Lab 1 Introduction to MATLAB

Objective

- Get familiar with MATLAB desktop and function areas. Learn how to use the live scripts.
- Ability to create signals with both numeric method and symbolic method.
- Mast the drawing, labeling and organization of graphics.
- Be familiar with the signal operations. Understand matrix operations and dot operations.

Content

Introduction to MATLAB

MATLAB is a powerful software tool for:

- Performing mathematical computations and signal processing.
- Analyzing and visualizing data (excellent graphics tools).
- Modeling physical systems and phenomena.
- Testing engineering designs.
- The fundamental data type is the array and the basic building block is the matrix.

The structure is shown in Figure 1.

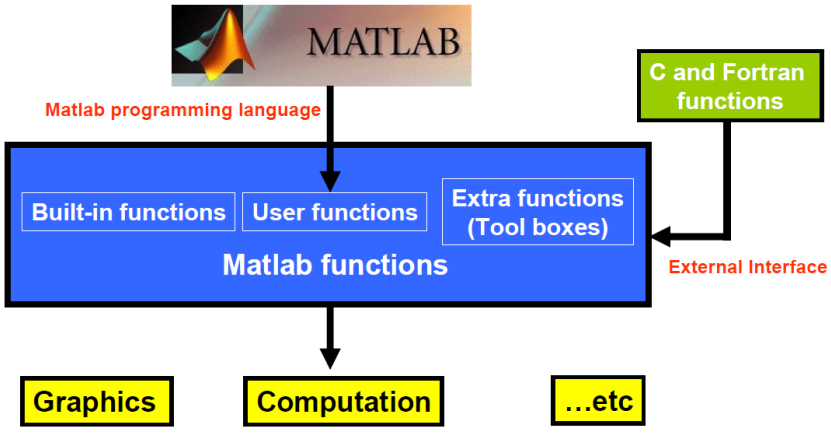

Figure 1 Structure of MATLAB

Desktop of MATLAB

The desktop of MATLAB is mainly composed of **Command Window**, **Workspace window**, **Command History** window, and **Current Folder** window, as shown in Figure 2.

- The **Command window** is where you type MATLAB commands following the prompt: >>.
- The **Workspace window** shows all the variables you have defined in your current session.

Here you can observe the variables' name, size, type, value, etc.

- The **Command History** window displays all the MATLAB commands you have used recently, even includes some past sessions. It is hidden by default.
- The **Current Folder** window displays all the files in whatever folder you select to be current.

There are still some other components. You can select what to display on your desktop by clicking on **Layout** on the **HOME** tab.

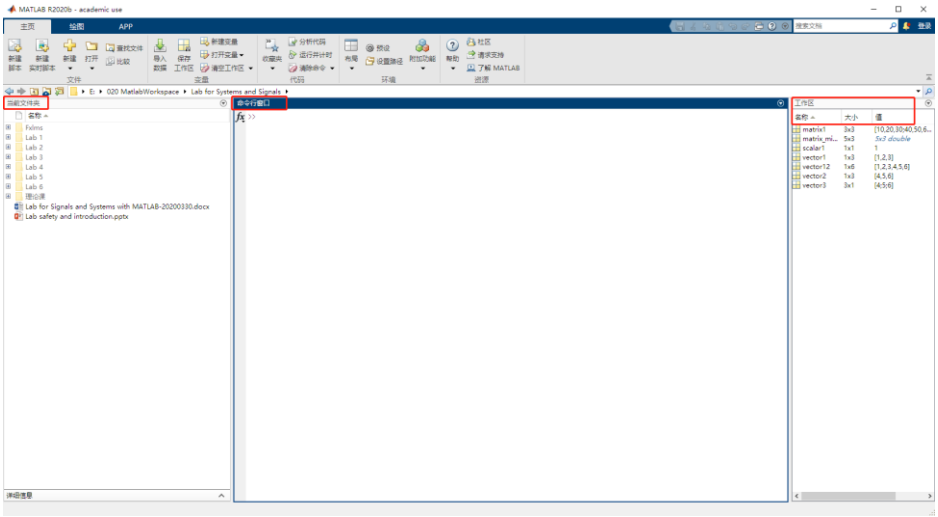

Figure 2 Desktop of MATLAB

Create Variables

A variable in MATLAB is an array. An array has dimensions $N \times M$, where N and M are in naturals. N is the number of rows and M is the number of columns. If M=N=1, the variable is a scalar. If N=1 and M>1, then the variable is a row vector. If N>1 and M=1, then the variable is a column vector. If both N and M are greater than one, then the variable is a matrix, and if N=M, then the variable is a square matrix.

