Maintaining Large Scale Julia Ecosystems

Chris Rackauckas, VP of Modeling and Simulation @ JuliaHub, Research Affiliate @ MIT

Agenda

Building an ecosystem is a large project over many years. In this talk I'll share

- **Opinions**: built with time and experience
- **Practice**: some Julia-specific tips for improving maintainability
- **Tools**: you cannot do it all by yourself

Who am I?

Foundation: Fast Differential Equation Solvers

1. Speed
2. Stability
3. Stochasticity
4. Adjoints and Inference
5. Parallelism

DifferentialEquations.jl is generally:

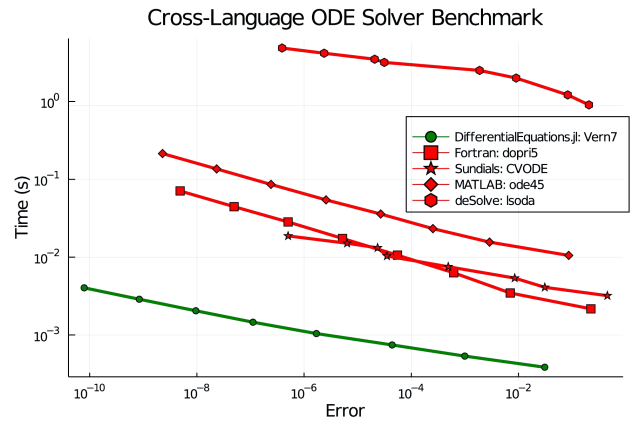
- 50x faster than SciPy
- 50x faster than MATLAB
- 100x faster than R's deSolve

https://github.com/SciML/SciMLBenchmarks.jl

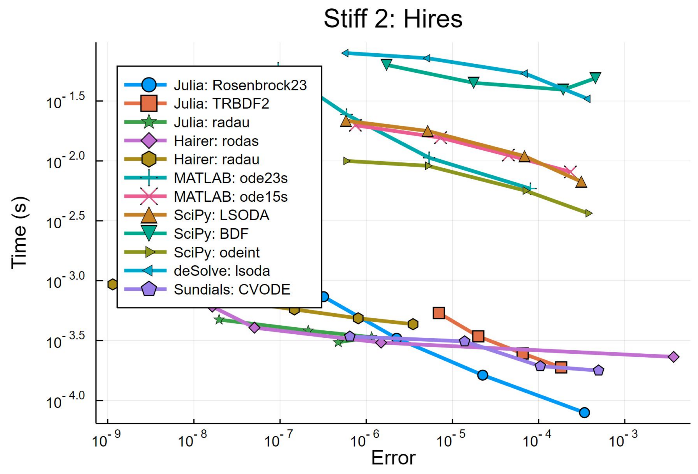
Rackauckas, Christopher, and Qing Nie. "Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia." Journal of Open Research Software 5.1 (2017).

Rackauckas, Christopher, and Qing Nie. "Confederated modular differential equation APIs for accelerated algorithm development and benchmarking." Advances in Engineering Software 132 (2019): 1-6.

