

Chap 10

직교화(Orthogonalization)

이번 장의 첫 번째 목적은 다음 문제에 대한 알고리즘을 제시하는 것이다.

Computational Problem : (여러 벡터들의 Span 내에 있는 가장 가까운 점) 주어진 벡터 b 와 실수 벡터들 v_1, \dots, v_n 에 대해, Span $\{v_1, \dots, v_n\}$ 내에 있으며 b 에서 가장 가까운 벡터를 찾아보자.

$A = \begin{bmatrix} & & \\ v_1 & \cdots & v_m \end{bmatrix}$ 라 하고, 행렬-벡터 곱셈의 선형결합 정의에 의하면, Span $\{v_1, \dots, v_m\}$ 내 벡터들의 집합은 Ax 로 표현할 수 있는 벡터들의 집합이다. 따라서, 결국 계수(좌표)들을 찾는 것은 $\|b - Ax\|$ 을 최소화 하는 벡터 x 를 찾는 것과 같다. 이것을 **최소제곱** (least-squares) 문제라고 한다.

10.1 복수의 벡터들에 직교하는 투영

9장에서 살펴보았던 [소방차 문제](#)와 같이 직교성과 투영(projection)을 사용하여 풀 수 있다.

10.1.1 벡터들의 집합에 대한 직교

9장에서는 한 벡터가 다른 벡터에 직교한다는 것이 무엇을 의미하는지 살펴 보았다면, 이번 장에서는 한 벡터가 벡터들의 집합에 직교하는 것이 무엇을 의미하는지를 살펴본다.

Definition : 벡터 v 는 만약 \mathcal{S} 내의 모든 벡터에 직교하면 벡터들의 집합 \mathcal{S} 에 직교한다.

Example 10.1.2 : 벡터 $[2, 0, -1]$ 은 $[0, 1, 0]$ 과 $[1, 0, 2]$ 에 직교하므로 집합 $\{[0, 1, 0], [1, 0, 2]\}$ 에 직교한다. 또한, $[2, 0, -1]$ 벡터는 무한 집합 $\mathcal{V} = \text{Span } \{[0, 1, 0], [1, 0, 2]\}$ 에 직교한다. 그 이유는 \mathcal{V} 내의 모든 벡터는 $\alpha[0, 1, 0] + \beta[1, 0, 2]$ 이고 다음이 성립하기 때문이다.

$$\begin{aligned} \langle [2, 0, -1], \alpha[0, 1, 0] + \beta[1, 0, 2] \rangle &= \alpha \langle [2, 0, -1], [0, 1, 0] \rangle + \beta \langle [2, 0, -1], [1, 0, 2] \rangle \\ &= \alpha 0 + \beta 0 \end{aligned}$$

Lemma : 벡터 v 가 벡터들 a_1, \dots, a_n 각각에 직교할 필요충분조건은 v 가 Span $\{a_1, \dots, a_n\}$ 내의 모든 벡터에 직교하는 것이다.

- **Proof :** v 는 a_1, \dots, a_n 에 직교한다고 하고, w 는 Span $\{a_1, \dots, a_n\}$ 내의 임의의 벡터라고 하자. v 는 w 에 직교한다는 것을 보이면 된다. Span의 정의에 의하면 다음을 만족하는 계수 $\alpha_1, \dots, \alpha_n$ 이 있다.

$$w = \alpha_1 a_1 + \cdots + \alpha_n a_n$$

- 그러므로, 직교의 성질을 사용하면 다음이 성립한다.

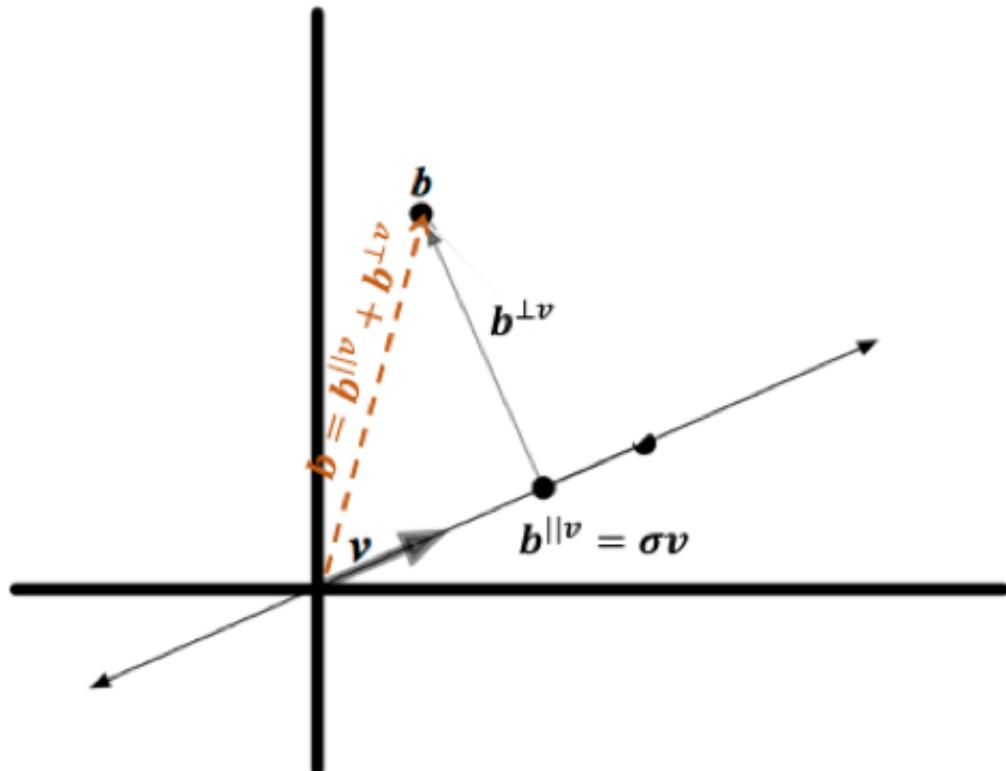
$$\begin{aligned}
 \langle v, w \rangle &= \langle v, \alpha_1 a_1 + \cdots + \alpha_n a_n \rangle \\
 &= \alpha_1 \langle v, a_1 \rangle + \cdots + \alpha_n \langle v, a_n \rangle \\
 &= \alpha_1 0 + \cdots + \alpha_n 0 \\
 &= 0
 \end{aligned}$$

- 따라서, v 는 w 에 직교한다. 이제, v 는 $\text{Span} \{a_1, \dots, a_n\}$ 의 모든 벡터에 직교한다고 해보자. $\text{Span} \{a_1, \dots, a_n\}$ 은 a_1, \dots, a_n 을 포함하므로 v 는 a_1, \dots, a_n 에 직교한다.

10.1.2 벡터공간상으로의 투영 및 벡터공간에 직교하는 투영

Definition: 벡터 b 와 벡터공간 \mathcal{V} 에 대해, b 의 \mathcal{V} 상으로의 투영 ($b^{\parallel \mathcal{V}}$)과 b 의 \mathcal{V} 에 직교하는 투영 ($b^{\perp \mathcal{V}}$)을 정의해 보자. 그러면, 다음과 같이 쓸 수 있다.

