MATLAB and its Application in Engineering

Assoc. Prof. Kirin Shi

Shanghai Jiao Tong University

About this course

About Myself

- 1998-2002 BEng National University of Defense Technology
- 2002-2005 MSc Central South University

- 2005-2010 PhD Manchester University
  Liverpool University, UK

- 2010-2013 Asst. Prof. Shanghai Jiao Tong University
- 2014- Assoc. Prof. Shanghai Jiao Tong University

Room 511, Building A, kirinshi@sjtu.edu.cn

About Myself

About this course

Course Structure

1. Introduction to Matlab
2. Vector and Matrix
3. Data type, input output
4. Script & Function Files
5. Relational & Logical operations; Conditional Control Structures
6. 2D, 3D Plots
7. Symbolic math; Polynomials
8. Curve Fitting; Interpolation
9. Linear Programming
10. Image processing

About this course

References

1. MATLAB : an introduction with applications, Amos Gilat Hoboken, N.J. ; Chichester : Wiley 4th ed., International student ed.. C2011
2. Linear Programming with Matlab, Michael C. Ferris, Olvi L. Mangasarian, Stephen J. Wright. MPS-SIAM, 2007
3. Digital Image Processing Using MATLAB, Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins, Gatesmark, 2009
4. Brian Hahn, Dan Valentine, Essential MATLAB for Engineers and Scientists, Academic Press,2010

About this course

Useful websites

http://www.mathworks.co.uk/matlabcentral/fileexchange/

FTP site:

- ftp://public.sjtu.edu.cn “public-files”→ “Matlab”
- user name: kirinshi password: public

  PPT slides; Books; Assignments

Teaching Assistant

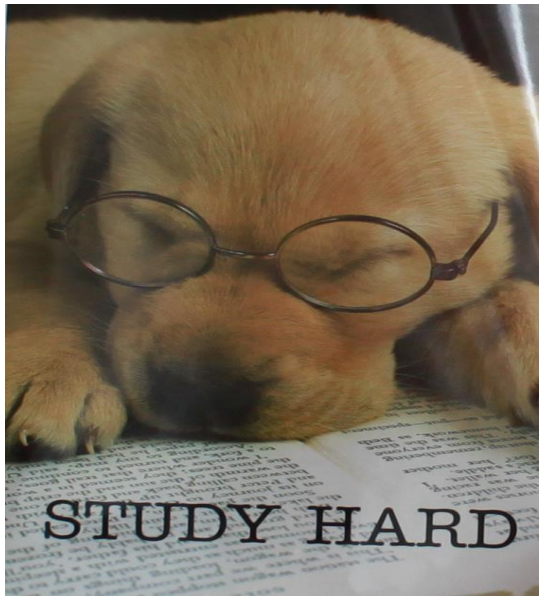
Mei Di 13641990275 572950987@qq.com

How will your study be evaluated

Class attendance	10%
Individual Assignment	20%
Project design (Teamwork) 10+15+15	40%
Final term	30%

About this course

- Participate actively in every class
- Interrupt me whenever you have a question
- PLAGIARISM will be given ZERO credit for the assignment
- Work effectively as a team player
- Show initiative to projects assigned
- Use only English in the class
- Study hard and play hard
- DO NOT eat, talk and use phones in the class



About this course

- Matlab programming skills
- Self-learning skills
- Team work
- Finding sources and Referencing
- Presentation skills
- English language skills

SESSION 1

INTRODUCTION TO MATLAB

MATLAB Basics

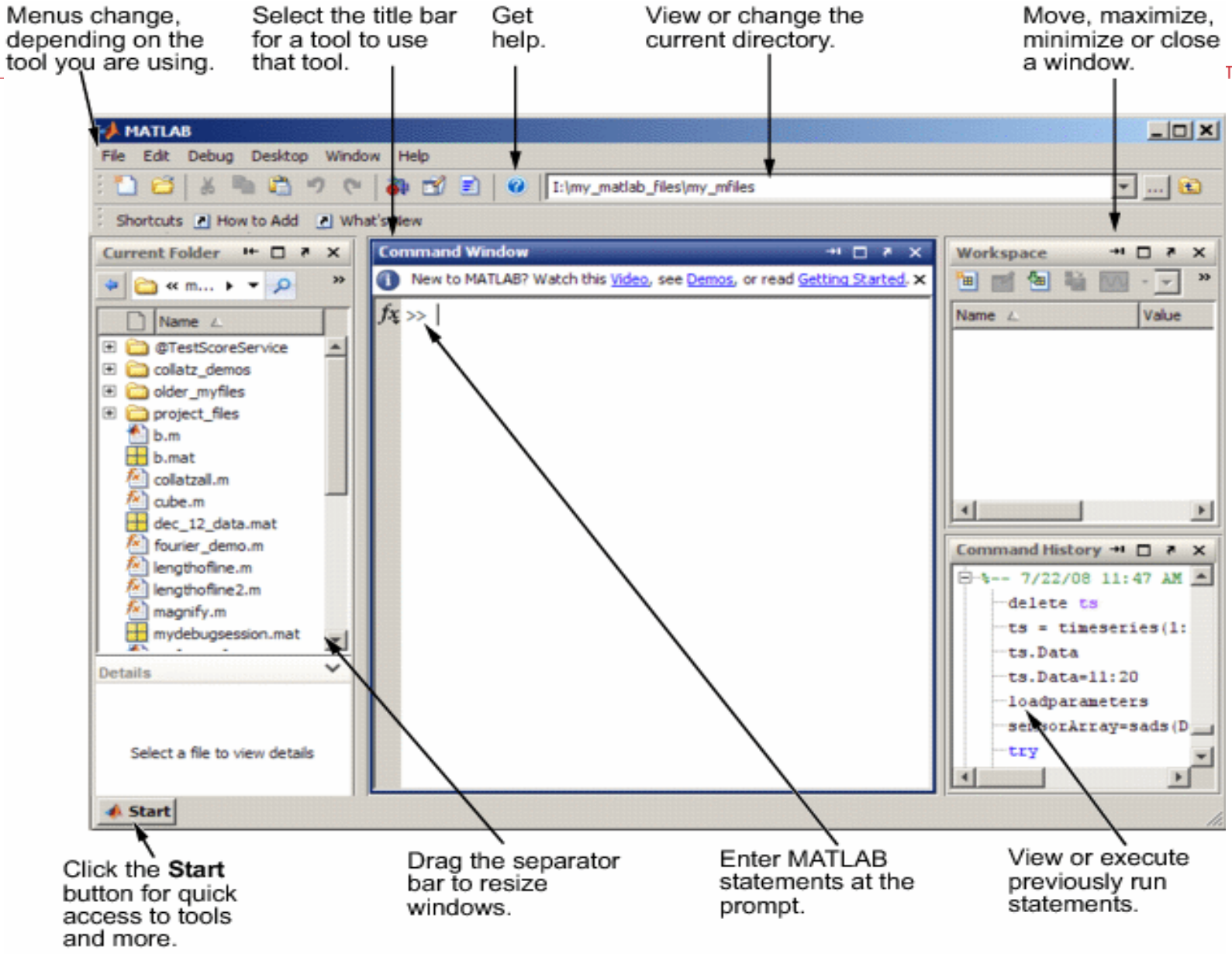
What is Matlab

- A computer program for engineering and scientific calculations.
- Popular because of
  - Ease of use
  - Excellent graphical capabilities
- Short for **MAT**rix **LAB**oratory
- Started life in mid 1980s as a program to perform various matrix operations.
- Now, it is flexible enough to be used in solving any technical problem.

MATLAB Desktop

- When you start MATLAB, a special window called the MATLAB desktop appears.

- MATLAB desktop is composed of the following windows:
  - Command Window
  - Command History Window
  - Current Directory Window
  - Workspace Window



