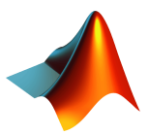


# 매트랩 인덱싱

만든이: (오지게) 게으른 맬랩

같이 보면 좋은 곳: <https://lazymatlab.tistory.com/28>



# 행렬이란?

※ 연산 방법에 주의할 것

- 1차 연립 방정식을 좀 더 간단하게 써보자.

$$\begin{cases} x + 2y - 4z = 5 \\ 2x + y - 6z = 8 \\ 4x - y - 12z = 13 \end{cases}$$

$$\begin{bmatrix} 1 & 2 & -4 \\ 2 & 1 & -6 \\ 4 & -1 & -12 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 5 \\ 8 \\ 13 \end{bmatrix}$$

3×3 행렬

3×1 행렬  
(벡터)

3×1 행렬  
(벡터)

$$2x + y + z = 1$$

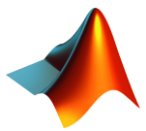
$$5x - y + 7z = 0$$

$$\begin{bmatrix} 2 & 1 & 1 \\ 5 & -1 & 7 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

2×3 행렬

3×1 행렬  
(벡터)

2×1 행렬  
(벡터)

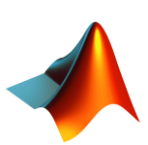
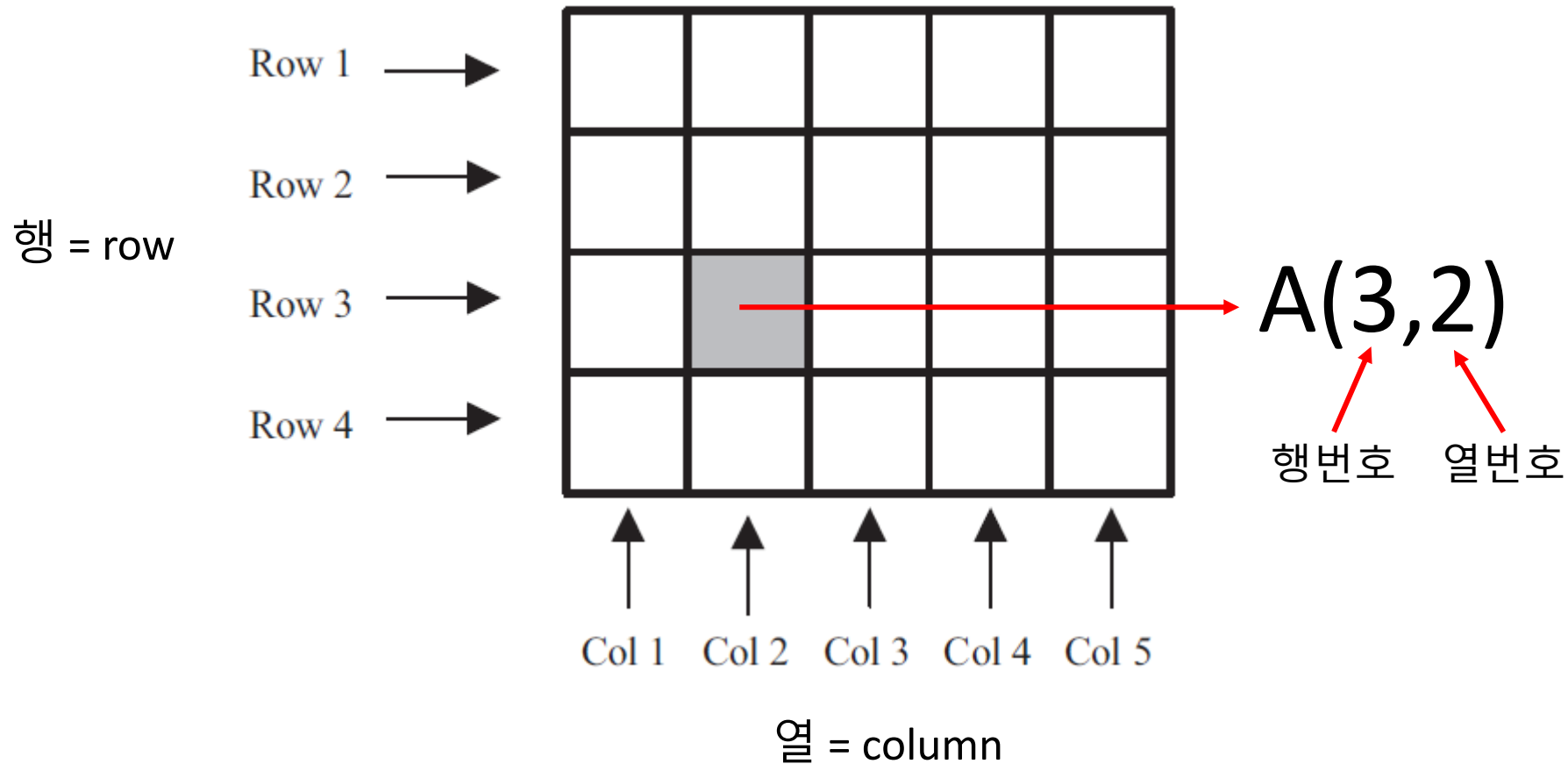


# 행렬이란?

엑셀?

- 1개 이상의 수, 식, 기호 등을 사각형의 배열로 나열한 것

A:  $4 \times 5$  행렬



# 행렬: 수학 vs 매트랩

- 매트랩에서는 모든 것이 행렬이다.
  - 순수 스칼라는 없다.
  - 모든 자료형에 size와 length를 쓸 수 있다.

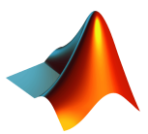
tip1. 매트랩에서...

- array와 matrix는 같은 말
- scalar는 vector의 부분집합
- vector는 matrix의 부분집합
- scalar  $\subset$  vector  $\subset$  matrix

tip2.  $M \times M$ : square matrix

tip3. 3차원? 4차원? N차원?

값	용어	수학에서	매트랩에서
100	스칼라	값 1개	$1 \times 1$ 행렬
$[3 \quad 9 \quad 17], \begin{bmatrix} 5 \\ 8 \\ 13 \end{bmatrix}$	벡터	$1 \times N$ 벡터 (행벡터) $N \times 1$ 벡터 (열벡터)	$1 \times N$ 행렬 (행벡터) $N \times 1$ 행렬 (열벡터)
$\begin{bmatrix} 1 & 2 & -4 \\ 2 & 1 & -6 \\ 4 & -1 & -12 \end{bmatrix}$	행렬	$M \times N$ 행렬	$M \times N$ 행렬



# 행렬을 만드는 방법 1 – 모든 원소를 직접 입력

a = [1,2,3,4];

b = [5 6 7 8];

c = [1,2,3; 4,5,6; 7,8,9];

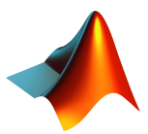
d = [1 10 100

2 20 200

3 30 300];

e = [];

