Introduction to Julia Programming Language

Francesco Chiocchio

April 2023

Outline

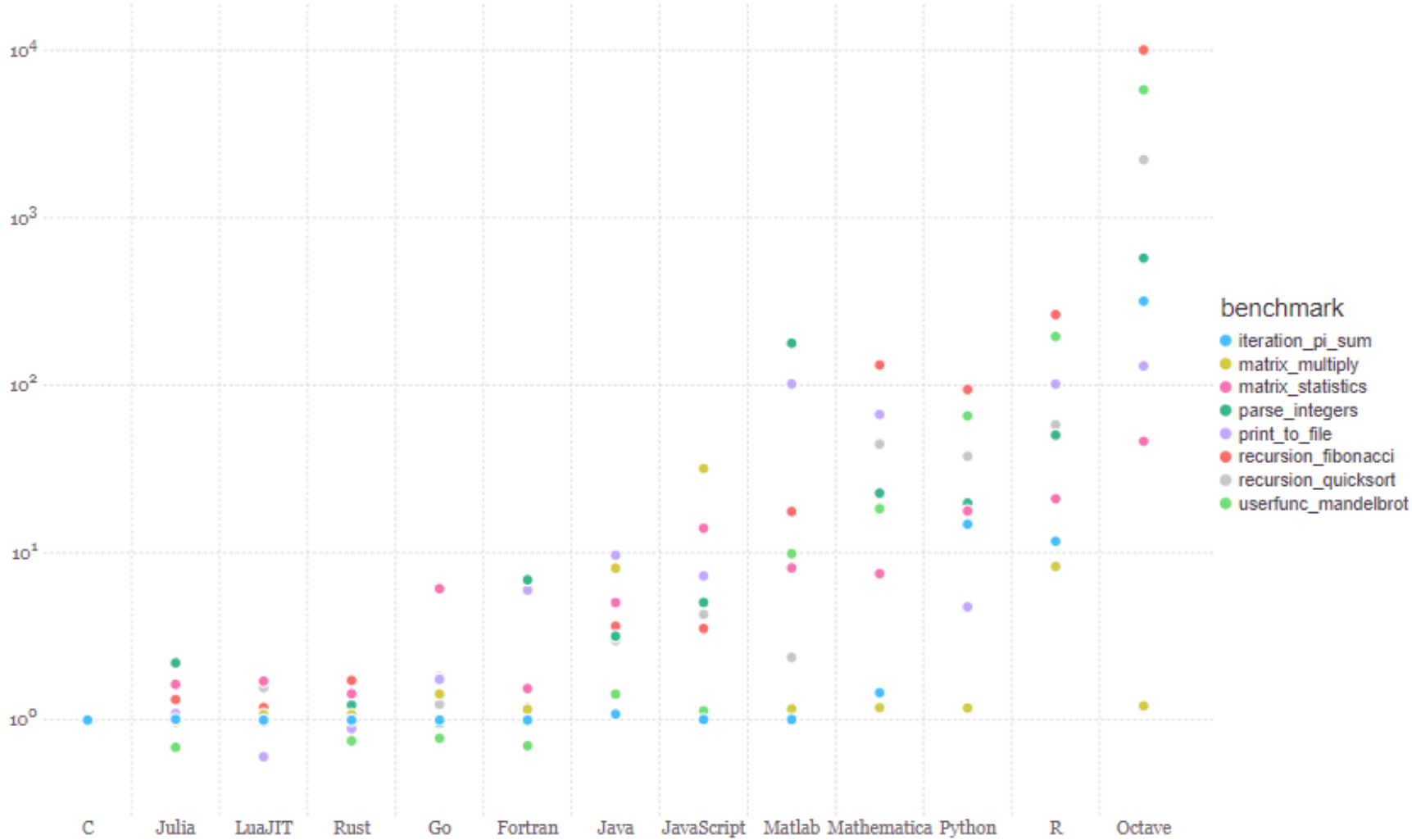
$\Rightarrow$ Aim for today: tools to go from Matlab to Julia

- Introduction to Julia - 30 minutes
- Examples of codes - 30 minutes
- **Key concepts:**
  - Types - More important than in matlab for efficient code
  - Scope of variables - local vs global
  - Be careful of copying arrays!
  - Functions - use them, parameters as argument

Introduction I

- What is Julia?
  - Programming language
  - High level - Uses words
  - Dynamic - Variable types are assigned and checked at runtime, no need to declare variable
- Why should we learn Julia?
  - Faster than Matlab, Python or R
  - Much easier to learn and use than Fortran or C
    - We spend majority of time coding, not running codes
  - Open source

Speed Comparison Across Languages



- Taken from here

Introduction II

- How can Julia achieve this speed?
  - Language: designed for Just In Time compilation
  - Multiple Dispatch - Central for Julia
  - Outside of scope of this lecture. Refer to this video for a long introduction
- How to learn?
  - Julia Documentation
  - QuantEcon
  - Youtube Channel
  - Cheat Sheet
  - Practice, google, forums, patience and exercises

Programming Environments vs Languages

- Matlab is an Integrated Development Environment
  - Text editor
  - Debugging support
  - Memory/workspace introspection ...
- Julia is a programming language
  - REPL - Read Evaluate Print Loop
  - Text editors - Write the code, save it, run it separately - VIM
  - IDE - Text editing combined with running the code - VScode
  - Notebooks - Pluto.jl - check Joel's GitHub
- I will use VScode. More info here
  - Closer to an IDE
  - Example 1: Open REPL, Open VScode, Save file

Key characteristics of Julia

- Packages and import
- Variable and Types
- Functions
- Debugging
- Macros

Packages

- Extend the functionality of Julia
  - You will need to use many of them!
- Read and understand what they do by going on their documentation and on github
- Install them using the Package manager
- Activate them in your program with `using`
- Specific commands with import
- Make sure that they are up to date!
- Download the QuantEcon Package!

