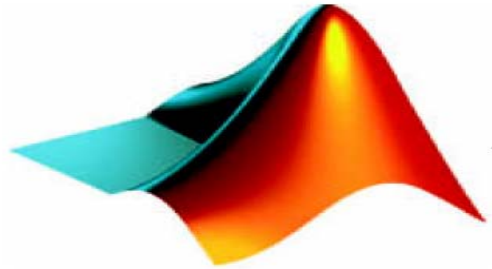

1. 배열과 행렬

출처 : 이 준 엽 교수 (이화여대)

연산은 여러 수학의 기본이 된다. 가장 기본적인 연산은 하나의 값만을 가진 스칼라(Scalar)를 대상으로 하고, 연산의 결과 값은 스칼라들의 결합으로 표현된다. 예를 들어 자연수에 대한 곱셈 연산의 경우, 두 자연수 3과 5를 결합한 곱셈 연산의 결과 값은 3*5로 표현된다. 그러나 많은 경우 하나 이상의 수에 대해서 같은 연산이 반복적으로 동시에 행해질 수 있다. 예를 들어, 1월부터 12월까지 수입을 표시하는 12개의 수(a_1, ... , a_{12})와 지출을 표시하는 12개의 수(b_1, ... , b_{12})가 있다고 하자. 매월별 수지동향(c_i) = 수입(a_i) − 지출(b_i)과 같은 계산을 12번 반복하는 대신, 이들 12개씩의 수를 가진 배열 a, b, c를 이용할 수도 있다. 그러면, 위 계산을 c=a−b와 같이 간단히 표현할 수 있을 것이다. 행렬(Matrix)은 이러한 개념을 확장한 것으로, m, n≥1에 대하여, m개의 행과 n개의 열 속에 mn개의 수를 다음과 같이 배열한 것이다.

$$A = \begin{bmatrix} a_{11}, & a_{12}, & a_{13}, & \cdots, & a_{1n} \\ a_{21}, & a_{22}, & a_{23}, & \cdots, & a_{2n} \\ \cdots, & \cdots, & \cdots, & \ddots, & a_{1n} \\ a_{m1}, & a_{m2}, & a_{m3}, & \cdots, & a_{mn} \end{bmatrix} \quad \text{혹은} \quad (a_{ij}), \quad i=1,\cdots,m, \quad j=1,\cdots,n$$

MATLAB®은 순수과학과 공학적 수치계산 및 시각화를 위하여, 행렬에 기초한 시스템이다. MATLAB에서는 Fortran이나 C 같은 프로그래밍 언어에서 요구한 시간의 몇 분의 일만으로도 복잡한 수치적 문제를 풀 수 있다. MATLAB이라는 이름은 MATrix LABoratory에서 가져온 것이다. 이 이름에서 알 수 있듯이 MATLAB의 가장 기본이 되는 대상은 복소수를 원소로 가지는 행렬이다. 지금부터 MATLAB을 이용하여 행렬을 표시하고, 이용하는 방법에 대하여 살펴보자.

1.1 행렬 만들기

행렬 중 하나의 행과 여러 개의 열을 가지는 경우를 행벡터(row vector)라고 말하고 여러 개의 행과 하나의 열을 가지는 경우를 열벡터(column vector)라고 말한다. 실수는 1행 1열을 가진 행렬임으로, 행벡터이자 열벡터라고 할 수 있다. MATLAB에서는 행벡터를 표현하고자 할 경우는 대괄호([ ]) 안에 원소들 사이에 쉼표(comma)나 빈칸(space)을 사용하면 된다. MATLAB에서는 일정한 값만큼 증감하는 등차수열을 원소로 하는 배열은 대괄호 없이 초기값:증감값:마지막값 형태로 표시할 수도 있다. 이때 증감값이 1인 경우는 초기값:마지막값의 형태로 간단히 쓸 수도 있다.

예제 1.1.1 (행벡터 만들기)

```
>> a=[1 2 3 4 5]                    %  a는 1에서 5까지
a =
     1     2     3     4     5
>> b=1:2:9                          %  b는 1에서 2씩 증가시켜 9까지
b =
     1     3     5     7     9

>> c=[b  a]
c =
     1     3     5     7     9     1     2     3     4     5

>> d=[a(1:2:5) 1 0 1]
d =
     1     3     5     1     0     1
```

빈칸은 배열 혹은 행벡터의 원소를 구분하는데 이용되므로, 복소수를 표시하기 위한 i 혹은 j를 쓸 때에는 빈칸을 없애거나 괄호(parentheses)를 이용하여야 하다.

예제 1.1.2 (복소수 사용하기)

```
>> [1 -2i 3 4 5+6i]              % contains 5 elements
ans =
  1.0000         0 - 2.0000i      3.0000        4.0000      5.0000 + 6.0000i

>> [ (1 -2i) 3 4 5+6i]           % contains 4 elements
ans =
  1.0000 - 2.0000i   3.0000      4.0000        5.0000 + 6.0000i

>> [ 1-2i 3 4 5+6i]              % contains 4 elements
ans =
  1.0000 - 2.0000i   3.0000      4.0000        5.0000 + 6.0000i
```

한편, 열벡터(column vector)로 나타내려면 대괄호(bracket) 안에 값들 사이에 세미콜론(semicolon)을 사용하면 된다. 이때, Return이나 Enter key를 입력하면 세미콜론과 마찬가지로 새로운 행을 입력할 수 있다.

