Doing Scientific Machine Learning with Julia’s SciML Ecosystem

CHRIS RACKAUCKAS

APPLIED MATHEMATICS INSTRUCTOR, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, MATHEMATICS

SENIOR RESEARCH ANALYST, UNIVERSITY OF MARYLAND, BALTIMORE, SCHOOL OF PHARMACY

DIRECTOR OF SCIENTIFIC RESEARCH, PUMAS-AI

What to expect from this workshop

We will go through "what is scientific machine learning" all the way through some examples of how to do scientific machine learning in Julia:

- Parameter inference of differential equation model
- Add events to equations, add stochasticity, add delays
- Bayesian inference of scientific models

This tutorial is meant to be raw:

- We're going to be live coding
  - "We", as in you and me!
- We are going to be learning the methods as we learn to write the code
- We are going to learn how to navigate the ecosystem, the documentation, the IDE, the error messages

How to Help!

- Star the repositories
  - DifferentialEquations.jl, DiffEqFlux.jl, DataDrivenDiffEq.jl, NeuralPDE.jl, ModelingToolkit.jl, Catalyst.jl
  - Flux.jl, Zygote.jl, SparseDiffTools.jl, etc.
- Report bugs
- Create tutorials
- Write blog posts
- Share data and challenge problems
- Help in chats and community channels
- Add your own packages to the common interface
- Contribute to the SciML packages

Studies in Pandemic Preparedness
Simon Frost

We thank all of those who have previously helped SciML development

Workshop Outline

- Overview of scientific machine learning and SciML
  - What is scientific machine learning?
  - What makes the SciML ecosystem unique?
- Introduction to challenge and learning problems
  - Workshop exercises (with answers!)
  - HelicopterSciML Challenge Problem
  - **Magnetic Navigation Challenge Problem**
- Modeling with differential equations
  - Solving differential equations with DifferentialEquations.jl
  - Adding stochasticity, delays, events
- Automated model discovery via universal differential equations
  - Parameter inference on differential equations
    - Local and global optimization
    - Bayesian optimization
  - Mixing DiffEqFlux.jl and DataDrivenDiffEq.jl!
- Solving differential equations with neural networks (physics-informed neural networks)

Scientific machine learning is the mixture of scientific models with data-driven machine learning components for data-efficient model-based decision making

Universal Approximation Theorem

NEURAL NETWORKS CAN GET ϵ CLOSE TO ANY $R^n \rightarrow R^m$ FUNCTION

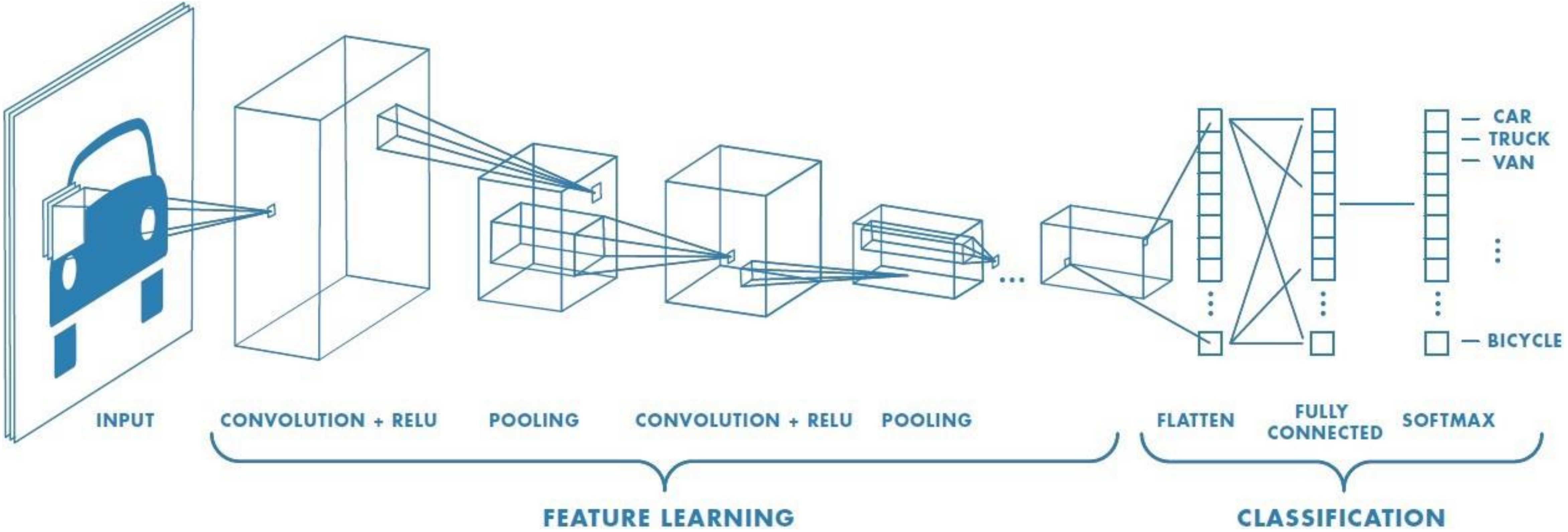
Neural networks are just function expansions, fancy Taylor Series like things which are good for computing and bad for analysis.

