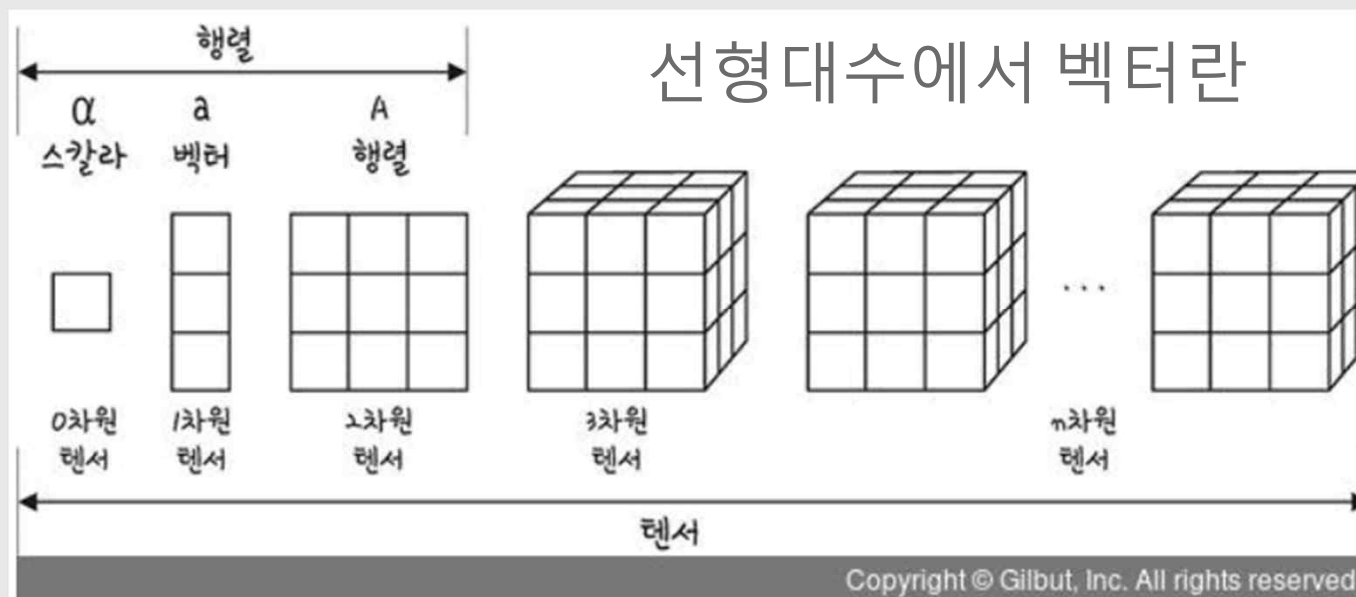




# “DS/AI 프로그래밍”

10주차  
인공신경망 기초  
(선형대수)

J.-K.- Seo  
데이터과학원



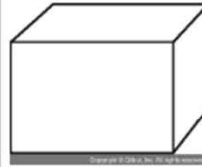


데이터 분석을 하려면 수많은 숫자로 구성된 데이터를 다루어야 합니다. 데이터 하나의 숫자가 수만 개에서 수억 개로 구성되어 있을 수도 있고, 이러한 데이터 수만 개가 집합 하나를 이루고 있을 수도 있습니다.

선형대수(linear algebra)는 데이터 분석에 필요한 각종 계산을 돕는 학문입니다.

선형대수에서 다루는 데이터는 개수나 형태에 따라 크게 스칼라(scalar), 벡터(vector), 행렬(matrix), 텐서(tensor) 유형으로 나뉩니다.

일반적으로 다음과 같이 스칼라, 벡터, 행렬을 평면상에서 이해하는 경향이 있는데, 이것은 수학적 표현에 한계가 있어서 그런 것이므로 텐서를 n차원의 공간에서 생각할 수 있어야 합니다.

구분	스칼라	벡터	행렬	텐서
영문 표기	Scalar	Vector	Matrix	Tensor
표기	$x \in \mathbb{R}$	$x \in \mathbb{R}_n$	$x \in \mathbb{R}_m \times n$	$A$
차원(dimension)	0차원	1차원	2차원	3차원 이상
공간에서 표현				
파이썬 코드 예시	<pre>x = np.array(1.2)</pre>	<pre>x = np.array([1, 2, 3])</pre>	<pre>x = np.array([[1, 2, 3], [4, 5, 6]])</pre>	<pre>x = np.array([[[[1, 2, 3], [4, 5, 6], [10, 20, 30], [200, 300, 400]]]])</pre>

## 인공지능에서는 왜 벡터를 사용하는가?

인공지능에서 벡터 의미:

인공지능에서 벡터는 **숫자의 집합 및 배열**이라고 이해하면 쉽습니다. 인공지능에서는 특성 벡터 (feature vector)라는 명칭을 사용합니다.

