Lecture15. Projections onto Subspaces

▼ 목차

1. 해가 존재하지 않는 선형연립방정식 Ax=b (Solving Ax=b when there is no solution)

- Lecture8에서 어떤 시스템의 선형방정식인 A**x**=b 를 만족시키는 해 **x**를 구할 때, b가 행렬 A의 column space상에 존재한다면 이 선형시스템은 가해 조건(solvability condition)을 만족하며 해를 구할 수 있다고 배웠다.
- 만약 b가 가해 조건을 만족하지 않는다면? 즉, b가 A의 column space상에 존재하지 않는다면 어떻게 될까? 가해 조건을 만족하지 않아 **해가 존재하지 않는 경우** 말이다. 이러한 경우에도 해를 구할 수 있을까?
    - 해가 없기 때문에 아주 정확한 해(exact solution)는 구할 수 없지만 **가장 근사한 해를 구하는 것**이다.
- 행렬 A의 미지수(unknown)보다 방정식(equation)이 더 많은 경우(m>n), rank는 m보다는 당연히 작고 최소 n이 된다. 이 경우 column 벡터의 차원이 rank보다 작기때문에 우변의 벡터 b에 상응하는 해가 없을 가능성이 훨씬 높을 것이다. 이를 **overdetermined**라 한다. 미지수와 방정식의 수가 같은 경우(m=n)를 **determined**, 방정식이 미지수보다 적을 경우(m<n) **underdetermined**라 한다. 그 형태는 아래와 같다.

  $$\textbf{overdetermined} : (m > n) \text{ more equations (No solutions in most cases)}$$

$$Ax = b \rightarrow \begin{bmatrix} 1 & 1 & 2 \\ 3 & 4 & 1 \\ 3 & 7 & 1 \\ 4 & 2 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 11 \\ 2 \\ -7 \\ 21 \end{bmatrix} \ (m = 4 \ > \ n = 3)$$

$$\textbf{determined} : (m = n) \text{ same equ. and unknown (exact solution)}$$

$$Ax = b \rightarrow \begin{bmatrix} 1 & 1 & 2 \\ 3 & 4 & 1 \\ 3 & 7 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 11 \\ 2 \\ -7 \end{bmatrix} \ (m = 3 \ == \ n = 3)$$

$$\textbf{underdetermined} : (m < n) \text{ more unknowns (Infinitely many solutions)}$$

$$Ax = b \rightarrow \begin{bmatrix} 1 & 1 & 2 \\ 3 & 4 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 11 \\ 2 \end{bmatrix} \ (m = 2 \ < \ n = 3)$$

- Overdetermined는 방정식이 미지수보다 많은 경우이며 일반적으로 해가 존재하지 않는다. **해가 존재하는 경우가 있다면 우변의 b벡터가 A의 column space에 존재할 때 해(solution) x가 존재**한다. 위의 첫번째 식에서 overdetermined행렬에 우변 벡터 b를 붙여 Augmented matrix를 만든 다음 소거(elimination)를 통해 row reduced echelon form을 만들면 row4는 영벡터인데, b4의 원소는 0이 아닌 상수가 나온다. 애초에 식 자체가 성립이 안되기 때문에 해가 존재하지 않는다.
- Determined는 미지수와 방정식의 수가 같은 경우이며, full rank일 경우 유일한 해(unique solution)를 갖는다.
- Underdetermined는 미지수가 방정식보다 많은 경우이다. 이 경우엔 무수히 많은 해가 존재한다.

- overdetermined행렬의 소거를 생각해보자. 이때 행렬의 형태는 직사각형(Rectangular)형태일 것이고 바로 위에서 언급한 것처럼 소거했을 때 pivot row아래의 row들, 즉 row4는 [0 0 0]=[1]와 같이 말이 안되는 식이 될 것이다. 소거(Elimination)에 실패한 것이다. 소거의 결과는 우리에 해가 존재하는지, 혹은 존재하지 않는지에 대한 것을 말해준다. overdetermined인 경우에는 소거가 해가 존재하지 않는다고 말해준다. 그럼 어떻게 해야 할까?
- 우선 A^TA**행렬을 이해하는 것이 굉장히 중요**하다. 이 행렬은 우리의 문제를 이해하는데 핵심적인 역할을 하는 행렬이다. 이때 행렬 A는 m by n크기이며 m이 n보다 더 큰, 즉 m>n 의 overdetermined형태이다.
  - A에 A^T를 앞에 곱해주면 **정사각행렬(square matrix)이 된다**.

$$A^TA$$

$$(n \times m)(m \times n) \rightarrow (n \times n)$$

- A^TA의 다른 특징은 바로 **대칭(symmetric)**이다. 대칭 행렬(symmetric matrix)의 특성은 전치를 해도 원래 자기 자신과 똑같은 특성을 가지고 있다.
    (A^TA)^T=A^TA^{TT}=A^TA

- **A^TA는 Full rank일때 역행렬이 존재한다. 역행렬(Inverse matrix)의 조건은 정방행렬이면서 full rank여야 한다**. 따라서 full rank라는 조건만 충족된다면 식 (4)는 역행렬이 존재한다. 사실 full rank가 되기 위한 조건은 원래 행렬의 column이 독립(independent)하면 된다.

