

## MA5233 Computational Mathematics

# Homework Sheet 1

Simon Etter, 2019/2020

Recommended deadline: 22 August 2019

Final deadline: 29 August 2019, 11.59pm

**Exercises 1** and **2** of this sheet provide an introduction to Julia and are unmarked. The marked **Exercises 3** and **4** may be answered using pen and paper only, although access to Julia may be convenient to check your findings.

## 1 Installing Julia (unmarked)

**Note:** If you run into any problems with Julia, please contact me ([ettersi@nus.edu.sg](mailto:ettersi@nus.edu.sg)), but please do so well before the submission deadline. Julia-related issues will not be accepted as an excuse for late/incomplete submission.

The homework for this module will require you to write code using the Julia programming language. Please go to

<https://julialang.org/downloads>

and follow the instructions there to install Julia v1.1.1 on your computer. Any reasonably recent computer should be 64-bit, so it is best to download this version first and only try the 32-bit version if 64-bit did not work.

You are free to use any text editor you like to write your Julia code, but I recommend the Juno Integrated Development Environment (IDE). Juno is built on top of Atom, which is a general purpose text editor; hence in order to install Juno, you will have to install Atom first. A complete guide on how to install both Atom and Juno can be found under

<http://docs.junolab.org/latest/man/installation>.

## 2 Getting started with Julia (unmarked)

You may find the following information useful to get started with Julia.

### Starting Julia.

The main interface to Julia is the REPL, which stands for Read-Eval-Print-Loop. If you installed Juno, you can open a REPL as described in Step 4 in the installation instructions linked above.

Otherwise, if you are on Windows or Mac, there should be an app called Julia. Opening it like any other app will open up a Julia REPL. If you are on Linux, run the executable `bin/julia` in the terminal to open the REPL.

Once the REPL is open, you can type simple mathematical formulae to check that everything is working. For example, typing `exp(1)` should give you the following.

```
julia> exp(1)
2.718281828459045
```

## Loading code

You will be asked to submit your code as a text file which can be loaded into Julia. In order to simulate this process, create a file `test.jl` somewhere in your file system and add the following line:

```
println("hello world")
```

The command to run this file is `include("test.jl")`, but this command looks for the file relative to the *working directory*. You can see your current working directory using `pwd()` (Path to Working Directory), and you can change the working directory using `cd(path)` (Change Directory). In the following example, I created the file `test.jl` in the folder `/tmp/`, so I first move the working directory accordingly, then I run the file.

```
julia> pwd()
"/home/ettersi"

julia> cd("../tmp")

julia> pwd()
"/tmp"

julia> include("test.jl")
hello world
```

Note that on Windows, you have to use `\\` instead of `/` (double backslashes are required because `\` is a special character in a Julia string).

Alternatively, you can also use absolute file paths, which are paths starting with `/` (or `\\` on Windows).

```
julia> include("/tmp/test.jl")
hello world
```

## Basic syntax

The following example illustrates the syntax for function definition, if-else clauses and variable assignment.

```
function foo(x)
    if x == 1
        reply = "You called foo(1)"
    else
        reply = "You called foo(x) with an argument which is not 1"
    end
    return reply
end
```

```
julia> foo(1)
"You called foo(1)"
```

```
julia> foo(2)
"You called foo(x) with an argument which is not 1"
```

Julia also provides a shorthand notation for function definitions illustrated below.

```
julia> f(x) = x^2
```

```
julia> f(2)
4
```

## Where to get help

- Type `?` followed by a function name to get help for a particular function, e.g. `?pwd`.
- Julia documentation: <https://docs.julialang.org/>
- Google. Recommended websites are [discourse.julialang.org](https://discourse.julialang.org) and [stackoverflow.com](https://stackoverflow.com).
- Contact me: [ettersi@nus.edu.sg](mailto:ettersi@nus.edu.sg).

## 3 Stable and unstable implementations

Consider the function

$$f(x) = \frac{1 - \sqrt{1-x}}{1 + \sqrt{1-x}}$$

and its Julia implementation

```
f(x) = (1-sqrt(1-x)) / (1+sqrt(1-x)).
```

A useful tool to assess the accuracy of an implementation is the `BigFloat` type, which uses 256 bits for representing the significand rather than 52 as in `Float64`. This means that results computed using `BigFloat` may be considered exact relative to `Float64`, which allows us to compute forward errors. For the above function, we observe the following.

```
julia> x = 1e-12;
      abs( f(x) - f(big(x)) ) / abs( f(big(x)) )
      # big(x) converts x to BigFloat type
8.890058209099938...e-05
```

We see that for  $x = 10^{-12}$ , our implementation of  $f(x)$  loses about 11 digits of accuracy relative to the machine precision  $\text{eps}(\text{Float64}) = 2.2 \times 10^{-16}$ , and the question arises whether such loss in accuracy is unavoidable in floating-point arithmetic or whether our implementation could be improved.

1. Provide an argument that the condition number  $\kappa(f, x)$  is bounded for  $x \in [0, 0.5]$ .  
*Hint:* Your argument should be convincing but it does not have to be mathematically rigorous. You are allowed to use symbolic computer algebra systems like WolframAlpha for this part.
2. Propose a modified implementation of  $f(x)$  which avoids the loss in accuracy described above.

## 4 Conditioning and stability of $\sin(x)$

1. Compute the condition number of  $\sin(x)$ .
2. Typing `sin(pi)` in Julia yields `1.2246467991473532e-16`. Compute the relative forward error of this result as an approximation to  $\sin(\pi)$ .
3. Does the answer to [Task 2](#) mean that the Julia implementation of  $\sin(x)$  violates the IEEE specification of returning the exact result rounded to the nearest representable number?  
*Hint:* `sin(pi)` is equivalent to `sin(Float64(pi))`, where `Float64(pi)` rounds  $\pi$  to the nearest `Float64` value.
4. Provide a connection between the observations in [Tasks 1](#) and [2](#).