

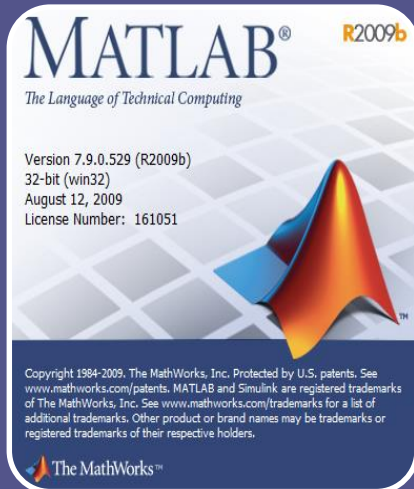


北京航空航天大学
BEIHANG UNIVERSITY

MATLAB Programming (Lecture 2)

Dr. SUN Bing
School of EIE
Beihang University

Contents



MATLAB basics

- Introduction to M-file
- Keywords and Variables
- Initializing Variables
- Array Accessing and sub array
- Displaying output data
- Data files
- Build-in Functions
- Introduction to plotting
- Good practices

2.1 Introduction to M-file

2.1.1 What is the M-file ?

M-file has two types:

(1). M-file script

(2). M-file Function

M-file script is a script file with an extension .m. It is collections of MATLAB commands or statements that are stored in the file.

2.1.2 The characteristics of M-files script.

It shares the command window's workspaces; It has no input arguments and returns no results.

2.1.3 How the M-file can be created?

Use the Edit/Debug Window

2.1.4 How to execute the M-file?

Type in the file name in command windows.

2.1 Introduction to M-file

2.1.5 Statement delimiters

- MATLAB accepts executable statements from the keyboard, or from a M-file, in effect one line at a time.
- To continue a statement over more than one line, every non-final line of the statement must end in an **ellipsis (...)**.

For example, here is a legal statement:

```
y = ( x^2 + 5*x + 6 ) ...  
/ ( x^2 + 6*x + 7 ) ;
```

- But if the ellipsis is omitted, line 1 is accepted as legal while line 2 is regarded as a separate but malformed statement, producing an error.
- Statements can also be ended with **comma (,)** or **semicolon (;)**, enabling the placement of more than one statement on a line. Ending a statement with **;** also suppresses those mostly unwanted printouts of results.

2.1 Introduction to M-file

2.1.6 Blanks and comments

- Other features that improve readability are the **blank and the comment**.
- Blanks may be inserted anywhere except in the middle of a name (e.g. delta, but not del ta) or operator (e.g. a .* b, but not a. *b), or number (e.g. 1e6, not 1 e6).
- The **character %** anywhere on a line tells MATLAB to ignore the rest of the line, including other %'s. Thus the above example can be commented as follows:

```
% -- Compute the rational function --  
y = ( x^2 + 5*x + 6 ) ... % Numerator.  
/ ( x^2 + 6*x + 7 ) ;      % Denominator.
```

2.2 Keywords and Variables

2.2.1 Keywords

- MATLAB reserves certain words for its own use as keywords of the language.
- To list the keywords, type the command
`>> iskeyword`
- **Note:** You should not use MATLAB keywords other than for their intended purpose.

Command Window

```
>> iskeyword

ans =

    'break'
    'case'
    'catch'
    'classdef'
    'continue'
    'else'
    'elseif'
    'end'
    'for'
    'function'
    'global'
    'if'
    'otherwise'
    'parfor'
    'persistent'
    'return'
    'spmd'
    'switch'
    'try'
    'while'
```

2.2 Keywords and Variables

- ◆ The fundamental unit of data in any MATLAB program is an array/matrix . Even scalars are treated as arrays/matrix with one row and one column.
- ◆ An array is a collection of data values organized into rows and columns, and known by a simple name.(variable name)
- ◆ An individual element value within an array can be accessed by array name followed by subscripts in parentheses.

2.2 Keywords and Variables

2.2.2 Variables

1. A Variable is a region of memory contained an array/matrix, which is known by user specified name.
2. A Variable name must begin with a *letter*, followed by any combination of letters, digits and the underscore() character.

2.2 Keywords and Variables

2.2.2 Variables (*continued*)

3. Although variable names can be of any length, MATLAB uses only the first N characters of the name, (where N is the number returned by the function `namelengthmax`), and ignores the rest. Hence, it is important to make each variable name unique in the first N characters to enable MATLAB to distinguish variables.

```
>> N = namelengthmax  
N = 63
```

2.2 Keywords and Variables

2.2.2 Variables (*continued*)

- The MATLAB language is case-sensitive. Thus the variable name ***vname*** , ***VNAME*** and ***Vname*** are all different in MATLAB. It is customary to use all lowercase letters for ordinary variable name.
- It is important to pick a meaningful names for the variables. Meaningful names make a program much easier to read and to maintain.
- The most common type of MATLAB variables are *double* and *char*. (see data types)

2.2 Keywords and Variables

2.2.2 Variables (*continued*)

- In a language such as C, the type and name of every variable must be explicitly declared in program before it is used. These languages are called strongly typed languages. In contrast, MATLAB is a weakly typed language. *Variables can be created at any time by simply assigned values to them.*
- MATLAB includes a number of predefined special values. They are stored in ordinary variables, such as `pi`, `eps`, `j`, `i` etc. (see 2.2.3).
- So Never redefine the meaning of a predefined variables.

2.2 Keywords and Variables

2.2.3 *Predefined special Variables(1)*

- **ans** A special variable used to store the result of an expression if that result is not explicitly assigned to other variables.
- **eps** This variable name is short for *epsilon*. It is the smallest difference between two numbers that can be represented on computer.
- **date** contains the current date in string form.
- **clock** This variable contains the current date and time.

2.2 Keywords and Variables

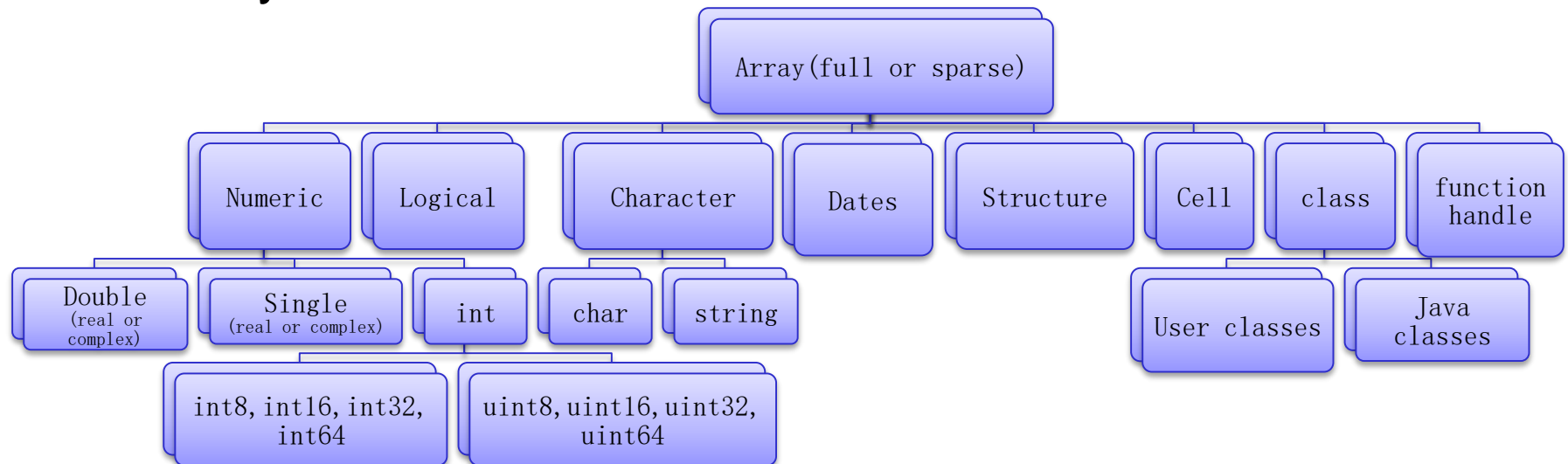
2.2.3 Predefined special Variables(2)

- realmax Largest floating-point number
- realmin Smallest floating-point number
- pi contains π to 15 significant digits
3.1415926535897....
- i, j, 1i, 1j contain the imaginary unit value ($\sqrt{-1}$)
- Inf, inf ∞ (machine infinity)
- NaN, nan Not-a-Number

2.2 Keywords and Variables

2.2.4 data types

- The most basic data structure in MATLAB® is the matrix(or two dimensional array).
- There are 8 kinds of fundamental data types in MATLAB. Each of these data types is in the form of a matrix or array.



