순천향대학교
SOON CHUN HYANG UNIVERSITY

2024년 2학기

# 선형대수

1장

# Chapter 01

- SECTION 01-1 NumPy로 벡터 생성 및 시각화하기
- SECTION 01-2 벡터 연산
- SECTION 01-3 벡터 크기와 단위벡터
- SECTION 01-4 벡터-내적
- SECTION 01-5 그 외 벡터 곱셈
- SECTION 01-6 직교벡터 분해
- SECTION 01-7 정리

SCH

# SECTION 01-1 NumPy로 벡터 생성 및 시각화하기(1)

- 벡터(vector): 수를 순서대로 나열한 것
- 벡터의 특징
  - 차원(Dimensionality): 벡터가 가진 원소의 수
    - $\mathbf{R}^N$으로 나타내는데, 여기서 **R**은 실수(Real number)를 **N**은 차원
    - 예를 들어 두 개의 원소가 있는 벡터는 $\mathbf{R}^2$에 속함
    - **R**문자는 $\mathbf{R}_2$이나 **R**2, **R**^2로 쓸 수도 있음
  - 방향(orientation): 벡터가 열 방향(높이 세워진)인지 행 방향(길고 평평하게 누운)인지를 나타냄
    - x는 4차원 열벡터, y는 2차원 열벡터, z는 4차원 행벡터
    - 또는 $x \in \mathbf{R}^4$ 와 같이 쓸 수도 있음

$$x = \begin{bmatrix} 1 \\ 4 \\ 5 \\ 6 \end{bmatrix}, \quad y = \begin{bmatrix} .3 \\ -7 \end{bmatrix}, \quad z = \begin{bmatrix} 1 & 4 & 5 & 6 \end{bmatrix}$$

# SECTION 01-1 NumPy로 벡터 생성 및 시각화하기(2)

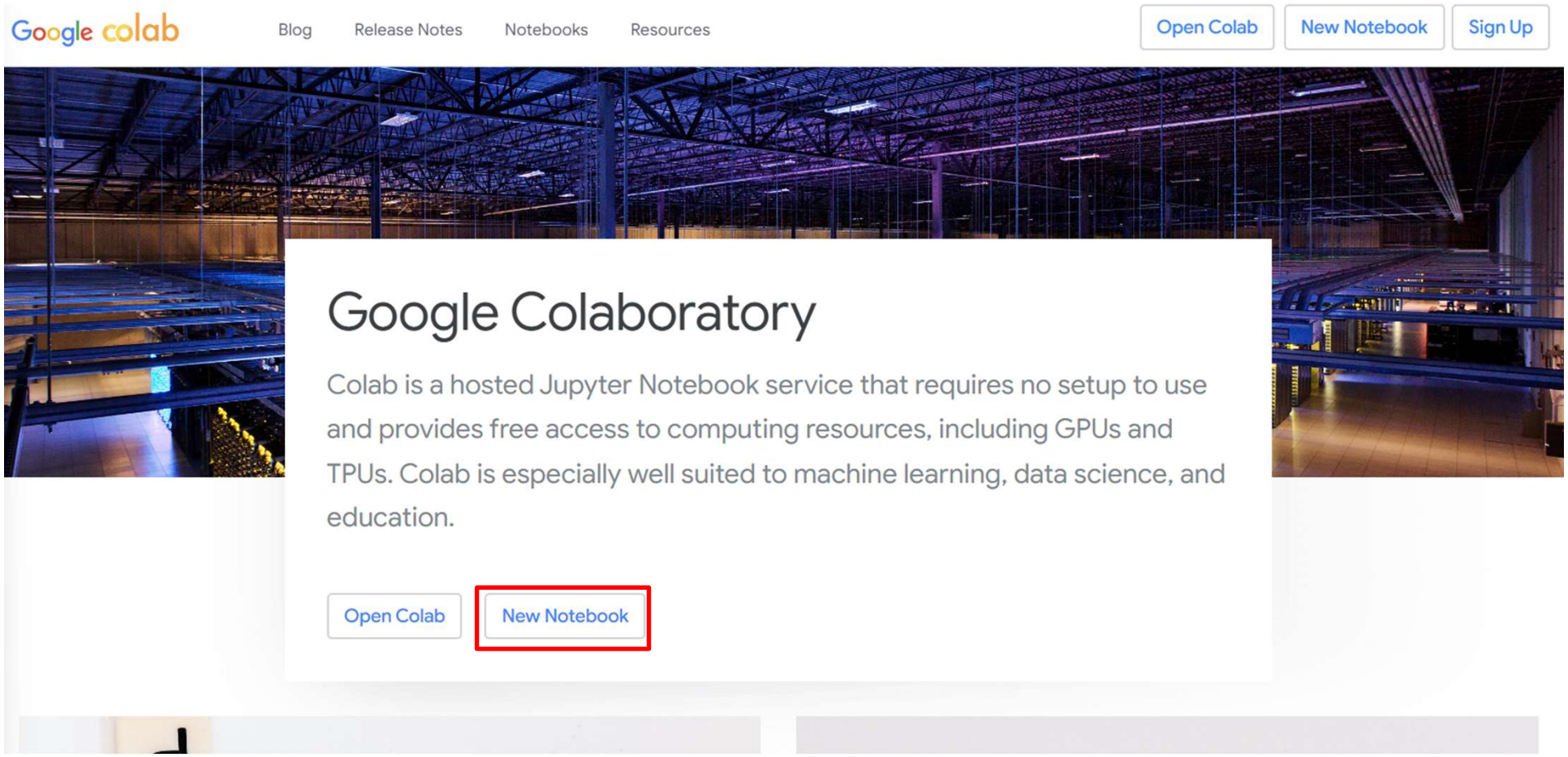
- 파이썬에서 벡터 또는 행렬의 차원
  - 수 객체를 출력하는 데 사용되는 기하학적 차원의 수
  - 모든 벡터는 벡터가 가진 원소의 수(수학적 차원)에 상관없이 NumPy에서 '2차원 배열'로 간주
  - 특정 방향이 없는 수 나열은 원소 수에 상관없이 파이썬에서 1차원 배열
    - 이 배열은 행으로 출력되지만 나중에 나오는 행벡터와 다름
  - 수학적 차원, 즉 벡터의 원소 수는 파이썬에서 벡터의 길이(length) 또는 모양(shape)이라고 함