이직 유무	평균 급여	업무 시간	직군	연봉
0	96	3	support	low
1	97	3	sales	low
1	98	4	marketing	medium
1	99	2	IT	low
0	112	6	accounting	high

표에서 직군과 연봉을 숫자로 변환해야 합니다. 이해하기 쉽도록 직군 특성에서 'support'를 '1'로, 'sales'를 '2'로, 'marketing'을 '3'으로, 'IT'를 '4'로, 'accounting'을 '5'로 표현할 수 있습니다. 연봉의 'low'를 '1'로, 'medium'을 '2'로, 'high'를 '3'으로 표현하면 표 10-5와 같습니다.

이직 유무	평균 급여	업무 시간	직군	연봉
0	96	3	1	1
1	97	3	2	1
1	98	4	3	2
1	99	2	4	1
0	112	6	5	3

따라서 첫 번째 행은 벡터 [0, 96, 3, 1, 1]처럼 표현할 수 있습니다  
 (물론 다른 특성과 평균 급여의 데이터 범위가 다르기 때문에 전처리는 필요합니다).

## 벡터의 사칙연산

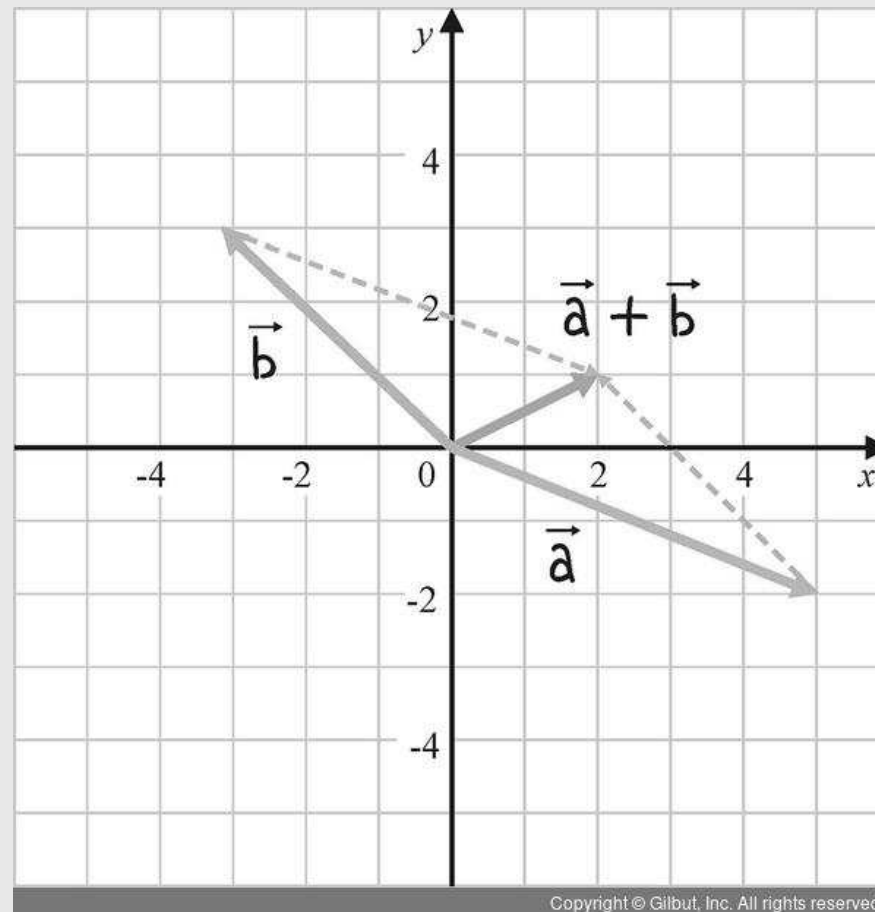
$$\vec{v} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \vec{w} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Copyright © Gilbut, Inc. All rights reserved.

$$\vec{v} + \vec{w} = \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{bmatrix}$$

Copyright © Gilbut, Inc. All rights reserved.

평행사변형 법칙과 삼각형 법칙에 대한 설명이 조금 어려운데, 예시로 확인해 보겠습니다. 예를 들어 두 벡터가  $\vec{a} = \begin{bmatrix} 5 \\ -2 \end{bmatrix}$ ,  $\vec{b} = \begin{bmatrix} -3 \\ 3 \end{bmatrix}$  ( $\vec{a}, \vec{b} \in \mathbb{R}^2$ )일 때 평행사변형 법칙과 삼각형 법칙을 알아보겠습니다.



## 벡터 덧셈의 성질

벡터의 덧셈은 다음 성질이 있습니다.

