

Numerical Methods
MATLAB Environment
(Initiation)

MATLAB Desktop

Introduction

- MATLAB is a software package ,
- developed by MathWorks.
- It provides a high-level programming language
- and an interactive environment
- designed for numerical computation, visualization and programming.

Introduction

- MATLAB is widely used as a computational tool in science and engineering (encompassing the fields of physics, chemistry, math and all engineering streams).
- It is used in a range of applications including :
 - signal processing and communications
 - image and video Processing
 - control systems
 - test and measurement
 - computational finance
 - computational biology

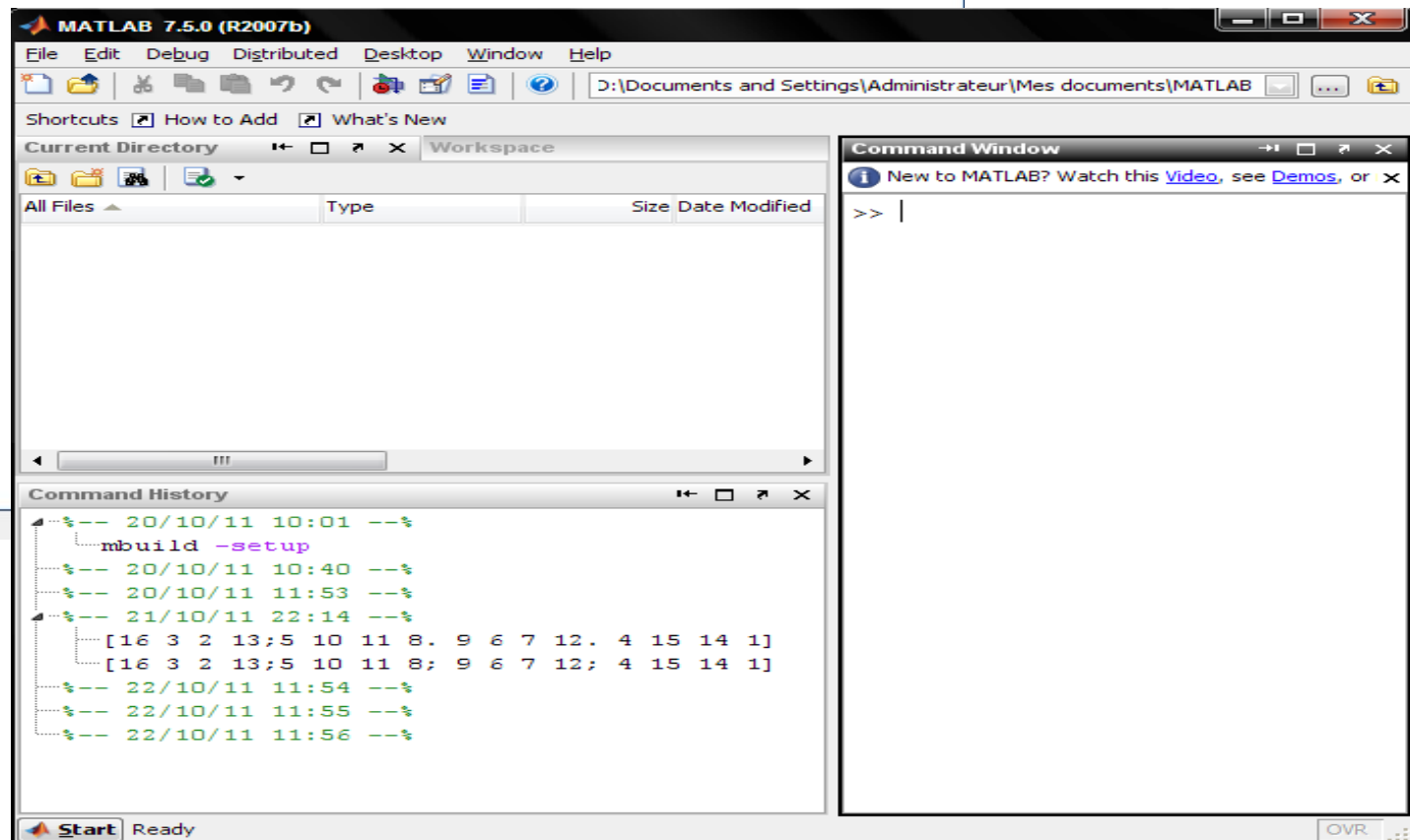
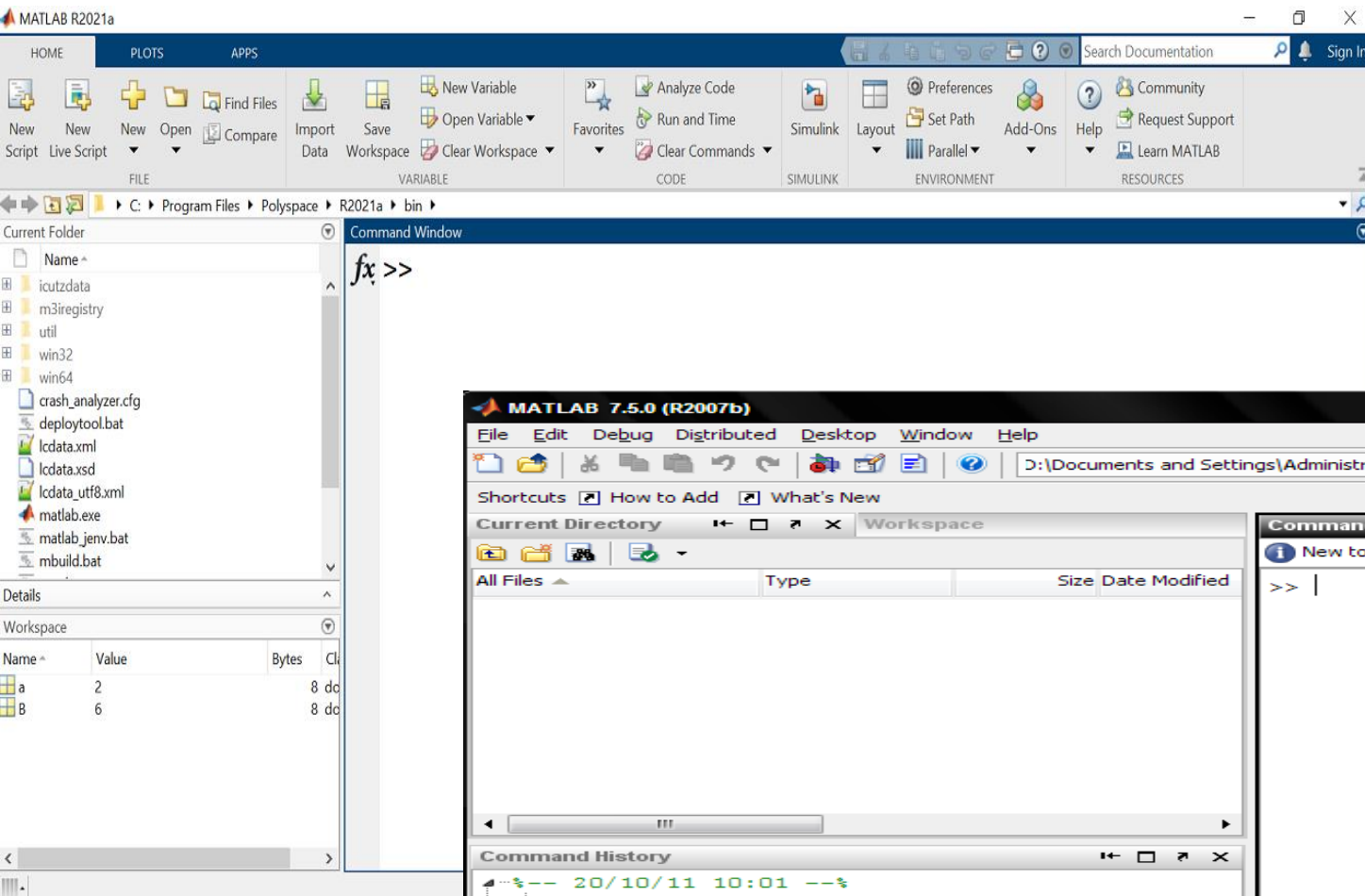
Introduction

- MATLAB name is derived from MATrix LABoratory allows:
- matrix manipulations;
- plotting of functions and data;
- implementation of algorithms;
- creation of user interfaces;
- interfacing with programs written in other languages, including C, C++, Java, and FORTRAN;
- analyzing data; developing algorithms; and creating models and applications (for AI).