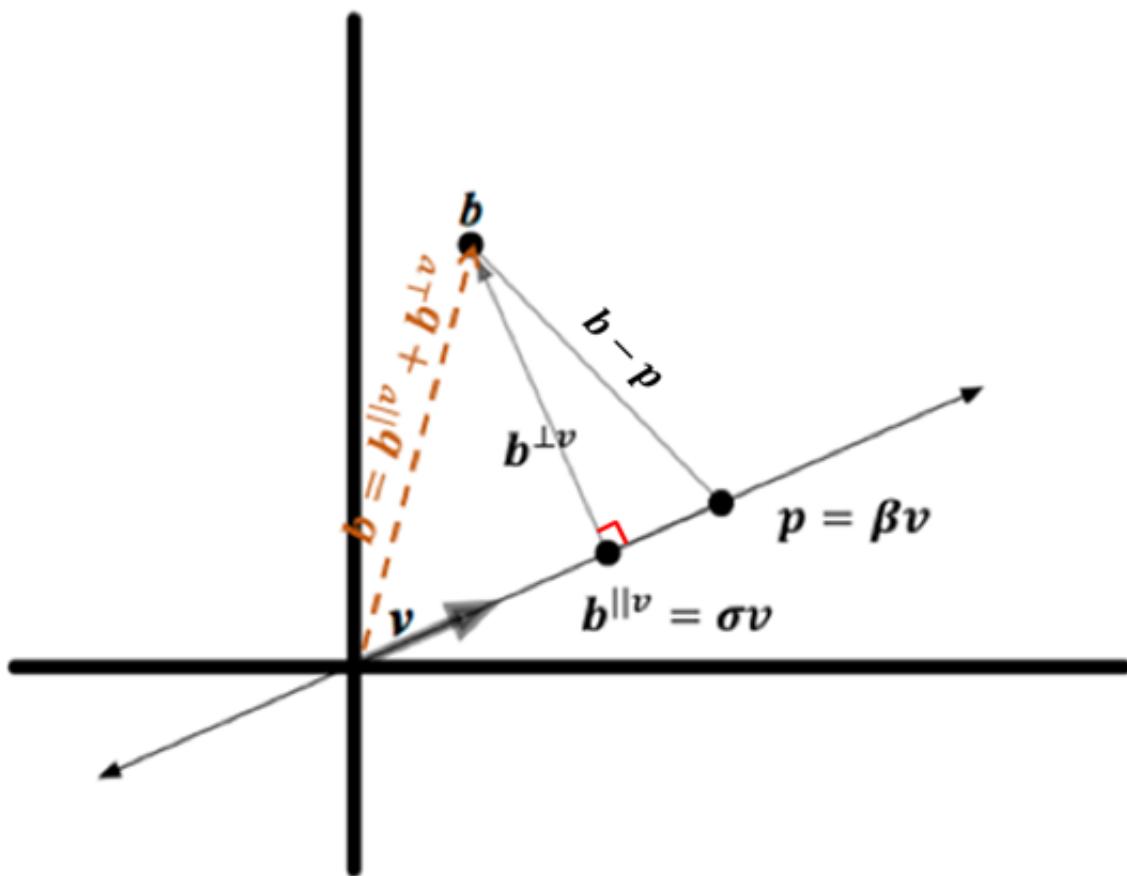
$$b = b^{\parallel \mathcal{V}} + b^{\perp \mathcal{V}}$$



그리고, $b^{\parallel \mathcal{V}}$ 는 \mathcal{V} 에 속하고, $b^{\perp \mathcal{V}}$ 는 \mathcal{V} 에 속하는 모든 벡터에 직교한다.

\mathbf{b} projection onto \mathcal{V} \mathbf{b} projection orthogonal to \mathcal{V}
 $\mathbf{b} = \mathbf{b}^{\parallel \mathcal{V}} + \mathbf{b}^{\perp \mathcal{V}}$

Lemma: \mathcal{V} 는 벡터공간이라 하고, \mathbf{b} 는 벡터라고 하자. \mathcal{V} 에 속하며 \mathbf{b} 에 가장 가까운 점은 $\mathbf{b}^{\parallel \mathcal{V}}$ 이고, 그 거리는 $\|\mathbf{b}^{\perp \mathcal{V}}\|$ 이다.



- **Proof:** \mathbf{b} 와 $\mathbf{b}^{\parallel \mathcal{V}}$ 사이의 거리는 $\|\mathbf{b} - \mathbf{b}^{\parallel \mathcal{V}}\|$ 이다. \mathbf{p} 는 \mathcal{V} 의 임의의 점이라고 하면, \mathbf{p} 는 $\mathbf{b}^{\parallel \mathcal{V}}$ 보다 \mathbf{b} 에 가깝지 않다는 것을 보여 보자.

$$\mathbf{b} - \mathbf{p} = (\mathbf{b} - \mathbf{b}^{\parallel \mathcal{V}}) + (\mathbf{b}^{\parallel \mathcal{V}} - \mathbf{p})$$

- 우변에서 $\mathbf{b} - \mathbf{b}^{\parallel \mathcal{V}} = \mathbf{b}^{\perp \mathcal{V}}$ 이고, $\mathbf{b}^{\parallel \mathcal{V}} - \mathbf{p}$ 는 \mathcal{V} 에 속하는 두 벡터의 차이이므로 \mathcal{V} 내에 있다. $\mathbf{b}^{\perp \mathcal{V}}$ 는 \mathcal{V} 에 직교하므로, 피타고拉斯 정리에 의하면 다음이 성립한다.

$$\|\mathbf{b} - \mathbf{p}\|^2 = \|\mathbf{b} - \mathbf{b}^{\parallel \mathcal{V}}\|^2 + \|\mathbf{b}^{\parallel \mathcal{V}} - \mathbf{p}\|^2$$

- 따라서, $p \neq b^{\perp\mathcal{V}}$ 이면 $\|b - p\| > \|b - b^{\perp\mathcal{V}}\|$ 이 성립한다.

10.1.3 벡터들의 리스트에 직교하는 투영 - 첫 번째 시도

파이썬을 이용해서 다음을 만족하는 `project_orthogonal()` 함수를 작성해보자.

- input*: 벡터 b , 벡터들의 리스트 $vlist$
- output*: Span $vlist$ 에 직교하는 b 의 투영 (즉, $b^{\perp\mathcal{V}}$)

아래의 코드는 9.3.4에서 작성한 `project_along()` 함수를 이용하여 `project_orthogonal()` 함수를 구현한 코드이다.

```

1  def project_along(b, v):
2      bv = 0
3      vv = 0
4      for u, w in zip(b, v):
5          bv += u*w
6      for u,w in zip(v, v):
7          vv += u*w
8
9      sigma = (bv / vv) if vv > 1e-20 else 0
10     return [sigma*e for e in v]
11
12
13 def project_orthogonal(b, vlist):
14     for v in vlist:
15         b = [e1 - e2 for e1, e2 in zip(b, project_along(b, v))]
16     return b

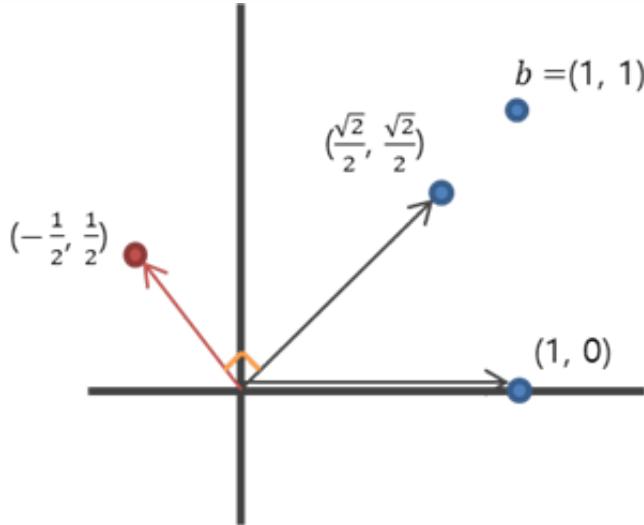
```

```

1  >>> vlist = [[1, 0], [2**(.5)/2, 2**(.5)/2]]
2  >>> b = [1,1]
3
4  >>> project_orthogonal(b, vlist)
5  [-0.5, 0.5]

```

위의 `project_orthogonal()` 함수는 문제가 있다. 예를 들어 $b = [1, 1]$, $vlist = [[1, 0], [\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]]$ 라고 할 경우 `project_orthogonal()`의 출력값은 $[-0.5, 0.5]$ 이다. 이 벡터는 아래의 그림과 같이 $vlist$ 의 $[1, 0]$ 과 직교하지 않는다.



10.2 서로 직교하는 벡터들의 리스트에 직교하는 b 의 투영

그렇다면, 위의 `project_orthogonal()` 함수가 동작하도록 하는 *vlist*들을 만들어 보자. 예를 들어, *vlist*에 $v_1 = [1, 2, 1]$, $v_2 = [-1, 0, 1]$ 과 $b = [1, 1, 2]$ 라고 하면 다음과 같은 결과가 나온다.

```

1  >>> vlist = [[1, 2, 1], [-1, 0, 1]]
2  >>> b = [1, 1, 2]
3
4  >>> b_orthogonal = project_orthogonal(b, vlist)
5  >>> b_orthogonal = [round(e, 2) for e in b_orthogonal] # 소수점 둘째자리까지 출력
6  >>> print(b_orthogonal)
7  [0.67, -0.67, 0.67]

```

이 결과값은 *vlist*의 v_1 과 v_2 두 벡터에 각각 직교한다. 또한 v_1 과 v_2 는 서로 직교한다. 이것을 9.3.4에서 배웠던 공식을 이용하면 다음과 같다.

$\langle v_1, b \rangle = 0$ 이고, $\langle v_1, v_2 \rangle = 0$ 이므로,

$$\begin{aligned} \langle v_1, b - \sigma v_2 \rangle &= \langle v_1, b \rangle - \langle v_1, \sigma v_2 \rangle \\ &= \langle v_1, b \rangle - \sigma \langle v_1, v_2 \rangle \\ &= 0 + 0 \end{aligned}$$

즉, 10.1.3과 달리 input 값을 아래와 같이 설정해주면 `project_orthogonal()` 은 제대로 동작한다.

