[수치해석] Linear_equation1

Linear Algebra 사전 지식

행렬 A: M * N matrix

Column space: v가 N차원의 모든 가능한 vector일 때, 모든 가능한 Av의 집합

Rank(A): Column space of A의 dimension

- **Rank(A) ≤ min(M, N)**이다.
- M보다 작아야 하는 이유는 Av가 M차원이기 때문에 Column space의 차원은 M보다 작다.
- N보다 작아야 하는 이유는 Column space의 **특정 벡터는 A의 Linearly independent한 Column vector의 Linear combination**이다. N이 M보다 작은 경우, **N개의 A의 Column vector의 Linear combination이기 때문에 Rank(A)는 N (< M)보다 작거나 같은 차원**을 가진다.
    - N개의 Vector의 Linear combination으로 만들 수 있는 차원은 최대 N차원이다.
    - Av를 결과로 나오는 벡터가 아니라 N개의 Vector의 Linear combination가 Span하는 공간 자체로 보면 각 Vector가 방향을 가진 선 하나, **Space는 각 Vector를 축으로 하는 Space**로 이해할 수 있다.
    - 각 Vector가 Basis vector가 된다.

Av 벡터는 M개의 Element를 갖기 때문에 기본적으로 Av 벡터는 M차원 공간에 존재한다.

하지만, A의 Column space를 이용하면 **Av 벡터가 Rank(A)차원 공간 (Rank(A) 차원은 M차원 공간의 Subspace)에 존재한다고 특정**할 수 있다.

이제, **Matrix A를 Linear transformation** 관점에서 설명할 수 있다.

Matrix A는 N차원 공간에 속하는 벡터 v를 M차원 공간에 속하는 벡터 Av로 변환시킨다.

- 이때, 변환된 벡터 Av가 실제로 존재할 수 있는 영역은 M차원 공간 전체가 아니라, 그 안의 더 작은 부분 공간인 **Rank(A)차원의 Column Space**로 한정된다.
- **Matrix A는 N차원 공간에 존재하는 벡터를 M차원 공간에 존재하는 벡터로 변환시켜주는 함수**이다.
- 결과로 나오는 **M차원 공간의 존재하는 벡터의 실제 활동 공간은 Rank(A)차원 공간**이다.

이 장의 목표는 **Ax = b**라는 Linear equation을 해결하는 것이다.

Ax = b라는 방정식은 본질적으로 **Input space (N 차원)에서의 어떤 벡터가 Linear transformation (Function) A를 거친 후에 벡터 b에 도달 할 수 있도록 하는 Input space (N 차원)에서의 벡터 x를 찾는 것**이다.

- Ax의 결과인 **벡터 b가 실제 존재하는 공간인 A의 Column space (Rank(A) 차원)에 반드시 존재**해야 한다.

복잡한 비선형 문제를 해결하는 방법 중 **Non-linear fucntion을 Linear function으로 근사**하여 해결하는 방법이 있다.

- 이전에 N-R Method와 같은 방법이 근사하여 해결하는 방법이다.
- N-R method는 1차 근사이다. **고차 근사로 가면 $f'(x)$ 대신 $J(A)$라는 Jacobian matrix를 사용하여 근사한다.**
- 이 관점에서 **Matrix는 Linear approximation을 수행하는 함수**이다.

Diagonal dominance

행렬 A에서 각 행의 **대각 원소가 각 행의 나머지 원소들의 모든 합보다 큰** 경우

$$|a_{ii}| >= \sum_{j=1, j!=i} a_{ij}$$

Matrix multiplication

$$c_{ij} = \sum_{k=1} a_{ik} b_{kj}$$

Determinant

Square matrix A에 대해서만 정의되고 Det(A)로 표시하며, 계산 방식은 아래와 같다.

$$|\boldsymbol{A}| = \det(\boldsymbol{A}) = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ & \vdots & & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix}$$

$$= \sum_{j=1}^{n} a_{ij}(-1)^{i+j} M_{ij}$$

where

i : fixed

M_{ij}: minor. Determinant of $(n-1)\times(n-1)$ matrix

※ Cofactor $\alpha_{ij} = (-1)^{i+j} M_{ij}$

|det(A)|는 **행렬 A의 Column vector들이 나타내는 도형의 부피(넓이)**이다.