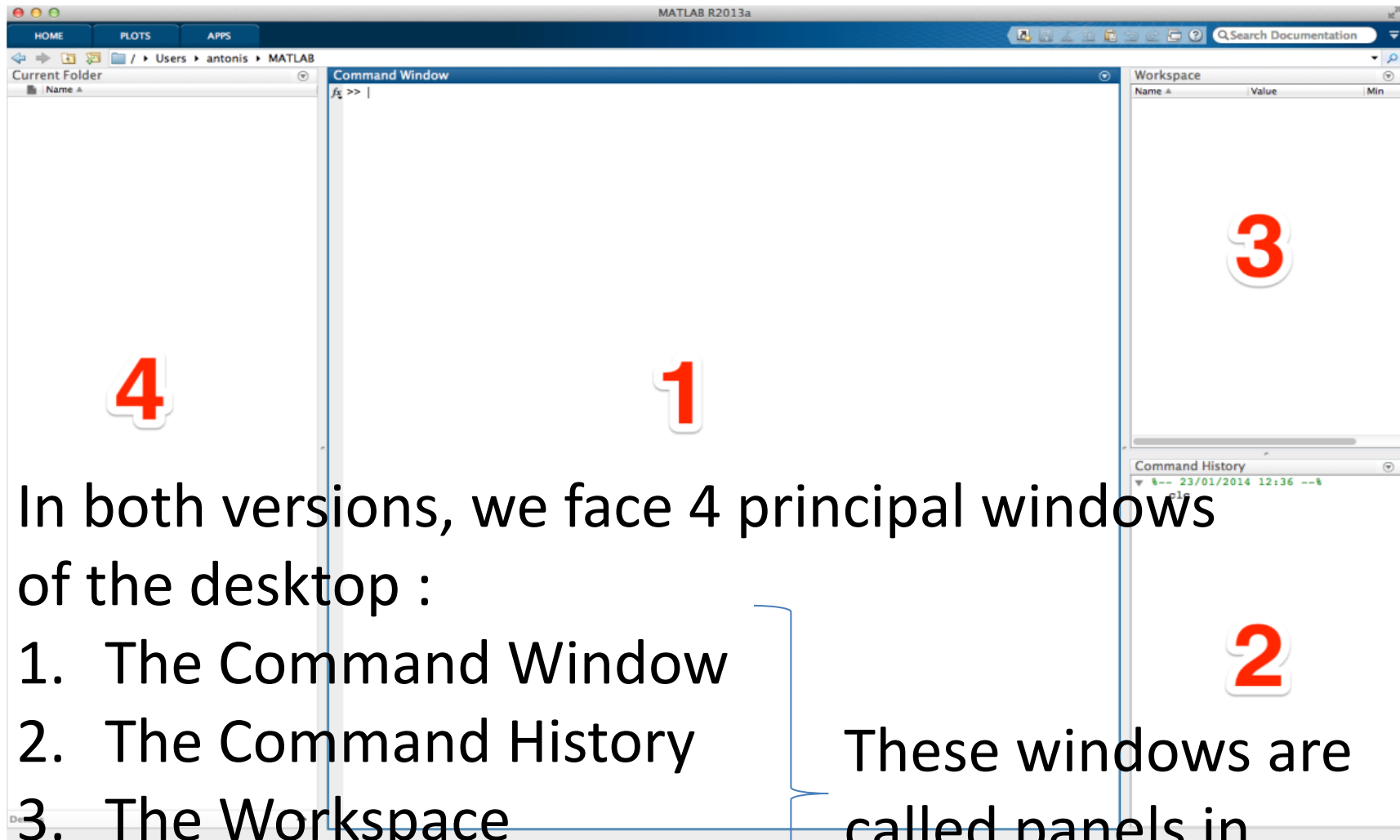
Principal Desktop components

- When MATLAB starts, a special screen appears :
it is called the **MATLAB desktop**.
- The desktop is a window that contains *other* windows.
- Working on a MATLAB session is equivalent to working on one of these windows or more.

Principal Desktop components



Principal Desktop components



In both versions, we face 4 principal windows of the desktop :

1. The Command Window
2. The Command History
3. The Workspace
4. The Current Directory

These windows are called panels in latest versions

Command Window

- This window (panel) is the main window in MATLAB
- It works by command line indicated by the command prompt (>>)
- It provides an interactive environment :
 - It is used to type lines of commands and expressions behind the command prompt.
 - Also, it displays MATLAB answers or commands result.

Command Window

→ MATLAB creates a default variable

`ans`

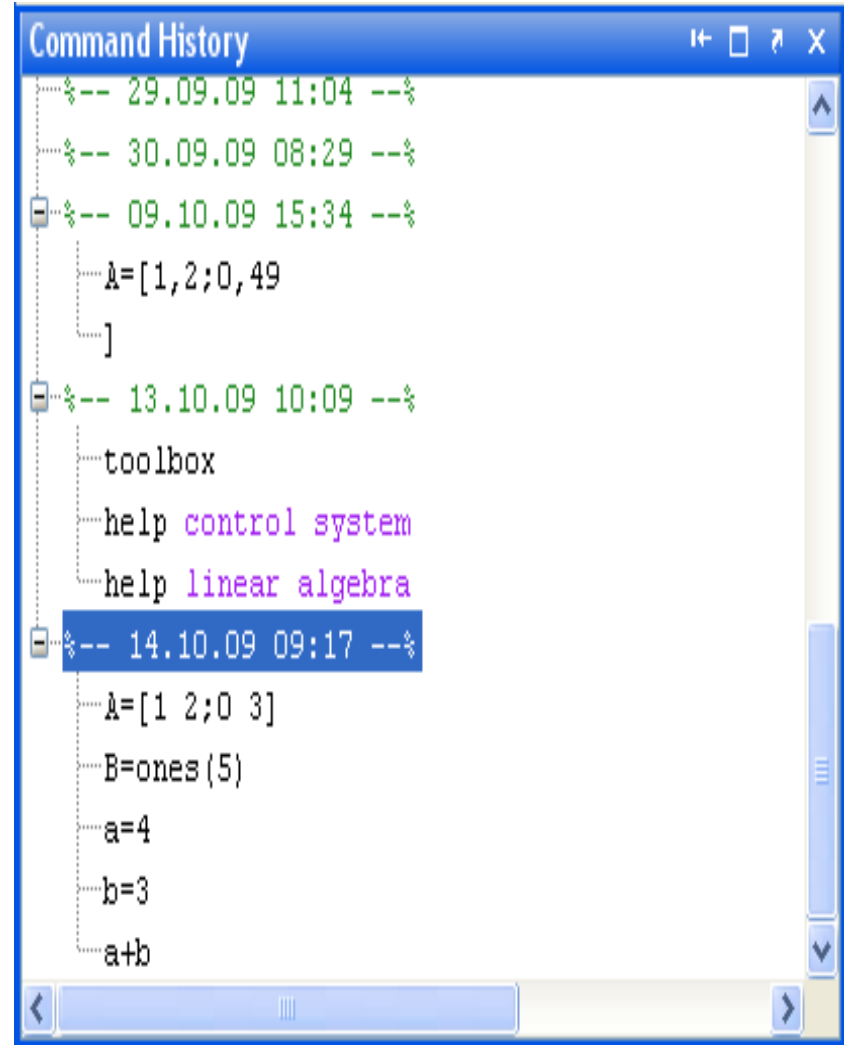
- (shortness of answer)
- `ans` stores the results of the current calculation.
- if the variable `ans` already exists from previous computations, it will be overwritten,
- To avoid this, you may assign the expression value to a variable or an output argument name.

Command Window

- In the assignment statements
`>> x = expression`, expression is a combination of numerical values, mathematical operators, variables, and function calls.
- Type the `clc` command to clean the command window.

Command History

- This window is directly related with command window :
it saves and shows a list of previously typed commands.

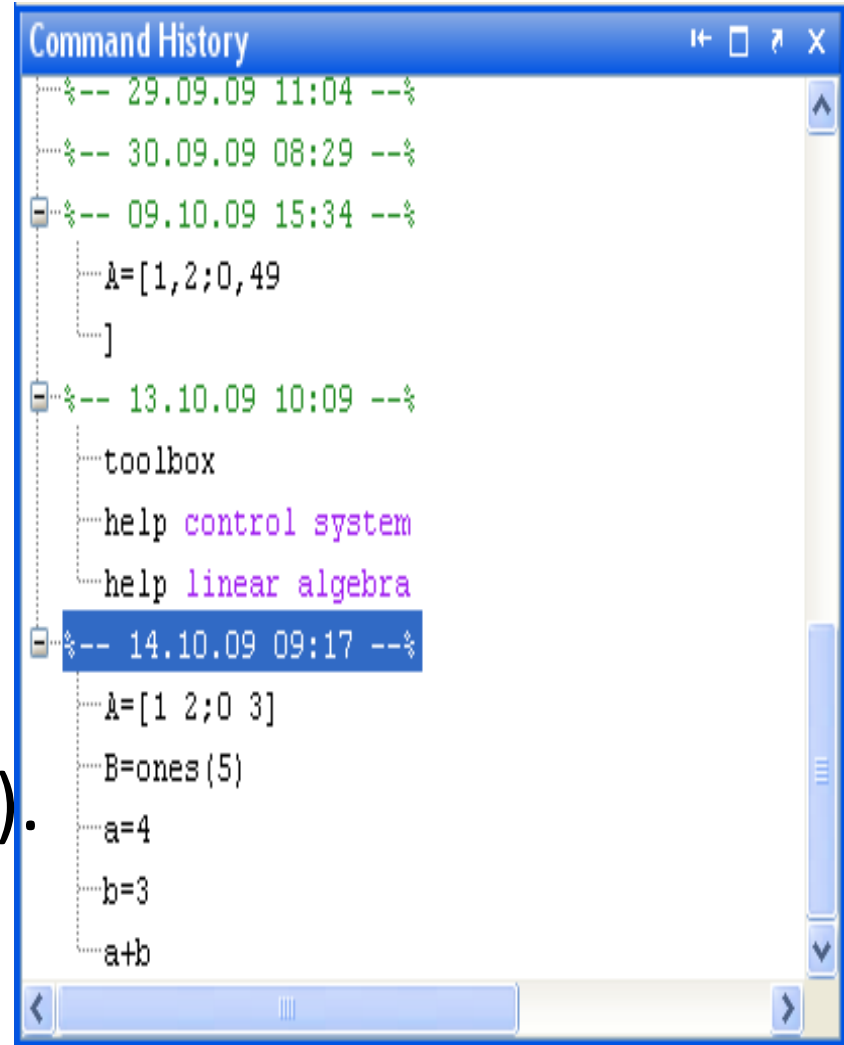


The screenshot shows the 'Command History' window in MATLAB. The window has a blue title bar with the text 'Command History' and standard window controls (minimize, maximize, close). The main area is a list of commands with their timestamps. The commands are grouped by date. The most recent command is highlighted with a blue background.

```
%-- 29.09.09 11:04 --%  
%-- 30.09.09 08:29 --%  
%-- 09.10.09 15:34 --%  
    A=[1,2;0,49  
    ]  
%-- 13.10.09 10:09 --%  
    toolbox  
    help control system  
    help linear algebra  
%-- 14.10.09 09:17 --%  
    A=[1 2;0 3]  
    B=ones(5)  
    a=4  
    b=3  
    a+b
```

