1.10 Mathematical Operations with Arrays

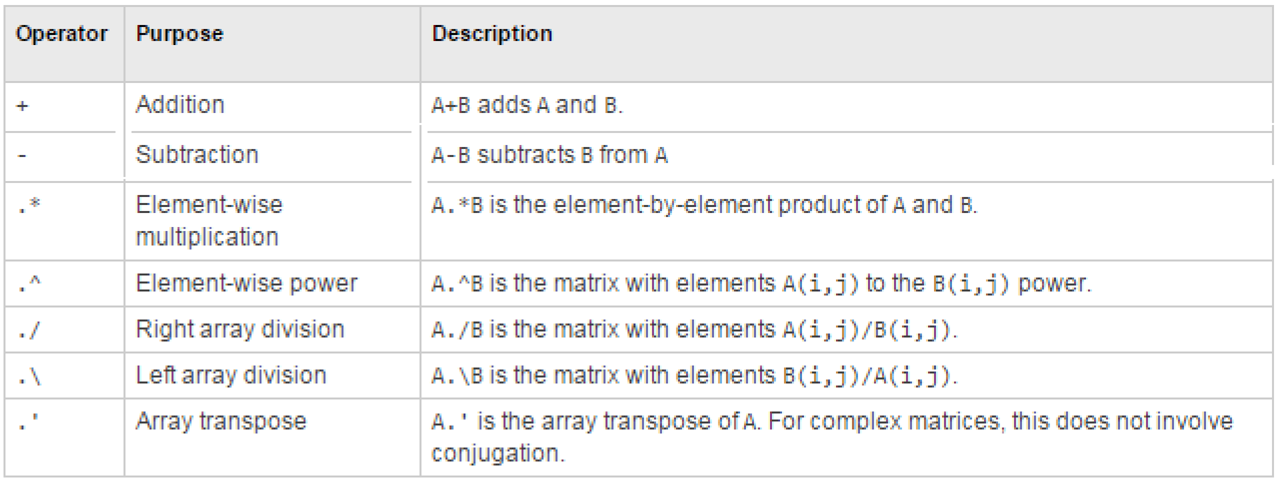
MATLAB has two different types of arithmetic operations: **array operations** and matrix operations. You can use these arithmetic operations to perform numeric computations, for example, adding two numbers, raising the elements of an array to a given power, or multiplying two matrices.

Matrix operations follow the rules of linear algebra. By contrast, array operations execute element by element operations and support multidimensional arrays. The **period** character **(.)** distinguishes the array operations from the matrix operations. However, since the matrix and array operations are the same for addition and subtraction, the character pairs (**.+)** and (**.-)** are unnecessary.

1.10.1 Array Operations

Array operations (shown in table (1.7)) work on corresponding elements of arrays with **equal dimensions**. For vectors, and matrices both operands must be the same size. Each element in the first input gets matched up with a similarly located element from the second input.

Table 1.7: A Summary of Array Arithmetic Operators in MATLAB



Operator	Purpose	Description
`+`	Addition	`A+B` adds `A` and `B`.
`-`	Subtraction	`A-B` subtracts `B` from `A`
`.*`	Element-wise multiplication	`A.*B` is the element-by-element product of `A` and `B`.
`.^`	Element-wise power	`A.^B` is the matrix with elements `A(i,j)` to the `B(i,j)` power.
`./`	Right array division	`A./B` is the matrix with elements `A(i,j)/B(i,j)`.
`.\`	Left array division	`A.\B` is the matrix with elements `B(i,j)/A(i,j)`.
`.'`	Array transpose	`A.'` is the array transpose of `A`. For complex matrices, this does not involve conjugation.