(1) 교환 법칙:  $A + B = B + A$

예시

$\vec{v} = (12, 17)$ ,  $\vec{w} = (2, -5)$ 일 때,

$$\vec{v} + \vec{w} = (12, 17) + (2, -5) = (14, 12)$$

$$\vec{w} + \vec{v} = (2, -5) + (12, 17) = (14, 12)$$

(2) 결합 법칙:  $(A + B) + C = A + (B + C)$

예시

$\vec{v} = (12, 17)$ ,  $\vec{w} = (2, -5)$ ,  $\vec{u} = (4, -3)$ 일 때,

$$(\vec{v} + \vec{w}) + \vec{u} = ((12, 17) + (2, -5)) + (4, -3) = (18, 9)$$

$$\vec{v} + (\vec{w} + \vec{u}) = (12, 17) + ((2, -5) + (4, -3)) = (18, 9)$$

(3) 벡터 덧셈의 항등원이 존재:  $A + 0 = A$

(4) 벡터 덧셈의 역원이 존재:  $A + (-A) = 0$





파이썬의 NumPy 라이브러리(고성능 수치 계산에 이용)를 사용하면 행렬이나 벡터 연산을 쉽게 할 수 있습니다. 먼저 파이썬의 기본 라이브러리를 사용했을 때 벡터의 덧셈을 알아보시다.

In [17]:

```
# 파이썬 기본 라이브러리를 사용한 벡터의 덧셈
x = [2,3]
y = [3,1]

# zip() 함수를 사용하여 두 벡터의 첫 번째 원소끼리 더하고,
# 두 번째 원소끼리 더합니다
# 그리고 그 결과를 리스트 형태로 z 변수에 저장합니다
z = [i+j for i, j in zip(x,y)]
print(z)
```

[5, 4]



## NOTE

`zip(*iterable)`

`zip(*iterable)`은 동일한 개수로 된 자료형을 묶어 주는 역할을 하는 함수입니다. 여기에서 사용한 `*iterable`은 '반복 가능(iterable)한 자료형 여러 개를 입력할 수 있다'는 의미로 다음과 같이 사용할 수 있습니다.