Command History

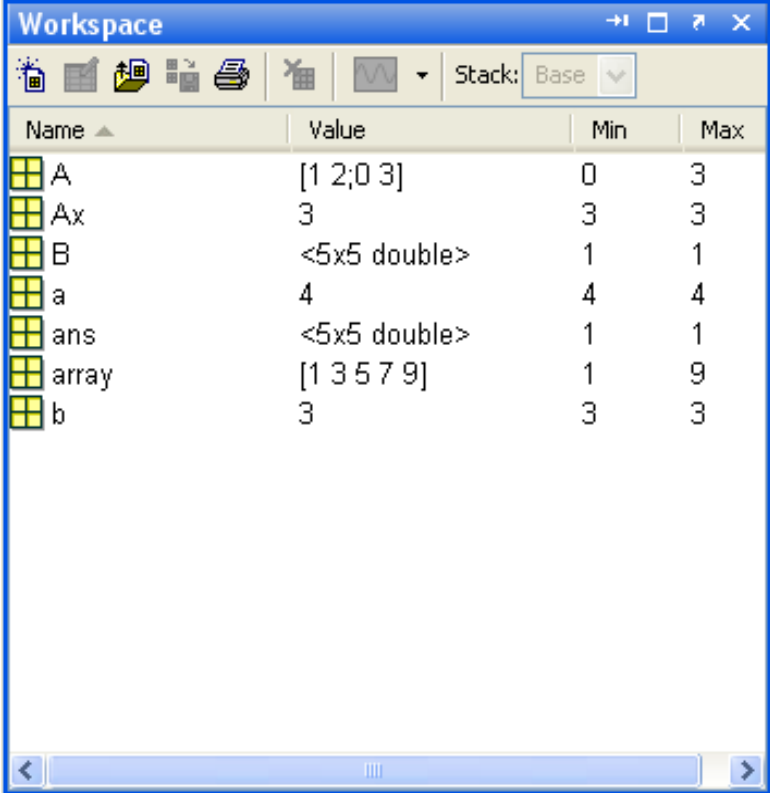
- The command history persists across multiple sessions :
each command typed during a session is saved under a title indicating the date and starting time of the session (the green line).

A screenshot of the MATLAB Command History window. The window has a blue title bar with the text "Command History" and standard window controls. The main area is white with a vertical scrollbar on the right. It displays a list of command sessions, each starting with a green line indicating the date and time. The sessions are separated by small square icons. The most recent session is highlighted with a blue background. The commands in this session are: `A=[1,2;0,49]`, `]`, `toolbox`, `help control system`, `help linear algebra`, `A=[1 2;0 3]`, `B=ones(5)`, `a=4`, `b=3`, and `a+b`.

```
Command History
-- 29.09.09 11:04 --
-- 30.09.09 08:29 --
-- 09.10.09 15:34 --
    A=[1,2;0,49
    ]
-- 13.10.09 10:09 --
    toolbox
    help control system
    help linear algebra
-- 14.10.09 09:17 --
    A=[1 2;0 3]
    B=ones(5)
    a=4
    b=3
    a+b
```

Workspace

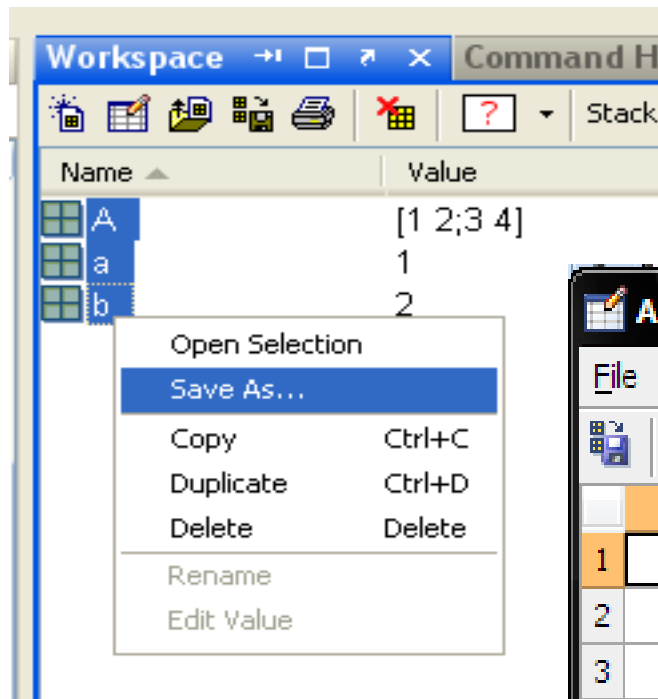
- Lists all the variables generated in the current session i.e : variables used since MATLAB have been opened last time.
- It shows the value, type and size of variables,
- it can be used to quickly plot, or inspect the values of these indicated variables.



The screenshot shows the MATLAB Workspace window. It has a title bar 'Workspace' and a toolbar with icons for saving, deleting, and other actions. Below the toolbar is a 'Stack' dropdown menu set to 'Base'. The main area is a table with four columns: 'Name', 'Value', 'Min', and 'Max'. The table lists several variables: A, Ax, B, a, ans, array, and b. Each variable has a small grid icon to its left. The 'Value' column shows the current value of each variable, such as '[1 2; 0 3]' for A and '3' for Ax. The 'Min' and 'Max' columns show the minimum and maximum values of the variables, respectively.

Name	Value	Min	Max
A	[1 2; 0 3]	0	3
Ax	3	3	3
B	<5x5 double>	1	1
a	4	4	4
ans	<5x5 double>	1	1
array	[1 3 5 7 9]	1	9
b	3	3	3

Workspace



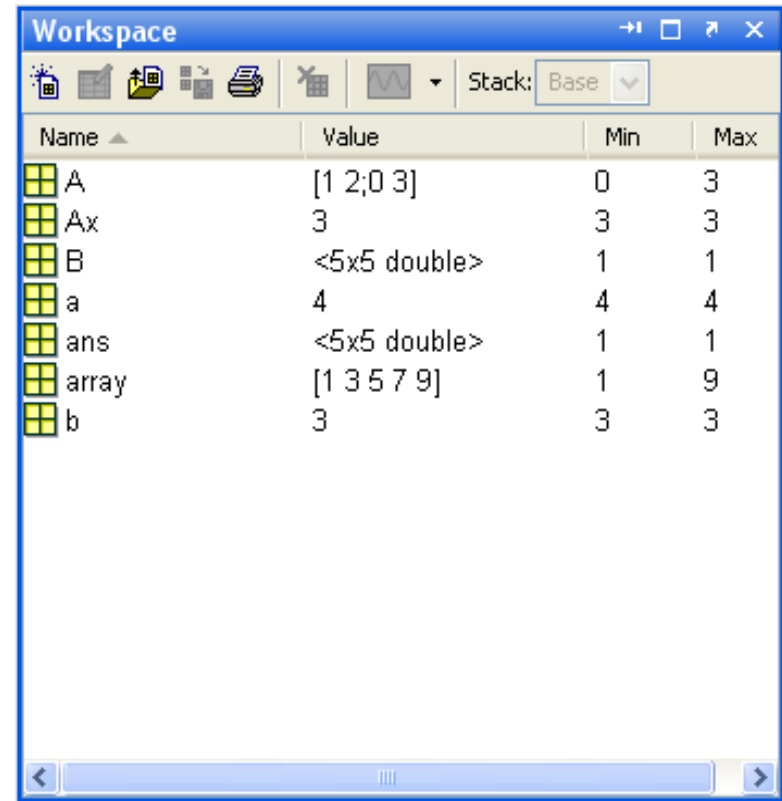
Variables values and content can be shown and modified using another window named Array Editor

The Array Editor - runout window displays a 10x6 matrix of numerical data. The first column is indexed 1 to 10, and the first four columns are indexed 1 to 4. The data is as follows:

