# 통계 분석

2019-2020년 겨울 계절 학기

강봉주

# 벡터와 행렬

# 벡터

# 벡터와 행렬

## [벡터의 표현]

- 벡터(vector)는 순서가 있는 숫자들의 목록이다. 즉 일종의 배열이다. 표현은 대괄호나 괄호를 통하여 표현한다.

$$\begin{bmatrix} 0.61 \\ 0.93 \\ 0.24 \\ 0.27 \end{bmatrix}, \qquad \begin{pmatrix} 0.61 \\ 0.93 \\ 0.24 \\ 0.27 \end{pmatrix}$$

$$\begin{pmatrix} 0.74 & 0.75 & 0.93 & 0.46 \end{pmatrix}$$

# 벡터와 행렬

## [벡터의 표현]

- 벡터(vector)는 순서가 있는 숫자들의 목록이다. 즉 일종의 배열이다. 표현은 대괄호나 괄호를 통하여 표현한다.

```
In [5]: # 벡터의 생성
        v = np.array([0.61, 0.93, 0.24, 0.27])

        print(type(v))
        print(v)

<class 'numpy.ndarray'>
[0.61 0.93 0.24 0.27]
```

**[벡터의 원소]**

- 벡터의 원소(element, entry, coefficient, component)는 배열의 값들을 의미한다.

```
In [41]: # 벡터의 원소
         v = np.array([0.61, 0.93, 0.24, 0.27])
         v[0]

Out[41]: 0.61
```

**[벡터의 크기]**

- 벡터의 크기(size, dimension, length)는 벡터의 원소의 개수를 의미한다.

```
In [42]: # 벡터의 크기
         v = np.array([0.61, 0.93, 0.24, 0.27])
         v.shape

Out[42]: (4,)
```

**[부분 벡터]**

- 벡터의 부분벡터(sub vector)는 쌍점(colon) 기호를 사용하여 표현한다.

$$a_{r:s} = \begin{pmatrix} a_r \\ a_{r+1} \\ \cdots \\ a_s \end{pmatrix}$$

## [부분 벡터]

- 벡터의 부분벡터(sub vector)는 쌍점(colon) 기호를 사용하여 표현한다.

```
In [43]: # 부분 벡터의 생성
         v = np.array([0.61, 0.93, 0.24, 0.27])
         v_sub = v[0:2]
         print(v_sub)

[0.61 0.93]
```

**[특별한 벡터]**

- 모든 원소가 0인 벡터를 0벡터

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

## [특별한 벡터]

- 모든 원소가 0인 벡터를 0벡터

```
In [11]: # 영 벡터
         zeros = np.zeros(shape=(4,))
         print(zeros)

[0. 0. 0. 0.]
```

**[특별한 벡터]**

- 모든 원소가 1인 벡터

$$1_3 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

## [특별한 벡터]

- 모든 원소가 1인 벡터

```
In [47]: # 1 벡터
size = 3
one = np.ones(shape=(size, ))
print(one)

[1. 1. 1.]
```

**[특별한 벡터]**

- 하나의 원소만 1이고 나머지는 모두 0인  단위(unit)벡터

$$e_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, e_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, e_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

## [특별한 벡터]

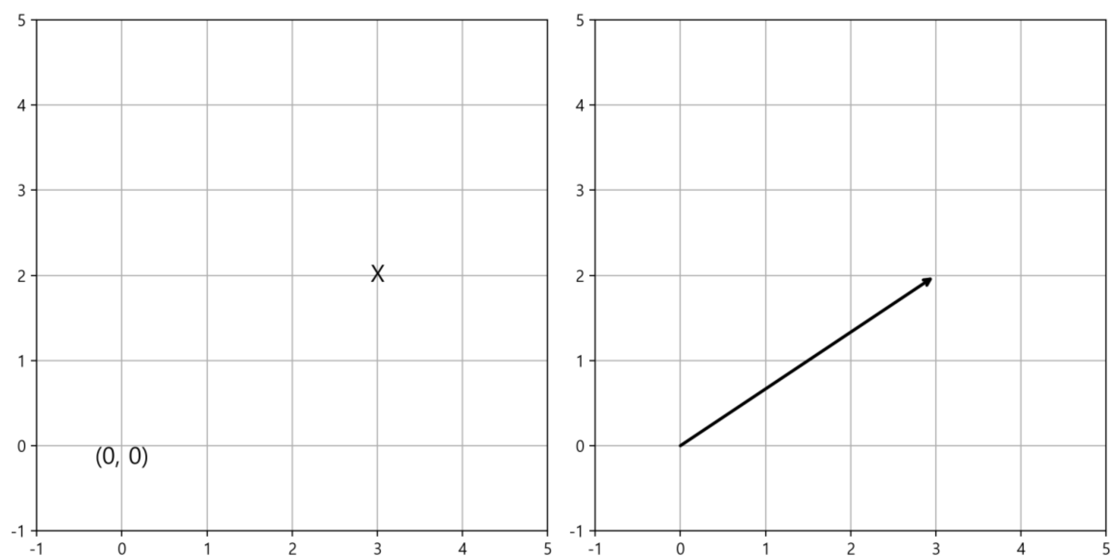
- 하나의 원소만 1이고 나머지는 모두 0인 단위(unit)벡터

In [46]:
```
# 단위 벡터
size = 3
e = np.diag(np.ones(shape=(size,)))
print(e)

# 하나의 단위  벡터의 예
print('하나의 단위벡터:', e[:,1])
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
하나의 단위벡터: [0. 1. 0.]
```