- 변수명 = [값1, 값2, 값3 ; 값4, 값5, 값6];
  - 값의 구분은 빈칸, 쉼표 모두 가능 (가독성?)
  - 빈 행렬: []
- 새로운 행 시작: 엔터 또는 세미콜론
  - 주의: 행이 너무 길어질 땐? → ...를 이용
    - 이때 새로운 행이 시작되는 것이 아님
- 명령창에서 입력하다가 실수했을 때?
  - ctrl+c
  - 대부분의 “취소”에 해당하는 단축키



# 행렬을 만드는 방법 2 – 콜론 연산자

`a1 = 1:10;`       $\rightarrow$       `a1 = [1 2 3 4 5 6 7 8 9 10]`

`a2 = 1:2:9;`       $\rightarrow$       `a2 = [1 3 5 7 9]`

`b1 = 10:-1:5;`       $\rightarrow$       `b1 = [10 9 8 7 6 5]`

`b2 = 10:5;`       $\rightarrow$       `b2 = [ ]`

`c1 = 5:5;`       $\rightarrow$       `c1 = 5`

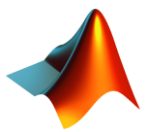
`c2 = 5:10:5;`       $\rightarrow$       `c2 = 5`

`d = 1:10:100;`       $\rightarrow$       `d = [1 11 21 ... 91]`

- 변수명 = 시작값:간격:끝값
  - 간격은 생략하면 1
  - 음수 간격도 가능
  - 시작값 > 끝값이고 간격 > 0이면  
 $\rightarrow$  빈 행렬
  - 시작값 < 끝값이고 간격 < 0이면  
 $\rightarrow$  빈 행렬
  - 끝값에 도달할 수 없으면  
 $\rightarrow$  끝값 전 마지막 값까지

`year = 1984:2:1996`

$\rightarrow$  `year = [1984 1986 1988 1990 1992 1994 1996]`



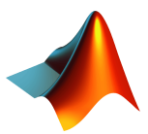
# 행렬을 만드는 방법 3 – linspace

a1 = linspace(1,10,10);      →      a1 = [1 2 3 4 5 6 7 8 9 10]  
a2 = linspace(1,9,5);      →      a2 = [1 3 5 7 9]  
a3 = linspace(1,100);      →      a3 = [1 2 3 ... 99 100]  
b1 = linspace(9,1,5);      →      b1 = [9 7 5 3 1]  
b2 = linspace(100,10,1);      →      b2 = 10  
b3 = linspace(100,10,2);      →      b3 = [100 10]  
c1 = linspace(10,10);      →      c1 = [10 10 10 ... 10]

- linspace(시작값, 끝값, 개수)
  - 간격은 알아서 계산 (=(끝값-시작값)/(개수-1))
  - 참고: 콜론 연산자는 개수를 알아서 계산
  - 개수를 생략하면 100개
  - 끝점은 항상 포함
- 콜론 연산자 vs linspace
  - 간격을 알 때 → 콜론 연산자
    - 1920년에서 5년 간격으로 2020년까지
  - 개수를 알 때 → linspace
    - 0~pi를 1000등분

x = linspace(0, pi, 1000);  
→      x = [0   0.0031   0.0063 ... 3.1416];

※ 열벡터를 만드는 방법  
1) 세미콜론  
2) 행벡터 + transpose( ' )



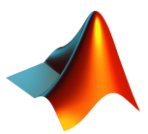
# 행렬을 만드는 방법 4 – 이미 있는 행렬을 합침

```
>> a1 = [1,2,3];  
>> a2 = [4,5,6];  
>> ar = [a1 a2]  
ar =  
      1      2      3      4      5      6  
>> ac = [a1; a2]  
ac =  
      1      2      3  
      4      5      6  
>> at = [a1' a2']  
at =  
      1      4  
      2      5  
      3      6
```

```
>> a3 = [1:3, 4:7]  
a3 =  
      1      2      3      4      5      6      7
```

```
>> a4 = [1:5; 6:10]  
a4 =  
      1      2      3      4      5  
      6      7      8      9     10
```

```
>> a1 = [linspace(10,20,6); linspace(20,30,6)]  
a1 =  
     10     12     14     16     18     20  
     20     22     24     26     28     30
```





# 행렬을 만드는 방법 4 – 이미 있는 행렬을 합침

```
>> a = [1 2; 3 4];
```

```
>> b = [a a ; a a]
```

```
b =
```

1	2	1	2
3	4	3	4
1	2	1	2
3	4	3	4

```
>> a = [1 2; 3 4];
```

```
>> b = [5 6]';
```

```
>> c = [7 8];
```

```
>> d = 9;
```

```
>> e = [a b ;c d]
```

```
e =
```

1	2	5
3	4	6
7	8	9

```
>> a = [1 2 ; 3 4];
```

```
>> b = [5 6];
```

```
>> a = [a;b]
```

```
a =
```

1	2
3	4
5	6

```
>> a = [a (1:3)']
```

```
a =
```

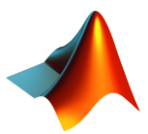
1	2	1
3	4	2
5	6	3

```
>> a = [1 2 3; 4 5 6];
```

```
>> b = [4 5 6]';
```

```
>> c = [a b]
```

다음 사용 중 오류가 발생함: **horzcat**  
결합하려는 배열의 차원이 일치하지 않습니다.



# 행렬을 만드는 방법 5 – 특별한 함수들

```
>> ones(3,4)
```

```
ans =
```

1	1	1	1
1	1	1	1
1	1	1	1

```
>> ones(5)
```

```
ans =
```

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

```
>> zeros(2,4)
```

```
ans =
```

0	0	0	0
0	0	0	0

```
>> zeros(3)
```

```
ans =
```

0	0	0
0	0	0
0	0	0

```
>> a = 1:5;
```

```
>> b = repmat(a,3,1)
```

```
b =
```

1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

```
>> a = 1:30;
```

```
>> b = reshape(a,5,6)
```

```
b =
```

1	6	11	16	21	26
2	7	12	17	22	27
3	8	13	18	23	28
4	9	14	19	24	29
5	10	15	20	25	30



# 행렬의 연산 – 더하기, 빼기, \*스칼라, /스칼라

## 1) 행렬 + 행렬

- 크기가 같은 행렬만 가능
- 각 위치 원소별로 더하기 or 빼기

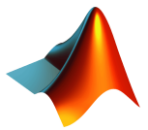
## 2) 행렬 + 스칼라 (1×1 행렬)

- 행렬의 모든 원소에 일괄적으로 스칼라를 더함

## 3) \*스칼라, /스칼라

- 모든 원소를 동일하게 곱하거나 나눔

```
>> a = (1:5);  
>> b = (11:15);  
>> c = a+b  
  
c =  
  
    12    14    16    18    20  
  
>> c = c+10  
  
c =  
  
    22    24    26    28    30  
  
>> c = c*2  
  
c =  
  
    44    48    52    56    60  
  
>> c = c/4  
  
c =  
  
    11    12    13    14    15
```