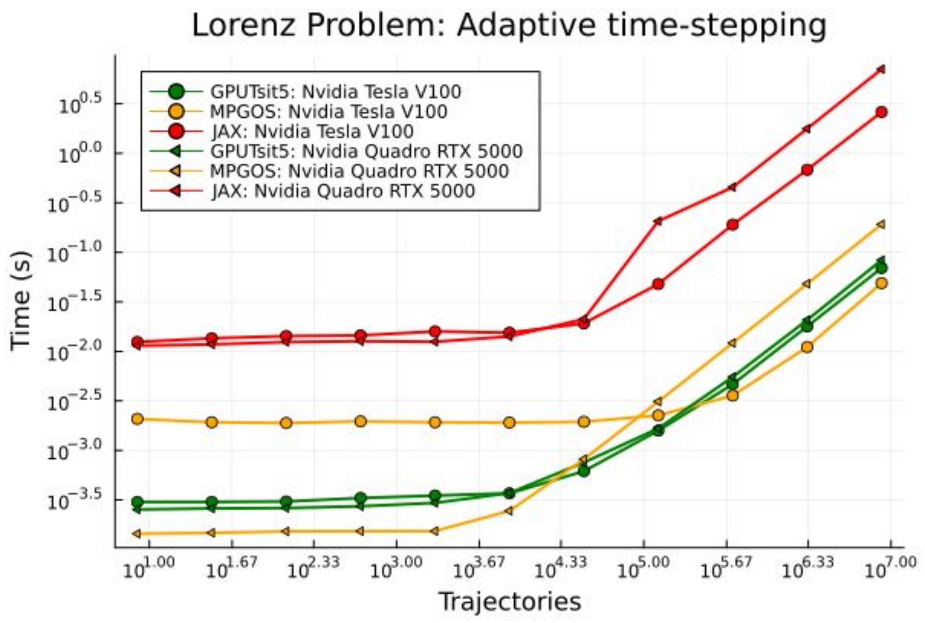
Non-Stiff ODE: Rigid Body System

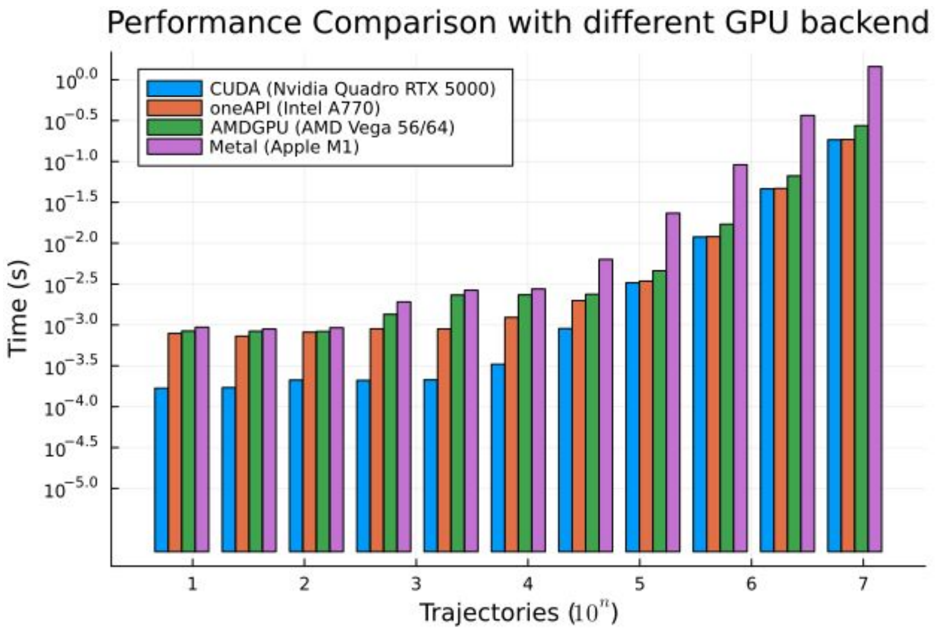


8 Stiff ODEs: HIRES Chemical Reaction Network



New Parallelized GPU ODE Parallelism: 20x-100x Faster than Jax and PyTorch





Matches State of the Art on CUDA, but also works with AMD, Intel, and Apple GPUs

Paper accepted!

Utkarsh, U., Churavy, V., Ma, Y., Besard, T., Gymnich, T., Gerlach, A. R., ... & Rackauckas, C. (2023). Automated Translation and Accelerated Solving of Differential Equations on Multiple GPU Platforms. *arXiv preprint arXiv:2304.06835*.

Understanding Julia's Development Speed: Why are Julia packages growing faster and better tested?

	Julia Scientific Computing	Julia Machine Learning	Normal Python	PyTorch
ODE Solver	DifferentialEquations.jl >100 Contributors throughout its dependency stack	DifferentialEquations.jl >100 Contributors throughout its dependency stack	SciPy.odeint ~5 developers	Torchdiffeq 1 contributor with more than one contribution
SDE Solver	DifferentialEquations.jl >100 Contributors throughout its dependency stack	DifferentialEquations.jl >100 Contributors throughout its dependency stack	N/A	TorchSDE 2 contributors with more than one contribution (last commit July 2021)
DDE Solver	DifferentialEquations.jl >100 Contributors throughout its dependency stack	DifferentialEquations.jl >100 Contributors throughout its dependency stack	N/A	N/A
DAE Solver	DifferentialEquations.jl >100 Contributors throughout its dependency stack	DifferentialEquations.jl >100 Contributors throughout its dependency stack	N/A	N/A

What is Scientific Machine Learning (SciML)?

Scientific Computing ↔ Machine Learning
AI with the robustness of classical modeling methods

Scientific Computing

- Model Building
- Robust Solvers
- Control Systems

+

Machine Learning

- Neural Nets
- Data Models
- Automated Procedures

→

Scientific Machine Learning

- Differentiable Simulators
- Surrogates and ROM
- Inverse Problems & Calibration
- Automatic Equation Discovery

and more ....

