RSDG @ UCL

Julia: A Fresh Approach to Numerical Computing

Mosè Giordano

@giordano m.giordano@ucl.ac.uk

IHI Code Club

June 24, 2020

Julia's Facts

- v1.0.0 released in 2018 at UCL
- Development started in 2009 at MIT, first public release in 2012
- Julia co-creators won the 2019 James H. Wilkinson Prize for Numerical Software
- Julia adoption is growing rapidly in numerical optimisation, differential equations, machine learning, differentiable programming
- It is used and taught in several universities (`https://julialang.org/learning/`)

Julia on Nature

TOOLBOX · 30 JULY 2019

Julia: come for the syntax, stay for the speed

Researchers often find themselves coding algorithms in one programming language, only to have to rewrite them in a faster one. An up-and-coming language could be the answer.

Jeffrey M. Perkel

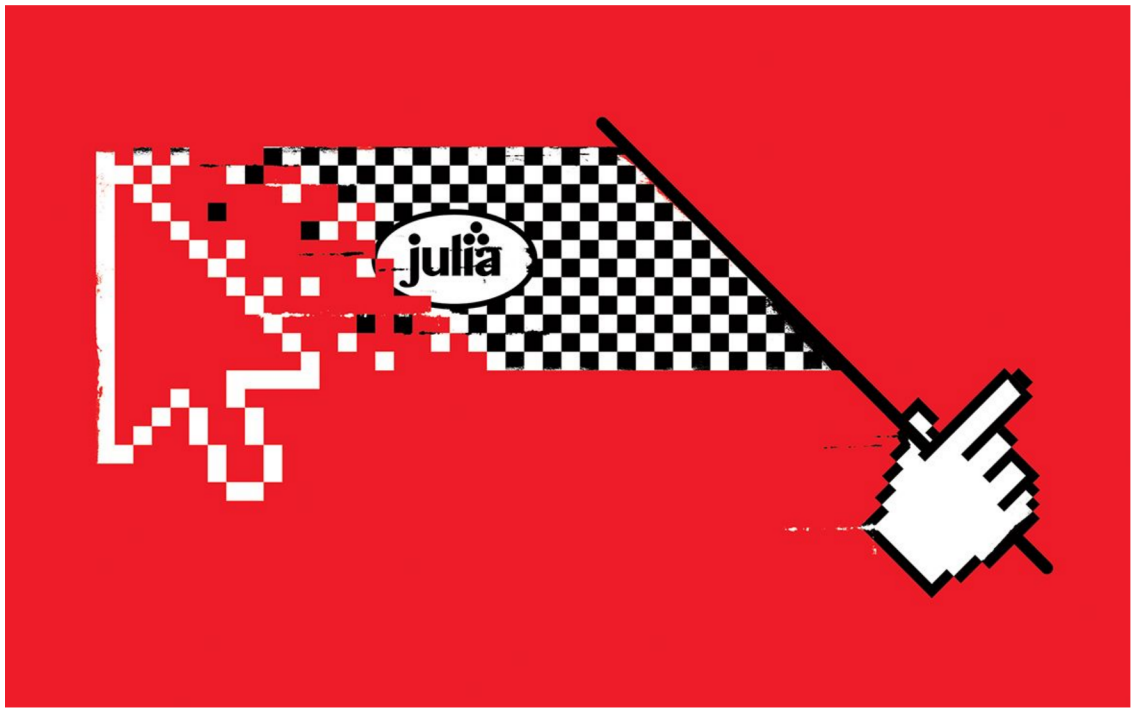
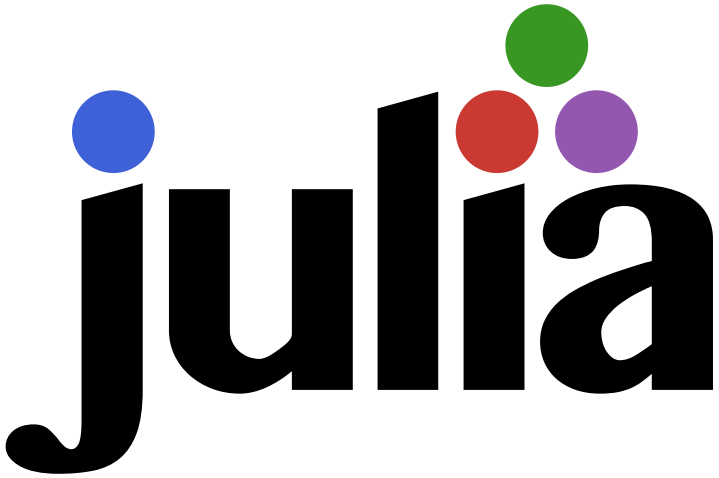