2.2 Keywords and Variables

2.2.4 *data types*

1. Numeric Types:

- `double` (real or complex)
- `single`
- `int`
 - `int8`, `int16`, `int32`, `int64`
 - `uint8`, `uint16`, `uint32`, `uint64`

Attention:

1. By *default*, MATLAB stores all numeric values as double-precision floating point.
2. Complex numbers consist of a real part and an imaginary part.
3. The basic imaginary unit is equal to the square root of -1. This is represented in MATLAB by either of two letters: *i* or *j*.

2.2 Keywords and Variables

2.2.4 *data types*

2. Logical Types :

The logical data type represents a logical true or false state using the numbers 1 and 0, respectively.

3. Character Types:

char or string.

4. Dates and Times:

MATLAB represents date and time information in either of three formats: date strings, serial date numbers, or date vectors.

5. Structure Types:

Structures are MATLAB arrays with named "data containers" called fields. The fields of a structure can contain any kind of data.

2.2 Keywords and Variables

2.2.4 *data types*

6. Cell Types:

A cell array provides a storage mechanism for dissimilar kinds of data. You can store arrays of different types and/or sizes within the cells of a cell array.

7. Function Handles :

A function handle is a MATLAB value and data type that provides a means of calling a function indirectly.

8. Class Types:

All MATLAB data types are implemented as object-oriented classes. But you can define your own data type user-defined classes.

help datatypes

2.3 Initializing Variables

- MATLAB variables are automatically created when they are initialized. There are three ways to initialize a variable.

- 1. Assign data to the variable in an assignment statement.***
- 2. Input data in to variable from the keyboard.***
- 3. Read data from a data file.***

2.3 Initializing Variables

2.3.1 Assignment statement(1)

The general form of assignment statement is

var = expression

Examples :

```
1.  c1 = 25+3*i;      % complex value
2.  c2 = c1/3;
3.  d = 23.56;        % real number
4.  A = [1 2 3 4];    % creates a 1x4 array(vector)
5.  B = [1;2;3;4];    % creates a 4x1 array(vector)
```

2.3 Initializing Variables

- **2.3.1 Assignment statement(2)**

- The array of data can also assign to the variables by using square bracket (**[]**). All of the elements of an array are listed in row order. Each row is separated by semicolons(**;**) or new lines.
- The values of elements within each row are separated by blank or commas(**,**).
- The numbers of elements in every row of an array must be the same, and The numbers of elements in every column of an array must be the same.
- Symbol **[]** stands for empty array, which contains no rows and no columns.

2.3 Initializing Variables

2.3.1 Assignment statement(3)

Colon operator and shortcut expression:

first : incr : last

Examples:

`A = 1:2:10; % equivalent to A= [1 3 5 7 9];`

`B = 1:6; % equivalent to B= [1,2,3,4,5,6];`

`Angles = (0.01:0.01:1.0)*pi;`

`%one hundred values: $\pi/100$, $2\pi/100$, ..., π`

2.3 Initializing Variables

2.3.1 Assignment statement(4)

With built-in function.

1. **zeros (n)** Generates an $n \times n$ matrix of zeros.
2. **zeros (n,m)** Generates an $n \times m$ matrix of zeros.
3. **ones (n)** Generates an $n \times n$ matrix of ones.
4. **eye (n)** Generates an $n \times n$ identity matrix.
5. **size(arr)** Returns two values specifying the numbers of rows and columns in **arr**.

Examples of built-in functions

- `zeros(2)` generates 2×2 matrix of zeros.
- `ones(2)` generates 2×2 matrix of ones.
- `eye(2)` generates 2×2 identity matrix .

`A=zeros(2,3)`

`B=zeros(2,3,'int32')`

```
Command Window

>> A = zeros(2)

A =

    0    0
    0    0

>> B = ones(2)

B =

    1    1
    1    1

>> C = eye(2)

C =

    1    0
    0    1

>> D = size(C)

D =

    2    2
```

Examples of built-in functions

```
A=zeros(2,3);
```

```
B=zeros(2,3,'int32');
```

```
A(:)=1:6
```

```
B(:)=1:6
```

```
A/6
```

```
B/6
```

```
help idivide
```


2.3 Initializing Variables

2.3.2 Input data in to variable from keyboard

- The input function allows a script file(M-file) to prompt a user for input data value to variable while the M-file is executing. The input function has two forms as following.

1) `var = input('prompt text string ');`

2) `var = input('prompt text string ','s');`

- Scalar or array data value can be input to specified variable by using input function. For detail explanation, see next slide.

2.3 Initializing Variables

INPUT function

```
R = input('enter input data : ')
```

- 1) It gives the user the prompt in the text string and then waits for input from the keyboard.
- 2) The input can be any MATLAB expression, which is evaluated, using the variables in the current workspace, and the result returned in R.
- 3) If the user presses the return key without entering anything, It returns an empty matrix.

2.3 Initializing Variables

INPUT function

```
R = INPUT ( 'What is your name', 's' )
```

- 1) It gives the prompt in the text string and waits for character string input.
- 2) The typed input is not evaluated, the characters are simply returned to R as a MATLAB string.

2.4 Array Accessing and sub array

- Accessing Single Elements : $A(\text{row}, \text{column})$
- Linear Indexing: MATLAB always allocates array elements in Column Major Order. Suppose A is a two dimensional array $r \times c$. If you supply two subscripts (i, j) representing row-column indices, the offset is :

$$k = (j-1) * r + i$$

- Accessing Sub Array

2.4 Array Accessing and sub array

- Linear Indexing: **Column Major Order**. Suppose A is a two dimensional array $r \times c$. If you supply two subscripts (i, j) representing row-column indices, the offset is :

$$k = (j-1) * r + i$$

C=

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

C(1,2)=2

C(2)=?

See example [linear_indexing.m](#)

2.4 Array Accessing and sub array

- MATLAB allows us to create arrays with as many dimensions as necessary. One and two dimensions arrays are more popular.
- It is possible to select and use subsets of arrays as though they are separate arrays. To select portions of an array, just include a list of all the elements to be selected in the parentheses after the array name.

2.4 Array Accessing and sub array

- For example, suppose array `arr1` is defined as follows.

```
arr1 = [1.1, -2.2, 3.3, -4.4, 5.5];
```

```
%one dimension array
```

(1) `arr1(4)` is -4.4

(2) `arr1([1 4])` is the array [1.1 -4.4]

(3) `arr1(1:2:5)` is the array [1.1 3.3 5.5]

- For two dimensions array, a colon(:) can be used in a subscript to select all the values of that subscript.

2.4 Array Accessing and sub array

For example, suppose two dimensional `arr2` is defined as follows.

```
arr2 = [1 2 3; -4 -5 -6; 7 8 9];
```

- The sub array expression:

1) `arr2(2, :)` is second row of the array `[-4 -5 -6]`

2) `arr2(:, 1)` is the first column of the array. `[1 -4 7]'`

3) `arr2(1:2, [1 3])` is the array `[1 3; -4 -6]`

4) `arr2(1, 2:end)` is the array `[2,3]`

Note: `end` function returns the highest values of a given subscripts. Here the `end` returns value 3.

