Introduction to MATLAB

Riccardo Antonello
(riccardo.antonello@unipd.it)

Giulia Michieletto
(giulia.michieletto@unipd.it)

Dipartimento di Tecnica e Gestione dei Sistemi Industriali

Università degli Studi di Padova

4 Marzo 2024

What is MATLAB ?

MATLAB (*MATrix LABoratory*) is a:

– Numerical computing environment.

– Programming language.

MATLAB easily allows:

– Matrix manipulations.

– Data analysis and visualization.

– Implementation of algorithms.

– Interfacing with other programming languages (Java, C/C++/C#, Fortran, Python).

What is MATLAB ?

MATLAB is primarily intended for *numerical computing*, not *symbolic computing* (such as the *Mathematica* or *Maple* environments) [1].

The additional **Simulink** package (tightly integrated with MATLAB) allows to model and simulate dynamical systems by using a *block-diagram-based graphical interface*.

(1) Limited support for symbolic computation is provided with the optional *Symbolic Math Toolbox* (based on the *MuPAD* symbolic engine)

What is MATLAB ?

The MATLAB programming language:

- supports multiple programming paradigms, e.g. *imperative*, *procedural*, *object-oriented*, *functional*.
- operates primarily on matrices.
- is mainly an *interpreted* (scripting) language (1).

The simplest type of MATLAB program is a text file (**MATLAB script**) containing a sequence of MATLAB commands and functions.

(1) Compilation of MATLAB scripts is also possible by using the **MATLAB Compiler.**

Why to use MATLAB ?

- Over 3 millions users worldwide (industry & academia).
- Large collection of readily available functions to perform several computational tasks:
  - **Built-in** functions.
  - Functions included in optional **toolboxes**.
  - Functions provided by large user community (see https://www.mathworks.com/matlabcentral/).
- Easy to learn the basics; it allows to quickly explore several alternatives to get a solution.

Preliminaries

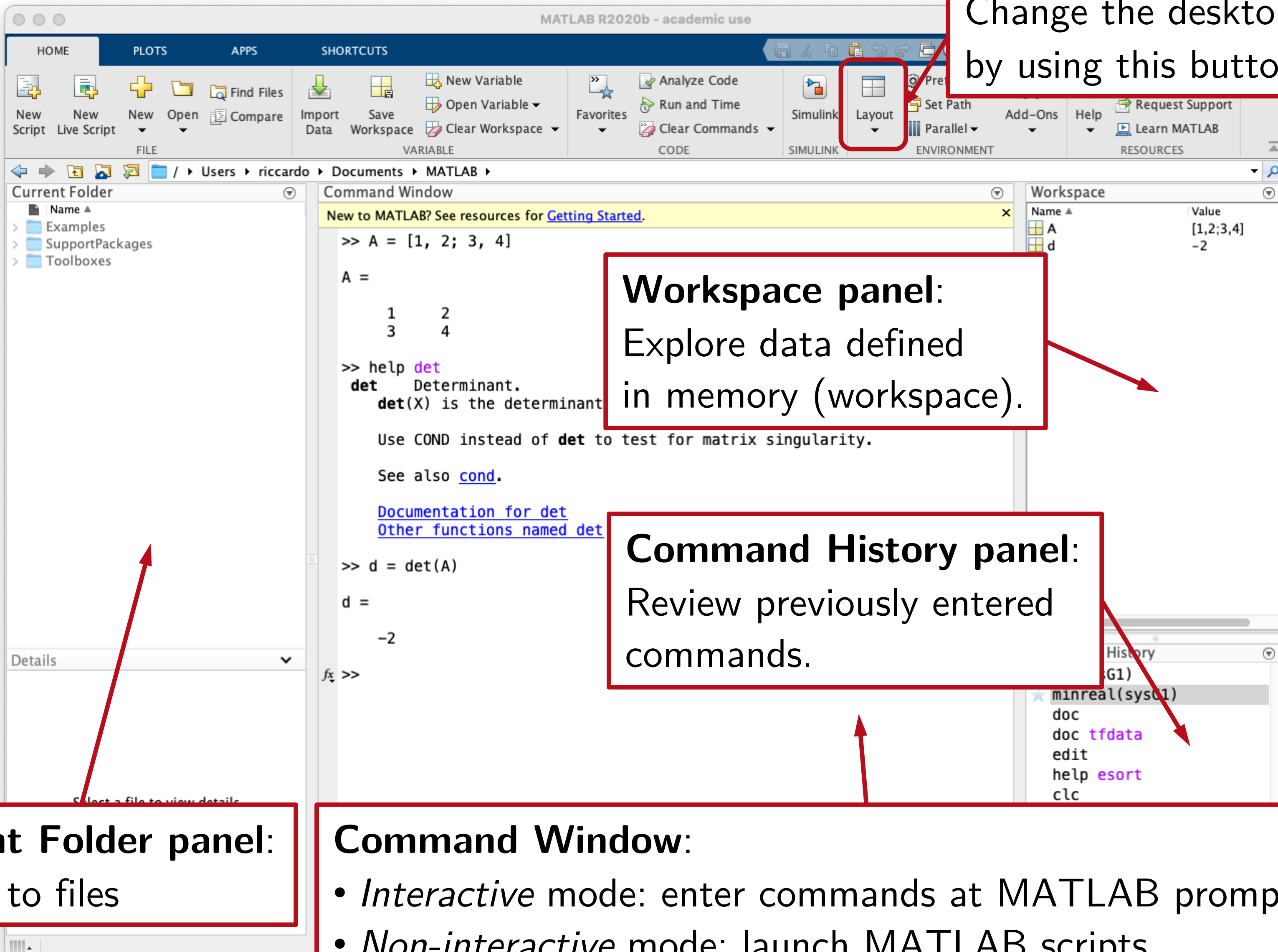
A MATLAB *Total Academic Headcount* (TAH) *License* is available for all the students and employees of University of Padova.

The license allows to install a full copy of MATLAB and companion toolboxes on personally-owned computers.

Instructions for downloading and installing the software can be found here:

https://www.csia.unipd.it/servizi/servizi-utenti-istituzionali/contratti-software-e-licenze/matlab

Desktop Basics



Change the desktop layout by using this button.

Workspace panel: Explore data defined in memory (workspace).

Command History panel: Review previously entered commands.

Current Folder panel: Access to files

Command Window:
- *Interactive* mode: enter commands at MATLAB prompt (>>).
- *Non-interactive* mode: launch MATLAB scripts.

