

1. Introduction

- [1. Introduction](#)
 - [Why Julia?](#)
 - [How I started Julia \(A personal story of two language problems!\)](#)
 - [Rough outline](#)
 - [About the course and philosophy](#)
 - [Acknowledgements](#)
 - [References](#)

Why Julia?

Let's hear from the 4 co-creators of Julia

We want a language that's open source, with a liberal license. We want the speed of C with the dynamism of Ruby. We want a language that's homoiconic, with true macros like Lisp, but with obvious, familiar mathematical notation like Matlab. We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab, as good at gluing programs together as the shell. Something that is dirt simple to learn, yet keeps the most serious hackers happy. We want it interactive and we want it compiled. (Did we mention it should be as fast as C?)Jeff Bezanson, Stefan Karpinski, Viral B. Shah and Alan Edelman.

I think I do not have to convince you that Julia could be a useful programming language for scientific computing, I mean this is why you are here right? At least that is what I am thinking. The important question is **why**? And if Julia is really fast, why not everyone is moving to Julia? Let's answer the **why** part over the course. I think there could be two reasons why people in the scientific computing community is still not switching,

1. It is often costly to switch for us (except those computer scientists who learn a new programming language per day)
2. Julia is still sort of its **just before** maturity age. For me everything in the language now seems to be more or less stable, however I still feel in terms of the overall maturity (with packages, with other fields integrating and so on) maybe it is not similar to Python or Matlab yet (but this is my feeling and I maybe 100% wrong), so many are scared whether the investment of time and other resources will be useful at the end. As you already know previously many languages died over the time, so people are still somewhat skeptic.
3. Some of us do not want to embrace the change so easily. If we use one language for 20 years, and we are in our comfortable zone, we tend to use the same thing even though we

know there are better alternatives out there and sometimes even worse, we want everyone to use the older stuffs. I think this is not healthy! Any technology should be judged based on its own merit, I should not advertise it just because I am comfortable with it. My friends and colleagues might disagree with me on this point. One thing I have to tell you before I start, for me healthy disagreements are fine and I am happy to learn and share this way!

How I started Julia (A personal story of two language problems!)

I have a strange story to share on how I started using Julia. So in 2018 I started working at this amazing university TU Dortmund, Germany (*to the students of TU Dortmund - you guys should be thankful that you are here, so please use the opportunity as best as possible to learn and do not waste this opportunity!*). Here I met an amazing colleague named Tileman Conring, and eventually we became good friends. I don't recall how, but somewhere during that time Tileman told me about Julia and how fast it is. Of course, I was skeptic and lazy to learn something new. On that time I was doing a simple bootstrap simulation for some testing problems. Of course, I was using **R**. But the issue I was facing is *every time I was experimenting with small changes it would take me one whole day to see the Monte Carlo simulation results*. This was so frustrating at some point I was thinking about using Python or **Rcpp**. I knew some Python and some C++ before. Now of course I am not claiming that my R code was fully high performance and was the best version of R code someone can write, but still I was not so happy with the performance. Personally I love coding in both R and Python. And honestly I still love R and Python but I think Julia has something exceptional to offer. Anyways, coming back to the story, the performance problems that I was facing on that time told me - *learning basic R is probably not enough for me and it seemed to me that I have to switch to Python or maybe use this C++ integration, Rcpp*. But somehow I was motivated by Tileman and wanted to try Julia. I found out that the syntax was easy and is similar to python in some way. So it did not take me a lot of time to write the code. Now here is the interesting part, why I was using R? The testing problem that I was facing had a *R package*. To do it in Julia I had two options, either I had to write everything from scratch, or I needed an equivalent package. Of course equivalent package was not an option because the problem was not a famous problem like linear regression or so, and writing everything from scratch was a real pain. However after googling, I found a package which implemented a similar function but not exactly same in Julia. So I was thinking, *I wish I could make this minor changes to this package*. What I could do after that was totally unexpected, I found that someone can go to the GitHub repo of this package and look at the function written in Julia. And it is possible to understand the code because it was written fully in Julia. This was completely mind-blowing for me, I was thinking *Yes! now I know how to implement this as well*. That's it after that I started to use Julia more or less always for my work whenever I can, of course not everyone adopted this yet. But I am hopeful soon this will change.

In R several times I wanted to look at the source code, but the best packages in R are usually written in C or Fortran code. And I am not good in either (I actually don't know Fortran!), so

this means I could never look into the original code someone wrote, and had no other option other than using the package blindly. However what blew my mind that for Julia, the code was written in Julia, I could look into the codes, copy it and then make some minor adjustments. This is what I did on that time, and Ah ha! "I have my function". And the best part is, it used to take me 10-15 minutes to do the whole simulation (also note my Julia code on that time was a complete mess!)

Okay so what are the morals of the story?