Questions

Assume an array A is defined as shown,

```
A = [1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16];
```

•Determine the contents of the following sub arrays:

(1) A(1,2)

(2) A(2,:)

(3) A(:,4)

(4) A(1:2,1:2);

(5) A([2 2],[3 3])

(6) A'

(7) A(:,end)

2.4 Array Accessing and sub array

- It is also possible to use sub array on the left-hand side of an assignment statement to update only some value of array. For example

$$A(1:2, [1 \ 4]) = [10 \ 11; \ 22 \ 24];$$

Only the elements $A(1,1)$, $A(1,4)$, $A(2,1)$ and $A(2,4)$ were updated, all others are unchanged.

- Note:** the shapes of the sub arrays on both sides of the equal sign (=) must match. MATLAB will produce an error, if they do not match.

Questions

Assume an array A is defined as shown,

```
A = [1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16]
```

What is the result of following statements.

- (1) `A(1:2,1:2) = 2;` % assign a scalar to subarray.
- (2) `A(1,:) = [];` % delete first row of A. (3 × 4)
- (3) `A(:,4) = [];` % delete 4th column of A. (3 × 3)
- (4) `A(1:2:3,[1 3]) = [100 300 ;100 300];`
 %A is still a (3 × 3) array.
- (5) `A = [100 300 ;100 300];` %A is a 2 × 2 array

2.5 Displaying output data(1)

1. If statement is typed without the semicolon(;), the results of statement are automatically displayed in the command window. The semicolon suppresses the automatic echoing of results.
2. The default format shows 5-digit scaled fixed point. The format command can change the display format.

format short or format long

2.5 Displaying output data(2)

You can use the format command to change the display format.

`short` Scaled fixed point format,
with 5 digits

`long` Scaled fixed point format,
with 15 digits

```
Command Window

>> pi

ans =

    3.1416

>> format long
>> pi

ans =

    3.141592653589793

fx >>
```

2.5 Displaying output data(3)

3. The `disp` function: `disp(var)`

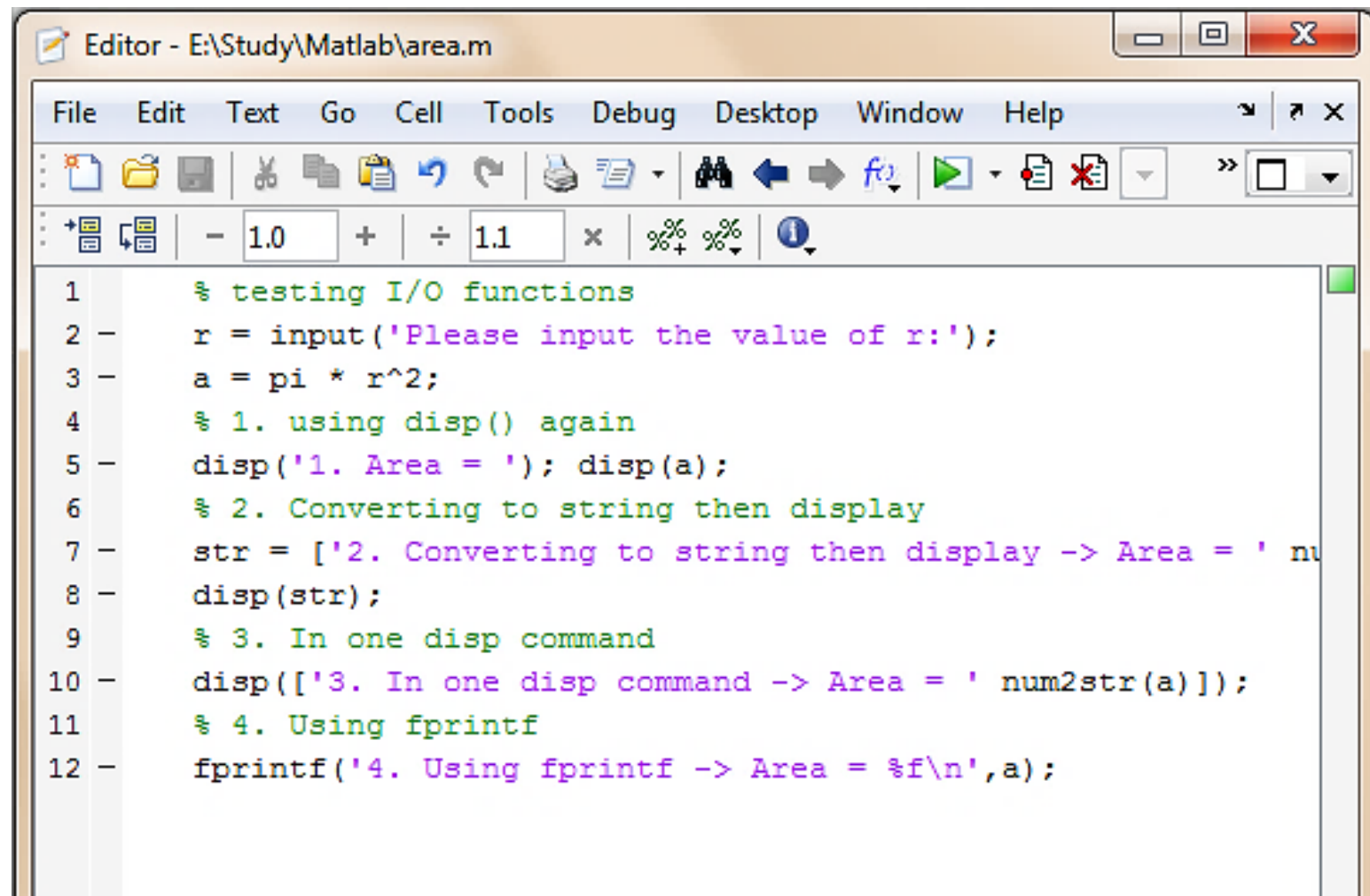
The `var` can be number or string. It is often combined with function `num2str` and `int2str` to create message to be displayed.

4. The `fprintf` function : `fprintf(format, vars)` format is a string containing C language conversion specifications such as `%f` , `%d` and so on.

`vars` is the list of the variables separated by comma(,).

Examples of output functions

The M-file area.m tests the I/O functions.



```
1      % testing I/O functions
2 -    r = input('Please input the value of r:');
3 -    a = pi * r^2;
4      % 1. using disp() again
5 -    disp('1. Area = '); disp(a);
6      % 2. Converting to string then display
7 -    str = ['2. Converting to string then display -> Area = ' num2str(a)];
8 -    disp(str);
9      % 3. In one disp command
10 -    disp(['3. In one disp command -> Area = ' num2str(a)]);
11     % 4. Using fprintf
12 -    fprintf('4. Using fprintf -> Area = %f\n',a);
```

After Running

Command Window

```
>> area
```

```
Please input the value of r:10
```

```
1. Area =
```

```
    3.141592653589793e+02
```

```
2. Converting to string then display -> Area = 314.1593
```

```
3. In one disp command -> Area = 314.1593
```

```
4. Using fprintf -> Area = 314.159265
```

```
 >>
```


2.6 Data files(1)

MATLAB provides many ways to load and save data files in order to simplify program running.

The most simplest commands are **save** and **load**

save Save the workspace to file `matlab.mat` in the current directory.

```
save filename vars -ascii
```

Save specified variable to file in text form.

load Load the `matlab.mat` in the current directory. to the workspace.

```
load filename -ascii
```

Load the text data file to the variable which name is same as filename.

