순천향대학교
SOON CHUN HYANG UNIVERSITY

2024년 2학기

선형대수

4장

Chapter 04

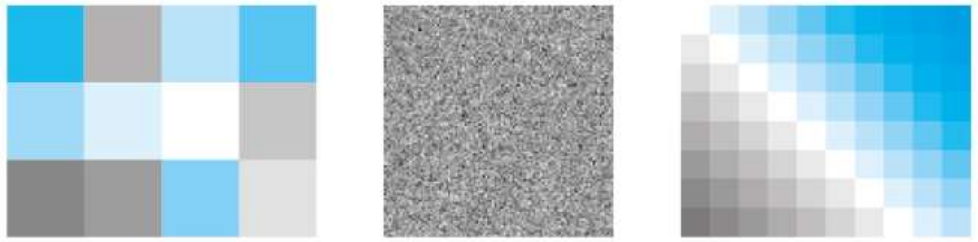
- SECTION 04-1 NumPy에서 행렬 생성과 시각화
- SECTION 04-2 행렬 수학: 덧셈, 스칼라 곱셈, 아다마르곱
- SECTION 04-3 표준 행렬 곱셈
- SECTION 04-4 행렬 연산: 전치
- SECTION 04-5 행렬 연산: LIVE EVIL(연산 순서)
- SECTION 04-6 대칭 행렬

SCH

SECTION 04-1 NumPy에서 행렬 생성과 시각화(1)

4.1.1 행렬 시각화와 인덱싱, 슬라이싱

- 큰 행렬은 이미지로 시각화
  - 행렬의 각 원소의 수치는 이미지의 색상에 대응
    - 일반적으로 임의로 수치를 색상에 대응하기 때문에 색은 중요하지 않음
    - 파이썬 라이브러리 matplotlib를 사용해서 이미지로 시각화된 행렬의 예제



이미지로 시각화된 세 개의 행렬

SECTION 04-1 NumPy에서 행렬 생성과 시각화(2)

- 행과 열 위치를 인덱싱해서 행렬의 특정 원소를 참조
  - 행렬 **A** 의 세 번째 행과 네 번째 열에 있는 원소는 $a_{3,4}$로 나타냄
  - 수학은 1 기반 인덱싱을 사용하지만 파이썬은 0 기반 인덱싱을 사용
    - 원소 $a_{3,4}$ 는 파이썬에서 **A[2,3]**으로 인덱싱

$$\begin{bmatrix} 1 & 3 & 5 & 7 & 9 \\ 0 & 2 & 4 & 6 & 8 \\ 1 & 4 & 7 & 8 & 9 \end{bmatrix}$$

- 하나의 큰 행렬에서 행 2~4와 열 1~5의 부분 행렬을 추출하는 코드

```
A = np.arange(60).reshape(6,10)
sub = A[1:3:1,0:4:1]
```

```
원본 행렬:
[[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]
 [20 21 22 23 24 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]]
부분 행렬:
[[10 11 12 13 14]
 [20 21 22 23 24]
 [30 31 32 33 34]]
```

SECTION 04-1 NumPy에서 행렬 생성과 시각화(3)

4.1.2 특수 행렬

- 난수 행렬(random numbers matrix)
  - 일반적으로 가우스(정규)와 같은 분포로부터 무작위로 추출된 숫자를 가진 행렬
  - 임의의 크기와 계수(rank)로 쉽고 빠르게 생성
  - 여기 학습에서는 주로 가우스 정규분포를 사용

```
Mrows = 4 # shape 0
Ncols = 6 # shape 1
A = np.random.randn(Mrows,Ncols)
```

- 정방 행렬과 비정방 행렬
  - 정방 행렬(square matrix) -행 수와 열 수가 같음 ($\mathbb{R}^{N\times N}$ 행렬)
  - 비정방 행렬(nonsquare matrix) -행 수와 열 수가 다름
    - 비정방 행렬의 행 수가 열 수보다 많으면 높다고 하고 열 수가 행 수보다 많으면 넓다고 표현

