomposition method
'''
import numpy as np
import math
import error
def choleski(a):
    n = len(a)
    for k in range(n):  0부터 n-1 까지
        try:
            a[k,k] = math.sqrt(a[k,k] \ 양의 정부호 행렬 검사
                    - np.dot(a[k,0:k],a[k,0:k]))
        except ValueError:
            error.err('Matrix is not positive definite')
        for i in range(k+1,n):
            a[i,k] = (a[i,k] – np.dot(a[i,0:k],a[k,0:k]))/a[k,k]
        for k in range(1,n):      choleski 분해
            a[0:k,k] = 0.0
    return a

def choleskiSol(L,b):   전진과 후진 대입으로 해를 구하기
    n = len(b)
 # Solution of [L]{y} = {b}  전진 대입
    for k in range(n):
        b[k] = (b[k] - np.dot(L[k,0:k],b[0:k]))/L[k,k]
 # Solution of [L_transpose]{x} = {y}   후진 대입
    for k in range(n-1,-1,-1):
        b[k] = (b[k] - np.dot(L[k+1:n,k],b[k+1:n]))/L[k,k]
    return b
```

예제 2.5

Doolittle 분해법을 사용하여 방정식 $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ 를 풀어 보아라.

$$\mathbf{A} = \begin{bmatrix} 1 & 4 & 1 \\ 1 & 6 & -1 \\ 2 & -1 & 2 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 7 \\ 13 \\ 5 \end{bmatrix}$$

[풀이] 먼저 가우스 소거로 계수 행렬을 분해한다.

열 2 ← 열 2 – 1 × 열 1 (A_{21} 제거)

열 3 ← 열 3 – 2 × 열 1 (A_{31} 제거)

삭제된 항목 대신에 승수 $L_{21} = 1$, $L_{31} = 2$ 를 저장하면 다음과 같이 된다.

$$\mathbf{A}' = \begin{bmatrix} 1 & 4 & 1 \\ 1 & 2 & -2 \\ 2 & -9 & 0 \end{bmatrix}$$

가우스 소거의 두번째 단계는, 열 3 ← 열 3 – 1(–4.5) × 열 2 (A_{32} 제거)

A_{32} 대신에 승수 $L_{32} = -4.5$ 를 저장하면,

$$\mathbf{A}'' = [\mathbf{L} \setminus \mathbf{U}] = \begin{bmatrix} 1 & 4 & 1 \\ 1 & 2 & -2 \\ 2 & -4.5 & -9 \end{bmatrix}$$

예제 2.5 (continued)

분해는 다음과 같이 완료된다.

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 2 & -4.5 & 1 \end{bmatrix} \quad \mathbf{U} = \begin{bmatrix} 1 & 4 & 1 \\ 0 & 2 & -2 \\ 0 & 0 & -9 \end{bmatrix}$$

다음으로 전진 대입에 의해 $\mathbf{L}\,\mathbf{y} = \mathbf{b}$ 의 해를 구한다.

$$[\mathbf{L}\,|\mathbf{b}] = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 7 \\ 1 & 1 & 0 & 13 \\ 2 & -4.5 & 1 & 5 \end{array}\right]$$

해는 $y_1 = 7, y_2 = 13 - 7 = 6, y_3 = 5 - 2(7) + 4.5(6) = 18$

방정식 $\mathbf{U}\,\mathbf{x} = \mathbf{y}$ 는 후진 대입으로 구한다.

$$[\mathbf{U}\,|\mathbf{y}] = \left[\begin{array}{ccc|c} 1 & 4 & 1 & 7 \\ 0 & 2 & -2 & 6 \\ 0 & 0 & -9 & 18 \end{array}\right]$$

해는 $x_3 = 18/(-9) = -2, x_2 = [6 + 2(-2)]/2 = 1, x_1 = 7 - 4(1) - (-2) = 5$

예제 2.6

다음 행렬 A 에 대해 Choleski 분해를 계산하라.

$$\mathbf{A} = \begin{bmatrix} 4 & -2 & 2 \\ -2 & 2 & -4 \\ 2 & -4 & 11 \end{bmatrix}$$

[풀이] A 는 대칭이고 양의 정부호 (positive definite) 행렬임을 알 수 있으므로 choleski 분해가 가능하다. 분해 알고리즘에 자체적으로 양의 정부호 행렬임을 확인하는 과정이 포함되어 있어 선행 테스트는 필요하지 않다. (음수의 제곱근이 발생하면 양의 정부호 행렬이 아니므로 분해가 실패한다.) 정의에 따라 다음과 같아진다.

$$\begin{bmatrix} 4 & -2 & 2 \\ -2 & 2 & -4 \\ 2 & -4 & 11 \end{bmatrix} = \begin{bmatrix} L_{11}^2 & L_{11}L_{21} & L_{11}L_{31} \\ L_{11}L_{21} & L_{21}^2 L_{22}^2 & L_{21}L_{31} + L_{22}L_{32} \\ L_{11}L_{31} & L_{21}L_{31} + L_{22}L_{32} & L_{31}^2 + L_{32}^2 + L_{33}^2 \end{bmatrix}$$

각 요소를 등식으로 풀면

$$L_{11} = \sqrt{4} = 2$$

$$L_{21} = -\frac{2}{L_{11}} = -\frac{2}{2} = -1$$

예제 2.6 (continued)

$$L_{31} = \frac{2}{L_{11}} = \frac{2}{2} = 1$$

$$L_{22} = \sqrt{2 - L_{21}^2} = \sqrt{2 - 1^2} = 1$$

$$L_{32} = \frac{-4 - L_{21}L_{31}}{L_{22}} = \frac{-4 - (-1)(1)}{1} = -3$$

$$L_{33} = \sqrt{11 - L_{31}^2 - L_{32}^2} = \sqrt{11 - 1^2 - (-3)^2} = 1$$

그러므로,

$$\mathbf{L} = \begin{bmatrix} 2 & 0 & 0 \\ -1 & 1 & 0 \\ 1 & -3 & 1 \end{bmatrix}$$

이 결과는 $\mathbf{L}\mathbf{L}^T$ 를 수행하면 확인할 수 있다.

예제 2.7

Doolittle 방법으로 **A X** = **B** 를 풀고 행렬식, |**A**| 를 계산하는 프로그램을 작성하시오. 함수는 Ludecomp, Lusolve를 사용하고 결과를 확인하라.

$$\mathbf{A} = \begin{bmatrix} 3 & -1 & 4 \\ -2 & 0 & 5 \\ 7 & 2 & -2 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 6 & -4 \\ 3 & 2 \\ 7 & -5 \end{bmatrix}$$

[풀이]