동시에 **|det(A)|**는 **특정 도형의 부피와 해당 도형에 A를 이용한 linaer transformation의 결과로 생기는 도형의 부피의 비율이다.**

- (변환 후 도형의 부피) / (변환 전 도형의 부피)

det(A)의 부호가 (+)라면 변환 전 후, 도형의 방향이 유지되는 것이다.

det(A)의 부호가 (-)라면 변환 전 후, 도형의 방향이 반대가 되는 것이다.

Hadamard's inequality

$$|det(A)| \leq ||a_1||||a_2||...||a_n||$$

- ||a_i||: 행렬 A의 i번째 row의 Euclidean length
- 우변의 결과는 A의 각 행의 길이를 갖는 직면체의 부피이다.

- 좌변의 $|\det(A)|$는 **Column vector들이 이루는 평행체의 부피이므로 직면체의 부피보다 항상 작을 수 밖에 없다.**

Augmented matrix

$$[\ \boldsymbol{A} : \boldsymbol{b}\ ] = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ & \vdots & & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \end{bmatrix}$$

Elementary row operations: Ax = b의 solution을 변경하지 않는 연산

1. 두 행의 순서를 바꿈
2. 한 행에 상수를 곱합
3. 한 행의 상수배를 다른 행에 더함

Cramer's rule

A가 Square matrix이고, det(A) ≠ 0일 때 Ax = b의 Unique solution을 구하는 방법

$$\bar{\boldsymbol{x}} = \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \vdots \\ \bar{x}_n \end{bmatrix} \quad \text{where} \quad \bar{x}_j = \frac{\det(\ \boldsymbol{A}_j\ )}{\det(\ \boldsymbol{A}\ )}$$

- det(A_j)는 **A의 j번째 Column vector를 b vector로 바꾼 행렬에 대해 Determinant를 계산한 값**이다.

Substition

Back Substitution

Upper triangular matrix

$$x_n = \frac{b_n}{a_{nn}}$$

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j=i+1}^{n} a_{ij}x_j\right) \quad i = n-1,\ n-2,\ \cdots, 2,\ 1$$

Forward Substitution

Lower triangular matrix

$$x_1 = \frac{b_1}{a_{11}}$$

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j\right) \quad i = 2,\ 3,\ \cdots,\ n$$

- x_i를 위 → 아래, 아래 → 위 방향으로 해결하는 방법이다.
- Lower triangle matrix 경우: x_1 ~ x_n 방향으로 구해야 하기 때문에 Forward substitution
- Upper triangle matrix 경우: x_n ~ x_1 방향으로 구해야 하기 때문에 Backward substitution

---

Ax = b 형태의 방정식을 풀 때, A가 M * N 행렬이라면 N개의 미지수, M개의 방정식을 갖는 형태라고 생각할 수 있다.

- 이때, A는 **Coefficient matrix**이다.

M, N의 크기에 따라 **문제 유형을 두 가지**로 세분할 수 있다.

1. **Over-determined** (M > N)
   - **방정식의 개수가 미지수의 개수보다 많은** 경우이다.

- 이 경우에는 **정확한 해가 존재하지 않는다.**
- **Least square** 등을 이용하여 정답에 가장 근접한 해를 찾는다.

2. **Under-determined** (M < N)

- **미지수의 개수가 방정식의 개수보다 많은** 경우이다.
- 이 경우에는 **해가 무한히 많이 존재**한다.
- **Norm이 가장 작은 해**를 사용한다.

Ax = b를 푸는 방법은 크게 **두 가지**로 분류할 수 있다.

- Direct method
- Iterative method

Direct methods

- 한번의 분해, 계산 등으로 **바로 해를 구함**

1. Gauss Elimination

과정

Step1: Gauss Reduction

- Forward elimination
- Coefficient matrix → Upper triangular matrix

Step2: Back Substitution

공식

$$m_{ki} = \frac{a_{ki}}{a_{ii}}$$

- a_{ii}: Pivot coefficient

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - m_{ik}a_{kj}^{(k-1)}$$

- Update 공식

문제점

$m_{ki} = \frac{a_{ki}}{a_{ii}}$ 에서 a_{ii}가 너무 작으면 m_{ki} 너무 커진다.

Round-off error가 누적된다.

a_{ii}가$a_{ii}m_{ki}$ 이 큰 경우에 Pivoting 없이 **Gauss Elimination을 진행하면 Error propagtion에 의해 Error가 누적**된다.