SECTION 04-1 NumPy에서 행렬 생성과 시각화(4)

- 대각 행렬
  - 행렬의 대각(diagonal)은 왼쪽 위에서 시작하여 오른쪽 아래로 내려가는 원소들
  - 대각행렬(diagonal matrix) - 모든 비대각형 원소가 0이며, 대각 원소는 0일 수도 있지만 0이 아닌 값을 가질 수 있는 유일한 원소
  - NumPy 함수 np.diag()는 입력에 따라 두 가지 방식으로 동작
    - np.diag에 행렬을 입력하면 대각 원소를 벡터로 반환
    - np.diag에 벡터를 입력하면 대각선에 해당 벡터 원소가 있는 행렬을 반환

```
a = np.array([1,2,3])
da = np.diag(a)
print(a)
print(da)

b = np.arange(9).reshape(3,3)
db = np.diag(b)
print(b)
print(db)
```

SECTION 04-1 NumPy에서 행렬 생성과 시각화(4)

- 삼각 행렬
  - 삼각 행렬(triangular matrix) - 주 대각선의 위 또는 아래가 모두 0
    - 상삼각 행렬 - 0이 아닌 원소가 대각선 위에 있음
    - 하삼각 행렬 - 0이 아닌 원소가 대각선 아래에 있음
  - 행렬의 위(np.triu()) 또는 아래(np.tril()) 삼각형을 추출하는 NumPy 전용 함수

```
a = np.arange(25).reshape(5,5)
print(a)
print(np.triu(a))
print(np.tril(a))
```

SECTION 04-1 NumPy에서 행렬 생성과 시각화(5)

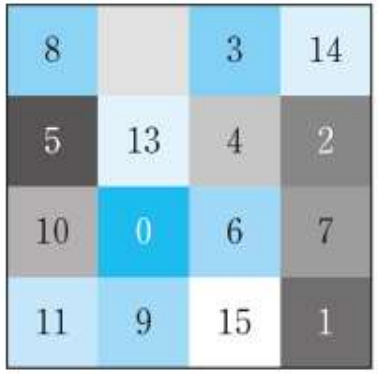
- 단위 행렬
  - 특수 행렬 중 가장 중요한 행렬은 단위 행렬(identity matrix)
  - 행렬 또는 벡터에 단위 행렬을 곱하면 동일한 행렬 또는 벡터가 된다는 점에서 숫자 1과 동
  - 단위 행렬은 모든 대각 원소가 1인 정방 대각 행렬(단위 행렬은 문자 I 로 나타냄)
  - 행렬의 크기는 아래 첨자로 표시(예를 들어 $\mathbf{I}_5$는 5 × 5 단위 행렬)
  - 파이썬에서는 np.eye()를 사용해서 단위 행렬을 구현

- 영 행렬
  - 영 행렬(zeros matrix)은 영벡터와 유사
  - 모든 원소가 0인 행렬
  - 파이썬에서 영 행렬은 np.zeros() 함수를 사용해서 생성

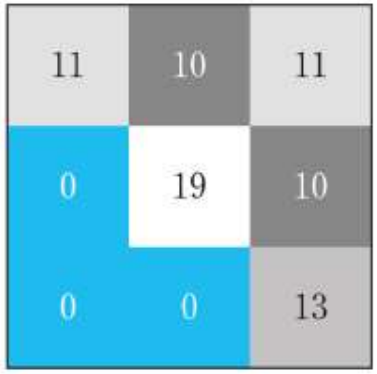
```
a = np.eye(5)
b = np.zeros(3)
c = np.zeros((3,3))
print(a)
print(b)
print(c)
```

SECTION 04-1 NumPy에서 행렬 생성과 시각화(6)