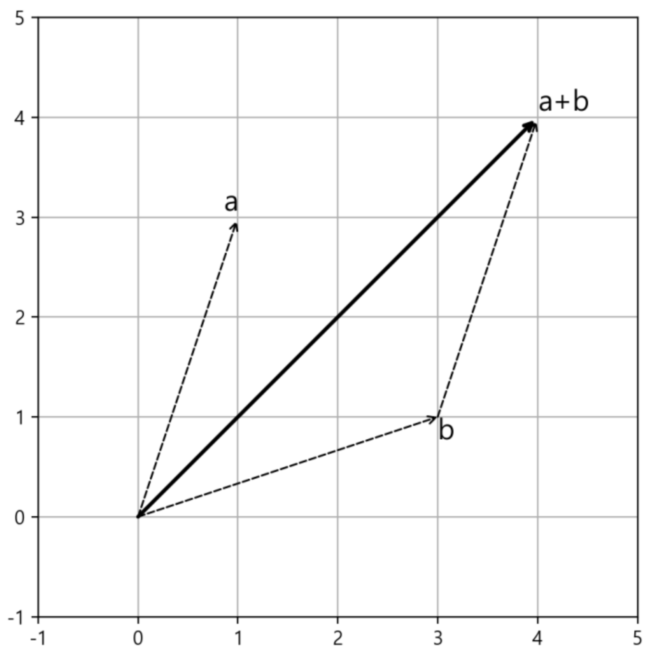
**[벡터의 기하학적인 의미]**

- $\begin{pmatrix} 3 \\ 2 \end{pmatrix}$ 벡터는 좌표 상의 1개의 점으로 표현되거나, 오른쪽 그림과 같이 $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$ 점을 $\begin{pmatrix} 3 \\ 2 \end{pmatrix}$로 옮기는 위치이동(displacement)을 표현하기도 한다.



**[벡터 덧셈]**

- 대응되는 각 원소의 합으로 정의
- 2개의 변으로 구성된 평행사변형의 대각선 벡터




$$\begin{pmatrix}1\\3\end{pmatrix} + \begin{pmatrix}3\\1\end{pmatrix} = \begin{pmatrix}4\\4\end{pmatrix}$$

## [벡터 덧셈]

- 대응되는 각 원소의 합으로 정의
- 2개의 변으로 구성된 평행사변형의 대각선 벡터

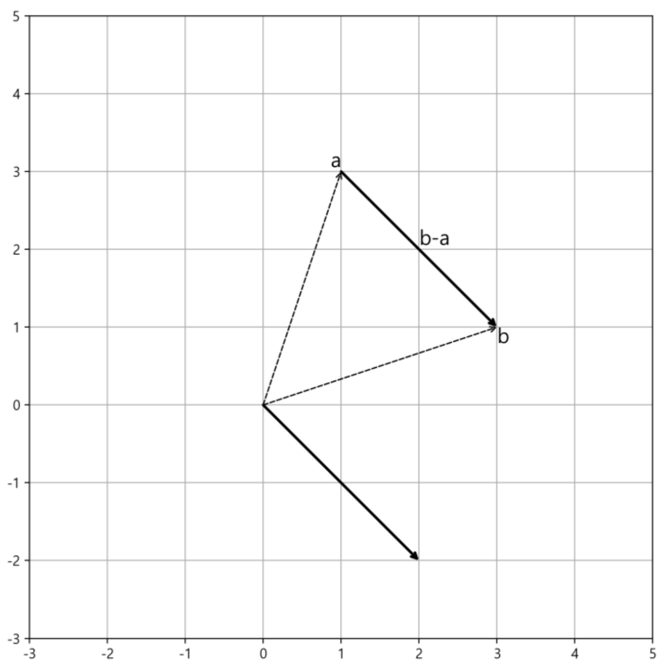
In [55]:
```
# 벡터의 덧셈
a = np.array([1, 3])
b = np.array([3, 1])

print(a+b)

# 교환 법칙
print('교환:', a + b == b + a)

# 결합 법칙
d = np.array([1, 2])
print('결합', (a + b) + d == a + (b + d))
```

```
[4 4]
교환: [ True  True]
결합 [ True  True]
```

