Julia tutorial

- Introduction
- Julia tutorial
- JuMP tutorial
- Markdown tutorial
- Plotting tutorial
- Submitting assignments

Why this tutorial?

To give you the resources and tools necessary to learn Julia, IJulia, and JuMP quickly and efficiently.

- Most of the learning will happen on your own as you work on homework assignments and the project
- The goal of this tutorial is to make that learning easy
- This tutorial was written on 1/14/2024. Current versions: Julia: 1.10.0, IJulia: 1.24.2, JuMP: 1.18.1.
  Things change quickly!

What is Julia?

- A modern programming language developed for scientific computing. Created in 2012 by a group of MIT students.
- **features**
  - designed for scientific computing but with the functionality of a modern object-oriented programming languages
  - simple efficient syntax similar to Matlab
  - dynamic language with speed comparable to statically compiled languages (e.g. C)
  - designed for parallelism and distributed computation
  - can call Python and C functions directly
  - IJulia notebooks
  - free and open-source

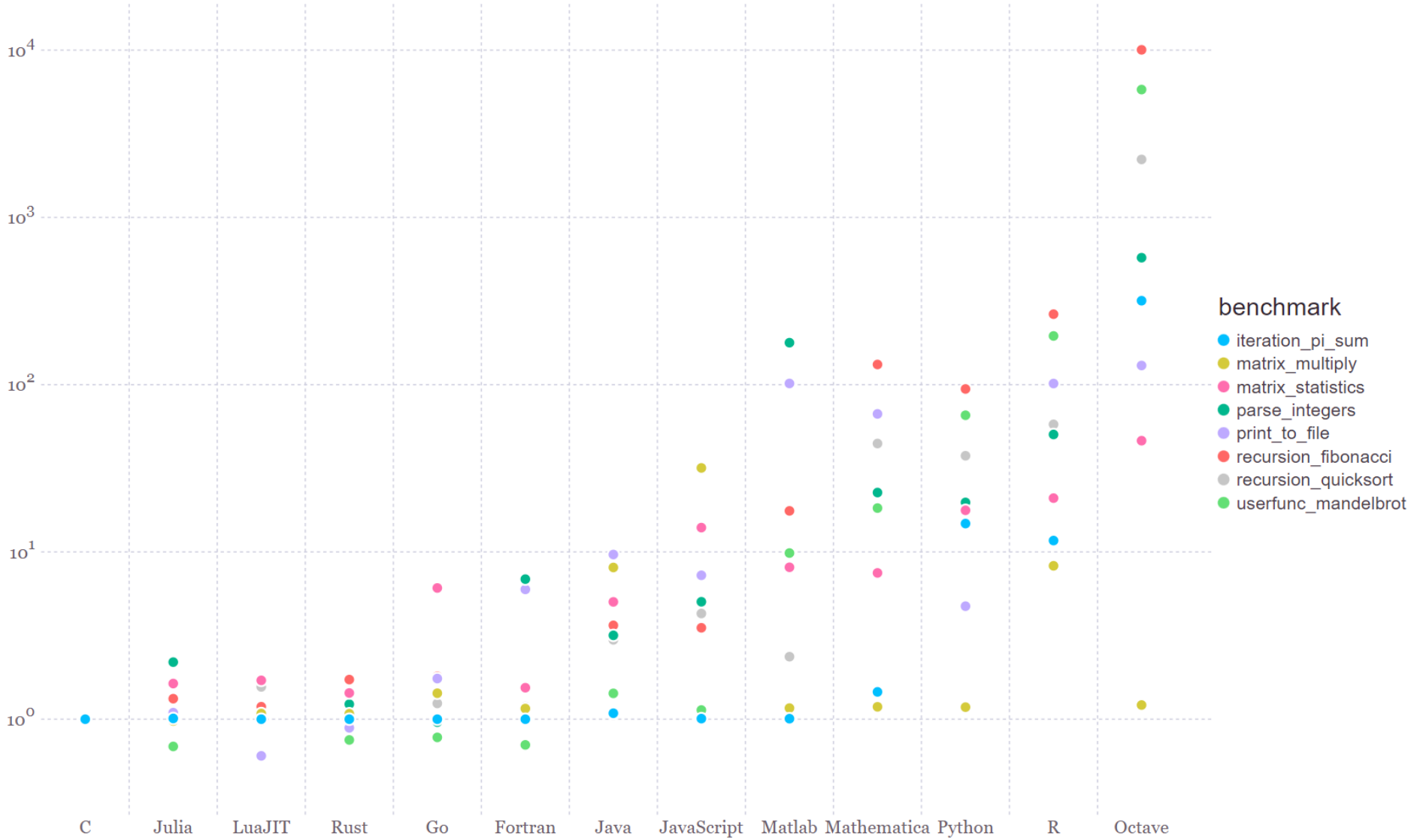
What is Julia?