- **Pivoting staregy**가 필요하다.
- Pivoting 이후에 Gauss Reduction을 진행한다.

Pivoting staregy

1. **Partial pivoting**

   Pivot coefficient으로 사용하고자 하는 **열의 가장 큰 값을 Pivot coefficient**으로 사용한다.

   $$|a_{pi}| = max_{i<=k<=n}|a_{ki}|$$

   - 이후 p 행과 i 행을 교체한다.

2. **Scaled partial pivoting**

   (1)의 Pivoting은 **행의 값 Scale 자체가 큰 경우**를 반영하지 못 한다.

   $a’_{ii}$를 결정할 때, 각 행의 a_{pi}을 그 행의 최댓값 s_{pi}로 나눈 $a’_{pi}$를 기준으로 아래 수식을 적용한다.

   $$|a'_{pi}| = max_{i<=k<=n}|a'_{ki}|$$

   - 이후 p 행과 i 행을 교체한다.

Complexity

$$\text{Multiplications/divisions:} \quad \frac{n^3}{3} + n^2 - \frac{n}{3}.$$
$$\text{Additions/subtractions:} \quad \frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6}.$$

- n: 행의 개수

단점

Gauss elimination은 너무 오래걸린다.

A가 Singular matrix인 경우에 Ax =b를 해결할 수 없다.

2. Gauss-Jordan Elimination

Augmented matrix $[A : b]$를 $[I : x]$로 만든다.

- Augmented matrix에 대해 **Gauss - reduction을 한 후, 모든 Off-diagonal elements를 0으로 만드는 과정**을 추가한다.

Gauss Elimination보다 **50% 더 많은 계산을 요구한다**.

Inverse matrix를 구하는 데 효율적이다.

- $[A : I]$ → $[I : A^{-1}]$

3. LU Decomposition

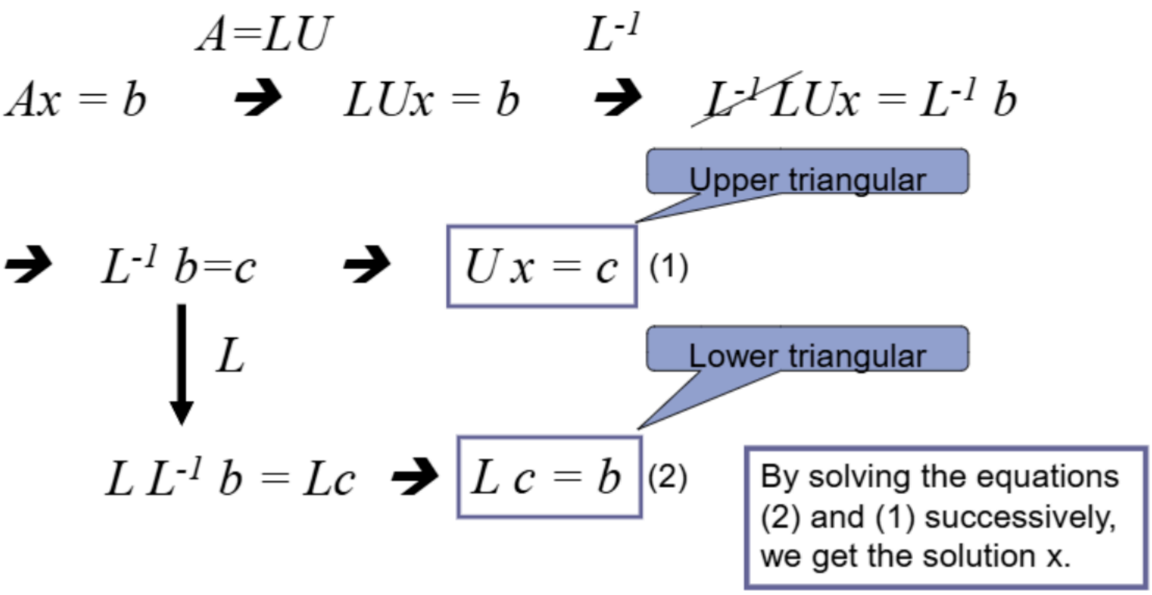
Ax = b의 방정식을 풀 때, **A를 LU로 분해한 후에 해결**하는 방법

- L: Lower triangular matrix

- U: Upper triangular matrix

$$Ax = b$$

과정



이후 $Lc = b$를 해결하고 $Ux = c$를 해결한다.

