MATLAB - ARITHMETIC OPERATIONS

MATLAB allows two different types of arithmetic operations −

- Matrix arithmetic operations
- Array arithmetic operations

Matrix arithmetic operations are same as defined in linear algebra. Array operations are executed element by element, both on one dimensional and multi-dimensional array.

The matrix operators and arrays operators are differentiated by the period . symbol. However, as the addition and subtraction operation is same for matrices and arrays, the operator is same for both cases.

The following table gives brief description of the operators −

Operator	Description
+	Addition or unary plus. A+B adds the values stored in variables A and B. A and B must have the same size, unless one is a scalar. A scalar can be added to a matrix of any size.
-	Subtraction or unary minus. A-B subtracts the value of B from A. A and B must have the same size, unless one is a scalar. A scalar can be subtracted from a matrix of any size.
*	Matrix multiplication. C = A*B is the linear algebraic product of the matrices A and B. More precisely, $$C(i,j) = \sum_{k=1}^{n} A(i,k)B(k,j)$$ For non-scalar A and B, the number of columns of A must be equal to the number of rows of B. A scalar can multiply a matrix of any size.
.*	Array multiplication. A.*B is the element-by-element product of the arrays A and B. A and B must have the same size, unless one of them is a scalar.
/	Slash or matrix right division. B/A is roughly the same as B*inv$A$. More precisely, B/A = $A'\backslash B'$'.
./	Array right division. A./B is the matrix with elements Ai,j/Bi,j. A and B must have the same size, unless one of them is a scalar.
\	Backslash or matrix left division. If A is a square matrix, A\B is roughly the same as inv$A$*B, except it is computed in a different way. If A is an n-by-n matrix and B is a column vector with n components, or a matrix with several such columns, then X = A\B is the solution to the equation $AX = B$. A warning message is displayed if A is badly scaled or nearly singular.
.\	Array left division. A.\B is the matrix with elements Bi,j/Ai,j. A and B must have the same size, unless one of them is a scalar.
^	Matrix power. X^p is X to the power p, if p is a scalar. If p is an integer, the power is computed by repeated squaring. If the integer is negative, X is inverted first. For other values of p, the calculation involves eigenvalues and eigenvectors, such that if [V,D] = eig$X$, then X^p = V*D.^p/V.
.^	Array power. A.^B is the matrix with elements Ai,j to the Bi,j power. A and B must

	have the same size, unless one of them is a scalar.
'	Matrix transpose. A' is the linear algebraic transpose of A. For complex matrices, this is the complex conjugate transpose.
.'	Array transpose. A.' is the array transpose of A. For complex matrices, this does not involve conjugation.

Example

The following examples show the use of arithmetic operators on scalar data. Create a script file with the following code:

```
a = 10;
b = 20;
c = a + b
d = a - b
e = a * b
f = a / b
g = a \ b
x = 7;
y = 3;
z = x ^ y
```

When you run the file, it produces the following result −

```
c =  30
d = -10
e =  200
f =  0.50000
g =  2
z =  343
```

Functions for Arithmetic Operations

Apart from the above-mentioned arithmetic operators, MATLAB provides the following commands/functions used for similar purpose −

Function	Description
uplus$a$	Unary plus; increments by the amount a
plus a, b	Plus; returns a + b
uminus$a$	Unary minus; decrements by the amount a
minusa, b	Minus; returns a - b
timesa, b	Array multiply; returns a.*b
mtimesa, b	Matrix multiplication; returns a* b
rdividea, b	Right array division; returns a ./ b
ldividea, b	Left array division; returns a.\ b
mrdivideA, B	Solve systems of linear equations $xA = B$ for x
mldivideA, B	Solve systems of linear equations $Ax = B$ for x
powera, b	Array power; returns a.^b
mpowera, b	Matrix power; returns a ^ b

