The Journal of Open Source Education

Scientific programming in Julia - An introductory course

Andreas Kuhn [1] **and Sabine C. Fischer** [1]

1 Center for Computational and Theoretical Biology , University of Würzburg

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Submitted: 01 January 1970
Published: unpublished

Summary

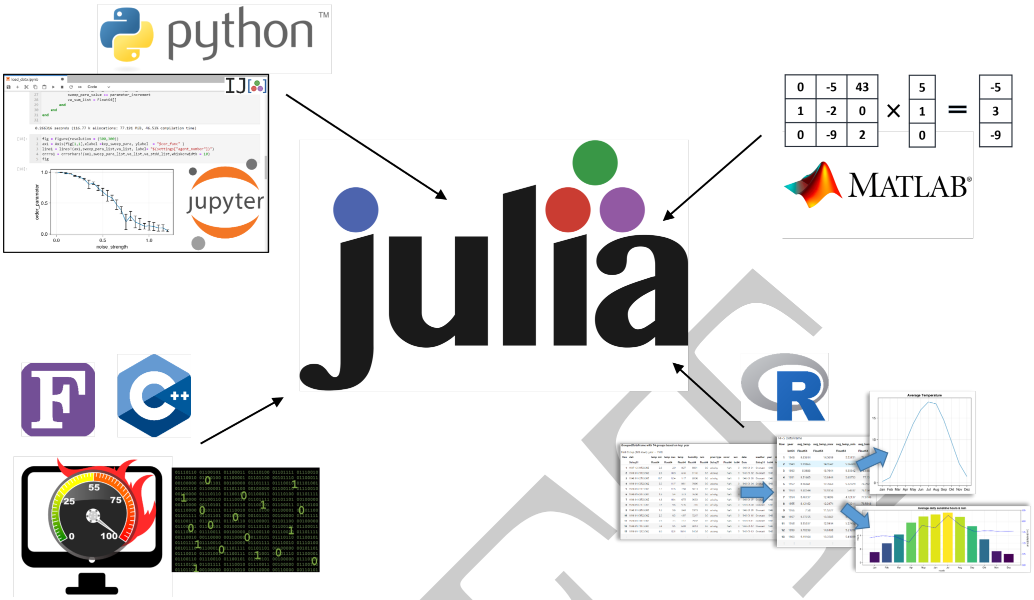
This course provides an introduction into the scientific programming language `Julia`. No previous knowledge is necessary to follow the nine course lectures. Every lecture is presented in Jupyter Notebooks. Code examples are shown together with markdown cells explaining and elaborating on the code. Additionally, general concepts about programming in `Julia` are introduced. The lectures build on each other and should be finished in numerical order. The first five lectures cover the basic language syntax of `Julia` (e.g. data types, operators, package installation, functions, …). The last four lectures are more specifically targeted towards scientific applications and cover the following topics: plotting, import/export of numerical data, data analysis and a simple agent-based simulation. Each lecture ends with exercises, where the students directly apply the concepts that have been introduced. Sample solutions for all exercises are provided.

Statement of Need

The `Julia` programming language first appeared in 2012 (Bezanson et al., 2012) and had been specifically designed for numerical computation and data science. Since then, the combination of speed and interactivity distinguishes `Julia` from other languages in that realm. Over the years, it has undergone a rapidly growing adaptation to various fields of science, ranging from black holes (Fernandes & Mulryne, 2022) to quantum systems (Gawron et al., 2018; Krämer et al., 2018) in physics to applications in biology (Knopp & Grosser, 2021; Roesch et al., 2021), economics (Coleman et al., 2021), machine learning (Gao et al., 2020) and many more. This fast acceptance of a new programming language is quite remarkable, as programming languages are a field that is historically known to be very slow at pivoting (e.g. `Python` and `C++` are both over 30 years old).

One reason for this is that `Julia` solves the two language problem (Perkel, 2019) that has been tormenting many researchers for decades. It essentially means, that different specialized languages are used for different tasks in the same project. To give an example, `Python` or `Matlab` are often used for plotting and fast prototyping, `R` for statistical analyses and compiled languages like `C/++` or `Fortran` for the computationally heavy parts. Using two or more languages in one project creates a lot of unnecessary overhead: Firstly, researcher have to spend more time learning to code in two languages (especially true for low level languages like `C/++` or `Fortran`), and secondly, exchanging data and logic between languages can be very tedious and time-consuming.

Julia solves this problem, as all the mentioned tasks can be accomplished in Julia alone. Its simple syntax is heavily inspired by Python, which allows equally fast prototyping (Kuhn, 2022). The plotting package `Makie` (Danisch & Krumbiegel, 2021) provides similar capabilities as `matplotlib` in `Python` or `ggplot2` in `R`. The meta package `StatsKit` includes all necessary tools to do sophisticated statistical analysis. And most importantly, `Julia` is fast. Depending on the benchmark, `Julia` is on pair or within a factor of 2 to `C` (Churavy et al., 2022; Jeff Bezanson, 2021).



Target audience

The main target audience of this course is scientists/students who want to use or already use programming in their work. However, no prerequisites apart from basic algebra and statistics (high school level) are needed. Anybody with that requirement should be able to follow the course without problems.

Programmers who already have experience in other languages should be able to finish the first five lessons quite fast, as a lot of general programming concepts are introduced there as well. But the last four more `Julia` specific lessons should be very useful for them, too.

Even though there are already free Julia courses available online (1,2,3), we believe, that this course offers great additional value for (new) scientific programmers, as it does not require any prior knowledge, is self-contained with chapters that build up on one another, introduces many relevant topics for scientific applications and offers exercises together with solutions to each chapter.

