**1.1 Introduction**

**- 선형대수란**

연립 선형방정식이 있을 때

X, Y가 있고 1차 제곱들인 것들 (x^1, y^1...). 루트, 2제곱 이런건 ㄴㄴ

**- 소거법 (Elimination)**

쉬움.

**- Determinants (Cramer’s Rule)**

1x + 2y = 3

4x + 5y = 6

이 있을 때

y = [[1, 3], [4, 6]] / [[1, 2], [[4. 5]] => (6 – 12) / (5 – 8) = -6 / -3 = 2

X = [[3, 2], [6, 5]] / [[1, 2], [[4, 5]] => (15 – 12) / (5 – 8) = 3 / -3 = -1

여기서 분모는 x, y의 계수를 넣고 y를 구할 때는 y계수에 오른쪽 상수를 넣고 x를 구할 때는 x 계수 대신에 오른쪽을 넣음.

크레이머 룰은 큰 시스템에 부적합. **가우스 소거법이 아직도 가장 유효한 풀이법.**

**- 선형 방정식의 4개 측면**

1. Geometry of planes - 2개의 직선이 만나는 교점 (기하학적)

2. Matrix notation: A = [[1, 2], [4, 5]] = L\*U = [[1, 0], [4, 1]] \* [[1, 2], [0, -3]]

L은Lower triangle, U는 Upper triangle . LU Decomposition.

3. Singular case:

2개의 평행한 라인, 2개의 겹치는 라인일 경우 각각 답이 없거나(불능 – 해를 구할 수 없음), 무한개(부정 – 해를 1개로 정하기가 어렵다).

4. 가우스 소거를 할 때 계산 상의 문제. (The number of elimination steps)

변수 n개, n개의 선형 방정식이 주어져 있을 때 계산 시간 = 1/3 \* n^3.

Rounf off error는 심각할 수 있음.

**1.2 The Geometry of Linear Equations**

**By rows**: 두 직선이 교점에서 만남.

**By columns:**

Column form: x\*[[2], [1]] + y\*[[-1], [1]] = [[1], [5]] (행렬은 세로로 되어 있음.)

칼럼 벡터들의 콤비네이션을 찾는 것.

Row picture: 평면의 교차. – Column picture: column의 조합

**- The Singular Case**

(A) 두 평면이 평행하면 – inconsistent. 불능.

(B) 각각 평면들의 쌍(pair)이 각각 한 선에서 만나고, 세 직선들이 평행할 때 – inconsistent. 불능.

한 평면이 다른 두 평면이 만나는 선과 평행함.

(C) 두 평면을 더해서 다른 평면과 같아지면 – infinity of solution. 부정.

(D) 3개의 평행한 평면 - 겹쳐 있으면 부정, 아니면 불능.

**- The Singular Case: Column Picture**

슬라이드 참조.

**1.3 Example of Gaussian Elimination**

(1) 2u + v + w = 5 / (2) 4u – 6n = -2 / (3) -2u + 7v + 2w = 9

1. (2)와 (1)의 u의 계수인 4 / 2 = 2를 (1)에 곱함.

1-1. (1)은 그대로 있음.

2. (1)에서 (2)를 뺌. -> -8v – 2w = -12가 나옴.

3. (1)이랑 (3)을 더함. -> 8n + 3w = 14가 나옴.

4. 계산한 (2)와 (3)을 더함. -> w = 2가 나옴.

지금까지 Forward Elimination

앞으로 Backward Substitution

5. w를 계산한 (2)에 대입하면 v = 1이 나옴.

6. w, v를 (1)에 대입하면 u = 1이 나옴.

- Remark

슬라이드 참조.

세번째 식에 왼쪽 3개의 column은 upper triangular matrix가 됨. (대각선 아래가 모두 0)

행렬은 array로 구현하면 아주 간단해짐.

대각선에 있는 원소(pivot 이라고 함)가 0이 되면 불능. (우변도 0이면 부정?)

**- The breakdown of Elimination**

Pivot은 0이 되면 안됨.

만약 pivot의 위치에 0이 나타날 경우.

1. Non-singular case - 식의 순서를 바꾸면 됨.
2. Singular case – 부정 혹은 불능. (챕터 2에서 부정 케이스 다룸)

**- Ex1 & Ex2**

Ex1 – non-singular 이면 빼놓고 두번째 식에 0이 있으면 세번째 꺼랑 바꾸면 됨.

Ex2 – singular 이면 순서 바꿔도 pivot이 0이 되는걸 피할 수가 없음.

Case 1: 불능이면 한 변수에 대한 결과가 달라짐. – unsolvable.

Case 2: 부정이면 한 변수에 대한 결과가 같아서 – consistent.

하지만 다른 변수들의 값을 정할 수가 없음.