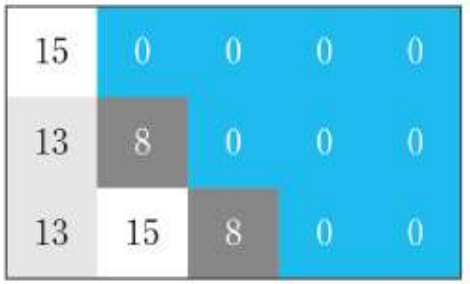
정사각



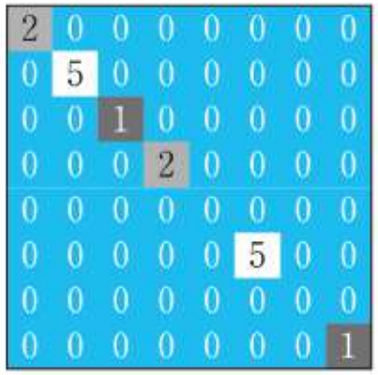
상삼각



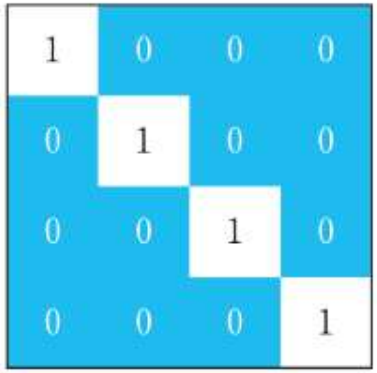
하삼각



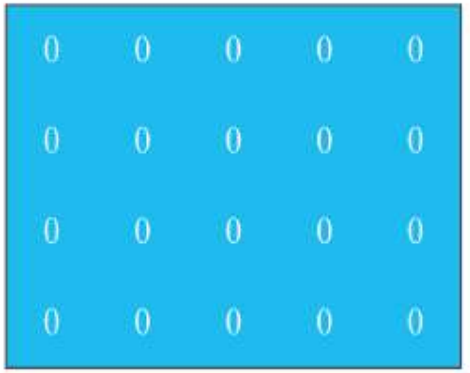
대각



단위



영 제로



SECTION 04-2 행렬 수학: 덧셈, 스칼라 곱셈, 아다마르곱(1)

4.2.1 덧셈과 뺄셈

- 두 행렬을 더할 때는 대응되는 원소끼리 더함
- 행렬 덧셈은 크기가 같은 두 행렬 사이에서만 성립

$$\begin{bmatrix} 2 & 3 & 4 \\ 1 & 2 & 4 \end{bmatrix} + \begin{bmatrix} 0 & 3 & 1 \\ -1 & -4 & 2 \end{bmatrix} = \begin{bmatrix} (2+0) & (3+3) & (4+1) \\ (1-1) & (2-4) & (4+2) \end{bmatrix} = \begin{bmatrix} 2 & 6 & 5 \\ 0 & -2 & 6 \end{bmatrix}$$

실습 1

- 아래의 식을 numpy 코드로 작성

$$\begin{bmatrix} 2 & 3 & 4 \\ 1 & 2 & 4 \end{bmatrix} + \begin{bmatrix} 0 & 3 & 1 \\ -1 & -4 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 6 & 5 \\ 0 & -2 & 6 \end{bmatrix}$$

SECTION 04-2 행렬 수학: 덧셈, 스칼라 곱셈, 아다마르곱(2)

4.2.2 행렬 '이동'

- 벡터와 마찬가지로 행렬에서도 λ+**A**처럼 스칼라를 더할 수 없음
  - 그러나 파이썬에서는 행렬의 요소에 스칼라를 추가하는 브로드캐스팅 연산(예를 들어 3+np.eye(2))이 가능
- 행렬 이동(shifting a matrix) - 정방 행렬에 스칼라를 더하는 방식
  - 대각에 스칼라 값을 더하는 것과 같이 단위 행렬에 스칼라를 곱해서 더하는 방식으로 구현

$$A + \lambda I$$

$$\begin{bmatrix} 4 & 5 & 1 \\ 0 & 1 & 11 \\ 4 & 9 & 7 \end{bmatrix} + 6 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 10 & 5 & 1 \\ 0 & 7 & 11 \\ 4 & 9 & 13 \end{bmatrix}$$