예제 1.1.3 (열벡터 만들기)

```
>> c=[1;2;3;4;5]
c =
     1
     2
     3
     4
     5
```

행렬 $A=\begin{bmatrix} a_{11}, & a_{12}, & a_{13}, & \cdots, & a_{1n} \\ a_{21}, & a_{22}, & a_{23}, & \cdots, & a_{2n} \\ \cdots, & \cdots, & \cdots, & \cdots, & \vdots \\ a_{m1}, & a_{m2}, & a_{m3}, & \cdots, & a_{mn} \end{bmatrix}$ 은 $A^1=\begin{bmatrix} a_{11} \\ a_{21} \\ \cdots \\ a_{m1} \end{bmatrix}$ 에서부터 $A^n=\begin{bmatrix} a_{1n} \\ a_{2n} \\ \cdots \\ a_{mn} \end{bmatrix}$ 까지 n개의 열벡터 혹은 $A_1=[a_{11}, a_{12}, \cdots, a_{1n}]$ 에서부터 $A_m=[a_{m1}, a_{m2}, \cdots, a_{mn}]$ 까지 m개의 행벡터로 구성되어 있다고 생각할 수 있다. 배열(array)이라고도 불리는 행렬은 행벡터와 열벡터의 조합 뿐 만 아니라 이미 만들어진 다른 행렬들의 조합으로도 만들 수 있다. 배열 안에서 행 안의 원소들은 빈칸( ) 혹은 쉼표(,)로 구별하고, 각 행의 구별은 세미콜론(;)이나 줄 바꿈(Return, Enter)으로 한다.

예제 1.1.4 (행렬만들기)

```
>> g=[1 2 3 4; 5 6 7 8]                % g는 2 by 4 matrix
g =
     1    2    3    4
     5    6    7    8

>> g=[g, g; 9:12, 9:12]                % g는 3행 8열로 확장됨
g =
     1    2    3    4    1    2    3    4
     5    6    7    8    5    6    7    8
     9   10   11   12    9   10   11   12

>> h=[1  2  3; 4  5  6  7]   % 모든 row가 같은 수의 column을 가져야만 한다.
??? All rows in the bracketed expression must have the same number of
column
```

x의 값들을 넣는 방법으로는 x의 원소의 개수가 적을 경우 직접 타이핑하지만, 수가 너무 많을 경우 콜론(:) 기호와 linspace라는 MATLAB 함수를 이용할 수 있다. 다수의 값을 콜론(:)이나 linspace 함수를 이용하여 부여하는 일은 샘플링이라는 작업에서 흔히 발생한다. $y=\sin(x)$, $0 \le x \le \pi$를 계산하는 문제가 있다고 하자. 이 범위내의 모든 점에서 계산하는 것은 불가능하므로(왜냐하면 무한개의 점이 있으니까) 유한개의 점들을 선택해야 한다. 이러한 과정을 함수를 샘플링(sampling)한다고 말한다. 여기서는 이 범위내의 0.1π마다, 즉, $x = 0, 0.1\pi, 0.2\pi, \cdots, 1.0\pi$에서 sin(x)의 값을 계산해보도록 한다. 이 경우, 배열을 이용하여 x와 그에 대한 y 값을 순서대로 나타낼 수 있다.

x	0	0.1π	0.2π	0.3π	0.4π	0.5π	0.6π	0.7π	0.8π	0.9π	π
y	0	0.31	0.59	0.81	0.95	1.0	0.95	0.81	0.59	0.31	0

위와 같은 배열을 MATLAB에서 만드는 법은 아래 예제와 같다. 아래 예제에서처럼 x가 11개의 원소를 가진 경우, MATLAB에서 sin 함수는 역시 11개의 열을 가진 배열로 결과를 되돌려 준다. 이 때, x는 1행 11열 행렬 또는 길이가 11인 행벡터로 불린다.

예제 1.1.5 함수의 샘플링(Sampling)

```
>> x=[0  .1*pi  .2*pi  .3*pi  .4*pi  .5*pi  .6*pi  .7*pi  .8*pi  .9*pi  pi]
x =
  Column 1 through 7
         0    0.3142    0.6283    0.9425    1.2566    1.5708    1.8850
  Column 8 through 11
    2.1991    2.5133    2.8274    3.1416

>> y=sin(x)
y =
   Column 1 through 7
          0    0.3090    0.5878    0.8090    0.9511    1.0000    0.9511
   Column 8 through 11
    0.8090    0.5878    0.3090    0.0000

>> plot(x,y)                    % y = sin(x)의 그래프
```

콜론(:)을 이용하여 등차수열을 원소로 가지는 배열을 만들거나, 초기 값과 말기 값을 일정한 간격으로 나누어주는 작업은 linspace라는 MATLAB 함수를 이용하면 보다 용이하게 할 수 있다. 이와 비슷한 기능을 가진 logspace는 로그적 등간격으로 나누어진 배열(등비수열)을 만들어 준다. linspace와 logspace의 기본 형태는

linspace(first_value, last_value, number_of_values).
logspace(first_exponent, last_exponent, number_of_values)