# 벡터와 행렬

## [벡터 뺄셈]

- 대응되는 각 원소의 차(difference)로 정의
- 빼는 벡터를 시작점으로 연결한 벡터




$$\begin{pmatrix}3\\1\end{pmatrix} - \begin{pmatrix}1\\3\end{pmatrix} = \begin{pmatrix}2\\-2\end{pmatrix}$$

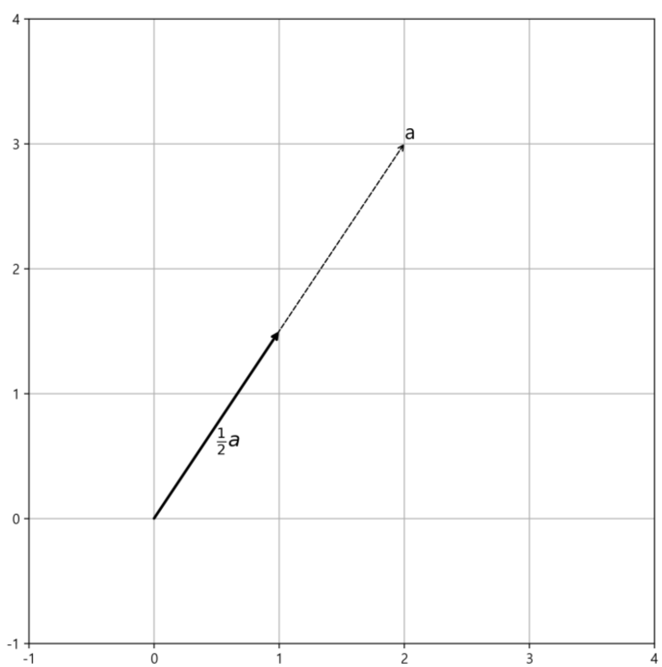
**[벡터 뺄셈]**

- 대응되는 각 원소의 차(difference)로 정의
- 빼는 벡터를 시작점으로 연결한 벡터

```
In [60]: # 벡터의 뺄셈
         a = np.array([3, 1])
         b = np.array([1, 3])

         print(b-a)

         [-2  2]
```

## [스칼라 곱]

- 벡터에 대한 실수 배(스칼라 곱: scalar multiplication)는 그 벡터의 모든 원소를 실수 만큼 곱한 것




$$\frac{1}{2}\begin{pmatrix}2\\3\end{pmatrix}=\begin{pmatrix}1\\1.5\end{pmatrix}$$

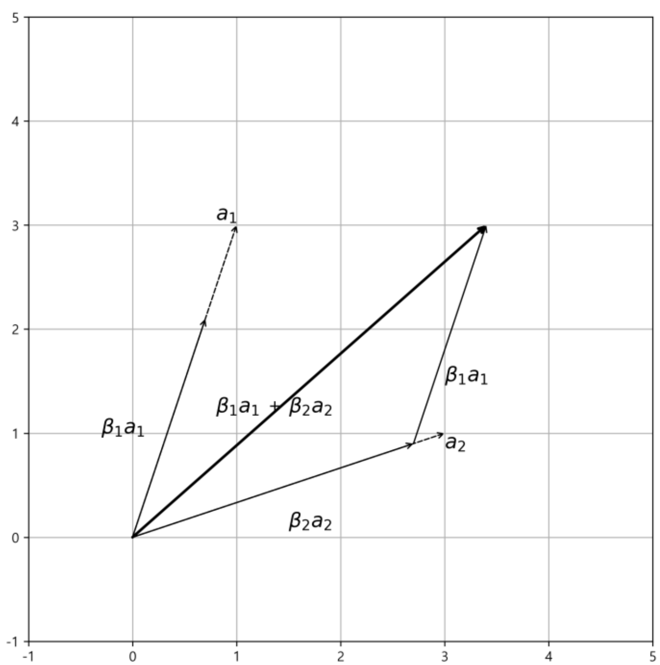
## [스칼라 곱]

- 벡터에 대한 실수 배(스칼라 곱: scalar multiplication)는 그 벡터의 모든 원소를 실수 만큼 곱한 것

```
In [61]: # 스칼라-벡터 곱
         alpha = 1/2
         a = np.array([2, 3])
         print(alpha * a)

         # 교환 법칙
         alpha = 0.5
         a = np.array([2, 3])
         alpha * a == a * alpha
         # array([ True,  True])

         # 분배 법칙
         beta = 0.7
         (alpha + beta) * a == alpha * a +beta * a
         # array([ True,  True])

         [1.  1.5]

Out[61]: array([ True,  True])
```

## [선형 결합]

- 벡터 $a_1, \dots, a_k$에 대한 선형결합(linear combination)
- $\beta_1 a_1 + \cdots + \beta_k a_k$




$$0.7 \begin{bmatrix} 1 \\ 3 \end{bmatrix} + 0.9 \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

