The very first step into MATLAB

The purpose of this introduction session is to get you started with MATLAB, a ubiquitous scientific computing software, that will prove very useful for you to learn signals and systems in the next chapters.

We begin by providing a brief description of the MATLAB interface in Section 1. This section will be demonstrated to you by one of the instructors (using the Impero software).

Section 2 contains several simple examples to showcase the basic commands and syntax used in MATLAB. You should first familiarise yourself with the interface and then reproduce the examples included in Section 2. Finally try to solve the last exercise.

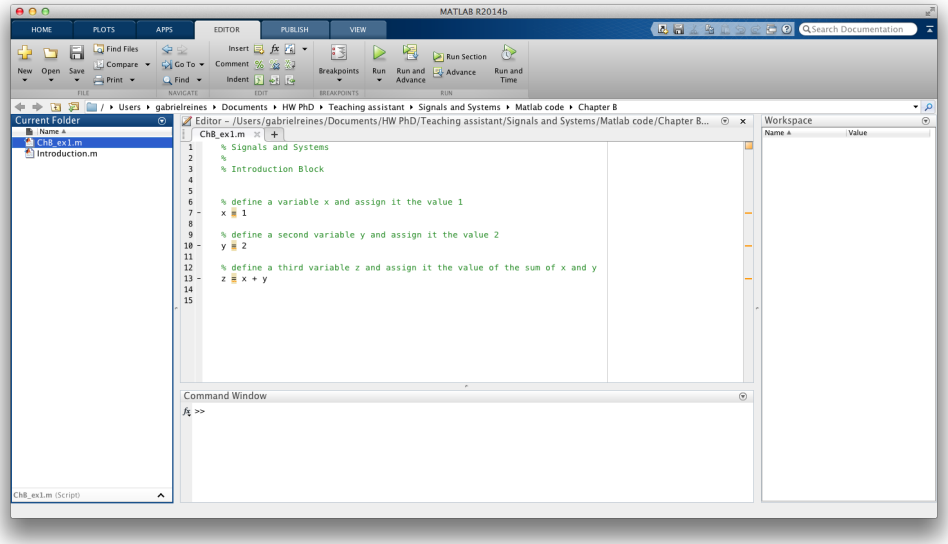
Additional information is available online: http://uk.mathworks.com/support/learn-with-matlab-tutorials.html

1 General layout

The default general layout of the MATLAB environment includes several windows. The **Editor** is in the centre of the screen. This is where you can write a *script*, a set of commands or instructions intended to be executed in a sequential way (i.e. one after the other). Just below it there is the **Command Window**, where you can write instructions and execute them. On the left side of the screen there is the **Current Folder** block, which shows the current working directory and the files contained there. Finally, on the right side we can see the **Workspace**, where all created objects and variables are stored. The layout can be easily changed by dragging the windows to different positions or even undocking the windows from the main MATLAB one.

1.1 Editor

MATLAB supports a specific kind of files called m-files (e.g. my_script.m), which can contain scripts and functions. You can edit these m-files on the Editor window.

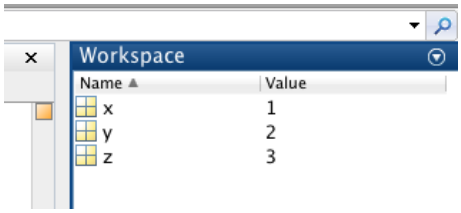


1.2 Command window

Detail of the Command Window. Three instructions have been executed. The prompt (>>) indicates that the program is ready to accept an instruction or input.