> A^TA의 세가지 특징
>
> - Square Matrix
> - Symmetric Matrix
> - Invertible (if Full rank)

- 다시 overdetermined한 경우를 생각해봤을때 식을 정확하게 만족하는 해를 구할 수는 없어도 모든 식을 최대한 만족시키는 해를 구할 수 있다.

$$Ax = b \ (m > n)$$

  - 위의 식은 정방행렬(square matrix)이 아니기 때문에 역행렬을 구할 수 없다. 따라서 해를 구할 수 없다. **식의 양변에 A의 전치(transpose)를 곱해서 최적의 해를 구할 수 있다.**

$$A^TA\hat{x} = A^Tb$$

  - $\hat{x}$은 **식을 정확히 만족시키는 해(solution)는 아니지만, A에 존재하는 모든 방정식(row, b)을 최대한 만족시키는 해,** 즉 최적해를 의미한다.
  - A^TA는 full rank이기만 하면 역행렬이 존재한다고 했다. 따라서 이것의 역행렬을 구하여 양변에 곱해주면 최종적인 해를 구할 수 있을 것이다.

$$\begin{aligned} A^TA\hat{x} &= A^Tb \\ (A^TA)^{-1}A^TA\hat{x} &= (A^TA)^{-1}A^Tb \\ \hat{x} &= (A^TA)^{-1}A^Tb \end{aligned}$$

  - 이렇게 하여 해가 존재하지 않는 Ax=b에 대한 최적해(best solution)를 구할 수 있다.

2. Examples of Ax=b (overdetermined case)

- 앞에서 구했던 최적해를 구하는 방법을 실제로 계산해보자.

1) Best solution for overdetermined equation

$\textbf{overdetermined}$:$(m > n)$ more equations (No solutions in most cases)

$$Ax = b \rightarrow \begin{bmatrix} 1 & 1 & 2 \\ 3 & 4 & 1 \\ 3 & 7 & 1 \\ 4 & 2 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 11 \\ 2 \\ -7 \\ 21 \end{bmatrix} \ (m = 4 \ > \ n = 3)$$

- 위의 식의 해를 구하기 위해 식을 아래와 같이 정리해보자

$$Ax = b \rightarrow$$
$$\begin{bmatrix} 1 & 1 & 2 \\ 3 & 4 & 1 \\ 3 & 7 & 1 \\ 4 & 2 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 11 \\ 2 \\ -7 \\ 21 \end{bmatrix}$$
$$A^T A\hat{x} = A^T b \rightarrow$$
$$\begin{bmatrix} 1 & 3 & 3 & 4 \\ 1 & 4 & 7 & 2 \\ 2 & 1 & 1 & 3 \end{bmatrix} \begin{bmatrix} 1 & 1 & 2 \\ 3 & 4 & 1 \\ 3 & 7 & 1 \\ 4 & 2 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 3 & 4 \\ 1 & 4 & 7 & 2 \\ 2 & 1 & 1 & 3 \end{bmatrix} \begin{bmatrix} 11 \\ 2 \\ -7 \\ 21 \end{bmatrix}$$
$$\begin{bmatrix} 35 & 42 & 20 \\ 42 & 70 & 19 \\ 20 & 19 & 15 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 80 \\ 12 \\ 80 \end{bmatrix}$$
$$(A^T A)^{-1} A^T A\hat{x} = (A^T A)^{-1} A^T b \rightarrow \hat{x} = (A^T A)^{-1} A^T b$$
$$\begin{bmatrix} 0.43 & -0.15 & -0.38 \\ -0.15 & 0.07 & 0.11 \\ -0.38 & 0.11 & 0.43 \end{bmatrix} \begin{bmatrix} 35 & 42 & 20 \\ 42 & 70 & 19 \\ 20 & 19 & 15 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} =$$
$$\begin{bmatrix} 0.43 & -0.15 & -0.38 \\ -0.15 & 0.07 & 0.11 \\ -0.38 & 0.11 & 0.43 \end{bmatrix} \begin{bmatrix} 80 \\ 12 \\ 80 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2.51 \\ -2.85 \\ 5.6 \end{bmatrix}$$
$$A\hat{x} = b \rightarrow$$
$$\begin{bmatrix} 1 & 1 & 2 \\ 3 & 4 & 1 \\ 3 & 7 & 1 \\ 4 & 2 & 3 \end{bmatrix} \begin{bmatrix} 2.51 \\ -2.85 \\ 5.6 \end{bmatrix} = \begin{bmatrix} 10.85 \\ 1.71 \\ -6.85 \\ 21.14 \end{bmatrix} \ comparison\ with\ b = \begin{bmatrix} 11 \\ 2 \\ -7 \\ 21 \end{bmatrix} = Ax$$

- 계산된 결과를 원래의 식과 비교해 봤을 때 약간의 차이가 있지만 그 정도가 크지 않은 것을 알 수 있다.

