

MATLAB: *The basics*

In this tutorial, the reader will learn about how to use operators, create variables, basic mathematical operation, and matrix operations.

Variables

Recall that MATLAB defines all variables as matrices. There is no declaration statement needed to define the variables. The declaration and assignment of values occur at the same time.

The user should be also be aware that MATLAB is case sensitive, which means “a ≠ A”

Once a variable is assigned, typing the variable name and pressing ENTER will display its value.

The Command Window can be used a calculator, i.e. the user can perform any calculations even without assigning a variable name. MATLAB will give it a default name “ans”

Before start defining variables, the user must be introduced to MATLAB operators. Next, most important and common MATLAB operators are introduced, then we return to variables.

Operators

1- Matrix operators

Operators	Description
+	Addition
-	Subtraction
*	Multiplication
^	Power
'	Conjugate transpose
/ or \	Matrix division
./ or .\	Array division

2- Relational and logical operator

Operator	Description	Notes
<	Less than	Compares real part only
<=	Less than or equal	Compares real part only
>	More than	Compares real part only
>=	More than or equal	Compares real part only
==	Equal	Compares both real and imaginary
~=	Not equal	Compares both real and imaginary
&	AND	
	OR	Shift and \
~	NOT	Shift and ` “first key left to 1”

Note that “==” is used for relation, while “=” is used for assignment.

3- Special Characters

Character	Description	Notes/Examples
[]	Used to form vectors or matrices	t=[1 1 1]
()	Function Argument or precedence	max(t); 2*(3+3)
,	Separate subscripts and arguments	zeros(2,2); t(1,1)
;	End matrix row, Echo off	g=[1 2 3;4 5 6]
:	Subscripting, vector generation	t=1:5
!	Execute operating system command	!dir & → open command prompt
%	Comment	Lines starting with “%” not executed

Concept introduction and Examples:

Matrix/Vector Creation

- 1- Create variable called **a** and assign 5 to its value

```
>> a=5

a =

    5
```

Note that when you press enter, MATLAB will repeat the statement. This is called ECHO. To suppress the echo or turn it off, place semicolon after the statement, such as:

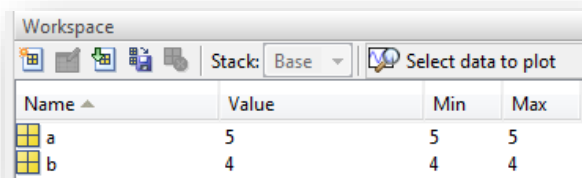
```
>> a=5;

a =

    5

>> b=4;
fx >> |
```

Note that, regardless of using semicolon or not MATLAB creates the variable. To make sure, inspect the Workspace



Name	Value	Min	Max
a	5	5	5
b	4	4	4

2- Create a row vector \mathbf{g} that contains the values from 2 to 7. There are few ways to perform this operation. Some of them may be an over kill for this simple operation.

a. Using colon, such as (take a look at the Workspace)

```
>> g=2:7  
  
g =  
  
     2     3     4     5     6     7
```

b. Using colon operator and defining the step size, such as

```
>> g=2:1:7  
  
g =  
  
     2     3     4     5     6     7
```

The step size can be a positive or negative number. For example to create a row vector \mathbf{g} with values from 10 to 1 in descending order, such as

```
>> g=10:-1:1  
  
g =  
  
    10     9     8     7     6     5     4     3     2     1
```

Note: to suppress the echo, terminate each statement with semicolon operator.

c. Another way to create the \mathbf{g} row vector is using `linspace(start,end,elements)` function, such as

```
>> g=linspace(2,7,6)  
  
g =  
  
     2     3     4     5     6     7
```

- 3- Create t column vector with values from 2 to 6.

```
>> t=[2:5]'
```

```
t =
```

```
2
3
4
5
```

```
>> t=(2:5)'
```

```
t =
```

```
2
3
4
5
```

- 4- Access all the elements of t column vector and all the elements of g row vector

```
>> t(:)
```

```
ans =
```

```
2
3
4
5
```

```
>> g(1,:)
```

```
ans =
```

```
2 3 4 5 6 7
```

- 5- Create the following matrix in MATLAB

$$A = \begin{bmatrix} 1.2 & 10 & 15 \\ 3 & 5.5 & 2 \\ 4 & 6.8 & 7 \end{bmatrix}$$

- a. Using the `[]` and semicolon operators. Note that columns are separated by spaces and rows are separated by semicolon, such as

```
>> A=[1.2 10 15;3 5.5 2; 4 6.8 7]
```

```
A =
```

```
1.2000    10.0000    15.0000
3.0000     5.5000     2.0000
4.0000     6.8000     7.0000
```

- b. Column separated by spaces and rows are inserted on different line, such as

```
>> A =[1.2 10 15
      3 5.5 2
      4 6.8 7]

A =

    1.2000    10.0000    15.0000
    3.0000     5.5000     2.0000
    4.0000     6.8000     7.0000
```

- 6- Create a **B** row vector consists of row #2 from **A** matrix above, and create **C** column vector consists of column #3 from **A** matrix.

```
>> B=A(2, :)

B =

    3.0000    5.5000    2.0000
```

```
C =

    15
     2
     7
```

- 7- Create a **D** 2x2 matrix from **A** matrix above.

```
>> D=A(1:2, 1:2)

D =

    1.2000    10.0000
    3.0000     5.5000
```