# SECTION 01-1 NumPy로 벡터 생성 및 시각화하기(2)

- 벡터의 표기
  - 일반적으로 벡터 **v**는 진한 로마자 소문자 **v**로 나타냄
    - 또는 이탤릭체($v$)를 사용하거나 위쪽에 화살표를 붙임($\vec{v}$)
- 선형대수학에서 보통 벡터에 아무런 표시가 없다면 열 방향이라고 가정
  - 행벡터는 $W^T$로 표시
  - T는 전치 연산(transpose operation)을 나타냄

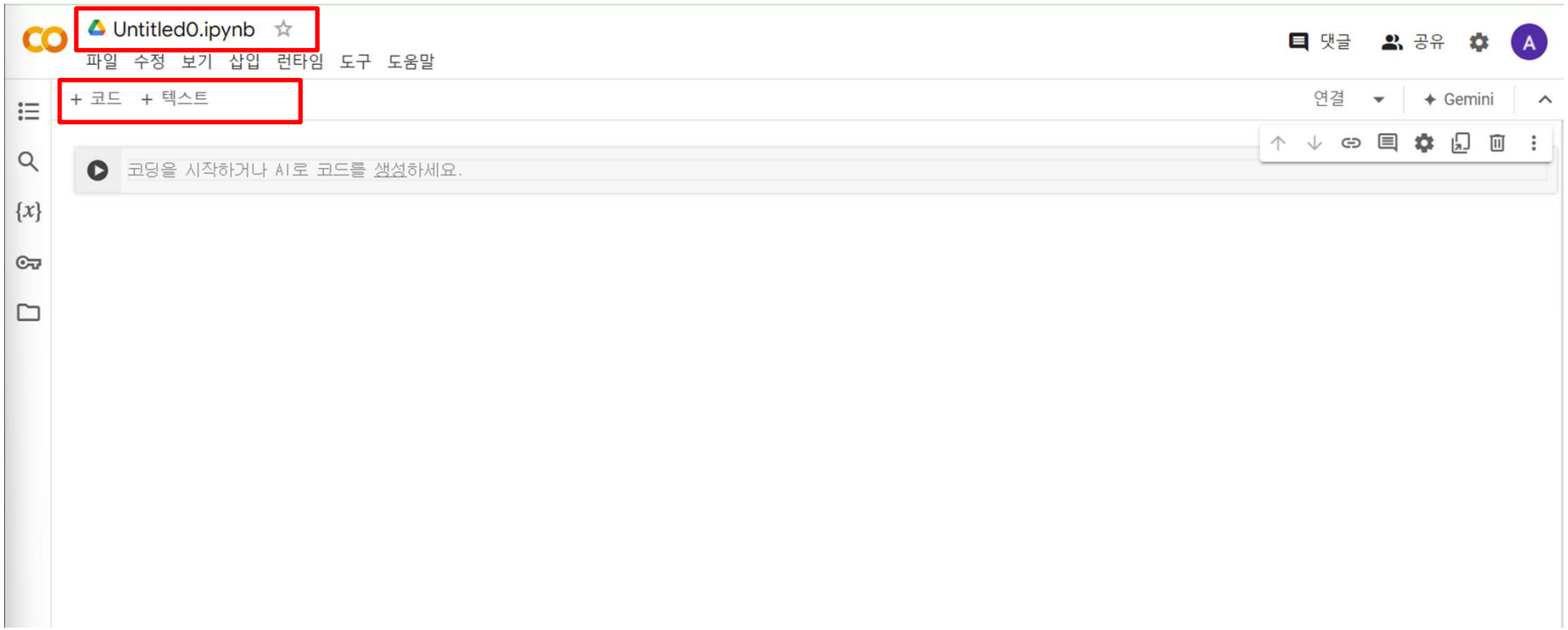
# Colab

- https://colab.google/



# SECTION 01-1 NumPy로 벡터 생성 및 시각화하기(3)

- 새노트 만들기
  - 파일명 작성, ipynb 확장자는 Jupyter Notebook의 파일
  - 코드 혹은 텍스트 창을 추가



# SECTION 01-1 NumPy로 벡터 생성 및 시각화하기(3)

## ■ 파이썬에서 벡터를 표현하는 데이터 타입

- 리스트 타입: 벡터를 표현하는 가장 간단한 방법이지만 선형대수학 응용 분야에서는 잘 사용하지 않음
  - 많은 선형대수학 연산은 파이썬 리스트에 대해 잘 동작하지 않음

```
asList = [1,2,3]
asArray = np.array([1,2,3]) # 1차원 배열
rowVec = np.array([ [1,2,3] ]) # 행
colVec = np.array([ [1],[2],[3] ]) # 열
```

- asArray 변수는 방향이 없는 배열로, 행이나 열벡터가 아니라 NumPy의 숫자 1차원 리스트
- NumPy의 방향은 대괄호로 지정
  - 가장 바깥쪽 대괄호는 모든 숫자를 하나의 객체로 묶음
  - 추가적인 내부 괄호 집합은 행을 나타냄
- 행벡터(변수 rowVec)는 하나의 행이 모든 숫자를 가지지만, 열벡터(변수 colVec)는 하나의 숫자를 가진 행이 여러 개

```
print(f'asList: {np.shape(asList)}')
print(f'asArray: {asArray.shape}')
print(f'rowVec: {rowVec.shape}')
print(f'colVec: {colVec.shape}')
```

⇨

```
asList:  (3,)
asArray: (3,)
rowVec:  (1, 3)
colVec:  (3, 1)
```

- 1차원 배열인 asArray는 모양이 (3)이지만 방향이 부여된 벡터는 2차원 배열이며 방향에 따라 모양이 (1,3) 또는 (3,1)
- 차수는 항상 (행, 열)로 표현

## 연습문제 01

1) 모든 값이 0인 2행 3열인 행렬을 출력하세요.