- 미지수(unknown)보다 방정식(equation)이 많아서 원래 해가 존재하지 않는 overdetermined case의 선형연립방정식에 대한 최적해를 구할 수 있었다. 딱 들어맞는 정확한 해는 아니지만, 각 방정식의 정보를 최대한 반영하여 각 방정식과의 오차(error)의 합을 최대한 줄이는 쪽으로 해를 계산할 수 있었다.

2) when the solution exists in overdetermined equation

- **우변의 벡터 b가 행렬 A의 column space에 존재할 때 overdetermined 방정식이라도 해가 존재한다. 즉, A의 column 벡터들의 선형 조합(Linear combination)으로 b를 만들 수 있을 때를 말한다**.

$$Ax = b \rightarrow$$
$$\begin{bmatrix} 1 & 1 & 2 \\ 3 & 4 & 1 \\ 3 & 7 & 1 \\ 4 & 2 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \\ 9 \\ 3 \end{bmatrix}$$
$$(A^T A)^{-1} A^T A\hat{x} = (A^T A)^{-1} A^T b \rightarrow \hat{x} = (A^T A)^{-1} A^T b$$
$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$$
$$A\hat{x} = b \rightarrow$$
$$\begin{bmatrix} 1 & 1 & 2 \\ 3 & 4 & 1 \\ 3 & 7 & 1 \\ 4 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \\ 9 \\ 3 \end{bmatrix} \; comparison\ with\ b = \begin{bmatrix} 0 \\ 6 \\ 9 \\ 3 \end{bmatrix} = Ax$$

  - b가 A의 column space에 존재할 경우 값이 정확하게 일치하는 것을 볼 수 있다.

3) Not invertible case

- 행렬 A의 column이 dependent할 때는 최적해조차 구할 수 없다. **Overdetermined system(m>n)에서 그나마 최적해라도 구하려면 행렬의 rank가 최소한 n과 같아야한다**. 그러나 rank가 n보다 작을 경우엔 이 조차 불가능하다.

- 다음과 같은 예시를 보자.

$$col1 + col2 = col3$$

$$Ax = b \rightarrow \begin{bmatrix} 1 & 1 & 2 \\ 3 & 4 & 7 \\ 3 & 7 & 10 \\ 4 & 2 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 11 \\ 2 \\ -7 \\ 21 \end{bmatrix} \quad (m = 4 \ > \ n = 3)$$

- 행렬 A의 col3은 col1+col2와 같다. 따라서 <u>column vector는 종속(dependent)</u>이다. A의 앞에 A의 transpose를 곱해보자.

$$A^T A$$

$$\begin{bmatrix} 1 & 3 & 3 & 4 \\ 1 & 4 & 7 & 2 \\ 2 & 7 & 10 & 6 \end{bmatrix} \begin{bmatrix} 1 & 1 & 2 \\ 3 & 4 & 7 \\ 3 & 7 & 10 \\ 4 & 2 & 6 \end{bmatrix} = \begin{bmatrix} 35 & 42 & 77 \\ 42 & 70 & 112 \\ 77 & 112 & 189 \end{bmatrix}$$

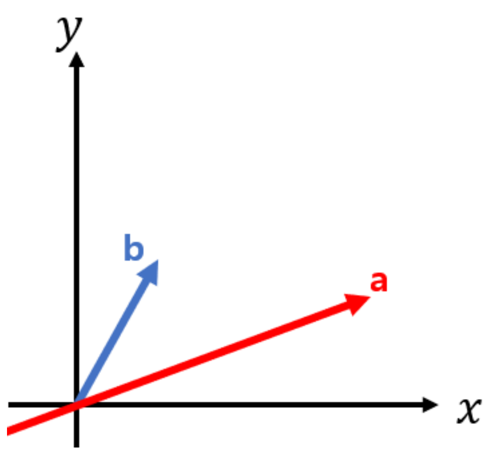
$$col1 + col2 = col3$$

- 연산 결과도 마찬가지로 col1+col2=col3인 것을 볼 수 있다. Full rank가 아니므로 역행렬이 존재하지 않는다. 따라서 행렬 A의 column이 dependent하면 최적해조차 구하는 것이 불가능하다.

- **결론적으로 $A^T A$가 역행렬을 가지기 위해선 반드시 A의 column이 독립(independent)이어야 한다.**

3. 2D-Vector Projection(2차원 벡터 투영)

- **벡터 투영(vector projection)**은 두 개의 벡터 중 하나의 벡터를 다른 하나의 벡터에 투영(projection)시키는 것을 말한다. 투영은 벡터의 관점에서 보면 하나의 벡터를 다른 벡터로 옮겨서 표현하는 것을 말한다.
- 아래의 그림으로 다시 이해해보자

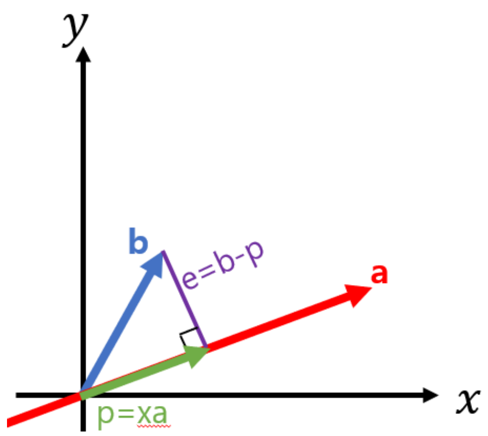


- 위의 그림은 두 벡터 a와 b를 나타내고 있다. 벡터 a는 1차원 공간인 line이며, 우리가 찾고자하는 것은 **벡터 b를 a에 투영시켰을 때 그 점이 벡터 a의 어느 지점에 위**

치할 것인가를 찾는 것이다. **벡터 a라는 1차원 공간의 어느 점이 벡터 b가 가리키는, 즉 벡터 b의 화살표 끝점과 가장 가까운지를 찾는 것**으로 생각할 수 있다.