	1	2	3	4	5	6
1	-0.4600	-0.2400	-0.0400	-0.3100		
2	-0.4600	-0.3100	-0.2500	-0.1500		
3	-0.1400	-0.3400	-0.0100	-0.0800		
4	-0.1400	-0.1700	-0.1200	0.0800		
5	0.0900	-0.1300	0.0600	-0.0600		
6	-0.2200	-0.0300	0.0200	0.0500		
7	-0.1400	-0.1900	0.0200	0.1500		
8	-0.1400	-0.0300	-0.0300	0.1700		
9	-0.0600	-0.0700	-0.1100	0.2300		
10	-0.1800	-0.2300	-0.1100	-0.2300		

Workspace

- As seen before in command window, MATLAB variables are created by an assignment statement,
- However, MATLAB allows variable creation by importing them from files :
- In fact, one may save some/all workspace variables to a text file (**.mat** file), using icons on workspace bar

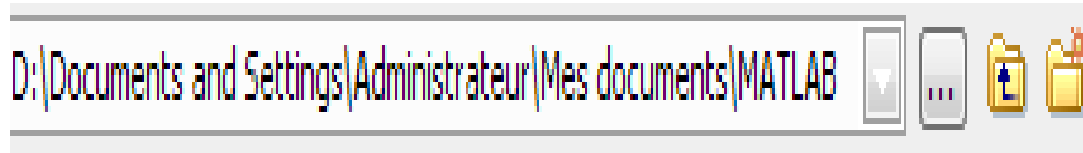
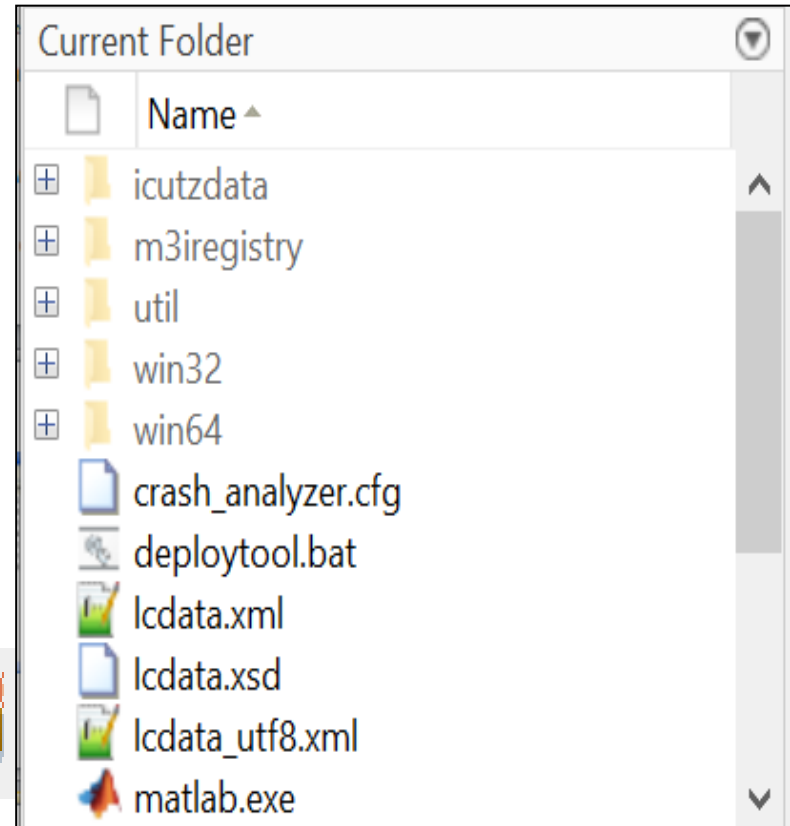


Current Folder

- This window → shows the files and directories available in the current folder.

But, What is current folder?

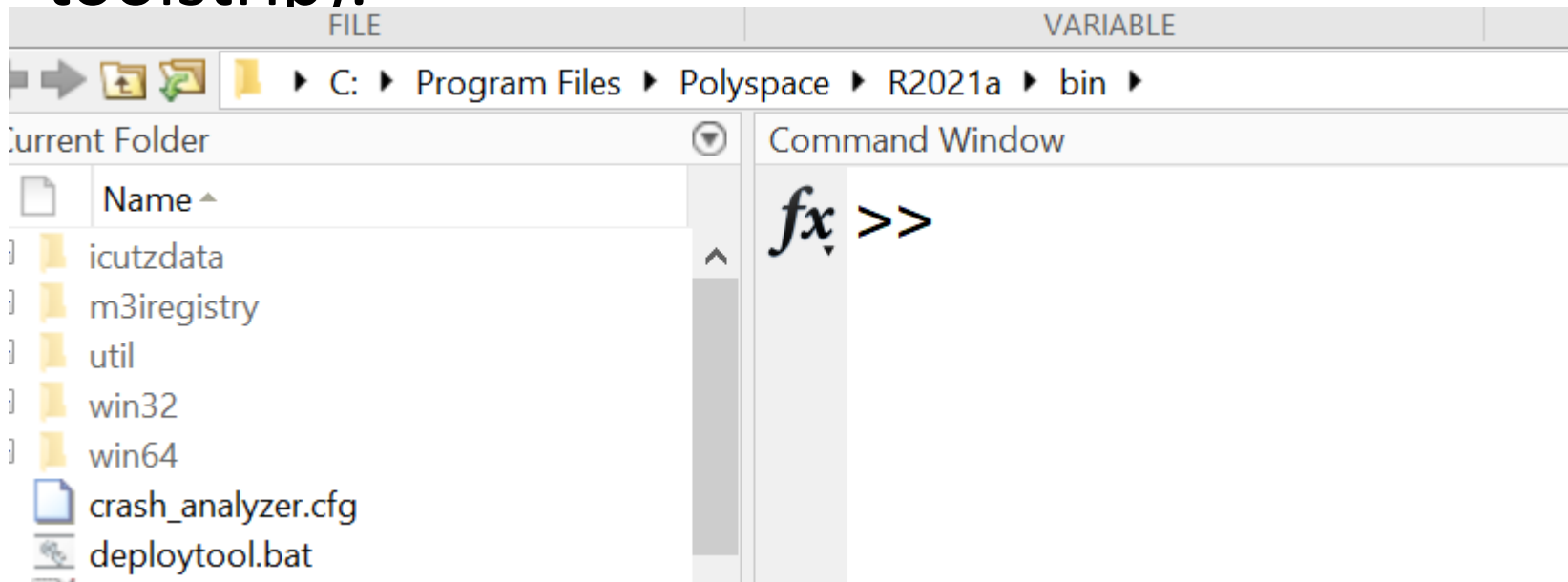
- By default, it is a directory created by MATLAB in your documents folder,



- and this is where you should save your work (saved as files!).

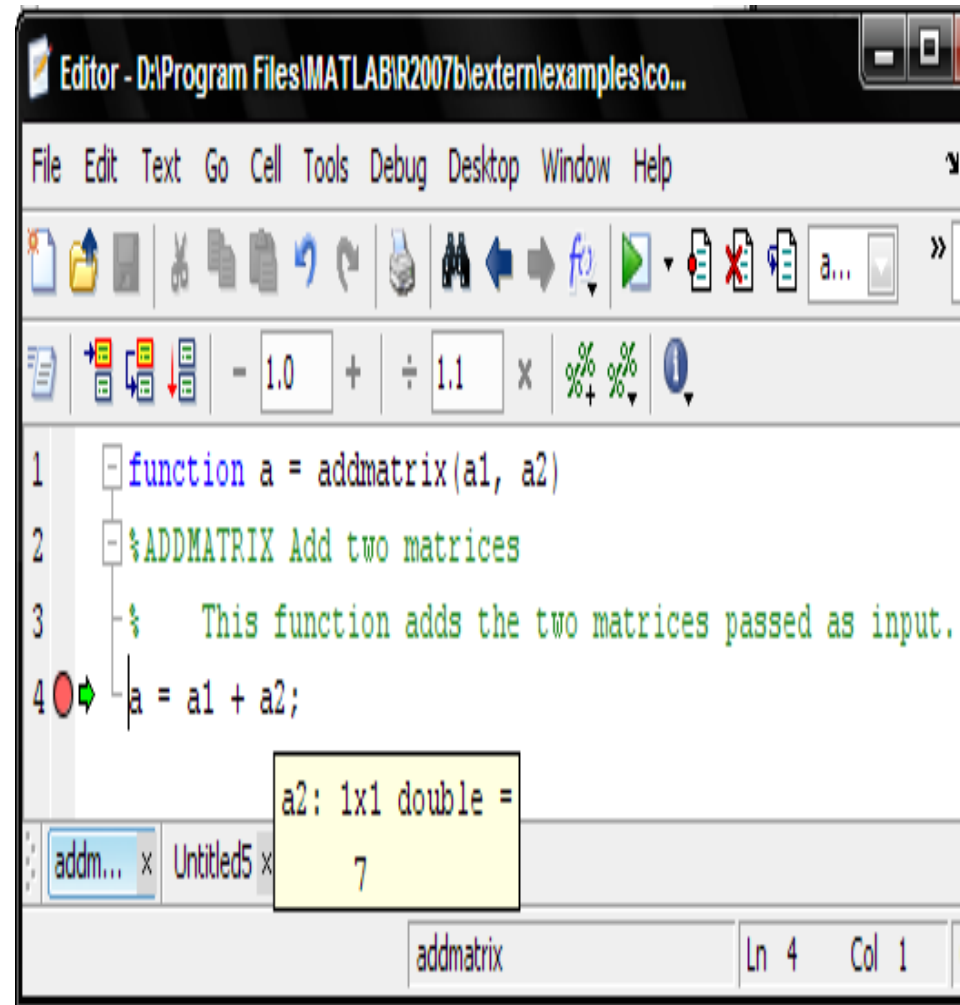
Current Folder

- The path to the current directory is listed/shown on the top of the MATLAB desktop. (on the address bar or in the toolstrip).