이고 보다 자세한 설명은

help linspace, help logspace

를 참고하면 된다.

```
>> x=(0:0.1:1)*pi                  % 0에서 0.1*pi씩 증가하여 1*pi까지 11개의 원소
x =
  Column 1 through 7
         0    0.3142    0.6283    0.9425    1.2566    1.5708    1.8850
  Column 8 through 11
    2.1991    2.5133    2.8274    3.1416

>> x=linspace(0,pi,11)             % 0에서 pi까지 11개의 원소를 표현
x =
  Column 1 through 7
         0    0.3142    0.6283    0.9425    1.2566    1.5708    1.8850
  Column 8 through 11
    2.1991    2.5133    2.8274    3.1416

>> logspace(0,2,11)          % 10^0부터 10^2까지 11개의 점이 포함되도록 만듦
ans =
  Column 1 through 7
    1.0000    1.5849    2.5119    3.9811    6.3096   10.0000   15.8489
  Column 8 through 11
   25.1189   39.8107   63.0957  100.0000
```

간단한 배열 만들기

x=[2 2*pi sqrt(2) 2-3j]	4개의 원소를 가진 행벡터 (2-3j 사이의 빈칸 주의)
x=first:last	first부터 last까지 1씩 증가하는 수들을 원소로 가지는 행벡터
x=first:increment:last	first부터 last까지 increment씩 증가하는 수들을 원소로 가지는 행벡터
x=linspace(first,last,n)	first부터 last까지 균일한 간격으로 증가하는 n개의 수들을 원소로 가지는 행벡터
x=logspace(first,last,n)	10^first부터 10^last까지 지수적으로 균일한 간격의 n개의 수들을 원소로 가지는 행벡터

1.2 행렬의 크기와 특수 행렬들

예제 1.2.1 (저장된 변수명과 변수의 크기와 길이)

```
>> whos
     Name      Size    Elements     Bytes     Density     Complex
     a         1 by 5          5        40        Full          No
     b         5 by 1          5        40        Full          No
     c         1 by 5          5        40        Full          No
     d         1 by 5          5        80        Full         Yes
     e         5 by 1          5        80        Full         Yes
     f         5 by 1          5        80        Full         Yes
     g         3 by 4         12        96        Full          No
Grand total is 42 elements using 456 bytes
leaving 2316084 bytes of memory free.

>> A=[1  2  3  4; 5  6  7  8]
A =
     1     2     3     4
     5     6     7     8

>> s=size(A)                    % size 함수는 행의 개수와 열의 개수를 알려 줌
s =
      2      4

>> [r,c]=size(A)
r =
      2
c =
      4

>> r=size(A,1),  c=size(A,2)   % 두 개이상의 명령어를 쓰려면 , 혹은 ;를 사용
r =
      2
c =
      4

>> length(A)      % length 함수는 행의 개수와 열의 개수 중 더 큰 값을 준다
ans =
      4

>> size([])
ans =
      0      0
```

예제 1.2.2 (특수행렬 만들기)

```
>> zeros(2)                % a 2-by-2 matrix of zeros
ans =
     0     0
     0     0

>> ones(2,4)*pi            % a 2-by-4 matrix of ones * pi
ans =
    3.1416    3.1416    3.1416    3.1416
    3.1416    3.1416    3.1416    3.1416

>> A=[1 2 3; 4 5 6];       % 명령문 마지막에 ;를 사용하면 결과가 출력 안됨
>> ones(size(A))
ans =
     1     1     1
     1     1     1

>> eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1
>> eye(3,2)
ans =
     1     0
     0     1
     0     0
```

```
>> rand(3,1)               % 0과 1 사이의 균일하게 분포된 난수들의 3 by 1 행렬
ans =
    0.2190
    0.0470
    0.6789

>> randn(2)                % 평균(mean)이 0이고 표준편차(variance)가 1인
                           % 정규 분포를 가진 수들의 2 by 2 행렬
ans =
    1.1650    0.0751
    0.6268    0.3516

>> a=[1 2 3;4 5 6];
>> b=find(a>10)
b =
     [ ]
```

1.3 행렬의 방향과 전치행렬

[초기값:증감값:말기값]에서와 같이 콜론(colon)을 사용하거나 linspaces나 logspace 같은 함수를 사용하면 행벡터를 쉽게 만들 수 있다. 이러한 행벡터를 열벡터를 바꾸려면 MATLAB 전치연산자(')를 이용하면 된다.

예제 1.3.1 (열벡터 만들기)

```
>> a=1:5
a =
     1     2     3     4     5

>> b=a'
b =
     1
     2
     3
     4
     5

>> c=b'
c =
     1     2     3     4     5
```

$A=(a_{ij})$를 m-by-n 행렬이라 할 때, 이를 n-by-m 행렬로 변환시켜 주는 연산자로는 복소 전치행렬 연산자와 (실수) 전치행렬 연산자가 있다. 이들 두 연산 결과 행렬은 $A^H := (\overline{a_{ji}})$, $A^T = (a_{ji})$라고 쓰고, 각각을 A의 Hermitian, A의 Transpose이라고 부른다. (경우에 따라서는 복소 전치행렬도 A의 Transpose라고도 부른다.) 배열이 복소수일 경우 MATLAB에서 전치행렬 연산자(')는 복소 전치행렬 연산자를 의미하며, 원소의 위치뿐만 아니라 원소의 복소수 부분의 부호도 반대로 바뀐다. 만약 복소수 배열에 실수 전치 행렬 연산자를 적용하려면 실수 전치행렬 연산자(.')를 사용하면 된다.

