MATLAB Tuturial

1 MATLAB Desktop

1.1 Command Window

This is the primary place where you interact with MATLAB. The prompt >> is displayed in this window, and when this window is active, a blinking cursor appears to right of the prompt. This cursor and prompt signify that MATLAB is waiting to perform a mathematical operation. In this window, you can enter variables and mathematical expressions, and run commands and MATLAB codes called M-files.

Practice: Enter

```
>> 1+1
```

and hit the enter key:

```
ans =
      2

>> x = 1
x =
      1

>> y = 1
y =
      1

>> z = x+y
z =
      2
>>
```

The expression $x = 1$ means that the value 1 is assigned to the variable x.

Multiple commands are separated by commas or semicolons, they can be placed on one line:

```
>> x = 1, y = 1;
x =
      1
>>
```

Commas tell MATLAB to display results; semicolons suppress printing.

1.2 Help Browser

Use the Help browser to search and view documentation and demos for all your MathWorks products. The Help browser is a Web browser integrated into the MATLAB desktop that displays HTML documents.

To open the Help browser, click the help button in the toolbar, or type helpbrowser in the Command Window.

The Help browser consists of two panes, the Help Navigator, which you use to find information, and the display pane, where you view the information.

The following are useful commands for online help:

- `help` lists topics on which help is available;
- `helpwin` opens the interactive help window;
- `helpdesk` opens the web browser based help facility;
- `help` *topic* provides help on *topic*;
- `lookfor` *string* lists help topics containing *string*;

1.3 Editor / Debugger

Use the Editor/Debugger to create and debug M-files, which are programs you write to run MATLAB functions. The Editor/Debugger provides a graphical user interface for basic text editing, as well as for M-file debugging.

You can use any text editor to create M-files, such as Emacs, and can use preferences (accessible from the desktop File menu) to specify that editor as the default. If you use another editor, you can still use the MATLAB Editor/Debugger for debugging, or you can use debugging functions, such as dbstop, which sets a breakpoint.

If you just need to view the contents of an M-file, you can display it in the Command Window by using the type function.

2 Simple Mathematics

2.1 Variables

MATLAB does not require any type declarations or dimension statements. When MATLAB encounters a new variable name, it automatically creates the variable and allocates the appropriate amount of storage. If the variable already exists, MATLAB changes its contents and, if necessary, allocates new storage. For example,

```
>> numOfSudents = 25
```

creates a 1-by-1 matrix named numOfSudents and stores the value 25 in its single element.

Variable names consist of a letter, followed by any number of letters, digits, or underscores. MATLAB uses only the first 31 characters of a variable name. MATLAB is case sensitive; it distinguishes between uppercase and lowercase letters. A and a are not the same variable. To view the matrix assigned to any variable, simply enter the variable name.

Variable names can contain up to 63 characters. Any characters beyond the 63rd are ignored.

2.2 Operators

Expressions use familiar arithmetic operators and precedence rules. Functions

+	Addition
-	Subtraction
*	Multiplication
/	Division
ˆ	Power
( )	Specify evaluation order

2.3 Build-in Functions

MATLAB provides a large number of standard elementary mathematical functions, including abs, sqrt, exp, and sin. Taking the square root or logarithm of a negative number is not an error; the appropriate complex result is produced automatically. MATLAB also provides many more advanced mathematical functions, including Bessel and gamma functions. Most of these functions accept complex arguments. For a list of the elementary mathematical functions, type

```
>> help elfun
```

For a list of more advanced mathematical and matrix functions, type

```
>>help specfun

>>help elmat
```

Some of the functions, like sqrt and sin, are built in. They are part of the MATLAB core so they are very efficient, but the computational details are not readily accessible. Other functions, like gamma and sinh, are implemented in M-files. You can see the code and even modify it if you want.

Several special functions provide values of useful constants.

pi	3.14159265...
i	Imaginary unit, $\sqrt{-1}$
j	Same as i
eps	Floating-point relative precision, 2^{-52}
realmin	Smallest floating-point number, 2^{-1022}
realmax	Largest floating-point number, 2^{1023}
Inf	Infinity
NaN	Not-a-number

Infinity is generated by dividing a nonzero value by zero, or by evaluating well defined mathematical expressions that *overflow*, i.e., exceed realmax. Not-a-number is generated by trying to evaluate expressions like 0/0 or Inf-Inf that do not have well defined mathematical values.