- *input*: 벡터 b , 서로 직교하는 벡터들의 리스트 *vlist*
- *output*: *vlist*에 속하는 벡터들에 직교하는 b 의 투영 b^\perp

10.2.1 프로시저 `project_orthogonal()` 이 맞게 동작하는지 증명하기

Theorem (`project_orthogonal()` 의 정확성): 벡터 b 와 서로 직교하는 벡터들의 리스트 `vlist`에 대해, 함수 `project_orthogonal(b, vlist)` 는 b^\perp 을 반환한다. 이때, b^\perp 은 `vlist` 에 속하는 벡터들에 직교하며 $b - b^\perp$ 은 `vlist` 에 속하는 벡터들의 Span에 속한다.

함수 `project_orthogonal()` 이 맞다는 것을 증명하기 위해, i 이터레이션 후 i 를 포함하는 `vlist` 가 $i = 0, 1, 2, \dots$ 에 대해 참이라는 것을 보여 준다. 이러한 방법을 루프불변 (*loop invariant*)

Lemma (`project_orthogonal` 에 대한 루프불변): $k = \text{len}(vlist)$ 라고 하면, $i = 0, \dots, k$ 에 대해, b_i 는 i 이터레이션 후 변수 `b` (`project_orthogonal` 의 결과값, 즉 b^\perp)의 값이라고 하자.

- b_i 는 `vlist`에 속하는 첫 i 개 벡터들에 직교한다.
- $b - b_i (= b - b_i^\perp)$ 은 `vlist`에 속하는 첫 i 개 벡터들의 Span에 속한다.

`project_orthogonal` 함수에 의해 반환되는 벡터 b^\perp 은 모든 k 이터레이션 후 b 의 값이며 b_k 로 나타낸다. 루프불변의 k 에 i 를 대입하면 다음을 얻는다.

- b_k^\perp 은 `vlist`의 첫 k 개 벡터에 직교하고 $b - b_k^\perp$ 는 $\text{Span}\{\text{vlist}\}$ 에 속한다.
 - **Proof:** 증명은 i 에 대해 귀납법을 사용한다. $i = 0$ 인 경우, 루프불변은 참이다: b_0 는 첫 0개 벡터들의 각각에 직교하고, $b - b_0$ 은 첫 0개 벡터들의 Span에 속한다(왜냐하면, 영벡터이기 때문에!). $i - 1$ 이터레이션에 대해 루프불변이 성립한다고 가정하자. 그런 다음 i -이터레이션에 대해 루프불변이 성립함을 증명하면 된다. `vlist`를 $[v_1, \dots, v_k]$ 로 나타내자.
 - i 번째 이터레이션에서 함수는 다음을 계산한다.

$$\begin{aligned} b_i^\perp &= b_{i-1}^\perp - \text{project}_{\text{along}}(b_{i-1}, v_i) \\ &= b_{i-1}^\perp - \sigma_i v_i \end{aligned}$$

- 여기서 $\sigma_i = \frac{\langle b_{i-1}, v_i \rangle}{\langle v_i, v_i \rangle}$ 이다. 귀납 가설은 b_{i-1} 이 첫 $i - 1$ 개 벡터들에 직교하는 b_0 의 투영이라는 것이다.
- b_i^\perp 가 $\{v_1, \dots, v_{i-1}, v_i\}$ 에 속하는 각 벡터에 직교한다는 것을 증명해야 한다. $i > j$ 라고 하면

$$\begin{aligned} \langle b_i^\perp, v_j \rangle &= \langle b_{i-1}^\perp - \sigma_i v_i, v_j \rangle \\ &= \langle b_{i-1}^\perp, v_j \rangle - \sigma_i \langle v_i, v_j \rangle \\ &= 0 - \sigma_i \langle v_i, v_j \rangle \\ &= 0 - \sigma_i 0 \end{aligned}$$

- 또한, $b_0 - b_i^\perp$ 는 `vlist`의 첫 i 개의 벡터들의 Span에 속한다는 것을 증명해야 한다. 귀납가설에 의하면, $b_0 - b_{i-1}^\perp$ 은 $i - 1$ 개 벡터들의 Span에 속한다.

$$\begin{aligned} b_0 - b_i^\perp &= b_0 - (b_{i-1}^\perp - \sigma_i v_i) \\ &= (b_0 - b_{i-1}^\perp) + \sigma_i v_i \\ &= (\text{첫 } i - 1 \text{ 개 벡터들의 Span에 속하는 벡터}) + \sigma_i v_i \\ &= \text{첫 } i \text{ 개 벡터들의 Span에 속하는 벡터} \end{aligned}$$

10.2.2 `project_orthogonal()` 보완하기

$b - b^\perp$ 이 벡터들 v_0, \dots, v_{k-1} 의 생성에 속한다는 것을 다음과 같이 표현할 수 있다.

$$\begin{aligned}b - b^\perp &= \sigma_0 v_0 + \dots + \sigma_{k-1} v_{k-1} \\b &= \sigma_0 v_0 + \dots + \sigma_{k-1} v_{k-1} + 1b^\perp\end{aligned}$$

$$\begin{bmatrix} b \\ 1 \end{bmatrix} = \begin{bmatrix} v_0 & \cdots & v_{k-1} & b^\perp \end{bmatrix} \begin{bmatrix} \sigma_0 \\ \sigma_1 \\ \vdots \\ \sigma_{k-1} \\ 1 \end{bmatrix}$$

위의 식을 반영하여, `project_orthogonal(b, vlist)` 를 보완한 `aug_project_orthogonal(b, vlist)` 함수를 구현해보자.

- input: 벡터 b , 서로 직교하는 실수 벡터들의 리스트 $[v_0, \dots, v_{k-1}]$
- output: 튜플 $(b^\perp, sigmadict)$ 이 튜플은 다음을 만족한다.
 - 튜플의 첫 번째 원소는 b 의 투영 b^\perp 이며 이것은 $\text{Span}\{v_0, \dots, v_{k-1}\}$ 에 직교한다.
 - 튜플의 두 번째 원소는 딕셔너리 $sigmadict = \{0 : \sigma_0, 1 : \sigma_1, \dots, (k-1) : \sigma_{k-1}, k : 1\}$ 이며 다음을 만족한다.

$$b = \sigma_0 v_0 + \dots + \sigma_{k-1} v_{k-1} + 1b^\perp$$

```
1 def aug_project_orthogonal(b, vlist):
2     sigmadict = {len(vlist): 1}
3     for i, v in enumerate(vlist):
4         bv = 0
5         vv = 0
6         for u, w in zip(b, v):
7             bv += u*w
8             for u,w in zip(v, v):
9                 vv += u*w
10            sigma = bv/vv if vv > 1e-20 else 0
11            sigmadict[i] = round(sigma, 2)
12            b = [round(e1 - sigma*e2, 2) for e1, e2 in zip(b, v)]
13    return b, sigmadict
```

```
1 >>> b = [1, 1, 2]
2 >>> vlist = [[1, 2, 1], [-1, 0, 1]]
3
4 >>> aug_project_orthogonal(b, vlist)
5 ([0.67, -0.67, 0.67], {0: 0.83, 1: 0.5, 2: 1})
```

10.3 생성자들의 직교집합 만들기

이번 장의 목표는 맨처음에서도 살펴 보았듯이 임의의 벡터들 v_1, \dots, v_n 으로 구성된 집합의 생성에 직교하게 b 를 투영하는 것이다. 10.1 ~ 10.2 까지 구현했던 함수들은 서로 직교하는 벡터들로 구성된 $vlist$ 에 직교하는 b 를 투영하는 것에서는 제대로 동작하였다.

이제는 서로 직교하지 않을 수도 있는 벡터들 v_1, \dots, v_n 의 생성에 직교하게 b 를 투영해보자. 이를 위해서는 $\text{Span}\{v_1, \dots, v_n\}$ 에 대한 서로 직교하는 생성자(generator)를 먼저 찾아야 한다.

따라서, 새로운 함수인 **직교화(orthogonalization)**가 필요하다.

- *input*: 실수 벡터들의 리스트 $[v_1, \dots, v_n]$
- *output*: 다음을 만족하는 서로 직교하는 벡터들 v_1^*, \dots, v_n^* 의 리스트

$$\text{Span}\{v_1^*, \dots, v_n^*\} = \text{Span}\{v_1, \dots, v_n\}$$

10.3.1 `orthogonalize()` 프로시저

`orthogonalize()` 함수를 구현하기 위해 위에서 작성한 `project_orthogonal()` 함수를 반복하여 서로 직교하는 벡터들의 리스트를 만들어 준다.