1. Julia is fast.
2. It solves the two language problem, for me it was clear.
3. Julia packages (most of the time, except for few exceptions) are written in pure Julia, and they are all freely available.
4. The last point means, for many there is no difference between being a Julia developer and Julia user (what a cool thing!)

Rough outline

I think we have enough motivation now, the serious students are probably asking themselves - *what are we doing here? Is this a story time? Tanvir, you are wasting our time?* So a sincere apology from my side to my serious friends, colleagues and students who are possibly more hardworking and energetic than me. So let's learn Julia. Here is a rough outline of the course.

- 1. A short tour of basic syntax
- 2. Generic programming
- 3. Type and type system in Julia
- 4. Type inference and type stability
- 5. Performance pitfalls or towards high performance code in Julia
- 6. Applications of some packages (Distributions, Dataframes, and maybe some Statistics packages)
- 7. Macro and Metaprogramming in Julia (???)
- 8. Parallel computing in Julia (???)

the "???", in brackets is there because I don't know how much time we will have at the end to cover these topics, but I will try, but if I fail, then I will try to give you enough resources so that you can learn them on your own.

About the course and philosophy

There are some important points I would like to discuss

- I would call this a **scientific programming course in Julia**. Maybe more appropriately we can say - *how to code in Julia and write fast codes in Julia*. This is all that is.

- But yes we will see some applications, but it will be limited. When it comes to applications, I feel it should be discussed from someone who have used Julia in that particular domain of applications and also he/she also has domain expertise. For example, maybe my physics friends are using differential equation solvers in Julia for different problems, and they are expert in that packages and also they are expert in the theory, so they should explain this, not me. If I try to do this, this would be a complete mess. Of course if time permits, I can show some applications where I used Julia, but not definitely something which I only have shallow idea.
-
- Extending on the last point, although I have some knowledge about Statistics (I am still learning), but I do not consider myself as an expert in machine learning, not even close. So I might show you some applications. Initially my plan was to study this during the prep time of this course, but I could not manage time to learn ML systematically. So I believe people who are using Julia every day for ML problems will be far more knowledgeable than I am, so even I show you something here, this will be very limited.
- Just wanted to say to the students, domain expertise is really crucial, and trying your best to know what you are doing is also really crucial. So be somewhat clear on what you are doing. This will be really important to apply good knowledge to practice. If you have bad foundations in theory, you cannot expect to apply the concept nicely. Sometimes things might happen by **fluke**, but often you might apply wrong methods in wrong places. Of course knowing good theory does not mean you have to reproduce all mathematical proofs at any time you are asked to do so. But it definitely means, you know the problem with a somewhat deeper understanding. You thought about this before and worked hard to get your basics clear, so now you have a feeling what may work and what may not. There will be always someone who will know and understand things better than you, but you can always try your best and understand things!
- Learning Julia for me was a bit *non-linear*. Meaning, for me, it was not always sequential, and probably I will have to follow the same non-linear style of teaching for the limitations of my knowledge and time constraints. So sometimes I will have to introduce some concepts vaguely to explain some things and then later on you will understand this better. Recently, I realized it is in fact possible to make it linear or to learn linear, but of course this will be seriously time consuming. And moreover this is more or less my recent discovery, when I was exposed to the branch of computer science - known as **programming language design**. So I recently came across this book [\[1\]](#), which might give you an idea what do I mean. But this book is too much theory, and to write a good program in Julia, you do not need it. I just mentioned the name for inquisitive readers. You can look this up later if you are really hooked in theory!

- I will share absolutely all materials from where I have learned Julia and used to make my lectures. But of course I was selective, and I would share in the lecture what I found interesting and useful. but surely I know that this may not be the best Julia course out there, so I want to give you everything that I know could be useful. Your task is to use these resources later on your own and build on it, and possibly learn better, be an expert and then teach better than me.
- Sometimes you can ask me something and I might have to say "I don't know" (just to tell you the limitations from my side). But of course if I don't know now then I will try my best to know in short future, given that the question is also intriguing to me.
- If all the above is ok for you, then **welcome to the course**.

Acknowledgements

I am thankful to everyone who helped me learn Julia, some of them are (and of course not limited to)

- My wife (who studied applied physics and scientific computing and still teaching me many things!)
- Tileman Conring (a great friend, and a previous TU Dortmund PhD student)
- Julia community in particular - David Sanders, Chris Rackauskas, Steve Johnson, and of course the co-creators Jeff Bezanson Stefan Karpinski, Viral B. Shah and Alan Edelman, for their amazing lectures in youtube.
- Dr. Carsten Bauer (Postdoctoral computational physicist at University of Cologne) (look at his personal web page)

Many of my materials are directly taken from their materials. Feel free to search more of their materials, and learn from them if you want.

References

1. Scott, M. L. (2016). Programming language pragmatics. Morgan Kaufmann. [↩](#)