The function names are not reserved. It is possible to overwrite any of them with a new variable, such as

```
>> eps = 1.e-6
```

and then use that value in subsequent calculations. The original function can be restored with

```
>> clear eps
```

2.4 Mathematical Expressions

Like most other programming languages, MATLAB provides mathematical expressions, but unlike most programming languages, these expressions involve entire matrices. Here are a few examples, and the resulting values.

```
>> rho = (1+sqrt(5))/2
```

```
rho =
      1.6180

>> a = abs(3+4i)
a =
      5
>>
```

Sometimes expressions or commands are so long that it is convenient to continue them onto additional lines. In MATLAB, statement continuation is denoted by three periods in succession, as shown in the following code:

```
>> b = 10/ ...
2
b =
      5
>>
```

Exercises

Use MATLAB to evaluate the following expressions.

1. $\frac{1}{2\times 3}$.
2. $2^{2\times 3}$.
3. $1.5 \times 10^{-4} + 2.5 \times \sqrt{2}$.
4. $e^3 + \ln(e^3) - \log_{10}(10)$.
5. $\sin(\pi/3) + 2\frac{\cos(1)}{\tan(\pi/3)}$ (π is pi in MATLAB).

3 Vectors and Matrices

For matrices, go to MATLAB Help → Getting Started → Manipulating Matrices.

3.1 Generating vectors

A vector can be generated in different ways:

- Enter an explicit list of elements

```
>> v1 = [16 3 2 7]

v1 =

    16     3     2     7

>>
```

- Use the colon operator

```
>> v2 = 1:10

v2 =

     1     2     3     4     5     6     7     8     9    10

>>
```

- Use MATLAB function linspace

```
>> v3 = linspace(0,1,10)

v3 =

  Columns 1 through 7

         0    0.1111    0.2222    0.3333    0.4444    0.5556    0.6667

  Columns 8 through 10

    0.7778    0.8889    1.0000

>>
```

3.2 Generating matrices

A Matrix can be generated by entering an explicit list of elements

```
>> A = [16 3 2 1; 3 5 8 9]

A =

    16     3     2     1
     3     5     8     9

>>
```

MATLAB provides four functions that generate basic matrices.

zeros	All zeros
ones	All ones
rand	Uniformly distributed random elements
randn	Normally distributed random elements

Here are some examples.

```
>> B = zeros(2,4)

B =

     0     0     0     0
     0     0     0     0

>>

> F = 5*ones(3,3)

F =

     5     5     5
     5     5     5
     5     5     5

>>
```

3.3 Array Addressing or Indexing

The element in row i and column j of A is accessed by A(i,j).

```
>> A(1,3)

ans =

     2

>>
```

The element in row i and column j of B can be changed by

```
>> B(1,1) = 1

B =

     1     0     0     0
     0     0     0     0

>>
```

3.4 Arithmetic Operations on Arrays

- Transpose: A'

  ```
  >> C = A'

  C =

      16     3
       3     5
       2     8
       1     9

  >>
  ```

- Addition: A+B

  ```
  >> A+B
  ```

```
ans =

    17     3     2     1
     3     5     8     9

>>
```

- Multiplication: *

```
>> A'*B

ans =

    16     0     0     0
     3     0     0     0
     2     0     0     0
     1     0     0     0

>>
```

- Element-by-element multiplication: .*

```
>> B.*A

ans =

    16     0     0     0
     0     0     0     0

>>
```

- Element-by-element division: ./

```
>> B./A

ans =

    0.0625         0         0         0
         0         0         0         0
```

```
>>
```

- Operation between scalars and arrays: Addition, subtraction, multiplication, and division by a scalar simply apply the operation to all elements of the array.

```
>> B./A+1

ans =

    1.0625    1.0000    1.0000    1.0000
    1.0000    1.0000    1.0000    1.0000

>>
```

- Change the first column of B to 1:

```
>> B(:,1) = 10

B =

    10     0     0     0
    10     0     0     0

>>
```