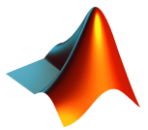
# 행렬의 연산 – 곱하기 (행렬곱)

$$\begin{cases} x + 2y - 4z = 5 \\ 2x + y - 6z = 8 \\ 4x - y - 12z = 13 \end{cases} \quad \begin{bmatrix} 1 & 2 & -4 \\ 2 & 1 & -6 \\ 4 & -1 & -12 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 5 \\ 8 \\ 13 \end{bmatrix}$$

$$\begin{cases} 2x + y + z = 1 \\ 5x - y + 7z = 0 \end{cases} \quad \begin{bmatrix} 2 & 1 & 1 \\ 5 & -1 & 7 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 4 & 3 \\ 2 & 6 & 1 \\ 5 & 2 & 8 \end{bmatrix} \begin{bmatrix} 5 & 4 \\ 1 & 3 \\ 2 & 6 \end{bmatrix} = \begin{bmatrix} (1 \cdot 5 + 4 \cdot 1 + 3 \cdot 2) & (1 \cdot 4 + 4 \cdot 3 + 3 \cdot 6) \\ (2 \cdot 5 + 6 \cdot 1 + 1 \cdot 2) & (2 \cdot 4 + 6 \cdot 3 + 1 \cdot 6) \\ (5 \cdot 5 + 2 \cdot 1 + 8 \cdot 2) & (5 \cdot 4 + 2 \cdot 3 + 8 \cdot 6) \end{bmatrix} = \begin{bmatrix} 15 & 34 \\ 18 & 32 \\ 43 & 74 \end{bmatrix}$$

- 행렬곱이 정의되려면  
→ (앞 행렬의 열 개수) = (뒤 행렬의 행 개수)
- $(M \times N \text{ 행렬}) \times (P \times Q \text{ 행렬})$ 
  - $N=P$ 이어야 행렬곱이 정의됨
  - 곱의 결과 =  $M \times Q$  행렬



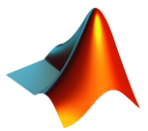
# 행렬의 연산 – 곱하기 (행렬곱)

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \\ A_{41} & A_{42} & A_{43} \end{bmatrix} \text{ and } B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \\ B_{31} & B_{32} \end{bmatrix}$$

$$A \times B = \begin{bmatrix} (A_{11}B_{11} + A_{12}B_{21} + A_{13}B_{31}) & (A_{11}B_{12} + A_{12}B_{22} + A_{13}B_{32}) \\ (A_{21}B_{11} + A_{22}B_{21} + A_{23}B_{31}) & (A_{21}B_{12} + A_{22}B_{22} + A_{23}B_{32}) \\ (A_{31}B_{11} + A_{32}B_{21} + A_{33}B_{31}) & (A_{31}B_{12} + A_{32}B_{22} + A_{33}B_{32}) \\ (A_{41}B_{11} + A_{42}B_{21} + A_{43}B_{31}) & (A_{41}B_{12} + A_{42}B_{22} + A_{43}B_{32}) \end{bmatrix}$$

- 행렬곱이 정의되려면  
→ (앞 행렬의 열 개수) = (뒤 행렬의 행 개수)
- $(M \times N \text{ 행렬}) \times (P \times Q \text{ 행렬})$ 
  - $N=P$ 이어야 행렬곱이 정의됨
  - 곱의 결과 =  $M \times Q$  행렬



# 행렬의 연산 – 곱하기 (행렬곱)

```
>> a
a =
     1     4     7    10
     2     5     8    11
     3     6     9    12

>> b
b =
    16     2     3
     5    11    10
     9     7     6
     4    14    15

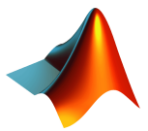
>> c = a*b
c =
   139   235   235
   173   269   269
   207   303   303
```

```
>> a = [1 2 3 4 5];
>> b = [1 2 3 4 5]';
>> c = a*b
c =
    55
```

```
>> s
s =
     8     1     6
     3     5     7
     4     9     2

>> s^2
ans =
    91    67    67
    67    91    67
    67    67    91
```

- 행렬곱이 정의되려면  
→ (앞 행렬의 열 개수) = (뒤 행렬의 행 개수)
- $(M \times N \text{ 행렬}) \times (P \times Q \text{ 행렬})$ 
  - $N=P$ 이어야 행렬곱이 정의됨
  - 곱의 결과 =  $M \times Q$  행렬
- 행렬의 거듭제곱
  - square matrix에 대해서만 정의됨
- 행렬곱의 특징
  - 결합법칙 O       $A(BC) = (AB)C$
  - 분배법칙 O       $A(B+C) = AB + AC$
  - 교환법칙 X       $AB \neq BA$



# 행렬의 연산 – 곱하기 (원소별 곱셈)

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

× × ×

$$b = \begin{bmatrix} 4 & 3 & 1 \end{bmatrix}$$

|| || ||

$$c = \begin{bmatrix} 4 & 6 & 3 \end{bmatrix}$$

```
>> a = [1 2 3];
>> b = [4 3 1];
>> a*b
```

다음 사용 중 오류가 발생함: \*  
행렬 곱셈의 차원이 잘못되었습니다. 첫 번째 행렬의 열 개수가 두 번째 행렬의 행 개수와 일치하는지 확인하십시오. 요소별 곱셈 연산을 수행하려면 .\*를 사용하십시오.

```
>> a.*b

ans =

     4     6     3
```

```
>> s = magic(3)
```

```
s =
```

```
     8     1     6
     3     5     7
     4     9     2
```

```
>> s^2
```

```
ans =
```

```
    91    67    67
    67    91    67
    67    67    91
```

```
>> s.*^2
```

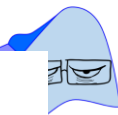
```
ans =
```

```
    64     1    36
     9    25    49
    16    81     4
```

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \text{ and } B = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix}$$

- \*는 행렬곱
- .\*는 원소별 곱셈
- 두 행렬의 크기는 같아야 함

$$A .* B = \begin{bmatrix} A_{11}B_{11} & A_{12}B_{12} & A_{13}B_{13} \\ A_{21}B_{21} & A_{22}B_{22} & A_{23}B_{23} \\ A_{31}B_{31} & A_{32}B_{32} & A_{33}B_{33} \end{bmatrix} \quad A.^n = \begin{bmatrix} (A_{11})^n & (A_{12})^n & (A_{13})^n \\ (A_{21})^n & (A_{22})^n & (A_{23})^n \\ (A_{31})^n & (A_{32})^n & (A_{33})^n \end{bmatrix}$$



# 행렬의 연산 – 곱하기 (원소별 곱셈)

- 행렬의 원소별 곱셈 → 연산자 앞에 점(.)을 찍음
  - 두 행렬을 원소별 곱셈 →  $A.*B$
  - 한 행렬을 원소별 거듭제곱 →  $A.^n$
  - 스칼라.^행렬 → 스칼라를 행렬 각 원소별로 거듭제곱

```
>> s1*s2
```

다음 사용 중 오류가 발생함: \*

행렬 곱셈의 차원이 잘못되었습니다. 첫 번째 행렬의 열 개수가 두 번째 행렬의 행 개수와 일치하는지 확인하십시오. 요소별 곱셈 연산을 수행하려면 `.*`를 사용하십시오.

```
>> s1.*s2
```

ans =

1	2	3	4
2	4	6	8
3	6	9	12

```
>>
```

```
>> s1^2
```

다음 사용 중 오류가 발생함: ^ (line 51)