Examples of save command

Command Window

```
>> A = [1,2,4,1; 2,8,6,4; 3,10,8,8; 4,12,10,6];
```

```
>> B = [21,52,79,82]';
```

```
>> X = A\B
```

```
X =
```

```
1
```

```
2
```

```
3
```

```
4
```

```
>> save
```

```
Saving to: matlab.mat
```

```
>> save A.dat A -ascii
```

```
>> save B.dat B -ascii
```

```
fx >> |
```

Example of load command

Command Window

```
>> load A.dat -ascii  
>> load B.dat -ascii  
>> X = inv(A) * B
```

X =

```
1.0000  
2.0000  
3.0000  
4.0000
```

fx >>

2.6 Data files(2)

- `csvwrite('filename',M)`
writes matrix M into filename as comma-separated values.
- `M = csvread('filename')`
reads a comma-separated value formatted file, filename. The result is returned in M. The file can only contain numeric values.
- `type('filename')` or `type filename`
Displays the contents of the specified file in the MATLAB Command Window.

2.7 Build-in Functions(1)

There are three types of built-in functions types available in MATLAB.

1. *Elementary built-in functions*

Typing *help elfun* at the command prompt will give you a list of the Elementary math functions. Such as Trigonometric, Exponential, Complex, and Rounding and remainder.

2. *Special Math functions*

Typing *help specfun* at the command prompt will give you a list of these functions, such as Bessel function, Beta function, Gamma function,...,etc

Also try the command *help elmat* , *help polyfun*,...

3. *Special functions* - toolboxes

Some Elementary Mathematical Functions

Symbol

exp

abs

cos, sin, tan

acos, asin, atan

cosh, sinh, tanh

sqrt

conj

fix

round

log

log10

real

imag

function

exponentiation (e^x)

absolute value

cosine, sine and tangent

inverse of cosine, sine and tangent

hyperbolic functions (also inverses)

square root

complex conjugate

round toward zero

round toward nearest integer

natural logarithm

base 10 logarithm

real part of complex number

imaginary part of complex number

2.7 Build-in Functions(2)

- Command *help elmat* will display a list of elementary matrices and matrix manipulation. such as:

rand *Uniformly distributed random numbers.*

randn *Normally distributed random numbers.*

linspace *Linearly spaced vector.*

linspace(a,b,n) *generates a row vector y of n points linearly spaced between and including a and b.*

**It is similar to the colon operator ":"*

logspace *Logarithmically spaced vector.*

- In the help window , You can find the function by category , mathematics.

2.7 Build-in Functions(3)

- Unlike other programming language ,The MATLAB build-functions can receive an array of input values, and return an array of output values on element-by-element basis. For example.

```
X=[0    pi/2    pi    3*pi/2    2*pi];  
Y=sin(X);
```

- The result is $Y = [0 \ 1 \ 0 \ -1 \ 0]$.

2.8 Introduction to plotting

- The one of MATLAB's most powerful features are device-independent plotting. They make it very easy to plot any set of data.
- MATLAB provides a variety of functions for displaying vector data as line plots, as well as functions for annotating and printing these graphs.
- The more common and basic plotting functions are:
 - (1) `plot(varlist)` Graph 2-D data with linear scales for both axes
 - (2) `plot3(varlist)` Graph 3-D data with linear scales for both axes. See the following examples.

2.8 Introduction to plotting

2.8.1 *The plot() function*

```
plot(x, y, 'PropertyName', PropertyValue, ...)
```

The property name:

LineWidth -- Specifies the width (in points) of the line.

MarkerEdgeColor -- Specifies the color of the marker or the edge color for filled markers (circle, square, diamond, pentagram, hexagram, and the four triangles).

MarkerFaceColor -- Specifies the color of the face of filled markers.

MarkerSize -- Specifies the size of the marker in units of points.

2.8 Introduction to plotting

2.8.1 *The Example of `plot(x,y)` function.*

Plot the curve of following function on interval $[0, 4\pi]$.

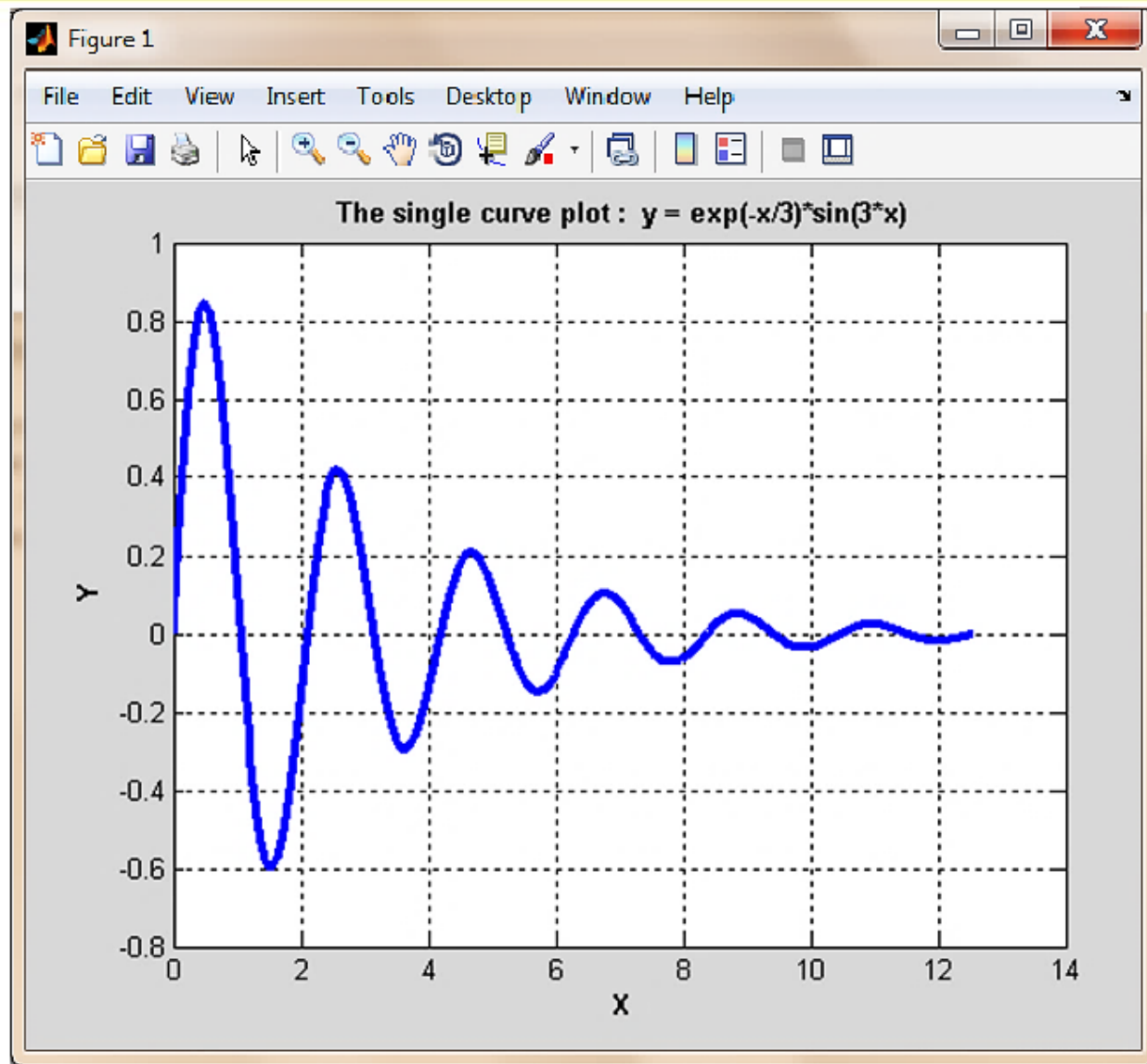
The M-file (plot1.m) and it's execution result as follows.

$$y = e^{-\frac{t}{3}} \sin(3t)$$

Example 1

```
% Scrip file name : plot1.m
% The example of plot fuction.
X= 0:pi/100:4*pi;
Y = exp(-X/3).*sin(3*X);
plot(X,Y, 'linewidth',3);
title('\bf The single curve plot :  $y = \exp(-x/3) \sin(3x)$ ');
grid on;
xlabel('\bf X');
ylabel('\bf Y');
axis([0 14 -0.8 1.0]);
```

Example 1: Output



2.8 Introduction to plotting

2.8.2 Line color, Line style and Maker style

- MATLAB allows a programmer to select the color, style of line to be plotted, and the type of maker to be used for data points. These traits can be selected using an attribute character string after the x and y vectors in the plot() function. The attribute characters are shown in following table.