In [18]:

```
list(zip([1, 2, 3], [4, 5, 6]))
```

Out [18]:

```
[(1, 4), (2, 5), (3, 6)]
```

# NumPy 라이브러리를 이용한 벡터의 덧셈

```
import numpy as np
```

# x, y 리스트를 NumPy의 Array 객체로 변환한 후 u, v 변수에 각각 저장합니다

```
u = np.array(x)
```

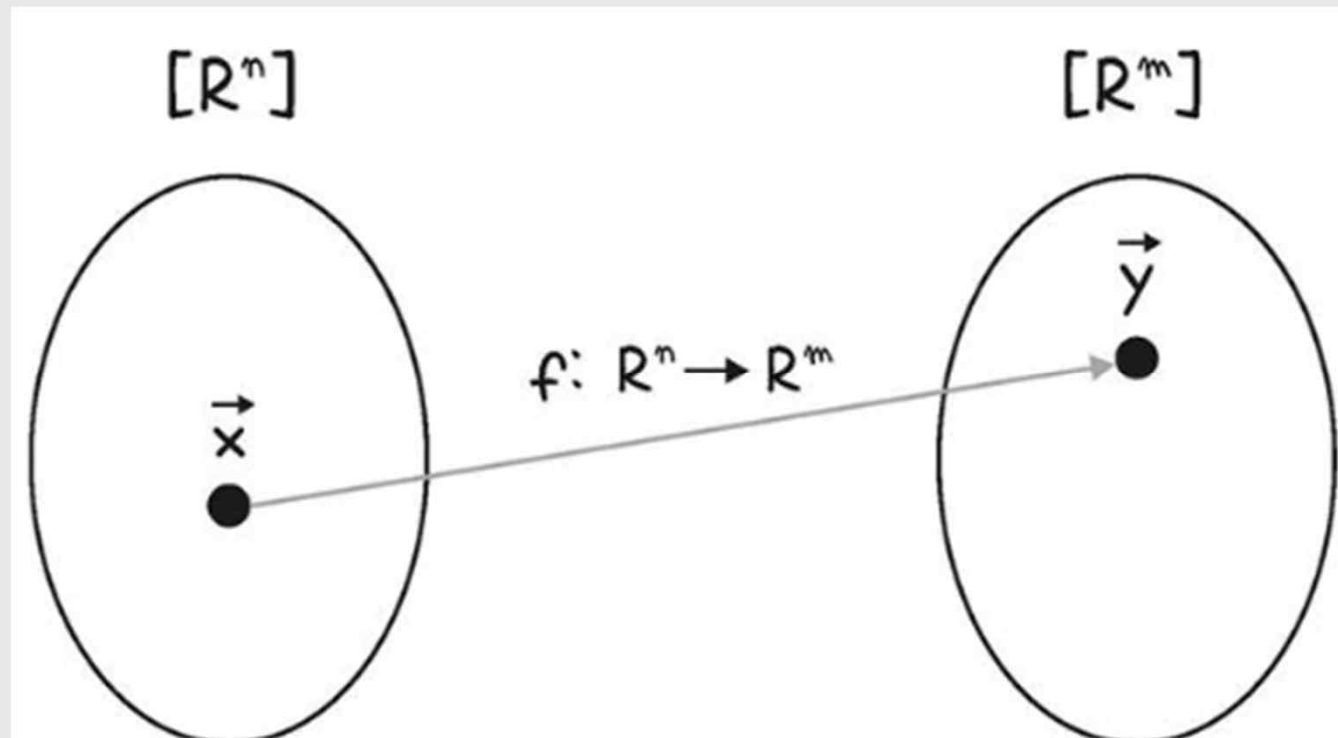
```
v = np.array(y)
```

# u, v 변수에 덧셈 연산자를 적용합니다

```
w = u+v
```

```
print(w)
```

## 벡터의 변환



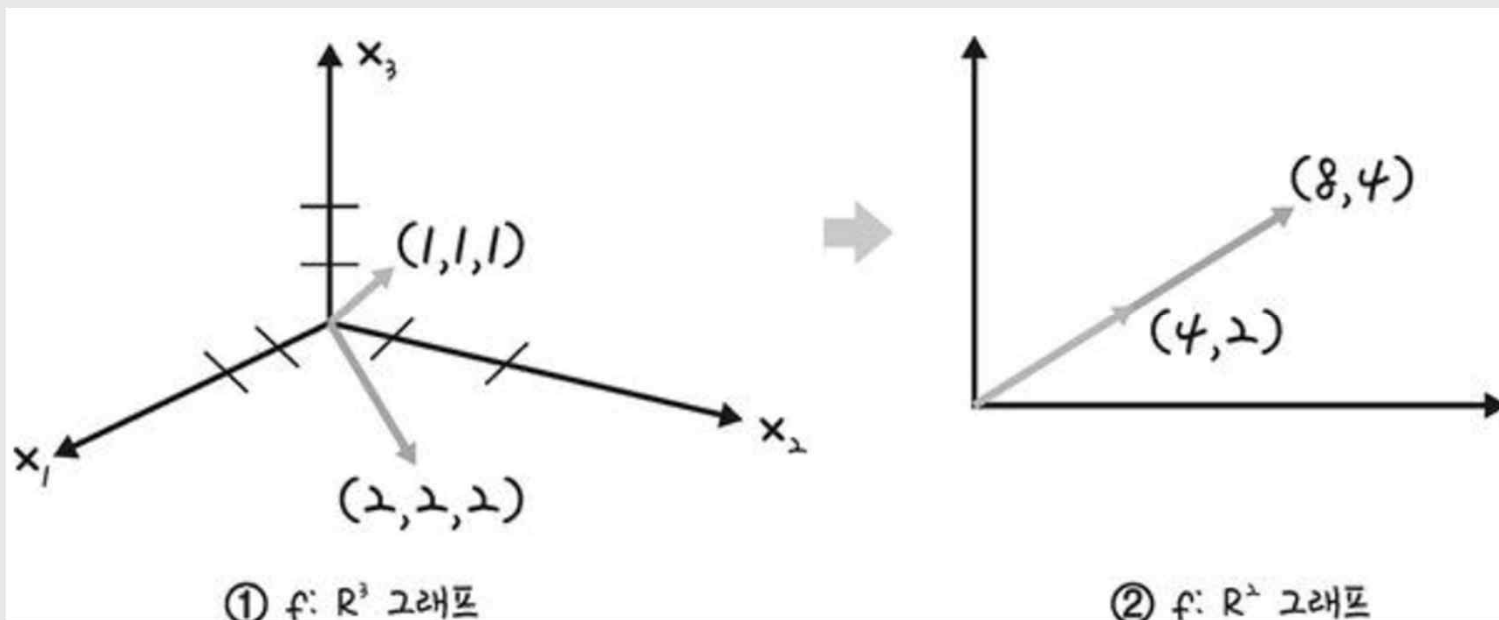
벡터의 변환은 무엇일까요? 예) 벡터의 변환은 3D 공간상에서 벡터에 여러 가지 변환을 수행하는 것입니다. 예시로 정확한 의미를 확인해 보겠습니다.

$$f: R^3 \mapsto R^2$$

$$f \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{bmatrix} x_1 + 3x_2 \\ 2x_3 \end{bmatrix}$$

0이때 (1)  $x_1 = x_2 = x_3 = 1$ 이라고 가정한다면

$$f \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{bmatrix} 1 + (3 \times 1) \\ 2 \times 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix} \text{가 되고,}$$



이때 (1)  $x_1 = x_2 = x_3 = 1$ 이라고 가정한다면

$$f \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{bmatrix} 1 + (3 \times 1) \\ 2 \times 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix} \text{가 되고,}$$

이때 (1)에서 (2)로 변환을 '벡터의 변환(Transformation)'이라고 합니다. 즉,  $x_1, x_2, x_3$ 의 값에 따라 다양한 형태로 변할 수 있습니다.

## 선형 변환

'선형적이다'는 영어로 'linear하다'고 표현합니다. linear란 line(선)의 형용사 형태로, '선형적이다'는 결국 어떤 성질이 변하는 데 그 변수가 1차원적이라는 의미로 유추할 수 있습니다. 즉, 함수에서는 그 모양이 '직선'이라는 의미로 사용합니다. 수학에서 선형성은 임의의 수  $x, y$ 와 함수  $f$ 에 대해 다음 두 조건을 동시에 만족해야 합니다.