- *Concatenation* is the process of joining small matrices to make bigger ones. The pair of square brackets, [ ], is the concatenation operator. For an example,

```
>> [A B]

ans =

    16     3     2     1    10     0     0     0
     3     5     8     9    10     0     0     0

>>
```

- *Deleting Rows and Columns*: You can delete rows and columns from a matrix using just a pair of square brackets. To delete the second column of A, use

```
>> A(:,2) = []

A =

    16     2     1
     3     8     9

>>
```

Exercises

1. Create the vector $v = [2\ 3\ \ -1]$.
2. Add 1 to each element of the vector.
3. Multiply each element of the vector by 3.
4. Square each element of the vector.
5. find the length of the vector.
6. Create a 3×4 matrix A with all elements of zero.
7. Create a 4×2 matrix B with all elements of 10.
8. Apply all possible arithmetic Operations between A and B.
9. Change the value of the element in row 2 and column 3 to 100.
10. Change the values of all elements in row 3 to 10.

4 Script M-Files

For script M-files, go to MATLAB Help → Programming and Data Types → M-File Programming.

MATLAB is a powerful programming language as well as an interactive computational environment. Files that contain code in the MATLAB language are called M-files. You create M-files using a text editor, then use them as you would any other MATLAB function or command.

There are two kinds of M-files:

- Scripts, which do not accept input arguments or return output arguments. They operate on data in the workspace.
- Functions, which can accept input arguments and return output arguments. Internal variables are local to the function.

4.1 Scripts

When you invoke a script, MATLAB simply executes the commands found in the file. Scripts can operate on existing data in the workspace, or they can create new data on which to operate. Although scripts do not return output arguments, any variables that they create remain in the workspace, to be used in subsequent computations.

For example, create a file called circle_area.m that contains these MATLAB commands.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Matlab code circle_area.m
%   Compute the area of a circle
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

r = 2;   %radius of a circle

% compute the area

A = pi*r^2
```

Typing the statement

circle_area

causes MATLAB to execute the commands and compute the area of the circle. After execution of the file is complete, the variables r and A remain in the workspace.

4.2 functions

Functions are M-files that can accept input arguments and return output arguments. The name of the M-file and of the function should be the same. Functions operate on variables within their own workspace, separate from the workspace you access at the MATLAB command prompt.

The following is an example:

```
function y = square(x)
% Compute the square of x
% Comments from this line are not displayed when you use the lookfor
% command or request help on a directory

y = x^2;
```

The first line of a function M-file starts with the keyword function. It gives the function name and order of arguments. In this case, there are one input argument and one output argument.

The next several lines, up to the first blank or executable line, are comment lines that provide the help text. These lines are printed when you type

help square

The first line of the help text is the H1 line, which MATLAB displays when you use the lookfor command or request help on a directory.

The rest of the file is the executable MATLAB code defining the function.

The function can be used in several different ways.

```
square(10)
y=square(10)
```

If youre a new MATLAB programmer, just create the M-files that you want to try out in the current directory. As you develop more of your own M-files, you will want to organize them into other directories and personal toolboxes that you can add to your MATLAB search path.

If you duplicate function names, MATLAB executes the one that occurs first in the search path.

To view the contents of an M-file, for example, square.m, use

type square.m

```
>> type square.m

function y = square(x)
% Compute the square of x
% Comments from this line are not displayed when you use the lookfor
% command or request help

y = x^2;
>>
```

Exercises

1. If C and F are Celsius and Fahrenheit temperatures, the formula

$$F = 9C/5 + 32$$

   converts from Celsius to Fahrenheit. Write a MATLAB code to convert a temperature of 37° (normal human temperature to Fahrenheit (98.6°).

2. write a function to compute $\sin(\pi t) + \cos(\pi t)$.

5 Basic Plotting

To learn basic plotting, go to MATLAB Help → Getting Started → Graphics.

The following example demonstrates how to use the plot function.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Matlab code basic_plotting.m
%   Plot the graph of a fuction
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x = 0:pi/100:2*pi;
y = sin(x);
figure(1)
plot(x,y) % produce a graph of y versus x

xlabel('x = 0:2\pi') %label the axes
ylabel('Sine of x')
title('Plot of the Sine Function','FontSize',12) %add a title

y2 = sin(x-.25);
y3 = sin(x-.5);
figure(2)
plot(x,y,x,y2,x,y3) %create multiple graphs with a single call to plot
legend('sin(x)','sin(x-.25)','sin(x-.5)') % provide an easy way to
                                          % identify the individual plots.

x1 = 0:pi/100:2*pi;
x2 = 0:pi/10:2*pi;
figure(3)
% specify color, line styles, markers, and line width
plot(x1,sin(x1),'r:',x2,sin(x2),'r+', 'linewidth', 2)
```

