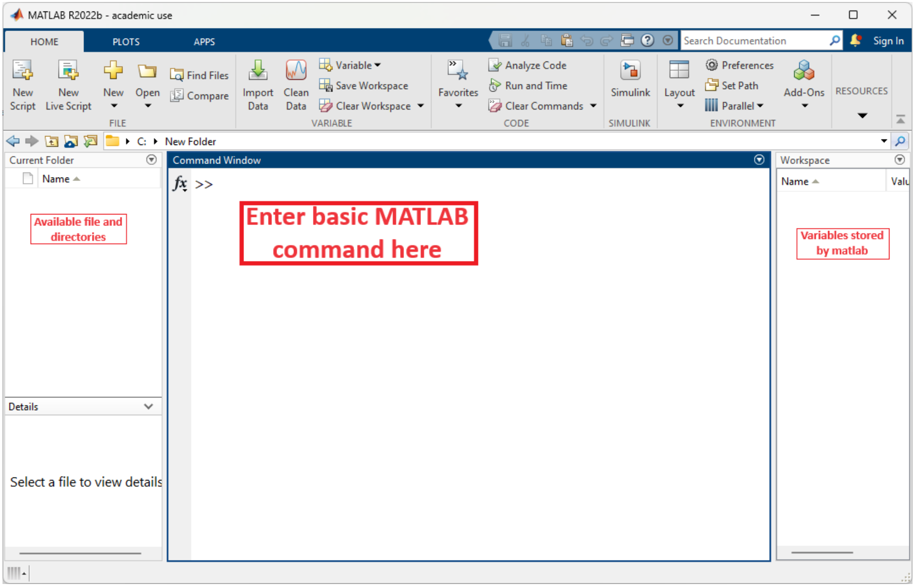
Introduction on MATLAB



Command Window

The command window accepts variables declaration, expressions, function calling and from it we can run scripts.

- Writing in the Command Window:
  $>> a = 1.54$
  create a variable a with value 1.54, and it is stored in MATLAB Workspace
- Adding ; at the end of a command prevent MATLAB from printing the result of the command in the Command Window
- Variable name can be max 31 character (letter or number) and the first must not be a number. Morever names are case-sensitive
- if we write an expression without specifying a variable name, MATLAB will use the default variable ans

Common commands

- clc: deletes all the contents of the Command Window, but mantains the variable stored in the Workspace.
- clear: deletes all the variable in the Workspace.
- clear var: deletes the variable var in the Workspace.
- help followed by a command (or a MATLAB function), prints in the Command Window a small guide on the command.
- If a and b are numbers (or numeric variables), then

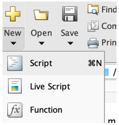
$>> \ a+b$	is the sum
$>> \ a-b$	is the difference
$>> \ a*b$	is the product
$>> \ a/b$	is a divided by b
$>> \ a\backslash b$	is b divided by a
$>> \ a\hat{}b$	is a raised to the power of b.

- Round brackets are used to specify the order of operations in composite expressions, for example the command:
  $>> a = (1+3)/(2\hat{}(1+2));$
  stores in the Workspace the variable a with value 0.5, without showing the results in the Command Window.
- Other usefull math functions available in MATLAB are: sin cos tan asin acos atan exp log log10 abs sqrt sign . . . . For a guide on how they works, use:
  $>>$ help sin, or $>>$ help cos, . . .

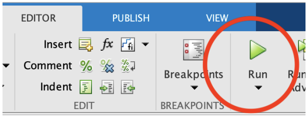
Scripts

A script is a text file (with extension .m, for example Script1.m) containing a sequence of instructions, one per line.

