# Module 2: MATLAB Tutorial Part 2

*© 2019 Christoph Studer (studer@cornell.edu); Version 0.2*

The main goals of this module are (i) to learn how to evaluate and plot functions and (ii) how to write MATLAB scripts, which simplifies automation of MATLAB commands. As before, it will be helpful to not only perform the main activities, but also to try out all of the examples we show during our explanations. ***Remember: Whenever you are stuck or are interested in learning more details about a specific aspect, please ask us—we are here to help!!***

## 2 Evaluating and Plotting Simple Functions

MATLAB is particularly good at quickly evaluating functions and plotting the results. We will now learn the basics of how to compute functions and how to plot these functions in MATLAB. We will make extensive use of function evaluation and plotting throughout this project.

### 2.1 Evaluating Functions

Since MATLAB is basically an overpowered calculator, it can also *numerically* evaluate arbitrary functions. Assume, for example, that you want to evaluate the quadratic the function $f(x) = x^2$ for values in the interval $x \in [-1, +1]$. Since it is not possible to store and manipulate continuous quantities in computers, you have to first discretize the interval in MATLAB, by generating a vector that contains a discrete set of values in this interval. Generate a vector for a few values in the interval $[-1, +1]$. This can be done by using the `linspace` command. For example, define

```
x = linspace(-1,+1,5)
```

which generates a row vector with 5 equally-spaced values in the range $-1$ to $+1$ as follows:

```
>> x = linspace(-1,+1,5)

x =

   -1.0000   -0.5000         0    0.5000    1.0000
```

We will now evaluate the quadratic function $f(x) = x^2$ on these values. To do so, type

```
fx = x.^2
```

which generates a vector called `fx` that contains the function values of the quadratic function evaluated at the values contained in the vector `x`:

```
>> fx = x.^2

fx =

    1.0000    0.2500         0    0.2500    1.0000
```

Note that the variable name `fx` was arbitrary. As you may have observed, instead of `x^2`, we used the MATLAB command `x.^2`, where we use an additional dot before the exponent (or power) operator "^". This additional dot implies that we are applying the exponent operator on every entry of the vector `x` separately. If you try to evaluate `x^2`, then MATLAB throws an error and actually informs you to "Use POWER `(.^)` for elementwise power."

**Activity 6: Evaluate a general quadratic function**

You are probably familiar with general quadratic functions of the form $f(x) = ax^2 + bx + c$ where the parameters $a$, $b$, and $c$ determine the shape of the function. Imagine you want to evaluate the function for the following parameters $a = 1$, $b = -1$, and $c = 2$. First, define three variables `a`, `b`, and `c` with the values as given above. Then, generate the base points $x = \{-2, -1, 0, +1, +2\}$ using either the `linspace` command or the "colon" operator (both work equally well in this case). Finally, evaluate the function

```
fx = a*x.^2+b*x+c
```

at the base points $x = \{-2, -1, 0, +1, +2\}$. You should obtain the function values $f(-2) = 8$, $f(-1) = 4$, $f(0) = 2$, $f(+1) = 2$, and $f(-2) = 4$. You can easily verify that these values are correct for the given parameters. Finally, type

```
plot(x,fx)
```

and hit enter. As it turns out, a new window opened and you have just plotted the function $f(x)$ at the base-points $x = \{-2, -1, 0, +1, +2\}$. Unfortunately, five base-points are not enough to see that this is indeed a quadratic function. Increase the resolution of the base-points from 5 points between $-2$ and $+2$ to 50 points, re-evaluate the function again (by typing `fx=a*x.^2+b*x+c`), and plot the function again by executing the plot command: `plot(x,fx)`. You should see a smooth curve that represents your quadratic function with a clear minimum at $x = 0.5$.

## 2.2 Function Plotting

In the above activity, you have already plotted your first function. We will frequently use the plot function to inspect the signals we want to transmit over the acoustic channel. There are various commands that you can use to plot data in MATLAB. You can plot multiple curves with different line-styles and colors, and add legends, grids, labels; see the example below that generates a neat-looking 2D plot.

```
% generate sine and cosine functions
x = -4*pi:0.1:4*pi;
y = sin(x);
z = cos(x);

% generate a nice plot
```

```
h = figure(1)
plot(x,y,'b-','LineWidth',3)
hold on
plot(x,z,'r:','LineWidth',3)
hold off
xlabel('x axis','FontSize',14)
ylabel('y axis','FontSize',14)
legend('sine','cosine')
grid on
set(gca,'FontSize',14)
axis([min(x),max(x),-1.1,+1.1])
```

