Introduction to MATLAB

MATLAB is a powerful computing system for handling the calculations involved in scientific and engineering problem. The name MATLAB means MATrix LABoratory. The system MATLAB is designed to make a matrix computations particularly easy , MATLAB language is high level matrix/array language with control flow statements functions, data structures input/output, and object oriented programming feature.

The uses of MATLAB

1. **Computation**
2. **Modeling and simulation**
3. **Data analysis**
4. **Graphics and visualization**
5. **Data acquisition**

Variables

In MATLAB the variables are :

1. **Scalar which is 1x1 matrix.**
2. **Vector which is (1xn) row vector (default)**
   Or column vector (nx1)
3. **Matrix (mxn) or (nxn)**

- **The name of variable is any combination of letter and digits starting with letter, the max. length of variable name is 31 character. The MATLAB is case sensitive (this mean its distinguishes between upper- and lower-case letters) in the names, commands, functions, and variables.**
- **The name must begin with a letter of the alphabet. After that, the name can contain letters, digits, and the underscore character (e.g., value_1), but it cannot have a space.**
- **To listed the variables in workspace the following commands are used**
  who
  whos
- **The variables can be cleared from the workspace with the commands**
  `clear` **this commands clear all the variables from the workspace**
  `clear all` **this command same as clear or**
  `clear variablename`
- **Names of built-in functions can, but should not, be used as variable names.**

- **The MATLAB statement has the form**
  `variable=expression` **or**
  `expression`
- **Local variables**
- **Global variables**

operators

operator	function	operator	function
+	addition	.*	Dot star
-	subtraction	./	Dot division
*	multiplication	'	Transpose and complex conjugate
/	Left division	.'	Transpose
\	Right division	,	Coma
^	power	;	semicolon
.^	Power of each elements of a vector or a matrix	:	colon

Order of precedence

MATLAB executes the calculations according to the order of precedence displayed below. This order is the same as used in most calculators.

Precedence	Mathematical Operation
First	Parentheses. For nested parentheses, the innermost are executed first.
Second	Exponentiation.
Third	Multiplication, division (equal precedence).
Fourth	Addition and subtraction.

In an expression that has several operations, higher-precedence operations are executed before lower-precedence operations. If two or more operations have the same precedence, the expression is executed from left to right. Parentheses can be used to change the order of calculations.

Constants

There are many constants are defined in MATLAB such as

MATLAB expression	constants
pi	π
I or j	$\sqrt{-1}$
eps	ϵ ,flooting point $\sim 10^{-52}$
inf	∞
NaN	Not a number

Note that the number in MATLAB are

- **Integer**
- **Real**
- **Complex**
- **Inf (result from the division by zeros)**
- **NaN (result from zero divided by zero)**

Adding some constants to workspace

If one have to add some common used constant to the work space , this can be done using m-file which the constant is saved and run this file to add these constants to workspace.

Format

The results can be appear in a certain format using the command

Command	Description	Example
format short	Fixed-point with 4 decimal digits	>> 351/7 ans = 50.1429
format long	Fixed-point with 14 decimal digits	>> 351/7 ans = 50.142857142857146
format short e	Scientific notation with 4 decimal digits	>> 351/7 ans = 5.0143e+001
format long e	Scientific notation with 15 decimal digits	>> 351/7 ans = 5.014285714285715e+001
format short g	Best of 5 digit fixed or floating point	>> 351/7 ans = 50.143
format long g	Best of 15 digit fixed or floating point	>> 351/7 ans = 50.1428571428571
format bank	Two decimal digits	>> 351/7 ans = 50.14

Functions

The MATLAB functions are

- **Built in functions ; like sqrt, abs , trigonometric function , log. Function , exponential function , hyperbolic function ,..**
- **Specialized mathematical functions ; like Arry function, Bessel functions, error function, gamma and beta functions,..etc**
- **User defined functions, like m-file and inline function**

ELEMENTARY MATH BUILT-IN FUNCTIONS

MATLAB has a very large library of built-in functions. A function has a name and an argument in parentheses. For example, the function that calculates the square root of a number is sqrt(x). Its name is sqrt, and the argument is x.