To create a script, use the button 'New'



To execute a script, either press 'Run' or use the file name (for example Script1) without an extension in the Command Window



Executing a script is equivalent to typing the commands in the Command Window

- it has not arguments
- It does not return the results
- All variables in the Workspace can be read/written

example of script: *Script1.m*

```
a=1
b=a+1
a=a*b
```

Executed with:

```
>> Script1
```

Functions

In MATLAB, it is possible to create new functions. You just need to create a file with the .m extension and a filename matching the desired function name. The first line of the file must start with the reserved word function followed by the output arguments, the function name and the input arguments. In the following lines, you should write the code for the function, and finally, the last line should contain the reserved word end.

example of function: *fun1.m*

```
function [x,y] = fun1(a,b)
x=(a+b)/2;
y=sqrt(a^2+b^2);
end
```

a and *b* are the input arguments, while *x* and *y* are the output arguments.

Variables used in a function are local and therefore independent of those in the calling environment.

To add comments to MATLAB code (scripts or functions), use the percent symbol % . Comment lines can appear anywhere in a code file, and you can append comments to the end of a line of code. **example of a function with comments:** *fun1.m*

```
function [x,y] = fun1(a,b)
x=(a+b)/2; % Arithmetic mean
% Second part of the code
y=sqrt(a^2+b^2);% Length of the vector (a,b)
end
```

Matrices and vectors

To define a matrix, you write the coefficients between '`[]`'
the coefficients are separated by '`,`' or by space ' '
the rows are separated by '`;`'

Example: `A=[1,4;2,5;3,6];` defines a variable that contains the matrix

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

Accessing the Coefficients

To access the coefficients, you use row and column indices enclosed in parentheses `()`, where indices start from 1.

Continuation of the example:

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

`A(1,2)`	returns 4
`A(3,2)`	returns 6
`A(1,3)`	Error!
`A(4,1)`	Error!
`A(0,1)`	Error!

The coefficients can be modified

`A(3,2)=9` A becomes $\begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 9 \end{pmatrix}$

Vectors are Matrices

To define a vector, you use a matrix of size $n \times 1$.

Example:

`v=[1;2;3];` defines a variable containing

$$v = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

Note that there are also row vectors: matrices of size $1 \times n$

`w=[1,2,3]` defines a variable containing

$$w = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$$

Accessing the Coefficients

$$v = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \qquad w = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$$

`v(1)`	returns 1
`v(4)`	Error!
`v(0)`	Error!
same for `w`	

Row vectors are $1 \times n$ matrices, and column vectors are $n \times 1$ matrices.

`v(2,1)`	returns 2
`v(1,2)`	Error!
`w(2,1)`	Error!
`w(1,2)`	returns 2

Matrices as Vectors

You can use a single index with matrices as well!

$$A = \begin{pmatrix} a_{1,1} & \cdots & a_{1,m} \\ \vdots & & \vdots \\ a_{n,1} & \cdots & a_{n,m} \end{pmatrix}$$

`A(k)` is `A(s,t)` for (s, t) that satisfies $(t-1)\,n + (s-1) = k$, which means it's the k-th coefficient when going:

- from top to bottom
- from left to right

`A(:)` is the column vector `[A(1) ...A(n*m)]'`.
You can modify `A` by assigning a vector to `A(:)`.
Example: If `A` is a matrix obtained with `A=zeros(2,2);`,

```
A(:)=[1 2 3 4]';
```

is equivalent to `A=[1 3;2 4];`

Scalars are Matrices

A scalar is a 1×1 matrix.

and you can access the single coefficient: if `a=3;`, then `a(1,1)` is 3.
Almost everything in MATLAB is a matrix!

Basic Operations

Knowing the dimensions:
`size(A)` returns a vector containing the number of rows and the number of columns

```
A=[1,4;2,5;3,6]
size(A)                                returns [3,2]
```

For a single dimension (e.g., number of rows), you can use `size(A,1)`

Transpose
An apostrophe `'` to the right of a matrix transposes it.
`v=[1 2 3; 4 5 6]';`
defines the matrix

$$v = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

Note: `[1 2 3]'` is a column vector.