Command Window Basics

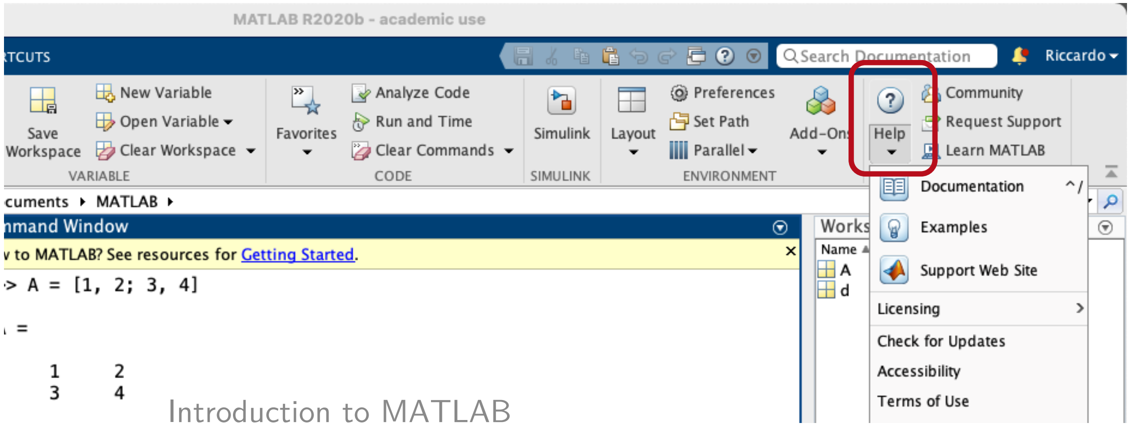
>>	MATLAB prompt.
>> <command> ⏎	Executes a command and *prints the output* on the command window.
>> <command>; ⏎	Executes a command, *without printing the output* on the command window.
>> <command>; <command>; ⏎	Enters multiple commands on a single prompt; *no output shown* on the command window.
>> <command>, <command> ⏎	Enters multiple commands on a single prompt; the *output of each command is shown* on the command window.
>> <long command line> ... ⏎ <continuation of command line> ⏎	Continue a statement to the next line using *ellipsis* (...).
>> ↑ ↓	Use *up* and *down* arrow keys to recall previously entered commands.

Online Help

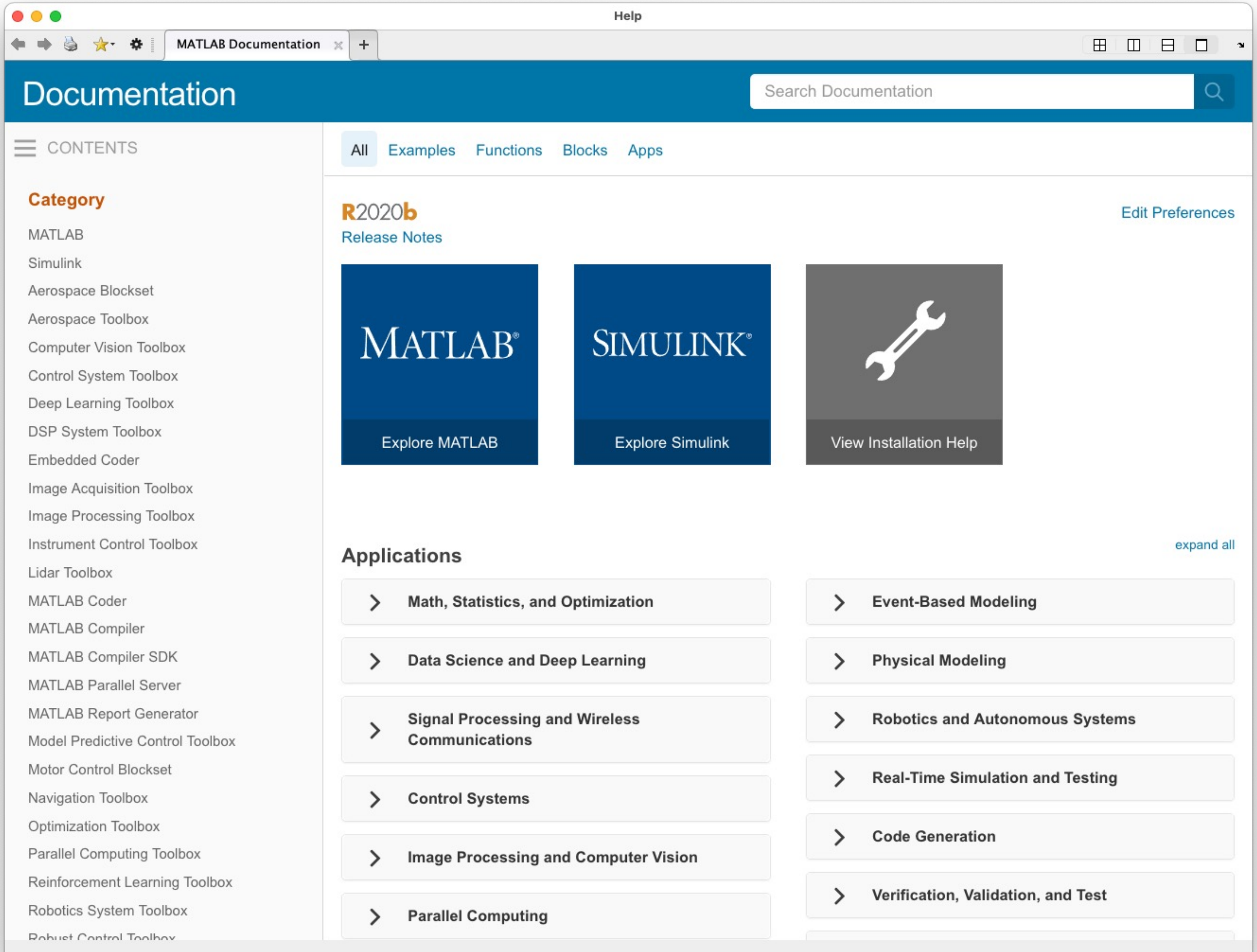
From Command Window:

>> help <name>	Displays the help text for <name> in the Command Window.
>> doc <name>	Displays the documentation for <name> in the Help Browser.
>> doc	Opens the **Help Browser**.
>> lookfor <keyword>	Searches for the specified keyword in the online help.
>> demo	Displays a list of features MATLAB and Simulink examples in the Help browser.

From toolbar:



Help Browser



Workspace

The **workspace** is the area of memory containing the variables created and used in a MATLAB session.

To define a variable in the workspace:

```
>> a = 1
```

Note: the *assignment* operator is denoted with =

`ans` (short for *answer*) is a special variable containing the result of the last computation.

```
>> 1+1
ans =
2
```

If the result of an operation is not assigned to a variable, then it is assigned to the *special variable* **`ans`**.

Workspace

Variables in the workspace are **dynamically allocated**; no *data type declaration* is required.

Variables names:

- start with a letter, followed by letters, digits, or underscores.
- *must* be different from MATLAB keywords.
- *should* be different from names of already existing commands or functions (avoid *name shadowing*).

MATLAB is **case sensitive** (e.g. `a` and `A` are different variables).

Workspace Management

`>> who`	Lists the variables in the current workspace.
`>> whos`	Lists the variables in the current workspace, including size, type, ...
`>> clear <variable>`	Removes a variable from workspace.
`>> clear all`	Removes all the variables from workspace.
`>> save <filename> <variables list>`	Stores variables into a MATLAB formatted binary data file (*MAT-file*, extension `.dat`).
`>> save <filename>`	Stores all the workspace variables into a MAT-file
`>> load <filename>`	Loads variables from a MAT-file into the workspace.
`>> load <filename> <variables list>`	Loads selected variables from a MAT-file into the workspace.