Elementary Built in functions

Function	Description	Example
`sqrt(x)`	**Square root.**	`>> sqrt(81)` `ans = 9`
`nthroot(x,n)`	**Real *n*th root of a real number *x*.** **(If *x* is negative *n* must be an odd integer.)**	`>> nthroot(80,5)` `ans = 2.4022`
`exp(x)`	**Exponential** (e^x)**.**	`>> exp(5)` `ans = 148.4132`
`abs(x)`	**Absolute value.**	`>> abs(-24)` `ans = 24`
`log(x)`	**Natural logarithm. Base *e* logarithm (ln).**	`>> log(1000)` `ans = 6.9078`
`log10(x)`	**Base 10 logarithm.**	`>> log10(1000)` `ans = 3.0000`
`factorial(x)`	**The factorial function** $x!$ **(*x* must be a positive integer.)**	`>> factorial(5)` `ans = 120`
`prod(x)`	**Product of x**	`>> prod([1 2 3 4 5 ])` `ans = 120`

Trigonometric math functions

Function	Description	Example

`sin(x)`
`sind(x)`
	Sine of angle *x* (*x* in radians).
	Sine of angle *x* (*x* in degrees).
		`>> sin(pi/3)`
		`ans =`
		`0.8660`

`cos(x)`
`cosd(x)`
	Cosine of angle *x* (*x* in radians).
	Cosine of angle *x* (*x* in degrees).
		`>> cosd(60)`
		`ans =`
		`0.5000`

`tan(x)`
`tand(x)`
	Tangent of angle *x* (*x* in radians).
	Tangent of angle *x* (*x* in degrees).
		`>> tan(pi/6)`
		`ans =`
		`0.5774`

`cot(x)`
`cotd(x)`
	Cotangent of angle *x* (*x* in radians).
	Cotangent of angle *x* (*x* in degrees).
		`>> cotd(30)`
		`ans =`
		`1.7321`

`asin(x)`	**Compute inverse sine of x, where x must be between -1 and 1, the function return an angle in radians between** $-\pi/2$ **and** $\pi/2$	
`acos(x)`	**Compute inverse cosine of x, where x must be between -1 and 1, the function return an angle in radians between 0 and** π	
`atan(x)`	**Compute inverse tangent of x, where x must be between -1 and 1, the function return an angle in radians between** $-\pi/2$ **and** $\pi/2$	

Hyperbolic functions

Function	Description	Example
`sinh(x)`	Compute hyperbolic sine of x	
`cosh(x)`	Compute hyperbolic cosine of x	
`tanh(x)`	Compute hyperbolic tangent of x	
`sech(x)`	Compute hyperbolic secant of x	
`csch(x)`	Compute hyperbolic cosecant of x	
`coth(x)`	Compute hyperbolic cotangent of x	
`asinh(x)`	Compute the inverse of hyperbolic sine of x	
`acosh(x)`	Compute the inverse of hyperbolic cosine of x	
`atanh(x)`	Compute the inverse of hyperbolic tangent of x	

Rounding functions

Function	Description	Example

`round(x)`	Round to the nearest integer.	`>> round(17/5)`
		`ans =`
		`3`

`fix(x)`	Round toward zero.	`>> fix(13/5)`
		`ans =`
		`2`

`ceil(x)`	Round toward infinity.	`>> ceil(11/5)`
		`ans =`
		`3`

`floor(x)`	Round toward minus infinity.	`>> floor(-9/4)`
		`ans =`
		`-3`

`rem(x,y)`	Returns the remainder after *x* is divided by *y*.	`>> rem(13,5)`
		`ans =`
		`3`

`sign(x)`	Signum function. Returns 1 if x>0 , , –1 if x<0 , and 0 if x=0.	`>> sign(5)`
		`ans =`
		`1`

Complex number functions

Function	Description	Example
`conj(x)`	Compute the complex conjugate of a number x	
`angle(x)`	Compute the angle between real and imaginary part of x	
`real(x)`	Compute the real part of a number x	
`imag(x)`	Compute the complex part of a number x	
`abs(x)`	Compute the absolute value of magnitude of the complex number x	

vector and matrices

A vector is a list of numbers in row or column (one dimensional matrix) or array, while the matrix is two dimensional array in which the data arranged in row and column and obey some mathematical operations. The matrix (or vector) can be enter to MATLAB in several different ways