차원이 정확하지 않아 행렬을 거듭제곱할 수 없습니다. 행렬이 정사각 행렬이고 지수 값이 스칼라인지 확인하십시오. 요소별 행렬 거듭제곱 연산을 수행하려면 `.^`을 사용하십시오.

```
>> s1.^2
```

ans =

1	1	1	1
4	4	4	4
9	9	9	9

```
>> s1
```

s1 =

1	1	1	1
2	2	2	2
3	3	3	3

```
>> s2
```

s2 =

1	2	3	4
1	2	3	4
1	2	3	4

```
>> 2.^[1 2 ; 3 4]
```

ans =

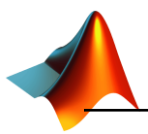
2	4
8	16





# 행렬의 연산 - 나누기 (역행렬)

	덧셈 항등원	곱셈 항등원	곱셈 역원
정의	더해도 값이 그대로인 수 $a + e = e + a = a$	곱해도 값이 그대로인 수 $a \times e = e \times a = a$	곱해서 곱셈 항등원이 나오는 수 $a \times x = x \times a = e$
실수	$e = 0$	$e = 1$	$x = \frac{1}{a} \ (a \neq 0)$
행렬	$O = \text{영행렬}$ $O = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ $O = \text{zeros}(N);$	$E = \text{단위행렬}$ $E = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ $E = \text{eye}(N);$ ※ $E$ 는 square matrix	$x = \text{역행렬}$ $x = a^{-1}$ $x = \text{inv}(a)$ ※ square matrix에 대해서만 존재 ※ $a$ 가 invertible할 때만 존재



# 행렬의 연산 - 나누기 (역행렬)

```
>> a = magic(3);  
>> b = inv(a)
```

b =

0.1472	-0.1444	0.0639
-0.0611	0.0222	0.1056
-0.0194	0.1889	-0.1028

```
>> a*b
```

ans =

1.0000	0	-0.0000
-0.0000	1.0000	0
0.0000	0	1.0000

```
>> b*a
```

ans =

1.0000	0	-0.0000
0	1.0000	0
0	0.0000	1.0000

a =

1	4	7	10
2	5	8	11
3	6	9	12

```
>> inv(a)
```

다음 사용 중 오류가 발생함: inv  
행렬은 정사각 행렬이어야 합니다.

a =

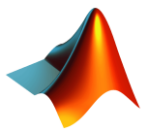
1	2	3
2	4	6
7	8	9

```
>> inv(a)
```

경고: 행렬이 설정된 작업 정밀도에서 특이 행렬입니다.

ans =

Inf	Inf	Inf
Inf	Inf	Inf
Inf	Inf	Inf



# 행렬의 연산 - 나누기 (역행렬)

$$\begin{cases} x + 2y - 4z = 5 \\ 2x + y - 6z = 8 \\ 4x - y - 12z = 13 \end{cases} \quad \begin{bmatrix} 1 & 2 & -4 \\ 2 & 1 & -6 \\ 4 & -1 & -12 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 5 \\ 8 \\ 13 \end{bmatrix}$$

$$A \cdot x = b$$

$$A^{-1} \cdot A \cdot x = A^{-1} \cdot b$$

$$E \cdot x = A^{-1} \cdot b$$

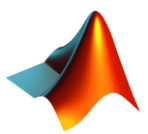
$$x = A^{-1} \cdot b$$

```
>> a
a =
     1     2    -4
     2     1    -6
     4    -1   -12

>> b
b =
     5
     8
    13

>> x = inv(a)*b
x =
    5.0000
    1.0000
    0.5000

>> a*x
ans =
     5
     8
    13
```



# 행렬의 연산 – 나누기 (원소별 나눗셈)

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

/ / /

$$b = \begin{bmatrix} 4 & 3 & 1 \end{bmatrix}$$

|| || ||

$$c = \begin{bmatrix} 1 & 2 \\ 4 & 3 \end{bmatrix}$$

```
>> a = [1 2 3];
```

```
>> b = [4 3 1];
```

```
>> a/b
```

```
ans =
```

```
0.5000 ??
```

```
>> a./b
```

```
ans =
```

```
0.2500 0.6667 3.0000
```

OK

- 원소별 곱셈과 동일
- / 대신 ./ 사용
- 두 행렬의 크기는 같아야 함

```
>> s1
```

```
s1 =
```

```
1 1 1 1
2 2 2 2
3 3 3 3
```

```
>> s2
```

```
s2 =
```

```
1 2 3 4
1 2 3 4
1 2 3 4
```

```
>> s1/s2
```

경고: 랭크 부족, rank = 1, tol = 4.864754e-15.

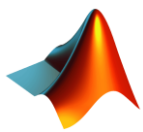
```
ans =
```

```
0.3333 0 0
0.6667 0 0
1.0000 0 0
```

```
>> s1./s2
```

```
ans =
```

```
1.0000 0.5000 0.3333 0.2500
2.0000 1.0000 0.6667 0.5000
3.0000 1.5000 1.0000 0.7500
```



# 행렬의 연산 – 나누기 (원소별 나눗셈)

- 행렬의 원소별 나눗셈 → 연산자 앞 점(.)을 찍음
  - 두 행렬을 원소별 나눗셈 →  $A./B$
  - 행렬의 각 원소의 역수 →  $1./A$

※ 실습: “복잡한 수식”을 원소별 연산으로

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \text{ and } B = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix}$$

$$A ./ B = \begin{bmatrix} A_{11}/B_{11} & A_{12}/B_{12} & A_{13}/B_{13} \\ A_{21}/B_{21} & A_{22}/B_{22} & A_{23}/B_{23} \\ A_{31}/B_{31} & A_{32}/B_{32} & A_{33}/B_{33} \end{bmatrix}$$

```
>> s = magic(3)
s =
     8     1     6
     3     5     7
     4     9     2

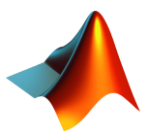
>> 1./s
ans =
    0.1250    1.0000    0.1667
    0.3333    0.2000    0.1429
    0.2500    0.1111    0.5000

>> 10./s
ans =
    1.2500   10.0000    1.6667
    3.3333    2.0000    1.4286
    2.5000    1.1111    5.0000
```



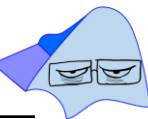
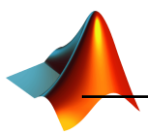
# 행렬을 다루는 내장함수들

함수명	동작	예시	참고
min [i, v] = min(A)	행렬의 최소값 [i, v] → 최소값의 위치도 반환	>> min([5 3 1 2 4]) ans = 1	2차원일 경우 각 열의 최소값을 1행으로 반환
max [i, v] = max(A)	행렬의 최대값 [i, v] → 최대값의 위치도 반환	>> max([5 3 1 2 4]) ans = 5	2차원일 경우 각 열의 최대값을 1행으로 반환
size	행렬의 크기	>> size(magic(5)) ans = 5 5	N차원 행렬일 경우 길이가 N인 벡터 반환
length	행렬의 길이	>> length(magic(5)) ans = 5	size의 결과 중 최대값과 같음
numel	모든 원소의 개수	>> numel(magic(5)) ans = 25	size의 결과 원소를 모두 곱한 것과 같음