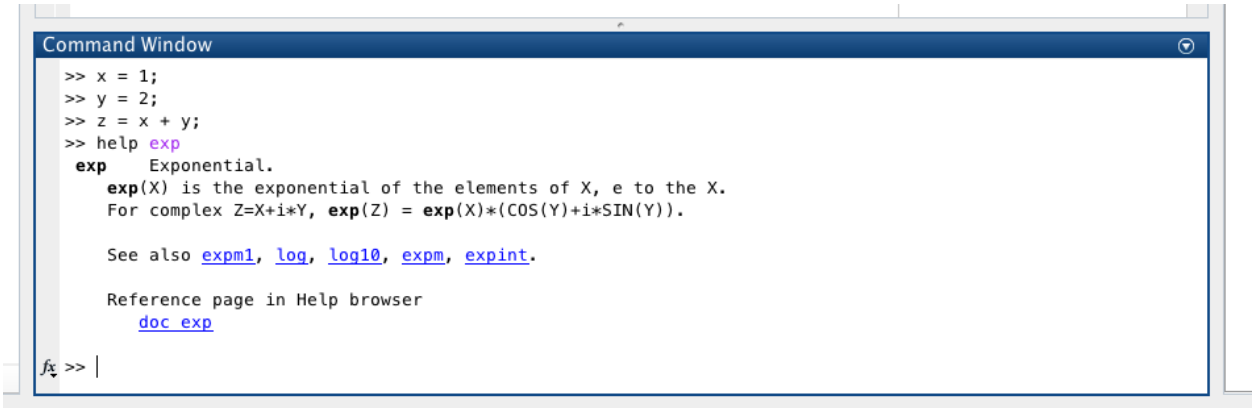
1.3 Workspace

Detail of the Workspace. Three different variables have been stored here.

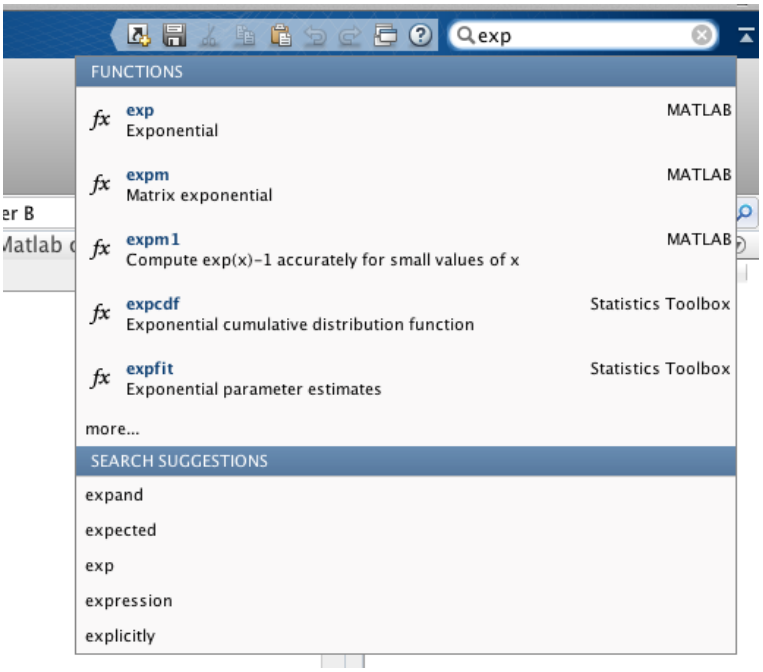


1.4 MATLAB `help` function

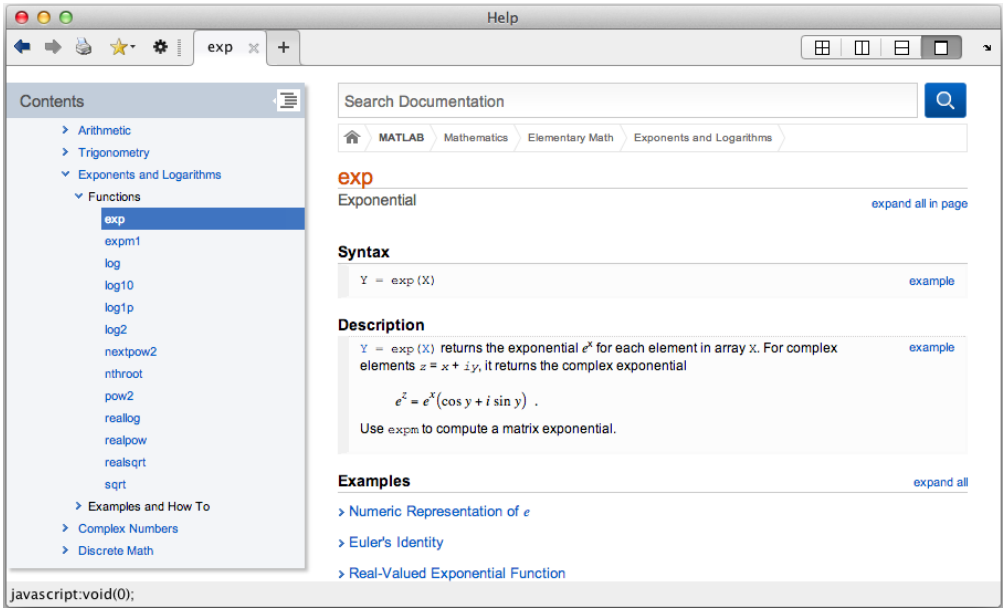
In this screenshot we can see how to get information about the MATLAB built-in exponential function `exp()` from the Command Window. By typing `help exp` the program retrieves the information about this function.