1. **Enter an explicit list of elements**
2. **Load matrix from external data file**
3. **Generate matrix using built in function**
4. **Generate matrix with your own function or m-file**

Vector is a special type of the matrix having only one row (or one column), to define the vector in MATLAB :

- ***Initializing vectors, explicit list*:**
  There are several ways to create row vector variables. The most direct way is to put the values that you want in the vector in square brackets, separated by either spaces or commas. For example,

```
x=[1  5  0  7  -1]
disp(x)
x=[1,5,0, 7,-1]
disp(x)
```

  Example:

  Vector variables can also be created using existing variables. For example, a new vector c is created here consisting first of all the values from *a* followed by all values from *b* :

```
a=[1  2  3]; b=[4  5  6];
c=[a  b]
d=[a'  b']

y=[]  this is empty vector  (0x0 matrix)
```

- ***Initializing vectors, colon operator:***
  If the values in the vector are regularly spaced, the *colon operator* can be used to *iterate* through these values. For example, 1:10 results in all the integers from 1 to 10:

```
X1=1:10
```

  Note that in this case, the brackets [ ] are not necessary to define the vector.
  With the colon operator, a *step value* can also be specified with another colon, in the form (first:step:last). For example, to create a vector from 1 to 2 in steps of 0.1:

```
X2=1: 0.1: 2;
X3=10:-1:5
X4=1:0  empty matrix
```

- ***Initializing vectors, using functions:***
  Similarly, the linspace function creates a linearly spaced vector; linspace(x,y,n) creates a vector with n values in the inclusive range from x to y. For example, the following creates a vector with five values linearly spaced between 3 and 15, including the 3 and 15:

```
s = linspace(3,15,5)
s=
3  6  9  12  15
```

```
X=linspace(0,pi/2,10)
```
X is a vector of 10 equally spaced points from 0 to $\pi/2$ (inclusive).

```
X1=logspace(0,2,20)
```
X1 is a vector of 20 elements from 10^0 to 10^2

MATLAB also has several functions that create special matrices. For example,the zeros function creates a matrix of all zeros, either one argument can be passed (which will be both the number of rows and columns), or Two arguments (first the number of rows and then the number of columns). Also there is ones function

```
>> Y=zeros(3,1)
Y=
     0
     0
     0

>> Y1=zeros(1,5)
Y1 =
     0     0     0     0     0

>> X=ones(4)
X=
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
```

Matrices of random numbers can be created using the rand and randint functions. The first two arguments to the randint function specify the size of the matrix of random integers. For example, the following will create a 2 × 4 matrix of random integers, each in the range from 10 to 20:

```
>> x=randint(2,4,[10,20])
x =
    13    11    17    20
    10    19    13    10

>> x=rand(2,4)
x =
    0.4387    0.7655    0.1869    0.4456
    0.3816    0.7952    0.4898    0.6463

>> x=randint(1,10,[1,10])
x =
     4     2     3     7     5     4     9     6     6    10

>> Z=rand(1,7)
Z =
    0.7094    0.7547    0.2760    0.6797    0.6551    0.1626    0.1190
```

Transposing vector

```
X=1:4 ; Y=X';
X=1  2  3  4
Y=
      1
      2
      3
      4
```

Subscripts

X(2)

X(1:2:5)

X([5 4 2 1])

Creating Matrix Variables

Creating a matrix variable is really just a generalization of creating row and column vector variables. That is, the Values within a row are separated by either spaces or commas, and the different rows are separated by semicolons. For example, the matrix variable *A* is created by explicitly typing values:

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

The length and size functions in MATLAB are used to find array dimensions. The length function returns the number of elements in a vector. The size function returns the number of rows and columns in a matrix. For a matrix, the length function will return either the number of rows or the number of columns, whichever is largest. For example, the following vector, *B* , has four elements so its length is 4. It is a row vector, so the size is 1 × 4.

```
>> B=-2:1
B =
    -2    -1     0     1

>> length(B)

ans =

     4

>> size(B)

ans =

     1     4
```