예제 1.3.3 (대칭행렬과 반대칭행렬)

```
>> a = rand(3), S = (a+a')/2 , K = (a-a')/2, sum = S+K
a =
    0.7564    0.2470    0.7534
    0.9910    0.9826    0.6515
    0.3653    0.7227    0.0727
S =
    0.7564    0.6190    0.5593
    0.6190    0.9826    0.6871
    0.5593    0.6871    0.0727
K =
         0   -0.3720    0.1940
    0.3720         0   -0.0356
   -0.1940    0.0356         0
sum =
    0.7564    0.2470    0.7534
    0.9910    0.9826    0.6515
    0.3653    0.7227    0.0727
```

행렬 중 행의 수와 열의 수가 n으로 같은 n-by-n 행렬을 n차 정방행렬이라 부른다. 전치행렬과 자기 자신이 동일한 행렬을 에르미트(Hermitian)행렬이라 하고, 특히 행렬이 실수인 경우 대칭(Symmetric)행렬이라고도 한다. 전치행렬과 자신의 합이 영인 행렬을 반에르미트(Skew-Hermitian) 행렬 또는 반대칭(Skew-Symmetric) 행렬(혹은 교대행렬)이라 한다. 임의의 행렬 A는 에르미트 행렬과 반에르미트 행렬 혹은 대칭행렬과 반대칭행렬의 합으로 표현가능하다.

1.4 상수와 배열, 배열과 배열의 연산

행렬의 덧셈(addition)과 뺄셈(subtraction)은 두 행렬의 크기가 같은 경우에만 가능하다. m*n 행렬 $A=(a_{ij})$와 m*n 행렬 $B=(b_{ij})$에 대하여, $(a_{ij} + b_{ij})$ [혹은 $(a_{ij} - b_{ij})$]를 i-행 j-열 원소로 가지는 행렬을 A와 B의 합 A+B [혹은 차 A-B] 이라 한다. 만약 행렬 $A=(a_{11}=a)$ 혹은 행렬 $B=(b_{11}=b)$가 상수 즉 1*1 행렬이면 합 A+B [혹은 차 A-B]는 상수를 각각의 행렬의 원소에 더한 [뺀] 것으로 생각한다. 즉, $A+B=(a+b_{ij})$, $(a_{ij}+b)$ [혹은 $A-B=(a-b_{ij})$, $(a_{ij}-b)$]와 같이 정의한다. 마찬가지로 상수 c와 행렬 $A=(a_{ij})$의 곱셈과 나눗셈은 행렬의 각각에 원소에 대해 곱하거나 나눈 것으로 한다. 즉, 상수 곱셈은 c*A = A*c = $(c*a_{ij})$로, 상수 나눗셈은 $A/c=(a_{ij}/c)$로 정의한다.

예제 1.4.1 (덧셈과 상수연산)

```
>> g, h                          % recall previous array
g =
     1     2     3     4
     5     6     7     8
     9    10    11    12
h =
     1     1     1     1
     2     2     2     2
     3     3     3     3
```

```
>> g+ones(3,3)
??? Error using ==> +
Matrix dimensions must agree.

>> g+h                 % add h to g on an element-by-element basis
ans =
     2     3     4     5
     7     8     9    10
    12    13    14    15

>> ans-h               % subtract h from the previous answer to get g back
ans =
     1     2     3     4
     5     6     7     8
     9    10    11    12

>> g-2                 % g의 모든 원소에서 2를 뺀 경우
ans =
    -1     0     1     2
     3     4     5     6
     7     8     9    10

>> 2*g-1
ans =
     1     3     5     7
     9    11    13    15
    17    19    21    23

>> 2*g-h               % multiplies g by 2 and subtracts h from the result
ans =
     1     3     5     7
     8    10    12    14
    15    17    19    21
```

상수와의 곱셈과 나눗셈을 일반화하여 같은 크기의 두 m*n 행렬 $A=(a_{ij})$와 행렬 $B=(b_{ij})$에 대하여, 행렬의 각각의 위치에 있는 원소끼리의 곱셈이나 나눗셈을 하고자 할 경우는 .*(dot multiplication symbol)이나 ./(dot division symbol)을 사용한다. 즉, $A.*B=(a_{ij}*b_{ij})$이고, $A./B=(a_{ij}/b_{ij})$이다. 역수를 곱하는 연산 .\는 $B.\backslash A = (b_{ij}^{-1}*a_{ij} = a_{ij}/b_{ij})$와 같이 정의되어, 원소끼리의 연산에서는 $A./B=(a_{ij}/b_{ij})$와 구별이 없다. 그러나 행렬연산에서는 교환법칙이 성립하지 않아, 행렬의 나눗셈에 사용되는 / 와 \ 연산은 서로 다른 결과를 가진다. 행렬의 곱셈과 나눗셈에 대해서는 다음 절에서 보다 자세히 공부할 것이다.

