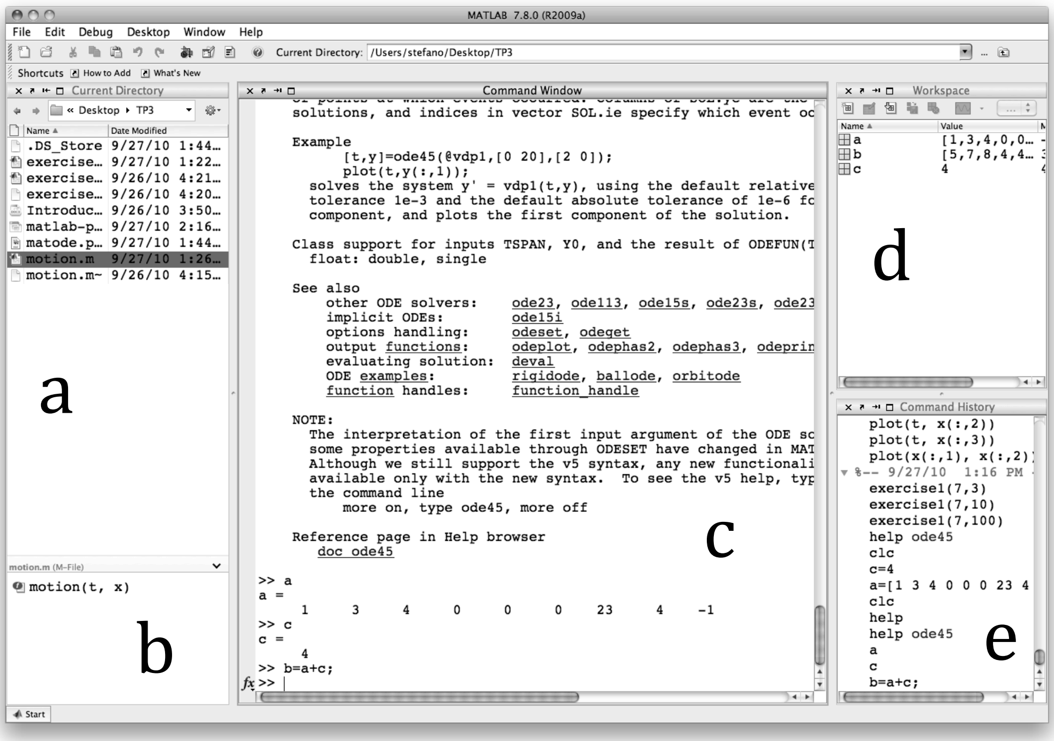
Step by step MATLAB tutorial with exercises

Stefano Andreozzi, Julien Racle, Ljubisa Miskovic
(v. 10.10.2011)

This tutorial is a short introduction to `MATLAB`, explaining the basic concepts needed. It is not intended to give a complete view on the broad capacities of `MATLAB`. Do not forget to use the very well documented help.



```
Name                Date Modified
.DS_Store           9/27/10 1:44…
exercise…           9/27/10 1:22…
exercise…           9/26/10 4:21…
exercise…           9/26/10 4:20…
Introduc…           9/26/10 3:50…
matlab-p…           9/27/10 2:16…
matode.p…           9/27/10 1:44…
motion.m            9/27/10 1:26…
motion.m~           9/26/10 4:15…
```

```
motion(t, x)
```

```
    solutions, and indices in vector SOL.ie specify which event oc

    Example
          [t,y]=ode45(@vdp1,[0 20],[2 0]);
          plot(t,y(:,1));
      solves the system y' = vdp1(t,y), using the default relative
      tolerance 1e-3 and the default absolute tolerance of 1e-6 fc
      component, and plots the first component of the solution.

    Class support for inputs TSPAN, Y0, and the result of ODEFUN(T
      float: double, single

    See also
        other ODE solvers:    ode23, ode113, ode15s, ode23s, ode23
        implicit ODEs:        ode15i
        options handling:     odeset, odeget
        output functions:     odeplot, odephas2, odephas3, odeprin
        evaluating solution:  deval
        ODE examples:         rigidode, ballode, orbitode
        function handles:     function_handle

    NOTE:
      The interpretation of the first input argument of the ODE sc
      some properties available through ODESET have changed in MAT
      Although we still support the v5 syntax, any new functionali
      available only with the new syntax.  To see the v5 help, typ
      the command line
          more on, type ode45, more off

    Reference page in Help browser
       doc ode45

>> a
a =
     1     3     4     0     0     0    23     4    -1
>> c
c =
     4
>> b=a+c;
>>
```

```
Name      Value
a         [1,3,4,0,0…
b         [5,7,8,4,4…
c         4
```

```
   plot(t, x(:,2))
   plot(t, x(:,3))
   plot(x(:,1), x(:,2)
%-- 9/27/10  1:16 PM
   exercise1(7,3)
   exercise1(7,10)
   exercise1(7,100)
   help ode45
   clc
   c=4
   a=[1 3 4 0 0 0 23 4
   clc
   help
   help ode45
   a
   c
   b=a+c;
```