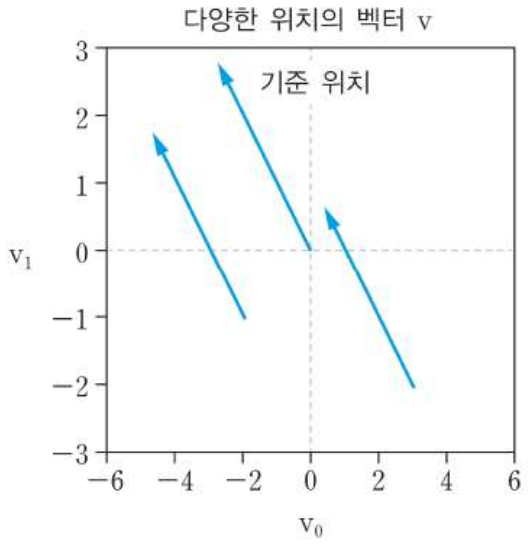
```
array([[0, 0, 0],
       [0, 0, 0]])
```

2) 모든 값이 0인 4행 3열인 행렬을 출력하세요.

```
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
```

# SECTION 01-1 NumPy로 벡터 생성 및 시각화하기(4)

- 순서대로 나열된 수 목록(ordered list of numbers)
  - 벡터의 대수학적 해석 - 순서대로 나열된 수 목록
  - 기하학적 해석 - 특정 길이(또는 크기(magnitude))와 방향(또는 각도(angle): 양의 x축을 기준으로 계산됨)을 가진 직선
- 기준 위치(standard position)
  - 모든 화살표는 동일한 벡터, 기준 위치의 벡터는 꼬리가 원점에 있고 머리는 기하학적 좌표를 가리킴



# SECTION 01-2 벡터 연산(1)

## 1.2.1 두 벡터의 덧셈

- 벡터 합은 동일한 차원을 갖는 벡터끼리만 가능 – 뺄셈도 마찬가지임

두 벡터의 덧셈 $\begin{bmatrix}4\\5\\6\end{bmatrix}+\begin{bmatrix}10\\20\\30\end{bmatrix}=\begin{bmatrix}14\\25\\36\end{bmatrix}$

두 벡터의 뺄셈 $\begin{bmatrix}4\\5\\6\end{bmatrix}-\begin{bmatrix}10\\20\\30\end{bmatrix}=\begin{bmatrix}-6\\-15\\-24\end{bmatrix}$

- 파이썬에서 두 벡터의 덧셈

```
v = np.array([4,5,6])
w = np.array([10,20,30])
u = np.array([0,3,6,9])
vPlusW = v + w
uPlusW = u + w # 오류! 차원 불일치!
```

# SECTION 01-2 벡터 연산(2)

## 덧셈에서 벡터 방향

- 열벡터에 행벡터를 더할 수 있나요?

$$\begin{bmatrix}4\\5\\6\end{bmatrix} + [10\ \ 20\ \ 30] = ?$$

- 파이썬의 연산 동작

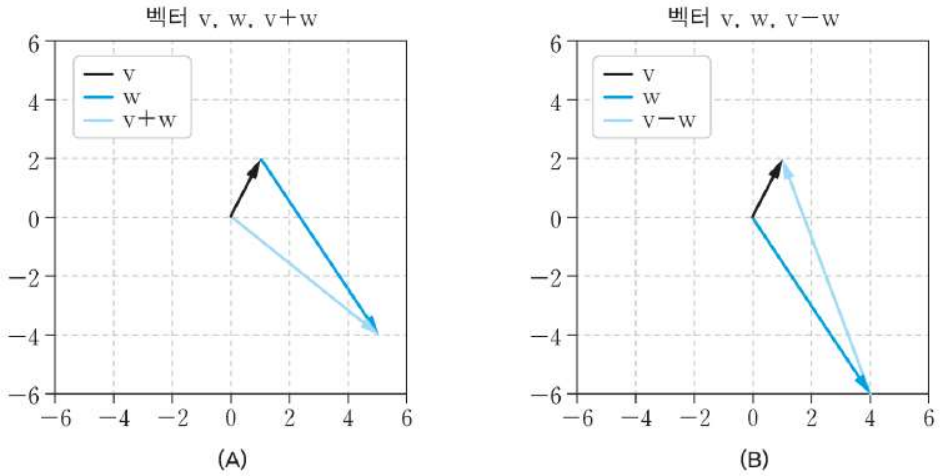
```
v = np.array([[4,5,6]]) # 행벡터
w = np.array([[10,20,30]]).T # 열벡터
v + w
>> array([[14, 15, 16],
          [24, 25, 26],
          [34, 35, 36]])
```

- 이전에 정의한 벡터 덧셈의 결과와 다름
- 파이썬의 브로드캐스팅(broadcasting) 연산
- 두 벡터의 차원과 방향이 같을 때만 더할 수 있음

# SECTION 01-2 벡터 연산(3)

## 1.2.2 벡터의 덧셈과 뺄셈의 기하학적 구조

- 두 벡터를 기하학적으로 더할 때
  - 한 벡터의 꼬리와 다른 벡터의 머리를 연결
- 두 벡터를 기하학적으로 뺄 때
  - 두 벡터의 꼬리들을 같은 좌표에 위치시킴(기준 위치에 두 벡터를 두면 쉬움)
  - 뺀 결과의 벡터는 두 번째 벡터의 머리에서 첫 번째 벡터의 머리로 가는 선




두 벡터의 합과 차

# SECTION 01-2 벡터 연산(4)

## 1.2.3 스칼라-벡터 곱셈

- 스칼라(scalar): 벡터나 행렬에 포함된 숫자가 아닌 수 그 자체
  - 스칼라는 일반적으로 $\alpha$(alpha)또는 $\lambda$(lambda)와 같은 그리스어 소문자로 나타냄
  - 예를 들면 스칼라-벡터 곱셈을 $\lambda \boldsymbol{w}$로 나타낼 수 있음

$$\lambda = 4,\ \mathbf{w} = \begin{bmatrix} 9 \\ 4 \\ 1 \end{bmatrix},\ \lambda\mathbf{w} = \begin{bmatrix} 36 \\ 16 \\ 4 \end{bmatrix}$$

- 스칼라-벡터 곱셈의 데이터 타입
  - 코드는 스칼라(변수 s)와 벡터 리스트(변수 a)를 생성하고 a를 NumPy 배열(변수 b)로 변환
  - 파이썬에서 별(*)표 연산은 변수 타입에 따라 다르게 동작하도록 재정의
  - 리스트 반복과 스칼라-벡터 곱셈
    - 만약, s=2.0이라면 a*s는 오류가 발생 - 리스트 반복은 정수로만 수행되기 때문임

```
s = 2
a = [3,4,5] # 리스트
b = np.array(a) # np 배열
print(a*s)
print(b*s)
```

```
>> [ 3, 4, 5, 3, 4, 5 ]
>> [ 6 8 10 ]
```

# SECTION 01-2 벡터 연산(5)

## 1.2.4 스칼라-벡터 덧셈

- 선형대수학에서 벡터와 스칼라는 별도의 수학적 객체이며 결합할 수 없음
- 그러나, 파이썬과 같은 수치 처리 프로그램에서는 벡터에 스칼라를 더할 수 있음
  - 연산은 스칼라-벡터 곱셈과 유사하여, 각 벡터 원소에 스칼라를 더하면 됨

