Exercise 2: MATLAB preliminaries for IN3015/IN4015

Håvard Kjellmo Arnestad (haavaarn@ifi.uio.no)

August 25, 2023

Introduction

In this first part of module 1 we will familiarize ourselves with some MATLAB concepts that will come in handy later in the course. This exercise will only go into a few of the details of MATLAB programming, and it is wise to use this opportunity to verify and/or learn what you need in order to be comfortable with basic scientific programming in MATLAB.

The best tip in general is to use the internet actively to find what you need. This includes YouTube, StackOverflow, and notably the documentation provided by MathWorks (the company behind MATLAB). It is well worth spending some time getting familiar with how functions (e.g., `sum`: `https://se.mathworks.com/help/matlab/ref/sum.html`) are documented.

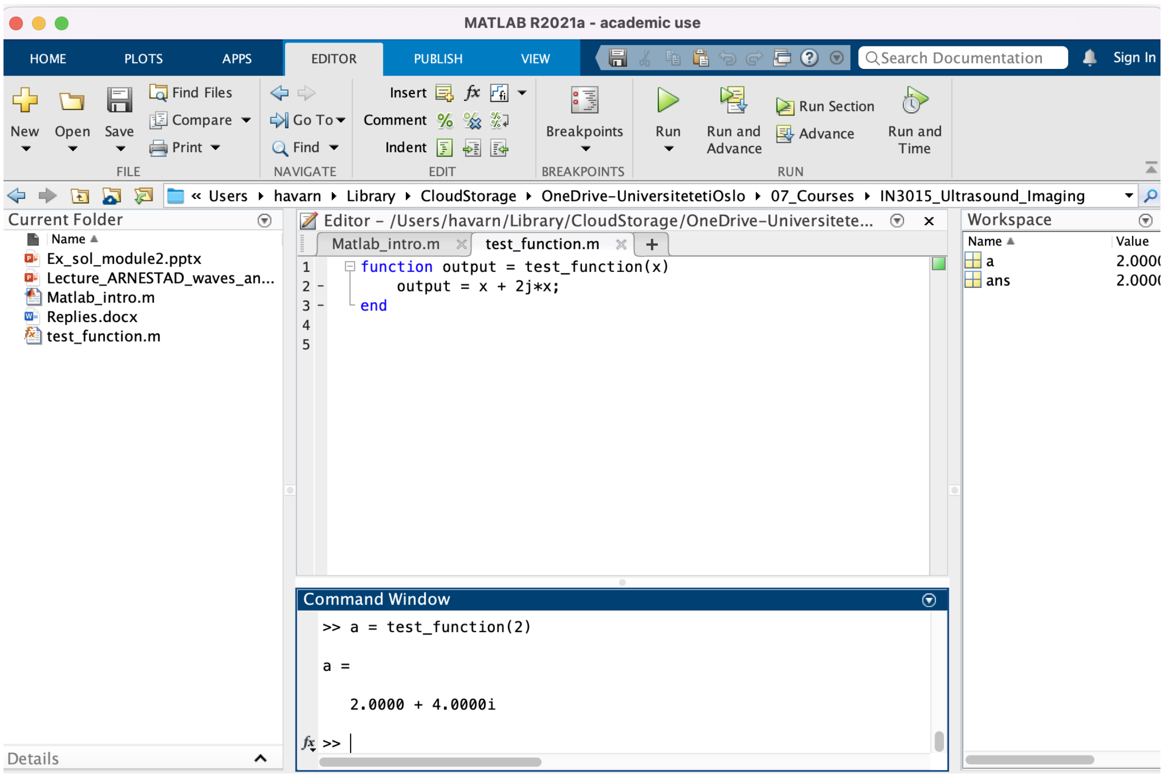
The experience from earlier years in IN3015/IN4015 is that students have varied backgrounds. Ideally you should know a programming language already, and for many this will be Python. The "scientific stack" in Python includes Numpy and Scipy, which is very similar to MATLAB. Two relevant resources for those with a Python background are *Numpy for Matlab users* at
`https://numpy.org/doc/stable/user/numpy-for-matlab-users.html`
or *Matlab for Python users* at
`https://blogs.mathworks.com/student-lounge/2021/02/19/introduction-to-matlab-for-python-users/`

MathWorks have a lot of official resources, for instance this *getting started* page:
`https://se.mathworks.com/help/matlab/getting-started-with-matlab.html`
An alternative is the onramp tutorial that starts from the very basics, and lets you program interactively in the web browser. Note that you do need to be logged in:
`https://matlabacademy.mathworks.com/details/matlab-onramp/gettingstarted`

Within MATLAB

In Fig. 1 we can see the MATLAB user interface. The left pane is the "current folder" which shows us what files MATLAB has access to. One of these functions is the `test_function.m` that is also shown in the "editor". This function takes in a number x, and returns a complex number $x + 2ix$. The "command window" allows us to run this function, and we assign the output to the variable `a`. This variable is now known to MATLAB as we can see it in the "workspace", but when we can use it depends on the *scope* (scope determines where in your program a name is visible).

In many cases, such as when we are working with USTB, our functions are not located in our current folder. We then need to make sure the functions are on the MATLAB "path". MATLAB uses the search path to locate



```
function output = test_function(x)
    output = x + 2j*x;
end
```

```
>> a = test_function(2)

a =

   2.0000 + 4.0000i

>> |
```

Figure 1: The MATLAB interface.

files in the system. You can include folders from anywhere to the path by navigating to them, right-clicking, and selecting add to path, or by using the `addpath` function.

TASKS

In the `MATLAB_intro.m` file, you will find sections of code (separated by `%%`) that can be run individually. Run the code and answer the questions in the file.

Debugging

At some point you will certainly run into bugs that are hard to fix. Learning to use MATLABs debug tools are very helpful. Writing `dbstop if error` is one way to set breakpoint when something goes awry. In nested function calls it can be difficult to asses why an error is raised, but from the toolbar during debug mode you can always access any layer of the "Function Call Stack". Write code that raises an error, and see if you can find the bug using the built-in tools.