The `MATLAB` main window is composed by

a) **Top left**: the current directory panel lists all the files in the current folder; the latter can be changed browsing the disk structure visualized at the top of the panel.
b) **Bottom left:** details on the selected file
c) **Center:** command window that allows entering the commands and seeing the output from the programs. The prompt is written as >>.
d) **Top right**: the workspace lists the variables currently defined, as well as their content. Double click on a variable to see it in more detail.
e) **Bottom right**: the command history shows the ordered list of the commands previously used.

In this part we give some instructions you should follow step by step in order to develop the sufficient skills needed for the exercises.

1) Define and initialize variables, writing each line on the MATLAB command prompt, followed by ENTER:

```
>> a = 5
>> b = 2
>> c = a + b
```

Note that in MATLAB we directly initialize the variables giving their content in this way. There is no need, in contrary to some other programming language, to tell that, for instance, a is of the type of integers or doubles, etc. MATLAB will directly define the proper type for the variable, accordingly with its content.

Names of variables, commands and functions in MATLAB are made of letters, numbers and underscores, **starting always with a letter or underscore**. Note also that these names are **cAsE sEnSiTiVe**.

2) The up and down arrows allows to retrieve previously typed commands. Use them to modify the value of b to 3. Note that it is also possible to start typing the initial part of a previous command and then use the arrows to scroll only through the commands starting with the typed part.

3) Typing an operation with an ending semicolon will result in hiding the output of the given command from the command window. Try:

```
>> a = 3;
```

With the semicolon it is possible to write also many commands in the same line. Pressing ENTER at the end, they will be processed sequentially.

4) If you want to put many commands in the same line, but displaying the result of each of them, use the comma instead of the semicolon:

```
>> a, c
```

Note that c has a value of 7, because its value was computed when a was 5 and b was 2. After that c is not anymore linked to the values of a and b (there is no symbolic relationship).

5) If the result of the last successful operation (if any) was not assigned to a variable, it is then stored in the variable ans.

```
>> 2 + 3 * 5
>> a = 2 * ans
```

Note that the first operation follows the mathematical hierarchical rules: the product is performed before the addition. To use these operations in a different order, place round brackets properly – e. g. `( 2 + 3) * 5`

6) In order to see which variables have been defined, use the command `who`, or `whos` (`whos` gives additional information on the size and types of the variables).
Once you quit `MATLAB`, all variables are lost. If you want to keep them for a later time, use the `save` command, followed by the name of the **"mat-file"** where to save and the name of the variables to save (or no variable name to save all variables present). You can then load again your data by using the `load` command followed by the name of the mat-file (the variables are saved in a `MATLAB` specific format **".mat"**).
In case you want to delete a variable use the `clear` command, followed by the name of the variable you want to erase. Delete the `b` variable. Note that `clear` alone will erase all the variables in the workspace (it works like `clear all`). Save your variables, delete them all and load them again.

7) In `MATLAB` it is also very easy to define row and column vectors. We just need to put values between squared brackets, separated by commas, in case of row vectors, or semicolons, in case of column vectors. Note that commas can be omitted (use a simple space instead). Use this to create:

$$v = \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix}, \quad w = \begin{pmatrix} 5 \\ 6 \\ 7 \\ 8 \end{pmatrix}.$$

8) A vector of equally spaced values can be created this way:

```
>> s = 2:10
```

(as only the extremes are given, the step between them is implicitly 1)

```
>> t = 1 : 0.1 : 3
```

(vector between 1 and 3 with elements spaced by 0.1)

9) Try to sum `s` and `t`. This sum is not consistent because `s` has 9 elements and `t` has 21 elements. This does not return a value, but an error message. If an erroneous operation is included in the flow of an `MATLAB` program, it will terminate the execution of it. Try then to construct two vectors of the same dimension and sum them.