Editor

- The **Editor** is used to create scripts and m-files.
- Click the “New Script” icon/button in the Toolbar



Illustrations above show the default configuration of the MATLAB desktop.

We can customize the arrangement of the windows to suit our needs or preferences.

We can return back to the default layout by following :

Desktop menu →

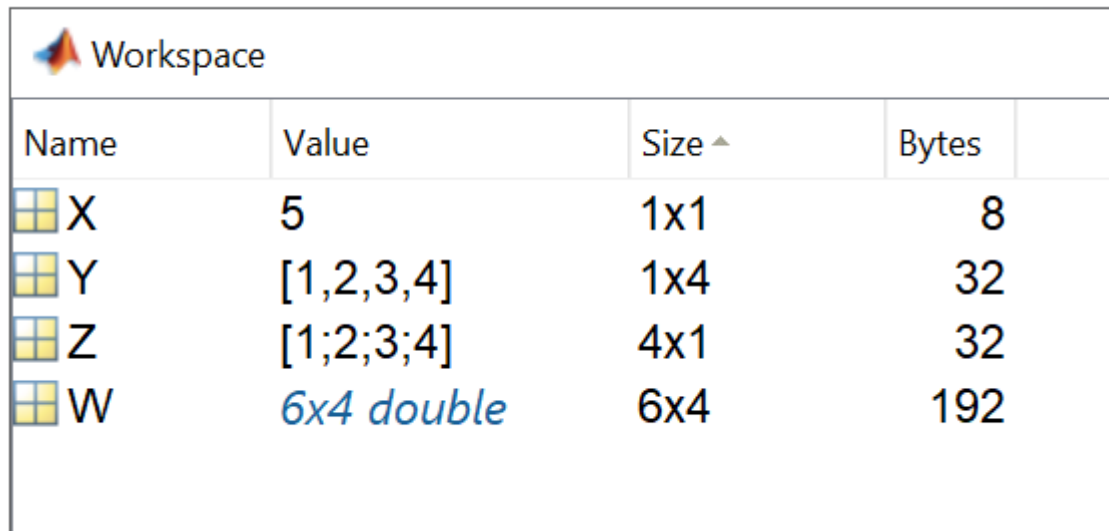
desktop layout →

default

Variables in MATLAB

1st principal

All MATLAB variables are multidimensional *arrays*, no matter what type of data.



The image shows a screenshot of the MATLAB Workspace window. It contains a table with four columns: Name, Value, Size, and Bytes. There are four rows of variables: X, Y, Z, and W. Variable X has a value of 5 and a size of 1x1. Variable Y has a value of [1,2,3,4] and a size of 1x4. Variable Z has a value of [1;2;3;4] and a size of 4x1. Variable W has a value of 6x4 double and a size of 6x4.

Name	Value	Size ▲	Bytes
X	5	1x1	8
Y	[1,2,3,4]	1x4	32
Z	[1;2;3;4]	4x1	32
W	6x4 double	6x4	192

two-dimensional arrays

Array Creation

row vector

- **By components listing**

```
>>a= [1 2 3 4]
```

```
>>a= [ 1 ,2 ,3 ,4]
```

```
a = 1 2 3 4
```

Array Creation

row vector

- By components description

```
>>a= 1:4
```

```
      a = 1 2 3 4
```

```
>>a= 1:2:8
```

```
      a = 1 3 5 7
```

```
>>a=1: -1: 9
```

```
      a=[] %empty
```

Array Creation

row vector

- **MATLAB functions**

linspace, logspace, rand, ones, zeros,

>> linspace(initial, final, cpt)

>> linspace(2, 9, 5)

ans =

2.0000 3.7500 5.5000 7.2500 9.0000

>> logspace(2, 9, 5)

ans =

100.00 5623.41 316227.77 17782794.10
1000000000.00

Array Creation

row vector

- **Other methods :**

- **concatenation**

- >>a=[1 2 3], b=[5 6 7],

- >> c=[a, b]

- c=

- 1 2 3 5 6 7

- **Import data**

- MAT files : >>Load or File menu

Array Creation

column vector

- By components listing

```
>>a= [ 1 ; 2 ; 3 ; 4]
```

```
a=
    1
    2
    3
    4
```

- By components description

```
>> a=(1:4)'
```

```
>>a= ( 1:2:8)'      1
```

```
                a = 3
```

```
>>a=(1: -1: 9)'      5
```

```
                a = 7
a=[] %empty
```

Array Creation

column vector

- **MATLAB functions**

```
>> linspace(2, 9, 5)'
```

```
ans =
```

```
2.0000
```

```
3.7500
```

```
5.5000
```

```
7.2500
```

```
9.0000
```

- **concatenation**

```
>> a=(1:4)'
```

```
>>b= ( 1:2:8)'
```

```
>>C=[a ; b]
```

- **Import data**

1

2

3

4

1

3

5

7

Array Creation matrix

- **By components listing**

```
>>a= [ 1 2 3 ; 4 5 6 ; 7 8 9]
```

```
a=
```

```
    1    2    3  
    4    5    6  
    7    8    9
```

- **By components description**

```
a= [1 : 3 ; 4 : 6 ; 7 : 9]
```

```
b=[(1:3:7)', (2:3:8)', (3:3:9)']
```

Array Creation

matrix

- **MATLAB functions**

>> function_name(rows? , columns?)

>> ones(3,5) >>rand(2,4)

>> zeros(3,5) >>randi([-3,2], 2,5)

Square matrices

>>ones(3)

>>zeros(4)

>>rand(2)

Can we create vectors using these functions?

Array Creation matrix

- **Concatenation**

convenient arrays size

```
>>C=[Z B; A D F]
```

```
>>cat(2,A,F)% [A,F].
```

```
>>cat(1,D,Z)% [D;Z].
```

variable	size
A	2X1
D	2X2
F	2X4
Z	3X2
B	3X5
C	?
ans	?
ans	?

Array Creation practice

- Create a vector of odd values from 1 to 10
- Create a matrix of 4 rows by 4 columns of value 3
- Create a matrix of 2x5 random integer values ranged from 1 to 6

Array Creation practice

```
>>A=1 : 2 : 10
```

```
>>M=3*ones(4)
```

```
>>Z=randi([1,6],2,5)
```

Predifined size functions

	A=[1:4;linspace(1,8,4)]	A=ones(1,3)	
>>numel(A)	ans = 8	ans = 3	
>>length (A)	ans = 4	ans = 3	
>> R=size(A)	R = 2 4	R = 1 3	
>>[n,m]=size(A)	n = 2 m = 4	n = 1	m = 3
>>size(A,1)	ans = 2	ans = 1	
>>size(A,2)	ans = 4	ans = 3	
>>ndims(A)	ans = 2	ans = 2	

Elements extraction/indexing

- variable_name(linear index)
- Variable_name(row,column)

~~A(0)~~	A=[1:4;linspace(1,8,4)] A = 1.00 2.00 3.00 4.00 1.00 3.33 5.66 8.00	A=ones(1,3) A=1 1 1
>>A(3)	ans = 2.00	ans = 1
>>A(2,3)	ans = 5.66	error
>>A(end)	ans = 8	ans= 1

Elements extraction /indexing

	A=[1:4;linspace(1,8,4)] A = 1.00 2.00 3.00 4.00 1.00 3.33 5.66 8.00		A=ones(1,3) A=1 1 1
>>A(:)	ans = 1.00 1.00 2.00	3.33 3.00 5.66 4.00 8.00	ans = 1 1 1
>>A(:,3)	ans = 3.00 5.66		1
>>A(1, [2 4]) >>A(1,2:2:4)	ans = 2.00 4.00		error
>> A(1:3)	ans = 1 1 3		1 1 1

Elements extraction

Command Window

Logical search

```
>> A=magic(4)
```

```
A =
```

16.00	2.00	3.00	13.00
5.00	11.00	10.00	8.00
9.00	7.00	6.00	12.00
4.00	14.00	15.00	1.00

```
>> find(A>5)
```

```
ans =
```

1.00
3.00
6.00
7.00
8.00
10.00
11.00
12.00
13.00
14.00
15.00

```
>> A(find(A>5))
```

```
ans =
```

16.00
9.00
11.00
7.00
14.00
10.00
6.00
15.00
13.00
8.00
12.00

fx >>

Elements extraction/ indexing

~~A(0)~~	A=[1:4;linspace(1,8,4)] A = 1.00 2.00 3.00 4.00 1.00 3.33 5.66 8.00	A=ones(1,3) A=1 1 1
>>A(3)=-5	A = 1.00 -5 3.00 4.00 1.00 3.33 5.66 8.00	A=1 1 -5
>>A(1,3)=-3	A = 1.00 -5 -3 4.00 1.00 3.33 5.66 8.00	A=1 1 -3
A(end,end)=0	A = 1.00 -5 -3 4.00 1.00 3.33 5.66 0	A=1 1 0

