A brief introduction to Julia

Alexandre Prieur

AstroCalcul Juin 2024

IMCCE Observatoire de Paris | PSL

Introduction

What is Julia?

julia is a high-level, high-performance dynamic language for technical computing.

What is Julia?

```julia
α = 2π
for i in 1:2
    if exp(i * α * im) ≈ 1
        print("Yey!")
    else
        print("Oh no...")
    end
end
```

What is Julia?

julia *is a high-level, high-performance dynamic language for technical computing.*

What is Julia?

julia *is a high-level, high-performance dynamic language for technical computing.*

- Easy to learn and use

What is Julia?

julia *is a high-level, high-performance dynamic language for technical computing.*

- Easy to learn and use
- Fast code

What is Julia?

julia *is a high-level, high-performance dynamic language for technical computing.*

- Easy to learn and use
- Fast code
- Compiled just-in-time

Some dates

- 2012: first release
- 2018: 1.0 release
- Current: 1.10.4

Some dates

- 2012: first release
- 2018: 1.0 release
- Current: 1.10.4

Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
2018								1.0				
2019	1.1							1.2			1.3	
2020			1.4					1.5				
2021					**1.6**						1.7	
2022								1.8				
2023					**1.9**							1.10

Table 1: Julia minor releases

[Interactive] Diving in the code

TO DO in the interactive session

- Installation

```
    curl -fsSL https://install.julialang.org | sh
```

TO DO in the interactive session

- Installation

```
curl -fsSL https://install.julialang.org | sh
```

- Basic manip
- Packages and environments
- REPL modes: code, package, help, terminal
- Quick look at Pluto, VS Code
- Showcase of JIT

Prime spiral:

`https://www.3blue1brown.com/lessons/prime-spirals`

Capacities of Julia

Strong points – Easy and fast

	Coding	Execution
Python, R, …	Fast	Slow
C, FORTRAN, …	Slow	Fast

Table 2: The two language problem

Strong points – Easy and fast

	Coding	Execution
Python, R, ...	Fast	Slow
C, FORTRAN, ...	Slow	Fast
Julia*	Fast	Fast

Table 2: The two language problem

Strong points – Easy and fast

	Coding	Execution
Python, R, …	Fast	Slow
C, FORTRAN, …	Slow	Fast
Julia*	Fast	Fast

Table 2: The two language problem

*Although it creates the 1.5 language problem

Strong points – Easy and fast

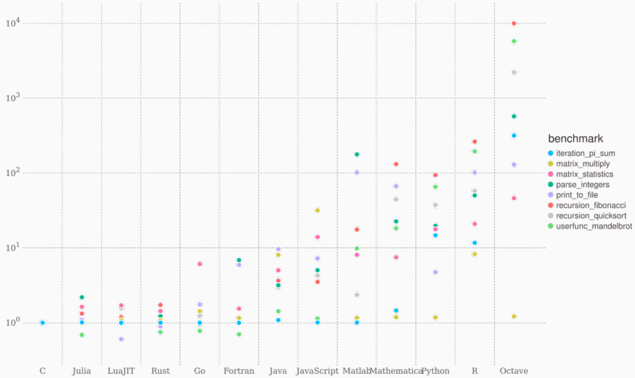


Figure 1: Micro-benchmarks, source: `https://julialang.org/benchmarks/`.

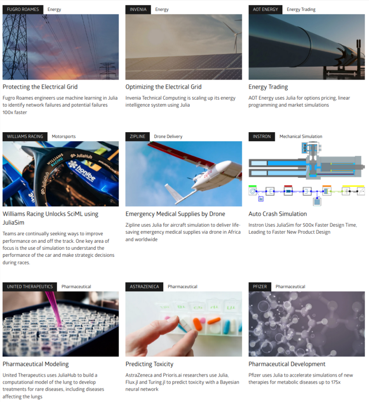
Strong points – Fluid coding

- Unicode support, clear syntax
- Eliminates points of friction
  - `help?>`, `@time`, `@edit`
  - Features-packed REPL
  - Packages, environments : `juliaup`, `Pkg.jl`

Strong points – Community

- Online community: `https://discourse.julialang.org`
- 100k+ available libraries
- State-of-the art in: ODE, ML, ...