예제 1.4.2 (원소끼리의 곱셈과 나눗셈)

```
>> g.*h
 ans =
       1     2     3     4
      10    12    14    16
      27    30    33    36

>> g./h
ans =
      1.0000    2.0000    3.0000    4.0000
      2.5000    3.0000    3.5000    4.0000
      3.0000    3.3333    3.6667    4.0000

>> h.\g      % 슬래쉬(slash)나 백슬래쉬(back slash) 모두 나눗셈을 정의할 때,
            % 슬래쉬 아래쪽이 슬래쉬 위쪽을 나눈다.
ans =
      1.0000    2.0000    3.0000    4.0000
      2.5000    3.0000    3.5000    4.0000
      3.0000    3.3333    3.6667    4.0000

>> g*h       % 행렬의 곱셈은 매우 특별한 방식으로 정의 되어 있다.
??? Error using ==> *
Inner matrix dimensions must agree.

>> g/h
Warning: Rank deficient, rank = 1  tol =    5.3291e-15
ans =
         0         0    0.8333
         0         0    2.1667
         0         0    3.5000

>> h\g       % 행렬의 두 나눗셈이 서로 다른 결과를 가지고 있다.
Warning: Rank deficient, rank = 1  tol =     3.3233e-15
ans =
    2.7143    3.1429    3.5714    4.0000
         0         0         0         0
         0         0         0         0
         0         0         0         0
```

배열의 지수승을 할 경우는 ^을 사용하는데, 이 연산자는 행렬 자체를 지수승 하는 것이고 .^을 이용하면 각각의 원소에 지수승을 한 것이 된다.

예제 1.4.3 (원소끼리의 지수승)

```
>> g,h
g =
      1     2     3     4
      5     6     7     8
      9    10    11    12
h =
      1     1     1     1
      2     2     2     2
      3     3     3     3

>> g.^2
ans =
     1     4     9    16
    25    36    49    64
    81   100   121   144
>> g.^-1          % MATLAB doesn't like this syntax
???  g.^-
        |
Missing variable or function

>> g.^(-1)        % However, with parentheses it works!
ans =
    1.0000    0.5000    0.3333    0.2500
    0.2000    0.1667    0.1429    0.1250
    0.1111    0.1000    0.0909    0.0833

>> 2.^g
ans =
       2       4       8      16
      32      64     128     256
     512    1024    2048    4096

>> g.^h
ans =
       1       2       3       4
      25      36      49      64
     729    1000    1331    1728
```

1.5 행렬의 곱셈과 나눗셈

m−by−p 행렬 A=(a_{ij}; i=1,⋯,m; j=1,⋯p)와 p−by−n 행렬 B=(b_{ij}; i=1,⋯,p; j=1,⋯n)에 대하여, 행렬의 곱 A*B를 정의하자. 즉, A의 열의 크기와 B의 행의 크기가 p로 같은 두 행렬

$$A=\begin{bmatrix} a_{11}, & a_{12}, & \cdots, & a_{1p} \\ a_{21}, & a_{22}, & \cdots, & a_{2p} \\ \cdots, & \cdots, & \cdots, & \vdots \\ a_{m1}, & a_{m2}, & \cdots, & a_{mp} \end{bmatrix}, \quad B=\begin{bmatrix} b_{11}, & b_{12}, & b_{13}, & \cdots, & b_{1n} \\ b_{21}, & b_{22}, & b_{23}, & \cdots, & b_{2n} \\ \cdots, & \cdots, & \cdots, & \cdots, & \vdots \\ b_{p1}, & b_{p2}, & b_{p3}, & \cdots, & b_{pn} \end{bmatrix}$$

에 대하여, 두 행렬의 곱 A*B의 i−행 j−열 원소는

$$(A*B)_{ij}=\sum_{k=1}^{p} a_{ik}b_{kj}=a_{i1}b_{1j}+a_{i2}b_{2j}+\cdots+a_{ip}b_{pj}$$

로 정의한다. 행렬 A를 구성하고 있는 m개의 행벡터를 차례로 $A_1=[a_{11}, a_{12}, \cdots, a_{1p}], \cdots, A_m=[a_{m1}, a_{m2}, \cdots, a_{mp}]$라 하고, B를 n개의 열벡터 $B^1=\begin{bmatrix} b_{11} \\ b_{21} \\ \cdots \\ b_{p1} \end{bmatrix}, \cdots, B^n=\begin{bmatrix} a_{1n} \\ a_{2n} \\ \cdots \\ a_{pn} \end{bmatrix}$의 조합이라 하면, AB의 i−행 j−열 원소는 두 개의 p차 벡터A_i와 B^j의 내적으로 표시할 수 있다.

$$A*B=\begin{bmatrix} A_1 \cdot B^1, & A_1 \cdot B^2, & A_1 \cdot B^3, & \cdots, & A_1 \cdot B^n \\ A_2 \cdot B^1, & A_2 \cdot B^2, & A_2 \cdot B^3, & \cdots, & A_2 \cdot B^n \\ \cdots, & \cdots, & \cdots & \cdots, & \vdots \\ A_m \cdot B^1, & A_m \cdot B^2, & A_m \cdot B^3, & \cdots, & A_m \cdot B^n \end{bmatrix}$$

이와 같이 행렬의 곱은 내적의 일반화로 생각할 수 있으므로, MATLAB에서는 벡터의 내적에 해당하는 연산자가 따로 없고 행렬의 곱을 이용한다. 이때, 내적연산자의 앞의 인수는 행벡터이고 뒤의 인수는 같은 길이의 열벡터인 것에 유의하여야 한다.

