8. 선형대수 기초

주요 내용

- 벡터
- 행렬

개요

앞으로 배울 `numpy.array` 자료형이 제공하는 다양한 기능의 이해에 도움을 주는 내용

8.1. 벡터

- 벡터와 차원
- 리스트와 벡터
- 벡터 항목별 연산
- 벡터 내적과 크기

벡터와 차원

- 벡터: 유한 개의 값으로 구성
- 차원dimension: 항목의 개수
- 벡터 예제:
    - 2차원 평면 공간에서 방향과 크기를 표현하는 2차원 벡터:

      ```
      [x, y]
      ```

    - 사람들의 키, 몸무게, 나이로 이루어진 3차원 벡터:

      ```
      [키, 몸무게, 나이]
      ```

    - 네 번의 시험 점수로 이루어진 4차원 벡터:

      ```
      [1차점수, 2차점수, 3차점수, 4차점수]
      ```

리스트와 벡터

- 벡터를 리스트로 구현 가능
- x축, y축 좌표로 구성된 2차원 벡터

In [1]:

```
# [x좌표, y좌표]

twoDVector1 = [3, 1]
twoDVector2 = [-2, 5]
```

- 키, 몸무게, 나이로 구성된 3차원 벡터

In [2]:

```
# [키, 몸무게, 나이]

height_weight_age1 = [70, 170, 50]
height_weight_age2 = [66, 163, 50]
```

- 1차부터 4차까지의 시험 점수로 구성된 4차원 벡터

In [3]:

```
# [1차점수, 2차점수, 3차점수, 4차점수]

grades1 = [95, 80, 75, 62]
grades2 = [85, 82, 79, 82]
```

벡터 항목별 연산

- 벡터 항목별 사칙연산
- 벡터 스칼라 곱셈
- 항목별 평균 벡터

벡터 항목별 덧셈

동일 차원의 두 벡터의 항목별 덧셈

$$[u_1, \cdots, u_n] + [v_1, \cdots, v_n] = [u_1 + v_1, \cdots, u_n + v_n]$$

In [4]:

```
def addV(u, v):
    assert len(u) == len(v)   # 두 벡터의 길이가 같은 경우만 취급

    return [u_i + v_i for u_i, v_i in zip(u, v)]
```

In [5]:

```
addV([95, 80, 75, 62], [85, 82, 79, 82])
```

Out[5]:

```
[180, 162, 154, 144]
```

벡터 리스트의 합

동일한 차원의 임의의 개수의 벡터를 항목별로 더하는 함수

In [6]:

```
def vector_sum(vectors):
    """
    vectors: 동일한 차원의 벡터들의 리스트
    반환값: 각 항목의 합으로 이루어진 동일한 차원의 벡터
    """

    # 입력값 확인
    assert len(vectors) > 0          # 1개 이상의 벡터가 주어져야 함
    num_elements = len(vectors[0])   # 벡터 개수
    assert all(len(v) == num_elements for v in vectors)   # 모든 벡터의 크기가 같아야 함

    # 동일한 위치의 항목을 모두 더한 값들로 이루어진 벡터 반환
    return [sum(vector[i] for vector in vectors) for i in range(num_elements)]
```

In [7]:

```
vector_sum([[1, 2], [3, 4], [5, 6], [7, 8]])
```

Out[7]:

```
[16, 20]
```

벡터 항목별 뺄셈

동일 차원의 벡터 두 개의 항목별 뺄셈

$$[u_1, \cdots, u_n] - [v_1, \cdots, v_n] = [u_1 - v_1, \cdots, u_n - v_n]$$

In [8]:

```
def subtractV(v, w):
    assert len(v) == len(w)   # 두 벡터의 길이가 같은 경우만 취급

    return [v_i - w_i for v_i, w_i in zip(v, w)]
```

In [9]:

```
subtractV([3, 1], [-2, 5])
```

Out[9]:

```
[5, -4]
```

벡터 항목별 곱셈

동일 차원의 벡터 두 개의 항목별 곱셈

$$[u_1, \cdots, u_n] * [v_1, \cdots, v_n] = [u_1 * v_1, \cdots, u_n * v_n]$$

In [10]:

```
def multiplyV(v, w):
    assert len(v) == len(w)   # 두 벡터의 길이가 같은 경우만 취급

    return [v_i * w_i for v_i, w_i in zip(v, w)]
```

In [11]:

```
multiplyV([3, 1], [-2, 5])
```

Out[11]:

```
[-6, 5]
```

벡터 항목별 나눗셈

동일 차원의 벡터 두 개의 항목별 나눗셈

$$[u_1, \cdots, u_n]/[v_1, \cdots, v_n] = [u_1/v_1, \cdots, u_n/v_n]$$

In [12]:

```
def divideV(v, w):
    assert len(v) == len(w)   # 두 벡터의 길이가 같은 경우만 취급

    return [v_i / w_i for v_i, w_i in zip(v, w)]
```

