POLYTECH
Peter the Great
St.Petersburg Polytechnic
University

Course

« Introduction to Biomedical Engineering»

Dr. Kirill Aristovich

Section 4: Basics of High-level programming: Matlab

Lecture 4.1: Basics of coding in Matlab

Basics of coding in Matlab

Hello, this section is all about high-level programming in Matlab. This is the final step before you will be rewarded by learning how to use acquired knowledge on practice systematically, and be able to design your own human-robot interface.

We will use Matlab, however, once knowing one, you can learn another language, like Python, yourself quite easily.

We will start by observing the Matlab environment, which, similar to other high level environments, offers extended features over the basic IDEs used for embedded programming. Matlab and similar languages, like Python, is a very powerful tool for computing and prototyping. It is easy to learn, and is an interpreter language, which means that it executes instructions directly and freely, without previously compiling a program into machine-language instructions. So you can type commands and immediately see the result.

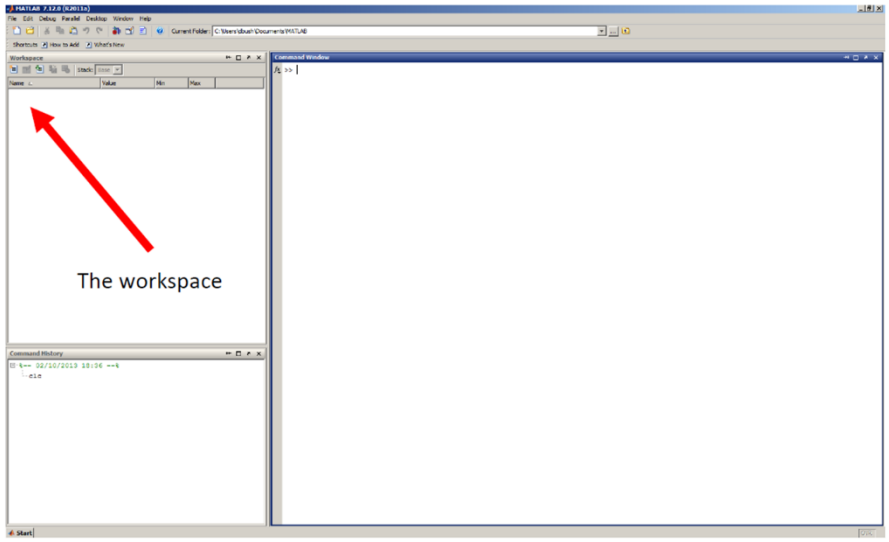

Figure 1 - Matlab environment

As usual with all programming languages, we will start with variables. There are plenty of differences compared to compiled languages. First of all, you do not have to declare the variable, Matlab will assign a type for you depending on the stuff that you are doing to it. All variables that has been used, and are not cleaned, appear in the workspace and are constantly located in the 'stack' memory. There are plenty of other features some of them are unique to Matlab, and some, which are common to other interpreter languages, are worth reading about.

Matlab is specifically designed to perform operations with matrices and vectors, which are assigned with square brackets, where rows are separated by a semicolons within those brackets. Numerical sequences can be assigned with colons, where the step size can also be defined, and defaults to 1.

All major matrix operations are overloaded, which means you can use the multiplication symbol to perform a matrix multiplication, or division symbol to perform matrix inverse. You can also perform element-wise operations, by adding the dot in front of the sign, for example A dot multiply B will multiply each element in A to a correspondingly located element in B.

Elements of a matrix can be accessed by location using round brackets, where row comes first, and column second. You can also create multidimensional matrices if you like to. One entire raw or column of a matrix can be accessed by a colon sign instead of a position, and matrices can be collapsed into vectors using single colon in brackets as well.

One important thing to notice is, contrary to most languages, numeration in matlab starts with 1, instead of 0. So if you need to access the first element of a vector, you write v(1), instead of v(0) in arduino or C. Annoying, I know.

Matlab has HUGE, and I mean HUGE number of built-in functions literally from all areas of mathematics and engineering, ranging from simple like 'mean', to a very specific complex functions like 'bsxfun'. The real trick to MATLAB is learning which functions exist and how to use them. If you want to perform some operation, just Google it – a function will exist!

The standard syntax for all functions is this, you need to specify output variables, which will be stored in the workspace, and pass the input variables, which either already exist, or created inside the function call.

Here is the list of simple functions for you to start experimenting.