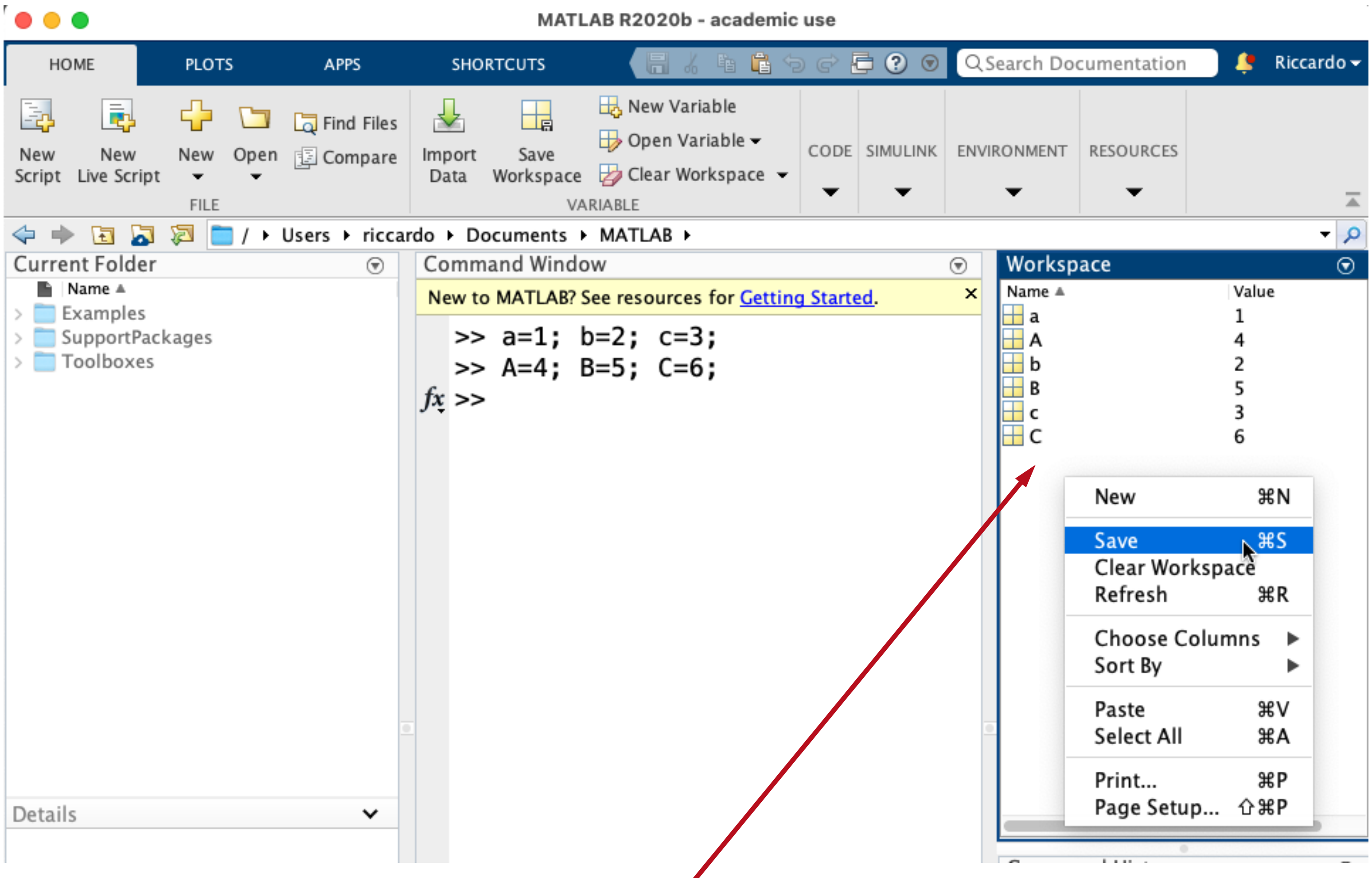
Workspace Management

Example

```
>> a=1; b=2; c=3;
>> A=4; B=5; C=6;

>> save UppercaseVars A B C
>> save LowercaseVars a b c

>> clear all
>> who

>> load UppercaseVars
>> who

Your variables are:

A   B   C

>> load LowercaseVars
>> whos

                ⋮
```

```
                ⋮

Your variables are:

A   B   C   a   b   c

>> clear a b c
>> whos

Your variables are:

A   B   C

>> load LowercaseVars a
>> who

Your variables are:

A   B   C   a
```

Workspace Management



The workspace can be also managed from the *Workspace panel* (right-click on it to access a pop-up menu with several options).

Working Directory Management

`>> pwd`	Print working directory (MATLAB current folder).
`>> dir <path>, ls <path>`	List content of directory specified by `<path>`.
`>> what <path>`	List MATLAB files (e.g. .m, .mat, ...) in directory specified by `<path>`.
`>> which <item>`	Locate functions and files specified by `<item>`.
`>> cd <path>`	Change current directory to `<path>`.
`>> copyfile <source> <dest>`	Copy file or directory `<source>` to `<dest>`.
`>> delete <path>`	Delete the file specified by `<path>`.
`>> mkdir <path>` `>> rmdir <path>`	Make/Remove the directory specified by `<path>`.
`>> !<cmd>` (or `system(<cmd>)`)	Execute operating system command `<cmd>` (Shell escape function).