In [13]:

```
divideV([95, 80, 75, 62], [85, 82, 79, 82])
```

Out[13]:

```
[1.1176470588235294, 0.975609756097561, 0.9493670886075949, 0.7560975609756098]
```

벡터 스칼라 곱셈

스칼라 곱셈은 벡터의 각 항목을 지정된 수로 곱하기

$$c * [u_1, \cdots, u_n] = [c * u_1, \cdots, c * u_n]$$

In [14]:

```
def scalar_multiplyV(c, v):
    return [c * v_i for v_i in v]
```

In [15]:

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

Out[15]:

```
[2, 4, 6]
```

항목별 평균 벡터

여러 개의 동일 차원 벡터가 주어졌을 때 항목별 평균 구하기

$$\frac{1}{3} * ([1, 2] + [2, 1] + [2, 3]) = \frac{1}{3} * [1 + 2 + 2, 2 + 1 + 3] = [5/3, 2]$$

In [16]:

```
def meanV(vectors):
    n = len(vectors)

    return scalar_multiplyV(1/n, vector_sum(vectors))
```

In [17]:

```
meanV([[3, 2, 6], [2, 5, 9], [7, 5, 1], [6, 3, 4]])
```

Out[17]:

```
[4.5, 3.75, 5.0]
```

벡터 내적

동일 차원의 벡터 두 개의 내적: 위치에 있는 항목끼기 곱한 후 모두 더한 값

$$[u_1, \cdots, u_n] \cdot [v_1, \cdots, v_n] = \sum_{i=1}^{n} u_i * v_i = u_1 * v_1 + \cdots + u_n * v_n$$

In [18]:

```
def dotV(v, w):
    assert len(v) == len(w), "벡터들의 길이가 동일해야 함"""

    return sum(v_i * w_i for v_i, w_i in zip(v, w))
```

In [19]:

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

Out[19]:

32

벡터 크기

벡터 $v = [v_1, \cdots, v_n]$의 크기: v 자신과의 내적의 제곱근

$$\|v\| = \sqrt{v * v} = \sqrt{v_1^2 + \cdots + v_n^2}$$

$$\|\,[3, 4]\,\| = \sqrt{3^2 + 4^2} = \sqrt{5^2} = 5$$

In [20]:

```
import math

def norm(v):
    sum_of_squares = dotV(v, v)
    return math.sqrt(sum_of_squares)
```

In [21]:

```
norm([3, 4])
```

Out[21]:

```
5.0
```

8.2. 행렬

- 행렬의 모양
- 행벡터와 열벡터
- 행렬 항목별 연산
- 행렬 곱셈
- 전치 행렬

행렬의 정의

- 행렬matrix: 숫자를 행과 열로 구성된 직사각형 모양으로 나열한 것
- $n \times k$ 행렬: n 개의 행과 k 개의 열로 구성된 행렬
- 리스트의 리스트, 즉 2중 리스트로 구현 가능

In [22]:

```
# 2x3 행렬

A = [[1, 2, 3],
     [4, 5, 6]]
```

In [23]:

```
# 3x2 행렬

B = [[1, 2],
     [3, 4],
     [5, 6]]
```

행렬의 모양

- $n \times k$ 행렬의 모양shape: (n, k)
- 예제: $1, 2, 3, 4, 5, 6$ 여섯 개의 항목을 가진 행렬의 모양은 네 종류
- (1, 6) 모양의 행렬: 한 개의 행과 여섯 개의 열

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

- (2, 3) 모양의 행렬: 두 개의 행과 세 개의 열

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

- (3, 2) 모양의 행렬: 세 개의 행과 두 개의 열

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

- (6, 1) 모양의 행렬: 여섯 개의 행과 한 개의 열

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix}$$

`shape()` 함수

`shape()` 함수: 주어진 행렬의 모양을 튜플로 반환

In [24]:

```
def shape(M):
    """
    M: 행렬
    M[i]의 길이가 일정하다고 가정
    """

    num_rows = len(M)    # 행의 수
    num_cols = len(M[0]) # 열의 수
    return num_rows, num_cols
```

In [25]:

```
shape(A)
```

Out[25]:

```
(2, 3)
```

행과 열의 인덱스

행 인덱스

		0	1	2	3	4
	0	10	15	23	31	3
열 인덱스	1	13	72	29	19	85
	2	61	42	1	5	27

행벡터와 열벡터

지정된 인덱스의 행과 지정된 인덱스의 열의 항목들로 구성된 행벡터와 열벡터 생성

In [26]:

```
# i번 행벡터
def get_row(M, i):
    """
    M: 행렬
    i: 행 인덱스
    """

    return M[i]
```