To create variables, type the equation at the MATLAB command prompt. The naming of variables should observe the following rules:

- Variable names must begin with a letter.
- Names can include any combinations of letters, numbers, and underscores.
- The maximum length for a variable name is 63 characters.
- MATLAB is case sensitive. That is the variable name $\underline{A}$ is different than the variable name $\underline{a}$.
- Avoid the following names: **pi** (π)**,** and all built-in MATLAB function names such as **length**, **char**, **size**, **plot**, **break**, **cos**, **log**, …
- Use 1i or 1j to represent $\sqrt{-1}$.
- It is good programming practice to name your variables to reflect their function in a program rather than using generic x, y, z variables.

Drawing Function

The main functions for drawing are listed in Table2.

Table 1 Drawing Functions

Function	Syntax	Description
plot	plot(X, Y)	Creates a 2-D line plot of the data in Y versus the corresponding values in X.
	plot(X, Y, LineSpec)	Sets the line style, marker symbol, and color.
	plot(X1, Y1,..., Xn, Yn)	Plots multiple X, Y pairs using the same axes for all lines.
	plot(Y)	Creates a 2-D line plot of the data in Y versus the index of each value.
stem	stem(Y)	Plots the data sequence, Y, as stems that extend from a baseline along the x-axis. The data values are indicated by circles terminating each stem.
	stem(X, Y)	Plots the data sequence, Y, at values specified by X. The X and Y inputs must be vectors or matrices of the same size. Additionally, X can be a row or column vector and Y must be a matrix with length(X) rows.
	stem(_, 'filled')	Fills the circles. Use this option with any of the input argument combinations in the previous syntaxes.
	stem(_, LineSpec)	Specifies the line style, marker symbol, and color.
fplot	fplot(f)	Plots symbolic input f over the default interval [-5 5].
	fplot(f, [xmin, xmax])	Plots f over the interval [xmin, xmax].
	fplot(xt, yt)	Plots xt=x(t) and yt=y(t) over the default range of t.
	fplot(_, LineSpec)	Uses LineSpec to set the line style, marker symbol, and line color.
subplot	subplot(m, n, p)	Divides the current figure into an m-by-n grid and creates an axes for a subplot in the position specified by p. The first subplot is the first column of the first row, the second subplot is the second column of the first row, and so on.
title	title(txt)	Adds the specified title at the top and in the center of the current axes. Reissuing the title command causes the new title to replace the old title.
xlabel	xlabel(txt)	Labels the x-axis of the current axes with the text specified by txt. Reissuing the xlabel command replaces the old label with the new label.
ylabel	ylabel(txt)	Labels the y-axis of the current axes with the text, txt. Reissuing the ylabel command causes the new label to replace the old label. Labels appear beside the axis in a two-dimensional view and to the side or in front of the axis in a three-dimensional view.

axis	axis([Xmin Xmax Ymin Ymax])	Specify the limits for the current axes. Specify the limits as a vector of four, six, or eight elements.
xlim	Xlim([Xmin Xmax])	Set the x-axis limits for the current axes or graph.
ylim	Ylim([Ymin Ymax])	Set the y-axis limits for the current axes or graph.
grid	grid	Displays or removes the major gridlines for the current axes. Major gridlines extend from each tick mark.
	grid minor	Toggles the visibility of the minor grid lines. Minor gridlines lie between the tick marks.
meshgrid	[X, Y]=meshgrid(x, y)	Returns 2-D grid coordinates based on the coordinates contained in vectors x and y. X is a matrix where each row is a copy of x, and Y is a matrix where each column is a copy of y. The grid represented by the coordinates X and Y has length(y) rows and length(x) columns.
hold	hold	Toggles the hold state between on and off.

When several graphics are displayed in the same frame, use **LineSpec** to make the graphics different from each other. The details of **LineSpec** used in **plot** and **stem** function are listed in Table 3.

Table 2 Details of LineSpec

LineSpec	Value	Description
Line Style	-	Solid line (default)
	--	Dashed line
	:	Dotted line
	-.	Dash-dot line
Marker	o	Circle
	+	Plus sign
	*	Asterisk
	.	Point
	x	Cross
	s	Square
	d	Diamond
	^	Upward-pointing triangle
	v	Downward-pointing triangle
	>	Right-pointing triangle
	<	Left-pointing triangle
	p	Pentagram
	h	Hexagram
Color	y	yellow

	m	magenta
	c	cyan
	r	red
	g	green
	b	blue
	w	white
	k	black

Representation of Signals

There are two methods to generate signals, which are numerical method and symbolic method.