먼저, v_1 벡터를 고려해 보자. 집합 $\{v_1^*\}$ 은 서로 직교하는 벡터들의 집합이므로 $v_1^* := v_1$ 이라고 정의하자. 다음으로, v_2^* 는 v_1^* 에 직교하는 v_2 의 투영이라고 정의하자. 그렇게 되면, $\{v_1^*, v_2^*\}$ 은 서로 직교하는 벡터들의 집합이다. 그 다음으로 v_3^* 는 v_3 의 투영이고 v_1^* 과 v_2^* 직교하는 벡터라고 하자. 이런 식으로 v_n^* 까지 진행하여 직교하는 벡터들을 찾는다. 따라서, i 번째에서는 v_i 를 v_1^*, \dots, v_{i-1}^* 에 직교하게 투영하는 v_i^* 를 찾는다.

위의 방법을 코드로 나타내면 다음과 같다.

```
1 def orthogonalize(vlist):
2     vstarlist = []
3     for v in vlist:
4         vstarlist.append(project_orthogonal(v, vstarlist))
5     return vstarlist
```

```
1 >>> vlist = [[2, 0, 0], [1, 2, 2], [1, 0, 2]]
2
3 >>> orthogonalize(vlist)
4 [[2, 0, 0], [0.0, 2.0, 2.0], [0.0, -1.0, 1.0]]
```

Example 10.3.2: 위의 출력 결과에서 알 수 있듯이, 아래의 `vlist`에 대해 `orthogonalize()` 호출하면, 다음과 같은 v^*list 가 반환된다.

$$v_1 = [2, 0, 0], v_2 = [1, 2, 2], v_3 = [1, 0, 2]$$

$$v_1^* = [2, 0, 0], v_2^* = [0, 2, 2], v_3^* = [0, -1, 1]$$

- 첫 번째 이터레이션에서, `v` 는 v_1 이고 `vstarlist` 가 빈 리스트이므로 `vstarlist`에 추가되는

첫 번째 벡터 v_1^* 은 v_1 이다.

- 두 번째 이터레이션에서, v 는 v_2 이고 `vstarlist` 는 v_1^* 만으로 구성되어 있어 v_2 의 v_1^* 에 직교하는 투영은 다음과 같다.

$$\begin{aligned} v_2 - \frac{\langle v_2, v_1^* \rangle}{\langle v_1^*, v_1^* \rangle} v_1^* &= [1, 2, 2] - \frac{2}{4} [2, 0, 0] \\ &= [0, 2, 2] \end{aligned}$$

- 세 번째 이터레이션에서, v 는 v_3 이고 `vstarlist` 는 v_1^* 와 v_2^* 로 구성되어 있어 v_3 의 v_1^* 에 직교하는 투영은 $[0, 0, 2]$ 이고, $[0, 0, 2]$ 의 v_2^* 에 직교하는 투영은 다음과 같다.

$$[0, 0, 2] - \frac{1}{2} [0, 2, 2] = [0, -1, 1]$$

10.3.2 `orthogonalize()` 가 맞게 동작하는지 증명하기

`orthogonalize()` 가 제대로 동작하는지 보여주기 위해, 리턴되는 벡터들로 구성된 리스트 (`vstarlist`)의 Span이 입력으로 제공된 벡터들로 구성된 리스트 (`vlist`)의 Span과 동일함을 보여야 한다.

Lemma: `orthogonalize` 를 n -원소 리스트 $[v_1, \dots, v_n]$ 에 적용하면, i 번째 이터레이션 후 $\text{Span}\{\text{vstarlist}\} = \text{Span}\{v_1, \dots, v_i\}$ 이다.

- **Proof:**

$$\text{Span}\{v_1^*, \dots, v_{i-1}^*\} = \text{Span}\{v_1, \dots, v_{i-1}\}$$

- 벡터 v_i 를 양변의 집합에 더하면 다음과 같다.

$$\text{Span}\{v_1^*, \dots, v_{i-1}^*, v_i\} = \text{Span}\{v_1, \dots, v_{i-1}, v_i\}$$

- 따라서, $\text{Span}\{v_1^*, \dots, v_{i-1}^*, v_i^*\} = \text{Span}\{v_1, \dots, v_{i-1}, v_i\}$ 임을 보여주면 된다.
- i 번째 이터레이션에서 `project_orthogonal` ($v_i, [v_1^*, \dots, v_{i-1}^*]$) 을 사용하여 v_i^* 을 계산한다.

$$v_i = \sigma_{1,i} v_1^* + \dots + \sigma_{i-1,i} v_{i-1}^* + v_i^*$$

- 위의 식은 $v_1^*, \dots, v_{i-1}^*, v_i$ 벡터들의 임의의 선형결합은 $v_1^*, \dots, v_{i-1}^*, v_i^*$ 벡터들의 선형결합으로 변환할 수 있고, 그 역도 성립한다.

이러한 직교화(orthogonalization)은 수학자 그램(Gram)과 슈미트(Schmidt)의 이름을 따 [Gram-Schmidt 직교화](#)라고 한다.

위의 식 $v_i = \sigma_{1,i} v_1^* + \dots + \sigma_{i-1,i} v_{i-1}^* + v_i^*$ 을 행렬형태로 나타내면 다음과 같다.

$$\begin{bmatrix} v_1 & v_2 & v_3 & \cdots & v_n \end{bmatrix} = \begin{bmatrix} v_1^* & v_2^* & v_3^* & \cdots & v_n^* \end{bmatrix} \begin{bmatrix} 1 & \sigma_{12} & \sigma_{13} & \cdots & \sigma_{1n} \\ 1 & \sigma_{23} & \cdots & \sigma_{2n} \\ 1 & \cdots & \sigma_{3n} \\ \ddots & & \sigma_{n-1,n} \\ 1 & & & & \end{bmatrix}$$

우변의 두 행렬은 특수한 형태를 띤다. 첫 번째 행렬은 서로 직교하는 열벡터들을 가진다. 두 번째 행렬은 정방행렬이며 ij 원소는 $i > j$ 이면 영(0)인 상삼각(Upper-Triangular) 행렬이다.

Gram-Schmidt Orthogonalization

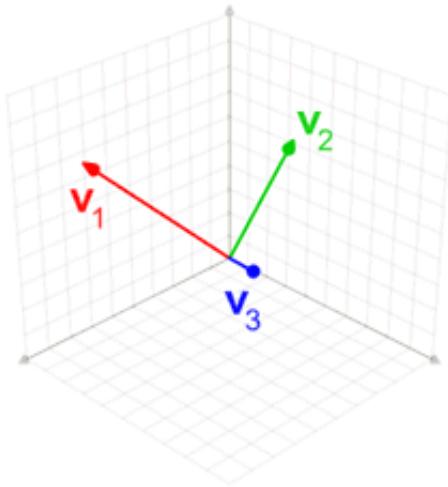
*Gram-Schmidt Orthogonalization*은 주어진 벡터들 v_1, \dots, v_n 을 생성할 수 있는 직교기저(*orthogonal basis*)를 구하는 과정이다.

주어진 벡터들 v_1, \dots, v_n 에 대하여, 이 벡터들을 생성할 수 있는 직교 벡터 v_1^*, \dots, v_n^* 들은 다음과 같이 구해진다.

$$\begin{aligned} v_1^* &= v_1 \\ v_2^* &= v_2 - \text{proj}_{v_1^*}(v_2) = v_2 - \sigma_{12}v_1^* \\ &= v_2 - \frac{\langle v_2, v_1^* \rangle}{\langle v_1^*, v_1^* \rangle} v_1^* \\ v_3^* &= v_3 - \text{proj}_{v_1^*}(v_3) - \text{proj}_{v_2^*}(v_3) = v_3 - \sigma_{13}v_1^* - \sigma_{23}v_2^* \\ &= v_3 - \frac{\langle v_3, v_1^* \rangle}{\langle v_1^*, v_1^* \rangle} v_1^* - \frac{\langle v_3, v_2^* \rangle}{\langle v_2^*, v_2^* \rangle} v_2^* \\ &\vdots \\ v_n^* &= v_n - \sum_{i=1}^{n-1} \text{proj}_{v_i^*}(v_n) \\ &= v_n - \sigma_{1n}v_1^* - \cdots - \sigma_{n-1,n}v_{n-1}^* \end{aligned}$$

위의 계산을 통해 얻어진 v_1^*, \dots, v_n^* 은 서로 직교(*orthogonal*)하며, 벡터공간 $\text{Span}\{v_1, v_2, \dots, v_n\}$ 에 대한 직교기저가 된다.

아래의 예제 이미지(출처: [Wikipedia](#))는 R^3 에 대한 직교화 과정을 통해 *Gram-Schmidt orthogonormalization*을 구하는 과정이다. 이미지의 u_i 의 값은 위의 식에서 v_i^* 이다.