- **벡터 b의 끝점에서 벡터 a의 수직(perpendicular) 방향으로 선을 그렸을 때 만나는 점**이 가장 가까운 점이다.

- 벡터 b의 끝점에서 벡터 a의 수직 방향으로 직선을 그으면 아래의 그림에서 보라색 선이 된다.



  - 이때 a와 만나는 점이 바로 p이고, 이는 초록색 벡터로 나타내었다. 이 초록색 벡터가 바로 벡터 b를 벡터 a에 투영시킨 벡터이다.
  - 보라색 선은 벡터 b와 p의 차이를 나타내는데, 원래의 벡터 b와 투영시켰을 때의 벡터 p사이의 **오차(error)**를 의미한다. 즉 투영시키기 전과 투영시킨 후가 거리상으로 얼마나 차이가 나는지를 나타내는 것이다. 이 보라색 벡터는 투영의 대상이 되는 벡터인 a와 직교(orthogonal)하며, 식으로는 b에서 p를 뺀 e=b-p로 나타낸다. 이 식을 이항하여 p=의 꼴로 다시 써보면 p=b-e가 된다. <u>벡터 b에서 오차 e만큼 빼면 바로 투영 벡터 p</u>가 되는 것이다.

- 위의 그림을 잘 보면 벡터 p는 벡터 a에 어떤 스케일 상수 x를 곱한 것과 같다. 즉 p=xa이며, 여기서 **우리가 찾고자 하는 것은 바로 스케일 상수 x이다**. 이제 이 x를 어떻게 찾을지 알아보도록 하자.
  - 먼저 가장 중요한 사실 한 가지는 **벡터 a와 e가 수직(perpendicular)**이라는 것이다. 이를 수식으로 나타내면 아래와 같다.

    $$\boldsymbol{a}^T(\boldsymbol{b} - x\boldsymbol{a}) = 0$$

  - bold체로 나타낸 벡터들은 column vector를 나타낸다. a의 transpose는 row 벡터이고 b-xa는 위의 보라색 벡터 e를 나타낸 것이다. 따라서 위의 식은 벡터 a와 벡터 e의 내적(dot product)을 표현한 것이고, 수직이기 때문에 내적의 결과는 0인 것이다.

- 위의 식을 전개하고 우변으로 이항하여 정리해보면 다음과 같다.

  $$\boldsymbol{a}^T(\boldsymbol{b}-x\boldsymbol{a})=0$$
  $$\boldsymbol{a}^T\boldsymbol{b}-x\boldsymbol{a}^T\boldsymbol{a}=0$$
  $$\boldsymbol{a}^T\boldsymbol{b}=x\boldsymbol{a}^T\boldsymbol{a}$$
  $$x=\frac{\boldsymbol{a}^T\boldsymbol{b}}{\boldsymbol{a}^T\boldsymbol{a}}$$

- $\boldsymbol{a}^T\boldsymbol{b},\boldsymbol{a}^T\boldsymbol{a}$는 모두 내적이며 상수이다. 따라서 x는 벡터 a에 곱해지는 스케일 상수가 된다.

- 관점을 다르게 해서 삼각법의 관점으로 정리를 해보면 다음과 같다.

  $$\boldsymbol{p}=(\|\boldsymbol{b}\|cos\theta)\hat{\boldsymbol{a}} \qquad \leftarrow a^Tb=\|a\|\|b\|cos\theta$$
  $$\boldsymbol{p}=(\cancel{\|\boldsymbol{b}\|}\frac{\boldsymbol{a}^T\boldsymbol{b}}{\|\boldsymbol{a}\|\cancel{\|\boldsymbol{b}\|}})\hat{\boldsymbol{a}} \qquad \leftarrow \hat{a}=\frac{a}{\|a\|}$$
  $$\boldsymbol{p}=(\frac{\boldsymbol{a}^T\boldsymbol{b}}{\|\boldsymbol{a}\|})\frac{\boldsymbol{a}}{\|\boldsymbol{a}\|}$$
  $$\boldsymbol{p}=(\frac{\boldsymbol{a}^T\boldsymbol{b}}{\|\boldsymbol{a}\|^2})\boldsymbol{a}$$
  $$\boldsymbol{p}=(\frac{\boldsymbol{a}^T\boldsymbol{b}}{\boldsymbol{a}^T\boldsymbol{a}})\boldsymbol{a}$$
  $$\boldsymbol{p}=x\boldsymbol{a}$$

- 투영 벡터의 식을 선형 대수적으로 표현한 것과 삼각법으로 표현한 것이 일치함을 증명하였다.

