Module 1: MATLAB Tutorial Part 1

© 2019 Christoph Studer (studer@cornell.edu); Version 0.1

The main goal of this module is to learn the basics of MATLAB, which is the main software we use throughout this project. In particular, you will learn how to evaluate simple mathematical expressions. Note that 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, have questions, or are interested in learning more details about a specific aspect, please ask us—we are here to help!!***

---

1 Introduction

MATLAB is a software package that is targeted to numerical computation. Initially, MATLAB was designed for solving linear algebra-type problems using matrix-vector operations and hence, its name is derived from MATrix LABoratory. Nowadays, MATLAB is used by over 3 million users worldwide and is the de-facto standard software for scientific computing in industry and academia in a broad range of fields, including engineering, science, and economics. Unfortunately, MATLAB is not free (it is, in fact, expensive; a license ranges from $99 for student use to $2,150 for professional use) but you will have access to the newest version of MATLAB in the ACCEL laboratories.[1] One of the key advantages of MATLAB is the fact that it is relatively easy to use. The goal of today's lab is to learn the basics of MATLAB using simple examples. Please remember that we will use MATLAB throughout this project with the goal of designing an acoustic communication system. Hence, everything you will learn in this module will become very useful later—please keep this module as a reference.

1.1 How to Start MATLAB

We will use Microsoft Windows in the ACCEL labs for this research project. To open MATLAB, first have one of the Program Assistants to log into a computer and then, simply click on the START button and select MATLAB R2019a. It usually takes a few seconds for MATLAB to start (you first see a splash screen showing the MATLAB logo and the version number). Another way to open MATLAB is to press the search button and type MATLAB and start it from there. After loading MATLAB, you should see the desktop that looks like the one in Figure 1. The MATLAB desktop consists of three main panels:

- Left: "Current Folder"' – This panel is used to access and manage files, functions, and scripts.
- Center: "Command Window" – This panel is used to enter MATLAB commands at the command line, which is indicated by the prompt >>
- Right: "Workspace"' – This panel shows variables that have been defined in your workspace.

[1] Free, open-source alternatives to MATLAB exist, such as GNU Octave, but they are unfortunately not 100% compatible. If you are interested in playing around with GNU Octave, you can download it here for free: `https://www.gnu.org/software/octave/`

In case some of these panels are missing or the desktop looks differently, you can restore the default appearance by clicking on the "Layout" icon and selecting the "Default layout."

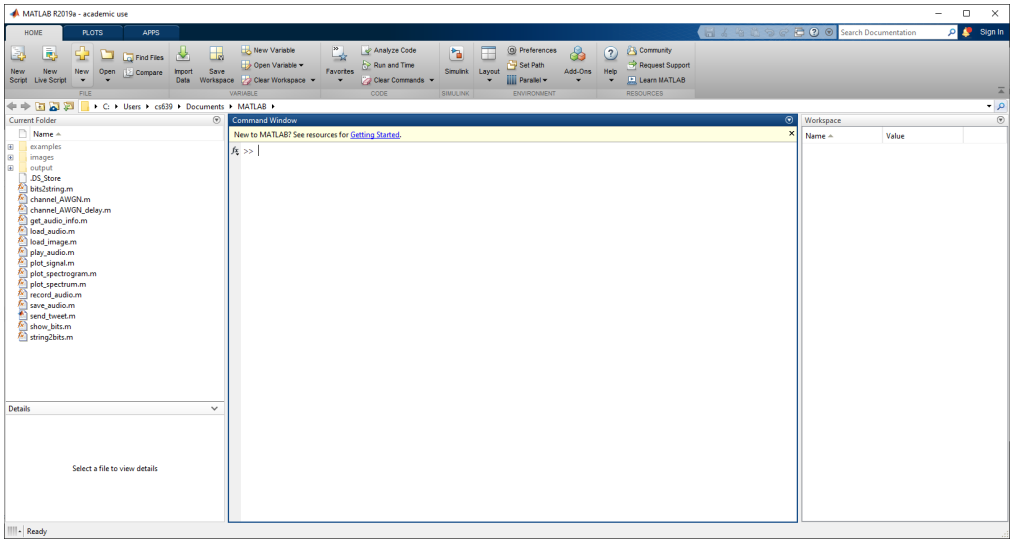

Figure 1: Default appearance of the MATLAB desktop.