JuliaHub

Accurate Model Extrapolation Mixing in Physical Knowledge

Upon denoting $\mathbf{x} = (\phi, \chi, p, e)$, we propose the following family of UDEs to describe the two-body relativistic dynamics:

$$\dot{\phi} = \frac{(1 + e\cos(\chi))^2}{Mp^{3/2}}(1 + \mathcal{F}_1(\cos(\chi), p, e)), \tag{5a}$$

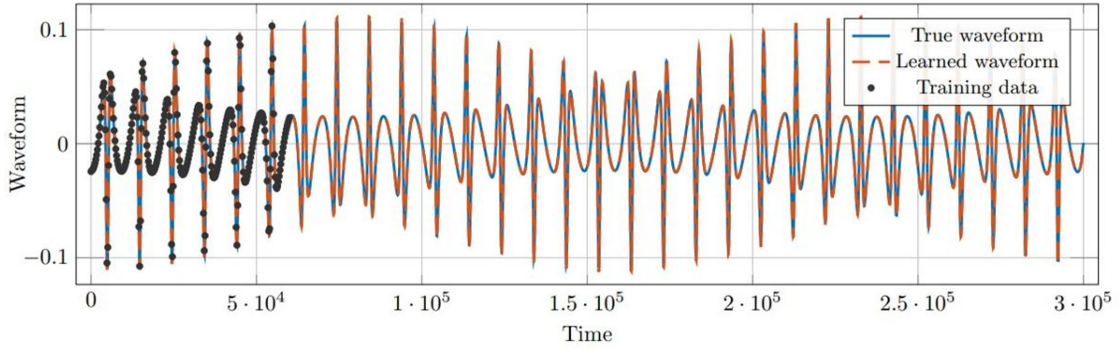
$$\dot{\chi} = \frac{(1 + e\cos(\chi))^2}{Mp^{3/2}}(1 + \mathcal{F}_2(\cos(\chi), p, e)), \tag{5b}$$

$$\dot{p} = \mathcal{F}_3(p, e), \tag{5c}$$

$$\dot{e} = \mathcal{F}_4(p, e), \tag{5d}$$

Automated discovery of geodesic equations from LIGO black hole data: run (a simplified version of) the code yourself!