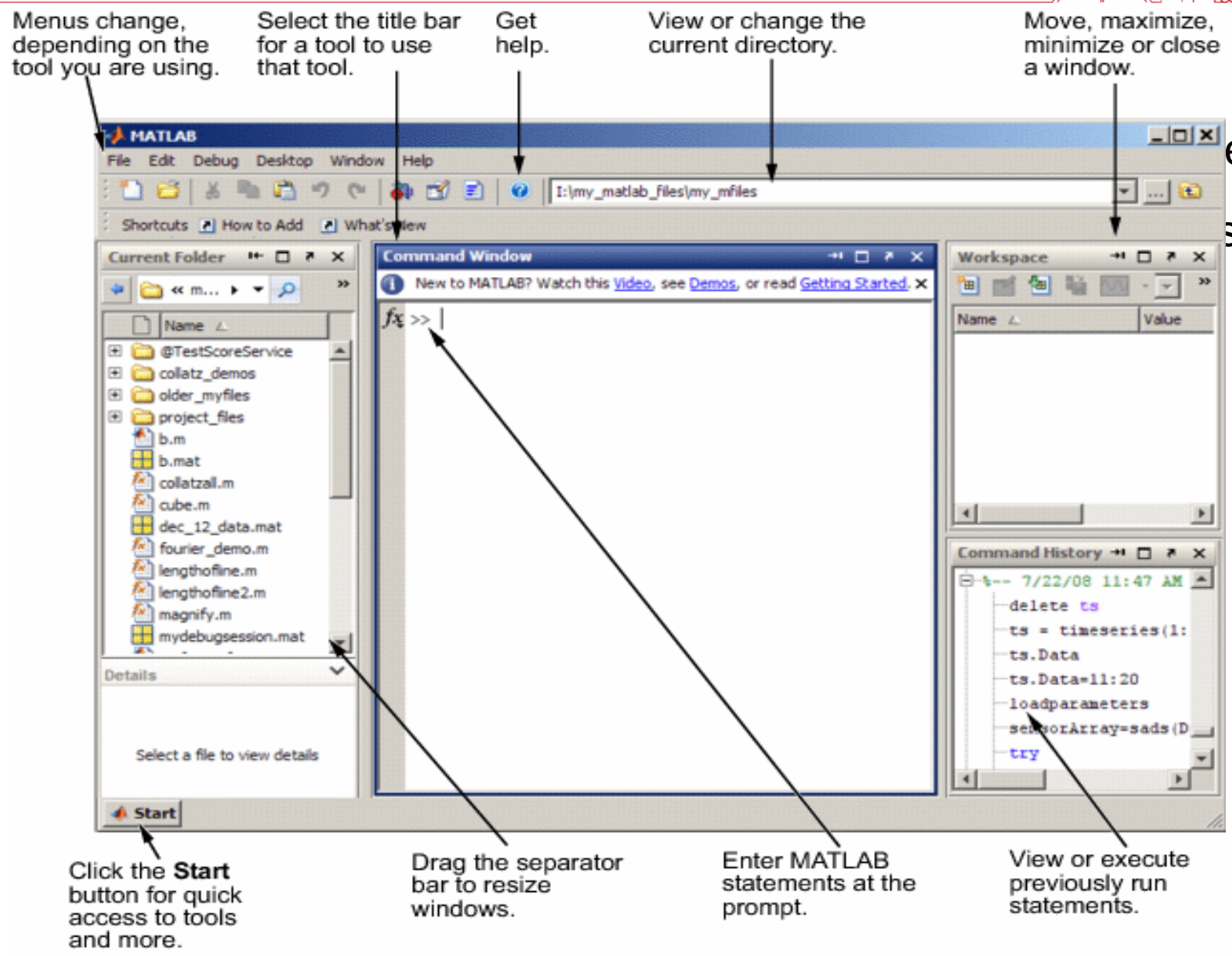
Default Layout of MATLAB DESKTOP

Window	Purpose
Command Window	Main window, enters variables, runs programs.
Figure Window	Contains output from graphic commands.
Editor Window	Creates and debugs script and function files.
Help Window	Provides help information.
Command History Window	Logs commands entered in the Command Window.
Workspace Window	Provides information about the variables that are used.
Current Folder Window	Shows the files in the current folder.

Command Window

- Main window of the MATLAB desktop.
- When MATLAB is ready to accept a new command, it shows the command prompt **>>** in this window.
- To execute a command, you should first type it at the command prompt and then press the **enter** key.

Command Window



Command Window

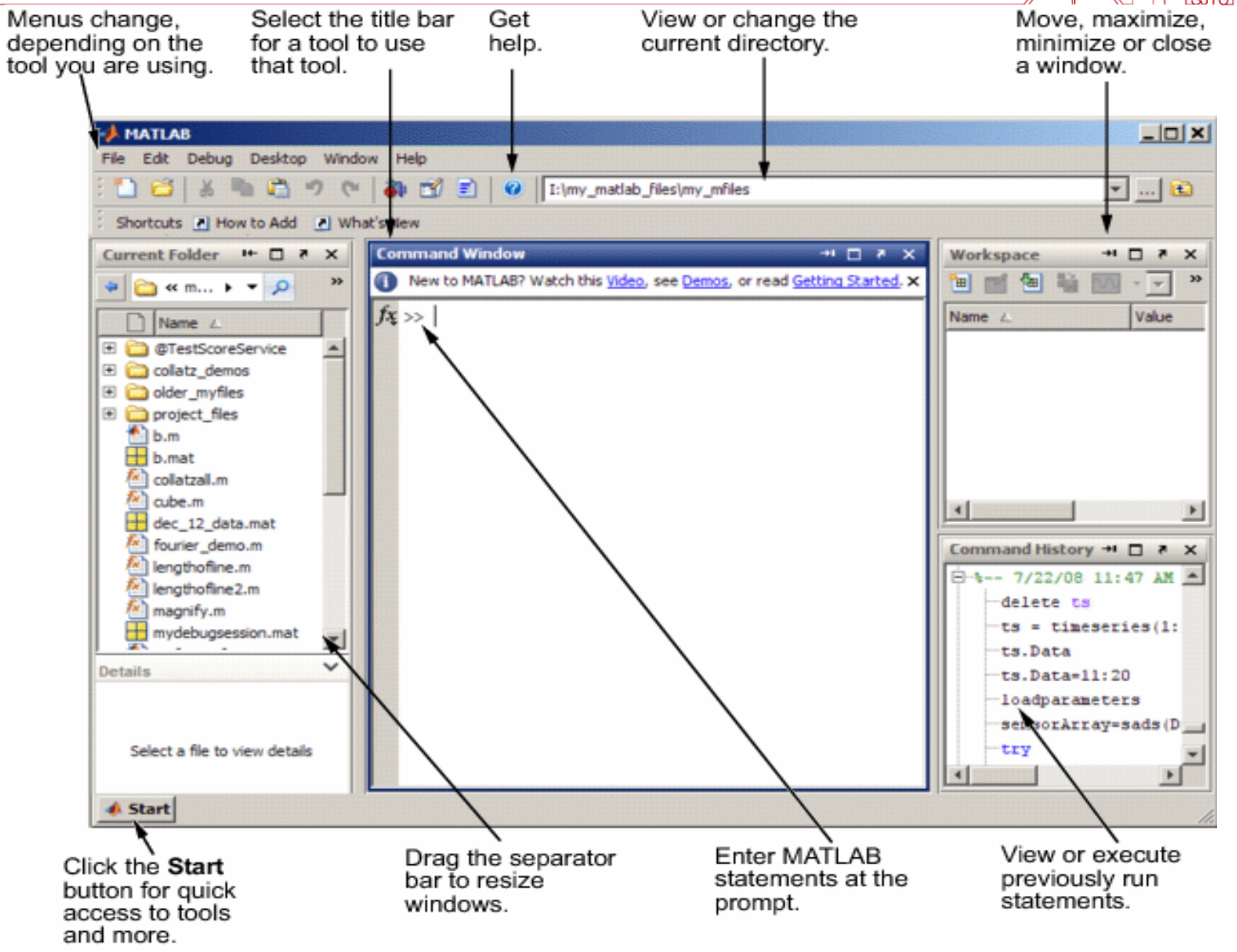
- Typing %

When the symbol % (percent) is typed at the beginning of a line, the line is designated as a comment.

- The clc command (clear, close all)