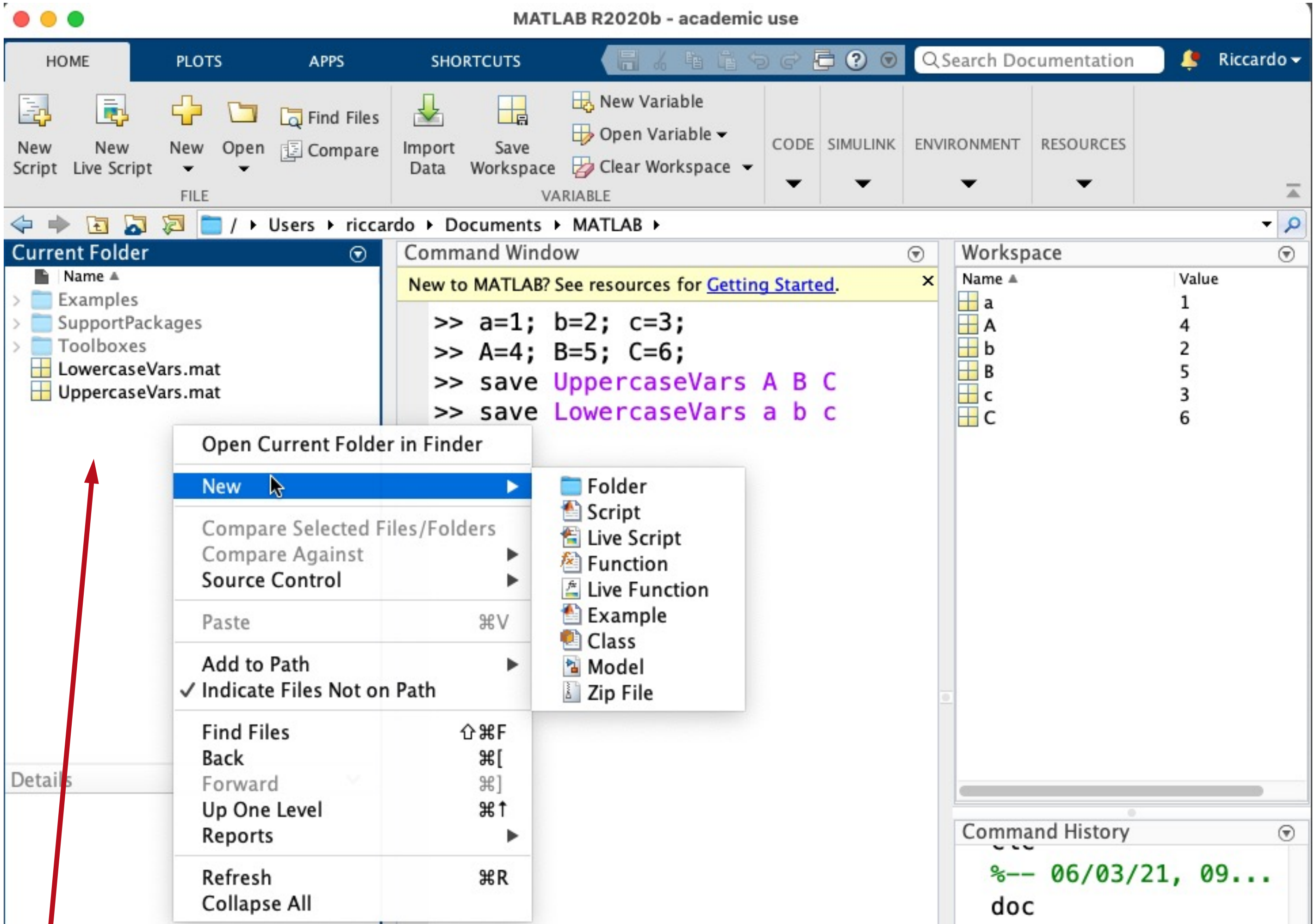
Working Directory Management

Example

```
>> pwd

ans =
 '/Users/riccardo/Documents/MATLAB'

>> ls
Examples          SupportPackages
UppercaseVars.mat
LowercaseVars.mat      Toolboxes

>> what

MAT-files in the current folder
/Users/riccardo/Documents/MATLAB

LowercaseVars  UppercaseVars

                ⋮
```

```
                ⋮

>> delete LowercaseVars.mat
>> what

MAT-files in the current folder
/Users/riccardo/Documents/MATLAB

UppercaseVars
```

Working Directory Management



The working directory can be also managed from the *Current Folder panel* (right-click on it to access a pop-up menu with several options).

Representation of real numbers

Real numbers are internally represented with the finite set of *double-precision* floating-point numbers (IEEE 754 format).

`realmax, realmin`	Largest and smallest IEEE double-precision positive floating-point number.
`Inf`	IEEE arithmetic representation for positive infinity. Every value above `realmax` is represented with `±Inf`; every value below `realmin` is represented with 0.
`NaN`	IEEE arithmetic representation of *Not-a-Number*. This value is used for operations which have an *undefined result* (e.g. $0/0$, ∞/∞, ...).
`eps`	Floating-point relative accuracy. It is the distance from 1.0 to the next larger double-precision number. Its value is 2^{-52}.

Representation of real numbers

When representing real numbers, distinction is made between **internal** and **external** format:

↳ **Internal format**: used *to perform computations*.
It is always the double-precision format.

↳ **External format**: used *to display the numeric values* on the Command Window.
It can be controlled with the **`format`** command.

Representation of real numbers

External format selection

format long	Long, fixed-decimal format with 15 digits after the decimal pt.	`3.141592653589793`
format short	Short, fixed-decimal format with 4 digits after the decimal pt.	`3.1416`
format longE	Long scientific notation with 15 digits after the decimal pt.	`3.141592653589793e+00`
format shortE	Short scientific notation with 4 digits after the decimal pt.	`3.1416e+00`
format longG	More compact format between long and `longE`.	`3.14159265358979`
format shortG	More compact format between short and `shortE`.	`3.1416`
format longEng	Long engineering notation with 15 digits after the decimal pt. Exponent is a multiple of 3.	`3.14159265358979e+000`
format shortEng	Short engineering notation with 4 digits after the decimal pt. Exponent is a multiple of 3.	`3.1416e+000`
format rat	Ratio of small integers.	`355/113`
format compact	Suppress blank lines to show more output on a single screen.	
format loose	Add blank lines to make output more readable.	

Representation of real numbers

Example

```
>> format short
>> a = 1/7

a = 0.1429

>> format compact
>> a

a = 0.1429

>> format long
>> a

a = 0.142857142857143

>> format shortE
>> a

a = 1.4286e-01
            ⋮
```

```
            ⋮

>> format shortEng
>> a

a = 142.8571e-003

>> format loose
>> a

a = 142.8571e-003

>> format rat
>> a

a = 1/7
```

Representation of complex numbers

Complex numbers are represented by a pair of double-precision floating-point numbers (real and imaginary parts).

`1i, 1j`	Imaginary unit. Variables **`i`** and **`j`** can also be used, but: • numerical robustness in complex arithmetic is reduced. • can be easily overridden by user-defined variables.
`<complex num> = <real part> + 1i*<imag part>`	Algebraic (*cartesian*) notation.
`<complex num> = <mag> * exp(1i*<arg>)`	*Polar* notation.
`<complex num> = complex(<real part>,<imag part>)`	Using **`complex`** function.

Representation of complex numbers

Example

```
>> format short
>> a = 1+1i

a =

   1.0000 + 1.0000i

>> b = complex(-2,3)

b =

  -2.0000 + 3.0000i

>> c = 2*exp(1i*pi/2)

c =

   0.0000 + 2.0000i

                    ⋮
```

```
                    ⋮
>> d = 1+i

d =

   1.0000 + 1.0000i

>> i = 1;
>> e = 1+i

e =

     2
```

`i` and **`j`** also denote, by default, the imaginary unit; however, differently from the special quantities **`1i`** and **`1j`**, they can be assigned to different values, so that they no longer refer to the imaginary unit.

Representation of vectors & matrices

Definition of vectors and matrices:

- *Square brackets* (`[]`): enclose the elements.
- *Comma* (`,`) or *space* : separate elements on the same row.
- *Semicolon* (`;`) : separates the rows.

Row vector (1×m)	`>> A=[1 2 3]` or `>> A=[1, 2, 3]`
Column vector (n×1)	`>> A=[1; 2; 3]` *or* `>> A=[1` ↵ `2` ↵ `3]`

Matrix (n×m)	`>> A=[1 2 3; 4 5 6]` *or* `>> A=[1, 2, 3; 4, 5, 6]`
	or
	`>> A=[1 2 3;` ↵
	`4 5 6]`

<u>Note</u>: vectors are treated as single-column/single-row matrices.

Representation of vectors & matrices

Definition of matrices with particular structure:

[]	Empty matrix.
eye(n)	$n{\times}n$ identity matrix.
zeros(n,m)	$n{\times}m$ matrix with elements equal to 0.
ones(n,m)	$n{\times}m$ matrix with elements equal to 1.
diag(<vector>)	Diagonal matrix with elements of <vector> on the leading diagonal.
<min_val>:<max_val>	Row vector with increasing elements from <min_val> to <max_val>, with incremental step equal to 1.
<min_val>:<step>:<max_val>	Row vector with increasing elements from <min_val> to <max_val>, with incremental step equal to <step>.