- 중첩성:  $f(x + y) = f(x) + f(y)$
- 동질성: 임의의 수  $a$ 에 대해  $f(ax) = af(x)$

선형 변환이란:

선형 변환은 선형 결합을 보존하는 두 벡터 공간 사이의 함수입니다.

- 합의 법칙:  $T(\vec{a} + \vec{b}) = T(\vec{a}) + T(\vec{b})$
- 스칼라 곱의 법칙:  $T(c\vec{a}) = cT(\vec{a})$

## 연습 문제

$$X = \begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix}, Y = \begin{bmatrix} 4 \\ 1 \\ 2 \end{bmatrix}, X + Y = \begin{bmatrix} 6 \\ 4 \\ 7 \end{bmatrix} \text{ 일 때, } f(x, y, z) = (3x + 3y, 3z, 8y + 2x, 4z) \text{ 가 선형 변환임을 증명:}$$

$$(1) T(X) = f(2, 3, 5) = (15, 15, 28, 20)$$

$$T(Y) = f(4, 1, 2) = (15, 6, 16, 8)$$

$$T(X + Y) = f(6, 4, 7) = (30, 21, 44, 28)$$

즉, 다음 합의 법칙을 만족합니다.

$$T(X + Y) = T(X) + T(Y)$$

$$(30, 21, 44, 28) = (15, 15, 28, 20) + (15, 6, 16, 8)$$

$$(2) \text{ 또 } x \text{ 에 임의의 실수 값 } 2 \text{ 를 곱하면 다음과 같습니다. } 2X = 2 \begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \\ 10 \end{bmatrix} = f(4, 6, 10) = 2 \times f(2, 3, 5)$$

따라서 스칼라 곱의 법칙  $T(2X) = 2T(X)$  를 만족하기 때문에 주어진 함수는 선형 변환입니다.



## 역행렬

수에 역수가 있다면 행렬에는 역행렬이 있습니다. 어떤 행렬 A와 곱했을 때 곱셈에 대한 항등원인 단위행렬 E가 나오게 하는 행렬을 행렬 A의 역행렬이라고 합니다.

$$AB = BA = E$$

앞의 식처럼 A와 곱해서 E가 나오게 하는 행렬 B를 A의 역행렬이라고 하며,  $A^{-1}$ 로 표기합니다(A 인버스(inverse)라고 읽습니다).

$$B = A^{-1}$$

일반적으로 행렬의 곱셈에 대한 성질에서는  $AB \neq BA$ 이지만, 행렬과 그 역행렬에서는  $AA^{-1} = A^{-1}A$ 가 성립합니다(동일하게 단위행렬을 도출하므로  $AA^{-1} = A^{-1}A$ 가 성립합니다).

## 행렬식

$n \times n$  인 정방행렬  $A = \{a_{ij}\}$ 의 행렬식을  $\sigma_j = (j_1, j_2, \dots, j_n)$ 으로 된 모든 순열에 대하여 다음의 합으로 정의한다.

$$|A| = \sum_{j=1}^{n!} (\text{sgn } \sigma_j) a_{1j_1} a_{2j_2} \cdots a_{nj_n}$$

$\sigma_j = (j_1, j_2, \dots, j_n)$ 의 순서쌍은  $n$ 개의 자연수를 나열한 순열의 수가 되고, 이 순서쌍에서 나타나는 역순의 수가 짝수이면  $(\text{sgn } \sigma) = +1$ 이라 정의하고 역순의 수가 홀수이면  $(\text{sgn } \sigma) = -1$ 로 정의한다.

참고1: “통계학을 위한 행렬대수, 양완연 저, 도서출판 연학사, 2001년”

참고2: “나무위키, <https://namu.wiki/w/%ED%96%89%EB%A0%AC%EC%8B%9D>”

- 다중 선형성<sup>[2]</sup>

각 열벡터  $\mathbf{v}_i, \mathbf{u} \in F^n$ 와 임의의 스칼라  $a$ 에 대해

$$\det(\mathbf{v}_1 \cdots a\mathbf{v}_i + \mathbf{u} \cdots \mathbf{v}_n) = a \det(\mathbf{v}_1 \cdots \mathbf{v}_i \cdots \mathbf{v}_n) + \det(\mathbf{v}_1 \cdots \mathbf{u} \cdots \mathbf{v}_n)^{[3]}$$

- 교대성<sup>[4][5]</sup>

$$\det(\mathbf{v}_1 \cdots \mathbf{v}_i \cdots \mathbf{v}_j \cdots \mathbf{v}_n) = -\det(\mathbf{v}_1 \cdots \mathbf{v}_j \cdots \mathbf{v}_i \cdots \mathbf{v}_n)^{[6]}$$

- 단위 행렬의 값

$$\det I = \det(\mathbf{e}_j) = 1$$

[2] 각 인수들에 대해 선형이다.