Neural networks work well in high dimensions

The major advances in machine learning were due to encoding more structure into the model

MORE STRUCTURE = FASTER AND BETTER FITS FROM LESS DATA

Convolutional Neural Networks Are Structure Assumptions



Universal Differential Equations: Differential Equations defined in part by universal approximators

Use all known scientific features, use all numerical methods, have neural networks cover the last mile

Demonstration of UDEs on a toy model

$$
\begin{aligned}
S' &= -\frac{\beta_0 SF}{N} - \frac{\beta(t)SI}{N} - \mu S, \\
E' &= \frac{\beta_0 SF}{N} + \frac{\beta(t)SI}{N} - \left(\sigma + \mu\right)E, \\
I' &= \sigma E - (\gamma + \mu)I, \\
R' &= \gamma I - \mu R, \\
N' &= -\mu N, \\
D' &= d\,\gamma I - \lambda D, \quad \text{and} \\
C' &= \sigma E,
\end{aligned}
$$

$$\kappa = 1117.3$$

$$\beta(t) = \beta_0(1-\alpha)\left(1 - \frac{D}{N}\right)^{\kappa}$$

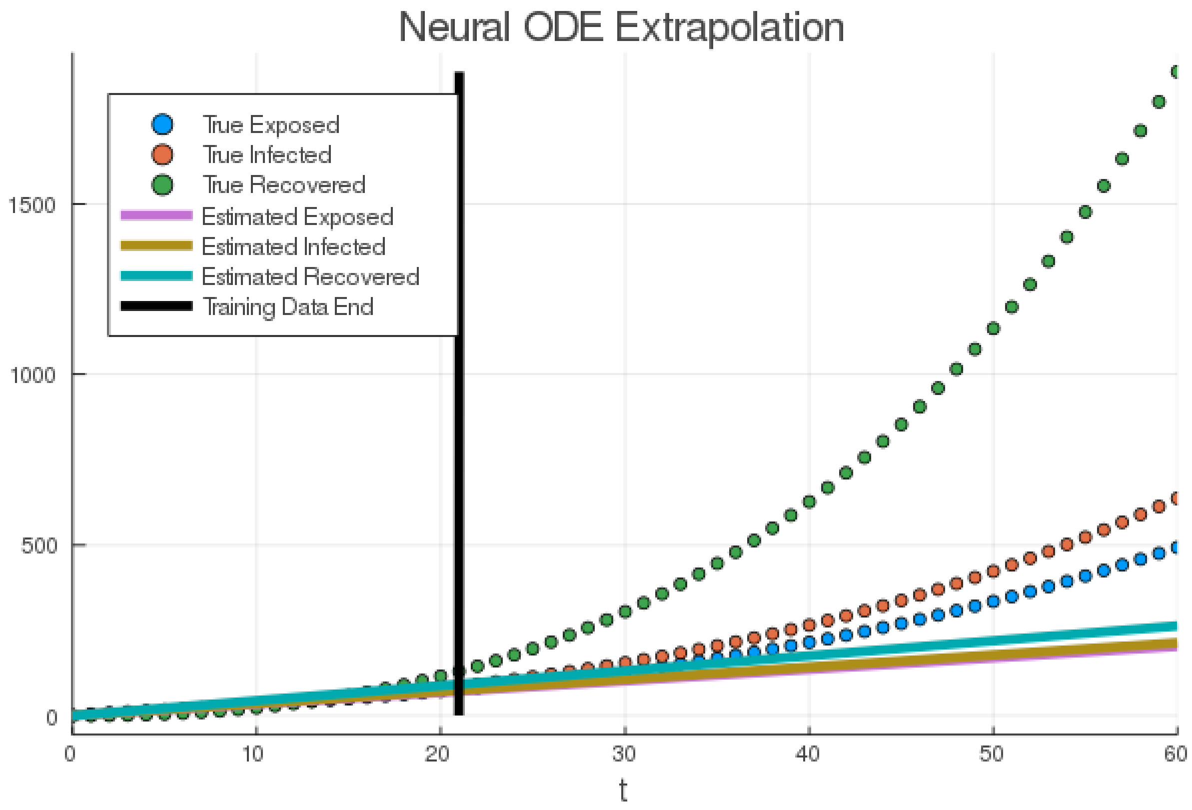
A conceptual model for the coronavirus disease 2019 (COVID-19) outbreak in Wuhan, China with individual reaction and governmental action
Lin, Qianying et al.
International Journal of Infectious Diseases, Volume 93, 211 - 216