1) Projection matrix of n-dimensional vectors

- 위에서 벡터의 투영에 대한 식을 벡터에 대한 식으로 정리하였다. 이제 이것을 행렬에 대한 식, 즉 **투영 행렬(projection matrix)**로 표현해보자.

- **p**=x**a**로 벡터 p에 대한 식을 표현하였다. 여기서 x는 스칼라(scalar) 값이고 x를 먼저 곱하던, a를 먼저 곱하던 같은 값이 나온다. 즉 **p**=x**a** 나 **p**=**a**x나 똑같다는 말이다. 두 번째 방법으로 다시 정리해보면 아래와 같다.

  $$\boldsymbol{p}=\boldsymbol{a}x, x=\frac{\boldsymbol{a}^T\boldsymbol{b}}{\boldsymbol{a}^T\boldsymbol{a}}$$
  $$\boldsymbol{p}=\boldsymbol{a}\frac{\boldsymbol{a}^T\boldsymbol{b}}{\boldsymbol{a}^T\boldsymbol{a}}$$

  - 벡터 b의 길이가 두 배가 되면 투영된 벡터 p의 길이도 두 배가 될 것이다. $\boldsymbol{p}=\boldsymbol{a}\frac{\boldsymbol{a}^T\boldsymbol{b}}{\boldsymbol{a}^T\boldsymbol{a}}$의 b에 2를 곱하면 당연히 벡터 p도 2배가 늘어날 것이다.
  - 이번엔 벡터 a의 길이가 두 배가 된다고 생각해보자. 각 a의 앞에 2씩을 곱해주면 분자의 두 개의 a에도 각각 2, 분모의 두 개의 a에도 각각 2가 곱해질 것이다. 2는 결국 소거되어 식은 원래와 같게 된다. 즉 **a는 아무리 늘어나거나 줄어들어도 p에는 영향을 미치지 않는다.**

- 결국 벡터 b가 어떤 투영시키는 매개체에 의해서 벡터 a로 투영되는 것이다. 여기서 투영시키는 매개체가 바로 **투영 행렬(projection)**이다. 그리고 **투영행렬은 이미 식 $\boldsymbol{p}=$**

$\boldsymbol{a}\frac{\boldsymbol{a}^T\boldsymbol{b}}{\boldsymbol{a}^T\boldsymbol{a}}$**에 나타나있다**. 벡터 b의 a로의 투영을 식으로 나타내면 아래와 같다.

$$\text{projection } \boldsymbol{b} \text{ onto } \boldsymbol{a} \qquad (P: \ proj \ matirx)$$
$$\boldsymbol{p} = P\boldsymbol{b}, \qquad P = \frac{\boldsymbol{a}\boldsymbol{a}^T}{\boldsymbol{a}^T\boldsymbol{a}}$$

(bold체의 소문자 p는 투영된 벡터이고 대문자 P가 투영 행렬)

- $\boldsymbol{p} = \boldsymbol{a}\frac{\boldsymbol{a}^T\boldsymbol{b}}{\boldsymbol{a}^T\boldsymbol{a}}$를 정리하여 P를 구할 수 있다.
- column vector x row vector 순으로 곱하면 행렬이 됨을 배웠다. a는 column vector, a^T는row vector이고, a^Ta는 상수가 된다.
- 따라서 P는 행렬이 되는 것이다. 이것이 **b를 a로 투영시키는 투영행렬(projection matrix)**
  이다. 이것은 n차원 벡터에 대해서도 성립한다.

2) Properties of projection matrix

- 투영행렬의 특징에 대해서 살펴보기 전에 어떤 행렬의 column space에 대해서 한 번 생각해보자.
- 어떤 임의의 행렬 A가 있다고 했을 때 A의 column vector들의 선형조합(Linear combination)을 통해 만들 수 있는 공간을 우리는 행렬 A의 column space라고 한다. 그렇다면 이 행렬 A에 어떤 임의의 벡터 **x**를 곱하면 어떻게 될까? 이것이 의미하는 것이 무엇일까? 바로 **A에 곱해진 그 벡터 x가 A의 column space에 안착(landing)하는 것**이다. 아래의 예를 보고 이해해 보도록 하자.

$$\underset{A}{\begin{bmatrix}1 & 1\\2 & 3\end{bmatrix}} \underset{x}{\begin{bmatrix}4\\7\end{bmatrix}} = 4\underset{col1}{\begin{bmatrix}1\\2\end{bmatrix}} + 7\underset{col2}{\begin{bmatrix}1\\3\end{bmatrix}} = \underset{b}{\begin{bmatrix}11\\29\end{bmatrix}}$$

  - A에 벡터 x를 곱하는 것은 A의 각 column vector에 x의 각 원소들을 곱하여 더하는, 즉 column vector들의 선형조합 연산에서 x의 원소들은 각 벡터에 곱해지는 상수가 된다.
  - 결국 column vector의 선형조합으로 표현되기 때문에 x는 당연히 column space 안으로 들어가는 것이다.
- $P = \frac{\boldsymbol{a}\boldsymbol{a}^T}{\boldsymbol{a}^T\boldsymbol{a}}$**의 투영행렬 P의 column space는 벡터 a를 지나가는 Line**이다.
- 투영 행렬의 rank는 어떻게 될까? Lecture11에서 **어떤 두 벡터를 column vector x row vector의 순서로 곱하면 반드시 rank 1 행렬이 만들어진다**고 배웠다. 따라서 $P = \frac{\boldsymbol{a}\boldsymbol{a}^T}{\boldsymbol{a}^T\boldsymbol{a}}$에 나온 것 처럼 column x row($\boldsymbol{a}\boldsymbol{a}^T$)의 곱으로 만들어졌기 때문에 투영 행렬은 rank가 1이며 **벡터 a가 행렬 P의 column space의 기저(basis)**가 된다.