6 Relational Operators

For the relational operators, go to MATLAB Help → Programming and Data Types → M-File Programming → Operators → Relational Operators.

The MATLAB relational operators compare corresponding elements of arrays with equal dimensions. Relational operators always operate element-by-element. When an array is compared with a scalar, the scalar is compared with every element of the array.

```
>> a = [1 2]

a =

     1     2

>> b = [1, 3]

b =

     1     3

>> tf = a==b

tf =

     1     0

>> a = 1

a =

     1

>> tf1 = (a==b)

tf1 =

     1     0

>> c = [2 3]

c =

     2     3

>> tf = (c>b)

tf =
     1     0

>>
```

7 Logical Operators

For the logical operators, go to MATLAB Help → Programming and Data Types → M-File Programming → Operators → logical Operators.

```
>> A = [1 2 0]

A =

     1     2     0

>> B = [0 3 4]

B =

     0     3     4

>> C = A&B

C =

     0     1     0

>> D = A | B

D =

     1     1     1

>>
```

8 Flow Control

For flow control constructs, go to MATLAB Help → Getting Started → Programming with MATLAB → Flow Control.

8.1 If-Else-End Constructions

The if statement evaluates a logical expression and executes a group of statements when the expression is *true*. The optional elseif and else keywords provide for the execution of alternate groups of statements. An end keyword, which matches the if, terminates the last group of statements. The groups of statements are delineated by the four keywords–no braces or brackets are involved.

```
>> a = 1

a =

     1

>> if a ==1
      A = zeros(3,3)
   else
      B = ones(4,4)
   end

A =

     0     0     0
     0     0     0
     0     0     0

>> a = 0

a =

     0

>> if a ==1
      A = zeros(3,3)
   else
```

```
      B = ones(4,4)
   end

B =

     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1

>>
```

8.2 For Loops

The for loop repeats a group of statements a fixed, predetermined number of times. A matching end delineates the statements.

The following program uses the for loop to compute the sum: $1 + 2 + \cdots + 9 + 10$.

```
>> sum = 0

sum =

     0

>> for n=1:10
       sum = sum +n;
   end
>> sum

sum =

    55

>>
```

The following program uses the for loop to make sine wave movie.

```
 % movie_maker.m
 %use the function movie2avi to make a movie.
 clear all
 t = 0:0.01:4*pi;
 y =  sin(t);
 fc = 0;
 t_length = length(t);
 for n = 1:10:t_length
    plot(t(1:n),y(1:n), 'b -', t(n),y(n), 'r >', 'linewidth', 2)
    xlabel('t','fontsize', 18);
    ylabel('y','fontsize', 18);
    grid on
    axis([0 max(t)+0.1 -1.1 1.1]),
    pause(0.1);
    str = ['Sine Wave: t = ' num2str(t(n))];
    title(str,'fontsize', 16);
    fc = fc+1;
    F(fc)=getframe(gcf);
end

moviename= ['sine_wave.avi'];
movie2avi(F, moviename, 'fps', 6, 'colormap', jet, 'quality', 100)
```

To find functions of Audio and Audio/Video, go to MATLAB Help → MATLAB →endverbatim Functions - By Category → File I/O → Audio and Audio/Video.

8.3 While Loops

The while loop repeats a group of statements an indefinite number of times under control of a logical condition. A matching end delineates the statements.

The following program uses the while loop to compute the sum: $1+2+\cdots+9+10$.

```
>> sum = 0

sum =

     0

>> n = 1

n =
     1

>> while n<=10
         sum = sum + n;
         n = n +1;
   end
>> sum

sum =

    55

>>
```