Representation of vectors & matrices

linspace(<min_val>, <max_val>, <num_of_elements>)	Vector of <num_of_elements> elements evenly spaced from <min_val> to <max_val>.
logspace(<min_val>, <max_val>, <num_of_elements>)	Vector of <num_of_elements> elements logarithmically spaced (base 10) from <min_val> to <max_val>.
rand(n,m)	$n \times m$ matrix with uniformly-distributed random real numbers in the interval $(0,1)$.
randn(n,m)	$n \times m$ matrix with normally-distributed random real numbers (mean $=0$, variance $=1$).
randi(N,n)	$n \times n$ matrix with uniformly-distributed random integer numbers in the interval $[1, N]$.
toeplitz, **magic**, **hilb**, **invhilb**, **vander**, **pascal**, **hadamard**, **hankel**, **rosser**, **wilkinson**, …	Matrices with special structure (consult online documentation).

Representation of vectors & matrices

<u>Example</u>

```
>> A = ones(1,3)

A =
     1     1     1

>> B = zeros(2,1)

B =
     0
     0

>> C = diag([2 3])

C =
     2     0
     0     3

              ⋮
```

```
              ⋮
>> a = 1:5

a =
     1     2     3     4     5

>> b = 1:2:10

b =
     1     3     5     7     9

>> C = linspace(-1,1,4)

C =
   -1.0000   -0.3333    0.3333    1.0000

>> D = randi(10, 1, 5)

D =
     9     7     4    10     1
```

Array Indexing

`A(i,j)`	Element at i^{th} row and j^{th} column of matrix `A`. Note: *row/column indexes start from 1 (not 0 !).*
`A(i,:)`	i^{th} row of matrix `A`.
`A(:,j)`	j^{th} column of matrix `A`.
`A(<vector of row indexes>, <vector of column indexes>)`	Submatrix of `A` composed by elements located at rows indexed by `<vector of row indexes>`, and columns indexed by `<vector of column indexes>`. Note: the vectors of row/column indexes can be generated with the notation: `<min_val>:<step>:<max_val>` **`end`** can be used to index the last row/column.
`length``(X)`	Length of vector `X`.
`[N,M]=`**`size`**`(X)`	Size (rows `N` and columns `M`) of matrix `X`.

Array Indexing

Example

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

A =

     1     2     3
     4     5     6
     7     8     9

>> A(2,:)

ans =
     4     5     6

>> A([1 3], [2 3])

ans =

     2     3
     8     9

          ⋮
```

```
          ⋮
>> A(1:2,:)

ans =

     1     2     3
     4     5     6

>> size(A)

ans =

     3     3

>> length(A(1,:))

ans =

     3
```

Dynamic Resizing

Size of vectors and matrices is *dynamically* (i.e. *on-the-fly*) adjusted when needed.

If `A` has not been previously defined ... `>> A(2,3)=1`	... *creates* a 2×3 matrix with element (2,3) equal to 1, and the remaining to 0.
If `A` is a 2×2 matrix ... `>> A(2,3)=1`	... *adds* an extra column to `A`, with bottom element equal to 1, and the remaining to 0.
If `A` is a 2×2 matrix and `B` a 2×3 matrix... `>> A=[A,B]`	... *resizes* A to 2×5 matrix, whose columns are those of the original `A`, followed by those of `B`.
If `A` is a 2×2 matrix and `B` a 3×2 matrix... `>> A=[A;B]`	... *resizes* `A` to 5×2 matrix, whose rows are those of the original `A`, followed by those of `B`.

Representation of strings

Strings are *row* vectors of characters.

`'a'`	A character is defined by enclosing it within *single quotes*.
`'abcd'`	A string is defined by enclosing its characters within *single quotes*.
`['a','b','c','d']`	A string can be alternatively defined as a *row vector of characters*.

`>> s='abc';`
`>> s(2)='a'; s(3)='';`
`>> s`
`s =`
`    'aa'`
	String characters can be indexed as elements of conventional arrays.

`>> s1='abc'; s2='def';`
`>> s=[s1,s2]`
`s =`
`    'abcdef'`
	Strings concatenation is performed as a concatenation of conventional row vectors.

Representation of polynomials

Polynomials are represented as *row vectors* containing coefficients ordered by *descending* powers of the independent variable.

```
>> p = [2, 0, -1, 3];
```

Representation of the polynomial:

$$p(x) = 2x^3 + 0x^2 - x + 3$$

Representation of boolean values

Boolean variables are called **logical variables** in MATLAB.

```
>> a=true
a =
  logical
  1

>> b=false
b =
  logical
  0
```

A logical variable can assume only the value **`true`** or **`false`** (*predefined keywords* for **`logical`**`(1)` and **`logical`**`(0)`).

```
>> logical([0,1,2])
ans =
    1×3 logical array
     0   1   1
```

Any nonzero numerical value is casted to logical true when using **`logical`**`(...)` .

Struct arrays

A **structure array** (**struct**) is a data type that groups related data using data containers called *fields.*

Each field can contain any type of data; use the "dot-notation" to access the data in a field:

```
>> student.firstName = 'Charlie';
>> student.lastName = 'Brown';
>> student.age = 8;
```

`student` is a struct with 2 string fields (`firstName`, `lastName`) and a numeric field (`age`).

Operations with matrices

Matrix addition and subtraction:

```
>> A = [0 1; 2 3];
>> B = ones(2,2);
>> A+B
ans =
     1  2
     3  4
```

Operators **+** (**plus**) and **–** (**minus**) are used to *add* and *subtract* matrices of the same size.

```
>> A+1
ans =
     1  2
     3  4
```

If one of the operands is a scalar, then the same operation is repeated for each element of the other operand.

```
>> minus(1,B)
ans =
     0  0
     0  0
```

(*Rarely used*) **plus** and **minus** are the function equivalents to operators **+** & **–**.

Operations with matrices

Matrix multiplication (row-by-column):

```
>> A = [0 1; 2 3];
>> B = [0 1; 1 0];
>> A*B
ans =
     1   0
     3   2
```

Operator **`*`** (**`mtimes`**) is used to compute the *matrix multiplication* (row-by-column).

```
>> A*2
ans =
     0   2
     4   6
```

If one of the operands is a scalar, then the same operation is repeated for each element of the other operand.

```
>> mtimes(A,B)
```

(*Rarely used*) Function equivalent to operator **`*`**.

Operations with matrices

Matrix power:

```
>> A = [0 1; 2 3];
>> A^2
ans =
      2    3
      6   11
```

Operator **^** (**mpower**) is used to compute *matrix powers* (e.g. `A^2 = A*A`).

```
>> A^-1
ans =
   -1.5000    0.5000
    1.0000         0
```

```
>> inv(A)
ans =
   -1.5000    0.5000
    1.0000         0
```

`A^-1` denotes (if exists) the *inverse* of the (square) matrix `A`.

The inverse of a square matrix can be alternatively computed with the function **inv**(...).