예제 1.5.1 (행렬의 곱과 내적을 이용한 사이 각 구하기)

```
>> A=[2 1 5; 1 3 2], B=[3 4; -1 2; 2 1]
A =
     2     1     5
     1     3     2
B =
     3     4
    -1     2
     2     1
>> A*B
ans =

    15    15
     4    12

>> x = 1:4, y=2:2:8
x =
     1     2     3     4
y =
     2     4     6     8
>> x * y
??? Error using ==> *
Inner matrix dimensions must agree.
>> x * y'
ans =
    60

>> a=[-1 2 4]; b=[1.5 2 -1];              % angle between vector a and b
>> theta = acos( a*b' / sqrt(a*a'*b*b') )
```

행렬의 곱셈은 앞 행렬의 열의 수와 뒤 행렬의 행의 수가 일치할 경우에만 정의할 수 있다. 즉, 3*2 행렬 A와 2*2 행렬 B의 경우 A*B는 3*2 행렬이지만 B*A는 정의되지 않는다. 또한, 두 행렬 모두 차수가 같은 정방행렬이라도, 일반적으로 행렬의 곱셈 연산은 교환법칙이 성립하지 않는다.

$$AB \neq BA$$

그러나 AB와 BC의 행렬 곱셈연산이 각각 정의되는 경우에 대하여는 (AB)와 C의 곱과 A와 (BC)의 곱의 연산을 계산할 수 있고, 그 결과는 같다. 이를 행렬 곱셈에 대한 결합법칙이라 한다.

$$(AB)C = A(BC) = ABC$$

또한, AB와 AC가 각각 정의되는 경우, 행렬의 곱과 합에 대한 배분법칙이 성립한다.

$$A(B+C) = AB + AC$$

행렬의 곱셈은 교환법칙이 성립하지 않으므로 행렬 나눗셈을 정의하는데 스칼라 수의 경우보다 더 큰 주의가 요구된다. 곱셈이 정의되는 행렬 A와 B에 대해 A*B=C라 하면, 나눗셈은 A=C/B 혹은 B=A\C에서와 같이 (우측)나눗셈과 (좌측)나눗셈의 두 가지로 정의할 수 있다. 이러한 행렬 나눗셈 연산자의 일반적 정의는 보다 확장된 수학개념을 필요로 하므로 다음 장에서 보다 자세히 설명하기로 하고, 여기에서는 특이하지 않는(non-singular) 정방행렬 A와 B에 대해서만 살펴보기로 하자. 만약 $AX = XA$가 단위행렬일 때 X를 A의 곱셈에 대한 역원이라 하고 A^{-1}이라고 표시한다. 곱셈에 대한 역원을 곱하는 것을 나눗셈이라 하고, 정방행렬의 경우에는 다음의 두 가지 경우가 있다.

$$A*B = C \quad \rightarrow \quad \begin{cases} A = A*B*B^{-1} = C/B \\ B = A^{-1}*A*B = A\backslash C \end{cases}$$

예제 1.5.3 (행렬의 나눗셈)

```
>> A=[2 1 5; 1 3 2], B=[3 4; -1 2; 2 1]  % 정방행렬이 아닌 경우의 나눗셈
A =
     2     1     5
     1     3     2
B =
     3     4
    -1     2
     2     1
>> (A*B)/B
ans =
    4.5000   -1.5000         0
    2.0000    2.0000         0
>> A\(A*B)
ans =
         0         0
   -0.7692    2.3077
    3.1538    2.5385

>> A=rand(3,3), B=rand(3,3)                    % 일반적 정방행렬의 나눗셈
A =
    0.2190    0.6793    0.5194
    0.0470    0.9347    0.8310
    0.6789    0.3835    0.0346
B =
    0.0535    0.0077    0.4175
    0.5297    0.3834    0.6868
    0.6711    0.0668    0.5890
>> (A*B)/B
ans =
    0.2190    0.6793    0.5194
    0.0470    0.9347    0.8310
    0.6789    0.3835    0.0346
>> A\(A*B)
ans =
    0.0535    0.0077    0.4175
    0.5297    0.3834    0.6868
    0.6711    0.0668    0.5890
```

1.6 배열의 주소할당

MATLAB®에서 배열 안의 원소들은 괄호 속에 인덱스를 사용하여 원소의 위치를 나타낼 수 있다. 즉 x(1)는 배열 안의 첫 번째 원소를 나타낸다. 한 번에 여러 원소 원소를 나타내 주기 위해서는 콜론(:)표시를 사용할 수 있다. 예를 들어 x의 1번째부터 5번째 원소로 구성된 새로운 배열을 만들려면 x(1:5)와 같이 표시하면 된다.