**- The Cost of Elimination**

식: 1/3 \* n^3 + … -> O(n^3)

1. N개의 columns \* (N-1)개의 rows가 값이 바뀜. => multiply-subtract
2. (N-1) \* (N-2)
3. 반복.. 해서 2\*1
4. 즉 k = 1부터 n-1까지 (k+1)(k)를 더한 값이 됨. = 1/3\*(n-1)\*n\*(n+1)

**- Non-Linear 문제 푸는 법** -> Linear 문제를 여러 개 반복.

**- Linear Algebra for Curve-Curve Intersection**

M차 곡선과 N차 곡선이 있으면 m\*n개의 점에서 만남.

**- 일반 풀이법**

곡선 2개가 C(u) = (x(u), y(u)), D(v) = (alpha(v), beta(v)) 로 있을 때

x(u) = alpha(v), y(u) = beta(v)의 두 가지 조건을 만족하면 한 점에서 만남.

=> x(u) – alpha(v) = 0, y(u) – beta(v) = 0

=> f(u, v) = 0, g(u, v) = 0 => v를 소거 => f가 m차, g가 n차일 때

=> p(u) = 0, m\*n차까지 됨. 그리고 그럼 u값을 대입하고 그다음 v값 대입.

**=> 소거하는 방법은 어려움. 교수님께서 안 가르쳐주심.**

**- 선형 대수 이용**

C(u\_0) = (x(u\_0), y(v\_0)), D(v\_0) = (alpha(v\_0), beta(v\_0))에서 시작해서

접선을 그어서 선형대수로 변환해서 풂. -> Newton Iteration. -> 만족할 때까지 반복.

C(u\_0)의 접점 = C(u\_0) + S\*C’(u\_0). (C’은 도함수) = D(v\_0) + t\*D’(v\_0)

그 뒤 식 변환 과정은 pdf 참조.

**1.4 Matrix Notation and Matrix Multiplication**

이건 슬라이드 보면 이해됨. 그냥 선형 연립방정식을 Ax = b 식의 matrix form으로 나타낼 수 있다는 것.

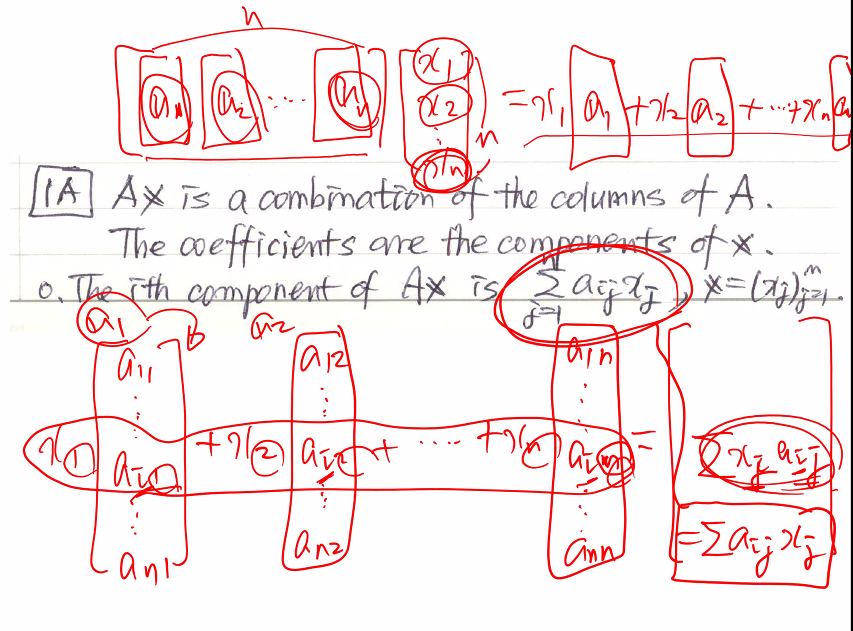
**- Two ways to multiply a matrix A by a vector x**

**- Ax by rows:**

A의 row 벡터들을 x랑 내적하는 것. (내가 일반적으로 하는 곱 계산)

**- Ax by columns:**

첫 번째 칼럼과 x의 첫번째 원소 곱 + 두번째 칼럼과 x의 두번째 원소 곱 + …



**- The Matrix Form of One Elimination Step**

Identity Matrix는 대각선이 모두 1, 나머지는 0인 행렬.

Pdf 행렬 E에서 값을 보면 -2의 의미가 b의 첫번째 원소에 -2를 곱한 값을 계산의 두번째에 더해주는 개념.

가우스 소거법에서의 각각의 단계에 이러한 계산이 들어감. 이러한 행렬을 Elimination Matrix라고 함.

1B는 위 내용의 일반화. B\_i – b\_j \* l. 이 경우는 b3 – l\*b1. 이부분 헷갈림. 식과 그림으로 이해하기.