https://docs.sciml.ai/Overview/stable/showcase/blackhole/



Improving Coverage of Automatic Differentiation Over Solvers

LinearSolve.jl: Unified Linear Solver Interface

$$A(p)x = b$$

NonlinearSolve.jl: Unified Nonlinear Solver Interface

$$f(u,p) = 0$$

DifferentialEquations.jl: Unified Interface for all Differential Equations

$$u' = f(u,p,t)$$

$$du = f(u,p,t)dt + g(u,p,t)dW_t$$

$$\vdots$$

The SciML Common Interface for Julia Equation Solvers

https://scimlbase.sciml.ai/dev/

Optimization.jl: Unified Optimization Interface

$$\text{minimize } f(u,p)$$
$$\text{subject to } g(u,p) \leq 0, h(u,p) = 0$$

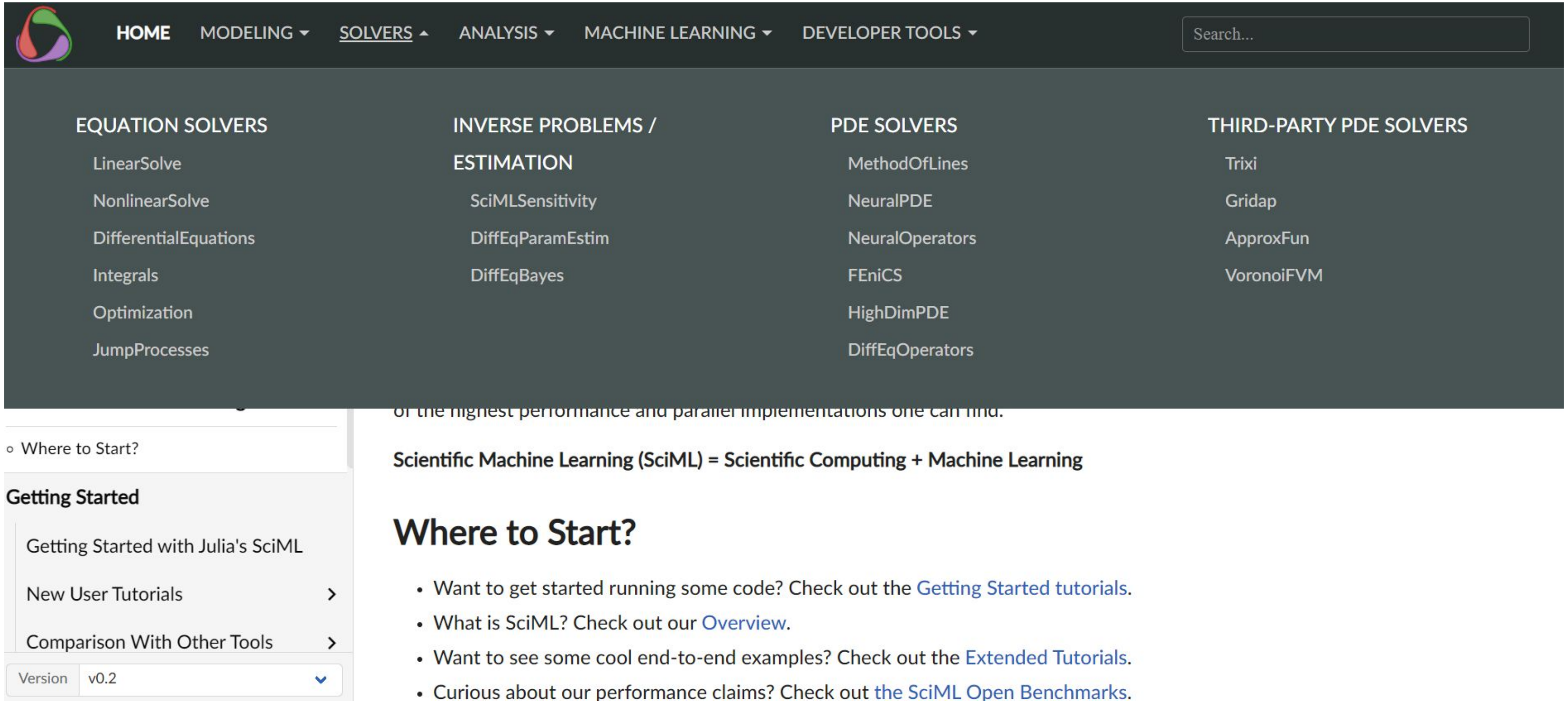
Integrals.jl: Unified Quadrature Interface

$$\int_{lb}^{ub} f(t,p)dt$$

Unified Partial Differential Equation Interface

$$u_t = u_{xx} + f(u)$$
$$u_{tt} = u_{xx} + f(u)$$
$$\vdots$$

SciML Docs: Comprehensive Documentation of Differentiable Simulation



EQUATION SOLVERS
- LinearSolve
- NonlinearSolve
- DifferentialEquations
- Integrals
- Optimization
- JumpProcesses

INVERSE PROBLEMS / ESTIMATION
- SciMLSensitivity
- DiffEqParamEstim
- DiffEqBayes

PDE SOLVERS
- MethodOfLines
- NeuralPDE
- NeuralOperators
- FEniCS
- HighDimPDE
- DiffEqOperators

THIRD-PARTY PDE SOLVERS
- Trixi
- Gridap
- ApproxFun
- VoronoiFVM

- Where to Start?

Getting Started
- Getting Started with Julia's SciML
- New User Tutorials
- Comparison With Other Tools