10) Try to do `v*w` and `w*v`. What do they give?

11) As you have seen in the previous point, `w*v` results in a 4x4 matrix. In order to build a matrix directly, you simply have to use the square brackets with each element of a row separated by space or comma and each row by a semicolon. For example, the matrix:

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

has to be entered in the command windows as:

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

12) To obtain a transpose of a matrix just write an apostrophe after the matrix name. Find out how to create the matrix `B` as the double of matrix `A`. Then create `C` as the transpose of `B`. Having the three matrices `A`, `B` and `C`, perform the following operations:

```
>> A + B
>> A * C
>> A * B
```

The last operation will give you an error. Why?

13) If instead of doing a matrix multiplication (or division or power) you want `MATLAB` to work element by element, simply add a point before the operator (this kind of element-wise operation in `MATLAB` is very useful):

```
>> A .* B
```

14) Elements of a matrix can be extracted or assigned using the round brackets:

Element at row 2, column 1 of `A`

```
>> A(2,1)
>> A(2,1) = -2
```

If you provide only one index, the resulting element will be the one at the position given counting from top to bottom, starting from the first column to the last, as if the columns of the matrix would be placed one after the other in a unique column vector.

```
>> B(4)
```

Using the colon you can refer to more than one element of the matrix at a time. If you use the colon alone, it means "all the row" or "all the column". The keyword "end" means "to the last line or column".

```
>> A( 1 , 2 : 3 )
>> A( 1 , : )
>> C( 2 : end , : )
>> A( : , 2 ) = 0
```

In the latter command, the scalar value 0 is assigned to each element of the second column.

```
>> A( 100, 1)
```

What is the error?

15) There exist some ready-made matrices you can use:

```
>> ones(2, 5)
>> zeros(3, 2)
>> eye(3)
```

16) Till now we have entered all our commands directly in the command window, but it can be useful to store our commands in some external files that can be used again some later time. For this we use "**m-files**" that can contain scripts or functions.
In order to know which files to read, `MATLAB` will first look if it finds the appropriate file in the current directory (use `pwd` to see in which directory you currently are; you can use the same syntax as in Unix systems to navigate through directories, with commands like `cd myFolder`, `ls`, ...).
If it does not find the file in the current directory, it will look in the folder present in the saved *path*. You can use the command `path` to see all folders in which `MATLAB` is looking (many are folders with the native functions from `MATLAB`). If you know you will always use some *m-files* present in a given folder, you can add this folder to the search path with the command

```
>> addpath('/path_to_the_folder/myFolder');
>> savepath
```

The first command adds `myFolder` to the search path (take care, you need to give the absolute path to `myFolder`) and the second command saves the search path (in the default file) to use it again in the next `MATLAB` sessions.

17) Till now we have entered all our commands directly in the command window, but it can be useful to store our commands in some external files that can be used again some later time. For this we use "**m-files**" that can contain scripts or functions. Open now a new m-file named `tutorial.m` (there are several ways to do it; for now just type `edit tutorial.m` in the command window). The editor window, showing the content of the file, will appear. Type in the following lines:

```
x = 0 : 0.1 : 10 ;
y = 1 ./ (1 + x) ;
plot(x,y)
```

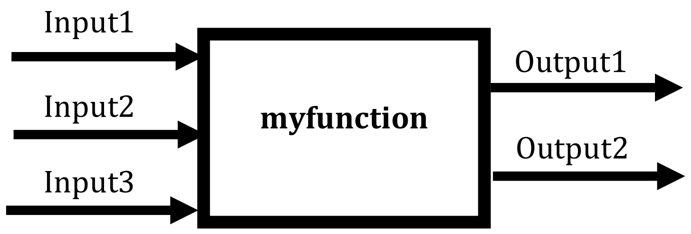
Save it and launch it (type `tutorial` in the command window, i.e. the same name as the "m-file" but not writing the extension ".m"). Now, back to the editor window, modify the file, replacing the expression for `y` with the following structure (initialization; `for` loop and some comments, marked with `%;` note that each comment line has to start with the symbol `% or #`):

```
i = 1 ;
kmin = 1;
kmax = 5;
for k = kmin : kmax %evaluation of y for diff
                    %k values
       y(i,:) = k ./ (1 + x);
       i = i+1 ;
end
```

For a good readability of the code, indent the lines inside your scripts, and always add lots of comments.