**- Matrix Multiplication**

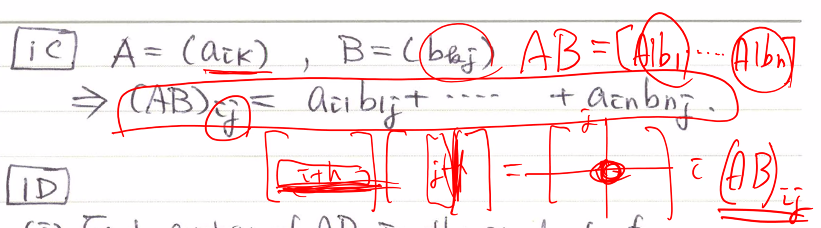
그림의 EA에서 A의 1번째, 3번째 row는 그대로 보존됨. 2번째 row만 기존 값에 1번째 row의 값\*2 한걸 뺀 값이 나옴.

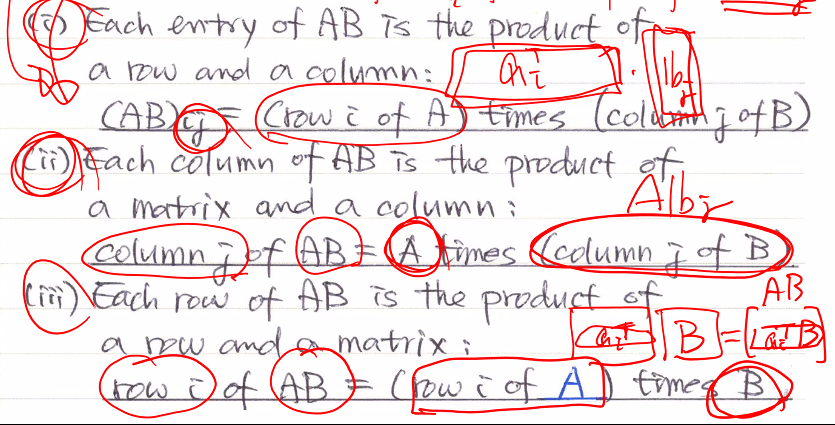
(EA)x = E(Ax) = EAx. 결합 법칙(Associative Law)

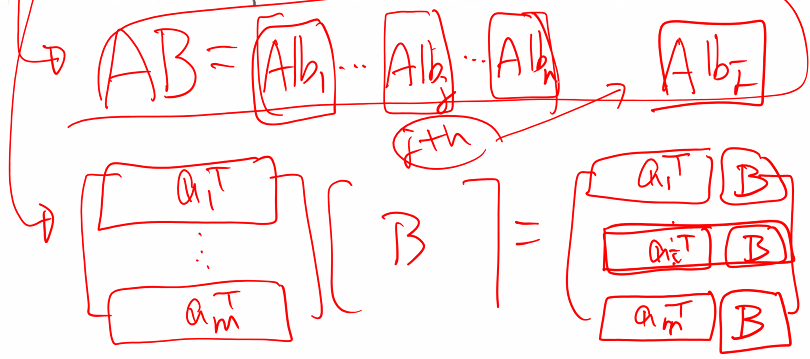
=> Ax = b => E(Ax) = Eb == (EA)x = Eb.

(EA)x = Eb 상태에서 변형을 하다보면 A가 upper angular가 되어 Ux = c 형태의 결과가 됨.

혹은 column별로 보면, AB = A[b1 b2 b3] = [Ab1 Ab2 Ab3] 으로 볼 수도 있음.