The clc command (type clc and press **Enter**) clears the Command Window

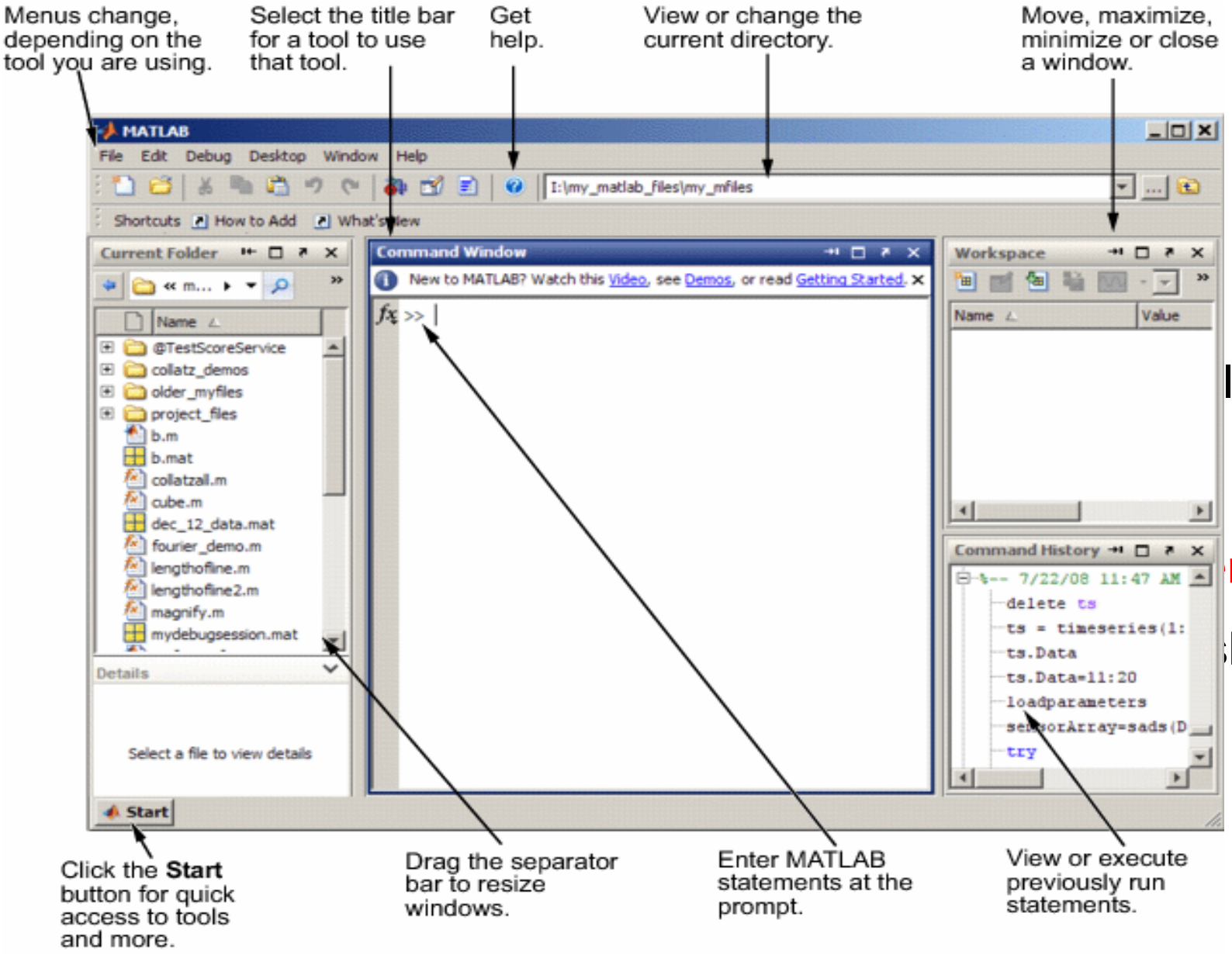
Command History Window



Command History Window

- To delete, copy or evaluate one or more commands from the Command History Window
  - Select the commands and right-click them with the mouse.
  - A popup menu will be displayed that allows you to delete, to copy or to evaluate the selected commands.

Current Directory Window

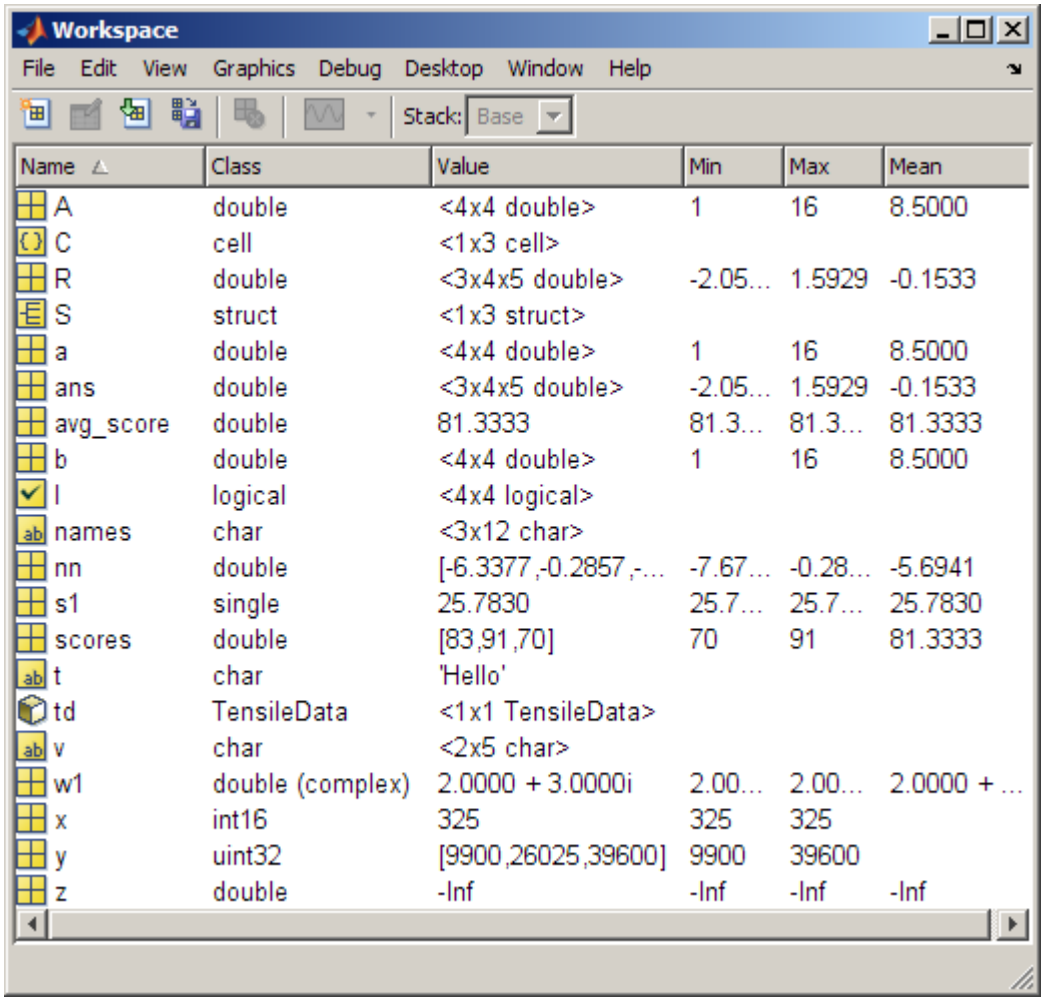


-
-
- le.
- rs’

sktop.

Workspace Window

- This window lists all the variables that you have generated together with their type & size.



Name	Class	Value	Min	Max	Mean
A	double	<4x4 double>	1	16	8.5000
C	cell	<1x3 cell>			
R	double	<3x4x5 double>	-2.05...	1.5929	-0.1533
S	struct	<1x3 struct>			
a	double	<4x4 double>	1	16	8.5000
ans	double	<3x4x5 double>	-2.05...	1.5929	-0.1533
avg_score	double	81.3333	81.3...	81.3...	81.3333
b	double	<4x4 double>	1	16	8.5000
l	logical	<4x4 logical>			
names	char	<3x12 char>			
nn	double	[-6.3377,-0.2857,-...	-7.67...	-0.28...	-5.6941
s1	single	25.7830	25.7...	25.7...	25.7830
scores	double	[83,91,70]	70	91	81.3333
t	char	'Hello'			
td	TensileData	<1x1 TensileData>			
v	char	<2x5 char>			
w1	double (complex)	2.0000 + 3.0000i	2.00...	2.00...	2.0000 + ...
x	int16	325	325	325	
y	uint32	[9900,26025,39600]	9900	39600	
z	double	-Inf	-Inf	-Inf	-Inf

Editor Window

- The Editor Window is used for writing and editing programs.