By running the instructions as in the code above, you will produce the plot shown in Figure 2. The first line defines a MATLAB comment, which starts with `%`; everything on that line that follows this command will not be executed. The next three lines define the sine and cosine functions in the interval $[-4\pi, +4\pi]$. Note that the semicolon at the end of these three lines suppresses MATLAB from displaying any output. This can be particularly useful if you define very long vectors. The remaining lines are used to plot the two functions. The details are explained as follows:

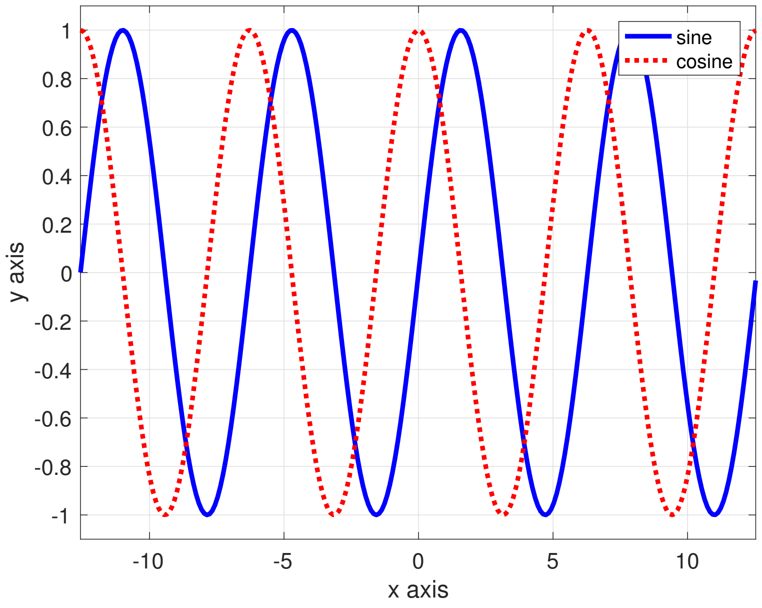


Figure 2: A nice plot generated in MATLAB.

- The command `h = figure(1)` generates a new figure with number 1. You can use other numbers to have multiple plots open at the same time. The variable `h` contains the function "handle" of your figure and is only required if you want to save your plot.

- The command `plot(x,y,'b-','LineWidth',3)` plots the sine function at base points `x`. The command `'b-'` draws a blue solid line. The two commands `'LineWidth',3` increase the line thickness.

- The command `hold on` makes sure that you can overlay more lines.

- The command `plot(x,z,'r:','LineWidth',3)` plots the cosine function at base points `x`. The command `'r:'` draws a red dotted line.
- The command `hold off` makes sure that you can no longer overlay more lines. If you would plot another function, you would now overwrite all functions drawn so far.
- The command `xlabel('x axis','FontSize',14)` labels the x-axis of the plot with "x axis." The two commands `'FontSize',14` increase the font size.
- The command `ylabel('y axis','FontSize',14)` labels the y-axis of the plot with "y axis." The two commands `'FontSize',14` increase the font size.
- The command `legend('sine','cosine')` creates a legend that labels the first and second curve with "sine" and "cosine," respectively.
- The command `grid on` turns on vertical and horizontal grid lines.
- The command `set(gca,'FontSize',14)` increases the font size of the x-axis and y-axis numbers.
- The command `axis([min(x),max(x),-1.1,+1.1])` defines the range in which the plot is generated. The first two arguments `min(x)` and `max(x)` make sure that the x-axis is plotted from $-4\pi$ to $+4\pi$. The second two arguments make sure that the y-axis is plotted in the range from $-1.1$ to $+1.1$.

Note that this is a quite complicated plot example and it is often sufficient to just type

```
h = figure(1)
plot(x,y)
hold on
plot(x,z)
hold off
```

especially if you know what you are plotting. However, if you need your plot for a presentation or, in general, if you show your plot to someone else (someone who is not familiar with the functions you are plotting), then it is always a good idea to make a clean plot as the elaborate example above.

If you would like to add a title to your figure, then simply use the following command after plotting

```
title('this is a figure title')
```

If you want to save a copy of your figure to your hard-drive, then type

```
print(h,'-loose','-depsc','testplot.eps')
```

where "testplot" is the filename and "eps" is the file type. There are other file types that you can save. Simply use the help function for the `print` command to obtain more details. Another approach to save your figure is to click "File" in the figure window and "Save As...". You can now select the file name as well as the format (common file formats are `eps` and `png`; the latter can be imported easily in PowerPoint).