Some rules about VARIABLES:

- MATLAB **does not care** about **spaces** in expressions
- In MATLAB, you can also assign values to **variables**
- **Mathematical operations** can then be performed on those **variables** in the **same way**
- All variables in the current '**stack**' (i.e. **in memory**) appear in the **workspace**
- **Unassigned** output is automatically placed in the variable '**ans**' (i.e. answer)
- **Output** to the **command window** can be **suppressed** with a **semi-colon** ';'…
- …but the value of the **variable** in the **workspace** will still be **updated**
- You can **display** the **value** of any variable by **typing its name** and pressing return
- You can **clear all variables** from the workspace by typing ***clear***
- You can clear **individual variables** by typing ***clear variable_name***

Some rules about Vectors and matrices:

- MATLAB is specifically designed to perform operations on **matrices** or **vectors**
- Matrices and vectors are assigned with **square brackets**
- Rows are **separated by semi-colons** within square brackets
- **Numerical sequences** can be assigned with **colons** (i.e. *start_value* : *finish_value*)
- The **step size** can also be **defined** (i.e. *start_value* : *step_size* : *finish_value*)

You can store and load the variables and data using specific .mat format. You can utilize the environment buttons, or use it in the code employing 'save' and 'load' functions. You can also save and load comma-separated text files using the commands 'dlmread' and 'dlmwrite'. Similar, you can employ 'imread' and 'imwrite' commands to read images into matrices and write matrices into image files.

Apart from matrices, there are strings, cells and structures, each one is good for a specific reason, so worth reading in the additional material. There are number of functions for conversion between those types in addition to usual numerical type conversion functions.

To start programming in matlab, you either type the command in the command window, or start a new script – the file that has list of all commands which will be executed sequentially, and has the extension .m

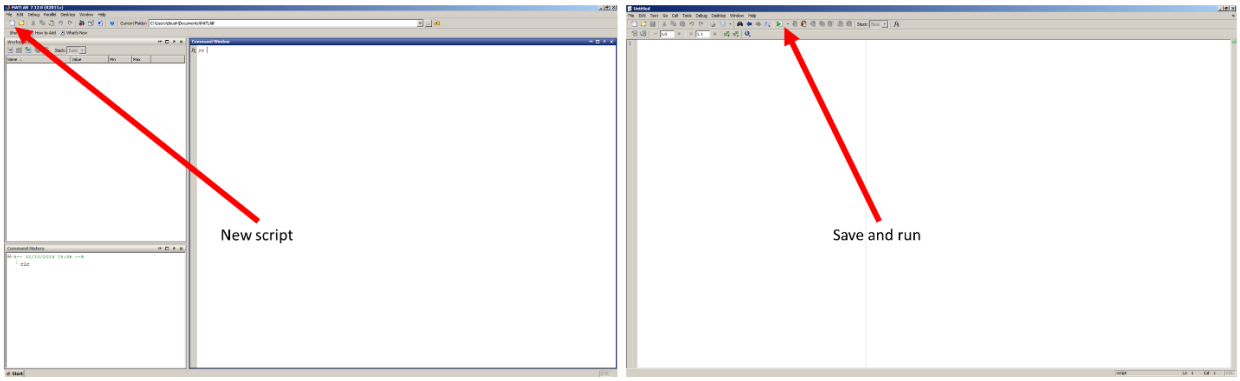

Figure 2 - Creating a script

It is good practice to annotate the code using the comments, which start with a % symbol. The code can also be divided into the cells, with double % symbol separating the cells. All code in the cell can be executed by pressing CTRL + ENTER when you cursor is in the cell, directly from the script file. This way you can easily debug chunks of code without running everything, by testing the result in each cell.

Obiously like in any program language, you can use loops – for and while. The syntax of those is on your screen. Operation is similar to the previous loops we have learned, with a slightly different syntax and the fact that you do not have to declare the variables. Also, the general rule is to avoid those in the code as in comparison to compiled languages they are very slow. Consider using matrix operations or built-in functions instead.

‘For’ Loops

- If you want to **repeat an operation multiple times**, use a ‘**for**’ loop
- The loop is executed ‘**for**’ **each of the entries** in a **counting array**
- In **some cases**, the **counting array** is **superfluous** to the code within the loop
- In **other cases**, the **code is executed** using **each value** in the **counting array**
- ‘**for**’ loops **must** always be **terminated** with an ‘**end**’ statement

```
for count = 1 : n
        output(count) = command(count);
end
clear count
```

- It is also generally **good practice** to ‘**clear**’ your **counting array**
- ‘**for**’ loops can also be **nested**

'While' Loops

- If you want to **repeat an operation until a condition is satisfied**, use a '**while**' loop
- The loop is executed '**while**' waiting for the condition to be satisfied
- Hence, the conditional variable **must be updated within the loop**
- '**while**' loops **must** always be **terminated** with an '**end**' statement

```
while condition(variable)
        update variable
end
```

- It is **easy** to get **stuck** in an **infinite while loop** (remember **ctrl+c**!)

Conditional statements are also there: if-else, with the specific set of logical operators, some of them operating structure-wise (to the entire matrix), and some can operate element-wise. There is also a switch-case structure, in case you are checking for the set of possible values for the same variable.