- 투영 행렬 P는 대칭(symmetric)이다. 똑같은 벡터 a의 column x row순으로 곱하여 만들어진 행렬이기 때문에 대칭 행렬(symmetric matrix)이다. 따라서 아래 식과 같이 나타낼 수 있다.

  $P^T = P$

- 만약 벡터 b를 투영 행렬 P에 두 번 투영시켜도 결과는 변하지 않는다. 위의 그림에서 벡터 b를 P에 투영시켜 벡터 p로 만든 다음, 다시 p를 투영행렬 P에 곱하여 투영하는 것이다. 결과는 변함없이 p이다. 따라서 아래 식과 같이 나타낼 수 있다.

  $P^2 = P$

📌 투영 행렬(projection matrix)의 두 가지 특성

- **투영 행렬의 rank는 1**
- **대칭 행렬(symmetric matrix)**이고 **P의 제곱은 P와 같다**.

- 투영 행렬의 특성을 바탕으로 위의 그림의 벡터 b는 투영 행렬 P의 column space인 a를 지나 가는 Line(rank=1)에 위치하게 되는 것이다.

4. N-Dimensional Vector Projection (N차원 벡터 투영)

1) Why projection?

- 위에서 해가 존재하지 않는 선형연립방정식 Ax=b에 대하여 공부하였다. 해가 존재하지 않는 경우는 미지수보다 방정식이 더 많은 경우이고 이를 overdetermined case라고 배웠다. 이를 해결하기 위해서 우리는 비록 정확한 해는 없지만 그래도 가장 근접한 해인 $\hat{x}$을 구하여 이 문제를 해결하였다. 그렇다면 무엇이 가장 근접한 해(closest solution)인가?

- Overdetermined case인 경우 A**x**=b에서 A**x**의 결과 벡터는 항상 A의 column space에 존재한다. 그러나 b는 column space에 존재하지 않는다. 식은 =로써 정의했지만, 사실은 양변의 불일치가 발생하는 것이다. A는 이미 정해진 시스템이라 바꿀 수 없고 x는 앞으로 구해야 할 해다. 따라서 b를 바꿔야 한다.

- 이를 어떻게 바꿀 것인가? 바로**A의 column space에서 가장 근접한 벡터로 바꾸는 것**이다. 즉 A의 column space에 존재하는 수 많은 벡터중에서 현재의 b와 가장 흡사한 벡터를 골라서 b대신 놓는 것이다.

- **이때 b와 가장 흡사한 벡터가 바로 A의 column space로 투영한 벡터 p다**. 우변에 p가 위치한 경우**x**는 $\hat{x}$이 된다. 즉 원래의 b를 만족시키는 정확한 해는 아니지만, **최대한**

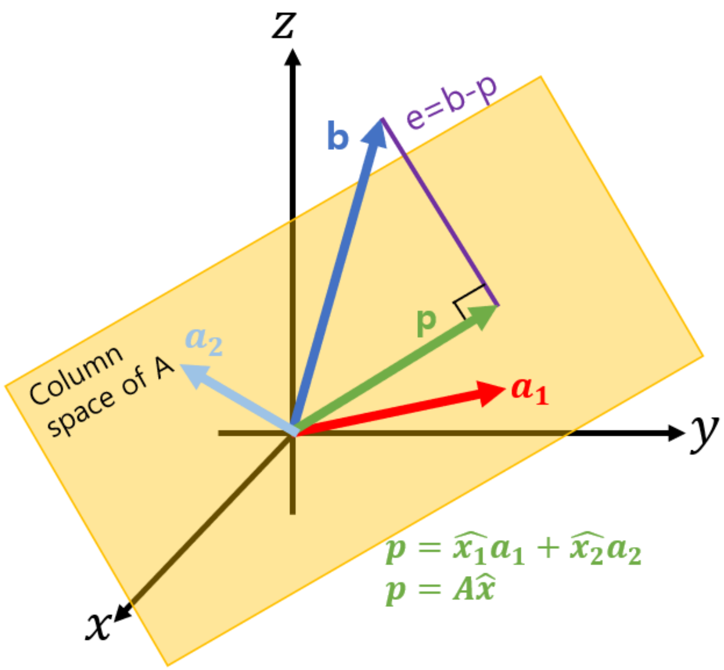
근접한 해를 의미한다.

Why project?
Because $Ax = b$ may have no solution (overdetermined)

$$\Downarrow$$

Instead, $A\hat{x} = p$ where p is proj. of b onto column space

- 우변의 벡터 b를 어떻게 column space로 투영(projection)시킬까?