Elements extraction/ indexing

	A=[1:4;linspace(1,8,4)] A = 1.00 2.00 3.00 4.00 1.00 3.33 5.66 8.00	A=ones(1,3) A=1 1 1
A(:)=5	A= 5 5 5 5 5 5 5 5	A= 5 5 5
A(:,3)=0	A = 1.00 2.00 0 4.00 1.00 3.33 0 8.00	0 0 0
A(1, 6)	error	error
A(1,6)=-1	1.00 2.00 0 4.00 0 -1.00 1.00 3.33 0 8.00 0 0	1 1 1 0 0 1

Delete operator

A =

```
>>A=[]
```

1.00	2.00	3.00	4.00
------	------	------	------

```
>>A(row, column)=[]
```

1.00	3.33	5.66	8.00
------	------	------	------

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

A =

1.00	2.00	3.33	3.00	5.66	4.00	8.00
------	------	------	------	------	------	------

```
>> A(2,3)=[] → error
```

```
>>A(:, 2) =[]→
```

A =

1.0000	3.0000	4.0000
--------	--------	--------

1.0000	5.6667	8.0000
--------	--------	--------

Elementwise operations

```
a=1:4  
b  
b=10:10:40
```

→

identical size of a and

```
>> a+b  
ans =  
11.00 22.00 33.00 44.00  
>> a-b  
ans =  
-9.00 -18.00 -27.00 -36.00  
>> a.*b  
ans =  
10.00 40.00 90.00 160.00  
>> a./b  
ans =  
0.10 0.10 0.10 0.10
```

→

```
>> plus(a,b)
```

```
>> b.^a
```

```
ans =
```

```
10 400 27000 256
```

→

```
>> times(a,b)
```

→

```
>> rdivide(a,b)
```

```
>> A
A =
     1     2     3
     4     5     6
>> 2+A
ans =
     3     4     5
     6     7     8
>> 2.*A
ans =
     2     4     6
     8    10    12
>> 2*A
ans =
     2     4     6
     8    10    12
>> 2./A
ans =
    2.0000    1.0000    0.6667
    0.5000    0.4000    0.3333
>> A./2
ans =
    0.5000    1.0000    1.5000
    2.0000    2.5000    3.0000
```

Scalar/array operations

```
>> A/2
ans =
    0.5000    1.0000    1.5000
    2.0000    2.5000    3.0000
>> 2/A
Error using /
Matrix dimensions must agree.
>> A*2
ans =
     2     4     6
     8    10    12
>> 2*A
ans =
     2     4     6
     8    10    12
>>
```

Functions /Operators

Relational operators.

<u>eq</u>	- Equal	==
<u>ne</u>	- Not equal	~=
<u>lt</u>	- Less than	<
<u>gt</u>	- Greater than	>
<u>le</u>	- Less than or equal	<=
<u>ge</u>	- Greater than or equal	>=

Logical operators.

<u>and</u>	- Element-wise logical AND	&
<u>or</u>	- Element-wise logical OR	
<u>not</u>	- Logical NOT	~
<u>xor</u>	- Logical EXCLUSIVE OR	
<u>any</u>	- True if any element of vector is nonzero	
<u>all</u>	- True if all elements of vector are nonzero	

Functions /Operators

```
>> 5<3
```

```
ans =
```

```
logical
```

```
0
```

```
>> 5>-1
```

```
ans =
```

```
logical
```

```
1
```

```
1:3==[1 2 4]
```

```
ans =
```

```
1×3 logical array
```

```
1 1 0
```

```
>> E= 3 & 7
```

```
E =
```

```
logical
```

```
1
```

```
>>R= 0 | 0
```

```
R =
```

```
logical
```

```
0
```

```
>> xor([1 1 0 0],[1 0 1 0])
```

```
ans =
```

```
1×4 logical array
```

```
0 1 1 0
```

Workspace				
Name ▲	Value	Size	Bytes	
☒ ans	<i>1x3 logical</i>	1x3	3	▼
☒ D	1	1x1	1	
☒ E	1	1x1	1	

Numeric Types

MATLAB represents floating-point numbers in either double-precision or single-precision format. The default is double precision.

name	byte	min	max
Double	8	-1.79769e+308	1.79769e+308
Single	4	-3.4 x 10 ³⁸	3.4 x 10 ³⁸

Numeric Types

name	byte	Min	max
uint8	1	0	2^8-1
uint16	2	0	$2^{16}-1$
uint32	4	0	$2^{32}-1$
uint64	8	0	$2^{64}-1$
int8	1	-2^7	2^7-1
int16	2	-2^{15}	$2^{15}-1$
int32	4	-2^{31}	$2^{31}-1$
int64	8	-2^{63}	$2^{63}-1$

Elementary Functions

Elementary Functions

Complex functions

- $z = \text{complex}(a,b)$ Construct complex output data, z , of real and imaginary parts, using two real inputs,
- The complex function provides a useful substitute for expressions, such as $a + 1i*b$ or $a + 1j*b$, when :
 - b is all zeros
 - i and j are already used
- Examples
 - $\gg z = \text{complex}(3)$
 $Z = 3.0000 + 0.0000i$
 - $\gg i=2, \quad Z=3+i, \quad Z=\text{complex}(3,1)$

Elementary Functions

Complex functions

```
>>Z= complex(3,5)
```

- `abs (Z)` - Absolute value and complex magnitude
- `angle (Z)` - Phase angle.
- `conj(Z)` - Complex conjugate.
- `imag (Z)` - Complex imaginary part.
- `real(Z)` - Complex real part.

```
ans =  
5.8310
```

```
ans =  
1.0304
```

```
ans =  
3.0000 - 5.0000i
```

```
ans =  
5
```

```
ans =  
3
```

Elementary Functions

Rounding and remainder functions

- **$r = \text{rem}(a,b)$** returns the remainder after division of a by b , where a is the dividend and b is the divisor.

This function is often called the remainder operation, which can be expressed as $r = a - b \cdot \text{fix}(a./b)$.

$$r = \text{rem}(27,5) \rightarrow r=2$$

- **$w = \text{mod}(a,m)$** returns the remainder after division of a by m , where a is the dividend and m is the divisor.

This function is often called the modulo operation, which can be expressed as $w = a - m \cdot \text{floor}(a./m)$.

$$w = \text{mod}(27,-5) \rightarrow w = -3$$

$$W=\text{mod}(-27,5) \rightarrow W = 3$$

Elementary Functions

Statistics (sum)

```
>>X=0:10:100;
```

```
>>sum(X)
```

```
>>X=(0:10:100)';
```

```
>>sum(X)
```

```
ans=
```

```
550
```

```
>>M=rand(3,5)
```

0.14	0.79	0.04	0.68	0.39
0.42	0.96	0.85	0.76	0.66
0.92	0.66	0.93	0.74	0.17

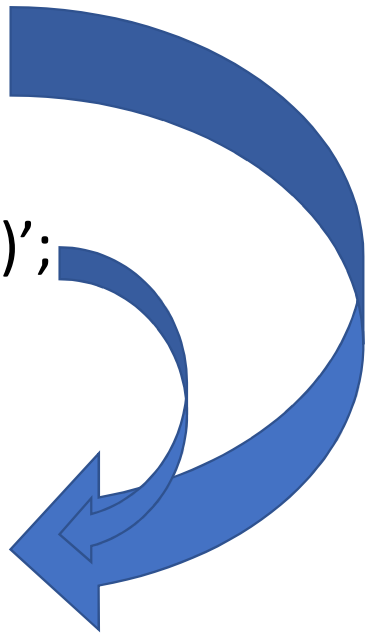
```
>>sum(M)
```

1.48	2.41	1.82	2.18	1.22
------	------	------	------	------

```
>>sum(M,1)
```

```
>>sum(M,2)
```

2.04
3.64
3.42



Elementary Functions

Statistics

- sum (M)
- sum (M,1)
- sum(M,2)
- prod (X)
- prod (X,1)
- prod(X,2)

- mean(X)
- mean (X,1)
- mean(X,2)

M=rand(3,5)

0.14	0.79	0.04	0.68	0.39
0.42	0.96	0.85	0.76	0.66
0.92	0.66	0.93	0.74	0.17

- sum(sum(M))
- all(prod(prod(M))==prod(M(:)))
- mean(mean(M))==mean(M(:))
- ans =1 (logical)

Elementary Functions

Statistics

X=randi(10,1,5)%X=rand(5,1)

X = 8.00 1.00 3.00 1.00 1.00

M=randi(10,5,5)

M = 9.00 5.00 5.00 3.00 5.00
7.00 4.00 5.00 7.00 10.00
4.00 8.00 7.00 7.00 4.00
10.00 8.00 8.00 2.00 6.00
1.00 2.00 8.00 2.00 3.00

Same for max

min (X)	ans= 1				
min(M) %by column	ans = 1.00	2.00	5.00	2.00	3.00
~~min(M,1)~~ ~~min(M,2)~~					
[val,pos]=min(M)					
val =					
1.00 2.00 5.00 2.00 3.00					
pos =					
5.00 5.00 1.00 4.00 5.00					
Pos = the row index that corresponds to the minimum value of M on each column. (according to the operating dimension)					