## [선형 결합]

- 벡터 $a_1, \ldots, a_k$에 대한 선형결합(linear combination)
- $\beta_1 a_1 + \cdots + \beta_k a_k$

In [64]:
```
# 선형 결합
beta = np.array([0.7, 0.9])
a = np.array([1, 3])
b = np.array([3, 1])
a_shrink = beta[0] * a
b_shrink = beta[1] * b

d = a_shrink + b_shrink
print(d)
```

```
[3.4 3. ]
```

## [내적]

- 내적(inner product) 또는 점적(dot product)
- $a^T b = \sum_i a_i b_i$

$$\begin{pmatrix} -1 \\ 2 \\ 3 \end{pmatrix}^T \begin{pmatrix} 1 \\ -2 \\ 4 \end{pmatrix} = sum \begin{pmatrix} -1 \\ -4 \\ 12 \end{pmatrix} = 7$$

## [내적]

- 내적(inner product) 또는 점적(dot product)
- $a^T b = \sum_i a_i b_i$

```
In [68]: # 내적 계산
         a = np.array([-1, 2, 3])
         b = np.array([1, -2, 4])

         print(np.sum(a*b))

         np.inner(a, b) == np.dot(a, b)

         7

Out[68]: True
```

## [내적]

- 내적의 성질

1) $a^T b = b^T a$

2) $(\alpha a)^T b = \alpha(a^T b)$

3) $(a + b)^T c = a^T c + b^T c$

## [내적]

- 내적의 성질

$$1)\ a^T b = b^T a$$

$$2)\ (\alpha a)^T b = \alpha(a^T b)$$

$$3)\ (a+b)^T c = a^T c + b^T c$$

In [72]:
```
# 내적의 성질
a = np.array([-1, 2, 3])
b = np.array([1, -2, 4])

# 교환 법칙
print(np.dot(a, b) == np.dot(b, a))

# 결합 법칙
alpha= 0.5
print(np.dot(alpha * a, b) == alpha * np.dot(a, b))

# 분배 법칙
c = np.array([1, 2, 3])
np.dot(a+b, c) == np.dot(a,c) + np.dot(b, c)
```

```
True
True
```

Out[72]:
```
True
```

## [내적]

- 내적의 활용

1) $1^T a = a_1 + \cdots + a_n = \sum_{i=1}^{n} a_i$

2) $\frac{1}{n}\ 1^T\ a = \frac{1}{n}(a_1 + \cdots + a_n) = \bar{a}$

3) $a^T a = a_1^2 + \cdots + a_n^2$

# 벡터와 행렬

## [내적]

- 내적의 활용

```
In [76]: # 내적의 활용

         a = np.array([1, 2, 3])
         size = len(a)
         ones = np.ones(shape=(size,))

         # 합의 표현
         print(np.dot(ones, a) == np.sum(a))

         # 평균의 표현
         print(np.dot(ones, a) / len(a) == np.mean(a))

         # 제곱합의 표현
         print(np.dot(a, a) == np.sum(np.square(a)))

         True
         True
         True
```

## [벡터 노름]

- 유클리디안 노름(Euclidean norm)
- $\|a\| = \sqrt{a^T a} = \sqrt{a_1^2 + \cdots + a_n^2}$

$$\left\|\begin{pmatrix}-1\\2\\3\end{pmatrix}\right\| = \sqrt{\begin{pmatrix}-1\\2\\3\end{pmatrix}^T \begin{pmatrix}-1\\2\\3\end{pmatrix}} = \sqrt{sum\begin{pmatrix}1\\4\\9\end{pmatrix}}$$

$$= \sqrt{14} = 3.741657$$

## [벡터 노름]

- 유클리디안 노름(Euclidean norm)
- $\|a\| = \sqrt{a^T a} = \sqrt{a_1^2 + \cdots + a_n^2}$

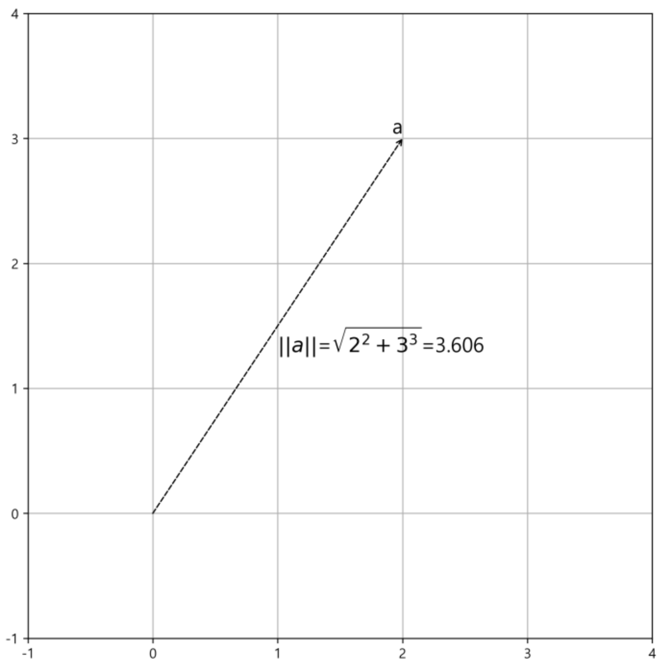
In [79]:
```
# 필요한 패키지
from scipy import linalg as la

# 벡터 노름의 계산
a = np.array([-1, 2, 3])
print("%.3f" %la.norm(a))

la.norm(a) == np.sqrt(np.dot(a, a))
```

```
3.742
```