Version v0.2

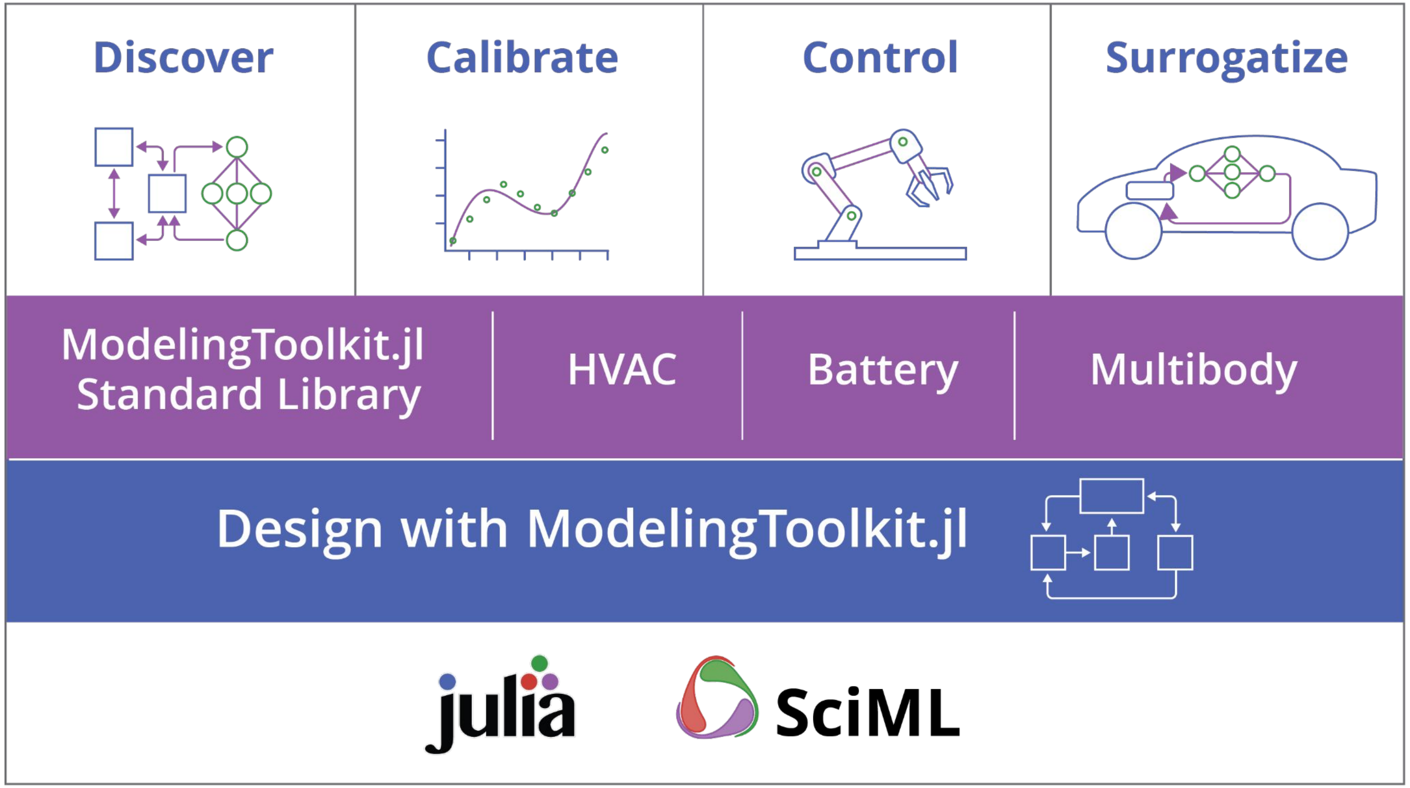
of the highest performance and parallel implementations one can find.