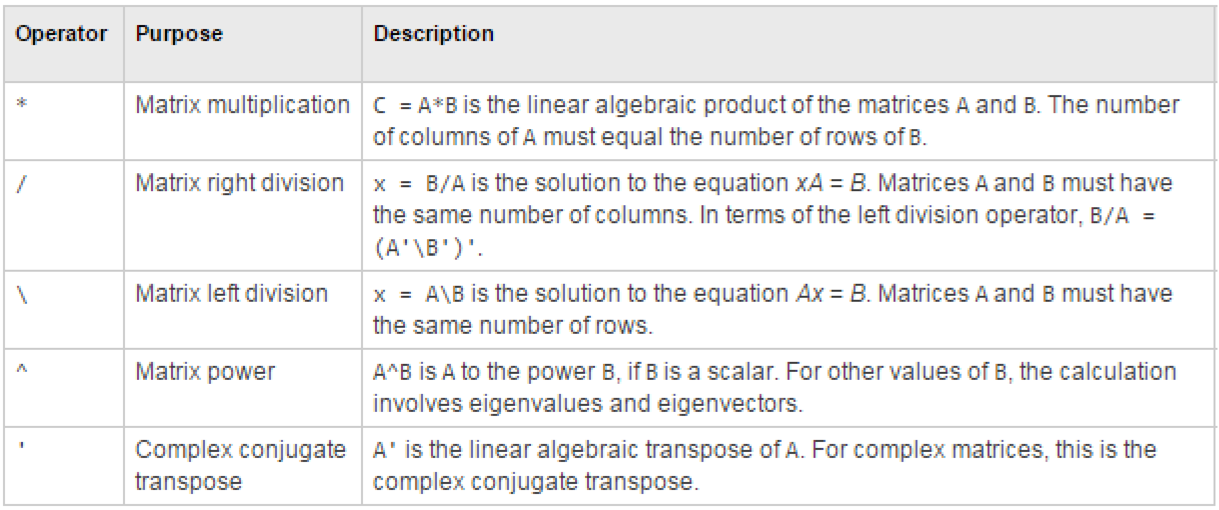
1.10.2 Matrix Operations

Matrix operations (shown in table(1.8)) follow the rules of **linear algebra** and are not compatible with multidimensional arrays. The required size and shape of the inputs in relation to one another depends on the operation. For nonscalar inputs, the matrix operators generally calculate different answers than their array operator counterparts.

For example, if you use the matrix right division operator, /, to divide two matrices, the matrices must have the same number of columns. But if you use the matrix multiplication operator, *, to multiply two matrices, then the matrices must have a common *inner dimension*. That is, the number of columns in the first input must be equal to the number of rows in the second input. The matrix multiplication operator calculates the product of two matrices with the formula,