Most engineers employ a "two-language solution"

- Use an interpreted language (Matlab, Python, R, etc.) for exploration and prototyping. Useful because it is easy to write, debug, design, visualize, and iterate.
- Use a compiled language (C, Fortran, etc.) when high-level details have been worked out and you need high performance code for a large-scale implementation or experiment.

Julia aims to be a one-language solution.

Benchmarks



source: https://julialang.org/benchmarks/

What is Julia?

- **disadvantages**
  - use is not widespread. Engineering community uses mostly Matlab, data science community uses mostly Python or R.
  - does not have the same level of support as other languages. Julia community is small (but very active!)
  - Julia is still rapidly evolving, with new features being added and old features being changed on a regular basis.
  - debugging support is weak (but improving).
  - weak toolboxes and packages compared to other languages. If you want something, you might have to code it yourself!

Overview

Short guides:
Installing Julia and Getting started

This tutorial:

- Julia tutorial
- JuMP tutorial
- Markdown tutorial
- Plotting tutorial

Julia tutorial

Julia tutorial.ipynb

- Notebook navigation
- Comments: `#` and `#= ... =#`.
- Types (Int, Float, String, Symbol), the `typeof` command
- Unicode, e.g. `\pi[TAB]` (I'll probably regret this...)
  http://docs.julialang.org/en/stable/manual/unicode-input/
- Ranges, arrays, and indexing. Stack operations.
- Tuples, dictionaries, and sets.
- Flow control (loops)
- Functions.

JuMP tutorial

JuMP tutorial.ipynb

- Modeling optimization problems
- Variables
- Constraints
- Objectives
- Solving and interpreting output

Markdown tutorial

https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

- Headings
- Lists and tables
- Images and hyperlinks
- Equations
- LaTeX (pronounced lay-tek) cheat sheet:
  http://users.dickinson.edu/~richesod/latex/latexcheatsheet.pdf
  detexify: http://detexify.kirelabs.org/classify.html

Plotting tutorial

Plotting tutorial.ipynb

Two methods currently available:

1. **The `PyPlot` package**
   - Identical to Python's `matplotlib` library.
   - Plotting syntax similar to matlab.
   - Resize your figure! `figure(figsize=(x,y))`

Basic usage: https://github.com/JuliaPy/PyPlot.jl
Examples: https://gist.github.com/gizmaa/7214002
matplotlib manual: http://matplotlib.org/

Plots in Julia

2. The `Plots` package

- Is being pushed as the defacto plotting package for Julia.
- Can create animations, save as animated gif!
- It's a front end that can call other plotting libraries using a unified syntax. Of course, has its own syntax...

Usage and examples: https://juliaplots.github.io/

Submitting assignments

- For homeworks, you will submit a single PDF file for each assignment. You can submit: a notebook, something hand-written, something typeset, or a combination. Either way, include your code + solutions, and submit a PDF.
  - To convert a Jupyter notebook to PDF, I recommend to "save and export as HTML" and then use your browser's print feature to print the HTML to PDF.
  - Alternative approach: Directly "print to PDF" in the browser window where you're running your Jupyter notebook. This can sometimes be buggy, but if it works, you can save yourself a step.
- For the final project, you will turn in your `.ipynb` notebook file along with any supplemental attachments called by the code (data files, images, etc.). Instructions to come later.

Final comments

- Go through the "learn X in Y minutes" tutorial yourself (sections 1–4). It's definitely worth it.
- For more complicated aspects we discussed, it's ok to "learn as you go". Just make sure to start your assignments early to give yourself time to debug, learn syntax, deal with random printing bugs, etc.
- Julia is rapidly evolving. If you find a better way to do something in Julia/IJulia/JuMP than how I've been teaching it, please let me know!