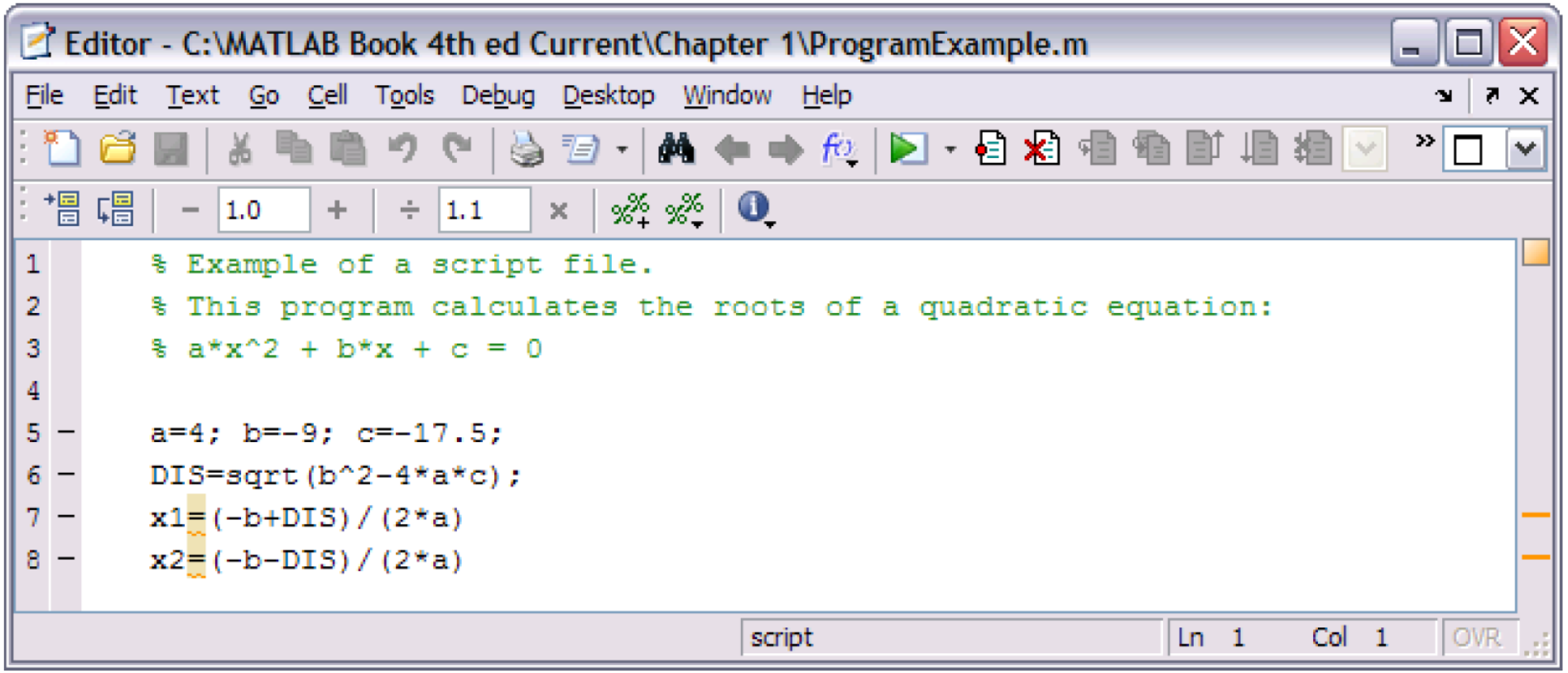
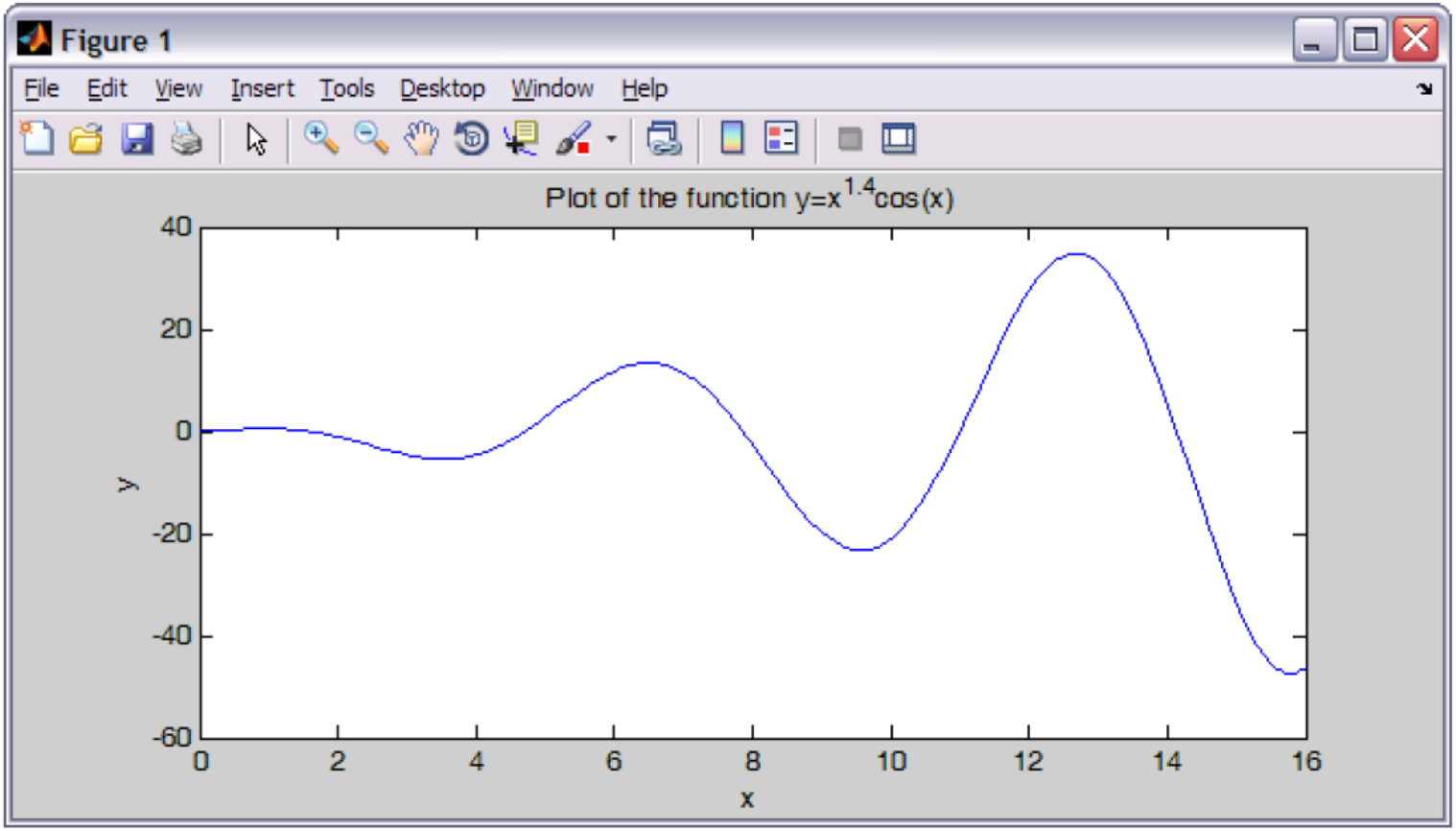


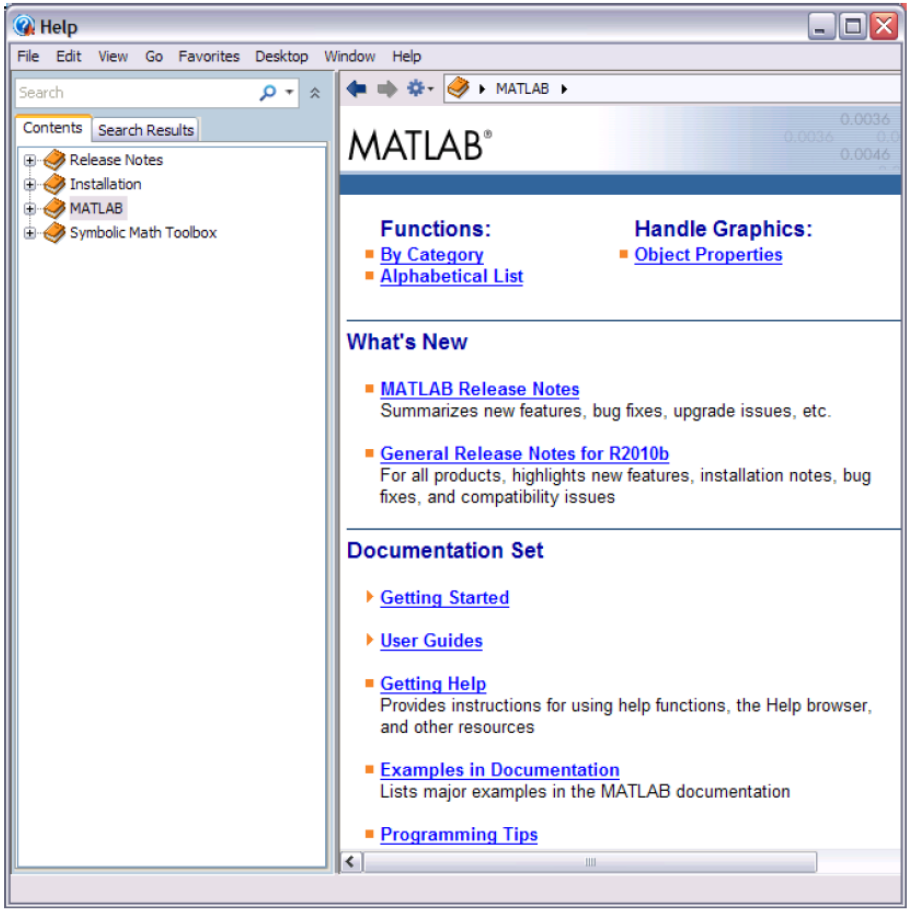
Figure Window

- The Figure Window opens automatically when graphics commands are executed, and contains graphs created by these commands.



Help Window

- The Help Window contains help information. This window can be opened from the **Help** menu in the toolbar of any MATLAB window. The Help Window is interactive and can be used to obtain information on any feature of MATLAB



End of
MATLAB Basics

Variable Assignment & Arithmetic Operations (Scalar)

Variables

Variable_name = A numerical value, or a computable expression

```
>> x=15
x =
     15

>> x=3*x-12
x =
     33
>>
```

The number 15 is assigned to the variable x.

MATLAB displays the variable and its assigned value.

A new value is assigned to x. The new value is 3 times the previous value of x minus 12.

Variables

Use previously defined variables to define a new variable

```
>> a=12
a =
    12
>> B=4
B =
     4
>> C=(a-B)+40-a/B*10
C =
    18
```

Assign 12 to a.

Assign 4 to B.

Assign the value of the expression on the right-hand side to the variable C.

Variables

With or without a semicolon

```
>> a=12;
>> B=4;
>> C=(a-B)+40-a/B*10;
>> C
C =
    18
```

The variables `a`, `B`, and `C` are defined but are not displayed since a semicolon is typed at the end of each statement.

The value of the variable `C` is displayed by typing the name of the variable.

Variables

Several assignments can be typed in the same line

```
>> a=12, B=4; C=(a-B)+40-a/B*10

a =

    12

C =

    18
```

The variable B is not displayed because a semi-colon is typed at the end of the assignment.

Variables

Variable that already exists can be reassigned a new value

```
>> ABB=72;
>> ABB=9;
>> ABB
ABB =
     9
>>
```

A value of 72 is assigned to the variable ABB.

A new value of 9 is assigned to the variable ABB.

The current value of the variable is displayed when the name of the variable is typed and the **Enter** key is pressed.