실습 2

- 아래의 식을 numpy 코드로 작성

$$\begin{bmatrix} 4 & 5 & 1 \\ 0 & 1 & 11 \\ 4 & 9 & 7 \end{bmatrix} + 6 = \textbf{?}$$

$$\begin{bmatrix} 4 & 5 & 1 \\ 0 & 1 & 11 \\ 4 & 9 & 7 \end{bmatrix} + 6 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 10 & 5 & 1 \\ 0 & 7 & 11 \\ 4 & 9 & 13 \end{bmatrix}$$

SECTION 04-2 행렬 수학: 덧셈, 스칼라 곱셈, 아다마르곱(3)

- 파이썬에서 이동하는 코드

```
A = np.array([ [4,5,1],[0,1,11],[4,9,7] ])
s = 6
A + s # 이동이 아님!
A + s*np.eye(len(A)) # 이동
```

- 행렬 '이동'의 두 가지 주요 응용
  - 행렬의 고유값을 찾는 메커니즘
  - 모델을 데이터에 적합시킬 때 행렬을 정규화하는 메커니즘

SECTION 04-2 행렬 수학: 덧셈, 스칼라 곱셈, 아다마르곱(4)

4.2.3 스칼라 곱셈과 아다마르곱

- 스칼라-행렬 곱셈은 행렬의 각 원소에 동일한 스칼라를 곱
  - 숫자 대신 문자로 이루어진 행렬을 사용한 예제

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

- 아다마르곱 - 두 행렬을 요소별로 곱(원소별 곱셈)

$$\begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} \odot \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} 2a & 3b \\ 4c & 5d \end{bmatrix}$$

  - np.multiply() 함수를 사용해서 구현

```
A = np.random.randn(3,4)
B = np.random.randn(3,4)
A*B # 아다마르곱
np.multiply(A,B) # 마찬가지로 아다마르
A@B # 아다마르가 아님!
```

```
a = np.arange(12).reshape(3,4)
b = np.ones(12).reshape(3,4)
print(a*b)
print(np.multiply(a, b))
print(a@b.T)
print(np.dot(a,b.T))
```

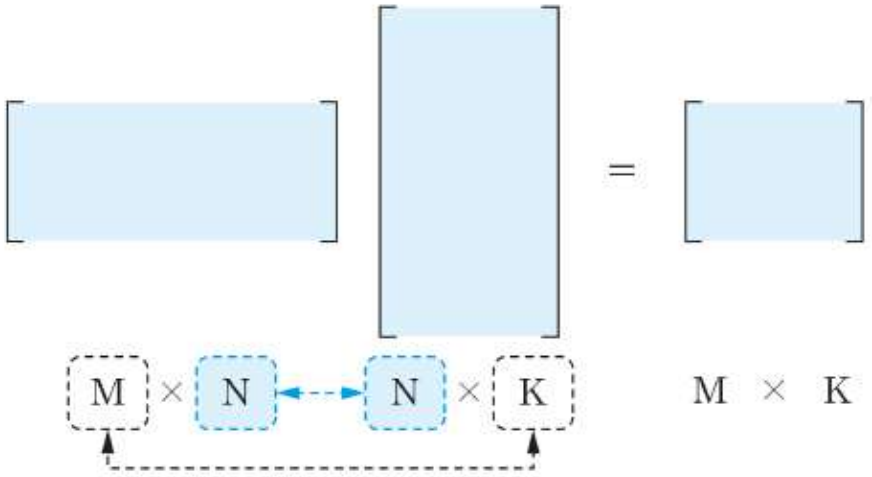
SECTION 04-3 표준 행렬 곱셈(1)