Programming language
of MATLAB

Introduction

- Like other computer programming languages, MATLAB has its own decision making structures for control of command execution.
- These decision making structures (often called *control flow* structures) include the following constructions :
 - for loops,
 - while loops,
 - if-else-end statements
 - Switch – case statements
- Control flow structures are often used in script M-files and function M-files.

if /else/ elseif

IF (MATLAB syntax)

```
if cond  
    commands  
end
```

IF /ELSE (MATLAB syntax)

```
if cond  
    commands1  
else  
    commands2  
end
```

ELSEIF (MATLAB syntax)

```
if cond1  
    commands1  
elseif cond2  
    commands2  
else  
    commands3  
end
```

Conditional statement:
evaluates to true or false

- No need for parentheses: command blocks are between reserved words
- elseif has no space between else and if (one word)
- no semicolon (;) is needed at the end of lines containing if, else, end
- indentation of if blocks is not required, but facilitate the reading.
- the end statement is required

if /else/ elseif (Example)

```
IF
discr = b*b - 4*a*c;
if discr < 0
    disp('Warning: discriminant is negative, roots are imaginary');
end
```

```
IF /ELSE
discr = b*b - 4*a*c;
if discr < 0
    disp('Warning: discriminant is negative, roots are imaginary');
else
    disp('Roots are real, but may be repeated')
end
```

if /else/ elseif (Example)

ELSEIF

```
discr = b*b - 4*a*c;  
if discr < 0  
    disp('Warning: discriminant is negative, roots are imaginary');  
elseif discr == 0  
    disp('Discriminant is zero, roots are repeated')  
else  
    disp('Roots are real')  
end
```

Switch-case statement

%variable on workspace

```
switch variable
    case value1
        commands
    case value2
        commands
    case value3
        commands3
    otherwise %optional
        commands4
end
```

- The switch statement executes groups of statements based on the value of a variable or expression.
- The keywords case and otherwise delineate the groups.
- Only the first matching case is executed:
Unlike the C language switch statement, MATLAB switch does not fall through. If the first case statement is true, the other case statements do not execute. So, break statements are not required
- The otherwise block is optional and is executed if none of the case values match the value of variable.
- There must always be an end to match the switch.

Switch-case statement (example)

use switch statement to
determine the number of
days in a given month.

```
month =input("which month?")
switch month
    case {1, 3, 5, 7, 8, 10, 12}
        disp("31 days");
    case {4, 6, 9, 11}
        disp("30 days");
    case 2
        disp("28 or 29 days");
    otherwise
        disp("Invalid month");
end
```

For loops

FOR (MATLAB syntax)

```
for var= initial : increment : final  
    commands  
end
```

1. The loop variable
 - Is defined as a vector var
 - Is a scalar within the command block
2. The command block
 - as needed commands and comments between the **for** line and the **end**

note: indent the loops for readability, especially when they are nested.

For loops (examples)

- Example 1:

```
for ii=1:5
    x=ii*ii           % same as (1:5).^2
end
```

- Example 2:

```
n = 5;
for j=2:n
    for i=1:j-1
        A(i,j)=i/j;
        A(j,i)=i/j;
    end
end
```

A=

0	0.50	0.33	0.25	0.20
0.50	0	0.67	0.50	0.40
0.33	0.67	0	0.75	0.60
0.25	0.50	0.75	0	0.80
0.20	0.40	0.60	0.80	0

isequal(A,A')
ans= 1

While loops

```
while (MATLAB syntax)  
while cond  
commands  
end
```

- The command block will execute while the conditional expression is true
 - You can use **break** to exit a loop

Preallocation

Consider this block which creates a vector a , of 100 elements, element by element.

```
» for n=1:100
»     res = % Very complex calculation %
»     a(n) = res;
» end

» a = zeros(1, 100);
» for n=1:100
»     res = % Very complex calculation %
»     a(n) = res;
» end
```

- Variable **a** is only assigned new values. No new memory is allocated

Vectorization

- Vectorized code is more efficient for MATLAB (LAB session N°1)
- Use indexing and matrix operations to avoid loops
- example: to add every two consecutive terms of a vector:

```
» a=rand(1,100);
```

```
» b=zeros(1,100);
```

```
» for n=1:100
```

```
»     if n==1
```

```
»         b(n)=a(n);
```

```
»     else
```

```
»         b(n)=a(n-1)+a(n);
```

```
»     end
```

```
» end
```

```
» a=rand(1,100);
```

```
» b=[0 a(1:end-1)]+a;
```

Programs in interactive mode

 Command Window

```
>> a=rand(1,100);  
>> b=rand(1,100);  
>> for i=1:100  
if n==1,b(n)=a(n);  
else b(n)=a(n-1)+a(n);end  
end  
fx >>
```

Programs using MATLAB Editor

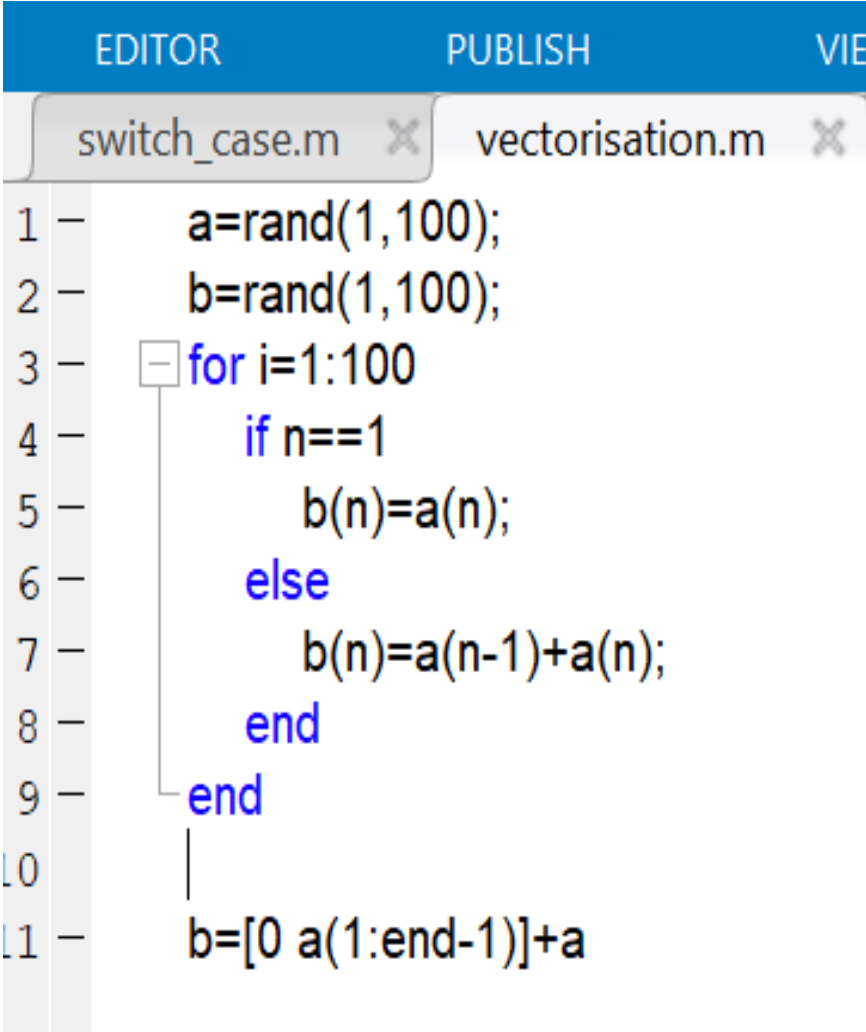
Use the MATLAB *editor* to create a file;

File → New → M-file.

Enter the statements ,

Save the file, for example, vectorization.m

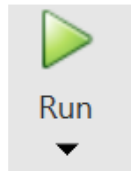
M-file script →



```
1 - a=rand(1,100);
2 - b=rand(1,100);
3 - for i=1:100
4 -     if n==1
5 -         b(n)=a(n);
6 -     else
7 -         b(n)=a(n-1)+a(n);
8 -     end
9 - end
10 -
11 - b=[0 a(1:end-1)]+a
```

M-Files Scripts

- A *script file* is file that contains a sequence of MATLAB statements.
- Script files have a filename extension .m and are often called m-files.
- Run script files using:
 - Filename on Command Window
 >>vectorisation
 - Run icon on the editor tool bar



- After m-file running,
(a, b, and i) variables appear
on the workspace.