```
>> mpower(A,2)
```

(*Rarely used*) Function equivalent to operator **^**.

Operations with matrices

Solving systems of linear equations:

```
>> A=[1, 2; 3, 4];
>> B=[1; 2];
>> x=A\B
x =
         0
    0.5000
```

The operator **** (**`mldivide`**) is used to compute (if exists) the *solution of systems of linear equations* of the type: $A\,x = B$.

If A is a $n{\times}n$ nonsingular matrix and B has n rows, then the operator \ computes (*with a numerically robust algorithm*) the solution: $x = A^{-1}B$.

```
>> A=[1, 2; 3, 4; 5, 6];
>> B=[1; 2; 4];
>> x=A\B
x =
  0.6667
  0.0833
```

If A is a $n{\times}m$ matrix with $n \neq m$ and B has n rows, then the operator \ computes the *least-squares solution*:

$$x = (A^T A)^{-1} A^T B$$

```
>> x=mldivide(A,B)
```

(*Rarely used*) Function equivalent to operator \.

Operations with matrices

```
>> A=[1, 2; 3, 4];
>> B=[1, 2];
>> x=B/A
x =
   1   0
```

The operator **/** (**`mrdivide`**) is used to compute (if exists) the *solution of systems of linear equations* of the type: $x\,A = B$.

If A is a $n{\times}n$ nonsingular matrix and B has n columns, then the operator / computes (*with a numerically robust algorithm*) the solution: $x = B\,A^{-1}$.

```
>> A=[1 1 3; 2 0 4; -1 6 -1];
>> B=[2 19 8];
>> x=B/A
x =
   1.0000    2.0000    3.0000
```

If A is a $n{\times}m$ matrix with $n \neq m$ and B has m columns, then the operator / computes the *least-squares solution*:

$$x = BA^T(A\,A^T)^{-1}$$

```
>> x=mrdivide(B,A)
```

(*Rarely used*) Function equivalent to operator /.

Operations with matrices

Matrix transpose:

```
>> A=[1, 2; 3, 4];
>> A'
ans =
      1    3
      2    4
```

The operator `'` is used to compute the *transpose* of a *real* matrix.

```
>> A =[1, 1+1i; 2-1i, 2];
>> A'
 1.0000 + 0.0000i   2.0000 + 1.0000i
 1.0000 - 1.0000i   2.0000 + 0.0000i
```

With a matrix of *complex numbers*, the operator `'` computes its *conjugate-transpose* (*Hermitian*).

```
>> A.'
 1.0000 + 0.0000i   2.0000 - 1.0000i
 1.0000 + 1.0000i   2.0000 + 0.0000i
```

To compute only the *transpose* (*without* conjugation), use the operator `.'`

```
>> transpose(A)
 1.0000 + 0.0000i   2.0000 - 1.0000i
 1.0000 + 1.0000i   2.0000 + 0.0000i
```

(*Less used*) Use the **`transpose`**(...) routine.

Operations with matrices

Element-wise operators: obtained by preceding the operator with `.` (dot).

```
>> A=[0,1;2,3]; B=[1,2;3,4];
>> A.*B
ans =
     0     2
     6    12
```

Element-wise multiplication (`times`).

```
>> A.^2
ans =
     0     1
     4     9
```

Element-wise power (`power`).

```
>> A./B
ans =
         0    0.5000
    0.6667    0.7500
```

Element-wise right divide (`rdivide`).

Relational operators

```
>> a=1; b=2;
>> a==b, a~=b
ans =
 logical
  0
ans =
 logical
  1
```

Equal to (**==** ; function equivalent: **eq**),
Not equal to (**~=** ; function equivalent: **ne**).

Note: a relational operator returns a logical value.

```
>> a>b, a>=b
ans =
 logical
  0
ans =
 logical
  0
```

Greater than (**>** ; function equivalent: **gt**),
Greater than or equal to (**>=** ; function equivalent: **ge**).

```
>> a<b, a<=b
ans =
 logical
  1
ans =
 logical
  1
```

Less than (**<** ; function equivalent: **lt**),
Less than or equal to (**<=** ; function equivalent: **le**).

Logical operators

```
>> a=0; b=1;
>> a & b
ans=
 logical
  0
```

Logical AND (function equivalent: **`and`**).

Note: any nonzero value is equivalent to logical **`true`**, and **`false`** otherwise.

```
>> a | b
ans=
 logical
  1
```

Logical OR (function equivalent: **`or`**).

```
>> ~a
ans=
 logical
  1
```

Logical NOT (function equivalent: **`not`**).

Logical operators

```
>> a=0; b=1; c=2;
>> (a > 0) && (b/c < 1)
ans=
  logical
   0
```

Logical AND with *short-circuiting.*

With *logical short-circuiting*, the 2nd operand is evaluated only when the result is not fully determined by the 1st operand.

In the example, the 2nd operand(`b/c < 1`) is not evaluated, because the result can be determined from the 1st operand.

```
>> (a > 0) || (b/c < 1)
ans=
  logical
   1
```

Logical OR with *short-circuiting.*

In the example, the 2nd operand(`b/c < 1`) is evaluated, because the result can not be determined from the 1st operand.

`all` & `any` functions

```
>> a = [0,2]; b = [3,1];
>> a & b
ans=
  1×2 logical array
  0    1

>> a < b
ans=
  1×2 logical array
  1    0
```

Note: logical and relational operators apply *element-wise* to vector and matrices operands.

```
>> all(a > 0)
ans=
 logical
  0

>> any(a > 0)
ans=
 logical
  1
```

Use the **`any`** and **`all`** functions to reduce each logical vector to a *single logical condition*.

`all` determines if *all* the array elements are nonzero or `true`.

`any` determines if *any* of the array elements are nonzero or `true`.

Operations with polynomials

```
>> A = [1, 1, 0]; B = [1, 1];
>> C = A + [0, B]
C =
    1    2    1
```

Given the polynomials $A(x) = x^2 + x$ and $B(x) = x + 1$, evaluates their sum:

$$C(x) = A(x) + B(x) = x^2 + 2x + 1$$

```
>> C = conv(B, B)
C =
    1    2    1
```

Polynomial multiplication is performed by using the function **conv** (*convolution*):

$$C(x) = B(x)B(x) = x^2 + 2x + 1$$