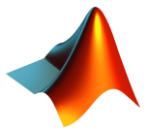
# 행렬을 다루는 내장함수들

함수명	동작	예시	참고
' transpose	전치행렬	>> (1:2)' ans = 1 2	정확히는 conjugate transpose (ctranspose) non-conjugate는 .'
flipud	행렬을 위아래로 뒤집음	>> flipud([1 2; 3 4]) ans = 3 4 1 2	
fliplr	행렬을 좌우로 뒤집음	>> fliplr([1 2; 3 4]) ans = 2 1 4 3	
reshape	행렬의 원소를 재배열	>> reshape(1:6,2,3) ans = 1 3 5 2 4 6	reshape(a,m,n) a의 원소 개수가 $m \times n$ 과 같아야 함
repmat	행렬을 복제하여 이어붙임	>> repmat(1:3,2,2) ans = 1 2 3 1 2 3 1 2 3 1 2 3	



# 행렬을 다루는 내장함수들

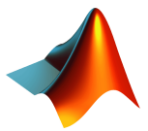
함수명	동작	예시	참고
ones	모든 원소가 1인 행렬 ones(m, n) → m×n 크기의 행렬 ones(n) → n×n 크기의 행렬	>> ones(1,5) ans = 1 1 1 1 1	모든 원소가 100인 행렬 100*ones(m,n)
zeros	모든 원소가 0인 행렬 zeros(m, n) → m×n 크기의 행렬 zeros(n) → n×n 크기의 행렬	>> zeros(1,4) ans = 0 0 0 0	
norm	벡터 노름(norm) $\ x\ _2 = \left( \sum_{i=1}^N  x_i ^2 \right)^{1/2} = \sqrt{x_1^2 + x_2^2 + \dots + x_N^2}$	>> norm([3,4]) ans = 5	
find [i, j] = find(a)	행렬의 0이 아닌 원소의 위치 반환 [i, j] = find(a) → 0이 아닌 원소의 위치를 2차원 인덱스로 반환	>> find([1 2 0 0 2]) ans = 1 2 5	
sort [i, v] = sort(a)	행렬을 오름차순으로 정렬 2차원 행렬 → 각 열을 정렬 [i, v] = sort(A) → 정렬 후 인덱스 변화도 반환	>> sort([1 3 5 4 2]) ans = 1 2 3 4 5	'descend' 옵션 사용 시 내림차순 정렬





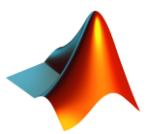
# 행렬을 다루는 내장함수들

함수명	동작	예시	참고
mean mean2	mean(A) 또는 mean(A,1) → 행방향 평균 mean(A,2) → 열방향 평균 mean2(A) → 2차원 평균	>> mean([1 3 5 ; 3 5 7]) ans = 2 4 6	
std std2	std(A) 또는 std(A,1) → 행방향 표준편차 std(A,2) → 열방향 표준편차 std2(A) → 2차원 표준편차	>> std(magic(4)) ans = 5.4467 5.1962 5.1962 5.4467	
sum	sum(A) 또는 sum(A,1) → 행방향 합 sum(A,2) → 열방향 합	>> sum([1 3 5 ; 3 5 7]) ans = 4 8 12	
cumsum	벡터 → 누적합 행렬 → 각 열의 누적합	>> cumsum([1 2 3 4 5]) ans = 1 3 6 10 15	
median	중간값 짝수개일 경우 두 중간값의 평균	>> median([1 3 5 7]) ans = 4	



# 행렬을 다루는 내장함수들

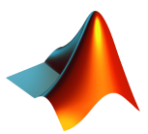
함수명	동작	예시	참고
rand	0~1 범위의 균등분포 난수 rand(m, n) → m×n 크기의 행렬 rand(n) → n×n 크기의 행렬	>> rand(2,4) ans = 0.8147 0.1270 0.6324 0.2785 0.9058 0.9134 0.0975 0.5469	hist로 분포 확인 가능
randn	평균 0, 표준편차 1인 정규분포 난수 randn(m, n) → m×n 크기의 행렬 randn(n) → n×n 크기의 행렬	>> randn(2,3) ans = 0.3480 -0.6357 -0.7615 -0.4551 -0.9799 -1.2835	hist로 분포 확인 가능
randi	균등 분포의 정수 난수 값 범위, 크기 설정 가능	>> randi([1,50],[2,3]) ans = 21 42 5 8 37 42	도움말 확인



# 행렬을 입력으로 받는 내장함수

- 여러 내장함수들 (sqrt, exp, log, sin, round, ...)
  - $M \times N$  행렬을 넣으면
    - 알아서 각 원소를 연산하여  $M \times N$  행렬을 반환