[3] 짧게 쓰자면,  $\det(a^{\delta_{ij}}\mathbf{v}_j + \delta_{ij}\mathbf{u}) = a \det(\mathbf{v}_j) + \det((1 - \delta_{ij})\mathbf{v}_j + \delta_{ij}\mathbf{u})$ . 여기서  $\delta_{ij}$ 는 크로네커 델타이다.

[4] 두 열벡터를 교환하면 부호가 바뀐다. 이 때  $\mathbf{v}_i = \mathbf{v}_j$ 이면  $\det(\mathbf{v}_j) = -\det(\mathbf{v}_j) \Rightarrow 2 \det(\mathbf{v}_j) = 0$ 에서  $\det(\mathbf{v}_j) = 0$ , 즉 두 열벡터가 같을 경우 그 행렬식의 값은 무조건 0이 된다. 행렬식은 전치를 해도 값이 같으므로 두 행벡터가 같은 경우에 대해서도 똑같이 적용할 수 있다.

[5] 다만, 이것은  $1+1=2$ 가 0이 아닐때의 얘기이다.  $F$ 의 표수(characteristic)가 2이면, 2로 나누는 것이 0으로 나누는 것이 되어버리기 때문이다. 이 경우에는 두 벡터가 같을때 행렬식이 0이라는 조건이 추가로 있어야 행렬식의 정의가 된다.

[6] 짧게 쓰자면, 호환  $\sigma = (i \ j)$ 에 대해,  $\det(\mathbf{v}_j) = -\det(\mathbf{v}_{\sigma(j)})$ 이다. 이것이 행렬식의 계산법에서 홀치환-짝치환을 따지는 이유이다.

정리:  
행렬식의 한 행(또는 한 열)의 모든 원소가 0이면 행렬식의 값은 0이다.

정리:  
행렬을 전치시켜도 행렬식의 값은 변하지 않는다. 즉,  $|A'| = |A|$

정리:  
행렬식의 한 행(또는 한 열)의 모든 원소에  $k$  를 곱한 것은 행렬식  $|A|$ 에  $k$  를 곱한 것과 같다.

정의:

$n$  차 행렬식  $|A|$  에서  $i$ 행과  $j$ 열의 원소들을 제외한 나머지로 만들어진 행렬식을 소행렬식(minor)이라 하고  $M_{ij}$  로 표시한다.

정의:

소행렬식에  $+$  또는  $-$  의 부호를 붙인 것을 여인자(cofactor)라 하며, 여인자를  $A_{ij}$  로 표시하기로 하면  $A_{ij} = (-1)^{i+j} M_{ij}$  이다.

[참고1]의 (예제 3-14), (정리3-10), (정리 3-11), (예제 3-16), (예제 3-17)에 의하여 다음을 알 수 있다.

차수가  $n \times n$  인  $n$  차 정방행렬의 역행렬을 구하기 위하여 (정리3-10), (정리 3-11)의 여인자 전개를 이용해 보자.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} A_{11} & A_{21} & \cdots & A_{n1} \\ A_{12} & A_{22} & \cdots & A_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ A_{1n} & A_{2n} & \cdots & a_{nn} \end{bmatrix} = |A| I_n$$

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \frac{1}{|A|} \begin{bmatrix} A_{11} & A_{21} & \cdots & A_{n1} \\ A_{12} & A_{22} & \cdots & A_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ A_{1n} & A_{2n} & \cdots & a_{nn} \end{bmatrix} = I_n$$

따라서  $A$ 의 역행렬  $A^{-1}$ 은

$$A^{-1} = \frac{1}{|A|} \begin{bmatrix} A_{11} & A_{21} & \cdots & A_{n1} \\ A_{12} & A_{22} & \cdots & A_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ A_{1n} & A_{2n} & \cdots & a_{nn} \end{bmatrix}$$

정리:

행렬  $A$ 의 역행렬  $A^{-1}$ 가 존재하려면 다음 조건을 만족해야 한다.

- i)  $A$ 가 정방행렬이어야 하고
- ii)  $|A|$ 의 값이 0이 아니어야 한다.

( $|A| = 0$ 이면  $1/|A|$ 이 존재하지 않기 때문에.)

## 연습 문제

$$A = \begin{bmatrix} 1 & 2 & -4 \\ -2 & 3 & -1 \\ 6 & 5 & 8 \end{bmatrix} \text{의 } \det A \text{를 구하세요.}$$

정리:  
행렬  $A$  의 역행렬은 오직 하나만 존재한다.

정리:  
 $|A^{-1}| = 1/|A|$  이다.

정리:  
 $(A^{-1})^{-1} = A$  이다.

정리:  
 $(A')^{-1} = (A^{-1})'$ 이다.

정리:  
행렬  $A$ 와  $B$ 의 역행렬  $A^{-1}$ 와  $B^{-1}$ 가 존재하면  $(AB)^{-1} = B^{-1}A^{-1}$ 이다.