```
>> [Q,R] = deconv(C, B)
Q =
    1    1

R =
    0    0    0
```

Polynomial division is performed by using the function **deconv** (*deconvolution*):

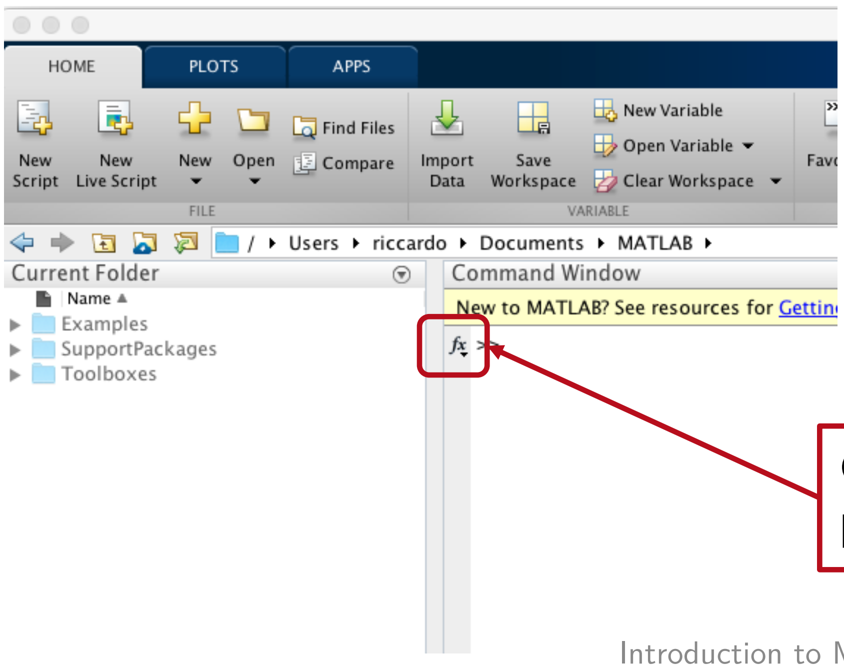
Note: Q and R are the quotient and the reminder, so that:

$$C(x) = Q(x)B(x) + R(x)$$

Functions Library

- MATLAB comes with a huge library of functions (either *built-in* or in optional *toolboxes*); it is impossible to provide an exhaustive list ...

- Explore the library with the **Function Browser**:



Click the **Browse for functions** (*fx*) button to open the *Function Browser.*

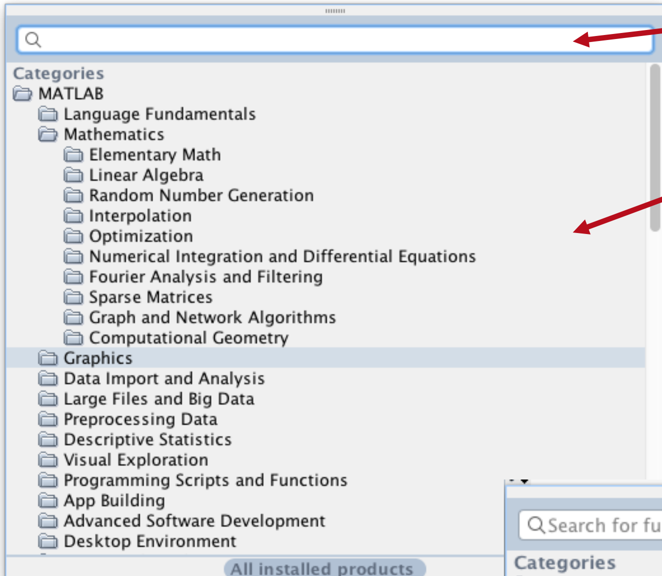
Functions Library

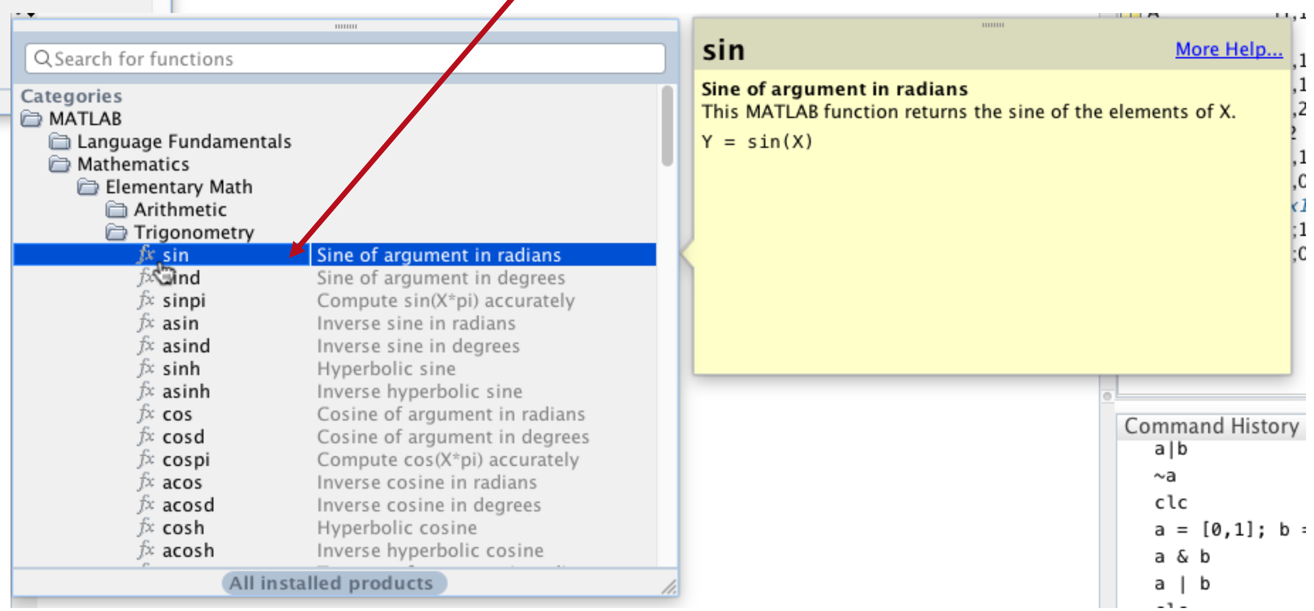
Function Browser

Search for a function.

Browse by category.

Select an entry to get a brief description.





Functions Library

- Most of the functions for scalars also operate (*element-wise*) with vector/matrix arguments.

- Calling scheme for a function:

```
[<out1>,<out2>, …] = <fcn_name>(<in1>, <in2>, … )
```

`<fcn_name>`	Function name.
`<out1>,<out2>, …`	*Return parameters* (output variables).
`<in1>, <in2>, …`	*Input arguments* (input variables).

Elementary Math: Arithmetic Functions

Rounding functions

round(x)	Round to nearest integer.
fix(x)	Round to the nearest integer toward zero.
ceil(x)	Round to the nearest integer greater than or equal to `x`.
floor(x)	Round to the nearest integer less than or equal to `x`.

Rational approximation functions

rem(a,b)	returns the remainder after division of a (dividend) by b (divisor); equivalent to: `r=a-b.*fix(a./b)`.
mod(a,m)	returns the remainder after division of a (dividend) by m (divisor); equivalent to: `r=a-m.*floor(a./m)`.
rats(x)	returns a character vector containing the rational approximations to the elements of `x`, using the default length of 13.

Elementary Math: Arithmetic Functions

Example

```
>>>> round([-0.4, 0.4, 0.6])

ans =

     0     0     1

>> fix([-0.4, 0.4, 0.6])

ans =

     0     0     0

>> floor([-0.4, 0.4, 0.6])

ans =

    -1     0     0

                ⋮
```

```
                ⋮

>> ceil([-0.4, 0.4, 0.6])

ans =

     0     1     1

>> rem(3,2)

ans =

     1

>> rats(1.5)

ans =

    '      3/2      '
```

Elementary Math: Arithmetic Functions

Sum & product or array elements

sum(X)	Sum of the elements of X [(1)].
diff(X)	Differences between adjacent elements of X [(1)].
prod(X)	Product of the elements of X [(1)].
cumsum(X)	Cumulative sum of the elements of X [(1)].
movsum(X,k)	Returns an array of local k-point sums, where each sum is calculated over a *sliding* window of length k across neighboring elements of X [(1)].
cumprod(X)	Cumulative product of the elements of X [(1)].

(1) Operation performed along the 1st array dimension whose size is greater than 1.

Elementary Math: Arithmetic Functions

Example