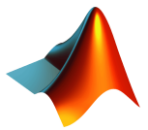
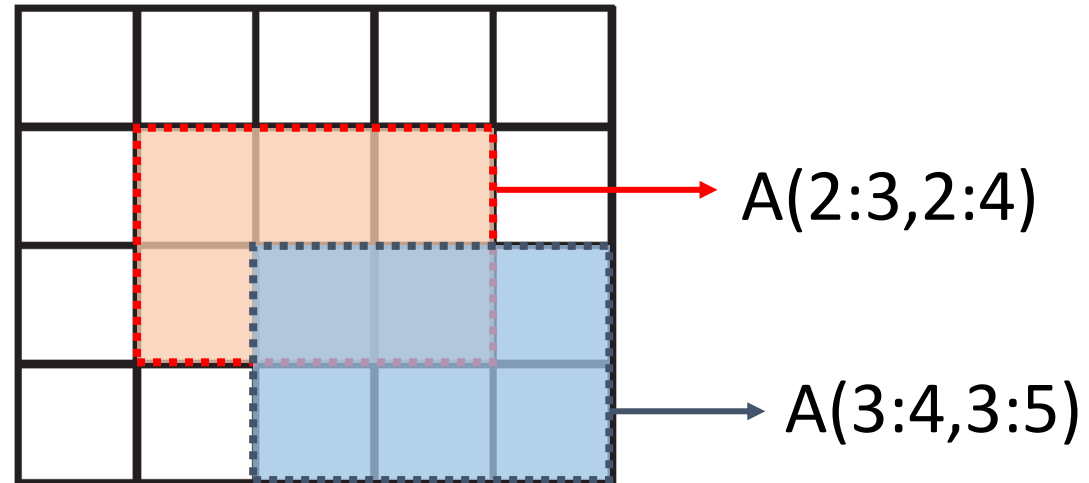
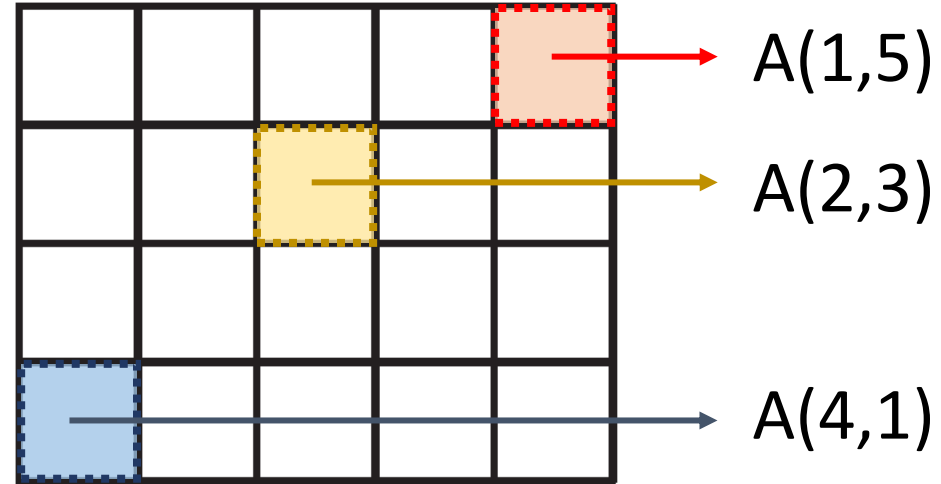
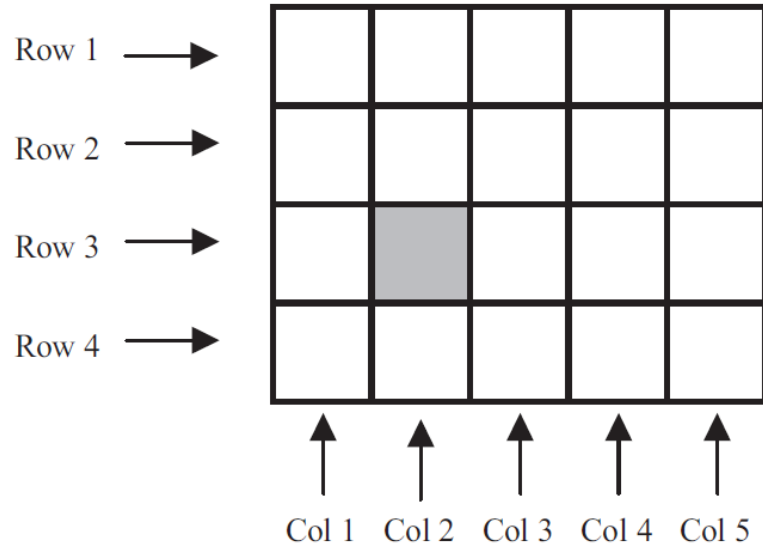
```
>> sqrt([1 4 9 16])
ans =
     1     2     3     4
>> sind(90*(0:4))
ans =
     0     1     0    -1     0
>> round(pi*(1:5))
ans =
     3     6     9    13    16
```



# 행렬의 인덱싱 (indexing)

변수명(행범위, 열범위)

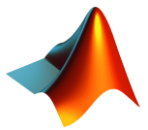
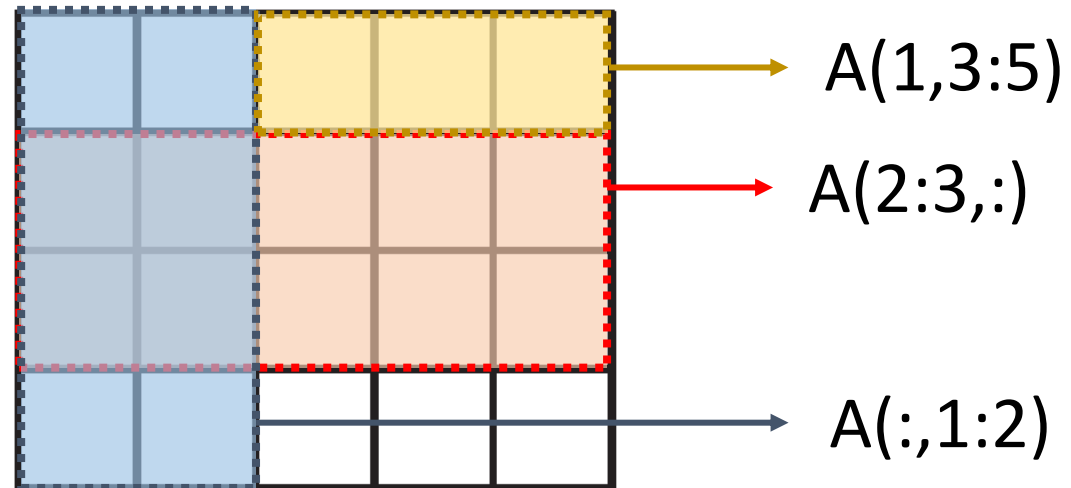
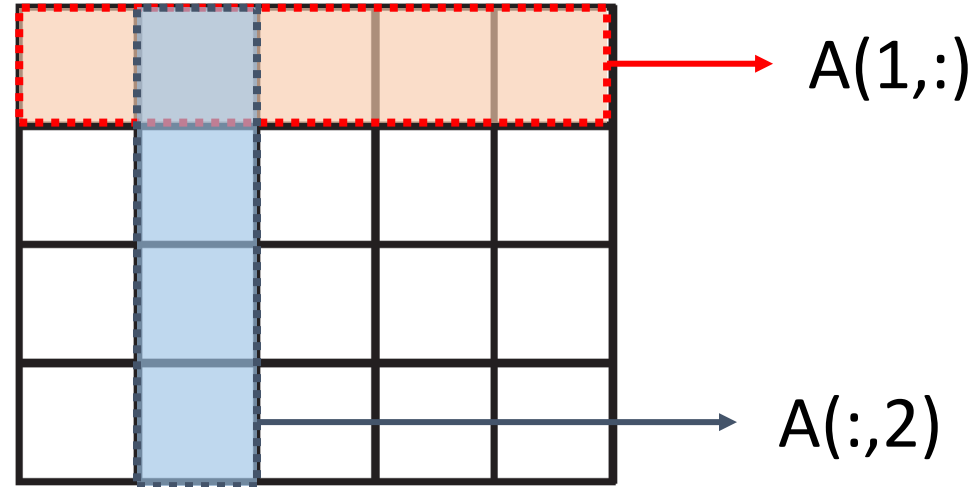
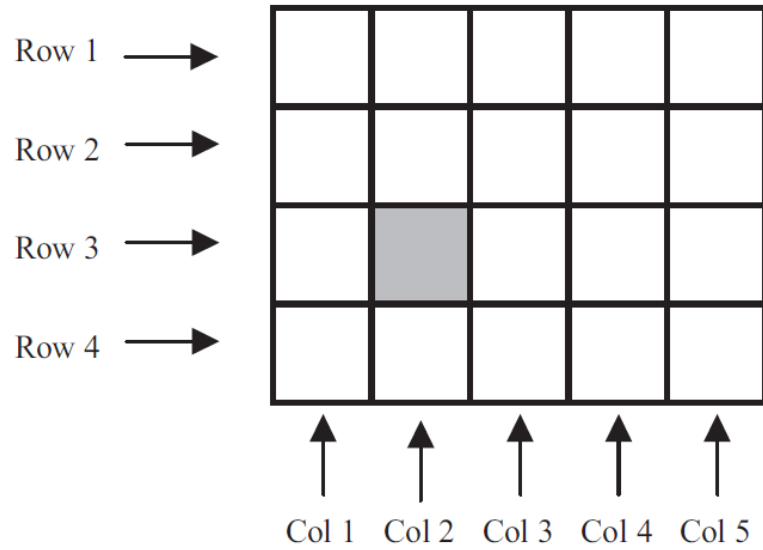
A: 4×5 행렬