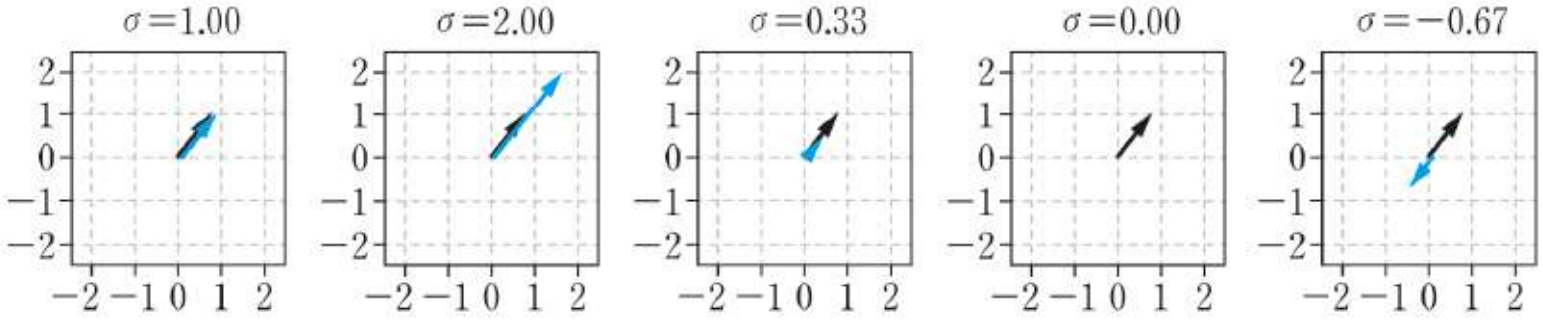
```
s = 2
v = np.array([3,6])
s + v
```

```
>> array([5, 8])
```

# SECTION 01-2 벡터 연산(6)

## 스칼라-벡터 곱셈의 기하학적 구조

- 스칼라는 벡터의 방향을 바꾸지 않고 크기만 조정
- 스칼라-벡터 곱셈의 결과는 스칼라가 1보다 큰지, 0과 1 사이인지, 정확히 0인지, 음수인지에 따라 다름
  - 동일한 벡터(검은 화살표)에 다양한 스칼라 σ(sigma, 구분하기 위해 회색 선을 살짝 이동)를 곱함
  - 그림에서 스칼라가 음수일 때 벡터 방향이 뒤집힘(즉 180도 회전)



# SECTION 01-2 벡터 연산(6)

## 스칼라-벡터 곱셈의 기하학적 구조

- 벡터의 평균(vector average) 계산
  - 벡터 합과 스칼라-벡터 곱셈을 이용
  - N개 벡터의 평균을 구하려면 모두 더하고 스칼라 1/N을 곱

```
b = [3, 4, 5]
np.mean(a)
```

```
4.0
```

```
b = [3, 4, 5]
c = [1, 2, 3]
np.mean(a+c)
```

```
3.0
```

```
b = [3, 4, 5]
c = [1, 2, 3]
(np.mean(b)+np.mean(c))/2
```

```
3.0
```

# SECTION 01-2 벡터 연산(7)

## 1.2.5 전치

- 전치(transpose) 연산: 열벡터를 행벡터로 또는 반대로 변환
  - 각 행렬 원소인 (행, 열) 인덱스를 맞바꾸는 것

$$\mathrm{m}^{\mathrm{T}}_{i,j} = \mathrm{m}_{j,i}$$

  - 두 번 전치하면 벡터는 원래 방향이 됨
    - 즉, $\mathbf{v}^{TT} = \mathbf{v}$

```
c = np.array([[1, 2, 3]]).T
c
```

```
array([[1],
       [2],
       [3]])
```

```
c = np.array([[1, 2, 3]])
c.T.T, c.T
```

```
(array([[1, 2, 3]]),
 array([[1],
        [2],
        [3]]))
```

# SECTION 01-2 벡터 연산(8)

## 1.2.6 파이썬에서 벡터 브로드캐스팅

- 브로드캐스팅 연산은 현대 컴퓨터 기반 선형대수학에서만 존재
  - 브로드캐스팅은 본질적으로 한 벡터를 다른 벡터의 각 원소로 연산을 여러 번 반복하는 것
  - 행렬 합을 브로드캐스팅으로 구현

$$\begin{bmatrix}1 & 1\\ 2 & 2\\ 3 & 3\end{bmatrix} + \begin{bmatrix}10 & 20\end{bmatrix} \Rightarrow$$

```
v = np.array([[1,2,3]]).T # 열벡터
w = np.array([[10,20]]) # 행벡터
v + w # 브로드캐스팅 덧셈
```

```
>> array([[11, 21],
          [12, 22],
          [13, 23]])
```

# SECTION 01-3 벡터 크기와 단위벡터(1)

- 벡터의 크기(기하학적 길이 또는 노름(norm)): 벡터의 꼬리부터 머리까지의 거리
  - 표준 유클리드(Euclidean) 거리 공식(벡터 원소들의 제곱합에 제곱근을 취함, [식 1-7] 참고)으로 구함
  - 벡터 크기는 벡터 양 옆에 이중 수직 막대로 표시($\|\mathbf{v}\|$)
    - 일부 응용에서 제곱 크기($\|\mathbf{v}\|^2$)를 사용하는 경우, 오른편에 있는 제곱근 항을 제거

$$\| \mathbf{v} \| = \sqrt{\sum_{i=1}^{n} v_i^2}$$

```
# 벡터의 크기를 계산하는 코드를 작성해보자!
- np.linalg.norm()함수를 쓰지 말고, 작성해보기!
```

# SECTION 01-3 벡터 크기와 단위벡터(1)

- 벡터의 크기(기하학적 길이 또는 노름(norm)): 벡터의 꼬리부터 머리까지의 거리

# 벡터의 크기를 계산하는 코드를 작성해보자!

- `np.linalg.norm()`함수를 쓰지 말고, 작성해보기!

```
a = np.array([[1,2]])
(a[0][0]**2 + a[0][1]**2)**(1/2), np.linalg.norm(a)
```

```
(2.23606797749979, 2.23606797749979)
```

```
a = np.array([[1,2]])
np.sum(a*a)**(1/2), np.linalg.norm(a)
```