1.2 First Steps in MATLAB

MATLAB is basically a very powerful numerical calculator. In the Command Window, simply type `1+2` and press enter (or return). As you can see, MATLAB immediately evaluates the expression $1 + 2 = 3$ and shows the result. The Command Window should looks as follows:

```
>> 1+2

ans =

     3
```

Here, `ans` stands for the *answer* of the expression you just evaluated. The prompt >> and the blinking cursor indicates that MATLAB is ready to accept another command. As in a calculator, you can also group operations using parentheses, e.g., you can compute `3*(1+2)` where `*` is the multiplication operator.

Activity 1: Try do do some more math!

MATLAB is extremely powerful and allows many mathematical operators you may be familiar with. For example, if you would like to compute the square root of 9, simply type `sqrt(9)` and press enter. Remember that function names are case sensitive! Other operators or functions are `3^2` if you want to square the number 3 or `exp(1)` if you want to evaluate the exponential function of 1, i.e., $e^1 \approx 2.7183$. Fractional numbers can be entered either by typing `0.3333` or directly as fractions `1/3`. Here, the operator `/` performs a division.

Note that MATLAB represents per default all numbers as double-precision floating-point numbers (please ask us if you want to know more about floating-point numbers; but the details are not important for this project), which are sufficiently accurate for our purpose. However, you should remember that MATLAB often does not show all digits, which is why `0.3333` and

`1/3` are not the same numbers.

Use MATLAB to carry out a few basic calculations; Table 1 lists some more functions you can use. For example, try to compute $\cos(2\pi)$ (the answer should be 1). If you do not remember what the exact value of π is, note that `pi` in MATLAB is a predefined constant!

Table 1: Summary of some basic MATLAB functions.

Function	Description
`abs`	Computes the absolute value
`sign`	Signum function (extracts the sign of a number)
`sqrt`	Square root
`sin`	Sine function
`cos`	Cosine function
`exp`	Exponential function
`log`	Natural logarithm
`round`	Round towards nearest integer
`max`	Finds the maximum in a vector

If you make a mistake in your mathematical expression (or MATLAB does not understand what you were trying to do), an error message will be displayed. Error messages are shown in red color and are generally very precise in pointing to the problem—sometimes MATLAB even recommends a solution (which, however, may not always be what you intended). For example, type `exn(1)` instead of `exp(1)` and MATLAB will indicate that what you typed is not a defined function or a known variable. At the same time, MATLAB suggests to write `exp(1)` instead.

1.3 MATLAB Help

Since MATLAB provides a large number of functions, it also has a very rich documentation built in. To get help about any of the available MATLAB functions, simply type

```
help function_name
```

and press enter (where `function_name` can be any of the MATLAB functions). For example, try

```
help exp
```

which explains how to use the exponential function and provides links to related functions. To get more details, you can click the link below "Reference page in Help browser," which opens a detailed Help window with even more explanations. Alternatively, you can select the function's name with your mouse and then press "F1" on the keyboard, so that the MATLAB help window for that function shows up. In the same Help browser, you can quickly search for all of the available MATLAB functions. As you will see during this project, the `help` function and "Help browser" are extremely useful.