Save and launch the script.

18) *m-files* can also contain functions instead of scripts. A generic function can be seen as:



In order to define this function, **you have to create an *m-file* having the same name as the function.**
The function must be defined on the top of the m-file in the following way:

```
function [Output1, Output2]= myfunction ...
  (Input1, Input2, Input3)

  % any comment before the body will be used as
  % content of the help associated with the new
  % function (try help myfunction)

       body of the function;
end
```

Note that the three dots `...` are used when you want to format a single command to be written in more lines, for the sake of readability of the code.

The function can then be called in another *m-file* script or function in the following way:

```
[a, b] = myfunction (c, d, e);
```

Make sure that inputs have the same number, type and, eventually, size, of the ones declared in the function *m-file*.

Modify the script of point 17 to make it a function taking `kmin` and `kmax` as input arguments and outputting `x` and `y`.

Try to call your function more than once, with different inputs. If you just call the function without assigning the output to variables - e. g. `myfunction(1, -2, 3);` - it will compute the operations in the body of the function without returning the outputs (the first one will be available in `ans`).

19) Create a function that takes the amplitude and period as inputs and plots sine and cosine with that amplitude and period.

Some suggestions:

- A function without any output argument is defined this way:
  `function myfun(Input1, Input2)`
- Use the keyword `hold on` before the second plot command in order to keep the content of the figure while drawing a new one; use `hold off` to disable this feature. The keyword `hold all` (instead of `hold on`) will also change the color of each plot within the same figure window (this is only working in `MATLAB`). In order to create a figure in a new window, use the keyword `figure` before plotting the second figure.
- For different colors and line styles check `help plot`. Try:

```
plot(x, y, 'c--');
```

- Other elements of the figure (such as variable range, x and y labels, title of the figure, legend) can be added with the following commands:

```
xlim([0 6]);
xlabel('x');
ylabel('sine, cosine');
title('Figure of the tutorial');
legend('sine',  'cosine');
```

20) Use a `while` loop to find all powers of 2 below 10000. How many powers of 2 are smaller than 10000? The format of `while` is as follows:

```
while condition
  commands;
  …
end
```

21) Use a `for` loop to find the sum of the odd numbers between 99 and 577. Repeat the same task but omit in the sum the numbers having the last digit 3 (use the command `mod`). The format of `for` is as follows:

```
for variable=start:step:end
  commands;
  …
end
```

The format of `if-elseif-else` statement is as follows:

```
if condition
  commands;
elseif condition
  commands;
else
  commands;
end
```

22) Solve the equation `sin x=e^x -5` using `fzero` (syntax: type `help fzero`). Verify the result by plotting the function `f(x)=0`. Can this equation be solved with `fsolve`. What is the difference between `fsolve` and `fzero`?

23) Generate 10 values of a cosine curve `y=cos(x)` between 0 and 2pi. Use these 10 points and MATLAB's command `interp1` to find the values of the underlying function `y` at 200 equally spaced points within the same interval. Refer to MATLAB's help to understand the usage of this command. (Remark: to create a vector with equally spaced points, the command `linspace` can be helpful).

24) Use MATLAB's function `polyfit` to find the coefficients of polynomials

a) $y=ax+b$

b) $y=ax^2+bx+c$

c) $y=ax^3+bx^2+cx+d$

d) $y=ax^4+bx^3+cx^2+dx+e$

that approximate the vector of 10 cosine values given in 23). Plot the corresponding approximating curves using `polyval` function. Refer to `help polyfit` and `help polyval` for the usage.

25) Use MATLAB's function `quad` to show that the area of a circle with unitary radius is pi. Use `trapz` to compute the underlying area of the 10-points dataset sample given in 23).