To Verify

Verify that by defining a matrix

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

`A(1)`	`returns 1`
`A(2)`	`returns 2`
`A(3)`	`returns 3`
`A(4)`	`returns 4`
`A(5)`	`returns 5`
`A(6)`	`returns 6`
`A(7)`	`Error!`

...Continuation

Defining a matrix and then modifying it:
Use the functions '`zeros`' or '`ones`'

or with arguments for rows and columns

`A=zeros(3,2)`	`A` is a 3×2 matrix with all zeros
`A=ones(3,2)`	`A` is a 3×2 matrix with all ones

or using a vector for dimensions

`sz=[3,2];`	
`A=zeros(sz)`	`A` is a 3×2 matrix with all zeros
`A=ones(sz)`	`A` is a 3×2 matrix with all ones

Blocks

The instructions in a function can be included in blocks.

In MATLAB, a block starts with a keyword such as

- `if`
- `for`
- `while`
- ...

and ends with the keyword `end`.
You can nest blocks inside another block.

Unlike Python, indentation does not delimit blocks!

if ...end

```
if a==1
  b=9; % executes between "if" and "end" only if "a" is 1
end
```

if ...else ...end

```
if a==1
  b=9; % executes between "if" and "else" if "a" is 1
else
  b=8; % executes between "else" and "end" if "a" is not 1
end
```

Syntax for Multiple `if`

The following three codes are equivalent:

Multiple `if`:

```
if a==1
  disp('one');
else
  if a==2
    disp('two');
  else
    if a==3
      disp('three');
    else
      disp('big')
    end
  end
end
```

if...elseif...:

```
if a==1
  disp('one');
elseif a==2
  disp('two');
elseif a==3
  disp('three');
else
  disp('big')
end
```

switch...case...:

```
switch a
  case 1
    disp('one');
  case 2
    disp('two');
  case 3
    disp('three');
  otherwise
    disp('big')
end
```

For knowing more about the function 'disp', use >> help disp

Conditional Syntax

Comparisons: Take two scalars `a` and `b` and return a logical value between `true` and `false`

`a==b`	true if they are equal
`a~=b`	true if they are not equal
`a<b`	true if `a` is strictly less than `b`
`a<=b`	true if `a` is less than or equal to `b`
`a>b`, `a>=b`	as for less than

Logical Operations:

`a|b`	`a` or `b` (true if at least one is true)
`a&b`	`a` and `b` (true if both are true)
`~a`	true if `a` is false and vice versa

For numeric values, "`a=true`" is equivalent to "$a \neq 0$".

`for ...end`

```
b=0;
for i=3:10
  b=b*b+i;
end
```

Executes between `for` and `end` first with $i = 3$, then with $i = 4$ up to $i = 10$.

`while ...end`

We cannot always use `for`. `a=1;`

```
b=0;
while a<10
  a=a+1;
  b=b*b+1;
end
```

Executes between "while" and "end" repeatedly. Before each iteration, it checks if $a < 10$, and if $a \geq 10$, it proceeds to the next instruction after the block.

`while true` starts an infinite loop.

Use *ctrl-c* to stop an infinite loop.

continue and break

```
a=1;
b=0;
while a<10 % continue jumps here
  a=a+1;
  if b>10
    continue; % or break;
  end
  b=b*b+1;
end
% break jumps here
```

`continue` goes to the condition check of the `while` loop.
`break` jumps to the first instruction after the `while` loop.

These can also be used with `for` loops.

Limitations

If a `while` loop contains another `while` loop, then `continue` and `break` affect the innermost `while` loop that encloses them.

```
i=0;
j=0;
a=1;
while i<10
  i=i+1;
  while j<5
    j=j+1;
    if i+j<10
      continue; % or break;
    end % if
  end % inner while
  a=a*(i+j);
end % outer while
```

To affect the outer `while`, you need to repeat the control block (`if i+j<10 continue; end`).