10.4 벡터들의 생성에 속하는 점에 가장 가까운 계산문제 풀기

이제 $\text{Span}\{v_1, \dots, v_n\}$ 에 속하며 b 에 가장 가까운 벡터를 찾는 문제를 해결할 수 있다. 9.3.3 의 Lemma에 따라, 가장 가까운 벡터는 $b^{\parallel \mathcal{V}}$, 즉 b 의 $\mathcal{V} = \text{Span}\{v_1, \dots, v_n\}$ 상으로의 투영이고, 이것은 $b - b^{\perp \mathcal{V}}$ 이다. 여기서 $b^{\perp \mathcal{V}}$ 는 \mathcal{V} 에 직교하는 b 의 투영이다.

$b^{\perp \mathcal{V}}$ 를 찾는 방법은 다음과 같다.

- 먼저, `orthogonalize()` 함수를 이용해 `vlist` $[v_1, \dots, v_n]$ 의 직교벡터인 `vstarlist` $[v_1^*, \dots, v_n^*]$ 을 구한다.
- 그 다음, `project_orthogonal()` 함수를 이용해 $b^{\perp \mathcal{V}}$ 를 구한다.

아래의 예제 코드는 `vlist=[[8, -2, 2], [4, 2, 4]]` 이고 `b=[5, -5, 2]` 일때 $b^{\parallel \mathcal{V}}$ 와 $b^{\perp \mathcal{V}}$ 를 구한 뒤 $b^{\parallel \mathcal{V}} + b^{\perp \mathcal{V}}$ 가 b 와 일치하는지 확인하는 코드이다.

```

1  >>> b = [5, -5, 2]
2  >>> vlist = [[8, -2, 2], [4, 2, 4]]
3
4  >>> vstarlist = orthogonalize(vlist)
5  >>> print('vstarlist =', vstarlist)
6  vstarlist = [[8, -2, 2], [0.0, 3.0, 3.0]]
7
8  >>> b_bot = project_orthogonal(b, vstarlist)
9  >>> print('b_bot =', b_bot)
10 b_bot = [-1.0, -2.0, 2.0]
11
12 >>> b_proj = [e1 - e2 for e1, e2 in zip(b, b_bot)]
13 >>> print('b_proj =', b_proj)
14 b_proj = [6.0, -3.0, 0.0]
15

```

```

16 >>> b_ = [e1 + e2 for e1, e2 in zip(b_proj, b_bot)]
17 >>> print('b =', b_)
18 b = [5.0, -5.0, 2.0]

```

이번에는 위 10.1.3에서 제대로 구하지 못했던 $b = [1, 1]$, $vlist = [[1, 0], [\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]]$ 일 때, $b^{\perp\mathcal{V}}$ 와 $b^{\parallel\mathcal{V}}$ 를 구한 뒤 $b^{\parallel\mathcal{V}} + b^{\perp\mathcal{V}}$ 가 b 와 일치하는지 확인해보자.

```

1 b = [1,1]
2 vlist = [[1, 0], [2**(.5)/2, 2**(.5)/2]]
3
4 vstarlist = orthogonalize(vlist)
5 print('vstarlist =', vstarlist)
6 vstarlist = [[1, 0], [0.0, 0.7071067811865476]]
7
8 b_bot = project_orthogonal(b, vstarlist)
9 print('b_bot =', b_bot)
10 b_bot = [0.0, 0.0]
11
12 b_proj = [e1 - e2 for e1, e2 in zip(b, b_bot)]
13 print('b_proj =', b_proj)
14 b_proj = [1.0, 1.0]
15
16 b_ = [e1 + e2 for e1, e2 in zip(b_proj, b_bot)]
17 print('b =', b_)
18 b = [1.0, 1.0]

```

10.5 orthogonalize() 를 사용하여 다른 문제 풀기

이제, 직교화를 다른 계산문제들에서 어떻게 사용할 수 있는지에 대해 알아 보도록 하자.

Proposition: 서로 직교하는 영(0)이 아닌 벡터들은 선형독립(일차독립)이다.

- **Proof:** $v_1^*, v_2^*, \dots, v_n^*$ 은 서로 직교하는 영이 아닌 벡터들이라 하고, $\alpha_1, \alpha_2, \dots, \alpha_n$ 은 다음 식을 만족하는 계수라고 하자.

$$0 = \alpha_1 v_1^* + \alpha_2 v_2^* + \cdots + \alpha_n v_n^*$$

- 계수들 $\alpha_1, \alpha_2, \dots, \alpha_n$ 이 모두 영(0) 임을 보여야 한다. α_1 이 0 인 것을 보이기 위해, 양변에 v_1^* 과의 내적을 취해 보자.

$$\begin{aligned}
 \langle v_1^*, 0 \rangle &= \langle v_1^*, \alpha_1 v_1^* + \alpha_2 v_2^* + \cdots + \alpha_n v_n^* \rangle \\
 &= \alpha_1 \langle v_1^*, v_1^* \rangle + \alpha_2 \langle v_1^*, v_2^* \rangle + \cdots + \alpha_n \langle v_1^*, v_n^* \rangle \\
 &= \alpha_1 \|v_1^*\|^2 + \alpha_2 0 + \cdots + \alpha_n 0 \\
 &= \alpha_1 \|v_1^*\|^2
 \end{aligned}$$

- 내적 $\langle v_1^*, 0 \rangle$ 은 0이고, $\alpha_1 \|v_1^*\|^2 = 0$ 이다. v_1^* 은 영이 아니므로, **norm** 은 영이 아니다. 따라서, 유일한 해는 $\alpha_1 = 0$ 이다.
- 마찬가지 방법으로 $\alpha_2 = 0, \dots, \alpha_n = 0$ 을 보여줄 수 있다.

10.5.1 기저 계산하기

함수 `orthogonalize()` 는 입력 벡터 `vlist` 의 벡터들이 선형독립이어야 한다는 조건이 없다. 그렇다면 만약 선형독립이 아닐 경우 어떻게 될까?

v_1^*, \dots, v_n^* 은 `orthogonalize([v1, ..., vn])`에 의해 반환되는 벡터들이라고 하면, 이 벡터들은 서로 직교하며 v_1, \dots, v_n 과 동일한 벡터공간을 생성한다. 하지만, 이 벡터들 중 일부는 영벡터 일 수 있다. S 는 $\{v_1^*, \dots, v_n^*\}$ 의 부분집합이며, 영이 아닌 벡터들이라고 하면, $\text{Span } S = \text{Span } \{v_1^*, \dots, v_n^*\}$ 이다. 왜냐하면 영벡터는 생성에 영향을 주지 않기 때문이다. 또한 v_1^*, \dots, v_n^* 벡터들 서로 직교하므로, S 의 벡터들은 선형독립이다. 그러므로, 이 벡터들은 $\text{Span } \{v_1^*, \dots, v_n^*\}$ 에 대한 기저이고, 또한 $\text{Span } \{v_1, \dots, v_n\}$ 에 대한 기저이다.

`orthogonalize()` 함수에서 반환된 `vstarlist` v_1^*, \dots, v_n^* 에서 기저를 찾는 함수 `find_basis()` 를 구현하면 아래 코드와 같다.

```
1 def find_basis(vlist):
2     'return the list of nonzero starred vectors'
3     vstarlist = orthogonalize(vlist)
4     return [v for v in vstarlist if v.count(0) is not len(v)]
```

```
1 >>> vlist = [[0, 0, 0], [8, -2, 2], [4, 2, 4]]
2 >>> print('basis =', find_basis(vlist))
3 basis = [[8, -2, 2], [0.0, 3.0, 3.0]]
```

또한, `find_basis()` 를 이용하여 다음의 문제들에 대한 계산도 가능하다.

- 벡터들로 구성된 리스트의 랭크 찾기
- 벡터들 v_1, \dots, v_n 이 선형독립인지 테스트 하기

10.5.2 생략

10.5.3 `augmented_orthogonalize()`

10.2.2 에서 구현한 `aug_project_orthogonal(b, vlist)` 와 마찬가지로 다음을 만족하는 `aug_orthogonalize(vlist)` 를 구현해 보자.

- input: 벡터들의 리스트 $[v_1, \dots, v_n]$
- output: 다음을 만족하는 벡터들의 리스트의 튜플 $([v_1^*, \dots, v_n^*], [u_1, \dots, u_n])$
 - v_1^*, \dots, v_n^* 은 서로 직교하는 벡터들이며 이들의 생성은 $\text{Span } \{v_1, \dots, v_n\}$ 과 동일하다.
 - $i = 1, \dots, n$ 에 대하여

$$\begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix} = \begin{bmatrix} v_1^* & \cdots & v_n^* \end{bmatrix} \begin{bmatrix} u_1 & \cdots & u_n \end{bmatrix}$$

```

1 def aug_orthogonalize(vlist):
2     vstarlist = []
3     sigma_vecs = []
4     for v in vlist:
5         vstar, sigmadict = aug_project_orthogonal(v, vstarlist)
6         vstarlist.append(vstar)
7         sigma_vecs.append(sigmadict)
8     return vstarlist, sigma_vecs

```

10.6 직교여공간 - Orthogonal complement

앞에서는 벡터 b 를 벡터공간 \mathcal{V} 에 직교하게 투영하는 것에 대해 알아보았다. 이번에는 전체 벡터공간을 또 다른 벡터공간에 직교하게 투영하는 것에 대해 알아보자.

