Experiments in Spectroscopy

Markus Lippitz

September 26, 2021

Contents

Part I

Fundamentals

Part II

Two Level Systems

Part III

Nonlinear Spectroscopy

Part IV

Plasmonics

Part V

Nanooptics

Appendices

Appendix A
Julia and Pluto

Markus Lippitz
September 24, 2021

We use the programming language *Julia*[1] for graphical illustrations and numerical 'experiments' in this course. I am convinced that only when you can persuade a computer to do something, to display a model, to calculate a value, only then you have really understood it. Before that, you just haven't seen all the problems.

You can use Julia with different user interfaces. We use *Pluto*.[2]

[1] https://julialang.org

[2] https://github.com/fonsp/Pluto.jl

Julia

Julia is a programming language designed for numerics and scientific computing. It is a middle ground between Matlab, Python, and R. In my view, it takes the best of each of these worlds, making it especially suitable for beginners. We will discuss several example scripts together during the semester, and there will also be numerical practice problems.

An example

First, let's look at a simple example.

```
using Plots
x = range(0, 2 * pi; length=100)
plot(x, sin.(x); label="ein Sinus")
```

For some things you need libraries which you can load with `using`. When choosing libraries, first stick to the examples I show.

Then we define a variable `x` (simply by using it) as an equidistant 'number string' between 0 and 2π with 100 values. Functions like `range` always have required parameters defined by their position in the parameter list (here: start and end value), plus other optional ones. These follow after a semicolon in the form `<parameter>=<value>`.

Finally, we draw the sine function over this range of values. Notice the dot in `sin.(x)`. It means 'apply `sin` to all elements of `x`'. This is very convenient.

Sources of information

The current version of Julia is 1.6.2. Some things have changed with version 1.0. Ignore websites that are older than 2 years or that refer to a version before 1.0.

official documentation on the website[3]. Or ask google with 'Julia' as keyword or with the library / function and appended extension '.jl' .

examples Julia by example[4], Think julia[5], Introduction to Computational Thinking[6]

differences comparison[7] with Matlab, Python, and other languages, and similarly as an overview table[8]

Cheat Sheets general[9] and for plots[10]

[3] https://docs.julialang.org/en/v1/

[4] https://juliabyexample.helpmanual.io/

[5] https://benlauwens.github.io/ThinkJulia.jl/latest/book.html

[6] https://computationalthinking.mit.edu/Spring21/

[7] https://docs.julialang.org/en/v1/manual/noteworthy-differences/

[8] https://cheatsheets.quantecon.org/

[9] https://juliadocs.github.io/Julia-Cheat-Sheet/

[10] https://github.com/sswatson/cheatsheets/

User interfaces

There are several ways to write shorter or longer programs in Julia. Here is a selection

command line and editor One can use Julia interactively at the command line (REPL, read-eval-print loop). In an external editor one could write repeating commands in script files.

IDE This is more comfortable with an integrated environment, for example Juno[11], or a Julia extension [12] for Visual Studio Code. This is certainly the approach for larger projects.

Jupyter notebook Jupyter[13] is composed of Julia, Python and R. These three languages can be used in a notebook format. Program code is in cells, the output and also descriptive text and graphics in between. This is particularly suitable if calculations are to be accompanied by descriptions or equations, for example in lab protocols or exercises.

Mathematica has a similar cell concept. One drawback is that cells affect the state of the kernel in the order in which they are executed. However, the order need not be the same as in the file; in particular, deleting cells does not change the kernel. This can be very confusing, or you may have to restart the kernel often.

[11] https://junolab.org/

[12] https://www.julia-vscode.org/

[13] https://jupyter.org/

Pluto One can also mix program code, text and graphics in Pluto[14]. The cell concept of Pluto is that of Excel, however, limited to one Excel column. The arrangement of equations in the cells does not matter. Everything is re-evaluated after each input. A logic in the background ensures that only absolutely necessary calculations are re-executed. From my point of view, this should be intuitive to use for beginners and should be quite sufficient for smaller projects. *We use Pluto as the user interface in this course*.

[14] https://github.com/fonsp/Pluto.jl

Installation

Server of EP III To simplify your first steps you can use the Jupyter & Pluto server [15] of EP III. For this you have to be inside the university or con-

[15] http://jupyter.ep3.uni-bayreuth.de

nected via VPN. You will receive access data during the first week of the semester. Log in to the server with these. You will get to a Juypter interface where you can manage files on the server. Click on the Pluto icon to start a Pluto interface in the web browser.

Please be considerate with this server. Its resources are rather limited.

Local installation Especially if you find the EP III server too slow you should install Julia and Pluto locally. A good guide is at MIT[16]. Short version: install Julia from the website, then install once the Pluto package in Julia locally (`import Pkg; Pkg.add("Pluto")`). To use it, call it from the Julia command line via `using Pluto; Pluto.run()`. This could go into the Julia startup file or could be passed via the (system) command line.

[16] https://computationalthinking.mit.edu/Spring21/installation/

Using Pluto

For a nice introduction to Pluto (and Julia), see the Pluto homepage[17], at MIT (here[18] or actually the whole site) and at WIAS.[19]

- Shift-Enter executes a cell
- The execution optimizer requires that each cell forms a closed block. So there must be only one command, or several need to be encapsulated by `begin` ... `end`.
- Each cell has only one output, that of the last line. The output is above the cell itself.
- Pluto manages libraries automatically, just use `using`. You do not need to download anything.
- Pluto automatically saves everything, but you can rename / move the file.

[17] https://github.com/fonsp/Pluto.jl/wiki
[18] https://computationalthinking.mit.edu/Spring21/basic_syntax/
[19] https://www.wias-berlin.de/people/fuhrmann/SciComp-WS2021/assets/nb01-first-contact-pluto.html