You can obviously create your own functions. Simplest way is to convert the existing script to a function by the following syntax, specifying input and output variables. Any comments that you write the top of the function will also be read by matlab help routine, and correctly displayed.

Writing Functions

- **Any script** can be **converted to a function** using the following **title line**:
  function[output1 output2 …] = *function_name*(input1, input2, …)
- Unlike scripts, **functions** use their own '**private**' **workspace**
- Any **required input must** be **passed directly** to the function
- Only **assigned output** will be **delivered** to the (base) **workspace**
- **Help information** can be entered in **comments at the top** of the function script

Unlike scripts, which can see all variables in the workspace, functions use their own 'private' workspace. So be careful when assuming you have already got some structure in place, a function will not see it unless you pass it to the input. Similar, none of the variables declared in the function will be seen outside of it, unless declared in the output.

MATLAB has very powerful built-in debugging for scripts and functions. Potential errors are underlined in red, they are also highlighted by orange or red marks in the warnings bar, MATLAB will suggest solutions to these potential errors, but they are not always appropriate or correct! Despite this, MATLAB cannot detect all potential errors, so be careful.

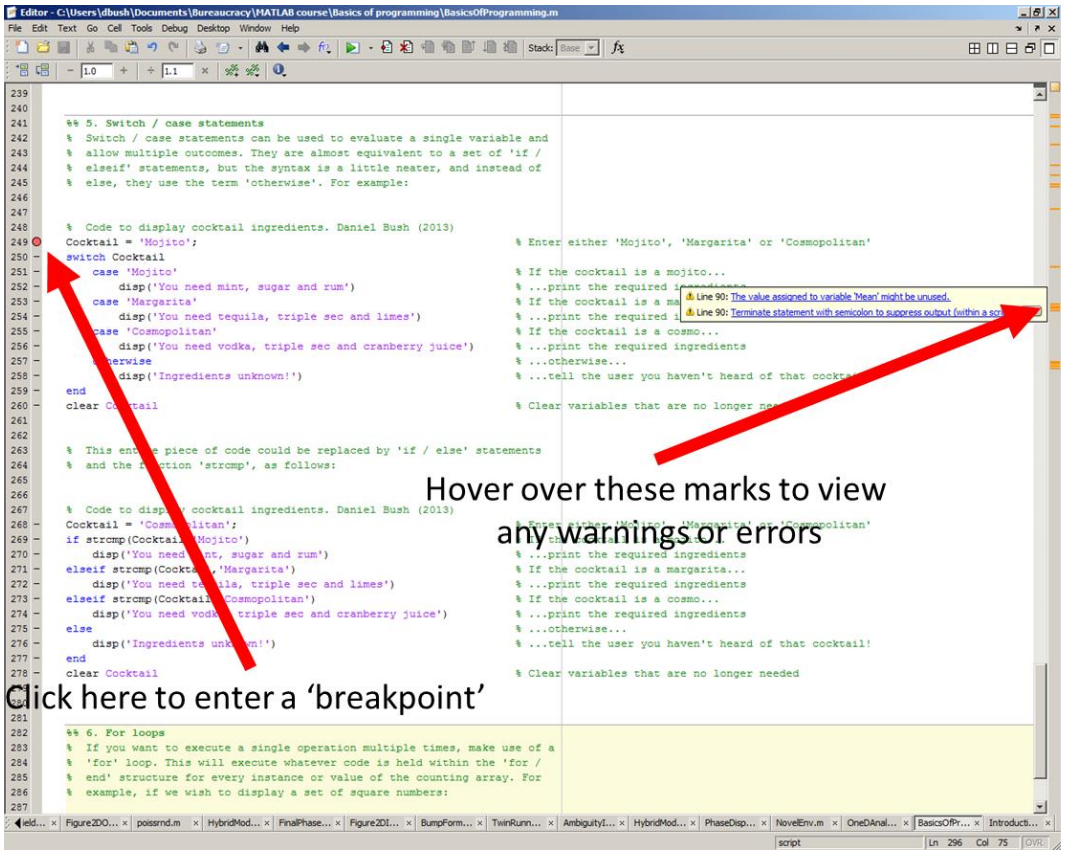

Figure 3 - Debugging

You can make use of the specific debugging mode. This is initiated by clicking in the left hand bar (by the line numbers), your code will then only run up to that line (where a red dot will appear). This is called a breakpoint. Clicking on the breakpoint again will remove it. A small green arrow will indicate current position within the code being executed. You can subsequently 'step through' your code one line at a time, which allows you to identify the location and source of errors. You can escape the debug mode by executing 'dbquit' command or clicking on the appropriate button.

Some graphic material used in the course was taken from publicly available online resources that do not contain references to the authors and any restrictions on material reproduction.

This course was developed with the support of
the "Open Polytech" educational project

Online courses from the top instructors of SPbPU