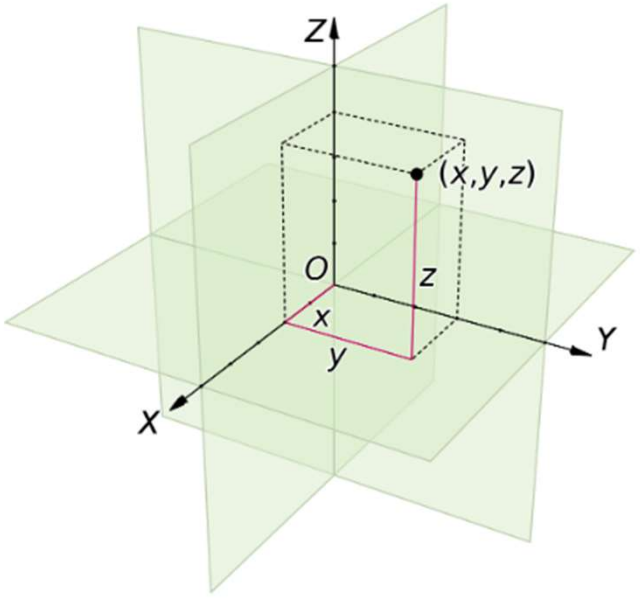
```
(2.23606797749979, 2.23606797749979)
```

https://numpy.org/doc/stable/reference/generated/numpy.sum.html

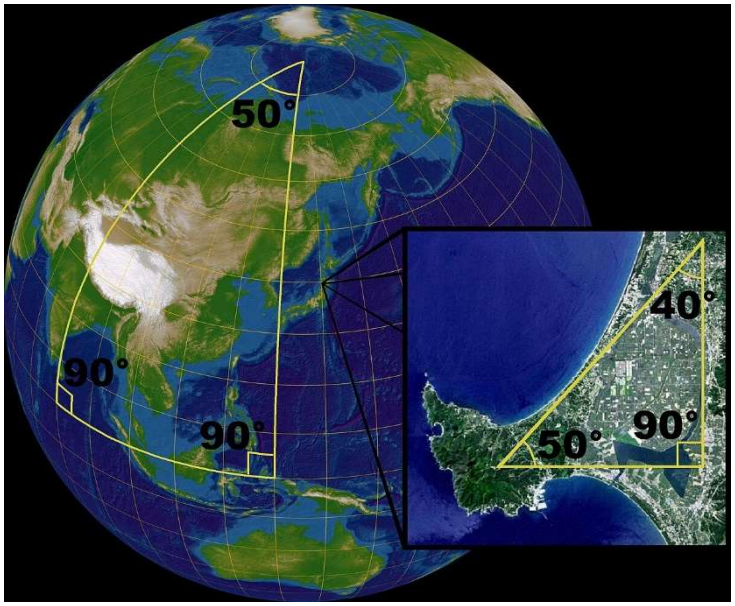
# 유클리드 공간

- 고대 그리스 수학자 유클리드
  - 유클리드 기하학(Euclidean geometry)는 수학자 유클리드가 구축한 수학의 체계
  - 2차원의 평면 기하학과 3차원의 공간 기하학을 다룸

유클리드(Euclid, BC 365년 경 ~ BC 275년 경)




유클리드 기하학




구면 기하학

# SECTION 01-3 벡터 크기와 단위벡터(1)

- 벡터의 크기(기하학적 길이 또는 노름(norm)): 벡터의 꼬리부터 머리까지의 거리
  - 수학과 파이썬의 용어 차이
    - 수학에서 벡터의 차원은 벡터의 원소 수, 길이는 기하학적 거리
    - 파이썬에서는 함수 len() (len은 length의 약자)은 배열의 차원을 반환하고 np.norm()은 기하학적 길이(크기)를 반환
    - 여기 학습에서는 길이 대신 크기(또는 기하학적 길이)라는 용어를 계속 사용

```
v = np.array([1,2,3,7,8,9])
v_dim = len(v) # 수학의 차원성
v_mag = np.linalg.norm(v) # 수학적 크기, 길이, 또는 노름
```

# 연습문제 02

- 벡터의 크기를 계산하는 코드(함수)를 만드세요.
  - np.sqrt(input): input의 제곱근 결과 리턴, input에 루트를 씌운다.

# SECTION 01-3 벡터 크기와 단위벡터(2)

## ■ 단위벡터

- 기하학적인 길이가 1인 벡터
- 응용의 예) 직교 행렬과 회전 행렬, 고유벡터, 특이벡터 등
- 단위벡터는 $\|\mathbf{v}\| = 1$ 로 정의

### ▪ 연관된 단위벡터를 만들기

- 벡터 노름의 역수를 스칼라 곱셈

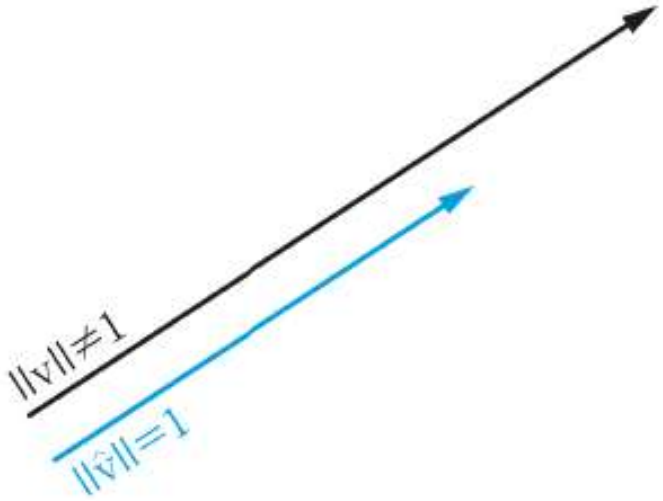
$$\hat{\mathbf{v}} = \frac{1}{\|\mathbf{v}\|}\mathbf{v}$$

```
w = np.array([1,2,3])
s = np.linalg.norm(w)
w = w/s
print(w, s, np.linalg.norm(w))
```

```
[0.26726124 0.53452248 0.80178373] 3.7416573867739413 1.0
```

# SECTION 01-3 벡터 크기와 단위벡터(2)

- 단위벡터
  - 부모 벡터(v)와 같은 방향의 단위벡터($\hat{v}$)를 표시하는 일반적인 규약
    - 단위벡터(강조색 화살표)는 비단위벡터(검은 화살표)를 가공해서 만듭니다. 두 벡터의 각도는 동일지만 크기는 다름