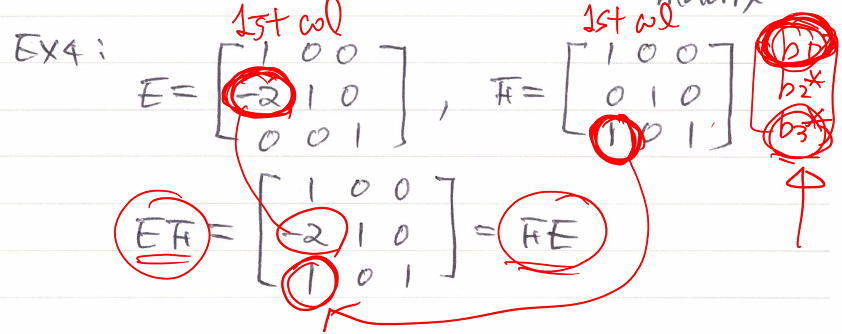


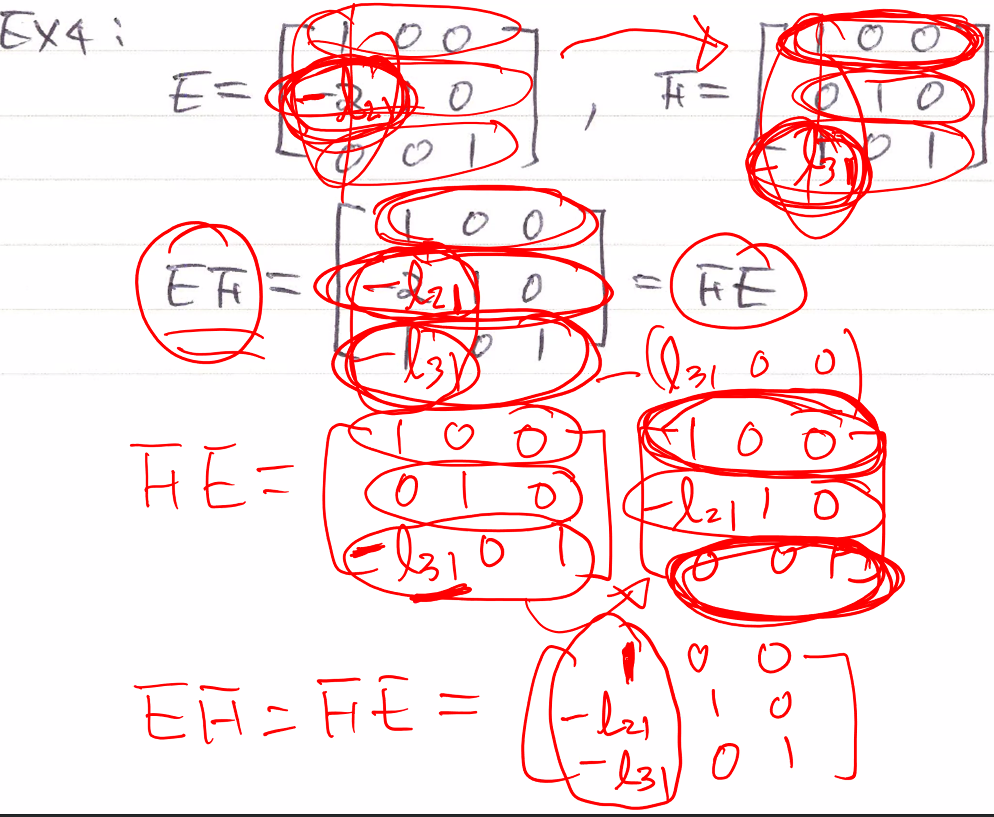




1E-1G는 그냥 pdf 참조.

EX1-EX3은 그냥 해보면 됨.





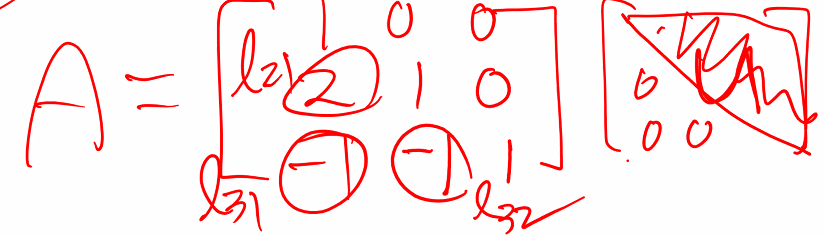
위 경우와 같이 Elementary Matrix에서 같은 column에서는 어떤 숫자가 들어가든 두 행렬의 곱에서 EF == FE가 성립함. (증명해보자.)

EX5는 G와 E가 서로 더해지는 칼럼이 다르기 때문에 곱 순서를 바꾸면 일반적으로 다른 원소가 나옴.

GFE (E에서 F곱하고 G곱하고. 즉 오른쪽에서 왼쪽으로 감) 형태로 곱해지게 되면 Elimination의 순서이고 EFG (혹은 FEG) 형태로 곱해지면 Elementary Matrix들의 원소들이 나옴. -> 깊숙한 곳에 있는 matrix부터 먼저 곱해야 원소들이 보전됨.

E, F, G의 inverse는 각각 multiplier들에 -1을 곱한 값이 됨. 예를 들어 E의 -2는 2가 됨. 대각선 1들은 그대로.





이런 식으로 A를 Lower Triangle Matrix \* Upper Triangle Matrix(LU)의 형태로 나타낼 수 있음. => A = LU = (E^-1F^-1G^-1)U

**1.5 Triangular Factors and Row Exchanges**

초반 부분은 pdf 참고.

Ax = b를 Ux = c로 바꿔서 풀게 됨.