10.6.1 직교여공간의 정의

Definition: \mathcal{W} 는 실수상의 벡터공간이라 하고, \mathcal{U} 는 \mathcal{W} 의 부분공간이라 하자. \mathcal{U} 의 \mathcal{W} 에 대한 직교여공간은 다음을 만족하는 집합 \mathcal{V} 라고 정의한다.

$$\mathcal{V} = \{w \in \mathcal{W} : w \text{는 } \mathcal{U} \text{ 내의 모든 벡터에 직교 한다.}\}$$

집합 \mathcal{V} 는 위의 정의에 의해 \mathcal{W} 의 부분집합이며, 다음과 같이 말할 수도 있다.

Lemma: \mathcal{V} 는 \mathcal{W} 의 부분공간이다.

- **Proof:** \mathcal{V} 내의 임의의 두 벡터 v_1 과 v_2 에 대해, $v_1 + v_2$ 도 또한 \mathcal{V} 에 속한다는 것을 보여 준다. \mathcal{V} 의 정의에 의해 v_1 과 v_2 는
 - (1) 둘 다 벡터공간 \mathcal{W} 에 속하고,
 - (2) \mathcal{U} 에 속하는 모든 벡터에 직교한다.
- (1)에 의하면 두 벡터의 합은 \mathcal{W} 에 속한다. (2)를 9.3.2의 Orthogonality Property 2에 대입하면 이 합은 \mathcal{U} 에 속하는 모든 벡터에 직교한다. 따라서 $v_1 + v_2$ 는 \mathcal{V} 에 속한다.
- 마찬가지로 임의의 벡터 $v \in \mathcal{V}$ 와 임의의 스칼라 $\alpha \in \mathbb{R}$ 에 대해, αv 가 \mathcal{V} 에 속한다는 것을 보여야 한다. v 는 벡터공간 \mathcal{W} 에 속하므로, αv 도 또한 \mathcal{W} 에 속한다. v 는 \mathcal{U} 에 속하는 모든 벡터에 직교하므로, 9.3.2의 Orthogonality Property 1에 따르면 αv 도 또한 \mathcal{U} 에 속하는 모든 벡터에 직교한다. 따라서 αv 는 \mathcal{V} 에 속한다.

Example 10.6.3: $\mathcal{U} = \text{Span}\{[1, 1, 0, 0], [0, 0, 1, 1]\}$ 이라고 하자. \mathcal{V} 는 \mathbb{R}^4 에 속하는 \mathcal{U} 의 직교여공간으로 나타내보자. 어떤 벡터들이 \mathcal{V} 에 대한 기저를 형성할까?

\mathcal{U} 내의 모든 벡터는 $[a, a, b, b]$ 형태를 가진다. 그러므로, $[c, -c, d, -d]$ 형태의 임의의 벡터는 \mathcal{U} 내의 모든 벡터에 직교한다.

예를 들어, $\text{Span}\{[1, -1, 0, 0], [0, 0, 1, -1]\}$ 에 속하는 모든 벡터는 \mathcal{U} 내의 모든 벡터에 직교하므로, $\text{Span}\{[1, -1, 0, 0], [0, 0, 1, -1]\}$ 은 \mathcal{V} 의 부분공간이며 \mathbb{R}^4 에 속하는 \mathcal{U} 의 직교여공간이다.

$\mathcal{U} \oplus \mathcal{V} = \mathbb{R}^4$ 이고 $\dim \mathcal{U} + \dim \mathcal{V} = 4$ 라는 것을 알고 있으며, $\{[1, 1, 0, 0], [0, 0, 1, 1]\}$ 은 선형독립이므로, $\dim \mathcal{U} = 2$ 이고 $\dim \mathcal{V} = 2$ 라고 할 수 있다. 또한 $\{[1, -1, 0, 0], [0, 0, 1, -1]\}$ 은 선형독립이고, $\dim \text{Span}\{[1, -1, 0, 0], [0, 0, 1, -1]\} = 2$ 이다. 그러므로, $\text{Span}\{[1, -1, 0, 0], [0, 0, 1, -1]\}$ 은 \mathcal{V} 와 동일하다.

10.6.2 직교여공간과 직합(Direct sum)

이제, 직교여공간과 직합(direct sum) 사이의 연관성을 알아보자.

Lemma: \mathcal{V} 는 \mathcal{U} 의 \mathcal{W} 에 대한 직교여공간이라고 하면, $\mathcal{U} \cap \mathcal{V}$ 에 속하는 유일한 벡터는 영벡터이다.

- **Proof:** \mathcal{V} 에 속하는 벡터 u 는 \mathcal{U} 내의 모든 벡터에 직교한다. 만약 u 도 또한 \mathcal{U} 에 속하면, u 는 자기자신과 직교한다. 즉, $\langle u, u \rangle = 0$ 이다. 9.1.1의 norm의 성질에 의하면 u 가 영벡터임을 의미한다.

따라서, 위의 Lemma에 의해 아래와 같이 직합 $\mathcal{U} \oplus \mathcal{V}$ 를 형성할 수 있다. (7.3.1 참고)

$$\{u + v : u \in \mathcal{U}, v \in \mathcal{V}\}$$

아래의 Lemma는 \mathcal{W} 가 \mathcal{U} 와 \mathcal{V} 의 직합이고, 따라서 \mathcal{U} 와 \mathcal{V} 는 \mathcal{W} 의 여부분공간(complementary subspace)이라는 것을 보여준다.

Lemma: 만약 \mathcal{U} 의 \mathcal{W} 에 대한 직교여공간이 \mathcal{V} 이면,

$$\mathcal{U} \oplus \mathcal{V} = \mathcal{W}$$

- **Proof:** 증명하는데 두 가지 방법이 있다.

- $\mathcal{U} \oplus \mathcal{V}$ 의 모든 원소는 $u \in \mathcal{U}$ 와 $v \in \mathcal{V}$ 에 대해 $u + v$ 형태를 가진다. \mathcal{U} 와 \mathcal{V} 는 둘 다 벡터공간 \mathcal{W} 의 부분집합이므로, 합 $u + v$ 는 \mathcal{W} 에 속한다. 즉, $\mathcal{U} \oplus \mathcal{V} \subseteq \mathcal{W}$ 임을 보여준다.
- \mathcal{W} 에 속하는 임의의 벡터 b 에 대해, $b = b^{\parallel \mathcal{U}} + b^{\perp \mathcal{U}}$ 라고 나타내자. $b^{\parallel \mathcal{U}}$ 는 b 의 \mathcal{U} 상으로의 투영이고 $b^{\perp \mathcal{U}}$ 는 b 의 \mathcal{U} 에 직교하는 투영이다. $b^{\parallel \mathcal{U}}$ 는 \mathcal{U} 에 속하고 $b^{\perp \mathcal{U}}$ 는 \mathcal{V} 에 속한다. 따라서, b 는 \mathcal{U} 에 속하는 벡터와 \mathcal{V} 에 속하는 벡터의 합이다. 즉, $\mathcal{W} \subseteq \mathcal{U} \oplus \mathcal{V}$ 임을 보여준다.

10.6.3 생성 또는 아핀 hull로 주어진 \mathbb{R}^3 평면의 법선

"평면의 법선(normal to a plane)"에서 법선(normal)이란, 수직이라는 의미이다.

예를 들어 두 개의 3-벡터 u_1 과 u_2 의 생성(Span)된 평면이라고 하자. 이 경우, 이 평면은 2차원의 벡터공간 \mathcal{U} 이다. n 은 \mathcal{U} 에 직교하는 영이 아닌 벡터라고 하면, $\text{Span}\{n\}$ 은 \mathbb{R}^3 에 속하는 \mathcal{U} 의 직교여공간의 부분공간이다. 또한, 7.3 직합에서 Direct-Sum Dimension에 의하면, 직교여공간의 차원은 $\dim \mathbb{R}^3 - \dim \mathcal{U} = 1$ 이다. 따라서, 직교여공간은 $\text{Span}\{n\}$ 이다. $\text{Span}\{n\}$ 에 속하는 임의의 영이 아닌 벡터는 법선 역할을 한다. 보통 법선벡터는 $\text{Span}\{n\}$ 에 속하는 norm이 1인 벡터를 말한다.