1.4 Variables

In MATLAB, you often want to temporarily store and re-use information, such as numbers, vectors (a one-dimensional array of numbers), matrices (a two-dimensional array of numbers), or strings (an array of characters to form a sentences or text). Variable names in MATLAB are case sensitive and they can contain

up to 63 characters (but short variable names are often easier to remember). Variable names must start with a letter and can be followed by letters, digits, and underscores. For example, type

```
a = 1
```

and press enter. MATLAB creates the variable named "a" and adds it to the workspace and *assigns* the value 1; this assignment is also shown in the Command Window. At the same time, MATLAB confirms that you stored the value 1 in the variable:

```
>> a = 1

a =

     1
```

In addition, you can see that the variable a now contains the value 1 in the Workspace Window (on the right side of the MATLAB desktop). From now on, you can type a at the MATLAB prompt to access the value stored in this variable. For example, if you type

```
a+2
```

you can see that MATLAB added 1 (which was stored in the variable a) to the value 2. As the name "variable" implies, you can change (or redefine) the value of the variable a. Simply type

```
a = 10
```

and press enter to assign the new value 10 to variable a. You can see in the Workspace that the variable a now contains the value 10.

Activity 2: Add and subtract two variables

Familiarize yourself with the concept of variables. Create two variables: var1 and var2. Assign the value 10 to var1 and the value 5 to var2. Add var1 to var2; you should obtain 15. From var1, subtract var2; you should obtain 5. Try to change the values of both variables to whatever you like and observe your changes in the Workspace.

1.5 Vectors

Instead of numbers, you can also store *vectors* in variables. A vector is nothing but a multidimensional array of numbers. As you will see later, audio signals, for example, can be represented as vectors, where each entry of the vector contains the amplitude value of the signal at a moment in time. In MATLAB, you can create either row or column vectors. To define a row-vector, simply type

```
v = [2,4,6,8]
```

which creates a four-element (or four-dimensional) row-vector containing the values 2, 4, 6, and 8. If you inspect the Workspace Window, then you can see a vector variable named v containing four numbers. If you type v in the Command Window, then you will see the contents of the entire row vector:

```
>> v

v =

     2     4     6     8
```

Sometimes it is useful to access a single entry of a vector. This is easy. To access the first entry of the vector `v`, simply type `v(1)`. In general, to access the kth entry of a vector, simply type `v(k)`. (Here, `k` should be replaced by a number; you can also use variables to index vectors—more about this later.) Also note that this implies that vectors in MATLAB are indexed starting by 1. If you try to access an entry that is not defined, e.g., if you type `v(5)`, MATLAB will throw an error.

If you want to create a column-vector, simply type

```
w = [2;4;6;8]
```

and you can see that the values are now stacked on top of each other (called a column vector):

```
>> w = [2;4;6;8]

w =

     2
     4
     6
     8
```

Accessing entries for column vectors is the same as for row vectors: simply type `w(k)`, where `k` should be replaced by a number ranging from one to four for this example. Note that you can convert a row vector into a column vector by typing `v'` or a column vector into a row vector by typing `w'`. The mathematical term for the `'` operator is called the "transpose operator." If you want to multiply all entries of a vector by a scalar (for example, 5), you can simply type `5*w`. Again, this works for both row and column vectors.

Activity 3: Add two column vectors

Sometimes we want to add two vectors. For example, you add two vectors if you want to play two audio signals at the same time. In this case, you have to make sure that both vectors have the same dimension (number of entries) and are either both row vectors or both column vectors. (You should never add a column vector to a row vector or a row vector to a column vector.)

Create two variables (you can pick the variable names) containing column vectors of the same length with different entries. Add both vectors using the `+` operator. You should observe that the plus operator performs entry-wise addition of both vectors. Furthermore, the result should also be a column vector of the same length as the two vectors you defined.

1.6 Strings

MATLAB can also store so-called "strings," which are one-dimensional arrays of characters (letters). Creating a string is super easy— simply type

```
test_string = 'MATLAB is a lot of fun!'
```

where you have to use the correct quote symbol `'`. If you found the correct quote symbol, then you should see the following output:

```
>> test_string = 'MATLAB is a lot of fun!'

test_string =

MATLAB is a lot of fun!
```

Strings are useful to define filenames or data that should be transmitted over the air. One particular aspect we will use is to define filenames, for example

```
filename = 'test.wav'
```

which can be used to quickly load files in MATLAB—more about this later. One can also access each character of the string independently. For example, to read out the second character of the `filename` string defined above, simply type `filename(2)`. The output should be the second character (or letter) "`e`."