Illustration by The Project Twins

Nature **572**, 141-142 (2019). DOI: 10.1038/d41586-019-02310-3

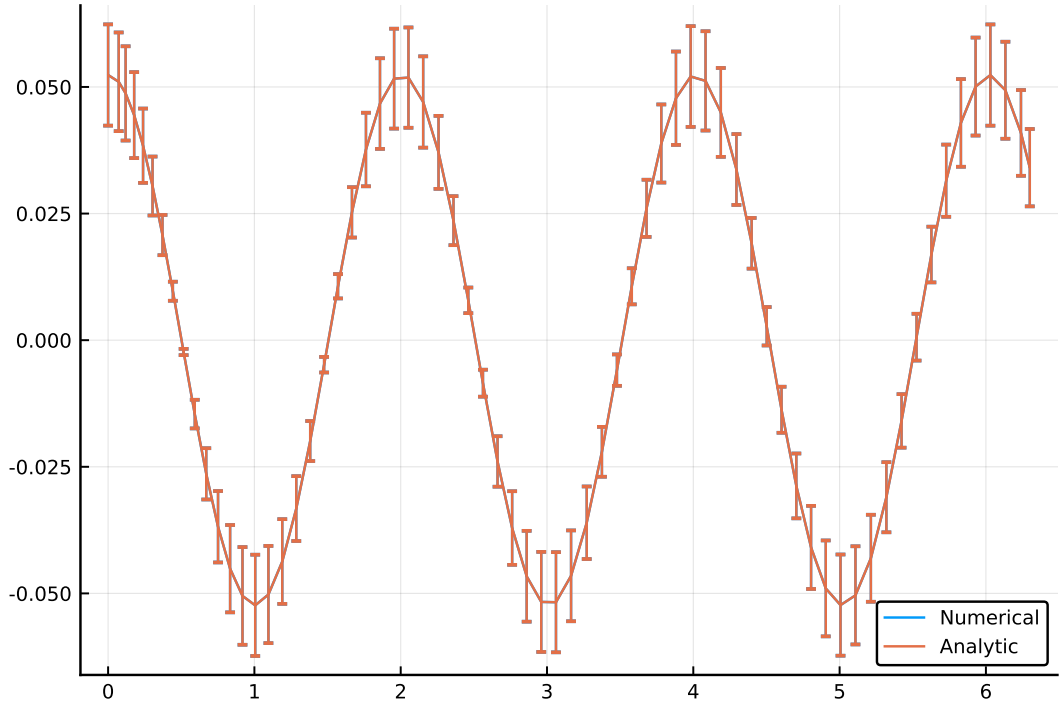
Solving the Two-Language Problem: Julia



- Multiple dispatch
- Dynamic type system
- Good performance, approaching that of statically-compiled languages
- JIT-compiled scripts
- User-defined types are as fast and compact as built-ins
- No need to vectorise: `for` loops are fast
- Garbage collection: no manual memory management
- Interactive shell (REPL) for exploratory work
- Call C and Fortran functions directly: no wrappers or special APIs
- Call Python functions: use the `PyCall` package
- Designed for parallelism and distributed computation