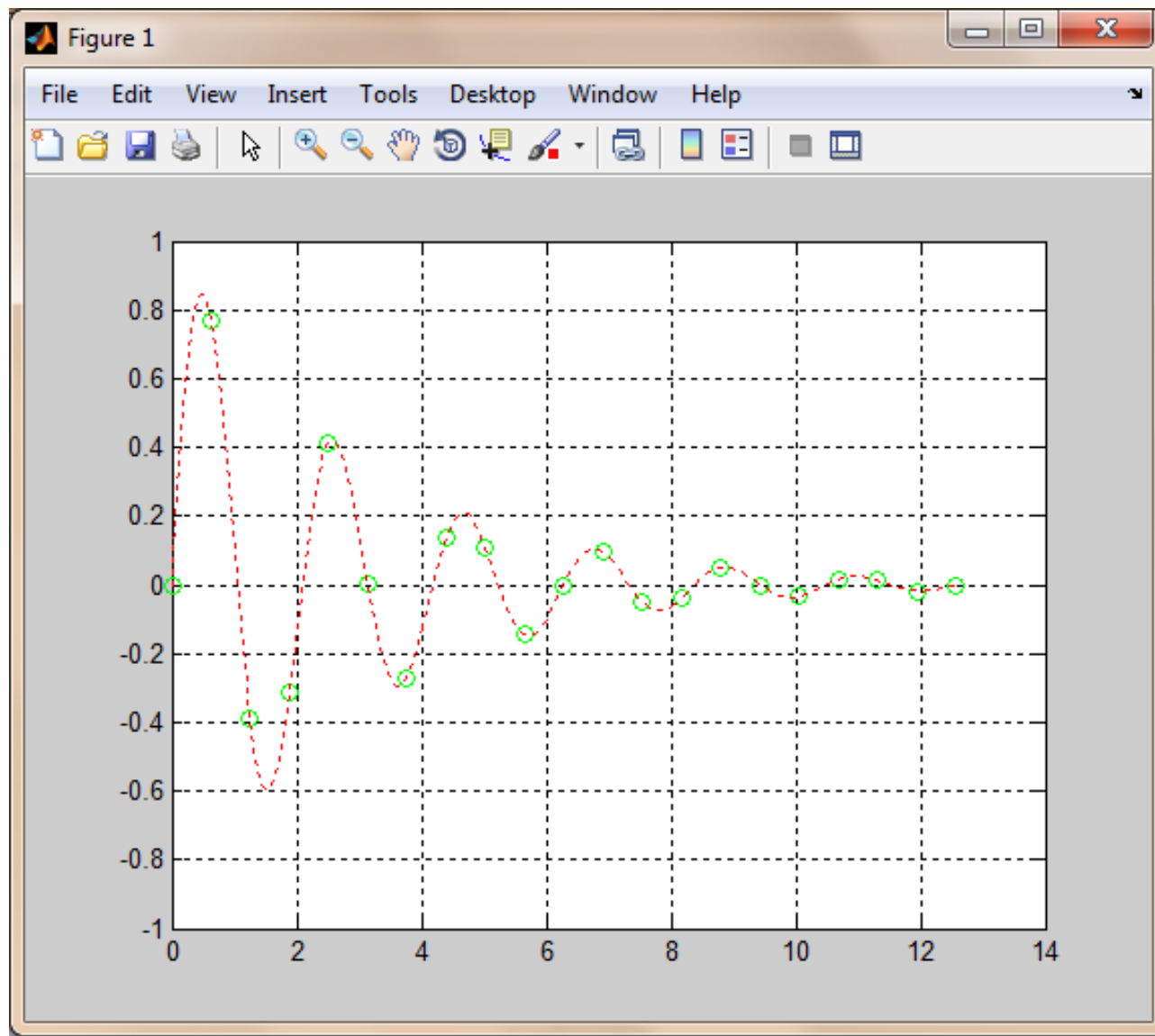
Table2.1 Line color, style and Maker style

Line color	Line style	Maker style
y yellow	- solid	. point
m magenta	: dotted	o circle
c cyan	-. dash-dot	x x-mark
r red	-- dashed	+ plus
g green		* star
b blue		s square
k black		d diamond
w white		^ triangle(up)
		v triangle(down)

Example 2 line , maker color and style

```
% Example line style and maker style
% M-file name plot1s.m
t1=0:pi/5:4*pi;
y1=exp(-t1/3).*sin(3*t1);
t = 0:pi/100:4*pi;
y = exp(-t/3).*sin(3*t);
plot(t1,y1,'go',t,y,'r:');
grid on
axis([0 14 -1.0 1.0]);
```


Example 2: Output



2.8 Introduction to plotting

2.8.3 Multiple plots

- (1). It is possible to plot multiple functions on the same graph simply by including more than one set of (x,y) values and attribute character string in the plot function.
- (2). `Plot(x, y)` with function hold on.