예제 1.6.1 (1차원 배열의 주소할당)

```
>> x=(0:0.1:1)*pi;                % 0에서 0.1*pi씩 증가하여 1*pi까지 11개의 원소
>> y=sin(x);
>> x(3)                           % The third element of x
ans =
  0.6283
>> y(5)                           % The fifth element of y
ans =
  0.9511
>> x(1:5)                         % 첫 번째에서 다섯 번째 원소까지 나타냄
ans =
    0    0.3142    0.6283    0.9425    1.2566
>> y(3:-1:1)
ans =
    0.5878    0.3090    0

>> x(2:2:7)      % 두 번째, 네 번째, 여섯 번째 원소.
                 % 즉, 두 번째 원소부터 7보다 작은 원소까지 2씩 증가시킴.
ans =
    0.3142    0.9425    1.5708
>> y([8  2  9  1])      % [8  2  9  1]는 y의 원소들의 위치를 나타내는 배열이다
ans =
    0.8090    0.3090    0.5878    0
```

예제 1.6.2 (2차원 배열의 주소할당)

```
>> A=[1  2  3; 4  5  6; 7  8  9]
A =
     1     2     3
     4     5     6
     7     8     9

 >> A(2,3)=0              % A(2,3)의 원소 [2nd row, 3rd column]가 0으로 바뀜
 A =
     1     2     3
     4     5     0
     7     8     9

 >> A(2,6)=1     % 원래 A가 6개의 열을 가지지 않음으로 0을 채워서 늘려놓았다.
 A =
     1     2     3     0     0     0
     4     5     0     0     0     1
     7     8     9     0     0     0

 >> A=[1  2  3; 4  5  6; 7  8  9];        % restore original data

 >> B=A(3:-1:1,1:3)                       % row의 순서를 거꾸로 쓰는 법
 B =
     7     8     9
     4     5     6
     1     2     3

>> B=A(3:-1:1,:)     % 마지막의 싱글 콜론(single colon)은 모든 열을 가지는
                     % 것을 의미한다. 즉, :은 1:3을 짧게 줄여 쓴 것이다.
B =
     7     8     9
     4     5     6
     1     2     3

>> C=[A  B(:,[1  3])]
C =
     1     2     3     7     9
     4     5     6     4     6
     7     8     9     1     3

>> B=A(1:2,2:3)
B =
     2     3
     5     6

>> C=[1  3]
C =
     1     3
>> B=A(C,C)      % 여기서는 배열C가 A의 인덱스(index)로 사용되어 진다.
B =
     1     3
     7     9
```

예제 1.6.3 (2차원 배열의 조작)

```
>> A = [1 2 3; 4 5 6; 7 8 9]
>> sum(A)                    % 합을 구하는 함수들은 행단위로 연산한다.
ans =
      12    15    18
>> sum(sum(A))
ans =
      45

>> A(1:9)                    % A를 행벡터로 바꿈
A =   1     4     7     2     5     8     3     6     9

>> B=A(:)                    % A의 열들을 하나의 열로 펼쳐 놓음
B =
      1
      4
      7
      2
      5
      8
      3
      6
      9

>> B=B.'                     % dot-transpose operation을 이용하여 row로 바꿈
B =
     1    4    7    2    5    8    3    6    9
```

```
>> B=[1 2 3; 4 5 6; 7 8 9];
>> B(:,2)=[]                 % [](empty matrix)는 앞의 B의 모든 행의 두 번째 열을
                             % 삭제하라는 의미이다
B =
     1    3
     4    6
     7    9
>> B=B.'
B =
      1    4    7
      3    6    9
```

```
>> B(2,:)=[]
B =
     1     4     7

>> A(2,:)=B      % 두 번째 열의 모든 행 원소를 B로 대체
A =
     1     2     3
     1     4     7
     7     8     9
>> B=A(:, [2   2   2   2])      % 모든 행을 두 번째 열에 의해 네 번 반복.
                                % 이런 것을 Tony's Trick이라고 함
B =
     2     2     2     2
     4     4     4     4
     8     8     8     8
```

위에서 살펴본 예에서와 같이 어떤 배열의 일부 원소들만으로 만들어진 배열을 부배열(Sub-array)이라고 부른다. 이와 같이 배열의 일부 원소를 선택하여 부배열을 만드는 방법들은 아래의 표로 요약된다.

배열의 주소 지정

A(r,c)	행과 열 색인 벡터 r과 c에 의하여 정의된 A 내에서 부배열
A(r,:)	행 색인 벡터 r과 모든 열에 의하여 정의된 A 내에서 부배열
A(:,c)	열 색인 벡터 c와 모든 행에 의하여 정의된 A 내에서 부배열
A(:)	열 우선 순서에 의해 나열된 열벡터로서의 A의 모든 원소의 배열
A(i)	단일 색인 벡터 i에 의하여 정의된 A 내에서 부배열
A(x)	0과 1을 원소로 가지는 논리 배열 x에 의하여 정의된 A 내에서 부배열 (단, x의 크기는 A와 같아야 함.)

MATLAB®에서는 이러한 주소할당 방법을 응용하여 행렬의 모양과 크기를 바꾸거나, 행렬 원소의 일부를 변화시킬 수 있는 다양한 배열 조작 함수들을 제공한다.