Create signals with numerical method

For numerical method, a key item to remember is that signals in MATLAB exist in discrete time. However, by generating sufficient samples, it is possible to make the waveform look like an analog signal. After the waveform is created, the plot command displays discrete data points as a graph of a continuous signal.

So to create a signal by numerical method, a discrete time axis should be created first, where the time range and the sampling time interval need to be specified. (Use colon to generate an array having regularly spaced elements or use build-in function **linspace**). Then use the time axis as an input to the signal creation function to get the signal value. After obtaining the time vector and the function value vector, we can use them to draw the graph.

To generate $\sin(\frac{\pi}{4}t)$ wave by numerical method, the code is as follows:

```
t = -5:0.1:5;              % time range: [-5 5]
                           % time interval: 0.1
y = sin(pi/4*t);           % calculate y point by point based on t
plot(t,y)                  % plotting
title('y=sin(pi/4*t)');
xlabel('t(s)');
ylabel('y(t)');
```

Create signals with symbolic method

The Symbolic method takes advantage of the Symbolic Math Toolbox of MATLAB. Symbolic objects do not have to be assigned, but variables or functions should be first defined as symbolic objects before being used. Use **syms** to create symbolic objects and then write out the expressions. Use **fplot** to plot symbolic signals. The details are listed in Table 4.

Table 4 Symbolic Functions

Function	Syntax	Description

syms (Create symbolic variables and functions)	syms var1 … varN	Creates symbolic variables var1 ... varN. Separate variables by spaces.
	syms f(var1,...,varN)	Creates the symbolic function f and symbolic variables var1... varN representing the input arguments of f. You can create multiple symbolic functions in one call. For example, syms f(x) g(t) creates two symbolic functions (f and g) and two symbolic variables (x and t).
fplot (Plot expression or function)	fplot(f)	Plots the curve defined by the function y = f(x) over the default interval [-5 5] for x.
	fplot(f,xinterval)	Plots over the specified interval. Specify the interval as a two-element vector of the form [xmin xmax].

To generate $\sin(\frac{\pi}{4}t)$ wave by symbolic method, the code is as follows:

Eg1:

```
syms x;                    % define the symbolic variable x
y = sin(pi/4*x);           % synbolic function
fplot(y,[-16,16])          % plotting
```

Eg2:

```
syms y(x);
y(x) = sin(pi/4*x);        % define symbolic function
fplot(f,[-16,16]);         % plotting
```

Typical signals

Some other examples of generating signals with the numerical method or symbolic method, or with both methods are listed in Table 5.

Table 5 Generation of Typical Signals

Function Used	Signal Example	Code

ft=A*sin(w*t+pha)	f(t)=sin(2πt+π/6)	t=0:0.01:10;
		A=1; w=2*pi; pha=pi/6;
		ft=A*sin(w*t+pha);
		plot(t, ft); grid on; hold on;
		
		syms t
		A=1; w=2*pi; pha=pi/6;
		ft=A*sin(w*t+pha);
		fplot(ft, [0,10] ,'--'); hold off;
		title(' f(t)=sin(2πt+π/6)'); xlabel('t'); ylabel('f(t)');

ft=A*cos(w*t+pha)	f(t)=cos(2πt+π/6)	t=0:0.01:10;
		A=1; w=2*pi; pha=pi/6;
		ft=A*cos(w*t+pha);

		plot(t, ft); grid on; hold on;
		
		syms t
		A=1; w=2*pi; pha=pi/6;
		ft=A*cos(w*t+pha);
		fplot(t, ft, [0,10], '--'); hold off;
		title(' f(t)= cos(2πt+π/6)'); xlabel('t'); ylabel('f(t)');