(GFE)^-1 = E^-1F^-1G^-1

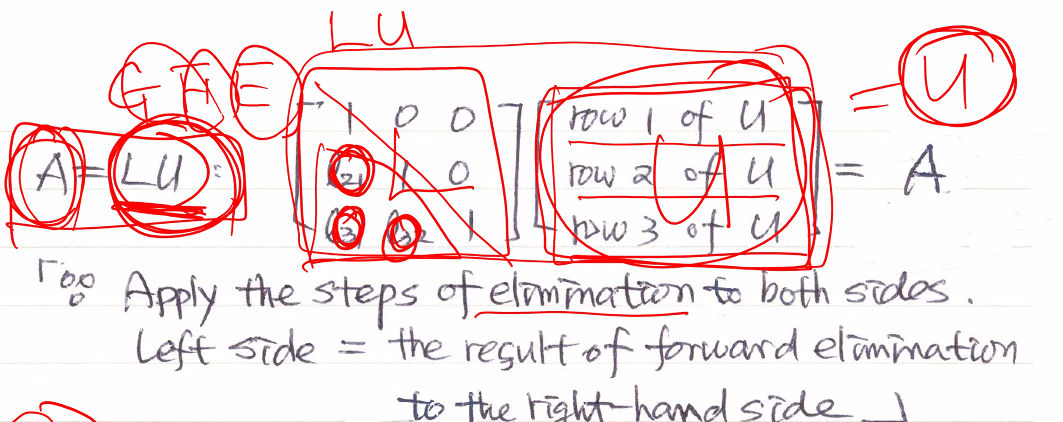
1H – row 순서를 바꾸지 않고 가우스 소거가 성공한다면 A = LU 형태로 나타낼 수 있음.

L은 대각선에 1이 들어가야 하지만, U는 대각선에 1이 아닌 원소가 들어갈 수도 있음.

EX1, EX3은 그냥 pdf 보면 됨.

EX2 – Permutation Matrix [[0, 1], [1, 0]을 곱해 PA = [[3, 4], [0, 2]] 이렇게 바꾸면 가능. 그러면 PA = [[0, 1], [1, 0]]\*[[3, 4], [0, 2]]

Ex4 – A의 대각선 원소가 1이면 L이 되므로 L = L\*I 형태로 나타내어짐. I는 대각선 원소가 모두 1인 행렬.



EX5 – A를 Band Matrix라고 함. 이건 계산해보면 될 듯.

One Linear System = Two Triangular Systems

1. Factor: A = LU 가 되므로 Ax = LUx = Lc = b

2. Solve: 따라서 해당 문제는 Lc = b와 Ux = c의 두 개의 문제로 바뀜. Lc = b는 b에 대해서 가우스 소거를 적용시키는 문제가 됨.

A = LU로 쪼개져 있으면 n^2 연산만에 답을 찾을 수 있음. (기존의 1/3\*n^3 보다 훨씬 낮다.)

Ex6 – Lc = b에서 L은 대각선은 1이고 그 바로 밑에만 -1인 행렬이니까 c에다가 대입하면 c가 바로 구해짐. 그리고 Ux = c에서도 u도 대각선은 1이고 그 바로 위에만 -1이니까 x가 바로 구해짐.

이런 경우에는 계산시간은 2n.

Remark: A = LDU에서 L, D, U는 각각 unique 하다. 즉 저렇게 나눌때 유일하게 나눠진다는 뜻. (이 때 L, U는 대각선이 모두 1인 행렬, D는 대각선에만 원소가 있는 행렬. 원래 기존에 U는 대각선이 꼭 1이 아니어도 됐음)

**LU -> LDU로 갈 때 U = DU에서 D는 왼쪽 U의 대각선 성분을 그대로 가짐. 즉 pdf에서 [[1, 2], [0, -2]] 이므로 D = [[1, 0], [0, -2]] 가 된다. 그리고 여기서 U는 대각선이 모두 1이기 때문에 나눠 가면서 구할 수 있음.**

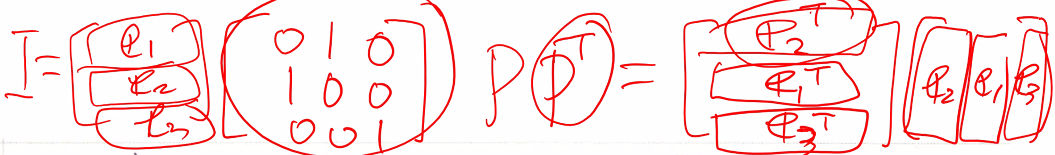
**unique인거 증명하는거 quiz나 숙제에 여러 번 나옴. 매우 중요!!**

**- Row Exchanges and Permutation Matrices**

아까 한 얘기. Pivot에 0이 나오면 permutation matrix를 곱해서 순서를 바꿔서 진행하면 됨.

iJ – nonsingular case에서는 P를 곱해서 pivot에 0이 나오지 않게 할 수 있고 Ax=b는 유일한 해답을 가지게 됨. Row를 재배치하는 것을 알고 있었다면 PA = LU로 변환할 수 있음.

Singular case에서는 어떠한 P를 곱해도 가우스 소거가 실패함. (불능 or 부정)



위의 과정에 의해 PP^t = I가 나오는데 따라서 P^t = P^-1이 됨.

현실에서 pivot이 0에 가까우면 roundoff error를 줄이기 위해 row를 교환함.

P를 왼쪽에 곱하면 row 교환, 오른 쪽에 곱하면 column 교환.