Neural ODE: Learn the whole model

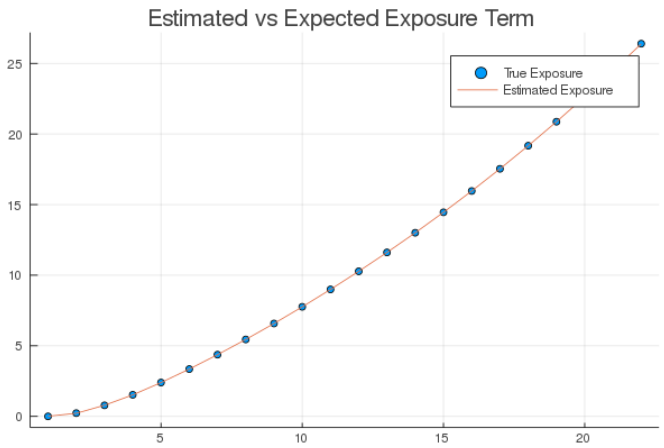
u'=NN(u)

Can fit, but not enough information to accurately extrapolate

Does not have the correct asymptotic behavior



Universal ODE



$$S' = -\frac{\beta_0 S F}{N} - \ldots - \mu S,$$

Replace Unknown Portion

$$E' = \frac{\beta_0 S F}{N} + \ldots - \left(\sigma + \mu\right) E,$$

Replace Unknown Portion

$$I' = \sigma E - (\gamma + \mu) I,$$

$$R' = \gamma I - \mu R,$$

$$N' = -\mu N,$$

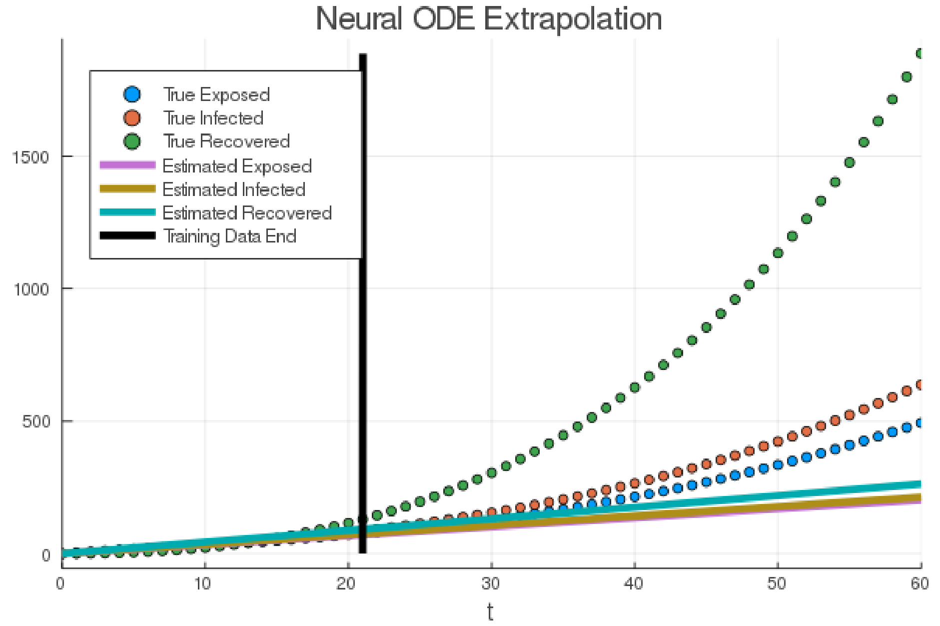
$$D' = d\,\gamma I - \lambda D, \quad \text{and}$$