$$C(i,j) = \sum_{k=1}^{n} A(i,k)B(k,j).$$

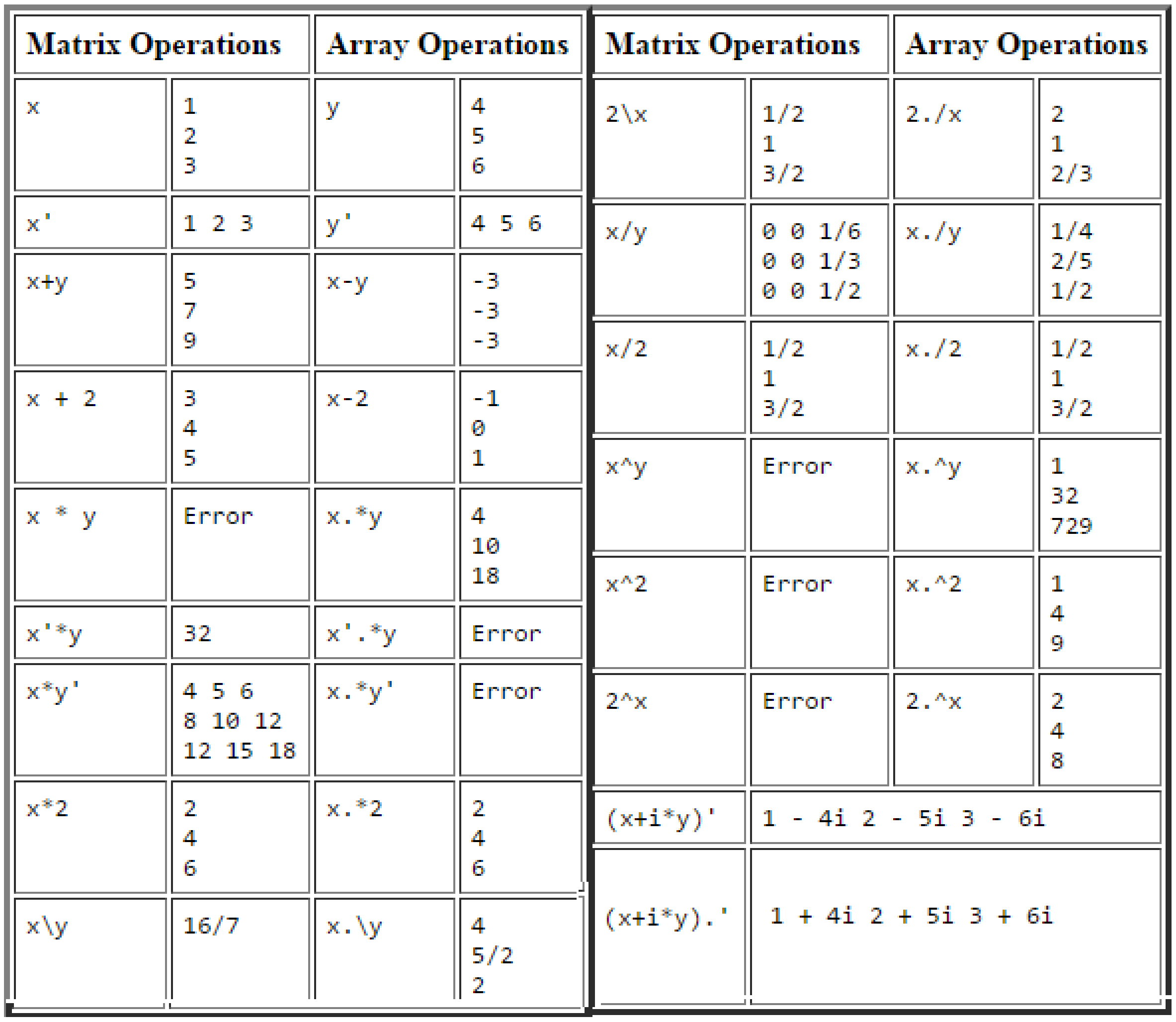
Table 1.8: A Summary of Matrix Arithmetic Operators in MATLAB.



Operator	Purpose	Description
*	Matrix multiplication	C = A*B is the linear algebraic product of the matrices A and B. The number of columns of A must equal the number of rows of B.
/	Matrix right division	x = B/A is the solution to the equation *xA* = *B*. Matrices A and B must have the same number of columns. In terms of the left division operator, B/A = (A'\B')'.
\	Matrix left division	x = A\B is the solution to the equation *Ax* = *B*. Matrices A and B must have the same number of rows.
^	Matrix power	A^B is A to the power B, if B is a scalar. For other values of B, the calculation involves eigenvalues and eigenvectors.
'	Complex conjugate transpose	A' is the linear algebraic transpose of A. For complex matrices, this is the complex conjugate transpose.

Examples

Here are two vectors, and the results of various matrix and array operations on them, printed with format rat.



Matrix Operations		Array Operations	
`x`	1 2 3	`y`	4 5 6
`x'`	1 2 3	`y'`	4 5 6
`x+y`	5 7 9	`x-y`	-3 -3 -3
`x + 2`	3 4 5	`x-2`	-1 0 1
`x * y`	Error	`x.*y`	4 10 18
`x'*y`	32	`x'.*y`	Error
`x*y'`	4 5 6 8 10 12 12 15 18	`x.*y'`	Error
`x*2`	2 4 6	`x.*2`	2 4 6
`x\y`	16/7	`x.\y`	4 5/2 2