Ex7 – 2,2가 0이 되므로 row 교환함. 그래서 PA = LU 가능해짐.

**1.6 Inverses and Transposes**

Inverse Matrix: b = Ax => A^-1b = x면 A^-1Ax = x

모든 매트릭스가 인버스가 있진 않음. Ax = 0 그리고 x != 0이면 inverse가 불가능.

**Det(A) != 0 이면 인버스가 존재한다. (근데 이건 너무 쉬워서 이거 쓰면 반칙. 감점함)**

1k: 인버스 행렬 B를 앞에 곱해도 뒤에 곱해도 I가 나옴. 따라서 인버스는 유일해서 A^-1로 나타냄. 증명은 pdf 참조.

AB = I 면 BA = I 증명은 <https://math.stackexchange.com/questions/3852/if-ab-i-then-ba-i> 여기 참조.

**Note**

1. 가우스 소거할 때 n개의 pivot을 생성할 때만 inverse가 존재. (row 변환도 허용.) PA = LU

Elimination은 Ax=b를 A^-1를 구하지 않고 푸는 과정. A^-1을 구하는 문제는 훨씬 복잡함.

A^-1을 구하는 문제는 Ax\_i = ci (ci는 i번쨰만 1 나머지는 0인 벡터)를 n번 구함. 가우스 소거 n번 해야 함.

1. Left inverse와 Right inverse는 같음. 결합 법칙 이용해서 증명 가능. B(AC) = (BA)C => BI = IC => B = C
2. A가 invertible이면 Ax = b의 유일한 솔루션이 x = A^-1b.

하지만 계산으로 역행렬을 구하는 것은 무식함. 수학적으로는 사실.

1. **(중요!)** 0이 아닌 벡터 x가 있을 때 Ax = 0이 되면 A는 inverse를 가질 수 없음.

A^-1Ax = 0 => x = 0이 되기 때문에 모순임.

**많은 문제를 이 성질을 이용해 증명할 것!**

1. 행렬 A가 invertible 하면 det(A) != 0.

[[a, b], [c, d]]^-1 = (1/ad-bc)[[d, -b], [-c, a]] 이기 때문

1. A가 대각선에만 0이 아닌 원소가 있는 행렬일 때 A^-1은 원소를 역수 취한 값의 행렬.

A^-1A = I가 나오기 때문.

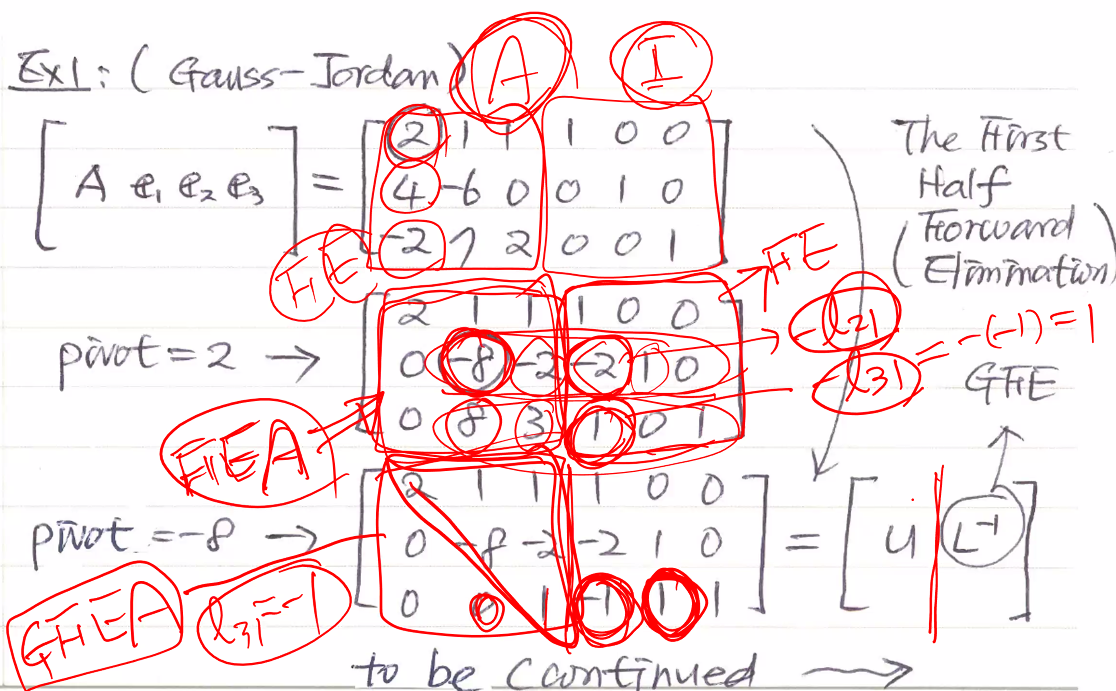
1L: A와 B가 invertible 하면 AB도 invertible 하고, (AB)^-1 = B^-1A^-1. 증명은 pdf 참조.