Multiple Dispatch

```
using DifferentialEquations, Measurements,
      Plots

g = 9.79 ± 0.02; # Gravitational constant
L = 1.00 ± 0.01; # Length of the pendulum

# Initial speed & angle, time span
u0 = [0 ± 0, π / 60 ± 0.01]
tspan = (0.0, 6.3)

# Define the problem
function pendulum(du,u,p,t)
    θ  = u[1]
    dθ = u[2]
    du[1] = dθ
    du[2] = -(g/L)*θ
end

# Pass to solvers
prob = ODEProblem(pendulum, u0, tspan)
sol = solve(prob, Tsit5(), reltol = 1e-6)

# Analytic solution
u = u0[2] .* cos.(sqrt(g / L) .* sol.t)

plot(sol.t, getindex.(sol.u, 2),
     label = "Numerical")
plot!(sol.t, u, label = "Analytic")
```



From `DifferentialEquations.jl` tutorial "Numbers with Uncertainties", by Mosè Giordano & Chris Rackauckas

JuliaCon 2019 talk "The Unreasonable Effectiveness of Multiple Dispatch": https://www.youtube.com/watch?v=kc9HwsxE10Y

Best Programming Practices

- Packages are git repositories
- Testing framework in standard library
- Continuous integration with several different services (Travis, AppVeyor, Cirrus, Drone, Gitlab Pipelines, Azure Pipelines, GitHub Actions, etc...)
- Code coverage: Coveralls, Codecov
- Documentation: docstrings, doctests
- PkgEval: test all registered packages

Tutorial on how to develop Julia packages:
`https://www.youtube.com/watch?v=QVmU29rCjaA`

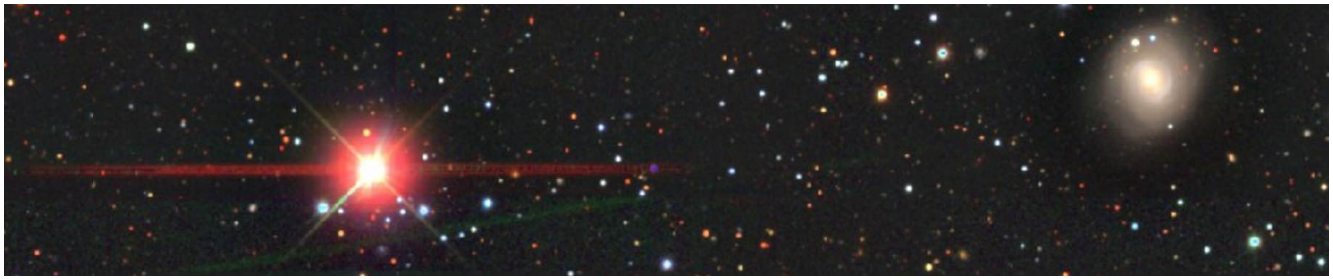
What's Bad About Julia

JuliaCon 2019 talk: `https://www.youtube.com/watch?v=TPuJsgyu87U`

What's Bad About Julia (cont.)

- Compilation latency can be annoying during development
- Plotting framework not exciting
- Ecosystem still young

Applications: Past – `Celeste.jl`



Project goals:

1. Catalog all galaxies and stars that are visible through the next generation of telescopes
   - The Large Synoptic Survey Telescope (LSST) will house a 3200-megapixel camera producing 15 TB of images nightly
2. Replace non-statistical approaches to building astronomical catalogs from photometrical data
3. Identify promising galaxies for spectrograph targeting
   - Better understand dark energy and the geometry of the Universe
4. Develop and extensible model and inference procedure, for use by the astronomical community
   - Future applications might include finding supernovae and detecting near-Earth asteroids

Applications: Past – `Celeste.jl` (cont.)