**Activity 7: Create a nice plot of a function of your choice**

Use the above example to generate a nice plot of a function of your choice. It's up to you what you would like to plot.

It is now—once again—time to take a break. Function evaluation and plotting can be quite tedious, but is actually one of the key strengths of MATLAB. Do not worry if you forgot some of the details. Type the commands `image`, `penny`, and `spy` to discover some of MATLAB's easter eggs.

## 3 Scripts and Functions in MATLAB

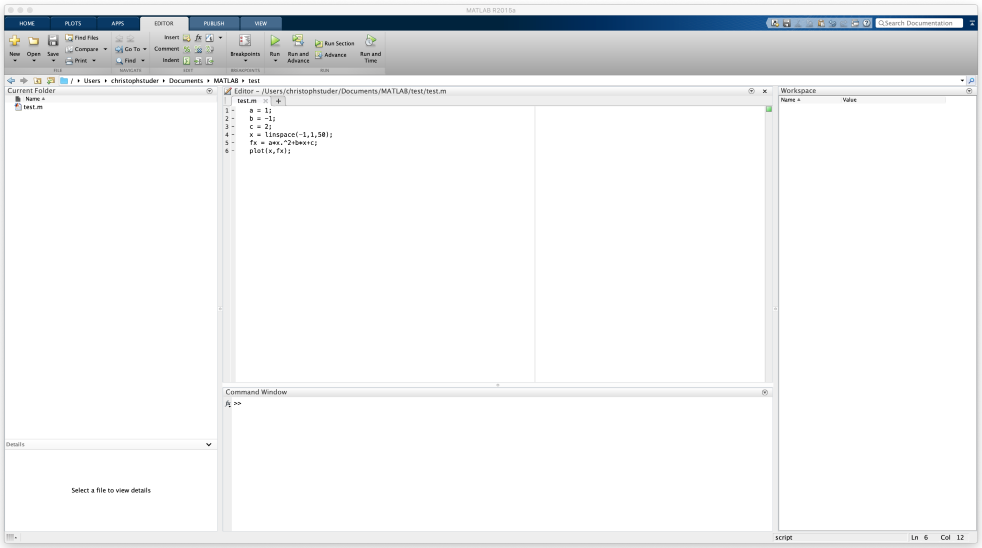
In MATLAB, you often want to repeat the same set of commands, without manually typing them into the Command Window. In what follows, you will learn how to write MATLAB scripts, which allow you to execute a set of commands by the push of a button. You will also learn how to write your own MATLAB functions, such as the `exp` or `sin` functions.

### 3.1 MATLAB Scripts

A MATLAB script is a file that contains a set of MATLAB commands. You can then execute this file and it will automatically run the script line-by-line, as if you would type the commands manually in the Command Window. To create a new MATLAB script, press the "+ New" icon on the top left of the MATLAB desktop and select "Script." This command opens a new script right above the Command Window (assuming that you have the default view). Save the script by pressing the "Save" icon under the name "test.m". In the Current Folder window (on the left side of the desktop), you should now see a file called "test.m". Write the following commands in the Editor and save the script again:

```
a = 1;
b = -1;
c = 2;
x = linspace(-1,1,50);
fx = a*x.^2+b*x+c;
plot(x,fx);
```

Your desktop should now look similar to that in Figure 3. You can now execute the script by pressing the "Run" icon at the top. This will execute the script "test.m" line-by-line, which includes (i) setting up the parameters, (ii) generating an array of values in the interval $[-1, +1]$, (iii) evaluating the quadratic function, and (iv) plotting a figure of the quadratic function. It is now very easy to change one or multiple parameters (for example, set `a=-1;`) and to re-run the script by pressing the "Run" icon at the top. Instead of manually typing one command after each other, you can now quickly change the parameters and re-generate the plot.



```
a = 1;
b = -1;
c = 2;
x = linspace(-1,1,50);
fx = a*x.^2+b*x+c;
plot(x,fx);
```

```
>>
```

Figure 3: MATLAB desktop with the Editor containing a simple script.