파이썬에서도 다음과 같이 역행렬을 계산하여 연립방정식을 구현할 수 있습니다.

```
# NumPy 라이브러리를 호출합니다
import numpy as np

# A에 4x4 행렬을 배치합니다
A = np.matrix([[1, 0, 0, 0], [2, 1, 0, 0], [3, 0, 1, 0], [4, 0, 0, 1]])

# A 행렬을 역행렬로 변환하기 위해 numpy.linalg.inv()를 사용합니다
print(np.linalg.inv(A))
```

```
[[ 1, 0, 0, 0] [-2, 1, 0, 0] [-3, 0, 1, 0] [-4, -0, -0, 1]]
```



```
import numpy as np  
A = np.matrix([[2,3],[4,6]])  
print(np.linalg.inv(A))
```

LinAlgError: Singular matrix --- 역행렬이 존재하지 않기 때문에 오류 발생

## 고유 값과 고유 벡터란

고유 벡터(eigenvector)는 선형 변환을 취했을 때 방향(direction)은 변하지 않고 크기(magnitude)만 변하는 벡터를 의미합니다.

여기에서 고유 벡터의 크기가 변한다고 했는데, 얼마나 변할까요? 바로 그 변한 크기가 고유 값(eigenvalue)을 의미합니다. 고유 값이 2라면 기존 벡터 크기의 2배만큼 길어진 것이고, 고유 값이  $\frac{1}{3}$ 이라면 기존 벡터 크기의  $\frac{1}{3}$ 만큼 줄어든 것입니다.



정방행렬  $A$ 에 대해  $A\vec{x} = \lambda\vec{x}$  ( $\lambda$ 는 상수)가 성립하는 0이 아닌 벡터  $\vec{x}$ 가 존재할 때,  $\lambda$  상수를 행렬  $A$ 의 고유 값이라고 하며,  $\vec{x}$ 를 이에 대응하는 고유 벡터라고 합니다.

---

정방행렬

행의 개수와 열의 개수가 같은 행렬 ( $n \times n$  행렬)을 의미합니다

---

$$A\vec{x} = \lambda\vec{x} \text{ (}\lambda\text{는 상수)}$$

이 수식을 행렬로 표현하면 다음과 같습니다.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

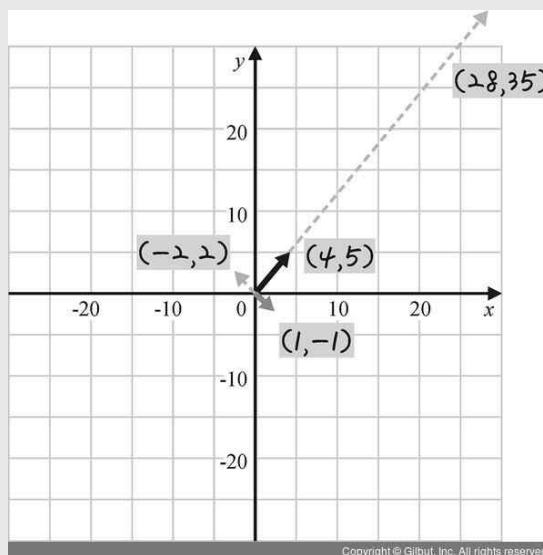


### 예시 1

정방행렬  $A = \begin{bmatrix} 2 & 4 \\ 5 & 3 \end{bmatrix}$ , 고유 값  $\lambda = 7$ , 고유 벡터  $\vec{x} = \begin{bmatrix} 4 \\ 5 \end{bmatrix}$  일 때  $A\vec{x} = \lambda\vec{x}$ 를 만족하는지 알아봅시다.

### 예시 2

정방행렬  $A = \begin{bmatrix} 2 & 4 \\ 5 & 3 \end{bmatrix}$ , 고유 값  $\lambda = -2$ , 고유 벡터  $\vec{x} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$  일 때  $A\vec{x} = \lambda\vec{x}$ 를 만족하는지 알아봅시다.



고유 값을 구하는 과정을 살펴봅시다. 고유 값을 구하는 공식은 다음과 같습니다.

$$A\vec{x} = \lambda\vec{x} \Leftrightarrow (A - \lambda I)\vec{x} = 0 \Leftrightarrow \det(A - \lambda I) = 0$$

앞의 2차 정방행렬  $A = \begin{bmatrix} 2 & 4 \\ 5 & 3 \end{bmatrix}$ 을 이용하여 고유 값을 구해 봅시다.