Changing Dimensions

In addition to the transpose operator, MATLAB has several built-in functions that change the dimensions or configuration of matrices, including reshape, fliplr, flipud, and rot90. The reshape function changes the dimensions of a matrix. The following matrix variable A is 3× 4, or in other words it has 12 elements.

```
>> A=[1 4 7 10;2 5 8 11;3 6 9 12]
A =
     1     4     7    10
     2     5     8    11
     3     6     9    12
```

```
>> reshape(A,1,12)
ans =
     1     2     3     4     5     6     7     8     9    10    11    12
```

The reshape function iterates through the matrix columnwise. For example, when reshaping *A* into a 2 6 matrix, the values from the first column in the original matrix (1, 2, 3) are used first, then the values from the second column (4, 5,6), and so forth.

```
>> reshape(A,2,6)

ans =

     1     3     5     7     9    11

     2     4     6     8    10    12
```

The rot90 function rotates the matrix counterclockwise 90 degrees, so for example the value in the top-right corner becomes instead the top-left corner and the last column becomes the first row:

```
>> A=[1 4 7 10;2 5 8 11;3 6 9 12]
A =
     1     4     7    10
     2     5     8    11
     3     6     9    12
>> rot90(A,1)
ans =
    10    11    12
     7     8     9
     4     5     6
     1     2     3
```

The fliplr function "flips" the matrix from left to right (in other words the left-most column, the first column, becomes the last column and so forth), and the flipud functions flips up to down. Note that in these examples *A* is unchanged; instead, the results are stored in the default variable *ans* each time.

```
A=[1 4 7 10;2 5 8 11;3 6 9 12]
A =
     1     4     7    10
     2     5     8    11
     3     6     9    12
fliplr(A)
ans =
    10     7     4     1
    11     8     5     2
    12     9     6     3
flipud(A)
ans =
     3     6     9    12
     2     5     8    11
     1     4     7    10
```

The function repmat can also be used to create a matrix; repmat(mat,m,n) creates a larger matrix, which consists of an *m* × *n* matrix of copies of mat. For example, here is a 2 × 2 random matrix:

```
a=[1 2;3 4]
a =
     1     2
     3     4
A=repmat(a,2,2)
A =
     1     2     1     2
     3     4     3     4
     1     2     1     2
     3     4     3     4
```

Addressing Arrays

A colon can be used in MATLAB to address a range of elements in a vector or a matrix.

- ***Colon for a vector***

  Va(:) – refers to all the elements of the vector Va (either a row or a column vector).
  Va(*m*:*n*) – refers to elements *m* through *n* of the vector Va.
  For instance,

```
V = [2  5  −1  11  8  4  7  −3  11]

u = V (2 :8)

u = 5  −1  11  8  4  7  −3  11
```

- ***Colon for a matrix***

  The following Table gives the use of a colon in addressing arrays in a matrix.

Function	Description
A(:, *n*)	Refers to the elements in all the rows of a column *n* of the matrix *A*.
A(*n*, :)	Refers to the elements in all the columns of row *n* of the matrix *A*.
A(:, *m*:*n*)	Refers to the elements in all the rows between columns *m* and *n* of the matrix *A*.
A(*m*:*n*, :)	Refers to the elements in all the columns between rows *m* and *n* of the matrix *A*.
A(*m*:*n*, *p*:*q*)	Refers to the elements in rows *m* through *n* and columns *p* through *q* of the matrix *A*.

Adding Elements to a Vector or a Matrix

A variable that exists as a vector or a matrix can be changed by adding elements to it. Addition of elements is done by assigning values of the additional elements, or by appending existing variables. Rows and/or columns can be added to an existing matrix by assigning values to the new rows or columns.

V=[1 2 3 4 5 5]

V(3:5)=[1 2 3]

Deleting Elements

An element or a range of elements of an existing variable can be deleted by reassigning blanks to these elements. This is done simply by the use of square brackets with nothing typed in between them.

V(3)=[]

V(1:2)=[]

Some function with matrices

Function	Description	Example
`sum(x)`	If *x* is a vector, returns the sum of the elements of the vector.	`x=1:20;` `sum(x)`

`C=max(x)`
 	
`[d,n]=max(x)`
	If *x* is a vector, *C* is the largest element in *x*.
	If *x* is a matrix, *C* is a row vector containing the largest element of each column of *x*.
	