Variables

Once a pre-defined variable as an argument in functions

```
>> x=0.75;
>> E=sin(x)^2+cos(x)^2
E =
     1
>>
```

Variables

A number of frequently used variables are already defined when MATLAB is started.

ans A variable that has the value of the last expression that was not assigned to a specific variable.

pi The number π.

eps The smallest difference between two numbers. Equal to 2^(–52), which is approximately 2.2204e–016.

inf Used for infinity.

i Defined as , which is: 0 + 1.0000i.

j Same as i.

NaN Stands for Not-a-Number. Used when MATLAB cannot determine a validnumeric value. Example: 0/0.

MATLAB as a Calculator

- SJTU is purchasing some fancy cars as prize for outstanding students

Type	Cost per car	Number
	2.2M	2
	4.0M	2
	3.8M	2
	0.5M	15
	0.3M	15

Variables

>> 2 * 2.2 + 2*4.0+2*3.8+15*0.5+15*0.3

and the monitor would display

ans =

32M

- The **ans** (short for answer) is a ***variable***.
- The number **32** has been **assigne**d to the variable **ans**.
- The variable name **ans** is what MATLAB uses as a default.

Variables

- We might choose to assign the purchasing cost to a variable named **OurCost** by typing the following:

```
>> OurCost = 2 * 2.2 + 2*4.0+2*3.8+15*0.5+15*0.3
```

which would generate the output

```
OurCost =

    32
```

- In any assignment statement, the variable receiving some value must be on **the left-hand side** of the equal sign.

Variables

- Now you can use the variable name **OurCost** instead of **32** for further calculations.
- For example, if the car dealer offered SJTU a **10 percent discount**, the new cost would be equal to:

```
>> NewCost = OurCost * 0.9;
```

- Note that there is no output in this case;
- **Semicolon** at the end of the line suppressed the output.

Variables

- However, if we typed

>> **NewCost**

NewCost =

28.8

- which is, of course, 90% of the value stored in OurCost.

>> **Kirin**

??? Undefined function or variable **‘Kirin'.**

Variable Names-Rules

- Always start a variable name with a letter
- Can be up to 63 characters long
- Can contain letters, digits, and the underscore character
- MATLAB is case sensitive: it distinguishes between uppercase and lowercase letters

Variable Names-Rules

- Cannot contain punctuation characters (e.g., period, comma, semicolon)
- No spaces are allowed between characters
- Do **not** use MATLAB reserved words (such as **length**, **sum**, **end**, **pi**, **i** , **j** , **eps**, **sin**, **cos** and **size**).
- Names should convey some meaning (e.g., a diameter might be stored in a variable named **dia**).

Variable Names-Rules

- ***20 Predefined Variables and Keywords***

>> iskeyword

break case catch classdef

continue else elseif

end for function global

if otherwise parfor

persistent

return spmd

switch try while

Variable Names

- Use the underscore (_) to make your variables more readable.
  - Use **Estimated_cost** rather than **Estimatedcost**. (both names are valid).
  - Alternatively: **EstimatedCost**

- Do **not** use **space** in variable names

>> Estimated Cost = 5

- **???** Undefined function or method 'Estimated' for input arguments of type 'char'.

Variable Names

- Remember that only the first **63 characters** in a variable name are significant.

- MATLAB is **case sensitive**

```
>> NewCost

NewCost =

    28.8

>> Newcost

??? Undefined function or variable 'Newcost'.
```

MATLAB as a Calculator

- Car purchasing problem.

Type	Cost per car	Number
Maserati	2.2M	2
Lamborghini	4.0M	2
Aston Martin	3.8M	2
Mini Coopers	0.5M	15
Beetle	0.3M	15

- >> **OurCost** = 2 * 2.2 +2*4.0+2*3.8+15*0.5+15*0.3

More Variables

>> NumMasti = 2; NumLambo = 2; NumAston = 2;

NumMini = 15; NumBeetle = 15;

>> **OurCost** = NumMasti * 2.2 + NumLambo *4.0+ NumAston *3.8+

NumMini *0.5+ NumBeetle *0.3

OurCost =

32

Variables

If we decide to buy **3** Lamborghini cars rather than **2**

>> NumLambo = 3;

MATLAB does not automatically give new values to other variables that depend on **NumLambo**. In other words,

>> OurCost

OurCost =

36

>> OurCost = NumMasti * 2.2 + NumLambo *4.0+ NumAston *3.8+ NumMini *0.5+ NumBeetle *0.3

Variables

>> **OurCost** = NumMasti * 2.2 + NumLambo *4.0+ NumAston *3.8+ NumMini *0.5+ NumBeetle *0.3

OurCost =

36

- **Tips**

  Rather than retyping the above command:

  - Scroll backward through your commands (by using the arrow key ↑) until returning to the command you wish and then press the enter key.
  - Alternatively, locate the command in the **Command History Window** and double-click it with the left mouse button.

Useful Commands for Managing

Command	Outcome
`clear`	Removes all variables from the memory.
`clear x y z`	Removes only variables `x`, `y`, and `z` from the memory.
`who`	Displays a list of the variables currently in the memory.
`whos`	Displays a list of the variables currently in the memory and their sizes together with information about their bytes and class (see Section 4.1).

Symbols for arithmetic operations

Operation	Symbol	Example
Addition	+	5 + 3
Subtraction	–	5 – 3
Multiplication	*	5 * 3
Right division	/	5 / 3
Left division	\	5 \ 3 = 3 / 5
Exponentiation	^	5 ^ 3 (means $5^3 = 125$)

Arithmetic Operations

- A common mistake is to omit the multiplication operator. For example,

```
>> b = 5(2*3)

??? b = 5(2*3)

Error: Unbalanced or unexpected parenthesis or bracket.
```

The correct syntax is

```
>> b = 5*(2*3)

b =

    30
```

Precedence of operators

- When an expression involves multiple operators
  - It may not be clear which operator should be applied first.
  - To avoid ambiguity, all operators have an associated precedence.
  - The operator with the highest precedence is applied first, followed by the second highest, and so on.

Precedence of Operators

- When an expression involves multiple operators
  - Operators of the same precedence are evaluated from left to right.
  - Use parentheses **( )** to make sure that the operations are carried out in the correct order.

Precedence of Operators

1.All bracketed expressions are evaluated first, starting from the **innermost** brackets and working out.

2.All **exponentiations** are evaluated, working **from left to right**.

3.All **multiplications** and **divisions** are evaluated, working from left to right.

4.All **additions** and **subtractions** are evaluated, working from left to right.

Precedence of operators

```
>> X = 2*(1+2/3) + 2.5*(3*(23+14.7- 4/6)/3.5 + 5) + 1

X =

   96.1905
```

(23+14.7- 4/6) is calculated first.

23+14.7- **0.6667**= **37.0333**

Therefore,

X = 2*(1+2/3) + 2.5*(3***37.0333**/3.5 + 5) + 1

Precedence of operators

X = 2*(**1+2/3**) + 2.5*(3*37.0333/3.5 + 5) + 1

X = 2*1.6667 + 2.5*(**3*37.0333**/3.5 + 5) + 1

X = 2*1.6667 + 2.5*(**111.0999/3.5** + 5) + 1

X = 2*1.6667 + 2.5*(**31.7428 + 5**) + 1

X = **2*1.6667** + 2.5*36.7428 + 1

X = 3.3334 + **2.5*36.7428** + 1.

X = 3.3334 + 91.8570 + 1

X =

96.1904

Precedence of operators

```
>> X = 2*(1+2/3) + 2.5*(3*(23+14.7- 4/6)/3.5 + 5) + 1

X =

   96.1905

>> x1 = 23+14.7- 4/6;

>> y1 = 3*x1/3.5 + 5;

>> y2 = 1+2/3;

>> X = 2*y2 + 2.5*y1 + 1

X =

   96.1905
```

Display the result

Command	Description	Example

`format short`	Fixed-point with 4 decimal digits for: $0.001 \le number \le 1000$ Otherwise display format `short e`.	`>> 290/7`
		`ans =`
		`41.4286`

`format long`	Fixed-point with 15 decimal digits for: $0.001 \le number \le 100$ Otherwise display format `long e`.	`>> 290/7`
		`ans =`
		`41.428571428571431`

`format short e`	Scientific notation with 4 decimal digits.	`>> 290/7`
		`ans =`
		`4.1429e+001`

`format long e`	Scientific notation with 15 decimal digits.	`>> 290/7`
		`ans =`
		`4.142857142857143e+001`

Display the result

`format short g`	Best of 5-digit fixed or floating point.	`>> 290/7`
		`ans =`
		`41.429`

`format long g`	Best of 15-digit fixed or floating point.	`>> 290/7`
		`ans =`
		
		`41.4285714285714`

`format bank`	Two decimal digits.	`>> 290/7`
		`ans =`
		`41.43`

`format compact`	Eliminates empty lines to allow more lines with information displayed on the screen.	
`format loose`	Adds empty lines (opposite of `compact`).	

Elementary math 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`

Elementary 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/6)`
		`ans =`
		`0.5000`

`cos(x)`
`cosd(x)`
	Cosine of angle *x* (*x* in radians).
	Cosine of angle *x* (*x* in degrees).
		`>> cosd(30)`
		`ans =`
		`0.8660`

`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), acos(x), atan(x),acot(x) returns the angle in radians

asind(x), acosd(x), atand(x), acotd(x) for the angle in degrees

Elementary math 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`

SESSION 2

Vector and Matrix

Vectors in MATLAB

Definition

- An **array**（数组） is a collection of data values organised into rows and columns.
- Although an **array** is composed of many elements, **a single name** is used to refer to it.
- The term **matrix**（矩阵） refers to an array with two or more dimensions.
- The term **vector** is usually used to describe an array with only one dimension.
- A vector can either be **horizontal** or **vertical**.