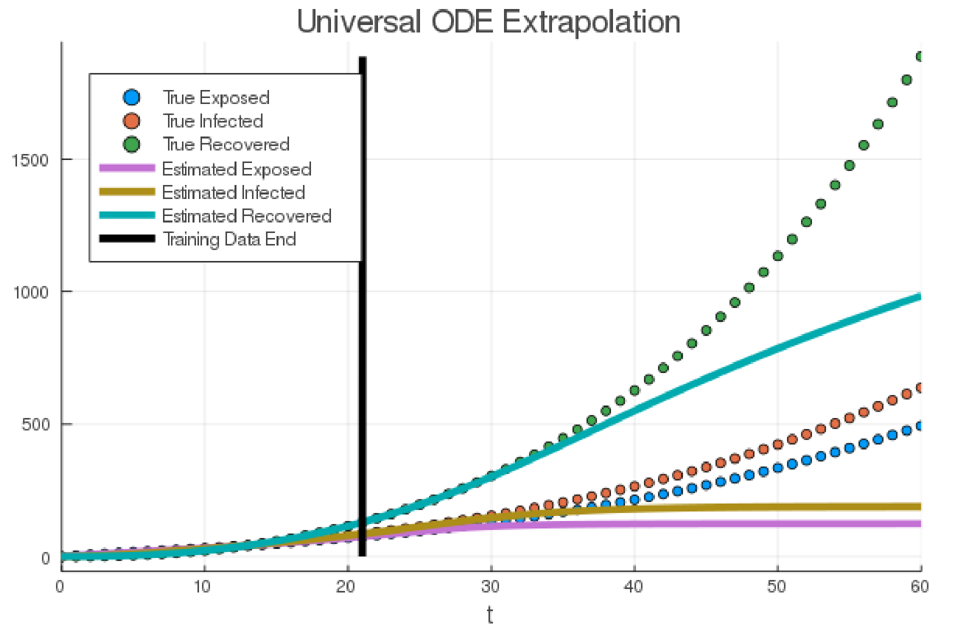
$$C' = \sigma E,$$

Exposure: **Unknown**

Infection rates: known
From disease quantities

Percentage of cases known to be severe, can be estimated





SInDy – Sparse Identification of Dynamical Systems

sparse vectors of coefficients $\Xi = [\xi_1\ \xi_2\ \cdots\ \xi_n]$ that determine which nonlinearities are active:

$$\dot{\mathbf{X}} = \Theta(\mathbf{X})\Xi. \quad [3]$$

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^T(t_1) \\ \mathbf{x}^T(t_2) \\ \vdots \\ \mathbf{x}^T(t_m) \end{bmatrix} = \overset{\text{state} \longrightarrow}{\begin{bmatrix} x_1(t_1) & x_2(t_1) & \cdots & x_n(t_1) \\ x_1(t_2) & x_2(t_2) & \cdots & x_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(t_m) & x_2(t_m) & \cdots & x_n(t_m) \end{bmatrix}} \downarrow \text{time}$$

$$\dot{\mathbf{X}} = \begin{bmatrix} \dot{\mathbf{x}}^T(t_1) \\ \dot{\mathbf{x}}^T(t_2) \\ \vdots \\ \dot{\mathbf{x}}^T(t_m) \end{bmatrix} = \begin{bmatrix} \dot{x}_1(t_1) & \dot{x}_2(t_1) & \cdots & \dot{x}_n(t_1) \\ \dot{x}_1(t_2) & \dot{x}_2(t_2) & \cdots & \dot{x}_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \dot{x}_1(t_m) & \dot{x}_2(t_m) & \cdots & \dot{x}_n(t_m) \end{bmatrix}.$$

Next, we construct a library $\Theta(\mathbf{X})$ consisting of candidate nonlinear functions of the columns of $\mathbf{X}$. For example, $\Theta(\mathbf{X})$ may consist of constant, polynomial, and trigonometric terms:

$$\Theta(\mathbf{X}) = \begin{bmatrix} | & | & | & | & & | & | & \\ 1 & \mathbf{X} & \mathbf{X}^{P_2} & \mathbf{X}^{P_3} & \cdots & \sin(\mathbf{X}) & \cos(\mathbf{X}) & \cdots \\ | & | & | & | & & | & | & \end{bmatrix}. \quad [2]$$

Not Enough Data! Unable to achieve a sparse basis

Brunton, Steven L., Joshua L. Proctor, and J. Nathan Kutz. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems." *Proceedings of the national academy of sciences* 113.15 (2016): 3932-3937.