There exists another way to execute this MATLAB script. Go to the Command Window and type `test`, without the suffix ".m". Press enter. This runs the script line-by-line. Note that you simply have to type the name of the script to run it. Note that this also means that you should not call your script by a name of an already existing MATLAB function (e.g., calling a function "exp.m" would cause issues with the built-in `exp` function). Another common mistake is to call a variable the same name as an existing function. For example, never define a variable called `sum` as there is a MATLAB `sum` command! Also be careful that the function you are trying to call must be visible in the Current Folder window (on the left side). If you are in another folder, then MATLAB cannot find the script and will throw an error. This is a simple mistake that happens all the time—even to us.

**Activity 8: Create a new MATLAB script**

Create a new MATLAB script called "test_plot.m" that plots the reciprocal function $f(x) = 1/x$ in the interval $[-5, +5]$. Note that in order to evaluate a function on a vector of base-points you have to use the dot-notation as follows:

```
fx = 1./x
```

Execute your script. You can then add other terms to your function to plot whatever you want.

### 3.2 MATLAB Functions

One major disadvantage of MATLAB scripts is the fact that you cannot run the same script for different parameters, unless you manually edit the script and run it again. Note that if you run, for example, the natural logarithm function with the argument 5 by typing `log(5)` you can easily run the same function with another argument (e.g., `log(10)`). Clearly, you do not have to edit the logarithm function. To create functions that allow one to pass one or multiple parameters, we have to generate a MATLAB function.

Generating MATLAB functions is as easy as generating MATLAB scripts. Press the "Home" tab on the top left of the MATLAB desktop and then, the "+ New" icon. By selecting "Function," MATLAB creates an untitled function with input arguments and output arguments. For example, the file in the Editor may contain the following contents:

```
function [ output_args ] = untitled5( input_args )
%UNTITLED5 Summary of this function goes here
%   Detailed explanation goes here
end
```

Save the function under the name "test_function.m" and change the function name "`untitled5`" to "`test_function`". It is important that the function name is identical to the name of the MATLAB file (without the suffix ".m"). You can also right-click on "untitled5" and select "Replace function name by file name," which automatically sets the function name to the file name.

Let us now generate a simple test function. Modify the file in the Editor to contain the following contents and save the file:

```
function [ y ] = test_function( x )
%test_function that computes the square of the input x
%   y = test_function(x)
%   x : input
%   y : output (y=x^2)
```

```
y = x.^2;
end
```

This test function has one input argument `x` and generates one output `y`. The function simply computes the square of the input argument. For example, if you type

```
test_function(2)
```

in the Command Window, then you see that the function output is, as expected:

```
>> test_function(2)

ans =

     4
```

Also, if you type `help test_function`, then you can see the comments (which start with %) right below the function declaration. Note that you have to enter this information for your own function.

Functions may contain multiple lines (similar to scripts) and one can pass one or multiple arguments (which is different to scripts); passing values to functions is particularly useful (as you will see later). To generate a function with multiple input arguments and multiple outputs, modify your test_function as follows and save the file:

```
function [ z,w ] = test_function( x,y )
%test_function that computes x+y and x*y
%   [z,w] = test_function(x,y)
%   x : input1
%   y : input2
%   z : output1 (z=x+y)
%   w : output2 (w=x*y)
z = x+y;
w = x.*y;
end
```

If you now type in the Command Window

```
[z,w] = test_function(1,2)
```

then MATLAB generates the following two outputs

```
>> [z,w] = test_function(1,2)

z =

     3


w =

     2
```

and directly assigns the outputs to the variables `z` and `w`.

**Activity 9: Create a new MATLAB function**

Create a new MATLAB function called ‘`test_recip.m`” that takes $x$ as an argument and computes the reciprocal function $f(x) = 1/x$ as output. Execute the function and check whether the result makes sense. You can then modify your function to compute whatever you want.

### 3.3 Clearing the Workspace

If you want to clear all variables in the Workspace, then simply type `clear` and press enter. If you want to remove everything (including functions etc.), then you can also type `clear all`. If you want to remove a specific variable, for example `annoying_var`, then you can type `clear annoying_var`. If you are annoyed by the screen being full with past commands you typed, then type `clc`, which clears the Command Window. If you still have a lot of figures open, then you can type `close all`, which closes all open figures.

***Important: This tutorial covered most of the MATLAB commands used to evaluate and plot functions. Do not worry if you think you may forget certain commands or if you are not 100% sure yet how to use all of the learned material. We will repeat how to use the most important commands when necessary. Also, whenever you have questions about MATLAB, feel free to ask one of us—we are here to help!***