Another way to get information about MATLAB functions and syntax is using its HTML-based help browser, situated at the top-right corner of the window. If we start typing, the program suggests some results.

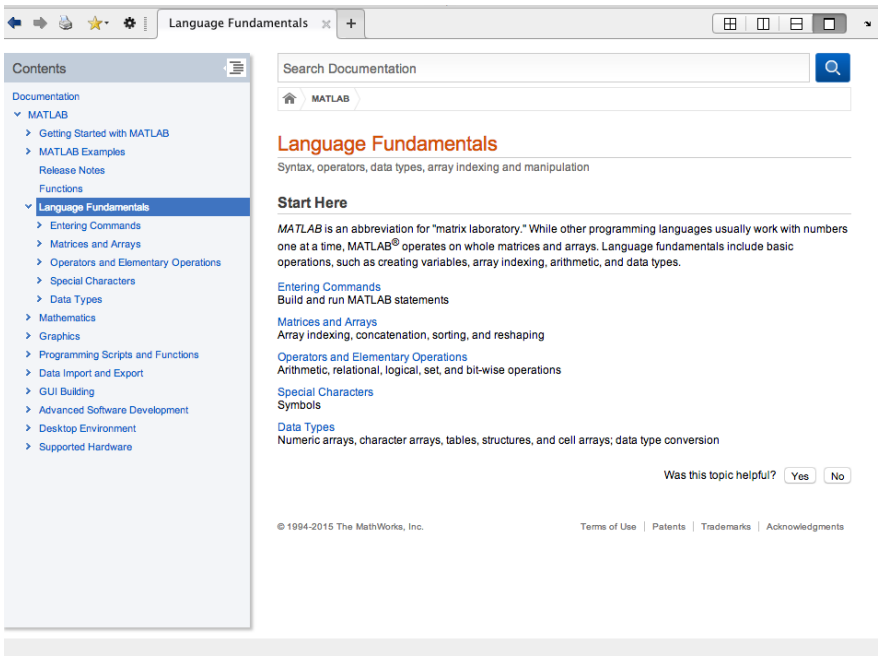


If we click on the function we are interested in, a pop-up window appears containing the help browser. This may also be achieved using the `doc exp` command.



2 Matlab syntax and commands

MATLAB has been developed as a scientific computing tool specifically aimed at simplifying code writing and provides large library of predefined functions and toolboxes. Please search the documentation and open the `Language Fundamentals` page which includes all the basic details for working with MATLAB. You should always read the documentation when dealing with unknown functions.



The following examples will guide you to understand the basic syntax and functionality. Please follow them and try the commands in MATLAB. Note that using copy-paste might produce some unrecognised characters (typically for the character `'` used when defining strings).

2.1 Example 1 - Defining variables

Scalars

We begin by defining the variables $x = 1$, $y = 3$ directly in the command line: first use `x=1` and then `y=3;`. Notice that the semicolon `;` suppresses the output of the variable on the screen. You may use this behaviour to debug your code when you write more complex scripts.

Vectors

MATLAB is specifically designed to simplify the coding required for using vectors and matrices. You may define a row vector v_1 containing the elements $\{1, 2, 3\}$ using the following syntax: `v1=[1 2 3]`. A similar way is used to define a column vector v_2: `v2=[1; 2; 3]`.

Matrices

A matrix is a two dimensional variable. The syntax for defining one is similar to the one used when defining the vectors: `M=[1 0 0; 0 1 0; 0 0 1]`. This creates a 3×3 identity matrix. The same can be achieved using the built-in function `M=eye(3,3);`.

Strings

Strings use the special character `'` as a delimiter. To define a string s try `s = 'this is a string'`. Note that if you copy-paste this you will receive an error due to the character `'`.