Out[79]:
```
True
```

## [벡터 노름]

- 유클리디안 노름(Euclidean norm)의 의미
- $\|a\| = \sqrt{a^T a} = \sqrt{a_1^2 + \cdots + a_n^2}$



## [벡터 노름]

- 노름의 성질

$$1)\ \|\beta a\| = |\beta|\|a\|$$

$$2)\ \|a + b\| \le \|a\| + \|b\|$$

$$3)\ \|a\| \ge 0$$

$$4)\ \|a\| = 0 \Rightarrow a = 0$$

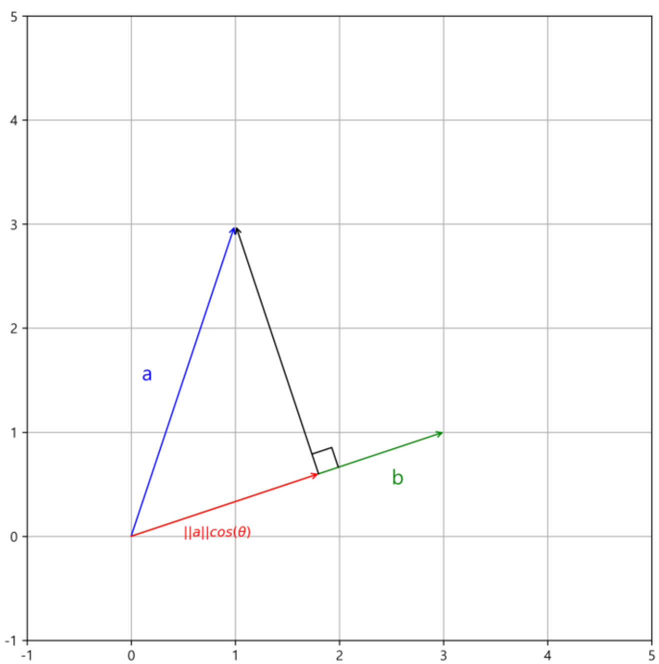
# [벡터 노름]

- 노름의 성질

```
In [84]: # 벡터 노름의 성질
a = np.array([-1, 2, 3])
b = np.array([1, -2, 4])

# 스칼라 곱
beta = 0.5
print(la.norm(beta * a) == np.abs(beta) * la.norm(a))

# 삼각형의 한 변의 길이는 다른 두변의 길이의 합보다 작다
print(la.norm(a + b) <= la.norm(a) + la.norm(b))

# 길이는 0보다 크거나 작다
print(la.norm(a) >= 0 )

True
True
True
```

**[내적과 노름]**

- $a^T b = \|a\| \|b\| \cos(\theta)$
- $\cos(\theta)$=1 의 의미: 2개의 벡터가 일치 할 때




$$\cos(\theta) = \frac{a^T b}{\|a\| \|b\|}$$

## [벡터 선형 종속]

- $n$ 차원 벡터 $a_1, \ldots, a_p$가 선형 종속(linear dependence)
- $\beta_1 a_1 + \cdots, \beta_p a_p = 0$ 가 0 벡터가 아닌 어떤 $\beta$들에 의하여 충족

서로 종속

$$a_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, a_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, a_3 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \qquad \beta_1 = 1, \beta_2 = 1, \beta_3 = -1$$

서로 독립

$$a_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, a_2 = \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix}, a_3 = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \qquad \beta_1 = 0, \beta_2 = 0, \beta_3 = 0$$