L, U의 형태의 따라 세 가지로 나뉜다.

- Doolittle decomposition
  - $L_{ii}=1$, for all i
- Crout decomposition
  - $U_{ii}=1$, for all i
- Cholesky decomposition
  - $L_{ii}=U_{ii}$
  - Appropriate for symmetric, positive-definite matrix

그 중 **Crout method에 대해 자세히 살펴보자.**

- NR.c에서 `ludcmp` 가 Crout method도 LU 분해를 진행한다.
- `ludcmp(a, N, indx, &d)` 에서 a에 좌하단은 L이, Diagonal element는 L의 diagonal element, 우상단은 U의 값이 들어간다.
- **Upper matrix**의 **Diagonal element가 전부 1**이다.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ & \vdots & & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ & \vdots & & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & \cdots & u_{1n} \\ 0 & 1 & \cdots & u_{2n} \\ & \vdots & & \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

$$\begin{aligned} l_{i1} &= a_{i1} && (i = 1, 2, \cdots, n) \\ u_{1j} &= \frac{a_{1j}}{l_{11}} && (j = 2, 3, \cdots, n) \\ l_{ij} &= a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} && (j \le i,\ \ i = 2, 3, \cdots, n) \\ u_{ij} &= \frac{1}{l_{ii}} \left[a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \right] && (i < j,\ \ j = 3, 4, \cdots, n) \end{aligned}$$

- Crout decomposition에서 j번째 열을 계산할 때, **L의 j번째 열 부분을 먼저 계산**해야만 그 **대각 원소인** l_{jj}를 알 수 있고, 이 값을 나눠줘야 **U의 j번째 행 부분을 계산**할 수 있습니다.

LU Decomposition 역시 Decomposition 과정에서 **Gauss-Elimination과 같은 행연산을 사용하기 때문에 Singular matrix인 경우에 해를 찾을 수 없다.**

4. Singular Value Decomposition

Gauss Elimination과 LU decomposition이 A가 Singular matrix인 경우에 처리할 수 없는 것과 다르게, **A가 Singular matrix 인 경우에도 Solution을 찾을 수 있는 방법**

행렬 A를 $A = UWV^T$로 분해한다.

- $r = Rank(A)$라고 하자.
- **U: Column-orthogonal matrix (회전)**
    - 첫 r개의 Column vector ($u_1, ..., u_r$): **A의 Column space**에 대한 **Orthogonal basis vectors**이다.
    - 나머지 (m-r)개의 Column vector ($u_{r+1}, ..., u_m$): **A의 Left null space** ($N(A^T)$)에 대한 **Orthogonal basis vectors**이다.
- **W: Diagonal matrix (Scaling)**
    - Sigular value배 된다.
- **V: Orthogonal matrix (회전)**
    - 첫 r개의 Column vector ($v_1, ..., v_r$): **A의 Row space**에 대한 **Orthogonal basis vectors**이다.
        - SVD의 형식에서 V가 Transpose 되어져 있기 때문에 Row space의 Basis라고 생각하면 된다.
    - 나머지 (n-r)개의 Column vector ($v_{r+1}, ..., v_n$): **A의 Null space**에 대한 **Orthogonal basis vectors**이다.

1. 첫 r개의 U 열벡터 ($u_1, ..., u_r$)가 A의 열공간(Column Space)의 기저인 이유

SVD의 핵심 관계식인 $Av_i = \sigma_i u_i$ 에서 시작합니다. 랭크(rank) r까지는 특이값 σ_i가 0이 아니므로, 식을 다음과 같이 바꿀 수 있습니다.

$$u_i = \frac{1}{\sigma_i}(Av_i)$$

이 식의 의미는 다음과 같습니다.

- Av_i**는 A의 열벡터들의 선형 결합입니다.** 행렬-벡터 곱셈의 정의에 따라 Av_i는 A의 열공간($Col(A)$) 안에 있는 벡터입니다.
- 따라서 u_i**는 열공간에 속한 벡터**입니다. (u_1부터 u_r까지 모두)
- 행렬 U는 직교 행렬이므로, U의 열벡터들인 $u_1, ..., u_r$은 서로 **정규직교(orthonormal)**하며 선형 독립입니다.
- 열공간의 차원은 정의에 따라 랭크(rank)인 r입니다.