Other data types

MATLAB also supports more advanced data types, for example structures and cell arrays. You can always find information about them in the documentation.

2.2 Example 2 - Basic operations

Basic operations

The operators in MATLAB use a simple syntax. For example, multiply the previously defined x by the scalar 5 and store it into variable z: `z = 5*x;`. To just display the results you can use `5*x` without adding `;`. You may use such approaches to display any variables from workspace as well. Try with the matrix, `M` or the newly define variable `z`. To repeat a similar operation for the vector case you may use the same syntax: `5*v1`. Now try to compute z^y: `z^y`.

Component wise operations

The same operation on the vector will not work since it is not well defined. The command `v1^y` will produce an error. MATLAB allows for scalar wise operation by using the operators with a `.` in front. Try `v1.^y`.

Matrix and vector operations

Operating with matrices and vectors is straightforward. You have to make sure that all the variables have the proper dimension however. For example, to compute the scalar product $v_1 v_2$ you just use the normal product operator: `v1*v2`. This will produce a scalar output.

The outer product $v_2 v_1$ is computed similarly: `v2*v1`. This will output a 3×3 matrix. Matrix vector products are similar, for the product Mv_2 you can use `M*v2`. Improper dimensions for the operands will produce an error, for example the product $Mv1$ is not possible and `M*v1` gives an error. You may use the transpose operator `.'` to transform the vector $v1$ to be a column vector. Try `M*v1.'`. If your vector is complex valued, you can perform the conjugate transpose using the operator `'`.

Selecting elements from a vector or matrix requires the specification of the element's location. For example, the second element of v_1 is selected using `v1(2)`. Note that in MATLAB the indexing starts from 1. Selecting a matrix element is done similarly by indicating the row and then the column: `M(1, 2)`.

The : operator

One of the most used operators in MATLAB is `:`. It can create vectors, subscript arrays, and specify `for` iterations by creating regularly spaced vectors. To create a vector t_1 with integer elements from 1 to 5 you can use the syntax: `t1 = 1:5`. It offers the flexibility of choosing the size of the space between elements. For example, `t2 = 0:1.5:6` creates a vector containing $\{0, 1.5, 3, 4.5, 6\}$. You can also use negative spacing `t3 = 6:-1.5:0` to create the reverse order for the elements.

You can also use the operator to subindex matrices and vectors. To select the first three elements of vector t_2 you can use, for example, `t2(1:3)`.

2.3 Example 3 - Displaying variables, creating figures and other useful commands

Displaying variables

As presented for the previous examples you may display a variable by omitting the `;` operator. A similar output is achieved using the `display` command. Try `display(M)`. It is advisable to be careful with deploying the variables when writing longer scripts to ensure that you do not pollute the command window with irrelevant information. MATLAB also supports C-like string formatting using the `sprintf` function. To display nicely your script output you should use it. Run `help sprintf` to see the details. For example try
`display(sprintf('The integer variable x equals %d. In float format that is %f', x, x))`.
Note that if you copy-paste this you will receive an error due to the character `'`.

Creating figures

The basic figure in MATLAB is created using the `plot` function. Run `help plot` for more info. Now lets create an uniformly random column vector r containing 11 elements. This is achieved using the built-in function `rand`: `r=rand(11, 1)`. To create a figure and display the vector try `plot(r)`. Notice that the y-axis is labelled from 1 to 10. For example, if our random vector r corresponds to an output of some random process measured every 0.1 seconds, you can include the information of the y-axis: `plot(0:0.1:1, r)`

Useful commands

Now, delete all variables from the workspace, close all figures created and clear the command window. Use the following commands: clear screen: `clc`; delete a workspace variable, for example the matrix M: `clear M`; delete all workspace variables: `clear all`; close all figures: `close all`;

2.4 Exercise 1

Create a script file and write a script that does the following:
Create an uniformly random column vector of size 20 with values between -2 and 5. Plot first 10 elements with red colour and the last 10 elements with blue colour, and all elements with an even index, i.e. $2, 4, ...$, with black, all on the same figure. Create a legend for the figure.

Hints: read the help for legend to see how to create a legend: `help legend`; read the help for plot to see how to specify colours: `help plot`; use `hold on` to force the plots to be on the same figure: `help hold`;