1.7 Automatically Generating Index Vectors

As you will see later, it is often extremely useful to create so-called index vectors that have a very specific set of entries. For example, assume that you want to create a row vector containing all the integers from 1 to 100. Clearly, it would be extremely tedious to actually type all 100 numbers in MATLAB to create such a vector. MATLAB enables one to create such vectors as follows:

```
a = 1:100
```

The first value (before the colon) indicates the start value; the second value (after the colon) indicates the stop value. If you inspect the Workspace, you can see that the variable `a` is of type 1×100 double (this implies that you created a row-vector with 100 entries). In case you want to extract the number of entries of a vector, simply type

```
length(a)
```

In the above example, this command should result in 100 (as the row vector you just generated has 100 entries). This means you have created a 1×100 array of double-precision numbers, which is nothing but a 100-dimensional row vector. If you double-click the variable `a` in the Workspace, a Variables window opens. Here, you can see the individual entries of this variable; you can also change the individual entries by editing the values. Simply press the $\times$ on the top-right corner of the Variables window to close it.

Activity 4: Read out multiple entries of a vector

Note that index vectors with integer valued entries are particularly useful to read out multiple values at once from a vector. Define a vector `testvector=[2,4,6,8,10]`. Assume that you want to read out the values of `testvector` corresponding to the indices $\{2,3,4\}$, then you can simply type `testvector(2:4)` which results in the output `[4,6,8]`. Equivalently, you can first define an index vector called `index=2:4` and then type `testvector(index)`, which reads the values of the vector `testvector` at the indices contained in the vector `index`. Try to read out the values at indices $\{3,4,5\}$ from the vector called `testvector`; this should result in $\{6,8,10\}$.

Assume that you want to generate a vector with entries between 2.5 and 5 but with 0.5 increment between neighboring values; this is known as arithmetic progression. To generate such a vector, type

```
b = 2.5:0.5:5
```

This command generates a vector with six entries and exactly the properties described above. The first value (before the first colon) indicates the start value, the second value (after the first colon and before the second colon) indicates the increment, and the third value (after the second colon) indicates the stop value.

It is often useful to generate vectors with `N` equally-spaced values between `x1` and `x2`. For example, 12 values between -8 and $+10$. To generate such vectors, simply use the `linspace` command:

```
c = linspace(-8,+10,12)
```

This command generates a 12-entry vector starting with -8 and ending in $+10$ with equally spaced entries (the spacing between entries is approximately 1.6364).

Activity 5: Generate column vectors with arithmetic progression

Generate a column vector with variable name `vec1` with entries from -10 to $+10$ with spacing 0.1. First use `vec1=-10:0.1:10` to create a column vector and transpose the result. Note that if you want to convert this row vector into a column vector, simply type `vec1'` in the Command Window. If you want to assign this column vector to variable `vec1`, then simply type `vec1=vec1'`. ***Remember: The equality operator in MATLAB is an* assignment *operator (it assigns the value or variable on the right to the variable on the left) rather than a mathematical equality operator (which you probably remember from calculus classes).***

Now, use the `linspace` command with the appropriate arguments to generate the same vector; call the vector `vec2`. Then, you need to transpose the resulting vector as the `linspace` command generates row vectors. Both vectors `vec1` and `vec2` should now be exactly the same.

It is now time to take a quick break from all the hard work. You have already learned a lot of the most important MATLAB functions we will use throughout the project. Type `why` in the Command Window and press enter to discover one of MATLAB's easter eggs; repeat the same command to get more answers to some of the most pressing questions.