Defining a Horizontal Vector

variable_name = [type vector elements]

```
>> A = [10, 3, -5, 20]

A =

    10     3    -5    20

>> A = [10  3  -5  20]

A =

    10     3    -5    20

>> A(2)

ans =

     3
```

Defining a Horizontal Vector

```
>> A = [10  3  -5  20];

>> A(2)

ans =

     3

>> A(4)

ans =

    20

>> A(5)
```

??? Attempted to access **A(5)**; **index out of bounds** because numel(A)=4.

Column (Vertical) Vectors

- To define vertical vectors, separate rows by means of semicolons.

```
>> vertical_vector = [7; 5; 10]
 vertical_vector =
        7
        5
       10
```

- Alternatively, we can code the following

```
>> vertical_vector = [7
5
10];
```

Transpose

- A row vector can be converted into a column vector, and vice versa, by transposition.
- MATLAB performs this using an apostrophe (or prime ').

```
>> vertical_vector = [7; 5; 10];

>> horizontal_vector = vertical_vector '

 horizontal_vector =

      7 5 10
```

Vertical Vectors

- The command **Y = X(:)** converts the horizontal vector **X** into a vertical vector.
- On the other hand, if X is already a vertical vector, it leaves X unchanged.
- Therefore, the vector **Y** would always be a vertical vector, regardless of the state of X.

```
>> x = [1  3];

>> x = x(:)

x =

     1

     3
```

Functions and Arrays

```
>> x = [0 .1*pi   0.2*pi   0.3*pi   0.4*pi   0.5*pi]

x =

0   0.3142   0.6283   0.9425   1.2566   1.5708

>> y = sin(x)

y =

0   0.3090   0.5878   0.8090   0.9511   1.0000

>> z = 1 + x(3) + y(2)

z =

    1.9373
```

Horizontal (Row) Vectors: Colon Notation

Creating a vector with constant spacing by specifying the first term, the spacing, and the last term:

```
variable_name = [m:q:n]
```

or

```
variable_name = m:q:n
```

(The brackets are optional.)

- The shorthand **m:q:n** means **m** to **n** in steps of **q**.
- **1:1:5** is equivalent to t **[1,2,3,4,5]**
- **1:5** is equivalent to t **[1,2,3,4,5]**
- **2:3:12** is equivalent to t **[2,5,8,11]**
- **8:-1:3** is equivalent to **[8,7,6,5,4,3]**.

Horizontal (Row) Vectors: Colon Notation

Creating a vector with constant spacing by specifying the first term, the spacing, and the last term:

```
>> x=[1:2:13]
x =
     1     3     5     7     9    11    13
>> y=[1.5:0.1:2.1]
y =
   1.5000   1.6000   1.7000   1.8000   1.9000   2.0000   2.1000

>> z=[-3:7]

z =
    -3    -2    -1     0     1     2     3     4     5     6
7
>> xa=[21:-3:6]
```

First element 1, spacing 2, last element 13.

First element 1.5, spacing 0.1, last element 2.1.

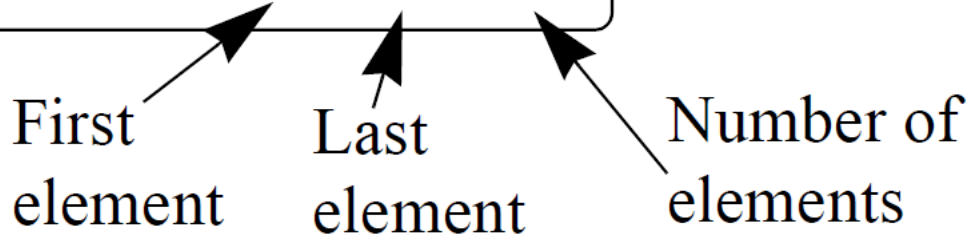
First element –3, last term 7.
If spacing is omitted, the default is 1.

First element 21, spacing –3, last term 6.

Function linspace

Creating a vector with linear (equal) spacing by specifying the first and last terms, and the number of terms

```
variable_name = linspace(xi,xf,n)
```



- **linspace** is a built-in MATLAB function. **linspace(Xi, Xf, N)** generates **N** equally-spaced points between **Xi** and **Xf**, starting with **Xi** and ending with **Xf**.

Function linspace

linspace(X1, X2, N) generates **N** equally-spaced points between **X1** and **X2**, starting with **X1** and ending with **X2**.

```
>> x = linspace(0,6,5)

x =

         0    1.5000    3.0000    4.5000    6.0000
```

Alternatively, **>> x** = 0:1.5:6

```
x =

         0    1.5000    3.0000    4.5000    6.0000
```

Function linspace

Creating a vector with linear (equal) spacing by specifying the first and last terms, and the number of terms

```
>> va=linspace(0,8,6)
va =
       0    1.6000    3.2000    4.8000    6.4000    8.0000
>> vb=linspace(30,10,11)
vb =
    30    28    26    24    22    20    18    16    14    12    10
>> u=linspace(49.5,0.5)

u =
  Columns 1 through 10
   49.5000   49.0051   48.5101   48.0152   47.5202   47.0253
46.5303   46.0354   45.5404   45.0455
............
Columns 91 through 100
    4.9545    4.4596    3.9646    3.4697    2.9747    2.4798
1.9848    1.4899    0.9949    0.5000
>>
```

6 elements, first element 0, last element 8.

11 elements, first element 30, last element 10.

First element 49.5, last element 0.5.

When the number of elements is omitted, the default is 100.

100 elements are displayed.

Accessing elements of an array

The address of an element in a vector is its position in the row (or column). For a vector named ve, ve(k) refers to the element in position k.

1. For example, if the vector *ve* has nine elements:

ve = 35 46 78 23 5 14 81 3 55

then

ve(4) = 23, *ve*(7) = 81, and *ve*(1) = 35.

Accessing elements of an array

- A single vector element, *v*(*k*), can be used just as a variable.
- It is possible to change the value of only one element of a vector by assigning a new value to a specific address *v*(*k*) = *value*
- A single element can also be used as a variable in a mathematical expression.

```
>> VCT=[35 46 78 23 5 14 81 3 55]
VCT =
    35    46    78    23     5    14    81     3    55
>> VCT(4)
```

Define a vector.

Display the fourth element.

Accessing elements of an array

- A single vector element, *v*(*k*), can be used just as a variable.
- It is possible to change the value of only one element of a vector by assigning a new value to a specific address *v*(*k*) = *value*
- A single element can also be used as a variable in a mathematical expression.

```
ans =
    23
>> VCT(6)=273
VCT =
    35    46    78    23     5   273    81     3    55


>> VCT(2)+VCT(8)
ans =
    49
>> VCT(5)^VCT(8)+sqrt(VCT(7))
ans =
   134
>>
```

Assign a new value to the sixth element.

The whole vector is displayed.

Use the vector elements in mathematical expressions.

Accessing elements of an array

```
>> x = [0 .1*pi   0.2*pi   0.3*pi   0.4*pi   0.5*pi]

x =

0   0.3142   0.6283   0.9425   1.2566   1.5708

>> z = x([1  3  5])

z =

        0    0.6283    1.2566

>> z = x(1:2:5)

z =

        0    0.6283    1.2566
```

Accessing elements of an array

- *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*.
- *va*(*m*:*q*:*n*) Refers to elements from *m to n with a step of q* .

```
>> v=[4 15 8 12 34 2 50 23 11]
v =
     4    15     8    12    34     2    50    23    11
>> u=v(3:7)
u =
     8    12    34     2    50
>>
```

A vector v is created.

A vector u is created from the elements 3 through 7 of vector v.

Accessing elements of an array

```
>> x = [0 .1*pi   0.2*pi   0.3*pi   0.4*pi   0.5*pi]

x =

0   0.3142   0.6283   0.9425   1.2566   1.5708

>> z = x(1:3:5)

z =

         0    0.9425

>> z = x(1:3:end)

z =

         0    0.9425
```

Accessing elements of an array

```
>> x = [0 .1*pi   0.2*pi   0.3*pi   0.4*pi   0.5*pi]

x =

0   0.3142   0.6283   0.9425   1.2566   1.5708

>> z = x(3:-1:1)

z =

 0.6283    0.3142         0

>> z = x(end:-1:1)

z =

1.5708   1.2566   0.9425   0.6283   0.3142  0
```

Long expressions

- If a vector is too long to easily fit on a single line, use the MATLAB continuation operator, "**...**“

```
>> x = [0    0.1*pi    0.2*pi ...
   0.3*pi      0.4*pi   0.5*pi]

x =

0  0.3142  0 .6283  0.9425  1.2566  1.5708
```

Adding elements

- Elements can be added to an existing vector by assigning values to the new elements.
- MATLAB assigns zeros to the elements that are between the last original element and the new element.