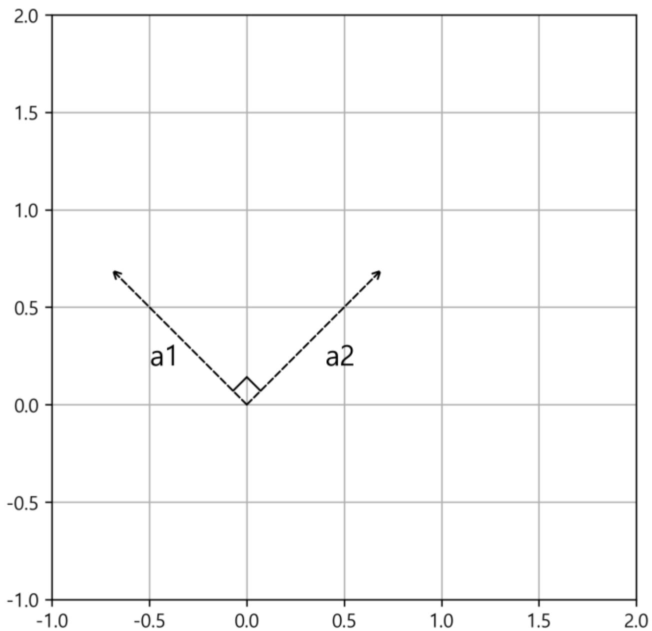
## [기저]

- 기저(base): $n$ 차원에서 $n$개의 선형 독립인 벡터
- 2차원 벡터에서는 선형 독립인 벡터의 수는 최대 2개
- $n$차원 벡터 $a_1, \ldots, a_n$가 기저이면, 임의의 $n$차원 벡터 $x$는

$$x = \beta_1 a_1 + \cdots + \beta_n a_n$$

## [직교정규 벡터]

- $a_i \perp a_j$
- $a_i^T a_j = 0, \|a_i\| = 1, \forall i$




$$a_1 = \frac{1}{\sqrt{2}}\begin{bmatrix}-1\\1\end{bmatrix}, a_2 = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}$$

## [직교정규 벡터]

- $a_i \perp a_j$
- $a_i^T a_j = 0, \|a_i\| = 1, \forall i$

In [24]:
```
# 직교정규 벡터 확인
a1 = 1/np.sqrt(2) * np.array([-1, 1])
a2 = 1/np.sqrt(2) * np.array([1, 1])

print(la.norm(a1))
print(la.norm(a2))
print(np.dot(a1, a2))
```

```
0.9999999999999999
0.9999999999999999
0.0
```

# 행렬

## [행렬 표현]

- 행렬(matrix)은 행(row)과 열(column)로 구분된 직사각형 배열

$$A = \begin{bmatrix} -1.09 & 1 & 0.28 & -1.51 \\ -0.58 & 1.65 & -2.43 & -0.43 \\ 1.27 & -0.87 & -0.68 & -0.09 \\ 1.49 & -0.64 & -0.44 & -0.43 \end{bmatrix}$$

$$B = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$$

## [행렬 표현]

- 행렬(matrix)은 행(row)과 열(column)로 구분된 직사각형 배열

In [25]:
```
# 정의: 2차원 배열
A = np.array(np.random.RandomState(123).normal(size=16)).reshape(4, 4)
print(np.round(A, 2))
```

```
[[-1.09  1.    0.28 -1.51]
 [-0.58  1.65 -2.43 -0.43]
 [ 1.27 -0.87 -0.68 -0.09]
 [ 1.49 -0.64 -0.44 -0.43]]
```

## [행렬 표현]

- 행렬의 크기
- $A_{n \times p}$

$$A_{3\times 4} = \begin{bmatrix} 3 & 3 & 7 & 2 \\ 4 & 11 & 10 & 7 \\ 2 & 1 & 2 & 10 \end{bmatrix}$$

```
In [26]: # 행렬의 크기
A = np.random.RandomState(123).randint(1, 12, size=12).reshape(3, 4)
print(A)
print(A.shape)

[[ 3  3  7  2]
 [ 4 11 10  7]
 [ 2  1  2 10]]
(3, 4)
```

## [행렬 표현]

- 행렬의 표기법

$$\mathrm{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1p} \\ a_{21} & a_{22} & \dots & a_{2p} \\ . & . & \cdots & . \\ a_{n1} & a_{n2} & \dots & a_{np} \end{bmatrix} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1p} \\ a_{21} & a_{22} & \dots & a_{2p} \\ . & . & \cdots & . \\ a_{n1} & a_{n2} & \dots & a_{np} \end{pmatrix} = \left(a_{ij}\right) \in \mathbb{R}^{n\times p}$$

원소(element, entry, coefficient)는 $a_{ij}, a_{i,j}, A[i,j], A_{i,j}, A_{ij}$

벡터와 행렬

## [행렬 표현]

- 행렬의 표기법

$$\mathrm{A} = \begin{bmatrix} 3 & 3 & 7 & 2 \\ 4 & 11 & 10 & 7 \\ 2 & 1 & 2 & 10 \end{bmatrix}$$

$$a_{11} = 3, a_{24} = 7$$

## [행렬 표현]

- 행렬의 표기법

```
In [28]: # 행렬의 표기법
         A = np.random.RandomState(123).randint(1, 12, size=12).reshape(3, 4)
         print(A)
         print('[1, 1] 원소', A[0, 0])

[[ 3  3  7  2]
 [ 4 11 10  7]
 [ 2  1  2 10]]
[1, 1] 원소 3
```