cumprod$A$	Cumulative product; returns an array of the same size as the array A containing the cumulative product. • If A is a vector, then cumprod$A$ returns a vector containing the cumulative product of the elements of A. • If A is a matrix, then cumprod$A$ returns a matrix containing the cumulative products for each column of A. • If A is a multidimensional array, then cumprod$A$ acts along the first non-singleton dimension.
cumprod$A$, *dim*	Returns the cumulative product along dimension *dim*.
cumsum$A$	Cumulative sum; returns an array A containing the cumulative sum. • If A is a vector, then cumsum$A$ returns a vector containing the cumulative sum of the elements of A. • If A is a matrix, then cumsum$A$ returns a matrix containing the cumulative sums for each column of A. • If A is a multidimensional array, then cumsum$A$ acts along the first nonsingleton dimension.
cumsum$A$, *dim*	Returns the cumulative sum of the elements along dimension *dim*.
diff$X$	Differences and approximate derivatives; calculates differences between adjacent elements of X. • If X is a vector, then diff$X$ returns a vector, one element shorter than X, of differences between adjacent elements: [X2-X1 X3-X2 ... Xn-X$n-1$] • If X is a matrix, then diff$X$ returns a matrix of row differences: [X$2:m, :$-X$1:m-1, :$]
diff$X$, n	Applies *diff* recursively n times, resulting in the nth difference.
diff$X$, n, *dim*	It is the nth difference function calculated along the dimension specified by scalar dim. If order n equals or exceeds the length of dimension dim, diff returns an empty array.
prod$A$	Product of array elements; returns the product of the array elements of A. • If A is a vector, then prod$A$ returns the product of the elements. • If A is a nonempty matrix, then prod$A$ treats the columns of A as vectors and returns a row vector of the products of each column. • If A is an empty 0-by-0 matrix, prod$A$ returns 1. • If A is a multidimensional array, then prod$A$ acts along the first non-singleton dimension and returns an array of products. The size of this dimension reduces to 1 while the sizes of all other dimensions remain the same. The prod function computes and returns B as single if the input, A, is single. For all other numeric and logical data types, prod computes and returns B as double.
prod$A$, *dim*	Returns the products along dimension dim. For example, if A is a matrix, prod$A$, 2 is a column vector containing the products of each row.

prod ___,datatype	multiplies in and returns an array in the class specified by datatype.
sum*A*	• Sum of array elements; returns sums along different dimensions of an array. If A is floating point, that is double or single, B is accumulated natively, that is in the same class as A, and B has the same class as A. If A is not floating point, B is accumulated in double and B has class double. • If A is a vector, sum*A* returns the sum of the elements. • If A is a matrix, sum*A* treats the columns of A as vectors, returning a row vector of the sums of each column. • If A is a multidimensional array, sum*A* treats the values along the first non-singleton dimension as vectors, returning an array of row vectors.
sum*A*, *dim*	Sums along the dimension of *A* specified by scalar *dim*.
sum. . . ,'*double*'	Perform additions in double-precision and return an answer of type double, even if A has data type single or an integer data type. This is the default for integer data types.
sum. . . , *dim*,'*double*'	
sum. . . ,'*native*'	Perform additions in the native data type of A and return an answer of the same data type. This is the default for single and double.
sum. . . , *dim*,'*native*'	
ceil*A*	Round toward positive infinity; rounds the elements of A to the nearest integers greater than or equal to A.
fix*A*	Round toward zero
floor*A*	Round toward negative infinity; rounds the elements of A to the nearest integers less than or equal to A.
idivide*a*, *b*	Integer division with rounding option; is the same as a./b except that fractional quotients are rounded toward zero to the nearest integers.
idivide*a*, *b*,'*fix*'	
idivide*a*, *b*,'*round*'	Fractional quotients are rounded to the nearest integers.
idivide*A*, *B*,'*floor*'	Fractional quotients are rounded toward negative infinity to the nearest integers.
idivide*A*, *B*,'*ceil*'	Fractional quotients are rounded toward infinity to the nearest integers.
mod *X*, *Y*	Modulus after division; returns X - n.*Y where n = floor*X*. /*Y*. If Y is not an integer and the quotient X./Y is within round off error of an integer, then n is that integer. The inputs X and Y must be real arrays of the same size, or real scalars *provided*$Y = 0$. Please note − • mod*X*, 0 is X • mod*X*, *X* is 0 • mod*X*, *Y* for X~=Y and Y~=0 has the same sign as Y

rem X, Y	Remainder after division; returns X - n.*Y where n = fix$X. / Y$. If Y is not an integer and the quotient X./Y is within roundoff error of an integer, then n is that integer. The inputs X and Y must be real arrays of the same size, or real scalars$provided Y = 0$. Please note that: • rem$X, 0$ is NaN • remX, X for X~=0 is 0 • remX, Y for X~=Y and Y~=0 has the same sign as X.
round$X$	Round to nearest integer; rounds the elements of X to the nearest integers. Positive elements with a fractional part of 0.5 round up to the nearest positive integer. Negative elements with a fractional part of -0.5 round down to the nearest negative integer.