  - 3차원 공간에서 3차원 벡터와 행렬을 이용하여 투영을 하는 모습을 나타낸 것이다. 노란색 평면은 행렬 A의 column space를 의미한다. 또한 이들의 기저(basis)는 그림에서 각각 a1과 a2이며 평면 위에 존재하는 column space의 임의의 벡터들이다.
  - 따라서 행렬 A는 아래 식과 같이 이 기저들로 이루어진 형태가 될 것이고 크기는 3x2가 될 것이다. 물론 이는 위의 그림을 기준으로 한 크기이며 차원은 임의의 n차원이 될 수 있다.

$$A = \begin{bmatrix} | & | \\ a_1 & a_2 \\ | & | \end{bmatrix}$$

  - 파란색 화살표로 나타낸 벡터 b는 column space의 평면 위에 존재하지 않는다. 이 벡터 b는 위에 Why project?에 있는 A**x**=b의 b이다.
  - overdetermined case에서 x는 a1과 a2의 선형 결합의 상수로써 그 결과가 b가 되도록 해야 한다. 그러나 보다시피 b는 column space 평면 위에 존재하지 않기 때

문에 이를 만족시키는 x는 애초에 존재하지 않는다. 그래서 최대한 만족시키는 해를 찾기 위해 b를 치환한다.

- 바로 A의 column space에 존재하는 무수히 많은 벡터들 중에 b와 가장 근접한 벡터로 말이다. 벡터의 화살표 끝점을 기준으로 봤을 때 가장 가까운 벡터는 당연히 평면에 수직(perpendicular)으로 내린 벡터가 될 것이다. 이것이 바로 벡터 p가 된다.
- 벡터 b에서 p의 방향으로 수직으로 연결한 선이 바로 두 벡터 사이의 오차(error)이며 보라색으로 표기되어 있다. 이 오차가 column space에 수직(perpendicular)하다는 것이 핵심이다. 오차는 e=b-p로 표현되었는데, p를 $p = A\hat{x}$로 치환하여 다시 정리하면 아래와 같다.

  $$\boldsymbol{p} = A\hat{\boldsymbol{x}},\ Find\ \hat{\boldsymbol{x}}$$
  $$\text{Key point:}$$
  $$b - p \rightarrow b - A\hat{\boldsymbol{x}} \perp plane$$

- 위의 식을 놓고 보면 우리는 a1과 a2 두 개의 방정식(equation)에 대해 정리할 수 있다.
  A는 a1과 a2의 두 개의 column vector로 이루어져 있고 x hat역시 x1과 x2로 이루어져 있기 때문이다. 즉 <u>오차에 대한 라인 e가 a1에도 수직이고 a2에도 역시 수직</u>이라는 의미다. a1, a2가 plane의 기저(basis)이므로 이 두 벡터하고 수직이면 모든 plane과 수직이다. 따라서 우리는 식을 a1과 a2로 나누어 식을 정리할 수 있다.

  $$a_1{}^T(b - A\hat{\boldsymbol{x}}) = 0,\ a_2{}^T(b - A\hat{\boldsymbol{x}}) = 0$$

- 위의 식은 오차 벡터 e와 a1, a2와의 내적(dot product)에 관한 식으로 정리한 것이다. 내적했을 때 0이라는 것은 수직인 것을 다들 이미 알고 있을 것이다. 위의 식을 행렬의 형태로 다시 정리해보자.

  $$\begin{bmatrix} a_1 \\ a_2 \end{bmatrix}(b - A\hat{\boldsymbol{x}}) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
  $$A^T(b - A\hat{\boldsymbol{x}}) = 0$$
  $$A^TA\hat{\boldsymbol{x}} = A^T\boldsymbol{b}$$

  앞의 a1과 a2로 이루어진 행렬은 원래 A의 전치(transpose)행렬이다. 따라서 행렬로 표현하면 마지막 식처럼 나타낼 수 있다. 이를 다시 전개하여 정리하면 위에서 봤던 x hat을 구하는 식이 나온다.

- <u>Lecture14</u>에서 배웠던 직교벡터와 부분 공간과 연결지어 생각해보자. $A^T(b - A\hat{\boldsymbol{x}}) = 0$가 **Left null space의 형태**인 것을 알 수 있다. 따라서 오차 벡터($e = (b - A\hat{x})$)는 행렬 A에 대한 **Left null space에 존재**한다.

- Left null space는 **행렬 A의 column space와 직교(orthogonal)**인 것을 배웠다. 위의 3차원 그림에서 행렬 A의 column space는 노란색 plane이고, 이와 직교인 오차 벡터 e는 Left null space에 있음을 증명했는데, 그림상에서도 plane과 수직(perpendicular)인 것을 알 수 있다. 이렇게 하여 행렬의 부분 공간(subspaces)과도 연결지어 이해할 수 있다.

  $$\boldsymbol{e} \text{ is in } N(A^T)$$
  $$\boldsymbol{e} \perp C(A)$$

- 결과적으로 문제를 해결하기 위해 핵심이 되는 식은 $A^T A\hat{\boldsymbol{x}} = A^T\boldsymbol{b}$이다.