## [행렬 표현]

- 행렬 연산: 덧셈
- $\mathrm{A} = \left(a_{ij}\right), \mathrm{B} = \left(b_{ij}\right)$
- $\mathrm{A} + \mathrm{B} = (a_{ij} + b_{ij})$
- 대응하는 위치의 원소의 값을 더한 것

$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} + \begin{pmatrix} 3 & 5 & 7 \\ 4 & 6 & 8 \end{pmatrix} = \begin{pmatrix} 4 & 8 & 12 \\ 6 & 10 & 14 \end{pmatrix}$$

## [행렬 표현]

- 행렬 연산: 덧셈
- $A + B = (a_{ij} + b_{ij})$
- 대응하는 위치의 원소의 값을 더한 것

In [35]:
```python
# 행렬 덧셈
A = np.arange(1, 7).reshape(2, 3)
B = np.arange(3, 9).reshape(2, 3)

print(A)
print(B)
print(A+B)
```

```
[[1 2 3]
 [4 5 6]]
[[3 4 5]
 [6 7 8]]
[[ 4  6  8]
 [10 12 14]]
```

## [행렬 표현]

- 행렬 연산: 덧셈
- $A + B = B + A$

In [36]:
```
# 행렬 덧셈의 성질
# 교환 법칙
A + B == B + A
```

Out[36]:
```
array([[ True,  True,  True],
       [ True,  True,  True]])
```

벡터와 행렬

**[행렬 표현]**

- 실수와 행렬 곱
- $\mathrm{A} = \left(a_{ij}\right), c \in \mathbb{R}$
- $c\mathrm{A} = (ca_{ij})$
- $c\mathrm{A} = \mathrm{A}c$

$$0.1 \times \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \end{bmatrix}$$

## [행렬 표현]

- 실수와 행렬 곱
- $\mathrm{A} = \left(a_{ij}\right), c \in \mathbb{R}, \ \ c\mathrm{A} = (ca_{ij})$
- $c\mathrm{A} = \mathrm{A}c$

```
In [38]: # 스칼라와 행렬 곱: *
         c = 0.1
         print(c * A)

         print(c*A == A*c)

[[0.1 0.2 0.3]
 [0.4 0.5 0.6]]
[[ True  True  True]
 [ True  True  True]]
```

## [행렬 표현]

- 행렬의 전치
- $\mathrm{A} = (a_{ij})$
- $\mathrm{A}^T = (a_{ji})$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

# [행렬 표현]

- 행렬의 전치
- $\mathrm{A} = (a_{ij})$
- $\mathrm{A}^T = (a_{ji})$

```
In [42]: # 행렬의 전치
         A = np.arange(1, 7).reshape(3, 2)
         print(A)

         A_transpose = A.T
         print(A_transpose)

[[1 2]
 [3 4]
 [5 6]]
[[1 3 5]
 [2 4 6]]
```

## [행렬 표현]

- 행렬의 전치 성질
- $A = (a_{ij})$
- $A^T = (a_{ji})$

*1)* $(A + B)^T = A^T + B^T$
*2)* $(cA)^T = cA^T$

In [54]:
```
# 행렬 전치의 성질
A = np.arange(1, 7).reshape(2, 3)
B = np.arange(3, 9).reshape(2, 3)

print(A)
print(B)

print((A + B).T)
print(A.T + B.T)

c = 0.1
print((c*A).T == c*A.T)
```

```
[[1 2 3]
 [4 5 6]]
[[3 4 5]
 [6 7 8]]
[[ 4 10]
 [ 6 12]
 [ 8 14]]
[[ 4 10]
 [ 6 12]
 [ 8 14]]
[[ True  True]
 [ True  True]
 [ True  True]]
```

## [행렬 표현]

- 행렬 곱
- $\mathrm{A} = \left(a_{ij}\right), i = 1, \dots, n, j = 1, \dots, p,\ \ \mathrm{B} = (b_{kl}), k = 1, \dots, p, l = 1, \dots, m$
- $\mathrm{A}_{n\times p}, \mathrm{B}_{p\times m}$: 앞 행렬의 컬럼 개수와 뒷 행렬의 행의 개수는 반드시 일치
- $\mathrm{AB} = (c_{il}) = \left(\sum_{r=1}^{p} a_{ir} b_{rl}\right)$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{2\times 3} \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}_{3\times 2} = \begin{bmatrix} 22 & 28 \\ \mathbf{49} & 64 \end{bmatrix}_{2\times 2}$$

$$c_{2,1} = \mathrm{A}_{2,:}^{\mathrm{T}} \mathrm{B}_{:,1} = 4 \times 1 + 5 \times 3 + 6 \times 5 = 49$$