2.8 Introduction to plotting

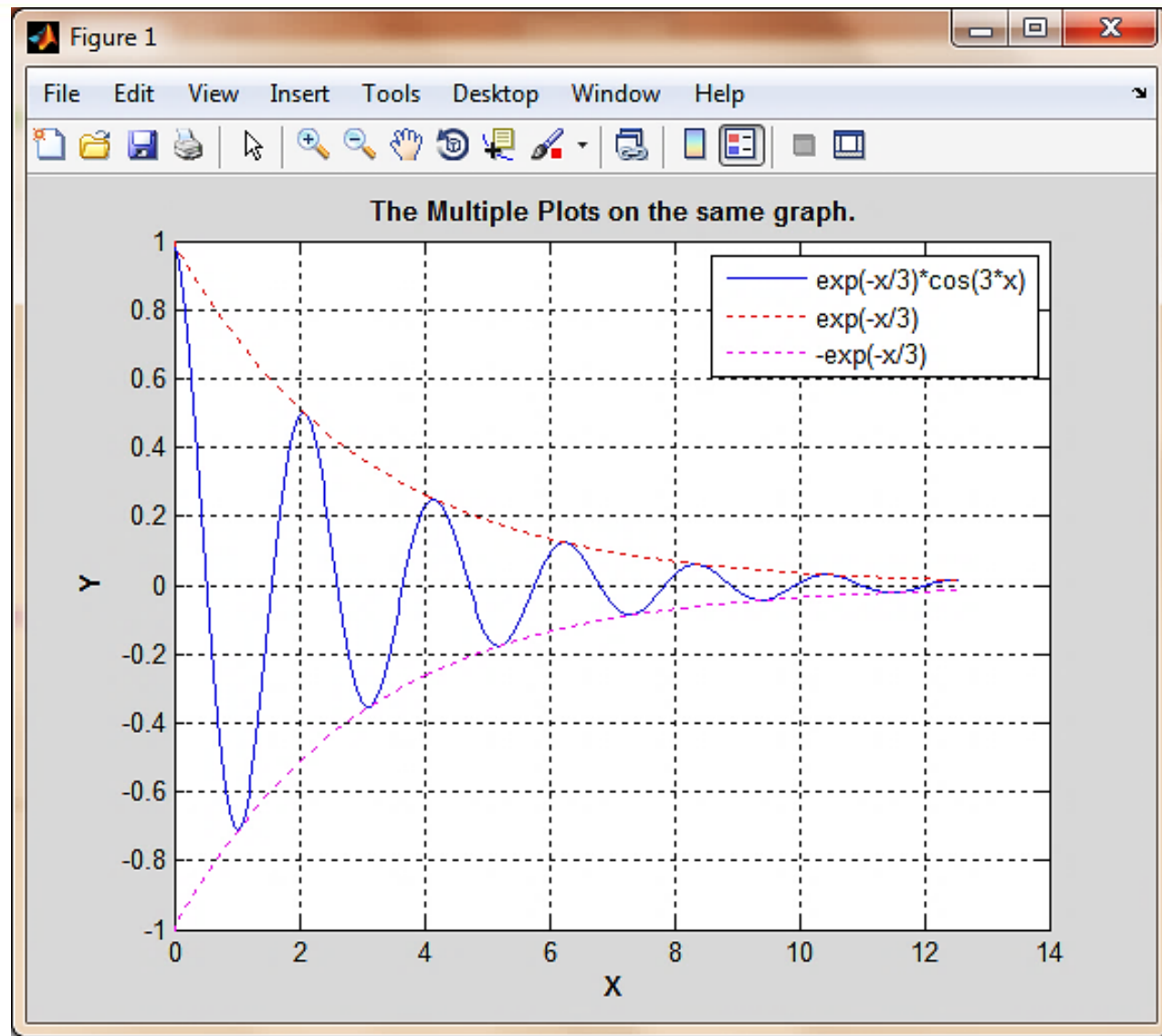
2.8.4 *The annotation functions*

- `xlabel('text')` adds text beside the X-axis on the current axis.
- `ylabel('text')` adds text beside the Y-axis on the current axis.
- `title('text')` adds text at the top of the current axis.
- `legend('string1', 'string2', ...)` displays a legend in the current axes using the specified strings to label each set of data.

Example 3:

```
% Scrip file name : plot2.m
% Multiple plots on the same graph.
X = 0:pi/100:4*pi;
Y1 = exp(-X/3);
Y = exp(-X/3).*cos(3*X);
% plot(X,Y,'b-',X,Y1,'r:',X,-Y1,'m:');
% equivalent to three plot() calls.
plot(X,Y,'b-');
hold on;
plot(X,Y1,'r:');
plot(X,-Y1,'m:');
title('\bf The Multiple Plots on the same graph. ');
grid on;
xlabel('\bf X');
ylabel('\bf Y');
axis([0 14 -1.0 1.0]);
legend('exp(-x/3)*cos(3*x)', 'exp(-x/3)', '-exp(-x/3)');
```

Example 3: Output



2.8 Introduction to plotting

2.8.5 Enhanced control of text strings

`\bf` Boldface

`\it` Italics

`\rm` Restore normal font

`\fontname` Specify the font name.

`\fontsize` Specify font size.

`_{\xxx}` The characters inside the braces are subscripts

`^{\xxx}` The characters inside the braces are
superscripts (see p.115)

Examples:

`'\bf{B}_ {\it s}'` displays as B_s

`'\tau_ {\it n} versus \omega_ {\it m}'`

display as τ_n versus ω_m

Example 4 function plot3()

```
% M_File name p3d_1.m  Show the plot3() Function
% Plot the 3-D Curve  $r(t) = \{ t \cos(t), t \sin(t), t \}$ .
t = -10*pi:pi/100:10*pi;
x = t.*cos(t);
y = t.*sin(t);
h = plot3(x,y,t,'linewidth',2);
title('\bf 3D Curve  $u(t) = [ t \cos(t), t \sin(t), t ]$ 
      ');
grid on;
xlabel('\bf X');
ylabel('\bf Y');
zlabel('\bf t');
```

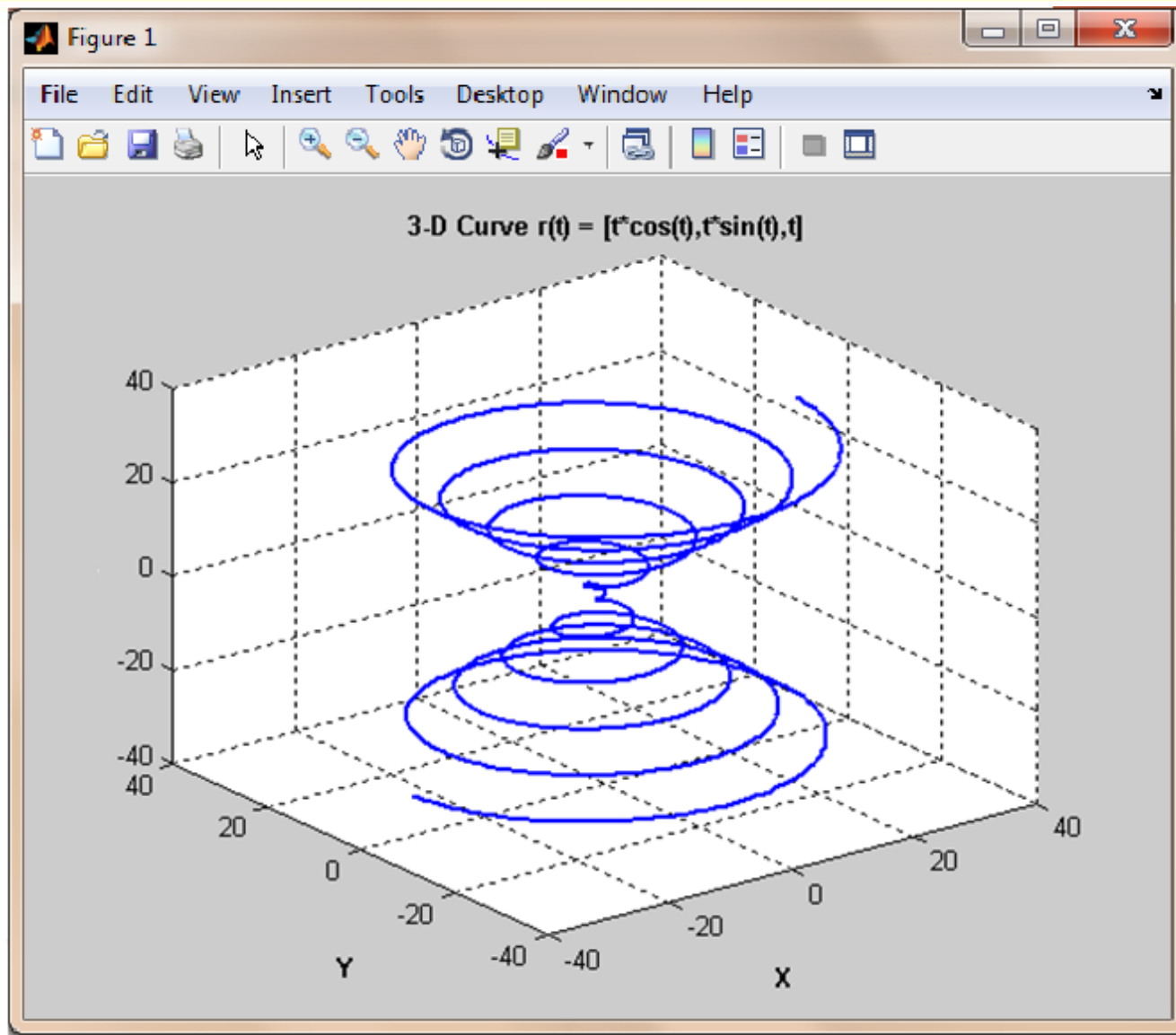
The M-file p3d_1.m

```
% Script File name p3d_1.m Shows plot3() Function
% Plot the 3-D Curve  $r(t) = \{t \cos(t), t \sin(t), t\}$ .
t = -10*pi:pi/100:10*pi;
x = t.*cos(t);
y = t.*sin(t);
h = plot3(x, y, t, 'linewidth', 2);
title(' \bf 3-D Curve  $r(t) = [t \cos(t), t \sin(t), t]$  ');
grid on;
xlabel(' \bf X ');
ylabel(' \bf Y ');
zlabel(' \bf t ');
% print out the figure in bitmap format.
% The file name is prntfl.map
print -dbitmap prntfl;
```