- 8- Create a matrix **F** consist of **A** matrix and a forth column of zeros

```
>> F=[A zeros(3,1)]

F =

    1.2000    10.0000    15.0000     0
    3.0000     5.5000     2.0000     0
    4.0000     6.8000     7.0000     0
```

- 9- Create a column vector E from A matrix consisting of elements of the first column, second column and third column.

```
>> E=A(:,1:3)
E =
    1.2000
    3.0000
    4.0000
   10.0000
    5.5000
    6.8000
   15.0000
    2.0000
    7.0000
```

- 10- Create a row vector G consists of the transpose of E vector

```
>> G=E'
G =
    1.2000    3.0000    4.0000   10.0000    5.5000    6.8000   15.0000    2.0000    7.0000
```

- 11- Enter the following complex number in MATLAB

$$x = 1 + i\sqrt{3}$$

```
>> x=1+i*sqrt(3)
x =
    1.0000 + 1.7321i
```

```
>> x=1+j*sqrt(3)
x =
    1.0000 + 1.7321i
```

Recall that any imaginary number has two parts, a real part and an imaginary part. To call out either use *real()* or *imag()* functions

```
>> real(x)
ans =
     1
```

```
>> imag(x)
ans =
    1.7321
```

To find the complex modulus (magnitude), use ***abs()*** function. The same function can be used to find the absolute of real number.

```
>> abs(x)

ans =

    2.0000
```

One additional property of a complex number is the conjugate. To calculate the conjugate of x, use ***conj()*** or **apostrophe (prime)** operator

```
>> conj(x)

ans =

    1.0000 - 1.7321i
```

```
>> x'

ans =

    1.0000 - 1.7321i
```

Note: the user can either use i or j to represent the complex operator or $\sqrt{-1}$.

Note: for complex matrices:

conj() function yields the conjugate of the matrix.

```
>> X=[1 j;-j*5 2]

X =

    1.0000          0 + 1.0000i
    0 - 5.0000i    2.0000
```

```
>> Y=conj(X)

Y =

    1.0000          0 - 1.0000i
    0 + 5.0000i    2.0000
```

dot prime operator yields the transpose of a complex matrix.

```
>> Y=X.'

Y =

    1.0000          0 - 5.0000i
    0 + 1.0000i    2.0000
```

prime operator creates the complex conjugate transpose.

```
>> Y=X'

Y =

    1.0000          0 + 5.0000i
    0 - 1.0000i    2.0000
```

Utility Matrices

12- Create 3x3 Identity matrix. Use *eye()* function.

```
>> I=eye(3)

I =

     1     0     0
     0     1     0
     0     0     1
```

Note that there is only one argument of the *eye()* function, because the Identity matrix is nxn square matrix.

13- Create a 3x3 matrix full of ones. Use *ones(n,m)* function, where n is the number row and m is the number of columns.

```
>> H=ones(3,3)

H =

     1     1     1
     1     1     1
     1     1     1
```

Note that if you enter only one argument, MATLAB will generate a square matrix of that size. For example,

```
>> K=ones(4)

K =

     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
```

Using the same logic, you can create nxm matrix full of zeros by using *zeros(n,m)* function, where n is the number row and m is the number of columns.

14- Use the *diag()* function to create an 3x3 Identity matrix.

```
>> M=diag(ones(1,3))  
  
M =  
  
    1    0    0  
    0    1    0  
    0    0    1
```

15- Use the *diag()* function to extract a vector contains the diagonal elements of the following square matrix

$$T = \begin{bmatrix} 1.2 & 10 & 15 \\ 3 & 5.5 & 2 \\ 4 & 6.8 & 7 \end{bmatrix}$$

```
>> c=diag(T)  
  
c =  
  
    1.2000  
    5.5000  
    7.0000
```

16- Create a square matrix with diagonal consists of sequence from 1 to 4 and all other elements are zeros.

```
>> F=diag(1:4)  
  
F =  
  
    1    0    0    0  
    0    2    0    0  
    0    0    3    0  
    0    0    0    4
```

17- Use the *diag()* function to create 5x5 zeros matrix.

```
>> V=diag(0,4)  
  
V =  
  
    0    0    0    0    0  
    0    0    0    0    0  
    0    0    0    0    0  
    0    0    0    0    0  
    0    0    0    0    0
```

Note that *diag(0,n)* creates an $(n+1) \times (n+1)$ matrix.

Trickery:

I. Command line editing keypresses:

Key	Control Equivalent	Operation
Up arrow	Ctrl+p	Recall previous line
Down arrow	Ctrl+n	Recall next line
Left arrow	Ctrl+b	Back one character
Right arrow	Ctrl+f	Forward one character
Ctrl left arrow	Ctrl+l	Left one word
Ctrl right arrow	Ctrl+r	Forward one word
Home	Ctrl+a	Beginning of line
Esc	Ctrl+u	Clear line
End	Ctrl+e	End of line
Del	Ctrl+d	Delete character under cursor
	Ctrl+k	Kill current operation

II. To list all variables in the workspace use *who* command

```
>> who

Your variables are:

A B C D E F G H I K M T V X Y ans c g t x y
```

III. To list current variables in the workspace with information about the size, number of bytes and class. Use *whos* command.

```
>> whos

Name      Size      Bytes  Class  Attributes

A         3x3         72  double

B         1x3         24  double

C         3x1         24  double

D         2x2         32  double

E         9x1         72  double

F         4x4        128  double

G         1x9         72  double
```

IV. To display today's date, use date command.

```
>> date

ans =

02-Oct-2011
```