Sa(t)=sinc(t)	Sa(t)=sin(t)/t	t=-3:0.01:3;
		ft=sinc(t);
		plot(t, ft); grid on; hold on; axis([-3, 3, -0.5, 1.2]);
		
		syms t
		ft = sinc(t);
		fplot(t, ft, [-3 3], '--'); hold off;
		title(' Sa(t)=sin(t)/t '); xlabel('t'); ylabel('Sa(t)');

ft=A*exp(a*t)	f(t)=e^(-0.4t)	t=0:0.01:10;
		A=1; a=-0.4;
		ft=A*exp(a*t);
		plot(t, ft); grid on; hold on;
		
		syms t
		A=1; a=-0.4; ft=A*exp(a*t);
		fplot(t, ft, [0 10], '--'); hold off;

ft=tripuls(t,w,s)
w for width
s for skew:-1<s<1
	Triangular signal with width of 4 and skew of 0.5.	t=-3:0.01:3;
		ft=tripuls(t, 4, 0.5);
		plot(t, ft); grid on; axis([-3, 3, -0.5, 1.5]);
		title(' Triangular signal '); xlabel('t'); ylabel('f(t)');

ft=sawtooth(t,xmax)
the period is 2π
w is a scalar parameter between 0 and 1
	Sawtooth wave with period 2π and maximum occurs at π.	t=-3*pi:0.01:3*pi;
		ft=sawtooth(t, 0.5);
		plot(t, ft); grid on; axis([-3*pi, 3*pi, -1, 1]);
		title(' Sawtooth wave '); xlabel('t'); ylabel('f(t)');

ft=rectpuls(t,w)
w for width
	Rectangle signal with width of 1 and amplitude of 2.	width=1;t=-2:0.01:3;
		ft=2*rectpuls(t, width);
		plot(t, ft);grid on;
		title(' Rectangle signal '); xlabel('t'); ylabel('f(t)');

ft=square(t,d)
the period is 2π
d for the duty cycle, which is the percent of the period in
	Square wave with period π and duty cycle 70.	t=-3*pi:0.01:3*pi;
		ft=square(2*t, 70);
		plot(t, ft); grid on; axis([-3*pi, 3*pi, -1, 1]);
		title(' Square wave '); xlabel('t'); ylabel('f(t)');

which the signal is positive.		

Besides, the unit step signal and impulse signal are of great importance. The two signals can be generated by **heaviside(t)** and **dirac(t)** for both numerical method and symbolic method.

To generate u(t):

```
% numeric method
t = -5:0.01:5;
u = heaviside(t);
subplot(2,1,1);plot(t, u);
xlabel('t');ylabel("u(t)");title('Numeric Method');

% symbolic method
syms t
u = heaviside(t);
subplot(2,1,2);fplot(u);
xlabel('t');ylabel("u(t)");title('Symbolic Method');
```

To generate $\delta(t)$:

```
% numeric method
t = -5:0.01:5;
d = dirac(t);
subplot(2,1,1); plot(t,d);
xlabel('t');ylabel("delta(t)");title('Numeric Method');

% symbolic method
syms t
d = dirac(t);
subplot(2,1,2);fplot(d);
xlabel('t');ylabel("delta(t)");title('Symbolic Method');
```

Since the value of delta is infinite, it cannot be displayed when you try to plot it. To view the wave of $\delta(t)$, use function **sign** to make it a unit pulse signal, and then display it .

```
syms t
d = dirac(t);
fplot(sign(d));
xlabel('t');ylabel("d(t)"); title('Impulse Signal');
```

Signal Operation

The basic operation of the signal includes multiplication, addition, scaling, rotation, translation, differential, integral and so on. Both numerical methods and symbolic methods can be used to realize the basic operation.

Some commonly used operators in MATLAB are listed in Table 6.

Table 6 Common Used Operator

Symbol	Description
+	Addition:A+B
-	Subtraction:A-B
*	Matrix multiplication. C = A*B is the linear algebraic product of the matrices A and B. For nonscalar A and B, the number of columns of A must equal 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.
^	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 A(i,j) to the B(i,j) power. 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).
./	Right array divide. A./B is the matrix with elements A(i,j)/B(i,j). A and B must have the same size unless one of them is a scalar.
\	Backslash or left matrix divide. A\B is roughly the same as INV(A)*B.
.\	Left array divide. A.\B is the matrix with elements A(i,j)\B(i,j). 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.
:	Colon: generates an array having regularly spaced elements.
,	Comma: separates elements of an array.
;	Semicolon: suppresses screen printing; also denotes a new row in an array.
…	Ellipsis: continues a line.

f1(t)=sinwt,f2(t)=sin8wt,w=2π. Plot f1(t)+f2(t) and f1(t)*f2(t) with numerical method.

```
w = 2*pi;t = -3:0.001:3;
f1 = sin(w*t);f2 = sin(8*w*t);
subplot(2,1,1);plot(t,f1+1,':',t,f1-1,':',t,f1+f2);
grid on;title('f1(t)+f2(t)');xlabel('t');ylabel('f(t)')
subplot(2,1,2);plot(t,f1,':',t,-f1,':',t,f1.*f2);
grid on;title('f1(t)*f2(t)');xlabel('t');ylabel('f(t)')
```

There are still some useful functions for signal operations in MATLAB. Some of which are listed in Table 7.