- 표준 행렬 곱셈은 원소별이 아닌 행과 열 단위로 동작
  - 한 행렬의 행과 다른 행렬의 열 사이 스칼라 곱셈의 조직적 집합
  - 행렬 곱셈(matrix multiplication)
    - 표준이라는 용어를 추가한 이유는 아다마르곱, 스칼라 곱셈과 명확하게 구별하기 위함
  - 두 행렬의 크기가 서로 짝이 맞을 때 두 행렬을 곱할 수 있음

SECTION 04-3 표준 행렬 곱셈(2)

4.3.1 행렬 곱셈 유효성에 관한 규칙

- 행렬 곱셈은 '내부' 차원의 수가 일치할 때만 유효하고 곱셈 행렬의 크기는 '외부' 차원의 수로 정의
  - 행렬 곱셈은 왼쪽 행렬의 열 수가 오른쪽 행렬의 행 수와 같을 때 유효
  - 곱셈 행렬의 크기는 왼쪽 행렬의 행 수와 오른쪽 행렬의 열 수로 정의
  - 표기법 - 아다마르곱은 점 원(**A⊙B**)으로 표시하지만 행렬 곱셈은 아무런 기호 없이(**AB**) 두 행렬을 나란히 적음
  - **C=AB** 및 **D=BA** 라면 일반적으로는 **C≠D**

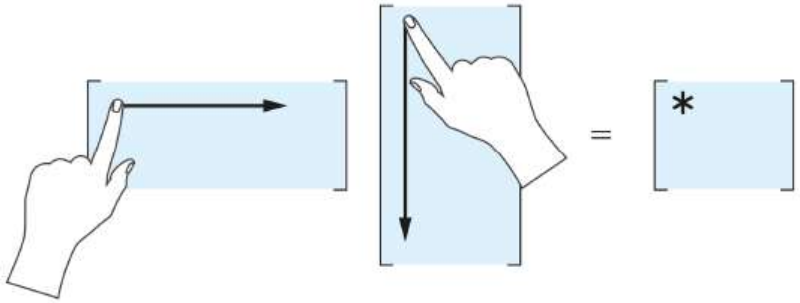


SECTION 04-3 표준 행렬 곱셈(3)

4.3.2 행렬 곱셈

- 왼쪽 행렬의 열 수가 오른쪽 행렬의 행 수와 일치하는 경우에만 행렬 곱셈이 유효한 이유
  - 곱셈 행렬의 (I,j)번째 원소가 왼쪽 행렬의 i번째 행과 오른쪽 행렬의 j번째 열 사이의 내적이기 때문임

$$\begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} (2a+3c) & (2b+3d) \\ (4a+5c) & (4b+5d) \end{bmatrix}$$



SECTION 04-3 표준 행렬 곱셈(4)

4.3.3 행렬-벡터 곱셈

- 행렬-벡터 곱셈의 기본
  - 행벡터가 아닌 열벡터만 행렬의 오른쪽에 곱할 수 있음
  - 열벡터가 아닌 행벡터만 행렬의 왼쪽에 곱할 수 있음
  - 즉 $\mathbf{Av}$ 및 $\mathbf{v}^{\mathrm{T}}\mathbf{A}$ 는 유효하지만 $\mathbf{Av}^{\mathrm{T}}$ 및 $\mathbf{vA}$ 는 유효하지 않음
  - $M \times N$ 행렬의 왼쪽에 $1 \times M$ 행렬(행벡터)을 곱하거나 오른쪽에 $N \times 1$ 행렬(열벡터)을 곱할 수 있음
- 행렬-벡터 곱셈의 결과는 항상 벡터
  - 결과 벡터의 방향은 곱하는 벡터의 방향에 따라 결정
  - 행렬에 행벡터를 앞에서 곱하면 다른 행벡터가 생성되지만 행렬에 열벡터를 뒤에서 곱하면 다른 열벡터가 생성됨