```
## example2_7
import numpy as np
from LUdecomp import *
a = np.array([[ 3.0, -1.0, 4.0], [-2.0, 0.0, 5.0], [ 7.0, 2.0, -2.0]])
b = np.array([[ 6.0, 3.0, 7.0], [-4.0, 2.0, -5.0]])
a = LUdecomp(a)                  # Decompose [a]
det = np.prod(np.diagonal(a))        행렬식 계산
print("\nDeterminant =",det)
for i in range(len(b)):           # Back-substitute one
   x = LUsolve(a,b[i])            # constant vector at a time
print("x",i+1,"=",x)
```

$\mathbf{Ax} = \mathbf{b} \rightarrow \mathbf{Ux} = \mathbf{e}$
$|\mathbf{A}| = |\mathbf{U}|$
상삼각행렬의 행렬식은 대각요소의 곱

예제 2.8

Choleski 분해로 방정식 $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ 를 풀고 결과를 확인하라.

$$\mathbf{A} = \begin{bmatrix} 1.44 & -0.36 & 5.52 & 0.0 \\ -0.36 & 10.33 & -7.78 & 0.0 \\ 5{,}52 & -7.78 & 28.40 & 9.0 \\ 0.0 & 0.0 & 9.0 & 61.0 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 0.04 \\ -2.15 \\ 0 \\ 0.88 \end{bmatrix}$$

[풀이]

```
## example2_8
import numpy as np
from choleski import *
a = np.array([[ 1.44, -0.36, 5.52, 0.0], [-0.36, 10.33, -7.78, 0.0], \
                    [ 5.52, -7.78, 28.40, 9.0], [ 0.0, 0.0, 9.0, 61.0]])
b = np.array([0.04, -2.15, 0.0, 0.88])
aOrig = a.copy()
L = choleski(a)
x = choleskiSol(L,b)
print("x =",x)
print('\nCheck: A*x =\n',np.dot(aOrig,x))
```