```
>> a = [1 2 4];
>> sum(a)

ans =

     7

>> diff(a)

ans =

     1     2

>> prod(a)

ans =

     8

                ⋮
```

```
                ⋮
>> cumsum(a)

ans =

     1     3     7

>> A = [1 5 3; 4 2 6];
>> sum(A,1)

ans =

     5     7     9

>> sum(A,2)

ans =

     9
    12
```

Elementary Math: Complex Numbers Functions

<u>Complex arithmetic functions</u>

i, j, 1i, 1j	*Imaginary unit*. Use `1i` and `1j` for improved numerical robustness (and to avoid possible *name-shadowing* with user-defined variables).
complex(a,b)	Defines the complex number `z = a + 1i*b`.
real(z)	Returns the real part of complex number `z`.
imag(z)	Returns the imaginary part of complex number `z`.
abs(z)	Returns the magnitude of complex number `z`.
angle(z)	Returns the phase angle of complex number `z`.
conj(z)	Returns the conjugate of complex number `z`. Alternatively, use `z'` (operator form).

Elementary Math: Complex Numbers Functions

Example

```
>> a = (1/2)+(sqrt(3)/2)*1i

a =

   0.5000 + 0.8660i

>> real(a)

ans =

    0.5000

>> imag(a)

ans =

    0.8660
```

⋮

⋮

```
>> abs(a)

ans =

    1.0000

>> angle(a)

ans =

    1.0472

>> rad2deg( angle(a) )

ans =

   60.0000
```

Elementary Math: Transcendental functions

Trigonometric functions

pi	Floating-point number nearest the value of π.
sin(x), **cos**(x), **tan**(x)	Sine, cosine and tangent of x in [rad]. Use **sind**, **cosd**, **tand** if x is in [deg] units.
sec(x), **csc**(x), **cot**(x)	Secant, cosecant and cotangent of x in [rad]. Use **secd**, **cscd**, **cotd** if x is in [deg] units.
asin(x), **acos**(x), **atan**(x)	Inverse sine, cosine and tangent of x; return value is in [rad]. Use **asind**, **acosd**, **atand** to get a return value in [deg] units.
atan2(y,x)	*Four-quadrant* inverse tangent of y and x; return value is in [rad].
sinh(x), **cosh**(x), **tanh**(x), **sech**(x), …	Hyperbolic functions.
deg2rad(x), **rad2deg**(x)	*deg-to-rad* and *rad-to-deg* unit conversions.

Elementary Math: Transcendental functions

Exponential functions

exp(x)	Exponential of x: e^x
pow2(x)	Base-2 power of x: 2^x
log(x), **log2**(x), **log10**(x)	Natural logarithm, base-2 logarithm and base-10 logarithm of x.
expm(X)	Matrix exponential: $$\exp(X) = \sum_{k=0}^{+\infty} \frac{1}{k!} X^k$$
sqrt(x), **nthroot**(x)	Square root and n^{th} real root of x.

Elementary Math: Polynomial Functions

Functions for polynomials

polyval(p,x)	Evaluates the polynomial p at each point in x.
roots(p)	Computes the roots of polynomial p.
poly(r)	Returns the polynomial whose roots are the elements of the vector r.
residue(b,a)	Partial fraction decomposition of the ratio of the polynomials a and b.
conv(a,b), **deconv**(a,b)	Multiplication (convolution) and division (deconvolution) of polynomials a and b.
polyint(p), **polyder**(p)	Integral and derivative of polynomial p.

Elementary Math: Polynomial Functions

Example

```
>> p = [1 2 1];
>> polyval(p, 0)

ans =
     1

>> roots(p)

ans =

    -1
    -1

>> poly([-1 -1])

ans =

     1     2     1
```

⋮

⋮

```
>> [R,P,K] = residue([2 1], [1 3 2])
R =
     3
    -1

P =
    -2
    -1

K =
     []
```

$$\frac{B(x)}{A(x)} = \frac{2x+1}{x^2+3x+2} = \cdots$$

$$\cdots = \frac{R(1)}{x-P(1)} + \frac{R(2)}{x-P(2)} + K = \frac{3}{x+2} + \frac{-1}{x+1}$$

Linear Algebra Functions

Linear Algebra Functions

det(X)	Determinant of square matrix X.
rank(X)	Rank of matrix X.
trace(X)	Sum of diagonal elements (trace) of square matrix X.
inv(X)	Inverse of non-singular square matrix X.
eig(X)	Eigenvalues and eigenvectors of square matrix X.
poly(X)	Characteristic polynomial of square matrix X.
svd(X)	Singular values decomposition of matrix X.

Basic statistics

Descriptive statistics – basic functions

min(X), **max**(X)	Minimum/maximum elements of array X.
mink(X,k), **maxk**(X,k)	k smallest/largest elements of array X.
mean(X)	Average (mean value) of elements of array X.
var(X), **std**(X)	Variance/Standard-deviation of elements of array X. Note: if X has N elements, then the variance is normalized to $N-1$ by default (*unbiased sample variance*).
median(X)	Median value of elements of array X (i.e. middle value separating the higher and lower halves of sorted X) .
mode(X)	Most frequent value (*mode*) in array X.
sort(X)	Sort elements of array X.

Basic statistics

Example

```
>> a = [1 9 7 2 5];
>> min(a)

ans =
     1

>> max(a)

ans =
     9

>> maxk(a,2)

ans =
     9     7

            ⋮
```

```
                  ⋮
>> mean(a)

ans =
    4.8000

>> var(a)

ans =
   11.2000

>> sort(a)

ans =
     1     2     5     7     9

>> median(a)

ans =
     5
```

Strings Functions

Functions for Characters & Strings

strcat(s1, …, sN)	Concatenates string s1,…,sN horizontally; use row vectors concatenation [s1,…,sN] to *preserve spaces*.
strrep(s, old, new)	Replaces all occurrence of the string old in the string s with the new string new.
lower(s), **upper**(s)	Converts string s to lower-case or upper-case.
strncmp(s1,s2,n)	Compares the first n characters of strings s1 and s2. Note: Comparison is case-sensitive; for case-insensitive comparison, use **strncmpi**.
num2str(A)	Converts the numeric array A into its character representation.
str2num(s)	Converts the string s to a numeric value.

Strings Functions

Example

```
>> s1 = 'Hello';
>> s2 = 'World';
>> s3 = strcat(s1,' ',s2,' !')

s3 =

    'HelloWorld !'

>> s4 = [s1, ' ', s2, ' !']

s4 =

    'Hello World !'

>> strrep(s4, s2, 'Folks')

ans =

    'Hello Folks !'

                ⋮
```

```
                ⋮

>> a = [1, 2, 3; 4, 5, 6]
>> num2str(a)

ans =

  2×7 char array

    '1  2  3'
    '4  5  6'

>> str2num('123.456')

ans =

  123.4560
```