SECTION 04-3 표준 행렬 곱셈(5)

4.3.3 행렬-벡터 곱셈

- 행렬-벡터 곱셈의 응용 분야
  - 통계에서 모델 예측 데이터값은 설계 행렬에 회귀 계수를 곱해서 구하며 이는 **X**β로 표시
  - 주성분 분석에서는 데이터 집합 *Y*의 분산을 최대화하도록 '특징 중요도' 가중치의 벡터를 얻음
  - 다변량 신호 처리에서 공간 필터를 다채널 시계열 데이터 **S**에 적용하여 축소된 차원 성분을 얻음
  - 기하학적 구조와 컴퓨터 그래픽에서 이미지 좌표 집합은 수학적 변환 행렬을 사용하여 변환될 수 있으며 **Tp** 로 나타냄
    - **T**는 변환 행렬이고 **p**는 기하학적 좌표 집합

SECTION 04-3 표준 행렬 곱셈(6)

4.3.3.1 선형 가중 결합

- 각 벡터를 행렬에 넣고 가중치를 벡터의 원소로 넣기
  - 스칼라 벡터의 각 원소를 행렬의 해당 열에 곱한 다음 가중된 열벡터를 합하여 곱을 구하는 것

$$4\begin{bmatrix}3\\0\\6\end{bmatrix}+3\begin{bmatrix}1\\2\\5\end{bmatrix}\Rightarrow\begin{bmatrix}3&1\\0&2\\6&5\end{bmatrix}\begin{bmatrix}4\\3\end{bmatrix}$$

  - 행벡터를 선형 가중 결합하는 방법은 계수를 행벡터에 넣고 앞에서 곱함

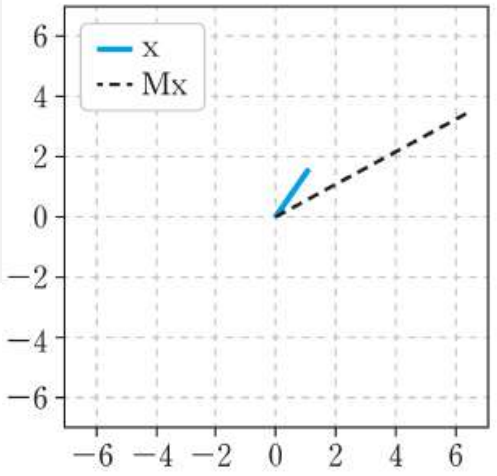
SECTION 04-3 표준 행렬 곱셈(7)

4.3.3.2 기하학적 변환

- 벡터를 기하학적 선으로 생각하면 행렬-벡터 곱셈으로 해당 벡터를 회전하고 크기를 조정
  - 2차원 예제 코드

```
import matplotlib.pyplot as plt
M  = np.array([ [2,3],[2,1] ])
x  = np.array([ [1,1.5] ]).T
Mx = M@x

plt.figure(figsize=(6,6))
plt.plot([0,x[0,0]],[0,x[1,0]],'-',linewidth=4,color=[.0,.5,.7],label='x')
plt.plot([0,Mx[0,0]],[0,Mx[1,0]],'--',linewidth=3,color=[.3,.3,.3],label='Mx')
plt.xlim([-7,7])
plt.ylim([-7,7])
plt.legend() #범례
plt.grid()
plt.savefig('Figure_04_05a.png',dpi=300)
plt.show()
```



SECTION 04-4 행렬 연산: 전치(1)

- 벡터와 마찬가지로 전치는 위 첨자 [ ]^T 로 표시(따라서 **C**^{T}는 **C**의 전치)
  - 행렬을 이중 전치하면 원래 행렬(**C**^{TT} = **C**)

$$a_{i,j}^{\mathrm{T}} = a_{j,i}$$

- 2차원 NumPy 배열을 사용하여 전치하는 코드 예제

```
A = np.array([ [3,4,5],[1,2,3] ])
A_T1 = A.T # 메서드
A_T2 = np.transpose(A) # 함수
```

