7b. When To Optimize Code?

Martin Alfaro
PhD in Economics

INTRODUCTION

Julia has been praised as solving the "two-language problem". This refers to the difficulty of finding a language that's fast, but still easy to read and write. Although it's true that Julia has some advantages relative to other languages, claims like this can be quite misleading for someone new to programming—it wrongly suggests that Julia is the only language you'll need to learn, regardless of your specific coding domain.

In reality, each programming language is designed with certain purposes in mind. Consequently, it's quite likely that you'll need to learn multiple programming languages, even if your focus is narrow. This is particularly true in data analysis, where a package implementing a specific task may only be available in one language. I, for one, tend to use Julia as my main language for data analysis, but complement it with libraries from R and Python when the task requires it. [1]

Getting the best performance in any language is also not immediate. It requires you to write code appropriately, with implementations that tend to be software-specific and involve several trade-offs. [2] Overall, the claim that "Julia is fast" should be replaced by "Julia *can* be fast." Considering this, the upcoming chapters aim to equip you with the essential tools to unlock Julia's performance capabilities.

WHEN SHOULD WE CARE ABOUT SPEED?

Achieving high performance often comes with trade-offs, and thus should never be the sole consideration when writing code. Optimizing performance frequently means rewriting parts of your script, which can reduce readability and make the code harder to maintain in the long run. Additionally, implementing these improvements requires significant time and effort, including tasks such as testing, identifying bottlenecks, and integrating third-party packages.

Considering this, you should assess your goals before embarking on any optimization efforts. Keep in mind that **most of YOUR time will be spent on writing, reading, and debugging code**—reducing the computer's execution time by a millisecond may not be worth the trade-off if it demands investing hours. Moreover, even if speed is crucial for your project, you should prioritize which parts of the code to optimize. Typically, only a few operations impact runtime critically, with the rest having a negligible effect.

With these caveats in mind, the suggestions we'll present in the upcoming chapters serve a dual purpose. Firstly, they represent essential rules for speed—not adhering to them would severely undermine performance, thereby negating any advantages of using Julia. Secondly, several tips we'll

consider have a minimal impact on code's readability, if any. In summary, the procedures to be presented will help you unlock Julia's speed, without sacrificing code readability or entailing excessive additional work.

FOOTNOTES

[1.] Julia has the capacity of calling programs from other software such as R or Python. Python and R also have this feature.

[2.] This explains the disparate results often seen in online benchmarks, where code can be written inefficiently in one language and highly optimized in another. Moreover, since languages tend to excel at certain tasks, it's possible to cherry-pick examples that make a particular language appear faster.