Example 10.6.6: 위의 10.4에서 예제코드에서도 보았듯이, $\text{Span}\{[8, -2, 2], [0, 3, 3]\}$ 에 직교하는 하나의 영이 아닌 벡터는 $[-1, -2, 2]$ 이고, 이것은 법선벡터이다. norm이 1인 법선을 구하는 방법은 $[-1, -2, 2]$ 의 norm으로 나누면 된다. 따라서, 법선은 $[-\frac{1}{9}, -\frac{2}{9}, \frac{2}{9}]$ 이다.

10.6.4 직교여공간, 영공간, 소멸자

A 는 R 상의 $R \times C$ 행렬이라 하면, A 의 영공간(null space)은 $Au = 0$ 이 되는 C -벡터들 u 의 집합이다. 행렬 -벡터 곱셈의 도트곱 정의에 의하면, A 의 각 행과의 도트곱이 영이 되는 C -벡터들 u 의 집합이다. \mathbb{R} 상의 벡터들에 대한 내적은 도트곱이므로, \mathbb{R}^C 에 속하는 Row A 의 직교여공간은 Null A 임을 의미한다. 또한, Null A 는 Row A 의 소멸자(annihilator)이다. 이것은 \mathbb{R}^C 의 임의의 부분공간 U 에 대해 \mathbb{R}^C 에 속하는 U 의 직교여공간은 소멸자 U° 임을 의미한다.

따라서, \mathbb{R} 상의 임의의 벡터공간 W 와 W 임의의 부분공간 U 에 대한 U 의 W 에 대한 직교여공간(U°)의 직교여공간($(U^\circ)^\circ$)은 U 자신이라는 것을 알 수 있다.

10.6.5 방정식으로 주어진 \mathbb{R}^3 평면의 법선

다시 평면의 법선을 찾는 문제로 가보자. 평면은 다음과 같이 선형방정식에 대한 해집합으로 주어진다.

$$\{[x, y, z] \in \mathbb{R}^3 : [a, b, c] \cdot [x, y, z] = d\}$$

4.6.1에서 보았듯이, 해집합은 아래의 동차선형방정식에 대한 해집합의 평행이동이라 할 수 있다.

$$\{[x, y, z] \in \mathbb{R}^3 : [a, b, c] \cdot [x, y, z] = 0\}$$

$U = Span\{[a, b, c]\}$ 하고, 해집합 $\{[x, y, z] \in \mathbb{R}^3 : [a, b, c] \cdot [x, y, z] = 0\}$ 은 소멸자 U° 이다. 소멸자 U° 에 속하는 벡터들로 구성된 평면의 법선을 찾고자 한다. 소멸자 U° 에 직교하는 벡터들의 집합은 소멸자의 소멸자 즉, $(U^\circ)^\circ$ 이다. 이것은 U 자기자신이다. 따라서, 법선이 될 수 있는 하나의 후보 벡터로는 $[a, b, c]$ 가 된다.

10.6.6 직교여공간 계산하기

W 의 부분공간 U 에 대한 기저 u_1, \dots, u_k 와 W 에 대한 기저 w_1, \dots, w_n 이 있다고 해보자. W 에 속하는 U 직교여공간에 대한 기저를 위에서 구현한 `orthogonalize()` 함수를 이용해 계산해보자.

입력벡터들 `vlist = [u1, ..., uk, w1, ..., wn]`의 출력 벡터인 `vstarlist = [u1*, ..., uk*, w1*, ..., wn*]`은 `vlist` 와 마찬가지로 동일한 공간, 즉 차원이 n 인 W 벡터공간을 생성한다. 따라서, $u_1^*, \dots, u_k^*, w_1^*, \dots, w_n^*$ 중 n 개는 영이 아니다. 또한, u_1^*, \dots, u_k^* 는 u_1, \dots, u_k 와 동일한 공간을 가지며, u_1, \dots, u_k 는 선형독립이기 때문에 모두 영이 아니다. 나머지 벡터들 w_1^*, \dots, w_n^* 중 $n - k$ 개는 영이 아니다. 따라서 모든 원소는 u_1, \dots, u_k 에 직교하며 U 에 속하는 모든 벡터에 직교한다. 따라서, U 의 직교여공간에 있다. 직교여공간은 $n - k$ 차원을 가진다.

아래의 예제코드는 직교여공간에 대한 기저를 구하는 코드이다.

```
1 def find_orthogonal_complement(u_basis, w_basis):
2     vstarlist = orthogonalize(u_basis + w_basis)
3     return [v for v in vstarlist if v.count(0) is not len(v)]
```

Example 10.6.7: 위의 `find_orthogonal_complement()` 함수를 이용해 \mathbb{R}^3 에 속하는 $\text{Span}\{[8, -2, 2], [0, 3, 3]\}$ 의 직교여공간에 대한 기저를 찾아보자. \mathbb{R}^3 에 대한 표준 기저, 즉, $[1, 0, 0], [0, 1, 0], [0, 0, 1]$ 을 사용한다.

```

1  >>> u_basis = [[8, -2, 2], [0, 3, 3]]
2  >>> w_basis = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
3
4  >>> find_orthogonal_complement(u_basis, w_basis)
5  [[8, -2, 2],
6  [0.0, 3.0, 3.0],
7  [0.1111111111111116, 0.2222222222222222, -0.2222222222222222],
8  [-8.326672684688674e-17, 1.6653345369377348e-16, 5.551115123125783e-17],
9  [8.326672684688674e-17, 5.551115123125783e-17, 1.6653345369377348e-16]]
```

출력 결과에서 처음과 두번째 벡터는 $\text{Span}\{[8, -2, 2], [0, 3, 3]\}$ 의 원소이고, 네번째와 다섯번째는 영벡터이다. 따라서 위의 직교여공간의 기저는 $[-\frac{1}{9}, -\frac{2}{9}, \frac{2}{9}]$ 이다.

10.7 QR 인수분해

이제 앞에서 배운 내용들을 토대로 행렬 인수분해에 대해 살펴보자. 행렬 인수분해는 수학적 및 계산적 역할을 한다.

- 수학적(*Mathematical*): 행렬 인수분해는 행렬의 본질에 대한 통찰(?)을 제공한다. 행렬에 대해 생각하는 새로운 방식을 제시한다.(나도 이해할 수 있겠지?...)
- 계산적(*Computational*): 행렬 인수분해는 행렬이 포함된 계산문제들에 대한 해를 계산하는 방안을 제공한다.

10.7.1 직교 및 열-직교 행렬

Definition: 서로 직교하는 벡터들은 만약 *norm*이 모두 1이면 정규직교(*orthonormal*)한다고 한다. 행렬은 만약 그 열들이 정규직교하면 열-직교(*column-orthogonal*)라고 한다. 정방 열-직교행렬(*square column-orthogonal matrix*)은 직교행렬이라 한다.

Q 는 열-직교 행렬이라 하고 그 열들을 q_1^*, \dots, q_n^* 이라 하고, Q^T 의 행들 또한 정규직교이다. 아래와 같이 행렬곱셈 $Q^T Q$ 를 해보자.

$$Q^T Q = \begin{bmatrix} q_1^* \\ \vdots \\ q_n^* \end{bmatrix} \begin{bmatrix} q_1^* & \cdots & q_n^* \end{bmatrix}$$

$Q^T Q$ 의 ij 번째 원소는 Q^T 행렬의 행- i 와 Q 행렬의 열- j 의 도트곱이다. 따라서, ij 원소는 $q_i^* \cdot q_j^*$ 이다. 만약 $i = j$ 라면, $q_i^* \cdot q_i^*$ 이며 $(q_i^*)^2$ 즉 *norm*의 제곱이므로 1이다. 만약 $i \neq j$ 라면, 서로 직교하는 두 벡터들의 내적이므로 그 값은 영(0)이 된다. 따라서 $Q^T Q$ 는 단위행렬(*Identity Matrix*)이 된다.

Lemma: 만약 Q 가 열-직교행렬이면 $Q^T Q$ 는 단위행렬이다.

Corollary(직교행렬의 역): 만약 Q 가 직교행렬이면, 그 역행렬은 Q^T 이다.

10.7.2 행렬의 QR 인수분해 정의하기

Definition: $m \times n$ 행렬 $A (m \geq n)$ 의 QR 인수분해는 $A = QR$ 이다. 여기서, Q 는 $m \times n$ 열-직교 행렬 Q 이고, R 은 상삼각행렬이다.

$$\left[\begin{array}{c} A \end{array} \right] = \left[\begin{array}{c} Q \end{array} \right] \left[\begin{array}{c} R \end{array} \right]$$

Lemma: A 의 QR 인수분해에서, 만약 A 의 열들이 선형독립이면 $\text{Col } Q = \text{Col } A$ 이다.

