MA5233 Computational Mathematics

Introduction to Julia

Simon Etter, 2020

We will be using the Julia programming language both for demonstrations in class and the homework assignments. This document will help you getting started.

1 Installing Julia and VS Code

We will be using Julia v1.5.0 in this module, which you can download from

https://julialang.org/downloads.

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.

Moreover, I will be using the VS Code programming environment in class, and I recommend you do the same for your homework. You can find instructions on how to install VS Code and its Julia plugin under

https://www.julia-vscode.org/docs/stable/setup/.

2 Using Julia

Starting Julia

The main interface to Julia is the REPL (Read-Eval-Print-Loop). To open a Julia REPL in VS Code, press `Ctrl+Shift+P`, type `Julia: Start REPL` and hit `Enter`. Note that starting the Julia REPL may take a few seconds.

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

The REPL is convenient to play around with short, one-line segments of code. For more advanced programming, you will want to write your code in a text file and then execute that file.

You can create a new file in VS Code as follows:

- Open the Explorer Side Bar either by pressing `Ctrl+Shift+E` or by clicking on the file icon in the top-left corner.
- Click on Open Folder and navigate to the folder where you want to create your file (e.g. Desktop for the sake of this simple demonstration).

- Click on the New File icon which will appear as the left-most icon when you hover over the folder which you just opened.
- Type in the filename (e.g. `test.jl`) and hit enter.

Once you have created the file, copy the following line into it.

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

To execute this code, open the Julia REPL (see above) and run the following command.

```
julia> include("test.jl")   # Copy-paste this line
hello world                 # This is the output that you should see
```

Basic syntax

Julia's syntax is very similar to Matlab, so if you are familiar with Matlab you should have no problem writing Julia code. However, there are a few noteworthy differences.

- Julia arrays are indexed with square brackets, i.e. write `A[i,j]` instead of `A(i,j)`.
- Comment lines start with `#` rather than `%`.
- In Matlab, `zeros(5)` returns a 5×5 matrix of zeros. In Julia, `zeros(5)` returns a vector of five zeros.
- To apply a function to each entry of an array, you have to add a `.` after the function name, otherwise you will get an error message. For example:

  ```
  julia> exp([0,1])
  ERROR: MethodError: no method matching exp(::Array{Int64,1})

  julia> exp.([0,1])
  2-element Array{Float64,1}:
   1.0
   2.718281828459045
  ```

  Here is another example for a binary operator:

  ```
  julia> [0,1]*[2,3]
  ERROR: MethodError: no method matching *(::Array{Int64,1}, ::Array{Int64,1})

  julia> [0,1].*[2,3]
  2-element Array{Int64,1}:
   0
   3
  ```

- Unlike Matlab, Julia distinguishes between integers and floating point numbers. Some examples:

  ```
  julia> a = -1; 2^a
  ERROR: DomainError with -1:
  Cannot raise an integer x to a negative power -1.
  Make x or -1 a float by adding a zero decimal (e.g., 2.0^-1 or
  2^-1.0 instead of 2^-1), or write 1/x^1, float(x)^-1, x^float(-1) or (x//1)^-1

  julia> a = [1,2]; a[1.0]
  ERROR: ArgumentError: invalid index: 1.0 of type Float64
  ```

Exercise: Write a function `squared(n)` which returns the vector `[1,4,9,...,n^2]`.

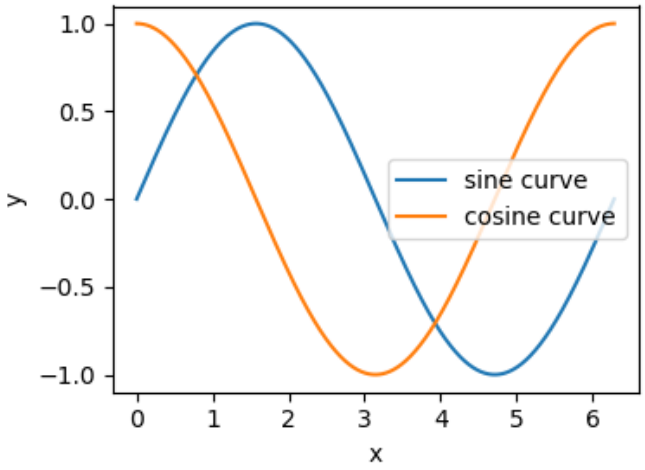
Hint. Type `?function` and hit enter to access the documentation for defining functions.

3 Creating plots using PyPlot

Plots can be created in Julia using the `PyPlot` package. To install this package, open the REPL, type `]` followed by `add PyPlot` and press enter. Once the installation has completed, you can load the package by typing `using PyPlot`.

The following example illustrates most of the commands that you will need for this class:

```
using PyPlot
figure(figsize=(4,3)) # Only needed to specify the figure size
x = LinRange(0,2pi,1000) # 1000 equispaced points in [0,2pi]
plot(x, sin.(x), label="sine curve")
plot(x, cos.(x), label="cosine curve")
legend(loc="best")
xlabel("x")
ylabel("y")
tight_layout()  # resizes the plot so x and y label are not cut off
display(gcf())  # display the figure in VS Code
savefig("test.png")
```



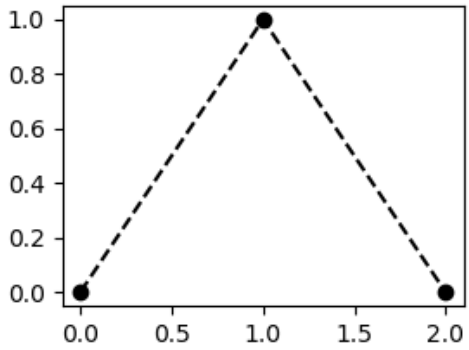
Exercise: Create a plot of $f(x) = x^2$ on $[-1, 1]$.

Other commands which may prove useful are:

- `semilogx()`: use logarithmic scale for x axis
- `semilogy()`: use logarithmic scale for y axis.
- `loglog()`: use logarithmic scale for both axes.
- `plot(x,y,format,...)` with `format` being a combination of the following allows you to change the line format.
  - Line styles: `-` (solid), `--` (dashed), `-.` (dash-dotted), `:` (dotted).

- Markers: see `https://matplotlib.org/3.1.1/api/markers_api.html`.
- Colours: `C0` to `C9` (default colour palette), `k` (black)

Example: `plot([0,1,2], [0,1,0], "k--o")` yields the following.



4 Where to get help

Please do not hesitate to contact me via the LumiNUS chat if you run into any issues with Julia, but please do so well before the submission deadlines. Julia-related issues will not be accepted as an excuse for late/incomplete submission.

Other useful resources:

- Type `?` followed by a function name or a keyword in the REPL to access the documentation for that function or keyword. In VS Code, this documentation will also pop up if you hover over a function name.
- Julia documentation: `https://docs.julialang.org/`
- Google. Recommended websites are discourse.julialang.org and stackoverflow.com.