결론적으로, 열공간($\mathbb{R}^m$의 부분공간)의 차원(r)과 동일한 개수의 정규직교 벡터들($u_1, ..., u_r$)을 열공간 안에서 찾았으므로, 이들은 **A의 열공간에 대한 정규직교기저**가 됩니다.

2. 나머지 m-r개의 U 열벡터 ($u_{r+1}, ..., u_m$)가 A의 좌측 영공간(Left Null Space)의 기저인 이유

좌측 영공간($N(A^T)$)은 $A^T x = 0$을 만족하는 모든 벡터 x의 집합입니다.

SVD 식 $A = U\Sigma V^T$를 전치(transpose)하면 $A^T = V\Sigma^T U^T$가 됩니다. 이 식의 양변에 u_i를 곱해봅시다.

$$A^T u_i = V\Sigma^T(U^T u_i)$$

- U가 직교 행렬이므로 $U^T U = I$ 입니다. 따라서 $U^T u_i$는 i번째 위치만 1이고 나머지는 0인 표준 기저 벡터 e_i가 됩니다.
- $V\Sigma^T e_i$를 계산하면, Σ^T는 i번째 대각 성분만 σ_i이고 나머지는 0이므로, 결과적으로 $\sigma_i v_i$가 됩니다.
- 따라서 $A^T u_i = \sigma_i v_i$ 입니다.

이제 **랭크(rank)보다 큰 인덱스 $i > r$**인 경우를 생각해봅시다. 이때 특이값 σ_i는 0입니다.

$$A^T u_i = 0 \cdot v_i = 0$$

이 식은 $u_{r+1}, ..., u_m$ 벡터들이 $A^T x = 0$을 만족시킨다는 것을 의미하며, 이 벡터들은 **좌측 영공간에 속합니다.** 또한 이 벡터들은 U의 일부이므로 서로 정규직교합니다. 좌측 영공간의 차원은 $m - r$이므로, 이 $m - r$개의 벡터들은 **A의 좌측 영공간에 대한 정규직교기저**가 됩니다.

3. 첫 r개의 V 열벡터 ($v_1, \dots, v_r$)가 A의 행공간(Row Space)의 기저인 이유

행공간($Row(A)$)은 A^T의 열공간($Col(A^T)$)과 같습니다. 위에서 유도한 관계식 $A^T u_i = \sigma_i v_i$를 사용합니다. 랭크(rank) r까지는 σ_i가 0이 아니므로 식을 다음과 같이 바꿀 수 있습니다.

$$v_i = \frac{1}{\sigma_i}(A^T u_i)$$

이 식의 의미는 다음과 같습니다.

- $A^T u_i$**는** A^T**의 열벡터들의 선형 결합입니다.** 즉, A^T의 열공간에 속하며, 이는 곧 A의 행공간에 속한다는 의미입니다.
- 따라서 v_i**는 행공간에 속한 벡터**입니다. (v_1부터 v_r까지 모두)
- 행렬 V는 직교 행렬이므로, V의 열벡터들인 $v_1, \dots, v_r$은 서로 **정규직교(orthonormal)**하며 선형 독립입니다.
- 행공간의 차원은 정의에 따라 랭크(rank)인 r입니다.

결론적으로, 행공간($\mathbb{R}^n$의 부분공간)의 차원(r)과 동일한 개수의 정규직교 벡터들($v_1, \dots, v_r$)을 행공간 안에서 찾았으므로, 이들은 **A의 행공간에 대한 정규직교기저**가 됩니다.

4. 나머지 n-r개의 V 열벡터 ($v_{r+1}, \dots, v_n$)가 A의 영공간(Null Space)의 기저인 이유

영공간($N(A)$)은 $Ax = 0$을 만족하는 모든 벡터 x의 집합입니다. SVD의 핵심 관계식 $Av_i = \sigma_i u_i$를 다시 사용합니다.