## [행렬 표현]

- 행렬 곱
- $\mathrm{AB} = (c_{il}) = \left(\sum_{r=1}^{p} a_{ir} b_{rl}\right)$

In [56]:
```
# 행렬 곱
A = np.arange(1, 7).reshape(2, 3)
B = np.arange(1, 7).reshape(3, 2)

print(A)
print(B)

print(A.shape)
print(B.shape)
```

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

In [57]:
```
# 행렬곱 연산자: @
np.matmul(A, B) == A @ B
print(A @ B)

# 행렬곱 계산 방식
C = A @ B
print(C[1, 0] == np.dot(A[1,:], B[:,0]))
# True
```

```
[[22 28]
 [49 64]]
True
```

**[행렬 표현]**

- 행렬 곱의 성질
- $AB \neq BA$

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix} \neq \begin{pmatrix} 23 & 34 \\ 31 & 46 \end{pmatrix} = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

**[행렬 표현]**

- 행렬 곱의 성질
- $AB \neq BA$

In [58]:
```python
# 행렬곱의 성질
A = np.arange(1, 5).reshape(2, 2)
B = np.arange(5, 9).reshape(2, 2)

print(A)
# [[1 2]
#  [3 4]]
print(B)
# [[5 6]
#  [7 8]]

A @ B == B @ A
# array([[False, False],
#        [False, False]])
```

```
[[1 2]
 [3 4]]
[[5 6]
 [7 8]]
```

Out[58]:
```
array([[False, False],
       [False, False]])
```

## [행렬 표현]

- 부분 행렬(submatrix)

- $A_{p:q,r:s} = \begin{pmatrix} A_{p,r} & \cdots & A_{p,s} \\ . & \cdots & . \\ A_{q,r} & . & A_{q,s} \end{pmatrix}$

$$\mathrm{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$$

$$A_{1:2,2:3} = \begin{bmatrix} 2 & 3 \\ 5 & 6 \end{bmatrix}$$

$$A_{:,2:3} = \begin{bmatrix} 2 & 3 \\ 5 & 6 \\ 8 & 9 \\ 11 & 12 \end{bmatrix}$$

$$A_{(1,3),(2,3)} = \begin{bmatrix} 2 & 3 \\ 8 & 9 \end{bmatrix}$$

## [행렬 표현]

- 부분 행렬(submatrix)

- $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$

$$A_{1:2,2:3} = \begin{bmatrix} 2 & 3 \\ 5 & 6 \end{bmatrix}$$

$$A_{:,2:3} = \begin{bmatrix} 2 & 3 \\ 5 & 6 \\ 8 & 9 \\ 11 & 12 \end{bmatrix}$$

$$A_{(1,3),(2,3)} = \begin{bmatrix} 2 & 3 \\ 8 & 9 \end{bmatrix}$$

In [59]:
```python
# 부분 행렬
A = np.arange(1, 13).reshape(4, 3)
print(A)

# 유형 1
print(A[0:2, 1:3])

# 유형 2
print(A[:, 1:3])

# 유형 3
print(A[[0, 2], :][:, [1, 2]])
```

## [행렬 표현]

- 특별한 행렬: 영 행렬

```
In [61]: # 영 행렬
         size = 3
         np.zeros(shape=(size, size))

Out[61]: array([[0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.]])
```

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

## [행렬 표현]

- 특별한 행렬: 항등 행렬

```
In [62]: # 항등 행렬
         np.identity(n=size)

Out[62]: array([[1., 0., 0.],
                [0., 1., 0.],
                [0., 0., 1.]])
```

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

## [행렬 표현]

- 특별한 행렬: 대각 행렬(diagonal matrix)
- 대각선 값을 제외한 모든 원소의 값이 0인 행렬

```
In [64]: # 대각 행렬
         np.diag([1, 2, 3])

Out[64]: array([[1, 0, 0],
                [0, 2, 0],
                [0, 0, 3]])
```

$$D_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

## [행렬 표현]

- 특별한 행렬: 삼각 행렬(triangular matrix)
- 대각선을 기준으로 하여 한 쪽이 모든 0인 행렬

```
In [65]: # 상삼각 행렬
         np.triu([[1,2,3],[1, 2, 7],[7,8,5]], k=0)

Out[65]: array([[1, 2, 3],
                [0, 2, 7],
                [0, 0, 5]])
```

상삼각행렬 예시: $\begin{bmatrix} 1 & 2 & 3 \\ 0 & 2 & 7 \\ 0 & 0 & 5 \end{bmatrix}$

## [행렬 표현]

- 특별한 행렬: 직교 행렬(orthogonal matrix)
- $A^T A = AA^T = I$

```
In [68]: # 직교 행렬(orthogonal matrix)
         A = np.array([[1,0], [0, -1]])
         print(A)
         print(A.T @ A)
         print(A @ A.T)

[[ 1  0]
 [ 0 -1]]
[[1 0]
 [0 1]]
[[1 0]
 [0 1]]
```

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$