Adding elements

```
>> DF=1:4
DF =
     1     2     3     4
>> DF(5:10)=10:5:35
DF =
     1     2     3     4    10    15    20    25    30    35
>> AD=[5   7   2]
AD =
     5     7     2
>> AD(8)=4
AD =
     5     7     2     0     0     0     0     4
>> AR(5)=24
AR =
     0     0     0     0    24
>>
```

Define vector DF with 4 elements.

Adding 6 elements starting with the 5th.

Define vector AD with 3 elements.

Assign a value to the 8th element.

MATLAB assigns zeros to the 4th through 7th elements.

Assign a value to the 5th element of a new vector.

MATLAB assigns zeros to the 1st through 4th elements.

Combining Vectors

```
>> a = 1:5, b = 1:2:9

 a =

     1   2   3   4    5

b =

     1   3   5   7    9

>> c = [b  a]

c =

     1   3   5   7   9   1   2   3    4   5

>> d = [a(1:2:5) 1 0 1]
  d =

     1   3   5   1   0   1
```

Deleting elements

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

```
>> kt=[2 8 40 65 3 55 23 15 75 80]
kt =
    2    8   40   65    3   55   23   15   75   80
>> kt(6)=[]
kt =
    2    8   40   65    3   23   15   75   80
>> kt(3:6)=[]
kt =
    2    8   15   75   80
```

Define a vector with 10 elements.

Eliminate the 6th element.

The vector now has 9 elements.

Eliminate elements 3 through 6.

The vector now has 5 elements.

Some handy functions

Function	Description	Example

`length(A)`	Returns the number of elements in the vector `A`.	`>> A=[5 9 2 4];`
		`>> length(A)`
		`ans =`
		`4`

`size(A)`	Returns a row vector `[m,n]`, where `m` and `n` are the size $m \times n$ of the array `A`.	`>> A=[6 1 4 0 12; 5 19 6 8 2]`
		`A =`
		`6 1 4 0 12`
		`5 19 6 8 2`
		`>> size(A)`
		`ans =`
		`2 5`

Operations on arrays

- Can be divided into **scalar-array** operations and **array-array** operations.

scalar-array operations

```
>> A = [1  2  3  4  5];

>> C = A - 2
   C =
       -1   0   1   2   3

>> Z = 2 * A - 1

 Z =
        1   3   5   7   9
```

scalar-array operations

```
>> A = [1  2  3  4  5]

A =

    1    2    3    4    5

>> Z = 2 * A – 1,   >> Z = 2 .* A – 1

Z =

    1    3    5    7    9

>> Z = 2 * (A - 1),  >> Z = 2 .* (A - 1)

Z =

    0    2    4    6    8
```

Array-Array Operations

- For two vectors of the same size and orientation, "**.***", "**./**" and "**.^**" operators mean "**element by element**" multiplication, division and exponentiation, respectively.

```
>> A = [1, 2, 3]; B = [4, 5, 6];
>> P = A + B
P =
     5     7     9
```

Array-Array Operations

```
>> A = [1, 2, 3]; B = [4, 5, 6];
>> Q = A - B
Q =
    -3    -3    -3
>> C = A.*B
C =
     4    10    18
>> D = A./B
D =
    0.2500    0.4000    0.5000
```

Array-Array Operations

```
>> A = [1, 2, 3];  B = [4, 5, 6];
>> E = A.^B
E =
    1    32   729
```

A*B?

A/B?

A^B?

- Note the following commands involving a vector and a scalar

```
>> F = A.^2   But not  A^2
F =
    1     4     9
```

Array-Array Operations

```
>> A = [1, 2, 3]
>> T=2.*A     Or  2*A
T =
     2     4     6
>> S=A./2     Or A/2
S =
    0.5000    1.0000    1.5000
>> U=2./A     But not   2/A
U =
    2.0000    1.0000    0.6667
```

END OF
Vectors in MATLAB

MATRIX

DEFINITION

- A matrix is a two-dimensional array
- It is a collection of numbers organised into rows and columns.
- Each element of a matrix is identified by two indices referring to that element’s respective row and column numbers.
- The size of a matrix is specified by the number of its rows and the number of its columns, with the number of rows mentioned first.

DEFINITION

in MATLAB, matrix A can be entered as:

```
variable_name=[1st row elements; 2nd row elements; 3rd
               row elements; ... ; last row elements]
```

>> A = [1 2 3; 4 5 6];

Or

>> A = [1 2 3
4 5 6];

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

DEFINITION

If you type A at the command prompt, you will get the following response,

```
>> A

A =

     1     2     3

     4     5     6


>> A(2,3)

ans =

     6
```

DEFINITION

```
>> a=[5   35   43;   4   76   81;   21   32   40]
a =
     5     35     43
     4     76     81
    21     32     40
>> b = [7   2   76   33   8
1   98   6   25   6
5   54   68   9   0]
```

A semicolon is typed before a new line is entered.

The **Enter** key is pressed before a new line is entered.

```
b =
     7      2     76     33      8
     1     98      6     25      6
     5     54     68      9      0
>> cd=6; e=3; h=4;
>> Mat=[e, cd*h, cos(pi/3); h^2, sqrt(h*h/cd), 14]
Mat =
    3.0000    24.0000     0.5000
   16.0000     1.6330    14.0000
>>
```

Three variables are defined.

Elements are defined by mathematical expressions.

DEFINITION

The magic *colon* notation and *linspace*

```
>> A=[1:2:11; 0:5:25; linspace(10,60,6); 67 2 43 68 4 13]
A =
     1     3     5     7     9    11
     0     5    10    15    20    25
    10    20    30    40    50    60
    67     2    43    68     4    13
>>
```

Some special matrixes

The ***zeros(m,n)*** and the ***ones(m,n)*** commands create a matrix with *m* rows and *n* columns in which all elements are the numbers 0 and 1, respectively.

```
>> zr=zeros(3,4)

zr =

     0     0     0     0
     0     0     0     0
     0     0     0     0

>> ne=ones(4,3)

ne =

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

Some special matrixes

The ***eye(n)*** command creates a square matrix with *n* rows and *n* columns in which the diagonal elements are equal to 1 and the rest of the elements are 0. This matrix is called the identity matrix.

```
>> idn=eye(5)
idn =
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
>>
```

Notes about Variables in MATLAB

- All variables in MATLAB are arrays. A scalar is an array with one element, a vector is an array with one row or one column of elements, and a matrix is an array with elements in rows and columns.
- The variable (scalar, vector, or matrix) is defined by the input when the variable is assigned. **There is no need to define the size of the array** (single element for a scalar, a row or a column of elements for a vector, or a two-dimensional array of elements for a matrix) before the elements are assigned.
- Once a variable exists—as a scalar, vector, or matrix—it can be changed to any other size, or type, of variable.

Special matrices

>> Horizontal_vector = [1 3.15 7.8]

Horizontal_vector =

1.0000 3.1500 7.8000

>> Vertical_vector = [1; 3.15; 7.8]

Vertical_vector =

1.0000

3.1500

7.8000

Special matrices

- A scalar (single number) is a one-by-one matrix.
- A scalar does not need brackets.

- For example,

>> Gravitational_constant = 9.806;

The Transpose Operator

- The transpose operator, when applied to a vector, switches a row (column) vector to a column (row) vector.
- When applied to a matrix, it switches the rows (columns) to columns (rows).
- The transpose operator is applied by typing a single quote ’ following the variable to be transposed.

```
>> aa=[3   8   1]
aa =
     3     8     1
>> bb=aa'
```

Define a row vector `aa`.

Define a column vector `bb` as the transpose of vector `aa`.

The Transpose Operator

```
bb =
     3
     8
     1
>> C=[2 55 14 8; 21 5 32 11; 41 64 9 1]
C =
     2    55    14     8
    21     5    32    11
    41    64     9     1
>> D=C'
D =
     2    21    41
    55     5    64
    14    32     9
     8    11     1
>>
```

Define a matrix C with 3 rows and 4 columns.

Define a matrix D as the transpose of matrix C. (D has 4 rows and 3 columns.)

Addressing a matrix

- The address of an element in a matrix is its position, defined by the row number and the column number where it is located.
- As with vectors, it is possible to change the value of just one element of a matrix by assigning a new value to that element.

```
>> MAT=[3 11 6 5; 4 7 10 2; 13 9 0 8]
MAT =
     3    11     6     5
     4     7    10     2
    13     9     0     8
>> MAT(3,1)=20
MAT =
     3    11     6     5
     4     7    10     2
    20     9     0     8
>> MAT(2,4)-MAT(1,2)
ans =
    -9
```

Create a 3 × 4 matrix.

Assign a new value to the (3,1) element.

Use elements in a mathematical expression.

Addressing a matrix via colon notation

- A(:,n) Refers to the elements in all the rows of 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.

Addressing a matrix via colon notation

```
>> A=[1 3 5 7 9 11; 2 4 6 8 10 12; 3 6 9 12 15 18; 4 8 12 16
20 24; 5 10 15 20 25 30]

A =
     1     3     5     7     9    11
     2     4     6     8    10    12
     3     6     9    12    15    18
     4     8    12    16    20    24
     5    10    15    20    25    30

>> B=A(:,3)
```

Define a matrix A with 5 rows and 6 columns.

Define a column vector B from the elements in all of the rows of column 3 in matrix A.

Addressing a matrix via colon notation

```
B =
     5
     6
     9
    12
    15
```

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

```
C =
     2     4     6     8    10    12
```

Define a row vector `C` from the elements in all of the columns of row 2 in matrix `A`.