Strong points – Community



FUGRO ROAMES Energy

Protecting the Electrical Grid

Fugro Roames engineers use machine learning in Julia to identify network failures and potential failures 100x faster

INVENIA Energy

Optimizing the Electrical Grid

Invenia Technical Computing is scaling up its energy intelligence system using Julia

AOT ENERGY Energy Trading

Energy Trading

AOT Energy uses Julia for options pricing, linear programming and market simulations

WILLIAMS RACING Motorsports

Williams Racing Unlocks SciML using JuliaSim

Teams are continually seeking ways to improve performance on and off the track. One key area of focus is the use of simulation to understand the performance of the car and make strategic decisions during races.

ZIPLINE Drone Delivery

Emergency Medical Supplies by Drone

Zipline uses Julia for aircraft simulation to deliver life-saving emergency medical supplies via drone in Africa and worldwide

INSTRON Mechanical Simulation

Auto Crash Simulation

Instron Uses JuliaSim for 500x Faster Design Time, Leading to Faster New Product Design

UNITED THERAPEUTICS Pharmaceutical

Pharmaceutical Modeling

United Therapeutics uses JuliaHub to build a computational model of the lung to develop treatments for rare diseases, including diseases affecting the lungs

ASTRAZENECA Pharmaceutical

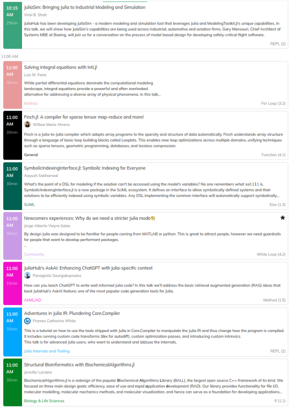
Predicting Toxicity

AstraZeneca and Prioris.ai researchers use Julia, Flux.jl and Turing.jl to predict toxicity with a Bayesian neural network

PFIZER Pharmaceutical

Pharmaceutical Development

Pfizer uses Julia to accelerate simulations of new therapies for metabolic diseases up to 175x



10:15 AM 25min — JuliaSim: Bringing Julia to Industrial Modeling and Simulation

Viral B. Shah

JuliaHub has been developing JuliaSim - a modern modeling and simulation tool that leverages Julia and ModelingToolkit.jl's unique capabilities. In this talk, we will show how JuloSim's capabilities are being used across industrial, automotive and aviation firms. Gary Mansouri, Chief Architect of Systems MBE at Boeing, will join us for a conversation on the process of model based design for developing safety-critical flight software.

REPL (2)

11:00 AM

11:00 AM 30min — Solving integral equations with Inti.jl

Luiz M. Faria

While partial differential equations dominate the computational modeling landscape, integral equations provide a powerful and often overlooked alternative for addressing a diverse array of physical phenomena. In this talk...

Math(s) — For Loop (3.2)

11:00 AM 30min — Finch.jl: A compiler for sparse tensor map-reduce and more!

Willow Marie Ahrens

Finch is a Julia-to-Julia compiler which adapts array programs to the sparsity and structure of data automatically. Finch understands array structure through a language of basic loop building blocks called Looplets. This enables new loop optimizations across multiple domains, unifying techniques such as sparse tensors, geometric programming, databases, and lossless compression.

General — Function (4.1)

11:00 AM 30min — SymbolicIndexingInterface.jl: Symbolic Indexing for Everyone

Aayush Sabharwal

What's the point of a DSL for modeling if the solution can't be accessed using the model's variables? No one remembers what `sol[1]` is. SymbolicIndexingInterface.jl is a new package in the SciML ecosystem. It defines an interface to allow symbolically defined systems and their solutions to be efficiently indexed using symbolic variables. Any DSL implementing the common interface will automatically support symbolically...

SciML — Else (1.3)

11:00 AM 30min — Newcomers experiences: Why do we need a stricter Julia mode🧐 ★

Jorge Alberto Vieyra Salas

By design Julia was designed to be familiar for people coming from MATLAB or python. This is great to attract people, however we need guardrails for people that want to develop performant packages.

...

Community — While Loop (4.2)

11:00 AM 10min — JuliaHub's AskAI: Enhancing ChatGPT with julia-specific context

Panagiotis Georgakopoulos

How can you teach ChatGPT to write well informed Julia code? In this talk we'll address the basic retrieval augmented generation (RAG) ideas that back JuliaHub's AskAI feature; one of the most popular code generation tools for Julia.

AI/ML/AD — Method (1.5)

11:00 AM 30min — Adventures in Julia IR: Plundering Core.Compiler

Frames Catherine White

This is a tutorial on how to use the tools shipped with Julia in Core.Compiler to manipulate the Julia IR and thus change how the program is compiled.
It includes running custom code transforms (like for autodiff), custom optimization passes, and introducing custom intrinsics.
This talk is for advanced Julia users, who want to understand and (ab)use the internals.