Matrix Operations		Array Operations	
`2\x`	1/2 1 3/2	`2./x`	2 1 2/3
`x/y`	0 0 1/6 0 0 1/3 0 0 1/2	`x./y`	1/4 2/5 1/2
`x/2`	1/2 1 3/2	`x./2`	1/2 1 3/2
`x^y`	Error	`x.^y`	1 32 729
`x^2`	Error	`x.^2`	1 4 9
`2^x`	Error	`2.^x`	2 4 8
`(x+i*y)'`	1 - 4i 2 - 5i 3 - 6i		
`(x+i*y).'`	1 + 4i 2 + 5i 3 + 6i		

1.10.3 Generation of random numbers

Simulations of many physical processes and engineering applications frequently require using a number (or a set of numbers) with a random value. MATLAB has three commands—**rand**, **randn**, and **randi**—that can be used to assign random numbers to variables.

1.10.3.1 The rand command:

The rand command generates uniformly distributed random numbers with values between 0 and 1. The command can be used to assign these numbers to a scalar, a vector, or a matrix, as shown in Table 1.9:

Table 1.9 The rand command

Command	Description	Example
rand	Generates a single random number between 0 and 1.	>> rand ans = 0.9501
rand(1, n)	Generates an n elements row vector of random numbers between 0 and 1.	>> a = rand(1, 3) a = 0.4565 0.0185 0.8214
rand(n)	Generates an $n \times n$ matrix with random numbers between 0 and 1.	>> b = rand(3) b = 0.7382 0.9355 0.8936 0.1763 0.9165 0.0579 0.4057 0.4103 0.3529
rand(m, n)	Generates an $m \times n$ matrix with random numbers between 0 and 1.	>> c = rand(2, 3) c = 0.2028 0.6038 0.1988 0.1987 0.2722 0.0153
randperm (n)	Generates a row vector with n elements that are random permutation of integers 1 through n.	>> randperm(7) ans = 5 2 4 7 1 6 3

Sometimes there is a need for random numbers that are distributed in an interval other than (0,1), or for numbers that are integers only. This can be done using mathematical operations with the rand function. Random numbers that are distributed in a range (a,b) can be obtained by multiplying rand by (b – a) and adding the product to a:

$$(b - a) \times rand + a$$

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 =

    7.2209     8.5869    -3.0952     8.7006     4.4854    -3.5369

   -0.8225     3.2032     9.3626     9.4733
```

1.10.3.2 The randi command:

The randi command generates uniformly distributed random integer. The command can be used to assign these numbers to a scalar, a vector, or a matrix, as shown in Table 1.10.

Table 1.10: The randi command

Command	Description	Example

`randi(imax)` (imax is an integer)	Generates a single random number between 1 and imax.	`>> a=randi(15)`
		`a =`
		`9`

`randi(imax, n)`	Generates an n × n matrix with random integers between 1 and imax.	`>> b=randi(15,3)`
		`b =`
		`4 8 11`
		`14 3 8`
		`1 15 8`

`randi(imax, m,n)`	Generates an m × n matrix with random integers between 1 and imax.	`>> c=randi(15,2,4)`
		`c =`
		`1 1 8 13`
		`11 2 2 13`

The range of the random integers can be set to be between any two integers by typing [imin imax] instead of imax. For example 3 × 4, a matrix with random integers between 50 and 90 is created by:

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

1.10.3.3 The randn command:

The randn command generates normally distributed numbers with mean 0 and standard deviation of 1. The command can be used to generate a single number, a vector, or a matrix in the same way as the rand command.

For example, a 3 × 4 matrix is created by:

```
>> 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
```

The mean and standard deviation of the numbers can be changed by mathematical operations to have any values. This is done by multiplying the number generated by the randn function by the desired standard deviation, and adding the desired mean. For example, a vector of six numbers with a mean of 50 and standard deviation of 6 is generated by:

```
>> v=4*randn(1,6)+50

v =

   47.6467   58.7327   49.4544   50.4557   54.2671   50.2371
```

Integers of normally distributed numbers can be obtained by using the round function.

```
>> w=round(4*randn(1,6)+50)

w =

    50    47    51    45    53    56
```