Chapter 3

Mathematical Operations with Arrays

MATLAB An Introduction With Applications, 6th Edition
Dr. Amos Gilat
The Ohio State University

Slide deck by
Dr. Greg Reese
Miami University

Previously dealt with scalars (single numbers). Will now work with arrays, which in general have more than one number

This chapter covers basics of using arrays

array – a rectangular arrangement of numbers with one or more dimensions

vector – an array with only one column or one row

scalar – an array with exactly one row and one column, i.e., a single number

Note – "matrix" and "array" are often used synonymously

- Use + to add two arrays or to add a scalar to an array
- Use – to subtract one array from another or to subtract a scalar from an array
  - When using two arrays, they must both have the same dimensions (number of rows and number of columns)
  - Vectors must have the same dimensions (rows and columns), not just the same number of elements

When adding two arrays *A* and *B*, MATLAB adds the corresponding elements, i.e.,

- It adds the element in the first row and first column of *A* to the element in the first row and column of *B*
- It adds the element in the first row and second column of *A* to the element in the first row and second column of *B*, etc.

This called *elementwise addition*

When subtracting two arrays *A* and *B*, MATLAB performs an elementwise subtraction

In general, an operation between two arrays that works on corresponding elements is called an *elementwise operation*

EXAMPLE

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

$$A + B = \begin{bmatrix} A_{11} + B_{11} & A_{12} + B_{12} & A_{13} + B_{13} \\ A_{21} + B_{21} & A_{22} + B_{22} & A_{23} + B_{23} \end{bmatrix}$$

$$A - B = \begin{bmatrix} A_{11} - B_{11} & A_{12} - B_{12} & A_{13} - B_{13} \\ A_{21} - B_{21} & A_{22} - B_{22} & A_{23} - B_{23} \end{bmatrix}$$

When adding a scalar to an array, MATLAB adds the scalar to every element of the array

When subtracting a scalar from an array, MATLAB subtracts the scalar from every element of the array

EXAMPLE

For c a scalar and $A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \end{bmatrix}$

$$A + c = \begin{bmatrix} A_{11} + c & A_{12} + c & A_{13} + c \\ A_{21} + c & A_{22} + c & A_{23} + c \end{bmatrix}$$

$$A - c = \begin{bmatrix} A_{11} - c & A_{12} - c & A_{13} - c \\ A_{21} - c & A_{22} - c & A_{23} - c \end{bmatrix}$$

There are two ways of multiplying matrices – matrix multiplication and elementwise multiplication

MATRIX MULTIPLICATION

- Type used in linear algebra
- MATLAB denotes this with asterisk (*)
- Number of columns in left matrix must be same as number of rows in right matrix

For example, if A is a 4×3 matrix and B is a 3×2 matrix:

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

then the matrix that is obtained with the operation $A*B$ has dimensions 4×2 with the elements:

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

A numerical example is:

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

Example:

```
>> A = [ 1 4 3; 2 6 1; 5 2 8 ]
A =   1     4     3
      2     6     1
      5     2     8
>> B = [ 5 4; 1 3; 2 6 ]
B = 5     4
    1     3
    2     6
>> A * B
ans = 15    34
      18    32
      43    74
```

Example:

Note that `B*A` is not even defined, because the number of columns in `B` is not equal to the number of rows in `A`. Attempting to compute `B*A` produces an error:

```
>> B * A
Error using  *
Inner matrix dimensions must agree.
```

When performing matrix multiplication on two square matrices

- They must both have the same dimensions
- The result is a matrix of the same dimension
- In general, the product is not commutative, i.e., $A * B \neq B * A$

```
>> A = randi(3,3)
A =
     3     3     1
     3     2     2
     1     1     3
>> B=randi(3,3)
B =
     3     3     1
     1     2     2
     3     3     3
>> AB = A*B
AB =
    15    18    12
    17    19    13
    13    14    12
```

```
>> BA = B*A
BA =
    19    16    12
    11     9    11
    21    18    18
>> AB == BA
ans =
     0     0     1
     0     0     0
     0     0     0
```

When performing matrix multiplication on two vectors

- They must both be the same size
- One must be a row vector and the other a column vector
- If the row vector is on the left, the product is a scalar
- If the row vector is on the right, the product is a square matrix whose side is the same size as the vectors

```
>> h = [ 2 4 6 ]
h =
     2     4     6
>> v = [ -1 0 1 ]'
v =
    -1
     0
     1
```

```
>> h * v
ans =
     4
>> v * h
ans =
    -2    -4    -6
     0     0     0
     2     4     6
```

`dot(a,b)` computes the *inner product*, also called the *dot product*

- `a` and `b` must be same size
- Any combination of vertical or horizontal vectors
- Result is always a scalar

EXAMPLE

```
>> h = [ 2 4 6 ]
h =
     2     4     6
>> v = [ -1 0 1 ]'
v =
    -1
     0
     1
>> dot(h,v)
ans =
     4
>> dot(v,h)
ans = 4
```