**A^-1의 계산: Gauss-Jordan Method**

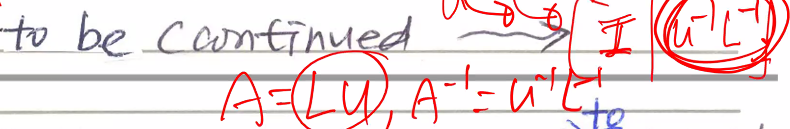


A^-1을 구할 때 Ax\_i = e\_i로 나타낼 수 있음. 이 때 e\_i는 i번째만 1인 벡터, x\_i는 A^1의 i번째 column.

Ex1: e1, e2, e3을 한번에 가우스 소거.



GFEA = U => (GFE)^-1GFEA = (GFE)^-1U => A = LU 이므로 L = (GFE)^-1이고 L^-1 = GFE이 되는데, 이 때 I에 GFE를 곱하기 때문에 오른쪽이 L^-1이 됨.



계속하면 저런 식으로 I와 A^-1로 나오게 됨.

A^-1이 되는 이유 => A = LU 로 factorization 될 때, 결과적으로는 forward substitution에서 A(좌측)와 I(우측)에 L^{-1}를 곱하게 되고, back substitution에서 U^{-1}를 곱하게 되어 좌측은 U^{-1}L^{-1}A = I가 되고, 좌측은 U^{-1}L^{-1}I = (LU)^{-1}I = A^{-1}I = A^{-1}가 되는듯 합니다

나오게 하려면 이제 마지막 pivot에서부터 위로 올라가면서 계산.

그리고 대각 행렬이 되면 I로 만들기 위해 나눠줌. 그 뒤 오른쪽의 값이 A^-1가 됨.

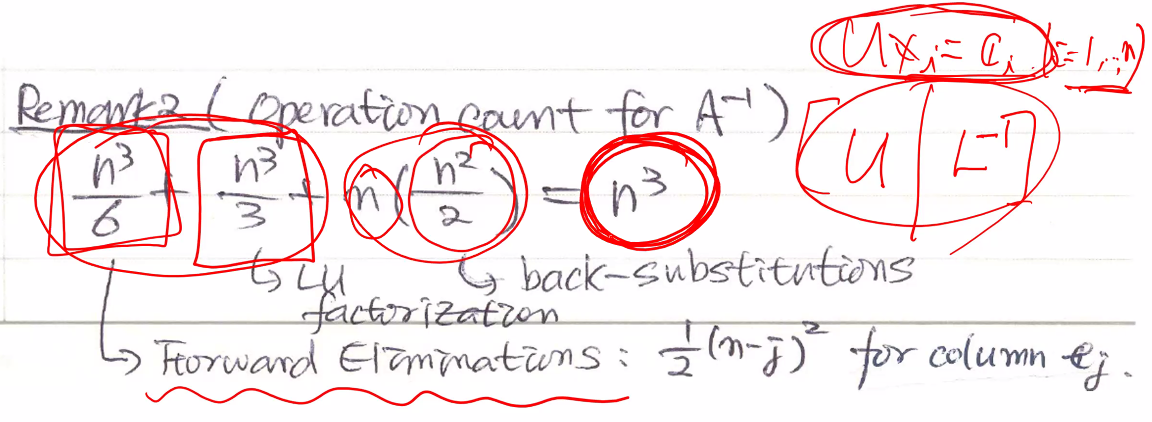
Det는 pivot들의 곱. 이 경우엔 2\*-8\*1 = -16

**Remark 1: Ax = b를 구할 때 A^-1을 구하지 마라.**

Ax\_i = e\_i 계산 시간이 x^3 이라서 1/3\*n^3 의 3배임.

Two triangular steps가 나음.

**Remark 2: A^-1 계산 시간**

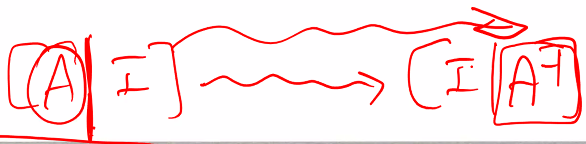


Forward Elimination이 n^3/6인 이유는 0을 원소로 가지고 있을 때 계산을 안 해도 되는데 이게 반 정도 되기 때문.

**난해함**

Invertible = Non-singular (n개의 pivot)

1. Nonsingular => invertible



이 때 A 에 곱해진 GFE.. 등등이 바로 A^-1이 됨. 수학적으로는 Left inverse = Right inverse.

따라서 nonsingular면 invertible 하다가 증명됨.

1. 반대로 invertible => nonsingular

설명은 pdf 참조.