Therefore, we invite everybody to either use the course materials in idependent self study to learn `Julia` or to use the course materials to teach Julia to others.

Details of the course

Learning objectives

The learning resource aims to enable learners to:

- use interactive notebooks
- understand the basic structure of the `Julia` programming language
- use Julia in a scientifc context:
  - import/export data
  - analyse data
  - plot data
  - write a simple simulation

Content

1. Jupyter Notebooks

2. Package Installation
3. Datatypes
4. Conditional Iterators
5. Functions
6. Plotting
7. Import/Export
8. Data Analysis
9. Application: Simple Random Walk Simulation

Instructional design

The Jupyter notebooks contain four different types of cells: - General explanations - Executable code examples - Exercises - Meta comments/hints/notes

The first three cell types are essential in understanding the contents of the course. The exercises are divided into three difficulties: easy, medium and hard. Easy and medium exercises can be solved using the content of the course alone, whereas some hard exercises require additional research to transfer the acquired knowledge to new problems. The "meta" cells offer additional but non-essential information. This can be comments about the differences of `Julia` compared to other languages, advice on good coding practices, etc. While first time programmers can skip these cells if they want to, we expect the meta cells to be particularly useful for already experienced programmers switching to `Julia`.

Experience of use

We started using `Julia` in our own work out of frustration with the execution speed of raw Python. The choice of `Julia` over other compiled and fast languages like C/++, FORTRAN or Rust was easy for us because `Julia` allowed us to keep the familiar and convenient interactive development workflow that Python established.

During the transition, we also had to learn `Julia` from scratch. And what is the best way to learn something new? Right, teaching it. Therefore, we developed this course to educate ourselves and new members of our group on `Julia`. Since the beginning of 2022, thirteen other people have completed the material as a self-study course. Furthermore, we have been providing the course material on our website as a free download since March 2022. The feedback was used to iteratively improve the course.

We believe that the course has reached a level of maturity so that it offers great value to everybody who wants to learn or teach `Julia`. We are convinced that the course could be readily taught in a weekly manner as part of an undergraduate or graduate program. Therefore, we invite everybody to either use the course materials in independent self study to learn `Julia` or to use the course materials to teach `Julia` to others.

Acknowledgements

We acknowledge contributions from Simon Schardt, Christopher Nauroth-Kreß, Alexander Leipold and the various other students who served as beta testers and proof readers for this course.

References

Bezanson, J., Karpinski, S., Shah, V. B., & Edelman, A. (2012). *Julia: A Fast Dynamic Language for Technical Computing*. arXiv. https://doi.org/10.48550/arXiv.1209.5145

Churavy, V., Godoy, W. F., Bauer, C., Ranocha, H., Schlottke-Lakemper, M., Räss, L., Blaschke, J., Giordano, M., Schnetter, E., Omlin, S., Vetter, J. S., & Edelman, A.

(2022). *Bridging HPC Communities through the Julia Programming Language*. arXiv. https://doi.org/10.48550/arXiv.2211.02740

Coleman, C., Lyon, S., Maliar, L., & Maliar, S. (2021). Matlab, Python, Julia: What to Choose in Economics? *Computational Economics*, *58*(4), 1263–1288. https://doi.org/10.1007/s10614-020-09983-3

Danisch, S., & Krumbiegel, J. (2021). Makie.jl: Flexible high-performance data visualization for julia. *Journal of Open Source Software*, *6*(65), 3349. https://doi.org/10.21105/joss.03349

Fernandes, P. G. S., & Mulryne, D. J. (2022). *A new approach and code for spinning black holes in modified gravity*. arXiv. https://doi.org/10.48550/arXiv.2212.07293

Gao, K., Mei, G., Piccialli, F., Cuomo, S., Tu, J., & Huo, Z. (2020). Julia language in machine learning: Algorithms, applications, and open issues. *Computer Science Review*, *37*, 100254. https://doi.org/10.1016/j.cosrev.2020.100254

Gawron, P., Kurzyk, D., & Pawela, Ł. (2018). QuantumInformation.jl—A Julia package for numerical computation in quantum information theory. *PLOS ONE*, *13*(12), e0209358. https://doi.org/10.1371/journal.pone.0209358

Jeff Bezanson, V. S., Stefan Karpinski. (2021). *Julia Micro-Benchmarks*. https://julialang.org/benchmarks/

Knopp, T., & Grosser, M. (2021). MRIReco.jl: An MRI reconstruction framework written in Julia. *Magnetic Resonance in Medicine*, *86*(3), 1633–1646. https://doi.org/10.1002/mrm.28792

Krämer, S., Plankensteiner, D., Ostermann, L., & Ritsch, H. (2018). QuantumOptics.jl: A Julia framework for simulating open quantum systems. *Computer Physics Communications*, *227*, 109–116. https://doi.org/10.1016/j.cpc.2018.02.004

Kuhn, A. (2022). How to solve the same numerical Problem in 7 different Programming Languages. In *Medium*. https://medium.com/@andreaskuhn92/how-to-solve-the-same-numerical-problem-in-7-different-programming-languages-a64daac3ed64

Perkel, J. M. (2019). Julia: Come for the syntax, stay for the speed. *Nature*, *572*(7767), 141–142. https://doi.org/10.1038/d41586-019-02310-3

Roesch, E., Greener, J. G., MacLean, A. L., Nassar, H., Rackauckas, C., Holy, T. E., & Stumpf, M. P. H. (2021). *Julia for Biologists*. arXiv. https://doi.org/10.48550/arXiv.2109.09973