Table 7 Functions for Signal Operations

Function	Syntax	Description

abs	abs(x)	Return the absolute value of each element in array x.
angle	angle(x)	Return the phase angle, in radians, for each element of complex array x. The angles lie between $\pm\pi$
real	real(x)	Return the real part of the elements of the complex array x.
imag	imag(x)	Return the imaginary part of the elements of array x.
floor	floor(x)	Round each element of x to the nearest integer less than or equal to that element.
ceil	ceil(x)	Round each element of x to the nearest integer greater than or equal to that element.

Try to use the function in Table 7 with $x = 1.2 + 3.6j$.

Function diff and int/trapz/cumtrapz are used to do differentiation and integration. The details of the four functions are listed in Table 8.

Table 8 Functions for Differentiation and Integration

Function	Syntax	Description
diff	diff(X)	Calculates differences between adjacent elements of X along the first array dimension whose size does not equal 1.
	diff(X,n)	Calculates the nth difference by applying the diff(X) operator recursively n times. In practice, this means diff(X,2) is the same as diff(diff(X)).
	diff(X,n,dim)	The nth difference calculated along the dimension specified by dim. The dim input is a positive integer scalar.
int	int(S)	The indefinite integral of S with respect to its symbolic variable as defined by SYMVAR. S is an SYM (matrix or scalar). If S is a constant, the integral is with respect to 'x'.
	int(S,a,b)	The definite integral of S with respect to its symbolic variable from a to b.
	int(S,v,a,b)	The definite integral of S with respect to v from a to b.
trapz/cumtrapz	trapz(x,y) cumtrapz(x,y)	Trapezoidal numerical integration.

What is the differential of unit step signal? Let's have a look.

```
% symbolic method
syms x
y = diff(heaviside(x),x);
fplot(sign(y));
xlabel('x');ylabel('y');title('Differential of u(t)');
```

```
axis([-2 2 -0.1 1.1])

% numeric method
t = -5:0.01:5;
f = diff(heaviside(t));
plot(t(1:end-1),f)
xlabel('t');ylabel('f(t)');title('Differential of u(t)');
ylim([-0.1 0.6])
```

Then what is the integral of unit step signal?

```
% set the integral constant to zero
c = 0;

% symbolic method
syms x
y = int(heaviside(x),x)+c;
fplot(y);
title('Integral of u(t)');xlabel('x');ylabel('y');

% numberic method
t = -5:0.01:5;
f = cumtrapz(t,heaviside(t))+c;
plot(t,f);
title('Integral of u(t)');xlabel('x');ylabel('y');
```

To find out the definite integral of the unit step signal in range [-1 2], we can do it with function **int** and **trapz**.

```
% symbolic method
syms t
int(heaviside(t),-1,2)

% numeric method
t = -1:0.01:2;
trapz(t, heaviside(t))
```

Program Structure

MATLAB has three basic program structures: sequential, loop and branch structure.

A sequential structure is the simplest program structure. Sequential statements are composed of assignment statements or functions. In which there is no control structure.

For-end and while-end are used to form loop structures. Use for-end to repeat statements a specific number of times. Use the while-end to repeat statements an indefinite number of times until a condition is no longer satisfied. So when the number of cycles is fixed, the for-end structure is more convenient, if not, while-end is more suitable.

The structure of for-end is like:

```
for i=m:s:n
        statement body
end
```

The structure of while-end is like:

```
while logical expression
        statement body
end
```

If-else-end and switch are used to form branch structures.

The structure of if-else-end includes three styles, which are:

```
if logical expression
        statement body
end
```

```
if logical expression
        statement body-1
else
        statement body-2
end
```

```
if logical expression-1
        statement body-1
elseif logical expression-2
        statement body-2
...
elseif logical expression-n
        statement body-n
else
        statement body-n+1
end
```

The structure of **switch** is used to judge multiple conditions in a succinct manner, the expression is like:

```
switch expression
case constant expression value-1
        statement body-1
...
case constant expression value-n
        statement body-n
otherwise
        statement body-n+1
end
```