# 행렬의 인덱싱 (indexing)

변수명(행범위, 열범위)

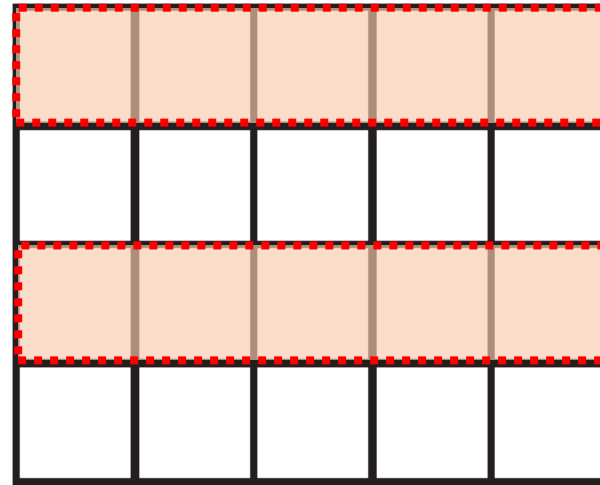
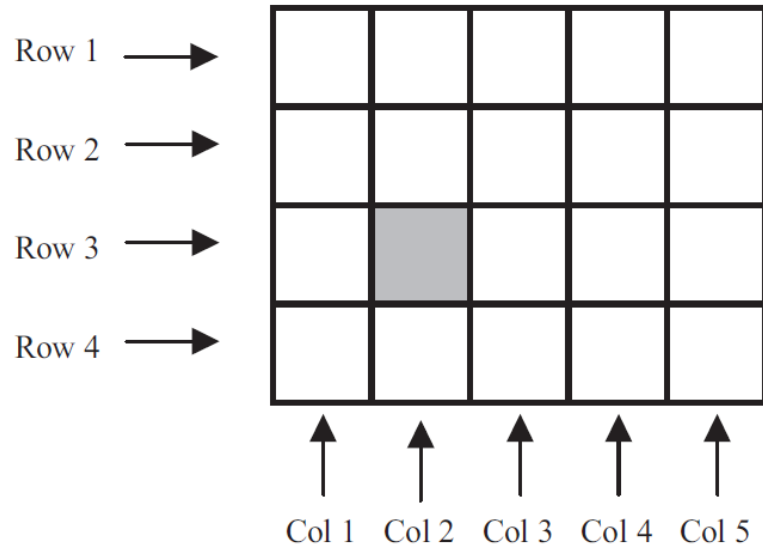
A: 4×5 행렬



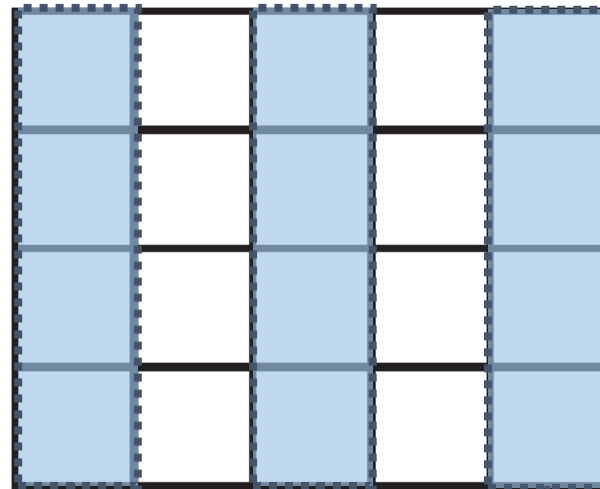
# 행렬의 인덱싱 (indexing)

변수명(행범위, 열범위)

A: 4×5 행렬



$A([1,3],:)$



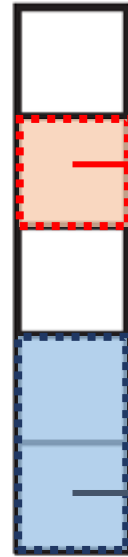
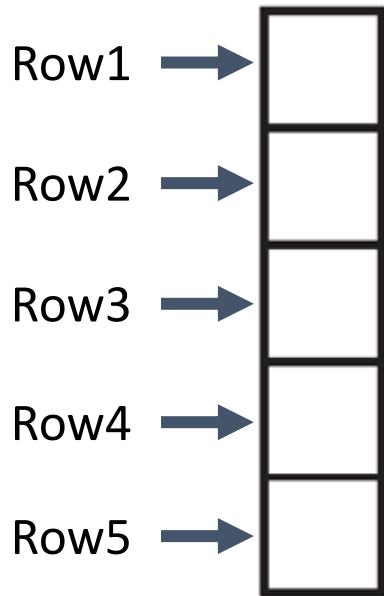
tip1.  $A(:,1:2:5)$ 도 가능  
tip2.  $A(:,[5\ 3\ 1])=?$

$A(:,[1,3,5])$



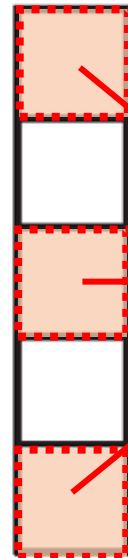
# 행렬의 인덱싱 (indexing) – 벡터의 경우

B:  $5 \times 1$  행렬



$B(2,1)$  또는  $B(2,:)$  또는  $B(2)$

$B(4:5,1)$  또는  $B(4:5,:)$  또는  $B(4:5)$   
또는  $B([4,5])$



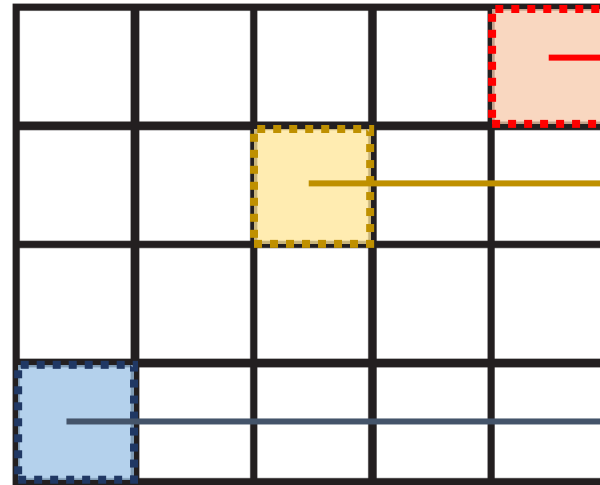
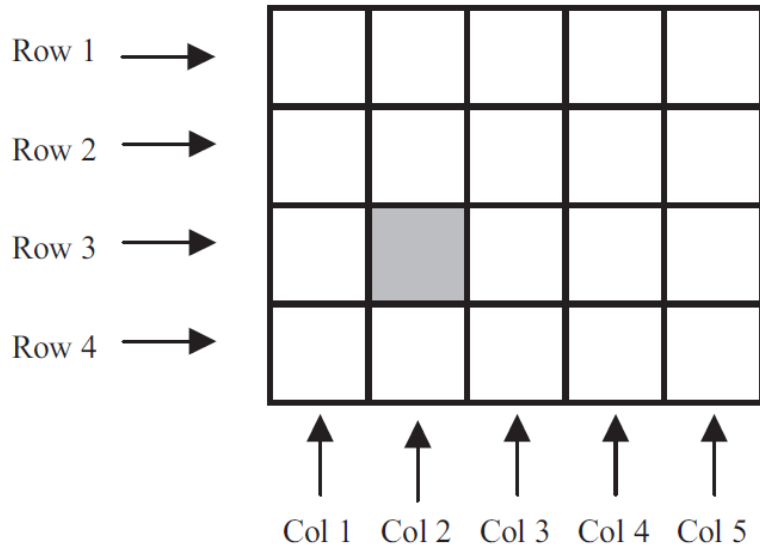
$B([1,3,5],1)$  또는  $B(1:2:5,:)$   
또는  $B([1,3,5])$

tip.  $B([5\ 3\ 1])=?$  (행렬의 reorder)