Linear algebra rules of array multiplication provide a convenient way for writing a system of linear equations. For example, the following system of three equations with three unknowns:

$$A_{11}x_1 + A_{12}x_2 + A_{13}x_3 = B_1$$
$$A_{21}x_1 + A_{22}x_2 + A_{23}x_3 = B_2$$
$$A_{31}x_1 + A_{32}x_2 + A_{33}x_3 = B_3$$

can be written in a matrix form by:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix}$$

and in matrix notation by:

$$AX = B \quad \text{where } A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}, X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \text{ and } B = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix}.$$

Identity matrix

- Square matrix with ones on main diagonal and zeros elsewhere
  - Main diagonal goes from top left to bottom right
- When do matrix multiplication on any array or vector with the identity matrix, array or vector is unchanged
  - True whether multiply with identity matrix on left or on right
- MATLAB command `eye(n)` makes an $n \times n$ identity matrix

Inverse of a matrix:

Matrix B is the *inverse* of matrix A if matrix product of A and B is the identity matrix I

- Both matrices must be square and same dimensions
- Multiplication can be from either side, i.e.,

$$BA = AB = I$$

EXAMPLE

$$\begin{bmatrix} 2 & 1 & 4 \\ 4 & 1 & 8 \\ 2 & -1 & 3 \end{bmatrix} \begin{bmatrix} 5.5 & -3.5 & 2 \\ 2 & -1 & 0 \\ -3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 5.5 & -3.5 & 2 \\ 2 & -1 & 0 \\ -3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 4 \\ 4 & 1 & 8 \\ 2 & -1 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In math, inverse of a matrix *A* is written as A^{-1}

In MATLAB, get inverse with `A^-1` or `inv(A)`

```
>> A=[2 1 4; 4 1 8; 2 -1 3]
A =
     2     1     4
     4     1     8
     2    -1     3
>> B=inv(A)
B =
    5.5000   -3.5000    2.0000
    2.0000   -1.0000         0
   -3.0000    2.0000   -1.0000
>> A*B
ans =
     1     0     0
     0     1     0
     0     0     1
```

Creating the matrix A.

Use the inv function to find the inverse of A and assign it to B.

Multiplication of A and B gives the identity matrix.

Determinants:

A *determinant* is a function associated with square matrices

- In math, determinant of A is written as $det(A)$ or $|A|$
- In MATLAB, compute determinant of A with `det(A)`
- A matrix has an inverse only if it is square and its determinant is not zero

If you don't remember much about determinants, go back to your linear algebra book and review them

Left division, \:

Left division is one of MATLAB's two kinds of array division

- Used to solve the matrix equation $AX=B$
  - A is a square matrix, X, B are column vectors
  - Solution is $X = A^{-1}B$

In MATLAB, solve by using left division operator (\), i.e.,

```
>> X = A \ B
```

When solving set of linear equations, use left division, not inverse, i.e., use `X=A\B` not `X=inv(A)*B`

Left division is

- 2-3 times faster
- Often produces smaller error than `inv()`
- Sometimes `inv()` can produce erroneous results

Right division, /:

Right division is the other kind of MATLAB's array division

- Used to solve the matrix equation $XC=D$
  - C is a square matrix, X, D are row vectors
  - Solution is $X = D \cdot C^{-1}$

In MATLAB, solve by using right division operator (/), i.e.,

```
>> X = D / C
```

Another way of saying *elementwise* operations is *element-by-element* operations

- Addition and subtraction of arrays is always elementwise
- Multiplication, division, exponentiation of arrays can be elementwise
- Both arrays must be same dimension

Do elementwise multiplication, division, exponentiation by putting a period in front of the arithmetic operator

Symbol	Description	Symbol	Description
.*	Multiplication	./	Right division
.^	Exponentiation	.\	Left Division

If two vectors a and b are $a=[a_1\ a_2\ a_3\ a_4]$ and $b=[b_1\ b_2\ b_3\ b_4]$, then element-by-element multiplication, division, and exponentiation of the two vectors gives:

$$a.*b = [\,a_1*b_1\ \ a_2*b_2\ \ a_3*b_3\ \ a_4*b_4]$$

$$a./b = \left[\,a_1/b_1\ \ a_2/b_2\ \ a_3/b_3\ \ a_4/b_4\right]$$

$$a.\hat{}b = \left[(a_1)^{b_1}\ (a_2)^{b_2}\ (a_3)^{b_3}\ (a_4)^{b_4}\right]$$

If two matrices A and B are

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

then element-by-element multiplication and division of the two matrices give:

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

Element-by-element exponentiation of matrix A gives:

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

Element-by-element multiplication, division, and exponentiation are demonstrated in Tutorial 3-2.