In [27]:

```
# j번 열벡터
def get_column(M, j):
    """
    M: 행렬
    j: 열 인덱스
    """

    return [M_i[j] for M_i in M]
```

예제:

In [28]:

```
get_row(A, 0)
```

Out[28]:

```
[1, 2, 3]
```

In [29]:

```
get_column(B, 1)
```

Out[29]:

```
[2, 4, 6]
```

행렬 초기화

`make_matrix(n, m, entry_fn)` 함수:

- 인자
    - `n`: 행의 수
    - `m`: 열의 수
    - `entry_fn`: i, j가 주어지면 i행, j열에 위치한 항목 계산
- 반환값: 지정된 방식으로 계산된 (i, j) 모양의 행렬

In [30]:

```python
def make_matrix(n, m, entry_fn):
    """
    n: 행의 수
    m: 열의 수
    entry_fn: (i, j)에 대해 i행, j열에 위치한 항목 계산
    """

    return [ [entry_fn(i, j) for j in range(m)] for i in range(n) ]
```

zeros() 함수

In [31]:

```
def zeros(x):
    """
    x = (n, m), 단 n, m은 양의 정수
    """

    n = x[0]
    m = x[1]
    zero_function = lambda i, j: 0

    return make_matrix(n, m, zero_function)
```

In [32]:

```
zeros((5,7))
```

Out[32]:

```
[[0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0]]
```

1-행렬

In [33]:

```
def ones(x):
    """
    x = (n, m), 단 n, m은 양의 정수
    """

    n = x[0]
    m = x[1]
    one_function = lambda i, j: 1

    return make_matrix(n, m, one_function)
```

In [34]:

```
ones((5,7))
```

Out[34]:

```
[[1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1, 1, 1]]
```

임의 행렬

In [35]:

```
import random

def rand(n, m):
    """
    n, m: 양의 정수
    """

    random_function = lambda i, j: random.random()

    return make_matrix(n, m, random_function)
```

In [36]:

```
rand(5,3)
```

Out[36]:

```
[[0.4592751016068926 7, 0.4876245092149326, 0.6911436516639086],
 [0.2087119141181471 6, 0.6413683277305281, 0.1394472933026124],
 [0.6696161483067866, 0.592467500785451 5, 0.4674828617026930 4],
 [0.2103931214319211 3, 0.9987082638557854, 0.5899566381059888],
 [0.9045701345816147, 0.0189305618563891 94, 0.8296168810066614]]
```

항등행렬

In [72]:

```
def identity(n):
    """
    n: 양의 정수
    """
    one_function = lambda i, j: 1 if i == j else 0
    return make_matrix(n, n, one_function)
```

In [76]:

```
identity(5)
```

Out[76]:

```
[[1, 0, 0, 0, 0],
 [0, 1, 0, 0, 0],
 [0, 0, 1, 0, 0],
 [0, 0, 0, 1, 0],
 [0, 0, 0, 0, 1]]
```

행렬 항목별 연산

- 행렬 항목별 사칙연산
- 행렬 스칼라 곱셈

행렬 항목별 덧셈

$$\begin{bmatrix} 1 & 3 & 7 \\ 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 5 \\ 7 & 5 & 0 \end{bmatrix} = \begin{bmatrix} 1+0 & 3+0 & 7+5 \\ 1+7 & 0+5 & 0+0 \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 3 & 12 \\ 8 & 5 & 0 \end{bmatrix}$$

In [77]:

```
def addM(A, B):
    assert shape(A) == shape(B)

    m, n = shape(A)
    return make_matrix(m, n, lambda i, j: A[i][j] + B[i][j])
```

In [79]:

```
C = [[1, 3, 7],
     [1, 0, 0]]

D = [[0, 0, 5],
     [7, 5, 0]]
```

In [44]:

```
addM(C, D)
```

Out[44]:

```
[[1, 3, 12], [8, 5, 0]]
```

행렬 항목별 뺄셈

$$\begin{bmatrix} 1 & 3 & 7 \\ 1 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 5 \\ 7 & 5 & 0 \end{bmatrix} = \begin{bmatrix} 1-0 & 3-0 & 7-5 \\ 1-7 & 0-5 & 0-0 \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 3 & 2 \\ -6 & -5 & 0 \end{bmatrix}$$

In [45]:

```
def subtractM(A, B):
    assert shape(A) == shape(B)

    m, n = shape(A)

    return make_matrix(m, n, lambda i, j: A[i][j] - B[i][j])
```