# 행렬의 인덱싱 (indexing) – end의 활용

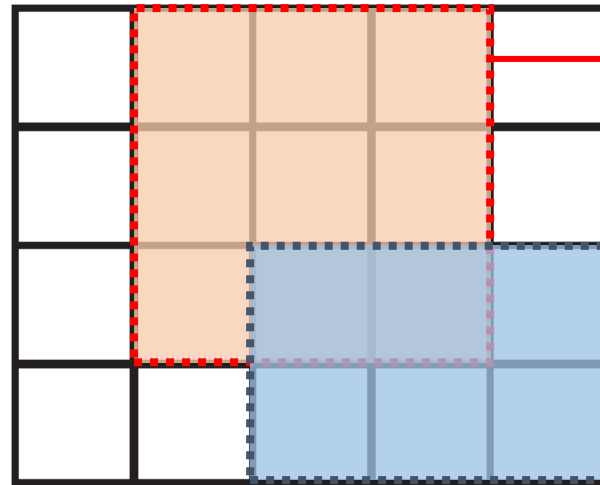
A: 4×5 행렬



$A(1,5)$  또는  $A(1,\text{end})$

$A(2,3)$  또는  $A(2, \text{end}-2)$

$A(4,1)$  또는  $A(\text{end},1)$



$A(1:3,2:4)$   
또는  $A(1:3, 2:\text{end}-1)$

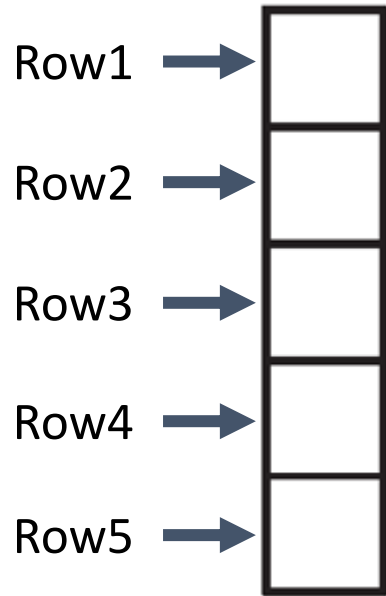
$A(3:4,3:5)$   
또는  $A(3:\text{end}, 3:\text{end})$



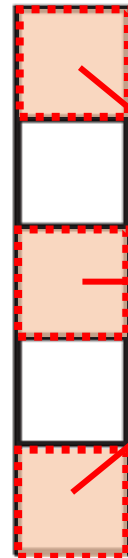


# 행렬의 인덱싱 (indexing) – end의 활용

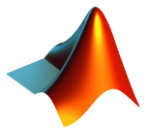
B:  $5 \times 1$  행렬



B(4:5) 또는 B(4:end)



B(1:2:5) 또는 B([1,3,end])



# 행렬의 인덱싱 (indexing) – (:)의 활용

```
>> a
a =
     1     5     9
     2     6    10
     3     7    11
     4     8    12
```

```
>> a(:)
ans =
```

```
1
2
3
4
5
6
7
8
9
10
11
12
```

1	2	3
4	5	6
7	8	9
10	11	12

a

Arrangement  
in Computer  
Memory

.	
.	
.	
1	a(1,1)
4	a(2,1)
7	a(3,1)
10	a(4,1)
2	a(1,2)
5	a(2,2)
8	a(3,2)
11	a(4,2)
3	a(1,3)
6	a(2,3)
9	a(3,3)
12	a(4,3)
.	
.	
.	



# 행렬을 수정하는 방법

```
>> a = reshape(1:20,4,5)
```

```
a =
```

1	5	9	13	17
2	6	10	14	18
3	7	11	15	19
4	8	12	16	20

```
>> a(2,3) = 100
```

```
a =
```

1	5	9	13	17
2	6	100	14	18
3	7	11	15	19
4	8	12	16	20

```
>> a(:,3) = [100 101 102 103]'
```

```
a =
```

1	5	100	13	17
2	6	101	14	18
3	7	102	15	19
4	8	103	16	20

```
>> a(:,3:4) = -1
```

```
a =
```

1	5	-1	-1	17
2	6	-1	-1	18
3	7	-1	-1	19
4	8	-1	-1	20

```
>> a(3:end,:) = 42
```

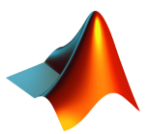
```
a =
```

1	5	-1	-1	17
2	6	-1	-1	18
42	42	42	42	42
42	42	42	42	42

```
>> a(3:4,1:2) = [10 11 ; 12 13]
```

```
a =
```

1	5	-1	-1	17
2	6	-1	-1	18
10	11	42	42	42
12	13	42	42	42



# 행렬을 수정하는 방법

```
>> a = [a ones(4,1)]
```

```
a =  
     1     5    -1    -1    17     1  
     2     6    -1    -1    18     1  
    10    11    42    42    42     1  
    12    13    42    42    42     1
```

```
>> a = [a 2*ones(size(a,1),1)]
```

```
a =  
     1     5    -1    -1    17     1     2  
     2     6    -1    -1    18     1     2  
    10    11    42    42    42     1     2  
    12    13    42    42    42     1     2
```

```
>> a = [a; 5*(1:size(a,2))]
```

```
a =  
     1     5    -1    -1    17     1     2  
     2     6    -1    -1    18     1     2  
    10    11    42    42    42     1     2  
    12    13    42    42    42     1     2  
     5    10    15    20    25    30    35
```

```
>> a = a(1:end-1, 1:end-2)
```

```
a =  
     1     5    -1    -1    17  
     2     6    -1    -1    18  
    10    11    42    42    42  
    12    13    42    42    42
```

```
>> a(end,:) = []
```

```
a =  
     1     5    -1    -1    17  
     2     6    -1    -1    18  
    10    11    42    42    42
```

```
>> a(:,4:end) = []
```

```
a =  
     1     5    -1  
     2     6    -1  
    10    11    42
```

```
>> a(5,5) = -100
```

```
a =  
     1     5    -1     0     0  
     2     6    -1     0     0  
    10    11    42     0     0  
     0     0     0     0     0  
     0     0     0     0   -100
```