파이썬의 `numpy` 모듈에서는 `numpy.linalg.qr` 를 통해 QR 분해를 계산할 수 있다(직접 구현하려고 했으나.. 시간이 부족했습니다.. ㅜㅜ)

```
1  >>> import numpy as np
2
3  >>> A = [[2, 0, 0], [1, 2, 2], [1, 0, 2]]
4
5  >>> Q, R = np.linalg.qr(A)
6
7  >>> print('Q =\n', Q)
8  >>> print('R =\n', R)
9  Q =
10 [[ -8.16496581e-01   3.65148372e-01  -4.47213595e-01]
11 [ -4.08248290e-01  -9.12870929e-01  -1.21369485e-17]
12 [ -4.08248290e-01   1.82574186e-01   8.94427191e-01]]
13 R =
14 [[-2.44948974 -0.81649658 -1.63299316]
15 [ 0.          -1.82574186 -1.46059349]
16 [ 0.           0.          1.78885438]]
```

10.8 QR 인수분해를 사용하여 행렬방정식 $Ax = b$ 풀기

10.8.1 정방행렬인 경우

실수상의 행렬방정식 $Ax = b$ 를 고려해보자. 행렬 A 는 정방행렬이고 A 의 열들은 선형독립인 경우, QR 인수분해를 기반으로 하여 이 방정식을 푸는 방법이 있다.

A 의 열들이 선형독립이라 하고 $A = QR$ 은 A 의 QR 인수분해라고 해보자. 이때, 다음 방정식을 만족하는 벡터를 찾고자 한다.

$$Ax = b$$

A 에 QR 을 대입하면 다음을 얻는다.

$$QRx = b$$

양변의 항의 왼쪽에 Q^T 을 곱하면 다음이 얻어진다.

$$Q^T QRx = Q^T b$$

Q 의 열들은 정규직교이므로 $Q^T Q$ 는 단위행렬 I 이고 다음과 같이 쓸 수 있다.

$$IRx = Q^T b$$

$$Rx = Q^T b$$

따라서 $Ax = b$ 를 만족하는 임의의 벡터 \hat{x} 는 또한 식 $Rx = Q^T b$ 를 만족해야 한다.

10.8.2 정방행렬인 경우 솔루션의 정확성

- $Ax = b$ 에 대한 임의의 해는 $Rx = Q^T b$ 에 대한 해라는 것은 보여주었다.
- $Rx = Q^T b$ 에 대한 해는 $Ax = b$ 에 대한 해라는 것을 보여야 한다.

Theorem: A 는 정방행렬이고 그 열들은 선형독립이라고 하면, 10.8.1에서 얻어진 벡터 \hat{x} 은 방정식 $Ax = b$ 를 만족한다.

- **Proof:**

$$R\hat{x} = Q^T b$$

- 양변의 항의 왼쪽에 Q 를 곱하면 다음을 얻는다.

$$QR\hat{x} = QQ^T b$$

- 이것은 다음과 동일하다.

$$A\hat{x} = QQ^T b$$

- A 는 정방행렬이기 때문에 Q 도 정방행렬이다. $QQ^T b = b$ 이고 다음이 성립한다.

$$A\hat{x} = b$$

위의 10.8.1과 10.8.2에서는 다음과 같이 특수한 경우의 행렬방정식 푸는 방법이다.

- 필드가 \mathbb{R} 일 때
- A 의 열들이 선형독립일 때
- A 가 정방행렬일 때

10.8.3 최소제곱 문제

필드는 \mathbb{R} 이고 A 의 열들은 선형독립이라고 가정하자. A 는 $R \times C$ 행렬이라 하고 함수 $f_A : \mathbb{R}^C \rightarrow \mathbb{R}^R$ 는 $f_A(x) = Ax$ 로 정의하자. 정의역은 \mathbb{R}^C 이고, 정의역의 차원은 $|C|$ 이다. 공역의 차원은 $|R|$ 이다. A 의 행 개수가 열 개수가 많은 경우 즉 $R > C$ 이면 공역의 차원은 치역의 차원보다 더 크다. 따라서, 공역에는 치역에 없는 벡터들이 있으므로 f_A 는 전사함수가 아니다. 벡터 b 는 이러한 벡터 중 하나라고 해보자. 즉 $Ax = b$ 에 대한 해는 없다. 이러한 경우 두 가지 문제를 생각해 볼 수 있다.

- A 의 열들의 선형결합 중에서 b 에 가장 가까운 벡터 찾기
- 가장 가까운 벡터를 선형결합으로 표현할 수 있는 계수(좌표)들 찾기

직교화는 첫 번째 문제를 풀 수 있다. b 에 가장 가까운 점은 b^{\parallel} 이고, 이것은 A 의 열공간상으로의 투영이다.

두 번째 문제는 최소제곱(least squares)법을 이용하여 해결할 수 있다. 최소제곱법은 주어진 벡터 x 에 대해서 벡터 $b - Ax$ 를 잉여벡터(residual vector, 오차 error라고도 함)의 norm 즉, $\|Ax - b\|$ 를 최소화하는 벡터 \hat{x} 를 찾는 것이다.

10.8.4 열-직교해렬의 열들에 대한 좌표 표현

여기서 잊지말아야 할것은 정방행렬에 대한 문제가 아닌 $R > C$ 인 행의 개수가 열의 개수보다 많을 때의 경우에 대한 설명이다.

Lemma: Q 는 열-직교 기저라 하고 $\mathcal{V} = \text{Col } Q$ 라고 하면, 정의역이 Q 의 행-라벨 집합과 동일한 임의의 벡터 b 에 대해, $Q^T b$ 는 Q 의 열들에 대한 $b^{\parallel \mathcal{V}}$ 의 좌표 표현이고, $QQ^T b$ 는 $b^{\parallel \mathcal{V}}$ 이다.

- **Proof:** $b = b^{\perp \mathcal{V}} + b^{\parallel \mathcal{V}}$ 라고 나타내면, $b^{\parallel \mathcal{V}}$ 는 \mathcal{V} 내에 있으므로, Q 의 열 q_1, \dots, q_n 의 선형결합으로 표현될 수 있다.

$$b^{\parallel \mathcal{V}} = \alpha_1 q_1 + \dots + \alpha_n q_n$$

- 따라서, $b^{\parallel \mathcal{V}}$ 의 좌표 표현은 $[\alpha_1, \dots, \alpha_n]$ 이다. 이제 이 벡터가 $Q^T b$ 와 동일하다는 것을 보여야 한다. 앞의 10.7.1에서 보았듯이 행렬-벡터 곱셈의 도트곱으로 나타내면 $Q^T b$ 의 원소 j 는 Q 의 열 j 와 b 의 도트곱이다. 이 도트곱은 α_j 와 동일하다는 것을 보여준다.

$$\begin{aligned} \langle q_j, b \rangle &= \left\langle q_j, b^{\perp \mathcal{V}} + b^{\parallel \mathcal{V}} \right\rangle \\ &= \langle q_j, b^{\perp \mathcal{V}} \rangle + \langle q_j, b^{\parallel \mathcal{V}} \rangle \\ &= 0 + \langle q_j, \alpha_1 q_1 + \dots + \alpha_n q_n \rangle \\ &= \alpha_1 \langle q_j, q_1 \rangle + \dots + \alpha_n \langle q_j, q_n \rangle + \dots + \alpha_n \langle q_j, q_n \rangle \\ &= \alpha_j \end{aligned}$$

- $j = 1, \dots, n$ 에 대해, $\alpha_j = \langle q_j, b \rangle$ 이기 때문에 이것은 $Q^T b$ 가 q_1, \dots, q_n 에 대한 $b^{\parallel \mathcal{V}}$ 의 좌표표현이라는 것을 알 수 있다.
- 벡터의 좌표표현에서 벡터 자체를 얻기 위해서는 열들이 기저를 형성하는 행렬, 여기서는 Q 를 곱한다. 따라서, $QQ^T b$ 는 $b^{\parallel \mathcal{V}}$ 자신이다.

10.8.5 A 의 행 개수가 열 개수보다 더 많을 때 QR 분해를 이용해 풀기

최소제곱 문제와 마찬가지로 $\|Ax - b\|$ 를 최소화 하는 벡터 \hat{x} 를 찾는 것이다. 이것은 $A\hat{x}$ 는 \mathcal{V} 상으로의 투영 $b^{\parallel \mathcal{V}}$ 와 동일하다. 여기서 \mathcal{V} 는 A 의 $\text{Col } A$ 열공간이다.

$$R\hat{x} = Q^T b$$

$$QR\hat{x}=QQ^Tb$$

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

$$A\hat{x}=b^{||\mathcal{V}}$$