Julia Internals and Tooling — REPL (2)

11:00 AM 30min — Structural Bioinformatics with BiochemicalAlgorithms.jl

Jennifer Leclaire

BiochemicalAlgorithms.jl is a redesign of the popular **B**iochemical **A**lgorithms **L**ibrary (BALL), the largest open source C++-framework of its kind. We focused on three main design goals: efficiency, ease of use and **r**apid **a**pplication **d**evelopment (RAD). Our library provides functionality for file I/O, molecular modelling, molecular mechanics methods, and molecular visualization, and hence can serve as a foundation for developing applications...

Biology & Life Sciences — If (1.1)

Figure 2: Left: `https://info.juliahub.com/case-studies`.
Right: JuliaCon 2024 talks, Wednesday morning.

Strong points – Extensibility

- Composability with other languages: `ccall`, `fcall`, `pycall`, `rcall`…
- Massive code re-use within Julia: multiple dispatch!
- Developing a package is easy (pkg creation, Julia written in Julia)

Strong points – Reproducibility

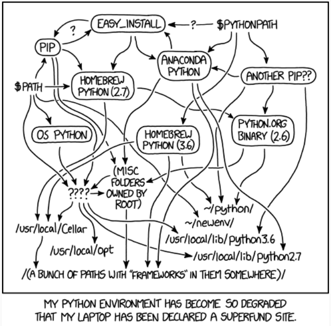


Figure 3: Python Environment (`https://xkcd.com/1987/`)

Strong points – Reproducibility

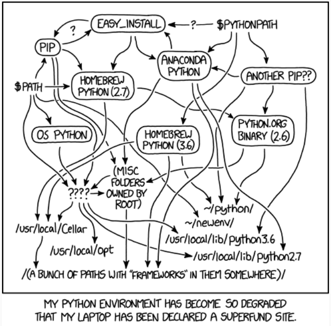


Figure 3: Python Environment (`https://xkcd.com/1987/`)

- Julia: `juliaup`, `Pkg.jl`, `pkg>` mode...
- Readability

[Interactive] optimizing Julia code

TO DO during interactive optimisation

- Key points:
  - Naive code can be very slow: follow simple rules !
  - Optimize *when you need to*

TO DO during interactive optimisation

- Key points:
  - Naive code can be very slow: follow simple rules !
  - Optimize *when you need to*
- Two critical points: type stability, and memory allocation!
- Secondary aspects: eg. encapsulating in functions

TO DO during interactive optimisation

- Key points:
  - Naive code can be very slow: follow simple rules !
  - Optimize *when you need to*
- Two critical points: type stability, and memory allocation!
- Secondary aspects: eg. encapsulating in functions
- Cf. `https://docs.julialang.org/en/v1/manual/performance-tips/`

TO DO during interactive optimisation

- Key points:
  - Naive code can be very slow: follow simple rules !
  - Optimize *when you need to*
- Two critical points: type stability, and memory allocation!
- Secondary aspects: eg. encapsulating in functions
- Cf. `https://docs.julialang.org/en/v1/manual/performance-tips/`
- Tools!
  - `@time` / BenchmarkTools.jl / Profiler or VS Code profiler
  - `@code_warntype` / JET.jl / Cthulhu.jl
  - AllocationCheck.jl

Advanced Julia concepts

- Just-in-time compilation
- Multiple dispatch
- Metaprogramming
- And much more!

Conclusion

Should you switch to Julia?

Comparing to other languages

Compared to...	Julia is
Python	Faster, one language does it all

Comparing to other languages

Compared to...	Julia is
Python	Faster, one language does it all
MATLAB	FOSS, easier, more general

Comparing to other languages

Compared to...	Julia is
Python	Faster, one language does it all
MATLAB	FOSS, easier, more general
C	Faster, quicker to write

Comparing to other languages

Compared to...	Julia is
Python	Faster, one language does it all
MATLAB	FOSS, easier, more general
C	Faster, quicker to write
TRIP	Released 🙃

Comparing to other languages

Compared to...	Julia is
Python*	Faster, one language does it all
MATLAB	FOSS, easier, more general
C*	Faster, quicker to write
TRIP	Released 🙃

*You can even call these from Julia without overhead!

Use Julia if...

- You want to quickly write code

Use Julia if...

- You want to quickly write code
- You want to write quick code

Use Julia if...

- You want to quickly write code
- You want to write quick code
- You want to use modern features and QOL

Use Julia if...

- You want to quickly write code
- You want to write quick code
- You want to use modern features and QOL
- You value free and open-source software (FOSS)

Use Julia if...