배열 조작 함수들

flipud(A)	주어진 행렬 A를 상하 방향으로 교환한다.
fliplr(A)	주어진 행렬 A를 좌우 방향으로 교환한다.
rot90(A)	행렬 A를 90 °씩 회전한다. rot90([1 2; 3 4]) = [2 4; 1 3]
reshape(A,m,n)	행렬 A로부터 열 방향으로 가져다 m×n 행렬을 만들어 준다. (단, 행렬 A는 m×n개의 원소를 가져야한다.)
diag(A)	행렬 A의 대각 성분을 추출한다.
diag(v)	대각 성분이 벡터 v인 정방행렬을 만든다.
tril(A)	행렬 A의 하단 삼각 행렬을 추출한다.
triu(A)	행렬 A의 상단 삼각 행렬을 추출한다.

1.7 논리배열과 부배열 찾기

0과 1로 이루어진 배열을 논리배열(logical array)라고 부른다. 논리배열은 대소를 비교하는 관계연산자이나 논리연산자 등의 결과 값으로 쉽게 얻어진다.

예제 1.7.1 (논리배열)

```
>> x=-3:3        % Create data
x =
    -3   -2   -1    0    1    2    3

>> abs(x)>1      % x의 절대값이 1보다 크면 1(참)을, 아니면 0(거짓)을 돌려준다
ans =
     1    1    0    0    0    1    1

>> y=x(abs(x)>1)     % x의 절대값이 1보다 큰 경우의 x값을 돌려준다
y =
    -3    -2     2     3

>> y=x([1  1  1  1  0  0  0])    % 단지 처음 네 개의 숫자만 선택하고
                                 % 나머지는 없앤다
y =
    -3    -2    -1     0

>> y=x([1  1  1  1])     % x의 첫 번째 원소를 네 번 반복
y =
    -3    -3    -3    -3
```

예제 1.7.3 (행렬의 관계연산)

```
>> B=[5  -3;2  -4]
B =
     5    -3
     2    -4

>> x=abs(B)>2                    % 행렬에 대한  관계 연사자의 적용
x =
     1     1
     0     1

>> y=B(abs(B)>2)
y =
       5
      -3
      -4
```

배열 찾기

i=find(x)	배열 x의 원소가 영이 아닌 인덱스를 반환한다.
[r,c]=find(X)	배열 X의 원소가 영이 아닌 행과 열의 인덱스를 반환한다.

```
>> x=-3:3
x =
    -3    -2    -1     0     1     2     3

>> k=find(abs(x)>1)      % find 함수를 이용해서 절대값이 1 보다 큰 경우의
                         % 배열 x의 인덱스를 찾음
k =
     1     2     6     7

>> A=[1  2  3; 4  5  6; 7  8  9]
A =
     1     2     3
     4     5     6
     7     8     9
>> [i,j]=find(A>5)       % 인덱스 i와 j에 각각 행과 열의 인덱스가 저장된다
i =
     3
     3
     2
     3
j =
     1
     2
     3
     3
```

1.9 기록 파일과 그림 파일 저장하기

MATLAB®을 사용하다 보면 자신이 입력한 명령어와 계산 결과를 저장하고 싶은 경우가 생긴다. 이때 이용할 수 있는 명령어가 diary 명령어이다. diary file_name.txt라는 명령어를 수행하면, 이후 사용자의 입력과 계산 결과를 모두 file_name.txt에 저장하여 준다. 이러한 저장 작업을 멈추려면 diary off, 다시 저장을 계속하려면 diary on이란 명령어들을 이용할 수 있다. 이렇게 저장된 명령어는 이후 텍스트 문서 편집기나 혼글과 같은 문서 편집기로 불러들여 작업할 수 있다. 계산한 결과 값을 save 명령어를 이용하여 mat 파일에 저장하였다가, 이후 필요한 때에 load 명령어를 이용하여 불러올 수도 있다. Diary 명령어가 계산 과정 중 화면상에 나타나는 출력을 저장하는데 반하여, save와 load 명령어는 MATLAB이 기억하고 있는 변수 값을 저장하거나 읽어 온다는 점에서 차이가 있다.

Plot 명령어를 이용하여 그려진 그림을 프린터로 출력하려면, 그림 창 상단에 있는 File 메뉴에서 출력(Print) 명령을 이용하면 된다. 직접 출력하는 대신, 윈도우 버퍼에 저장하여, 문서편집기나 그림편집기에서 편집할 수도 있다. 이와 같이 하려면, MATLAB 그림 창에 있는 Edit 메뉴의 그림복사 (Copy Figure)하기 명령을 이용하여 windows buffer에 저장한 후, 문서 편집 혹은 그림 편집 프로그램에서 붙여넣기(Paste) 명령을 이용하여 그림 창에 있는 그림을 복사하면 된다. 이 보다 더 확장된 그림 저장 기능을 원한다면, print 명령어를 이용할 수도 있다.

예제 1.9.1 (Diary 저장과, Mat-File 저장, 그리고 그림화일 저장)

```
>> diary sec19.txt          % 아래에 나타나는 출력을 sec19.txt에 저장한다.
>> x = -pi:0.01:pi;
>> % My work will be saved on sec19.txt
>> y = sin(x);
>> save sec19 x y           %변수 x, y의 값을 sec19.mat 파일에 저장한다.
>> clear                    %모든 저장 변수 값을 소거한다.
>> load sec19               %sec19.mat에 저장된 변수 값을 읽어 온다.
>> plot(x,y)
>> print -dbmp sec19.bmp    %그림 창에 나타난 그래프를 bmp형식으로 저장한다.
>> diary off
```