**직접 한번 해보자.**

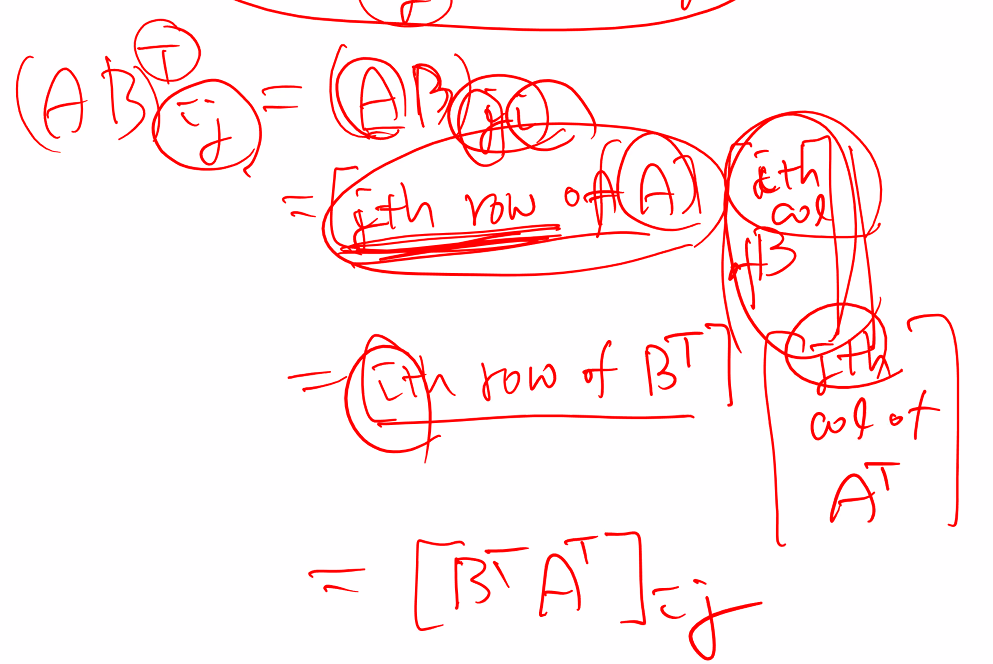
**The Transpose Matrix**

A^t = (A^t)\_ij = A\_ji

1M: 1. (AB)^T = B^TA^T

2. (A^-1)^T = (A^T)^-1

1번, 2번의 교과서 증명은 pdf에 있음. 1번의 더 간단한 증명은 밑에.



**Symmetric Matrices**

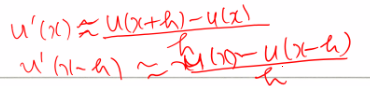
A^T = A이면 symmetric matrix이다. 이러면 A는 무조건 square matrix.

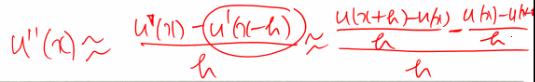
Symmetric matrix가 항상 invertible은 아니다. 하지만 Symmetric Matrix의 inverse A^-1가 존재하면 그 A^-1은항상 symmetric 하다.

1N: Symmetric matrix가 LDU factorization이 되면 U = L^T가 됨.

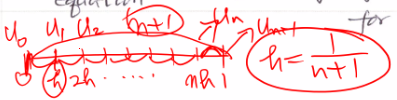
Elimination 과정 중 더 작은 matrix들도 모두 symmetric. 따라서 계산 시간을 반으로 줄일 수 있음.

**1.7 Special Matrices and Applications**



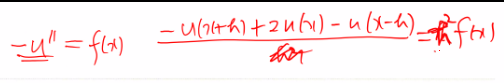


여기서 u’’(x) = ~~ 이거 도출.





여기서 -u\_j+1 + … = h^2\*f(jh) 도출



u\_j는 x = jh일 때.

J = 1부터 n까지를 행렬로 나타내면

J=1일 때 0과 j=5일 때 6은 boundary condition에 의해 0.

Tridiagonal : 대각선 기준 위, 아래 합쳐서 3개가 0이 아니라서 tridiagonal

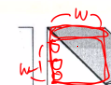
Positive definite: 가우스 소거 했을 때 pivot들이 양수만 나옴.

A = LDU로 나타낼 때 symmetric이므로 LDU = LDL^t로 나타낼 수 있음. 그리고 pivot을 보면 Positive definite.

L, U는 대각선 밑 다른 1줄만 0이 아니라서 bidiagonal.

Band matrix는 대각선 기준으로 일정 범위에만 0이 아닌 원소가 존재.

Tridiagonal이라면 w = 2

 따라서 각각 column당 w(w-1) 계산.

W를 n으로 잡고 대충 계산하면 n^3인데 너무 큼. <- 엉성한 계산.

실제로는, 

w=n일 때 ~= n^3/3

n, n-1, n+1은 연속된 3개의 수이므로 3의 배수는 최소 1개는 있음. 따라서 P는 무조건 정수.