Workspace			
Name ▲	Value	Size	B
 a	1x100 double	1x100	
 b	1x100 double	1x100	
 i	1	1x1	

M-Files Scripts

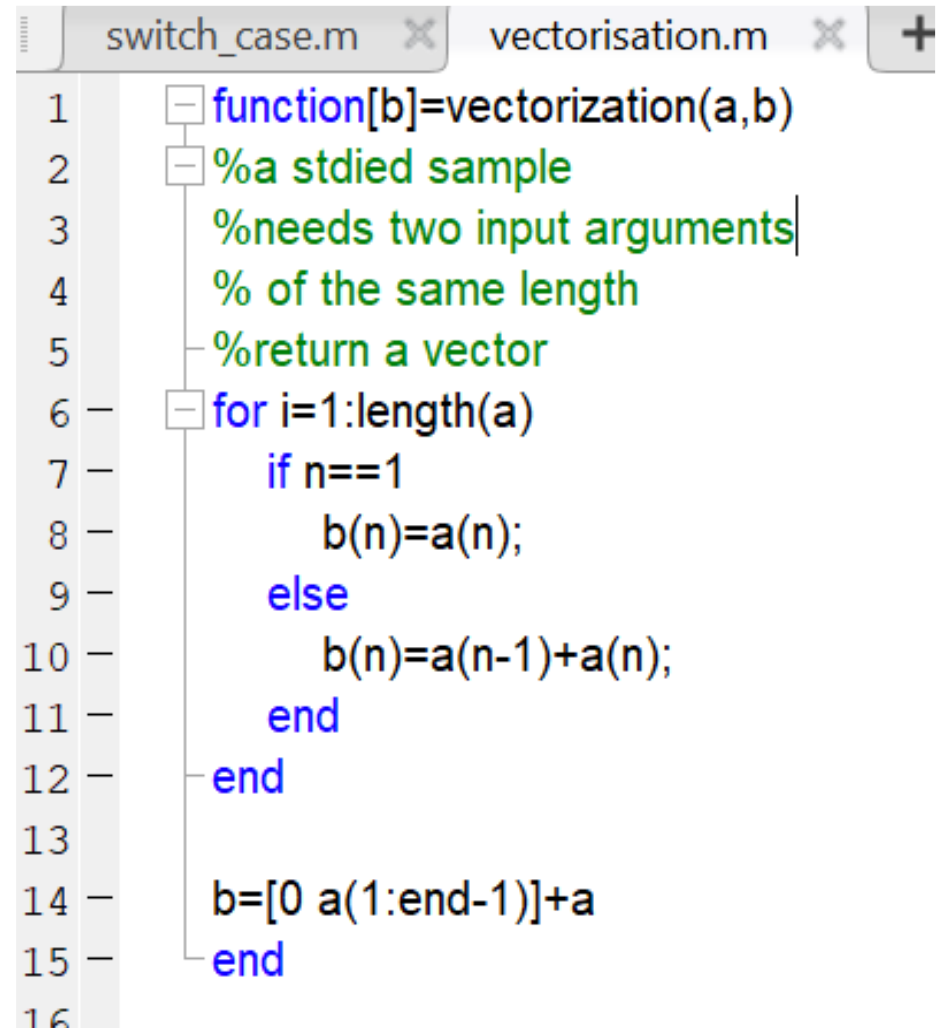
All variables created in a script file are added to the workspace. This may have undesirable effects, because:

- Variables already existing in the workspace may be overwritten.
- The execution of the script can be affected by the state variables in the workspace.

As a result, because scripts have some undesirable side-effects, it is better to code applications (specially complicated ones) using rather function M-file.

User-defined functions

- Functions look exactly like scripts, but for one difference
 - Functions must have a function declaration

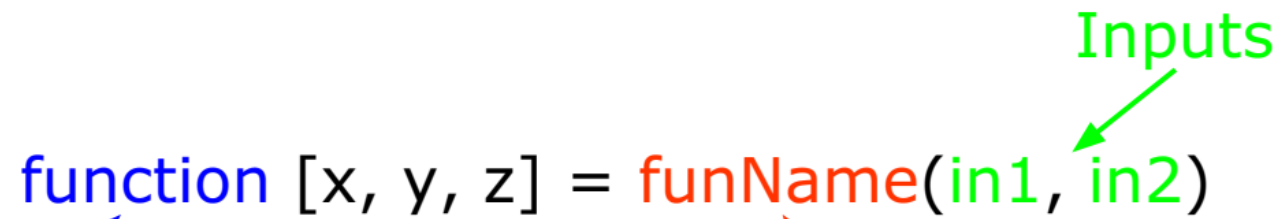


The screenshot shows a MATLAB script editor with two tabs: 'switch_case.m' and 'vectorisation.m'. The 'vectorisation.m' tab is active, displaying a function definition for 'vectorization(a,b)'. The function starts with a comment block explaining its purpose: '%a stdied sample', '%needs two input arguments', '% of the same length', and '%return a vector'. The function body consists of a 'for' loop from 1 to the length of 'a'. Inside the loop, an 'if' statement checks if 'n' is 1; if so, 'b(n)' is assigned the value of 'a(n)'. Otherwise, 'b(n)' is assigned the sum of 'a(n-1)' and 'a(n)'. The loop ends with 'end'. After the loop, 'b' is updated to '[0 a(1:end-1)]+a', and the function ends with 'end'.

```
1 function[b]=vectorization(a,b)
2 %a stdied sample
3 %needs two input arguments
4 % of the same length
5 %return a vector
6 for i=1:length(a)
7     if n==1
8         b(n)=a(n);
9     else
10        b(n)=a(n-1)+a(n);
11    end
12 end
13
14 b=[0 a(1:end-1)]+a
15 end
16
```

User-defined functions

`function [x, y, z] = funName(in1, in2)`



The diagram shows a MATLAB function definition: `function [x, y, z] = funName(in1, in2)`. The word `function` is blue. The output list `[x, y, z]` is black. The function name `funName` is red. The input arguments `in1` and `in2` are green. Annotations include: a blue arrow pointing to `function` with the text 'Must have the reserved word: function'; a black arrow pointing to the output list `[x, y, z]` with the text 'If more than one output, must be in brackets'; a red arrow pointing to `funName` with the text 'Function name should match m-file name'; and a green arrow pointing to the input arguments `in1, in2` with the text 'Inputs'.

Inputs

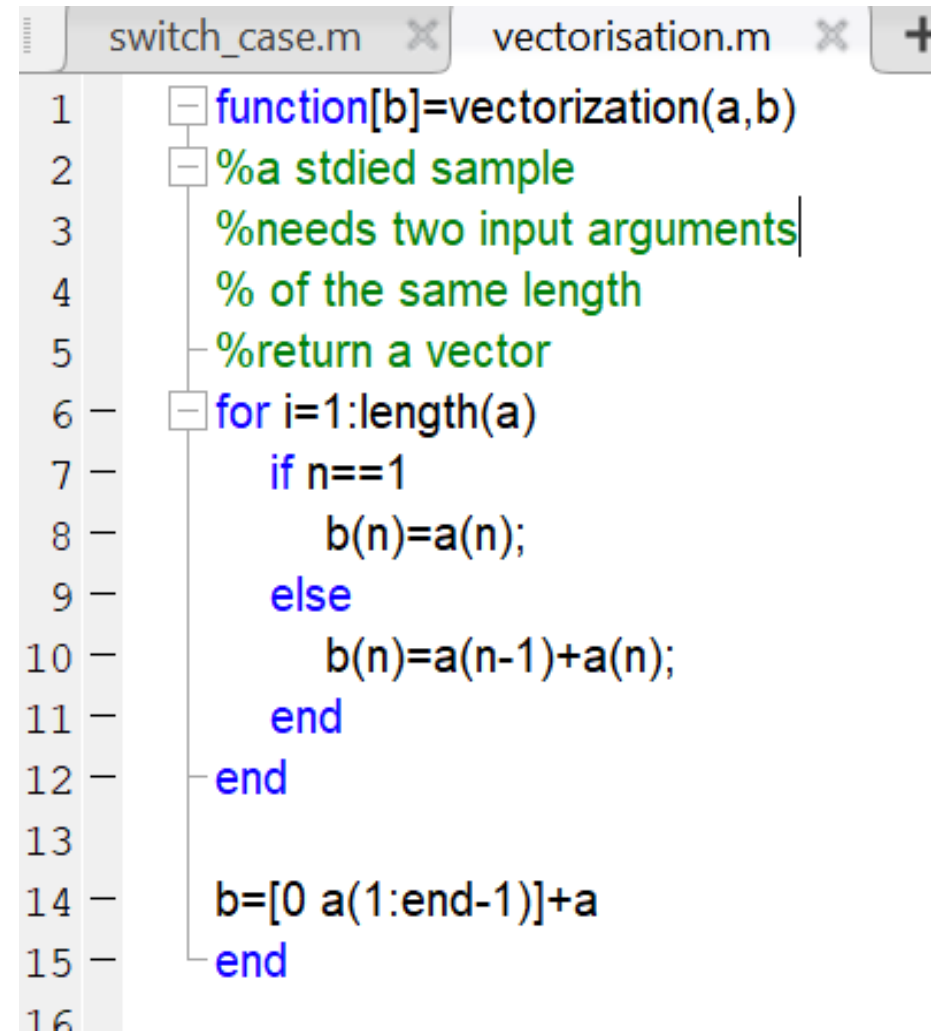
Must have the reserved
word: function

Function name should
match m-file name

If more than one output,
must be in brackets

User-defined functions

- No need for return
- Variable scope: “local variables”
- Invoking functions using their filename (script or command window)



The screenshot shows a MATLAB code editor with two tabs: 'switch_case.m' and 'vectorisation.m'. The 'vectorisation.m' tab is active, displaying a function definition for 'vectorization(a,b)'. The code is as follows:

```
1 function[b]=vectorization(a,b)
2 %a stdied sample
3 %needs two input arguments
4 % of the same length
5 %return a vector
6 for i=1:length(a)
7     if n==1
8         b(n)=a(n);
9     else
10        b(n)=a(n-1)+a(n);
11    end
12 end
13
14 b=[0 a(1:end-1)]+a
15 end
16
```