ELEMENTWISE MULTIPLICATION

- Use `.*` to get elementwise multiplication (notice period before asterisk)
- Both matrices must have the same dimensions

```
>> A = [1 2; 3 4];
>> B = [0 1/2; 1 -1/2];
>> C = A .* B
>> C =
     0   1
     3  -2
```

If matrices not same dimensions in elementwise multiplication, MATLAB gives error

```
>> A = [ 1 2; 3 4];
>> B = [1 0]';
>> A .* B % Meant matrix multiplication!
??? Error using ==> times
Matrix dimensions must agree.
>> A * B % this works
ans =
     1
     3
```

Be careful – when multiplying square matrices

- Both types of multiplication always work
- If you specify the wrong operator, MATLAB will do the wrong computation and there will not report an error!
  - Difficult to find this kind of mistake

EXAMPLE

```
>> A = [1 2; 3 4];
>> B = [0 1/2; 1 -1/2];
>> A .* B
>> ans
     0   1
     3  -2
>> A * B
ans =
     2.0000  -0.5000
     4.0000  -0.5000
```

Elementwise computations useful for calculating value of a function at many values of its argument

```
>> x=[1:8]
x =
   1   2   3   4   5   6   7   8
>> y=x.^2-4*x
y =
  -3  -4  -3   0   5  12  21  32
>>
```

Create a vector x with eight elements.

Vector x is used in element-by-element calculations of the elements of vector y.

Built-in MATLAB functions can accept arrays as inputs

- When input is array, output is array of same size with each element being result of function applied to corresponding input element

```
>> x=[0:pi/6:pi]

x =
    0    0.5236    1.0472    1.5708    2.0944    2.6180    3.1416

>>y=cos(x)

y =
  1.0000    0.8660    0.5000    0.0000   -0.5000   -0.8660   -1.0000

>>
```

Example with matrix argument

```
>> d=[1 4 9; 16 25 36; 49 64 81]
d =
     1     4     9
    16    25    36
    49    64    81
>> h=sqrt(d)
h =
     1     2     3
     4     5     6
     7     8     9
```

Creating a 3×3 array.

h is a 3×3 array in which each element is the square root of the corresponding element in array d.

Vectorization is MATLAB's ability to use arrays as arguments to functions. Vectorized computations are very efficient

MATLAB has lots of functions for operating on arrays. For a vector `v`

- `mean(v)` – mean (average)
- `max(v)` – maximum value, optionally with index of maximum
- `min(v)` – minimum value, optionally with index of minimum
- `sum(v)` – sum
- `sort(v)` – elements sorted into ascending order

- `median(v)` – median
- `std(v)` – standard deviation
- `det(A)` – determinant of square matrix `A`
- `dot(v,w)` – dot (inner product); `v`, `w` both vectors of same size but any dimension
- `cross(v,w)` – cross product; `v`, `w` must both have three elements but any dimension
- `inv(A)` – inverse of square matrix `A`

See Table 3-1 in book for details on the preceding functions

Note that in all functions of Table 3-1, except for `det(A)` and `inv(A)`, `A` is a vector, not a matrix. Except for those two functions, the table does not apply if `A` is a matrix

Random numbers often used in MATLAB engineering applications

- Simulate noise
- Useful in certain mathematical computations, such as Monte Carlo simulations

MATLAB has three commands that create random numbers – `rand`, `randn`, `randi`

- All can create scalars, vectors, or matrices of random numbers

`rand` generates random numbers uniformly distributed between 0 and 1

- To get numbers between `a` and `b`, multiply output of `rand` by `b-a` and add `a`, i.e., `(b-a)*rand + a`

For example, a vector of 10 elements with random values between –5 and 10 can be created by ($a = -5$, $b = 10$):

```
>> v=15*rand(1,10)-5
v =
   -1.8640    0.6973    6.7499    5.2127    1.9164    3.5174
6.9132   -4.1123    4.0430   -4.2460
```

See Table 3-2 in book for some of different ways of calling `rand`

`randi` generates uniformly distributed random integers in a specified range

For example, to make a 3×4 of random numbers between 50 and 90

```
>> d=randi( [50 90],3,4)
d =
     57    82    71    75
     66    52    67    61
     84    66    76    67
```

See Table 3-3 in book for some of different ways of calling `randi`

`randn` generates random numbers from a normal distribution with mean 0 and standard deviation 1

```
>> d=randn(3,4)
d =
    -0.4326    0.2877    1.1892    0.1746
    -1.6656   -1.1465   -0.0376   -0.1867
     0.1253    1.1909    0.3273    0.7258
```

Call `randn` in same ways as `rand`, as Table 3-2 in book shows

To get normally-distributed numbers with mean μ and standard deviation σ, multiply output of `randn` by μ and add σ, e.g.,

```
>> A = randn( 100, 100 ); % mean=0, std dev = 1
>> mu = 20;
>> sigma = 3;
>> B = sigma * A + mu; % mean = 20, std dev = 3
```

To get normally distributed integers apply the `round` function to previous formula, i.e.,

```
round( sigma * rand + mu )
```

EXAMPLE

```
>> w = round(4*randn(1,6)+50)
W =
51   49   46   49   50   44
```