In [46]:

```
subtractM(C, D)
```

Out[46]:

```
[[1, 3, 2], [-6, -5, 0]]
```

행렬 스칼라 곱셈

$$2 * \begin{bmatrix} 1 & 8 & -3 \\ 4 & -2 & 5 \end{bmatrix} = \begin{bmatrix} 2*1 & 2*8 & 2*-3 \\ 2*4 & 2*-2 & 2*5 \end{bmatrix} = \begin{bmatrix} 2 & 16 & -6 \\ 8 & -4 & 10 \end{bmatrix}$$

In [47]:

```
def scalar_multiplyM(c, M):
    return [[c * row_i for row_i in row] for row in M]
```

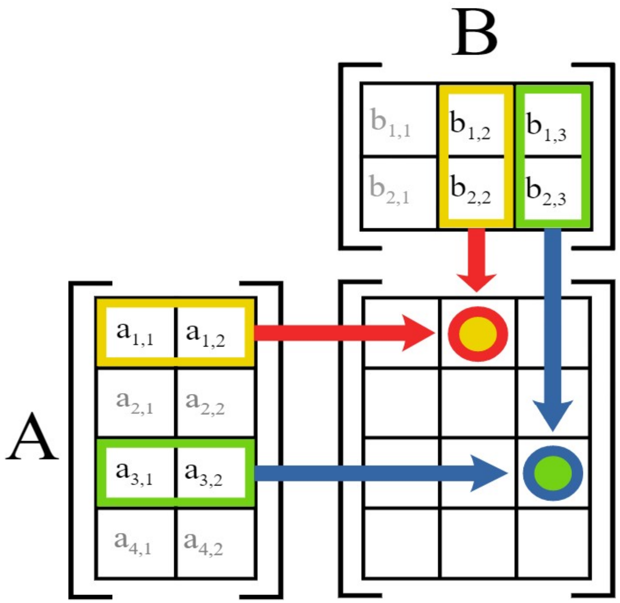
In [48]:

```
scalar_multiplyM(2, C)
```

Out[48]:

```
[[2, 6, 14], [2, 0, 0]]
```

행렬 곱셈



$$\begin{bmatrix} 1 & 0 & 2 \\ -1 & 3 & 1 \end{bmatrix} * \begin{bmatrix} 3 & 1 \\ 2 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} (1*3+0*2+2*1) & (1*1+0*1+2*0) \\ (-1*3+3*2+1*1) & (-1*1+3*1+1*0) \end{bmatrix}$$

$$= \begin{bmatrix} 5 & 1 \\ 4 & 2 \end{bmatrix}$$

In [65]:

```
def matmul(A, B):
    """
    A: (m, n) 모양의 행렬(2중 리스트)
    B: (n, p) 모양의 행렬(2중 리스트)
    """

    mat_mul = [[sum(a*b for a,b in zip(A_row, B_col)) for B_col in zip(*B)] for A_row in A]
    return mat_mul
```

In [66]:

```
# 3x2 행렬
A = [[2, 7],
     [4, 5],
     [7, 8]]

# 2x4 행렬
B = [[5, 8, 1, 2],
     [4, 5, 9, 1]]
```

In [67]:

```
matmul(A, B)
```

Out[67]:

```
[[38, 51, 65, 11], [40, 57, 49, 13], [67, 96, 79, 22]]
```

행렬 곱셈의 항등원

항등행렬은 행렬 곱셈의 항등원임

$$\begin{bmatrix} 3 & 1 \\ 2 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} (3*1+1*0) & (3*0+1*1) \\ (2*1+1*0) & (2*0+1*1) \\ (1*1+0*0) & (1*0+0*1) \end{bmatrix} = \begin{bmatrix} 3 & 1 \\ 2 & 1 \\ 1 & 0 \end{bmatrix}$$

In [68]:

```
# 3x2 행렬
M = [[3, 1],
     [2, 1],
     [1, 0]]

matmul(M, identity(2)) == M
```

Out[68]:

```
True
```

전치행렬

A^T: 행렬 A의 행과 열을 바꾼 것

$$\begin{bmatrix} 9 & 8 & 7 \\ -1 & 3 & 4 \end{bmatrix}^T = \begin{bmatrix} 9 & -1 \\ 8 & 3 \\ 7 & 4 \end{bmatrix}$$

In [69]:

```
def transpose(M):
    """
    M: (m, n) 모양의 행렬
    """

    return [list(col) for col in zip(*M)]
```

In [70]:

```
X = [[9, 8, 7],
     [-1, 3, 4]]
```

In [71]:

```
transpose(X)
```

Out[71]:

```
[[9, -1], [8, 3], [7, 4]]
```

전치행렬의 성질

a를 스칼라, A와 B를 크기가 같은 행렬이라 하자. 이때 다음이 성립한다.

- $(A^T)^T = A$
- $(A + B)^T = A^T + B^T$
- $(A - B)^T = A^T - B^T$
- $(a * A)^T = a * A^T$
- $(A \cdot B)^T = B^T \cdot A^T$