```
>> E=A(2:4,:)
```

```
E =
     2     4     6     8    10    12
     3     6     9    12    15    18
     4     8    12    16    20    24
```

Define a matrix `E` from the elements in rows 2 through 4 and all the columns in matrix `A`.

```
>> F=A(1:3,2:4)
```

```
F =
     3     5     7
     4     6     8
     6     9    12
```

Create a matrix `F` from the elements in rows 1 through 3 and columns 2 through 4 in matrix `A`.

Changing the value of a matrix

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

```
>> A(2,3) = 9;
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

If you type A, the new value of matrix A is,

```
>> A
```

```
A =

     1     2     3

     4     5     9
```

Changing the value of a matrix

```
>> AW=[3 6 9; 8 5 11]
AW =
     3     6     9
     8     5    11
>> AW(4,5)=17
AW =
     3     6     9     0     0
     8     5    11     0     0
     0     0     0     0     0
     0     0     0     0    17
>> BG(3,4)=15
BG =
     0     0     0     0
     0     0     0     0
     0     0     0    15
>>
```

Define a 2×3 matrix.

Assign a value to the (4,5) element.

MATLAB changes the matrix size to 4×5, and assigns zeros to the new elements.

Assign a value to the (3,4) element of a new matrix.

MATLAB creates a 3×4 matrix and assigns zeros to all the elements except BG(3,4).

Creating matrix from existing ones

```
>> B = A(1:2,[1 3])
B =
     1     3
     4     9
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

A colon is used to specify all the rows or all the columns of a matrix. For example,

```
>> A(1,:)
  ans =
       1   2   3
```

Creating matrix from existing ones

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

C = [A; B]?

Creating matrix from existing ones

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

A =

1	2	3
4	5	6

```
>> B = [7 8; 9 10];
```

B =

7	8
9	10

```
>> C = [A(1:2,1:2); B]
```

C =

1	2
4	5
7	8
9	10

Creating matrix from existing ones

```
>> v=4:3:34
v =
     4     7    10    13    16    19    22    25    28    31    34
>> u=v([3,  5,  7:10])
u =
    10    16    22    25    28    31
>> A=[10:-1:4; ones(1,7); 2:2:14; zeros(1,7)]
A =
    10     9     8     7     6     5     4
     1     1     1     1     1     1     1
     2     4     6     8    10    12    14
     0     0     0     0     0     0     0
>> B = A([1,3],[1,3,5:7])
```

Create a vector v with 11 elements.

Create a vector u from the 3rd, the 5th, and the 7th through 10th elements of v.

Create a 4×7 matrix A.

Create a matrix B from the 1st and 3rd rows, and 1st, 3rd, and the 5th through 7th columns of A.

Adding elements to a matrix

- Rows and/or columns can be added to an existing matrix by assigning values to the new rows or columns.
- This must be done carefully since the size of the added rows or columns must fit the existing matrix.

Adding elements to a matrix

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

Define a 2 × 4 matrix E.

```
>> E(3,:)=[10:4:22]

E =
     1     2     3     4
     5     6     7     8
    10    14    18    22
```

Add the vector 10 14 18 22 as the third row of E.

```
>> K=eye(3)
K =
     1     0     0
     0     1     0
     0     0     1
```

Define a 3 × 3 matrix K.

```
>> G=[E  K]
G =
     1     2     3     4     1     0     0
     5     6     7     8     0     1     0
    10    14    18    22     0     0     1
```

Append matrix K to matrix E. The numbers of rows in E and K must be the same.

Deleting elements from a matrix

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

```
mtr =
     5    78     4    24     9
     4     0    36    60    12
    56    13     5    89     3
>> mtr(:,2:4)=[]
mtr =
     5     9
     4    12
    56     3
>>
```

Eliminate all the rows of columns 2 through 4.

Some handy functions

Function	Description	Example

`diag(v)`	When `v` is a vector, creates a square matrix with the elements of `v` in the diagonal.	`>> v=[7 4 2];`
		`>> A=diag(v)`
		`A =`
		`7 0 0`
		`0 4 0`
		`0 0 2`

`diag(A)`	When `A` is a matrix, creates a vector from the diagonal elements of `A`.	`>> A=[1 2 3; 4 5 6; 7 8 9]`
		`A =`
		`1 2 3`
		`4 5 6`
		`7 8 9`
		`>> vec=diag(A)`
		`vec =`
		`1`
		`5`
		`9`

Some handy functions

`size(A)`	Returns a row vector `[m,n]`, where `m` and `n` are the size $m \times n$ of the array `A`.	`>> A=[6 1 4 0 12; 5 19 6 8 2]`
		`A =`
		`6 1 4 0 12`
		`5 19 6 8 2`
		`>> size(A)`
		`ans =`
		`2 5`

`reshape(A, m,n)`	Creates a `m` by `n` matrix from the elements of matrix `A`. The elements are taken column after column. Matrix `A` must have `m` times `n` elements.	`>> A=[5 1 6; 8 0 2]`
		`A =`
		`5 1 6`
		`8 0 2`
		`>> B = reshape(A,3,2)`
		`B =`
		`5 0`
		`8 6`
		`1 2`

Examples on manipulating matrixes

Using the ***ones*** and ***zeros*** commands, create a matrix in which the first two rows are 0s and the next two rows are 1s.

```
>> A(1:2,:)=zeros(2,5)
A =
     0     0     0     0     0
     0     0     0     0     0
>> A(3:4,:)=ones(2,5)
A =
     0     0     0     0     0
     0     0     0     0     0
     1     1     1     1     1
     1     1     1     1     1
```

First, create a 2×5 matrix with 0s.

Add rows 3 and 4 with 1s.

Examples on manipulating matrixes

Using the ***ones*** and ***zeros*** commands, create a matrix in which the first two rows are 0s and the next two rows are 1s.

```
>> A=[zeros(2,5);ones(2,5)]
A =
     0     0     0     0     0
     0     0     0     0     0
     1     1     1     1     1
     1     1     1     1     1
```

Create a 4×5 matrix from two 2×5 matrices.

Examples on manipulating matrixes

Create a 6×6 matrix in which the middle two rows and the middle two columns are 1s, and the rest of the entries are 0s.

```
>> AR=zeros(6,6)
AR =
     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     0     0     0     0     0
     0     0     0     0     0     0
```

First, create a 6 × 6 matrix with 0s.

```
>> AR(3:4,:)=ones(2,6)
AR =
     0     0     0     0     0     0
     0     0     0     0     0     0
     1     1     1     1     1     1
     1     1     1     1     1     1
     0     0     0     0     0     0
     0     0     0     0     0     0
```

Reassign the number 1 to the 3rd and 4th rows.

```
>> AR(:,3:4)=ones(6,2)
AR =
     0     0     1     1     0     0
     0     0     1     1     0     0
     1     1     1     1     1     1
     1     1     1     1     1     1
     0     0     1     1     0     0
     0     0     1     1     0     0
```

Reassign the number 1 to the 3rd and 4th columns.

Examples on manipulating matrixes

Given are a 5×6 matrix A, a 3×6 matrix B, and a 9-element vector v.

$$A = \begin{bmatrix} 2 & 5 & 8 & 11 & 14 & 17 \\ 3 & 6 & 9 & 12 & 15 & 18 \\ 4 & 7 & 10 & 13 & 16 & 19 \\ 5 & 8 & 11 & 14 & 17 & 20 \\ 6 & 9 & 12 & 15 & 18 & 21 \end{bmatrix}$$

$$B = \begin{bmatrix} 5 & 10 & 15 & 20 & 25 & 30 \\ 30 & 35 & 40 & 45 & 50 & 55 \\ 55 & 60 & 65 & 70 & 75 & 80 \end{bmatrix}$$

$$v = \begin{bmatrix} 99 & 98 & 97 & 96 & 95 & 94 & 93 & 92 & 91 \end{bmatrix}$$

Create the three arrays in the Command Window,
by writing **one command**:

1. replace the last four columns of the first and third rows of A with the first four columns of the first two rows of B,
2. the last four columns of the fourth row of A with the elements 5 through 8 of v,
3. the last four columns of the fifth row of A with columns 3 through 5 of the third row of B.

E

```
>> A=[2:3:17; 3:3:18; 4:3:19; 5:3:20; 6:3:21]
A =
     2     5     8    11    14    17
     3     6     9    12    15    18
     4     7    10    13    16    19
     5     8    11    14    17    20
     6     9    12    15    18    21
>> B=[5:5:30; 30:5:55; 55:5:80]
B =
     5    10    15    20    25    30
    30    35    40    45    50    55
    55    60    65    70    75    80
>> v=[99:-1:91]
v =
    99    98    97    96    95    94    93    92    91
>> A([1 3 4 5],3:6)=[B([1 2],1:4); v(5:8); B(3,2:5)]
```

4 × 4 matrix made of columns 3 through 6 of rows 1, 3, 4, and 5.

4 × 4 matrix. The first two rows are columns 1 through 4 of rows 1 and 2 of matrix B. The third row consists of elements 5 through 8 of vector v. The fourth row consists of columns 2 through 5 of row 3 of matrix B.