# SECTION 01-4 벡터-내적(1)

- 내적(또는 점곱, 스칼라곱)은 선형대수학에서 가장 중요한 연산
  - 합성곱(convolution), 상관관계(correlation), 푸리에 변환(Fourier transform), 행렬 곱셈, 선형 특징 추출, 신호 필터링 등 많은 연산과 알고리즘의 기본
  - 내적은 하나의 숫자로 두 벡터 사이의 관계를 나타냄
  - 두 벡터 사이의 내적을 표기하는 방법
    - 일반적인 표기법 $\mathbf{a}^{\mathrm{T}}\mathbf{b}$
    - a·b 또는 <a,b>
  - 내적 계산
    - 두 벡터에서 대응되는 원소끼리 곱한 다음 모든 결과를 더함
    - 내적은 동일한 차원의 두 벡터 사이에서만 성립

$$\delta = \sum_{i=1}^{n} a_i b_i$$

$$\begin{aligned}[1\ 2\ 3\ 4] \cdot [5\ 6\ 7\ 8] &= 1\times 5 + 2\times 6 + 3\times 7 + 4\times 8 \\ &= 5 + 12 + 21 + 32 \\ &= 70\end{aligned}$$

# SECTION 01-4 벡터-내적(2)

- 파이썬에서 내적을 구현하는 여러 가지 방법 중 np.dot() 함수

```
v = np.array([1,2,3,4])
w = np.array([5,6,7,8])
np.dot(v,w)
```

- 벡터에 스칼라를 곱하면 내적도 그만큼 커짐

```
s = 10
np.dot(s*v,w)
```

  - v와 w의 내적은 70이고, s*v(수학 표기법으로 $\sigma v^T w$)와 w의 내적은 700
  - 음수 스칼라, 예를 들어 s = -1를 시도하면, 내적 크기는 그대로지만 기호는 반대
  - s = 0이면 내적도 0

# SECTION 01-4 벡터-내적(2)

## 1.4.1 내적의 분배 법칙

- 벡터 합의 내적은 벡터-내적의 합과 같음

$$a^T(b+c) = a^Tb + a^Tc$$

- 파이썬 코드로 구현한 분배 법칙
  - 두 결과인 res1과 res2는 동일(정답은 110)
  - 이는 내적의 분배 법칙이 성립함을 나타냄

```
a = np.array([ 0,1,2 ])
b = np.array([ 3,5,8 ])
c = np.array([ 13,21,34 ])

# 내적 분배 법칙
res1 = np.dot( a, b+c )
res2 = np.dot( a,b ) + np.dot( a,c )
```

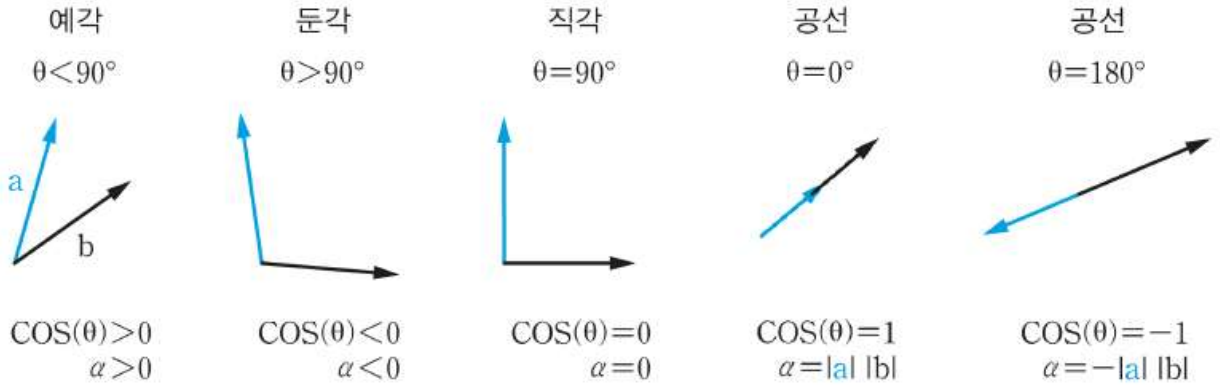
# SECTION 01-4 벡터-내적(3)

## 1.4.2 내적의 기하학적 해석

- 내적의 기하학적 정의
  - 두 벡터의 크기를 곱하고 두 벡터 사이의 각도에서 코사인값만큼 크기를 곱하면 내적이 계산됨. 수학적으로 동일하지만 다르게 표현된 형태
  - 벡터-내적의 기하학적 정의

$$\delta = \sum_{i=1}^{n} a_i b_i \qquad \alpha = \cos(\theta_{\mathbf{v},\mathbf{w}}) \| \mathbf{v} \| \| \mathbf{w} \|$$

- 두 벡터 사이의 각에 따른 내적 부호 다섯 가지 사례
  - 두 벡터의 내적의 부호는 벡터 사이의 기하학적 관계를 나타냄



# SECTION 01-5 그 외 벡터 곱셈(1)

## 1.5.1 아다마르곱

- 아다마르곱(Hardamard product)의 구현
  - 두 벡터의 대응되는 각 원소를 곱합
  - 곱의 결과는 두 벡터와 같은 차원의 벡터

$$\begin{bmatrix} 5 \\ 4 \\ 8 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 0 \\ .5 \\ -1 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ 4 \\ -2 \end{bmatrix}$$

- 파이썬에서 별표는 두 벡터나 행렬에서 원소 사이의 곱을 나타냄

```
a = np.array([5,4,8,2])
b = np.array([1,0,.5])
a * b
```

# SECTION 01-5 그 외 벡터 곱셈(2)

## 1.5.2 외적

- 내적은 열벡터와 행벡터를 이용해 행렬을 생성
  - 내적 행렬의 각 행은 행벡터 스칼라에 대응되는 열벡터 원소를 곱한 것
  - 내적 행렬의 각 열은 열벡터 스칼라에 대응되는 행벡터 원소를 곱한 것

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} [d\ e] = \begin{bmatrix} ad\ ae \\ bd\ be \\ cd\ ce \end{bmatrix}$$

- 외적의 특징
  - 외적은 스칼라 대신 행렬을 생성
  - 외적은 두 벡터를 곱하는 특수한 수학적 기법
  - NumPy를 이용해 각각 열과 행 방향인 두 벡터를 np.outer() 함수에 입력해서 외적을 계산