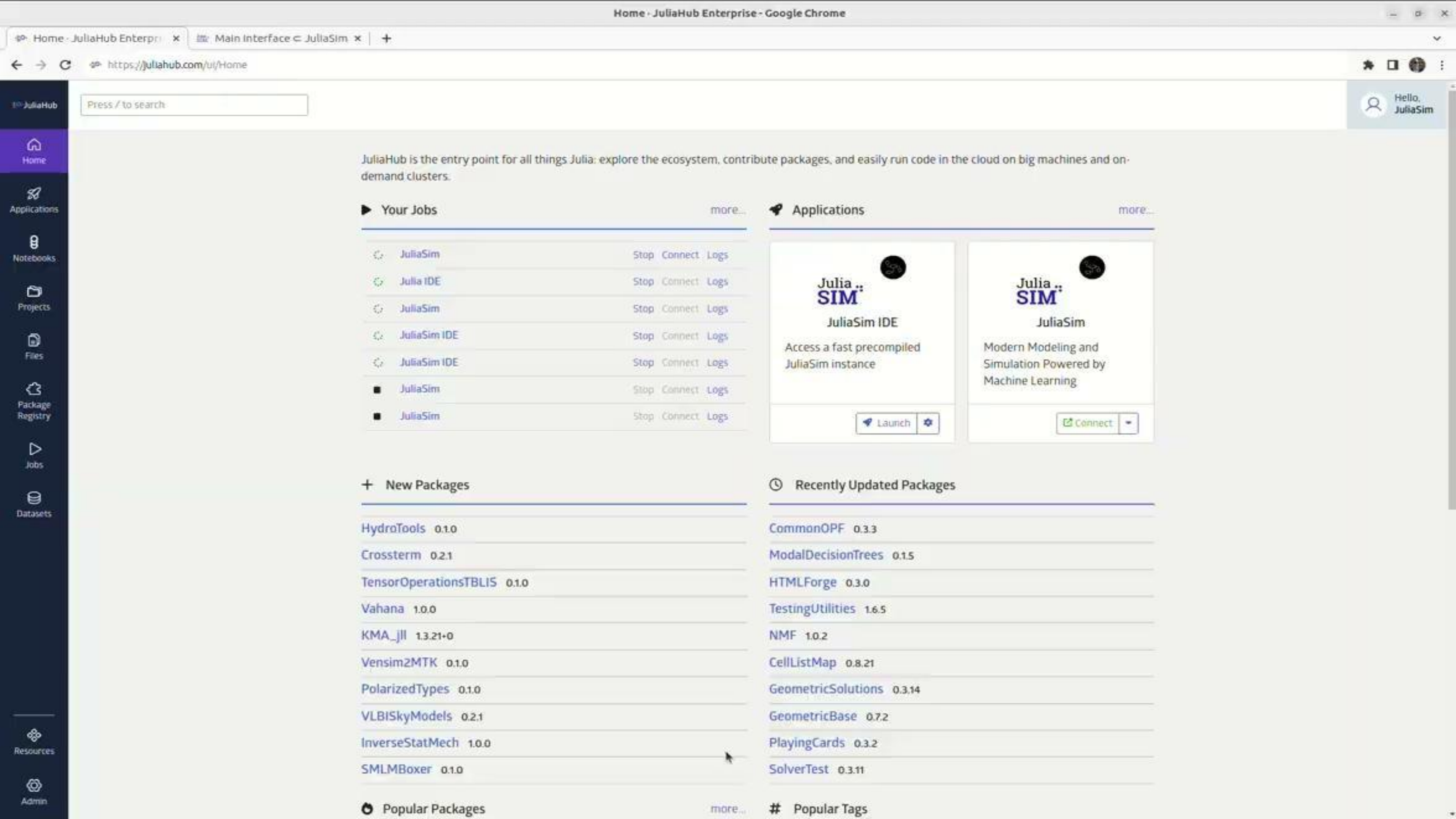
Scientific Machine Learning (SciML) = Scientific Computing + Machine Learning

Where to Start?

- Want to get started running some code? Check out the Getting Started tutorials.
- What is SciML? Check out our Overview.
- Want to see some cool end-to-end examples? Check out the Extended Tutorials.
- Curious about our performance claims? Check out the SciML Open Benchmarks.

JuliaSim Architecture


Discover
Calibrate
Control
Surrogatize
ModelingToolkit.jl Standard Library
HVAC
Battery
Multibody
Design with ModelingToolkit.jl
julia
SciML



Home · JuliaHub Enterprise - Google Chrome

Home · JuliaHub Enterpr | Main Interface ⊂ JuliaSim

https://juliahub.com/ui/Home

Press / to search

Hello, JuliaSim

JuliaHub is the entry point for all things Julia: explore the ecosystem, contribute packages, and easily run code in the cloud on big machines and on-demand clusters.

Your Jobs more...

JuliaSim	Stop	Connect	Logs
Julia IDE	Stop	Connect	Logs
JuliaSim	Stop	Connect	Logs
JuliaSim IDE	Stop	Connect	Logs
JuliaSim IDE	Stop	Connect	Logs
JuliaSim	Stop	Connect	Logs
JuliaSim	Stop	Connect	Logs

Applications more...

JuliaSim IDE
Access a fast precompiled JuliaSim instance
Launch

JuliaSim
Modern Modeling and Simulation Powered by Machine Learning
Connect

New Packages

HydroTools	0.1.0
Crossterm	0.2.1
TensorOperationsTBLIS	0.1.0
Vahana	1.0.0
KMA_jll	1.3.21+0
Vensim2MTK	0.1.0
PolarizedTypes	0.1.0
VLBISkyModels	0.2.1
InverseStatMech	1.0.0
SMLMBoxer	0.1.0