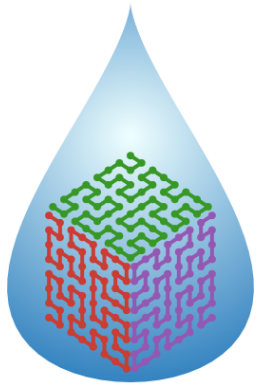
Accomplishments:

1. Reached 1.54 petaFLOPS performance (first First Julia application to exceed 1 petaFLOPS)
   - Julia is probably the first dynamic high-level language to enter the petaFLOPS club (other languages in it: Assembly, Fortran, C/C++)
   - Code ran on 9568 Intel Xeon Phi nodes of Cori (Phase II)
   - 1.3 milion threads on 650 000 KNL cores
2. Processed most of SDSS dataset in 14.6 minutes
   - Loaded and analysed 178 TB
   - Optimised 188 million stars and galaxies
3. First comprehensive catalog of visible objects with state-of-the-art point and uncertainty estimates
4. Demonstration of Variational Inference on 8 billion parameters
   - 2 orders of mangnitude larger than other reported results

Discover more:

- `https://github.com/jeff-regier/Celeste.jl`
- JuliaCon 2017 talk: `https://www.youtube.com/watch?v=uecdcADM3hY`

Applications: Present – PuMaS



PharmaceUtical Modeling And Simulation

- Suite of tools for developing, simulating, fitting, and analyzing pharmaceutical models
- Bring efficient implementations of all aspects of pharmaceutical modeling under one cohesive package
- Deliver personalised treatment schedules for each individual
- Seemless integration with the rest of Julia ecosystem (`Measurements.jl`, `JuliaDB.jl`, `Query.jl`, etc.)
- Collaboration between Center for Translational Medicine of University of Maryland, Baltimore and Julia Computing

Talks at JuliaCon 2018: `https://www.youtube.com/watch?v=KQ4Vtsd9XNw` and JuliaCon 2019: `https://www.youtube.com/watch?v=i8LGmT0mKnE`

Applications: Future – CLIMA

- Collaboration between Caltech, NASA JPL, MIT, Naval Postgraduate School, funded among others by NSF: `https://clima.caltech.edu/`
- First Earth model that automatically learns from diverse data sources
- Modeling platform that is scalable and built for growth
- It will need to run on the world's fastest supercomputers and on the cloud, using both GPU and CPUs
- Scalable for different resolutions, to have local and global climate
- Julia chosen to ensure performance on modern heterogeneous architectures without sacrificing scientific productivity information

Talk at JuliaCon 2019: `https://www.youtube.com/watch?v=gD5U_U9kZk8`

Take-Home Messages

- Great composability: complex packages can work together
- Incremental optimisation: from prototype to final product step by step
  - `https://docs.julialang.org/en/v1/manual/performance-tips/`
  - `https://mitmath.github.io/18337/lecture2/optimizing`
- Julia programs are organised around multiple dispatch
- Most of Julia is written in Julia itself
- My 2 cents: main Julia's strength is genericity, which increases productivity

Take-Home Messages

- Great composability: complex packages can work together
- Incremental optimisation: from prototype to final product step by step
  - `https://docs.julialang.org/en/v1/manual/performance-tips/`
  - `https://mitmath.github.io/18337/lecture2/optimizing`
- Julia programs are organised around multiple dispatch
- Most of Julia is written in Julia itself
- My 2 cents: main Julia's strength is genericity, which increases productivity

Got interested?

- Official website: `https://julialang.org/`
- Manual: `https://docs.julialang.org/en/`
- List of registered packages: `https://juliahub.com/`
- GitHub repository: `https://github.com/JuliaLang/julia`
- Discussion forum: `https://discourse.julialang.org/`
- Slack workspace: `https://slackinvite.julialang.org/`

JuliaCon 2020 Everywhere on Earth!

Wednesday 29th to Friday 31st of July, 2020. Register for free at `https://juliacon.org/2020/`