Format option: bitmap

Filename

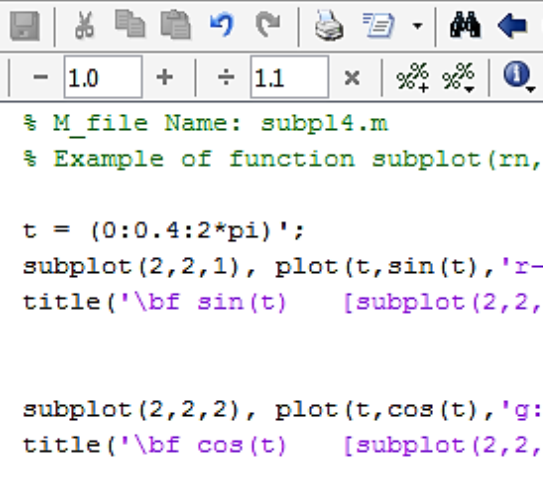
>> p3d_1<Enter>



2.8 Introduction to plotting

2.8.6 Putting several graphs in one window

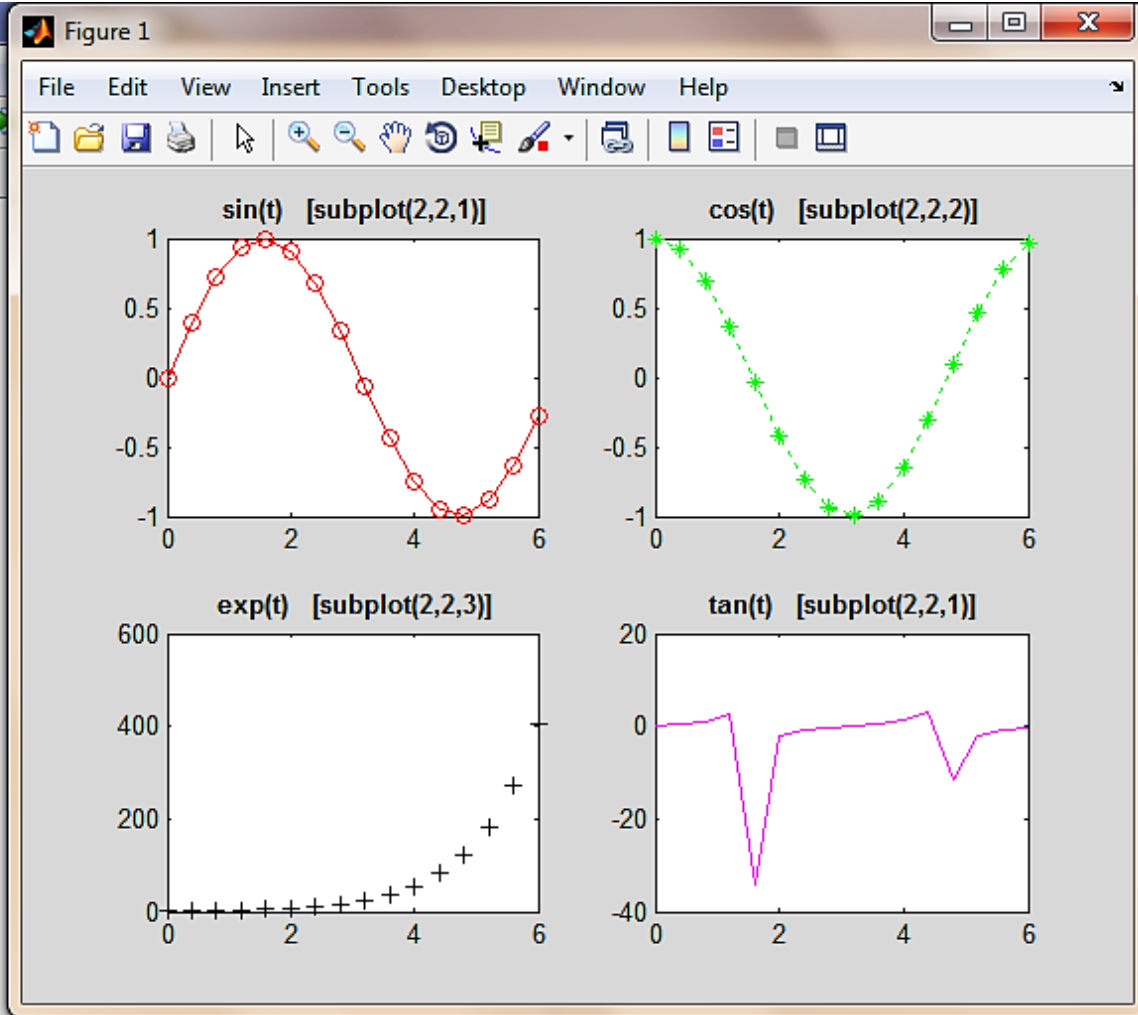
- Sometime it is convenient to have multiple plots in the same figure window. The command ***subplot(m,n,p)*** divides the current figure window into an ***m* \times *n*** matrix of plotting areas and chooses the ***p*th** one to be active.
- The ***subplot(m,n,i)*** command creates ***m* \times *n*** plots in the same figure window, arranged in an array with ***m*** rows and ***n*** columns. It also sets the next plot command to go to the ***i*th** coordinate system (counting across the rows).



```
Editor - E:\Study\Matlab\subplot4.m

File Edit Text Go Cell Tools Debug Desktop Window

1 % M_file Name: subplot4.m
2 % Example of function subplot(rn, cn, n)
3
4 t = (0:0.4:2*pi)';
5 subplot(2,2,1), plot(t,sin(t),'r-o');
6 title('\bf sin(t) [subplot(2,2,1)]');
7
8
9 subplot(2,2,2), plot(t,cos(t),'g:*');
10 title('\bf cos(t) [subplot(2,2,2)]');
11
12
13 subplot(2,2,3), plot(t,exp(t),'k+');
14 title('\bf exp(t) [subplot(2,2,3)]');
15
16
17 subplot(2,2,4), plot(t,tan(t),'m-');
18 title('\bf tan(t) [subplot(2,2,1)]');
```



Summary : The steps of plotting

The procedure of plotting a XY curve includes the following steps:

1. prepare x and y data set with (:) operator or function `linspace(x1,x2,n)` that generates n points between x1 and x2 linearly spaced vector.
2. Plot the curve with `plot()`
3. Annotate the graphics with annotation function.

2.9 Good practices

- Use meaningful variable names whenever possible.
- Create a data dictionary for each program to make program maintenance easier.
- Use only lowercase letters in variable names to avoid capitalization differences errors.
- Use a semicolon at the end of all MATLAB assignment statements, and remove the semicolon when debugging.
- Save MATLAB data in ASCII or MAT format for different programs.
- Use parentheses as necessary to make equations clear and easy to understand.
- Always include the appropriate units with any values that you read or write in a program.

Homework 1

HW1-1. Write a script (M-file) to make the following matrices.