2차 정방행렬  $A = \begin{bmatrix} 2 & 4 \\ 5 & 3 \end{bmatrix}$ 에 대해  $A\vec{x}$ 는  $\begin{bmatrix} 2 & 4 \\ 5 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ 고, 이것을 풀어 보면 다음과 같습니다.

$$\begin{bmatrix} 2 & 4 \\ 5 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \lambda \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0$$

$$\left( \begin{bmatrix} 2 & 4 \\ 5 & 3 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0$$

$$\begin{bmatrix} 2-\lambda & 4 \\ 5 & 3-\lambda \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0$$

$(A - \lambda I)\vec{x} = 0$ 이므로 다음과 같이 고유 값을 구할 수 있습니다.

$$\begin{bmatrix} 2-\lambda & 4 \\ 5 & 3-\lambda \end{bmatrix} = 0$$

$$(2-\lambda)(3-\lambda) - 4 \times 5 = 0 \quad \leftarrow \text{행렬식 적용}$$

$$\lambda^2 - 5\lambda - 14 = 0$$

$$(\lambda - 7)(\lambda + 2) = 0$$

$$\therefore \lambda = 7, -2$$

(1)  $\lambda = 7$ 에 대응하는 고유 벡터 구하세요.

$$\begin{bmatrix} 2-\lambda & 4 \\ 5 & 3-\lambda \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \vec{x}$$

$$\begin{bmatrix} 2-7 & 4 \\ 5 & 3-7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

식을 방정식으로 풀이하면 다음과 같습니다.

$$-5x_1 + 4x_2 = 0$$

$$5x_1 - 4x_2 = 0$$

따라서  $x_1 = 4, x_2 = 5$ 가 됩니다.

$$\therefore \text{고유 벡터는 } \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \end{bmatrix} \text{입니다.}$$



(2)  $\lambda = -2$ 에 대응하는 고유 벡터  $\vec{x}$  구하세요.

$$\begin{bmatrix} 2-\lambda & 4 \\ 5 & 3-\lambda \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 2-(-2) & 4 \\ 5 & 3-(-2) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 4x_1 + 4x_2 \\ 5x_1 + 5x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} x_1 + x_2 \\ x_1 + x_2 \end{bmatrix} \begin{bmatrix} 4 \\ 5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\therefore x_1 + x_2 = 0 \Leftrightarrow x_1 = -x_2$$

이때  $x_1$ 이 1일 때  $x_2$ 는 -1이 되므로 고유 벡터는 다음과 같습니다.  $\therefore$  고유 벡터는  $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ 입니다.

```
# NumPy 라이브러리를 호출합니다
import numpy as np

# 2차원 행렬 A
# np.linalg.eig(a)는 고유 값과 고유 벡터 도출을 위한 함수입니다
a = np.array([[5, -1], [-2, 1]])
w, v = np.linalg.eig(a)

# 고유 값 구하기
print(w)
print(v)
```

```
[5.44948974 0.55051026] --- 고유 값에 대한 결과값
[[ 0.91209559 0.21927526] --- 고유 벡터에 대한 결과값
 [-0.40997761 0.97566304]]
```

참고로 고유 벡터의 수학적 계산과 파이썬의 고유 벡터 결과가 다른 이유는 고유 벡터를 표시할 때는 보통 길이가 1인 단위 벡터가 되도록 정규화(normalization)하기 때문입니다.



## 연습 문제

행렬  $A = \begin{bmatrix} 1 & 3 \\ 4 & 2 \end{bmatrix}$ 에 대해 행렬  $A$ 의 고유 값과 고유 벡터를 구하세요.

```
import numpy as np
a = np.array([[1, 3], [4, 2]])
w, v = np.linalg.eig(a)
print(w)
print(v)
```

```
[-2.  5.]          --- 고유 값에 대한 결과값
[[-0.70710678 -0.6 ] --- 고유 벡터에 대한 결과값
 [ 0.70710678 -0.8 ]]
```

참고로  $A - \lambda I$ 를 **특성행렬(characteristic matrix)**이라고 하며,  $D(\lambda)$ 는 행렬  $A$ 의 **특성행렬식(characteristic determinant)**이라고 합니다. 그리고  $A - \lambda I = 0$ 은 **특성방정식(characteristic equation)** 혹은 **고유방정식(eigenvalue equation)**이라고 합니다.

$n$ 차 정방행렬  $A$ 의 고유 값은 적어도 하나 이상, 최대  $n$ 개의 서로 다른 고유 값을 갖게 됩니다. 다음 식에 특성방정식을 적용하여 고유 값을 구해 보겠습니다.

고유치와 고유벡터는 “행렬의 대각화”를 비롯하여 각종 “행렬의 분해”, “행렬의 정칙성”, “행렬의 계수(Rank)”, “직교행렬분해”, “차원축소 이론” 등 수많은 행렬의 성질과 이론들에서 다뤄지며 응용되는 개념이므로 심도있는 필요하며 추후 수채해석, 계산과학, 인공지능의 응용 연구에 활용될 수 있다.

# The End

# Reference

- 참고1: “통계학을 위한 행렬대수, 양완연 저, 도서출판 연학사, 2001년”  
참고2: “나무위키, <https://namu.wiki/w/%ED%96%89%EB%A0%AC%EC%8B%9D>”  
참고3: “모두의 인공지능 기초수학, 서지영, (주)도서출판 길벗, 2020년”  
참고4: “Linear Algebra, Jin Ho Kwak, Sungpyo Hong, Birkhäuser Boston, 2004”