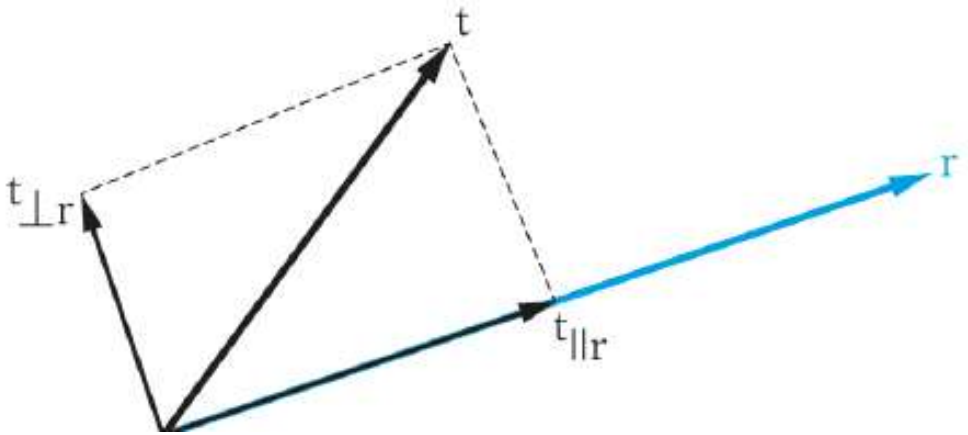
```
v = np.array([[1,2,3,4]])
w = np.array([[1,2,3,4]]).T
np.outer(v,w)
```

```
array([[ 1,  2,  3,  4],
       [ 2,  4,  6,  8],
       [ 3,  6,  9, 12],
       [ 4,  8, 12, 16]])
```

# SECTION 01-6 직교벡터 분해(1)

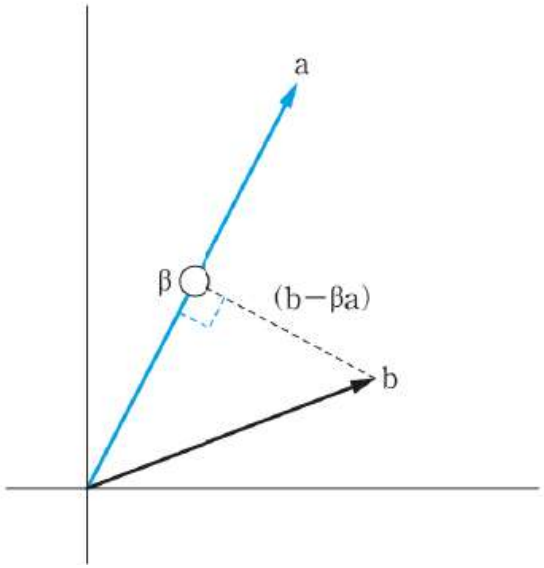
## 분해 개념

- 스칼라 분해
  - 숫자 42.01 = 42 + 0.01
  - 소인수 분해(prime factorization) - 숫자 42를 소수 2, 3, 7의 곱으로 분해
- 벡터 분해
  - 하나의 벡터를 두 개의 벡터로 분해하는데, 하나는 기준 벡터와 직교하고 다른 하나는 기준 벡터와 평행
    - 직교벡터 분해는 통계에서 그람-슈미트 과정(Gram-Schmidt Process)과 QR 분해에 직접적인 연관



# SECTION 01-6 직교벡터 분해(2)

- 벡터 분해 시각화
  - b 의 머리와 가장 가까운 벡터 a 위의 점을 찾으려면 투영 벡터 b-βa 의 길이가 최소가 되는 β를 구하는 공식이 필요함
  - 표준 위치에 두 개의 벡터 **a** 와 **b** 가 존재
    - **a**에서 **b** 의 머리와 최대한 가까운 점을 탐색
      - 최적화 문제로 표현 가능. 즉 투영 거리가 최소가 되도록 벡터 **b** 를 벡터 **a** 에 투영
      - 그 점은 **a** 의 크기를 줄인 즉 β**a**
      - 스칼라 β를 찾으면 됨



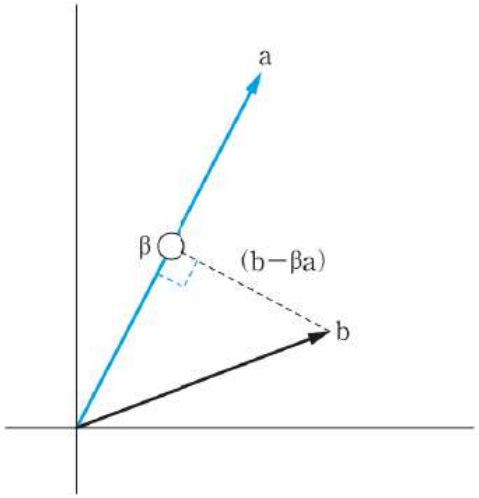
# SECTION 01-6 직교벡터 분해(3)

- 직교 투영법
  - 앞의 **b**-β**a**가 β**a**와 직교한다는 것을 추론 가능
  - 즉 이 벡터들은 수직. 따라서, 둘 사이의 내적이 0이 되어야 함
  - β 구하기

$$\mathbf{a}^{\mathrm{T}}(\mathbf{b}-\beta\mathbf{a})=0$$

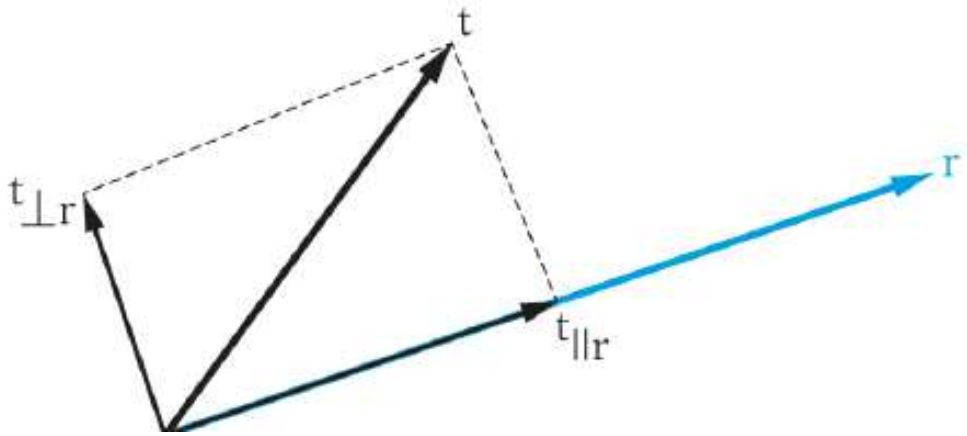
$$\begin{aligned}\mathbf{a}^{\mathrm{T}}\mathbf{b}-\beta\mathbf{a}^{\mathrm{T}}\mathbf{a}&=0\\ \beta\mathbf{a}^{\mathrm{T}}\mathbf{a}&=\mathbf{a}^{\mathrm{T}}\mathbf{b}\\ \beta&=\frac{\mathbf{a}^{\mathrm{T}}\mathbf{b}}{\mathbf{a}^{\mathrm{T}}\mathbf{a}}\end{aligned}$$