1. Create a 4x4 identity matrix A.

2. Take the transpose of the given array B $B = [101 \ 91 \ 81 \ 71 \ 61 \ 51 \ 41 \ 31 \ 21 \ 11]$

$$C = \begin{bmatrix} 1.3 & -0.4 & 0.3 & 0.1 & 2.1 \\ 3.4 & 2.8 & 6.6 & -1.1 & 1.0 \\ 0.5 & 0.0 & -3.5 & 2.5 & 6.0 \\ -2.0 & 1.1 & 1.0 & 5.1 & 1.1 \\ 6.8 & -1.5 & -3 & 3.2 & 0 \end{bmatrix}$$

3. Create a matrix C and find size of the matrix C

4. Using Array Accessing and Sub-Array, Assign a scalar 14 in matrix C (as given)

$$\begin{bmatrix} 1.3 & 14 & 0.3 & 14 & 2.1 \\ 3.4 & 2.8 & 6.6 & -1.1 & 1.0 \\ 0.5 & 14 & -3.5 & 14 & 6.0 \\ -2.0 & 1.1 & 1.0 & 5.1 & 1.1 \\ 6.8 & 14 & -3 & 14 & 0 \end{bmatrix}$$

5. Using Array Accessing and Sub-Array, Eliminate second and fourth rows

$$E = \begin{bmatrix} 1.3 & 14 & 0.3 & 14 & 2.1 \\ 0.5 & 14 & -3.5 & 14 & 6.0 \\ 6.8 & 14 & -3 & 14 & 0 \end{bmatrix}$$

6. if $C \times X = I$, find X using the 'inv()' command and using '\' operator. Consider I as 5x5 identity matrix *

Use matrix C from 3

7. $F = \left[\frac{1}{7}, \frac{2}{7}, \frac{3}{7}, \frac{4}{7}, \frac{5}{7}, \frac{6}{7}, 1 \right]$ *Use ':' colon operator

Homework 1

HW1-2. Please write a MATLAB script file to complete the following functions.

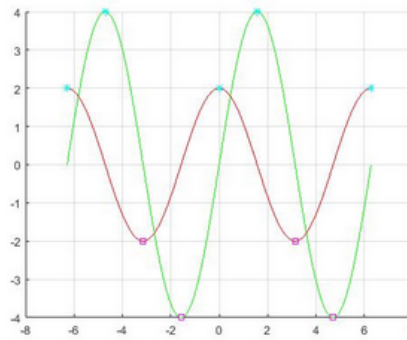
1. Generate a two-dimensional random array G of 10 rows and 8 columns. Use Matlab input function to enter rows and columns
2. Apply round function on the array G [New Array H]
3. In array H , how many 1s & 0s are present. Use Matlab output function to display your results

HW1-3. Write a script (M-file) for the following

```
a = (-360:360)*pi/180;
```

```
b = 4*sin(a);
```

1. Plot b against a with black line, draw marker style [red *] where the values of b are positive, and marker style [green circle] where the values of b are negative on the same graph.
2. Now create another variable c equal to the $2*\cos$ of a . i.e. $c = 2*\cos(a)$; Plot c against a with red line.
3. Plot both the graphs on the same figure and show their maximum and minimum values as shown below [use marker style (cyan *) for maximum values and marker style (magenta square) for minimum values]



How to submit homework



Go to **<http://www.bhmatlab.cn>**



Your username and initial password are all your student ID. After login, you can modify them if you need.



After entering the homepage of the website, you can see four sections.



user management

As for the user management in the upper left corner, hereby you can change the password or log out safely.



Matlab Programming



User management

Change password

Logout

SY8888888

Course introduction

Teaching team

Courseware resources

Homework management

Course introduction

At the section of Course introduction, you can browse the course outline, and then we'll briefly introduce MATLAB Programming and our course chapters.

Course introduction

Teaching team

Courseware resources

Homework management

Course outline

MATLAB is a powerful computing environment and tools for engineers. This course introduce it to students as a useful computing tool for their future use, so the objective of this course is:

Upon completion of this course, students will be able to solve basic numerical computational problems by implementation in MATLAB. In particular, students will:

- 1 Be able to using MATLAB interactive computational environment to solve simple calculation problem.
- 2 Be able to write, run and debug MATLAB program (M-file).
- 3 Be able to divide a large problem into a set of easier sub-problems and to implement MATLAB function or function functions to solve it.
- 4 Be able to plot two-dimensional or three-dimensional graphics using MATLAB graphic system for data visualization.
- 5 Be able to create a simple GUI.

CONTENTS are as follows.

Chapter 1 An Overview of MATLAB

1.1 The development of MATLAB

1.2 The Advantages of MATLAB

Courseware resources

Click Courseware resources, we provide the courseware used in teaching.



Matlab Programming



User management

instruction

Course introduction

Teaching team

Courseware resources

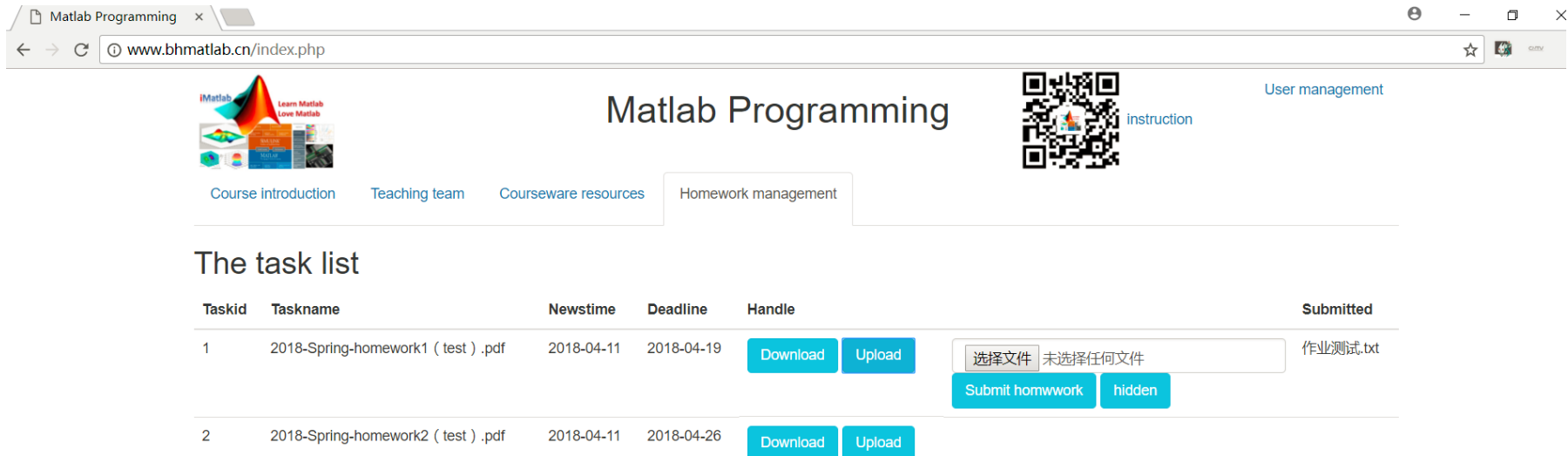
Homework management

The courseware list

Fileid	Filename	Upload time	Handle
1	2018-Spring-Lecture 1 An overview of MATLAB.pdf	2018-04-11	Download

Homework management

Click Homework management, here you can download the task and upload the solution. Click the button Upload, it will display the hidden upload part. **Please pay attention to the deadline and make sure that you submit your homework in time.** If the deadline passed, the button Upload will disappear automatically.



The screenshot shows a web browser window with the URL www.bhmatlab.cn/index.php. The page title is "Matlab Programming". The navigation menu includes "Course introduction", "Teaching team", "Courseware resources", and "Homework management" (which is highlighted). There is a QR code labeled "instruction" and a link for "User management".

The main content area is titled "The task list" and contains a table with the following data:

Taskid	Taskname	Newstime	Deadline	Handle	Submitted
1	2018-Spring-homework1 (test).pdf	2018-04-11	2018-04-19	Download Upload <input type="button" value="选择文件"/> 未选择任何文件 <input type="button" value="Submit homework"/> <input type="button" value="hidden"/>	作业测试.txt
2	2018-Spring-homework2 (test).pdf	2018-04-11	2018-04-26	Download Upload	



北京航空航天大学
BEIHANG UNIVERSITY

Thanks