$\Rightarrow$ Example 1

Unlike Matlab, without packages you will not do a great deal in Julia! Learn to find, understand and use them. More packages do the same thing (think of plot) take your time in understanding which one fits better your code

Variables and Types: Basic

- You create variables:
  - Store information
  - Manipulate them
  - Compare them
- Variables can be of **Type** Boolean, Integer, Float, String, ...
- Dynamic language: No need to declare them! Infers them
- Check the type of a variable using the `typeof()` function
- Try to make your code **Type stable**

$\Rightarrow$ Example 2 Part 1

Variable creation not very different from Matlab. Understanding types is important to understand errors, the Julia lexicon, and write good codes.

Variables and Types: Arrays

- Most common method of storing information
- They can have any dimension:
  - Arrays with 1 dimension can be interpreted as column vectors
  - Matrix is just a an array with more dimensions

$\Rightarrow$ Example 2 Part 2

- Array indexing similar to Matlab
- Be careful of column or row vector
- You need to use the funcion copy to create a new array that is equal to an old array!
- Note the difference between slices and view: The latter does not create more allocations

Variables and Types: Tuples

- Types of containers: contain collection of date
- Use these to input the parameters and use as argument in all functions
  - Enable you to not use global variables - We will talk about this more later on!
- Use the Parameters package to unpack tuples efficiently and give them defaults values!

$\Rightarrow$ Example 2 Part 3

Functions: General

- Functions are a key aspect of Julia! You will use and create many of them
  - Easy to change/debug
  - Code is clearer
  - Can make your code a lot faster
- Functions can be arguments of other functions
- Functions can be one liners or anonymous
- Variable created in functions will not exist outside of them - Global versus local scope

$\Rightarrow$ Example 3 part 1 shows how to create functions and some example, but there is a lot more!

- Understand how to name, define, put the arguments of functions. And how to return the values

Functions: Broadcasting and Comprehensions

- **Broadcasting:** Apply a function or an operator to each element!
  - More than just elementwise operation
  - It applies to user-defined functions as well
  - A convenience macro for adding broadcasting on every function call is @.
- **Comprehension** are basically loops in brackets
  - Convenient way of creating variable

$\Rightarrow$ Example 3 part 2

- Not very different from Matlab, just understand the vocabulary!
- They should be as fast as loops! But you can create a code to check for yourself.

Functions: Multiple-Dispatch and notes

- Conventions:
  - Name function with small letters
  - Write all your intermediate steps in functions
  - All functions for your project saved in a different file and include them in the main file with `include`
  - Functions which modify any of the arguments have the name ending with ! (in-place functions)
- Multiple-Dispatch:
  - Functions have different behavior depending on the type of the arguments - **different methods**
  - Will not affect your life, but interesting to see
  - Declaring types of your arguments is usually not important and no speed gain!
- main() function that encapsulates all your code
  - Making everything in local scope!
  - **Next and most important topic!**

$\Rightarrow$ Example 3 part 3

Scope of variables

- Region of code within which a variable is visible
  - Variable scoping helps avoid variable naming conflicts
- Local Variables
  - Variables defined inside a function, for, while, let, comprehensions, broadcast are local
  - Local variables only exist in the program where they are defined
  - Variables that are not global or are not function arguments are undefined inside a function
- Global Variables
  - Never erased, but they can be **overwritten**
  - Slow and more difficult to track down bugs

$\Rightarrow$ Example 4 to understand the importance of scope!

Debugging

- Use debug tool, display intermediate steps, short loops, check output of every function (also in global scope!)
- Learn to read the error message!
  - Where is the problem? Is there a problem with the argument? Are there cases that I am not considering? Did I forget to do a check on the input variables? Use different inputs as checks (for instance 0, 1, -1, Inf, nothing, missing,...maybe you forgot to check!)
  - Do a lot of testing, use **macros** (@, next slides!)
- Check documentation of the function, check online and ask questions!
  - Stack overflow, Julia discourse
  - Search by using error message, and correct vocabulary! i.e. Tuple, Array, Method

$\Rightarrow$ Example 5 to see how I wasted my weekend!

Macros

- Just a way to automatically write code that you could have written out by hand anyway. They come before the code with a @ (as we have seen!)
- Some macros are useful to debug, increase speed, show results, test (list here, google them!)
  - show, eval, views, assert
  - with_kw, unpack
  - time, btime, benchmark, allocate
  - methods(not a macro), which
  - code_warntype, edit, code_lowered

Programming

- In general:
  - Think before programming: pen and paper, whiteboard, small codes
  - Clean code: write functions, do not repeat operations, short comments, clear variable names, proper indentation
- For Julia:
  - Careful with the use of global and constants: use tuples for parameters
  - Careful with type stability
  - Efficiency: Read the Julia Performance Tips here
  - Use profiling tools: here (I have never done it)
  - Display only relevant information

$\Rightarrow$ **You are not programmers, you are economists. Invest more time in economics than in programming!**