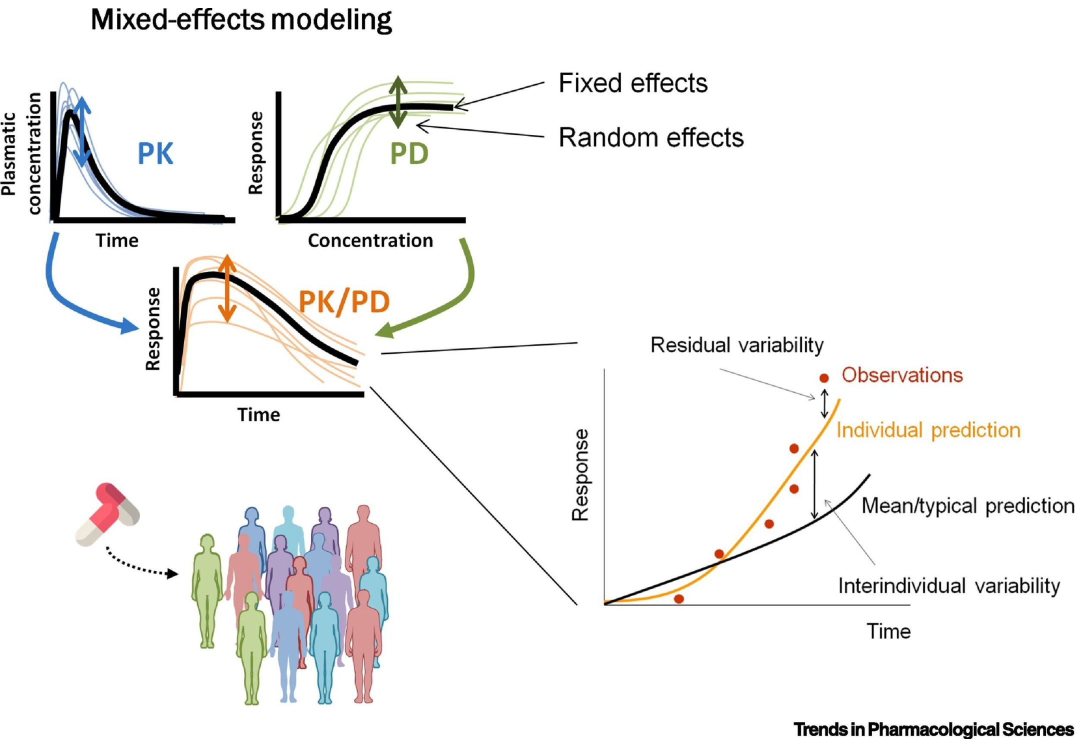
Recently Updated Packages

CommonOPF	0.3.3
ModalDecisionTrees	0.1.5
HTMLForge	0.3.0
TestingUtilities	1.6.5
NMF	1.0.2
CellListMap	0.8.21
GeometricSolutions	0.3.14
GeometricBase	0.7.2
PlayingCards	0.3.2
SolverTest	0.3.11

Popular Packages more...

Popular Tags

DeepNLME: Integrate neural networks into traditional NLME modeling
DeepNLME is SciML-enhanced modeling for clinical trials



pumas

- Automate the discovery of predictive covariates and their relationship to dynamics
- Automatically discover dynamical models and assess the fit
- Incorporate big data sources, such as genomics and images, as predictive covariates

The Impact of Pumas (PharmacUtical Modeling And Simulation)

"

We have been using Pumas software for our pharmacometric needs to support our development decisions and regulatory submissions.

Pumas software has surpassed our expectations on its accuracy and ease of use. We are encouraged by its capability of supporting different types of pharmacometric analyses within one software. **Pumas has emerged as our "go-to" tool for most of our analyses in recent months.** We also work with Pumas-AI on drug development consulting. We are impressed by the quality and breadth of the experience of Pumas-AI scientists in collaborating with us on modeling and simulation projects across our pipeline spanning investigational therapeutics and vaccines at various stages of clinical development

Husain A. PhD (2020)
Director, Head of Clinical Pharmacology and Pharmacometrics,
Moderna Therapeutics, Inc

moderna™
messenger therapeutics

Built on SciML

JuliaHub

How to develop: Always Forwards, Never Backwards

Principle 1: trust your tests

You will never be able to grow maintainer support beyond yourself if you do not trust your test suite to carry you forward.

- If you do not feel comfortable blind merging a small bugfix that passes all tests, that is a sign you do not trust your tests
- Trusting your tests will teach you what your tests are missing

The different kinds of tests

- Unit tests: small tests which test a single piece / behavior of the software
    - Good to have, but actually not the most important!
- Integration tests: tests which tests how the different parts integrate
    - The most important tests
- Interface tests: tests which demonstrate that the interface is as expected
- Regression tests: tests that the behavior of the software does not change
- Downstream tests: tests that your user's code doesn't break
    - Second most important tests

Tests for Alternative Hardware

The Julia Lab provides buildkite scripts for many alternative hardwares:

- NVIDIA GPUs
- AMD GPUs
- Intel GPUs
- Mac M-Series

Ask and you shall receive.