- '목표 벡터'와 '기준 벡터'
  - 목적은 목표 벡터를 두 개의 다른 벡터로 분해
    (1) 그 두 벡터의 합은 목표 벡터
    (2) 하나의 벡터는 기준 벡터와 직교하지만 다른 벡터는 기준 벡터와 평행



# SECTION 01-6 직교벡터 분해(4)

- 용어 정리
  - 목표 벡터 **t**, 기준 벡터 **r**
  - 표 벡터로부터 만들어진 두 벡터는 수직 성분 $\mathbf{t}_{\perp r}$과 평행 성분 $\mathbf{t}_{\parallel r}$
  - 벡터 t를 벡터 r과 직교하는 벡터와 평행한 벡터로 분해



# SECTION 01-6 직교벡터 분해(5)

- 평행 성분
  - **r** 의 크기를 조정한 벡터는 **r**과 평행
  - 이전의 직교투영 공식을 적용하면 $t_{||r}$ 을 계산할 수 있음
  - 이전에는 스칼라 값인 $\beta$만 계산했다면, $\beta r$을 계산함

$$\begin{aligned} \mathrm{a}^{\mathrm{T}}\mathrm{b} - \beta\mathrm{a}^{\mathrm{T}}\mathrm{a} &= 0 \\ \beta\mathrm{a}^{\mathrm{T}}\mathrm{a} &= \mathrm{a}^{\mathrm{T}}\mathrm{b} \\ \beta &= \frac{\mathrm{a}^{\mathrm{T}}\mathrm{b}}{\mathrm{a}^{\mathrm{T}}\mathrm{a}} \end{aligned}$$

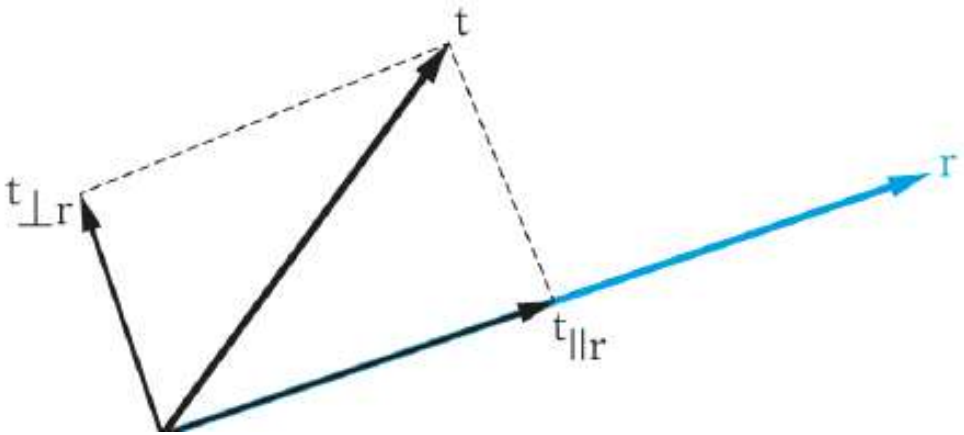
$$t_{||r} = r\frac{t^{\mathrm{T}}r}{r^{\mathrm{T}}r}$$

# SECTION 01-6 직교벡터 분해(6)

- 수직 성분
  - 수직 성분이 정말로 기준 벡터와 직교할까?
  - (증명) 수직 성분과 기준 벡터 사이의 내적이 0인지 계산

$$\mathbf{t} = \mathbf{t}_{\perp \mathbf{r}} + \mathbf{t}_{\parallel \mathbf{r}}$$

$$\mathbf{t}_{\perp \mathbf{r}} = \mathbf{t} - \mathbf{t}_{\parallel \mathbf{r}}$$



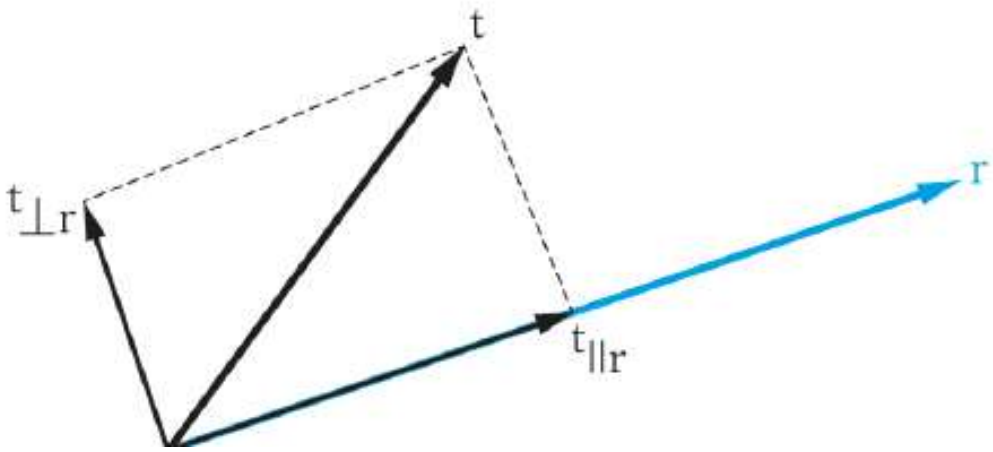
# SECTION 01-6 직교벡터 분해(6)

- 증명
  - 수직 성분이 정말로 기준 벡터와 직교할까?

  -> 수직 성분과 기준 벡터 사이의 내적이 0인지 계산

$$(\mathbf{t}_{\perp \mathbf{r}})^{\mathrm{T}}\mathbf{r}=0$$

$$\left(\mathbf{t}-\mathbf{r}\frac{\mathbf{t}^{\mathrm{T}}\mathbf{r}}{\mathbf{r}^{\mathrm{T}}\mathbf{r}}\right)^{\mathrm{T}}\mathbf{r}=0$$



# 연습문제 03

- 직교분해벡터 계산 코드를 작성하세요.
  - 목표 벡터 **t**와 기준 벡터 **r**에서 평행성분과 수직성분 계산
  - (증명1) 수직성분 + 평행성분 = 목표벡터
  - (증명2) 수직성분과 기준벡터 **r**이 서로 직교하는지 확인