2) Solution of the x hat and projection matrix

- $A^T A\hat{\boldsymbol{x}} = A^T\boldsymbol{b}$로부터 해(solution)를 구해보자. 해가 존재하지 않는 overdetermined case에서 우변의 b를 column space의 가장 유사한 벡터로 치환했으니 이제 유사한 해가 존재한다. $A^T A\hat{\boldsymbol{x}} = A^T\boldsymbol{b}$을 $\hat{x}$에 대하여 정리하면 아래와 같다.

  $$\hat{\boldsymbol{x}} = (A^T A)^{-1}A^T\boldsymbol{b}$$
  $$\boldsymbol{p} = A\hat{\boldsymbol{x}} = A(A^T A)^{-1}A^T\boldsymbol{b}$$
  $$P = A(A^T A)^{-1}A^T$$
  $$\boldsymbol{p} = P\boldsymbol{b}$$

  - 구하고자 하는 $\hat{x}$은 맨 위의 식과 같다. 이를 통해 $\hat{x}$의 값을 구할 수 있다.
    그 다음엔 원래의 식에 이 $\hat{x}$을 곱해서 b에서 A의 column space로 투영된 행렬 P를 구할 수 있다.
    $\hat{x}$을 A에 곱하면 A의 column space로 들어가게 되는데, 이것이 원래의 벡터 b를 A의 column space로 투영한 p 벡터가 된다. 이 식을 나타낸 것이 두번째 식의 왼쪽이다.

  - **같은 차원의 공간에 있는 어떠한 벡터이든지 행렬 A의 column space로 투영시킬 수 있는 투영 행렬(Projection matrix)을 구하고 싶다.** 이를 위해 $\hat{x}$을 먼저 구하고, $\hat{x}$과 A를 정리하여 투영 행렬 P를 구하는 것이다. 두번째 식으로부터 이 투영 행렬을 구할 수 있다. 두번째 식의 $\hat{x}$을 첫번째 식으로 치환해서 정리하면 두번째 식의 오른쪽과 같이 된다. 여기서 파란색 부분이 바로 투영 행렬 P에 대한 식이다. 이미 **3. 2D-Vector Projection(2차원 벡터 투영)**에서 1차원 벡터에 대한 투영 행렬 P를 $P = \frac{\boldsymbol{a}\boldsymbol{a}^T}{\boldsymbol{a}^T\boldsymbol{a}}$와 같이 구했다. 그러나 이는 1차원에 해당하는 방법이다. **위의 식은 n차원에 대한 투영 행렬을 구하는 방법을 나타낸다**.

- $P = A(A^T A)^{-1}A^T$를 보고 전개 해서 없앨 수 있을 것 같다고 생각할 수 있다. 일단 전개해보자.

  $$P = A(A^T A)^{-1}A^T$$
  $$P = AA^T(A)^{-1}A^T = I$$

- 전개해보니 결국 단위 행렬(Identity matrix)가 되었다. 그러나 이것은 특정 조건일 때만 맞는 이야기이다. 바로 **행렬 A가 정방행렬(square matrix)이고 역행렬이 존재할 때(invertible) 위의 전개가 성립**하는 것이다. 우리가 기본적으로 가정하는 것은 overdetermined case, 즉 직사각 행렬(Rectangular matrix)이며 방정식이 미지수보다 많은 경우이다. 이 경우엔 당연히 역행렬이 존재하지 않는다. 따라서 **위의 식은 직사각 행렬의 경우엔 성립하지 않는다**.

- 행렬 A가 역행렬을 가지는 경우(invertible)엔 이것을 어떻게 생각해야 할까? 즉 투영 행렬 P가 n x n의 정방행렬이며 역행렬이 존재하는 경우 말이다. n차원 공간에서 n차원 공간으로의 투영, 즉 3차원 공간에서 3차원 공간으로 투영시키는 것이고 단위 행렬(Identity matrix)을 의미한다. 투영 시켜도 아무 일도 일어나지 않는다.

- 3. 2D-Vector Projection(2차원 벡터 투영)에서 1차원 예시에서 투영 행렬이 대칭(symmetric)이며, 제곱해도 같음을 증명하였다. N차원의 경우에도 이것이 성립하는지 알아보자. $P = A(A^TA)^{-1}A^T$을 전치 시키면 아래와 같다.

  $$\begin{aligned} P^T &= P \\ P^T &= (A(A^TA)^{-1}A^T)^T \\ &= A^{TT}(A^TA^{TT})^{-1}A^T \qquad \text{since } (A^T)^{-1} = (A^{-1})^T \\ &= A(A^TA)^{-1}A^T \end{aligned}$$

  전치를 해도 식이 같음을 증명하였다.
  P의 제곱에 대해 증명해면 다음과 같다.

  $$\begin{aligned} P^2 &= P \\ P^2 &= A(A^TA)^{-1}A^TA(A^TA)^{-1}A^T \\ &= A(A^TA)^{-1}A^T \end{aligned}$$

  P를 제곱했더니 밑줄 친 부분이 단위 행렬이 되어 삭제된다. 따라서 P의 제곱이 원래의 P와 같음을 증명하였다.