SECTION 04-4 행렬 연산: 전치(2)

4.4.1 내적 표기법

- 벡터-내적의 표기
  - 크기가 M×1인 두 열벡터에서 첫 번째 벡터만 전치하면 크기가 1×M 이고 M×1 인 두 개의 '행렬'이 존재
  - '내부' 차원은 일치하고 '외부' 차원을 통해 곱셈의 결과가 1×1 (즉, 스칼라가 됨)
  - 내적을 $\mathbf{a}^{\mathrm{T}}\mathbf{b}$로 표시

SECTION 04-5 행렬 연산: LIVE EVIL(연산 순서)

- LIVE EVIL
  - 여러 행렬의 곱셈을 전치할 때 순서가 어떻게 되는지 기억하기 위한 귀여운 연상법(뒤집어도 철자가 동일)
  - 기본적으로 여러 행렬의 곱셈을 전치하면 개별 행렬을 전치하고 곱한 것과 동일하지만 순서는 뒤바뀜
    - [식 4-3]에서 **L** , **I**, **V** , **E**는 모두 행렬이며 곱셈이 가능하도록 행렬의 크기는 서로 짝이 맞다고 가정

$$(\mathrm{LIVE})^{\mathrm{T}} = \mathrm{E}^{\mathrm{T}}\mathrm{V}^{\mathrm{T}}\mathrm{I}^{\mathrm{T}}\mathrm{L}^{\mathrm{T}}$$

SECTION 04-6 대칭 행렬(1)

- 대칭 행렬(Symmetric matrix)은 수치적으로 안정적인 경향이 있어서 특히 컴퓨터 알고리즘을 처리할 때 편리
- 대칭 행렬은 자신의 전치 행렬과 같다는 것을 의미
  - $\mathbf{A}^T = \mathbf{A}$ 이면 행렬 $\mathbf{A}$ 는 대칭

$$\begin{bmatrix} a & e & f & g \\ e & b & h & i \\ f & h & c & j \\ g & i & j & d \end{bmatrix}$$

SECTION 04-6 대칭 행렬(2)

4.6.1 비대칭 행렬로부터 대칭 행렬 생성하기

- 어떤 행렬이든(비정방 또는 비대칭 행렬이라도) 자신의 전치를 곱하면 정방 대칭 행렬이 생성됨
  - 즉 $\mathbf{A}^T\mathbf{A}$ 와 $\mathbf{A}\mathbf{A}^T$ 모두 정방 대칭 행렬
    (위 첨자 $[\ ]^T$ 를 입력하기 번거롭다면 **AtA**와 **AAt** 또는 **A′A**와 **AA′**로 작성해도 됨)
- A^TA 가 정방이고 **그리고** 대칭임을 증명
  - 정방 - 만약 **A**가 $M \times N$ 이라면 $\mathbf{A}^T\mathbf{A}$ 는 $(N \times M)\ (M \times N)$. 따라서, 곱셈 행렬의 크기는 $N \times N$
    - $\mathbf{A}\mathbf{A}^T$에 대해서도 동일한 논리
  - 대칭성 증명은 LIVE EVIL 규칙을 이용

$$(\mathrm{A}^{\mathrm{T}}\mathrm{A})^{\mathrm{T}} = \mathrm{A}^{\mathrm{T}}\mathrm{A}^{\mathrm{TT}} = \mathrm{A}^{\mathrm{T}}\mathrm{A}$$

- 곱셈 기법(multiplicative method) - $\mathbf{A}^T\mathbf{A}$ 로 대칭 행렬을 만드는 것
  - 행렬이 정방이면서 비대칭일 때 유효한 덧셈 기법도 있음

실습 3

- 비대칭 행렬을 입력 받아 대칭 행렬 리턴하는 함수 정의