Using Test Groups for Parallelism

Parallelize your tests by grouping then with environment variables

- Allow people to easily run a subset of tests
- Run tests in parallel on CI
- Allow downstream to run a subset of tests
- Make every test an independent runnable script (SafeTestsets.jl)

See OrdinaryDiffEq.jl as a complex example!

Documentation as Tests

Force your documentation to act as interface tests

- Differentiate examples vs tutorials
- Establish doctests
- Cover real-world workflows
- Get more compute if you need to

Principle 2: Make simple things easy, complex things possible*

- Implement basic features that you may even thing are too basic for you
  - Examples: DiffEq’s solve(prob), ANOVA
- 90% of users use the most basic 10% of the code
- Let people grow with you
  - That undergrad may one day be your most helpful maintainer!
- Listen to your audience when they say something is hard
- Write a lot of tutorials, and turn any repeated structures into tutorials
- Make your error messages read your beginner user’s minds

* I stole this phrase from Mike Tiller

Principle 3: Build modularly

When should you be using more than one package?

- A single package should cover a single idea
  - If the users of different parts of different
- If changing one part of the code almost always requires changing the other, then those two should live in the same repository!
- Reuse other packages when you can
  - And contribute when you need to!
- Move all reusable data structures out to data structures packages

Principle 4: Your Contributors are your BFFs

Always find the time to respond to and validate your contributors

- Attempt to respond to a PR within the same day
  - If the feature isn't quite what you expected, explain the greater vision
  - If the code could be improved, use Github suggestions to make the improvements in a teachable way
  - Fix the code yourself the first few times and aim for faster merging
  - Help with rebasing and merge conflicts when you can!
  - Don't aim for perfection: merge and open issues with next steps
- **If you trust your tests, doing quick merges will be easier!**
- **You never know who your next maintainer will be**

Principle 5: Align Development with Research

If your ecosystem starts to catch on, it will be your life.

- Make sure that the ecosystem itself has research goals
- Don't put new contributors into spaces that will not be co-authors (and are boring!)
- Make your paper examples into documentation examples!
- **Don't do projects which are simply "rewrite it in Julia", chain it to some larger project, idea, or goal or it will die.**

Principle 6: Don’t Fall Behind in Compats

Small updates are exponentially simpler than large updates

- Try to stay up to date with dependencies
- Learn who writes your dependencies
- Upstream issues
- Work in the community

Testing and Manifests: Should you use one?

If you are developing the ecosystem, you want to keep a global view of how users will interact with your packages.

- Users should always use environments
- You as a developer shouldn’t: you should see what the naive user gets!

Principle 7: Listen To Your Users As If They Are Smart

Give users the benefit of the doubt: they are (hopefully) smart people

- If many people are asking the same question, it’s missing a tutorial
- If many people are writing the same weird code, then there’s bad documentation somewhere
    - Ask how they found out
- Solve people’s problems and then ask them to contribute to the documentation
- Know when to pull the plug on a help vampire

Principle 8: Interfaces Are The Hardest Part

A lot of people think that the interface is “the easy part”... NO!

- Writing a solver takes some expertise, but has a “local” view
- Creating an interface requires a global view of how everyone will see your packages, learn about your packages, and ultimately use your packages.
- Moral of the story: spend most of your time thinking about interfaces, and don’t leave this portion to the contributors.

Establishing Interfaces

Establishing an interface takes the following steps:

- Document how the interface is supposed to work
  - **Clearly define what is public vs private**
- Write the first 3 instantiations of the interface
- Write a higher level call (solve) which calls to a lower level call (__solve) and have users extend the lower level interface. This allows a high level that forces uniformization
  - Make high level error messages
- Religiously refine the interface and listen to your contributors

Principle 9: Democracy is a failure, own by force

Okay, maybe that’s too far, but…

- Repos die if there is no code ownership
  - Example: Matlab.jl’s dilemma
- Taking more ownership is always better than taking less ownership
- Tracking ownership via ownership files can be helpful if you do not have a BDFL
  - But even if you have this, you must have a succession system if someone goes missing!
- If dependencies are having issues keeping up to date, ask if they want help maintaining and bring it into an org

JuliaHub

Thanks!