이번에는 **랭크(rank)보다 큰 인덱스 $i > r$**인 경우를 생각해봅시다. 이때 특이값 σ_i는 0입니다.

$$Av_i = 0 \cdot u_i = 0$$

이 식은 $v_{r+1}, \dots, v_n$ 벡터들이 $Ax = 0$을 만족시킨다는 것을 명확하게 보여줍니다. 즉, 이 벡터들은 **영공간에 속합니다.** 또한 이 벡터들은 V의 일부이므로 서로 정규직교합니다. 랭크-보수성 정리(Rank-Nullity Theorem)에 따라 영공간의 차원은 $n - r$이므로, 이 $n - r$개의 벡터들은 **A의 영공간에 대한 정규직교기저**가 됩니다.

SVD에 의해 분해되는 $A = UWV^T$**는 거의 유일**하다.

- Ran
- U:

- V: A의 Null space의 **Orthogonal basis vectors, 특이값이 0인 열에 대응되는 V의 Column vector**
- W: A^TA의 Eigen values의 제곱근

A의 Square matrix인 경우 **A의 역행렬을 아래 공식**으로 구할 수 있다.

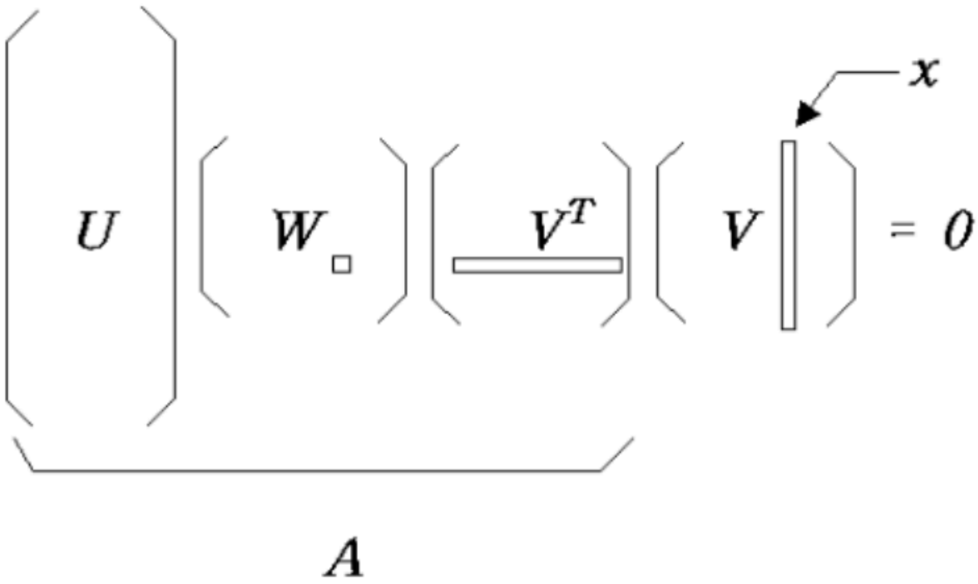
$$A^{-1} = V \cdot [diag(\frac{1}{w_j})] \cdot U^T$$

Homogeneous equation ($Ax = 0$)

A가 Non-Singular인 경우에는 x = 0 뿐이다.

A가 Singular인 경우에는 **SVD에서 Singular value가 0인 행에 대응되는 V의 열 자체가** x가 된다.

- **Singular value가 0인 행에 대응되는 V의 열은 Null space의 Orthogonal basis**이기 때문이다.



Nonhomogeneous equation ($Ax = b$)

A가 Non-singluar인 경우에는 위에서 본 **A의 역행렬 공식을 이용하여** x를 찾는다.

$$x = V \cdot [diag(\frac{1}{w_j})] \cdot (U^T \cdot b)$$

A가 Singluar인 경우에는 위에서 본 A의 역행렬 공식을 응용하여 x를 찾는다.

$$x = V \cdot [diag(\frac{1}{w_j})] \cdot (U^T \cdot b)$$

- w_j: Singular인 경우 Singular value w_j가 0이기 때문에 위 경우에서 $diag(\frac{1}{w_j})$ 대신 0을 사용하는 (Pseudo-inverse, A^+)를 사용한다.