	If *x* is a vector, *d* is the largest element in *x*, *n* is the position of the element (the first if several have the max value).	

`C=min(x)` `[d,n]=min(x)`	The same as max(x), but for the smallest element. The same as [d, n] = max(x), but for the smallest element.	
`mean(x)`	If *x* is a vector, returns the mean value of the elements	
`sort(x)`	If *x* is a vector, arranges the elements of the vector in ascending order.	
`diag(x)` `diag(x,k)`	Creates a diagonal matrix from the vector x. Creates a vector from the diagonal of a matrix	
`trace(x)`	Sum of the diagonal of a matrix x	
`fliplr(x)`	Flip the vector (or matrix) x left to right	
`flipud(x)`	Flip the vector (or matrix) x up to down	
`tril(x)`	Extract lower tridiagonal part	
`triu(x)`	Extract upper tridiagonal part	
`rot90(A,k)`	Rotate the matrix A counterclockwise by k*90	
`reshape(A,m,n)`	Rearrange a matrix that *A* has *r* rows and *s* columns to have *m* rows and *n* columns. *r* times *s* must be equal to *m* times *n* .	

`transpose(x)`
`x'`
`x.'`
	Transpose the matrix x
	Transpose and complex conjugate of a matrix x
	Transpose of a matrix x	

`det(A)`	Returns the determinant of a square matrix *A*.	
`inv(A)`	Returns the inverse of a square matrix *A*.	
`eye(m,n)`	Creates a $(m \times n)$ unit matrix.	
`ones(m,n)`	Creates a $(m \times n)$ matrix with ones	
`zeros(m,n)`	Creates a $(m \times n)$ matrix with zeros	
`linspace(xi,xf,n)`	Create a linear of n equal space from xi to xf .	
`logspace(d1,d2,n)`	Create a logarithm space of n elements from 10^{d1} to 10^{d2}	
`dot(a, b)`	Calculates the scalar (dot) product of two vectors *a* and *b*. The vector can each be row or column vectors.	
`cross(a, b)`	Calculates the cross product of two vectors *a* and *b*, (*a* × *b*). The two vectors must have 3 elements.	

The `disp` *Command*

The `disp` command is used to display the elements of a variable without displaying the name of the variable, and to display text. The format of the `disp` command is:

`disp(name of a variable)` or `disp('text as string')`

- Every time the `disp` command is executed, the display it generates appears in a new line. One example is:

```
>balance = 1000;
>rate = 0.09;
>interest = rate * balance;
>balance = balance + interest;
>disp( 'New balance:' );
>disp( balance );
```

When you press Enter to run it, you should get the following output in the Command Window:

```
New balance:
1090
```

- Only one variable can be displayed in a `disp` command. If elements of two variables need to be displayed together, a new variable (that contains the elements to be displayed) must first be defined and then displayed.

The `fprintf` *Command*

The `fprintf` command can be used to display output (text and data) on the screen or to save it to a file. With this command (unlike with the `disp` command) the output can be formatted. For example, text and numerical values of variables can be intermixed and displayed in the same line. In addition, the format of the numbers can be controlled.

With many available options, the `fprintf` command can be long and complicated. To avoid confusion, the command is presented gradually. First, this section shows how to use the command to display text messages, then how to mix numerical data and text, next how to format the display of numbers, and finally how to save the output to a file

- The **fprintf** statement is much more flexible (and therefore more complicated) than disp. For example it allows you to mix strings and numbers freely on the same line, and to control the format completely.
- The general form of **fprintf** is `fprintf( 'format string', list of variables )`

Same example

```
balance = 12345;
rate = 0.09;
interest = rate * balance;
balance = balance + interest;
fprintf( 'Interest rate: %6.3f New balance:
%8.2f\n', ... rate, balance );
```

When you run this, your output should look like this:

```
Interest rate: 0.090 New balance: 13456.05
```

Output may be sent to a disk file with `fprintf`. The output is *appended*, i.e. added on at the end of the file.
The general form is
`fprintf( 'filename', 'format string', list of variables )`
e.g.
`fprintf( 'myfile', '%g', x )`
sends the value of `x` to the disk file `myfile`.