- You want to quickly write code
- You want to write quick code
- You want to use modern features and QOL
- You value free and open-source software (FOSS)
- You want to look cool 😎

Caveats

- 1.5 language problem
- Language is still evolving
- Can do some things I'm not fond of (general metaprogramming)
- Large compiled binaries
- Subpar static analysis
- Large memory consumption

Why do *I* use Julia?

- Automatic differentiation
- Convinced by the advantages...

Why do *I* use Julia?

- Automatic differentiation
- Convinced by the advantages…
- And can contribute to fixing the flaws!

Resources – some tools

IDE REPL, `Pluto.jl`, VS Code...	

Resources – some tools

IDE REPL, `Pluto.jl`, VS Code...	Visualisation `Makie.jl`, `Plots.jl`...

Resources – some tools

IDE REPL, `Pluto.jl`, VS Code…	Visualisation `Makie.jl`, `Plots.jl`…
HPC `Threads.jl`, `Distributed.jl`, JuliaGPU…	

Resources – some tools

IDE REPL, `Pluto.jl`, VS Code...	Visualisation `Makie.jl`, `Plots.jl`...
HPC `Threads.jl`, `Distributed.jl`, JuliaGPU...	Package dev `Revise.jl`, `PkgTemplates.jl` ...

Resources – useful links

- MIT course: `https://computationalthinking.mit.edu/Spring21/`
- High-speed Julia: `https://gdalle.github.io/JuliaPerf-CERMICS/` and its references
- Performance tips: `https://docs.julialang.org/en/v1/manual/performance-tips/`

Resources – general

- Discourse: `https://discourse.julialang.org`
- Slack/Zulip
- Docs: `https://docs.julialang.org/en/v1/`

References

- Bezanson, Jeff et al. “Julia: A fresh approach to numerical computing”. In: *SIAM review* 59.1 (2017), pp. 65–98. URL: `https://doi.org/10.1137/141000671`.
- *Datseris/whyjulia-manifesto: Zenodo-citable release.* [Online; accessed 21. Mar. 2024]. Mar. 2024. DOI: `10.5281/zenodo.10252527`.

Thank you for listening!

Any questions?

Multiple dispatch - C code

```cpp
class Pet {
    public:
        string name;
};
string meets(Pet a, Pet b) { return "FALLBACK"; }

void encounter(Pet a, Pet b) {
    string verb = meets(a, b);
    cout << a.name << " meets " << b.name
         << " and " << verb << endl;
}
```

Multiple dispatch - C code 2

```
class Dog : public Pet {};
class Cat : public Pet {};

string meets(Dog a, Dog b){ return "sniffs"; }
string meets(Dog a, Cat b){ return "chases"; }
string meets(Cat a, Dog b){ return "hisses"; }
string meets(Cat a, Cat b){ return "slinks"; }
```

Multiple dispatch - C code 3

```c
int main() {
    Dog fido; fido.name = "Fido";
    Dog rex; rex.name = "Rex";
    Cat whiskers; whiskers.name = "Whiskers";
    Cat spots; spots.name = "Spots";

    encounter(fido, rex);
    encounter(fido, whiskers);
    encounter(whiskers, rex);
    encounter(whiskers, spots);

    return 0;
}
```

Multiple dispatch - Julia code

```julia
abstract type Pet end
struct Dog <: Pet; name::String end
struct Cat <: Pet; name::String end

function encounter(a::Pet, b::Pet)
    verb = meets(a, b)
    println("$(a.name) meets $(b.name) and $verb")
end

meets(a::Dog, b::Dog) = "sniffs"
meets(a::Dog, b::Cat) = "chases"
meets(a::Cat, b::Dog) = "hisses"
meets(a::Cat, b::Cat) = "slinks"
```

Multiple dispatch - Julia code 2

```julia
fido = Dog("Fido")
rex = Dog("Rex")
whiskers = Dog("Whiskers")
spots = Dog("Spots")

encounter(fido, rex)
encounter(fido, whiskers)
encounter(whiskers, rex)
encounter(whiskers, spots)
```

Multiple dispatch - results

```
$ julia pets.jl
Fido meets Rex and sniffs
Fido meets Whiskers and chases
Whiskers meets Rex and hisses
Whiskers meets Spots and slinks

$ clang++ pets.cxx -o pets
$ ./pets
Fido meets Rex and FALLBACK
Fido meets Whiskers and FALLBACK
Whiskers meets Rex and FALLBACK
Whiskers meets Spots and FALLBACK
```