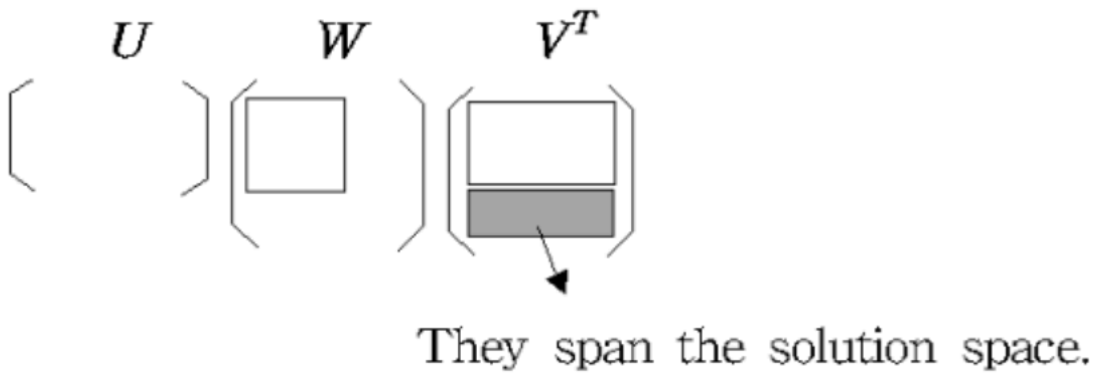
SVD를 통해 x를 찾는 방법은 **Over-determined와 Under-determined 상황에서도 Solution을 찾을 수 있다.**

- **Over-determined와 Under-determined 상황에서** SVD를 이용하여 해를 찾는 과정은 아래 원리를 갖는다.
- **Over-determined** 상황에서는 Least square를 이용하여 가장 유사한 Solution을 찾는다.
- **Under-determined** 상황에서는 Min-norm solution을 찾는다.

Under-determined (M < N)

SVD의 결과에서 V^T에서 Variable이 남는다.

- 남는 **Variable이 Span하는 Space가 있기 때문에 Solution이 무한**하다.
- V^T에서 흰색 부분을 v라고 하자.
- 남은 부분은 A의 Null space에 대응되는 부분으로, 회색으로 칠해진 부분의 벡터를 $v' = [v_{r+1}, \ldots, v_n]$이라고 하자.
- SVD 형식이 아니라, 간단하게 생각해서 $Av = A(v + v') = 0$**일 때, 어떤** v'**이던지 에 상관없이** v**로만 Solution을 찾을 수 있다.**
    - 때문에 v'**이 Free variable**이 되어 해가 무한하게 된다.



Over-determined (M > N)

- 보통 Non-singular이다.
- SVD Ax = b에서 **b가 A의 Column space 위에 존재하지 않는다면 b 대신 Least square로 구한 b′을 이용하여 Ax = b′**을 해결한다.
    - 만약 드물지만 **Ax = b′의 Solution이 무한한 경우 동일하게 최소 노름을 갖는 Solution**을 사용한다.

Iterative methods

- 초기값에서 시작해 Residual을 줄이며 해에 수렴하도록 하는 방법

True value: x, b

추정치: $\hat{x}, \hat{b}$

Error of x: True solution - 추정치 = $x - \hat{x}$ = δx

Error of b: True solution - 추정치 = $b - \hat{b}$ = δb

위 정의에 따르면 아래 공식이 유도된다.

$A(x + \delta x) = b + \delta b$

- $Ax = b$이기 때문에 $A\delta x = \delta b$ 이다.

$A\delta x = \delta b$ 를 풀면 $\delta x = A^{-1}\delta b$을 얻을 수 있고 **Next x = x +** δx**로 업데이트하여 더 좋은 값**을 얻을 수 있다.

- 이 과정을 수렴할 때까지 계속 반복한다.

기본적으로 **LU Decomposition을 이용**하여 x를 찾아놓고 그 x**를 초기값으로 하여 Iterative method를 진행**한다.

- 한 번의 Decomposition으로 δx씩 증가시키며 확인한다.

ill-conditioning 측정

- ill-condition은 **입력이 조금 바뀔 때 출력이 크게 변해 불안정한 상황**을 의미한다.

1. 행렬 A를 Normalization 한다.
    a. 각 행의 최댓값을 1으로 만든다.
2. Normalized A′의 **Inverse** A'^{-1}**의 Norm이 굉장히 크면 ill-condition**이다.

계산에서 $\delta x = A^{-1}\delta b$을 사용하기 때문에 A^{-1}**의 크기가 크면 불안정**해지기 때문이다.

- $\delta x = A^{-1}\delta b$의 식에서 A^{-1}**의 크기가 크면 Residual의 크기와 상관없이** δx**가 커져** 불안정 해진다.