- Operation[cos(u_1) * -0.0013108600297508188 + cos(u_2) * 0.0010487334669309 09 + sin(u_3) * 0.002524237642240494 + 4.582000697122147 + u_3 * 48.22745315102507 + u_3 ^ 2 * -0.5293305992835255 + u_2 * 39.085961651678964 + u_2 * u_3 * -0.6742175940650399 + u_2 * u_3 ^ 2 * 0.0018086945606415868 + u_2 ^ 2 * -0.7760315827702667 + u_2 ^ 2 * u_3 * -0.008270077072923 97 + u_2 ^ 2 * u_3 ^ 2 * -4.842020305460252 5e-5 + u_1 * 0.6927075862062384 + u_1 * u_3 * 2.547789638418767 5 + u_1 * u_3 ^ 2 * -0.00763369780134226 5 + u_1 * u_2 * -0.8050223920175605 + u_1 * u_2 * u_3 * -0.005893734488035572 + u_1 * u_2 * u_3 ^ 2 * -4.205818407350913e-5 + u_1 * u_2 ^ 2 * 0.0515477602256261 1 + u_1 * u_2 ^ 2 * u_3 * 0.00011401535262358879 + u_1 * u_2 ^ 2 * u_3 ^ 2 * -1.8409670007515867e-7 + u_1 ^ 2 * -1.480917344589218 + u_1 ^ 2 * u_3 * 0.022834435321810845 + u_1 ^ 2 * u_3 ^ 2 * -7.105050116056 66e-5 + u_1 ^ 2 * u_2 * -0.0811262292209696 + u_1 ^ 2 * u_2 * u_3 * 1.2503710381374686e-5 + u_1 ^ 2 * u_2 * u_3 ^ 2 * -1.5835869421530206e-7 + u_1 ^ 2 * u_2 ^ 2 * 0.0003756078420420898 + u_1 ^ 2 * u_2 ^ 2 * u_3 * 2.0403671083190194e-6 + u_1 ^ 2 * u_2 ^ 2 * u_3 ^ 2 * -4.0790059067580516e-10, cos(u_1) * 0.0018236630124880049 + sin(u_3) * -0.002857556410244201 + 0.738713743952307 + u_3 * -45.316633125282735 + u_3 ^ 2 * 0.497655234149502 7 + u_2 * -36.669905096040644 + u_2 * u_3 * 0.63405194300575 + u_2 * u_3 ^ 2 * -0.0016991894990091 62 + u_2 ^ 2 * 0.7292234161358288 + u_2 ^ 2 * u_3 * 0.0077828472509328 61 + u_2 ^ 2 * u_3 ^ 2 * 4.553783234311538 5e-5 + u_1 * -0.662837140886116 + u_1 * u_3 * -2.3955577736237044 + u_1 * u_3 ^ 2 * 0.007174813124917316 + u_1 * u_2 *
- 0.7564652530371222 + u_1 * u_2 * u_3 * 0.005539740817006857 + u_1 * u_2 * u_3 ^ 2 * 3.952859749575076e-5 + u_1 * u_2 ^ 2 * -0.04846972496409705 + u_1 * u_2 ^ 2 * u_3 * -0.00010714683124587004 + u_1 * u_2 ^ 2 * u_3 ^ 2 * 1.7315253185547634e-7 + u_1 ^ 2 * 1.3922758705496125 +
- u_1 ^ 2 * u_3 * -0.021478161074782457 + u_1 ^ 2 * u_3 ^ 2 * 6.675620535553527e-5 + u_1 ^ 2 * u_2 * 0.07628907557295377 + u_1 ^ 2 * u_2 * u_3 * -1.174623626431566e-5 + u_1 ^ 2 * u_2 * u_3 ^ 2 * 1.4858536352836396e-7 + u_1 ^ 2 * u_2 ^ 2 * -0.0003531614272747699 + u_1 ^ 2 * u_2
- ^ 2 * u_3 * -1.9178976768869506e-6 + u_1 ^ 2 * u_2 ^ 2 * u_3 ^ 2 * 3.840565924526202 7e-10, -0.04932474700217403 + u_2 * 0.17406814677977456 + u_1 ^ 2 * u_2 * -1.4594144102122378e-6]

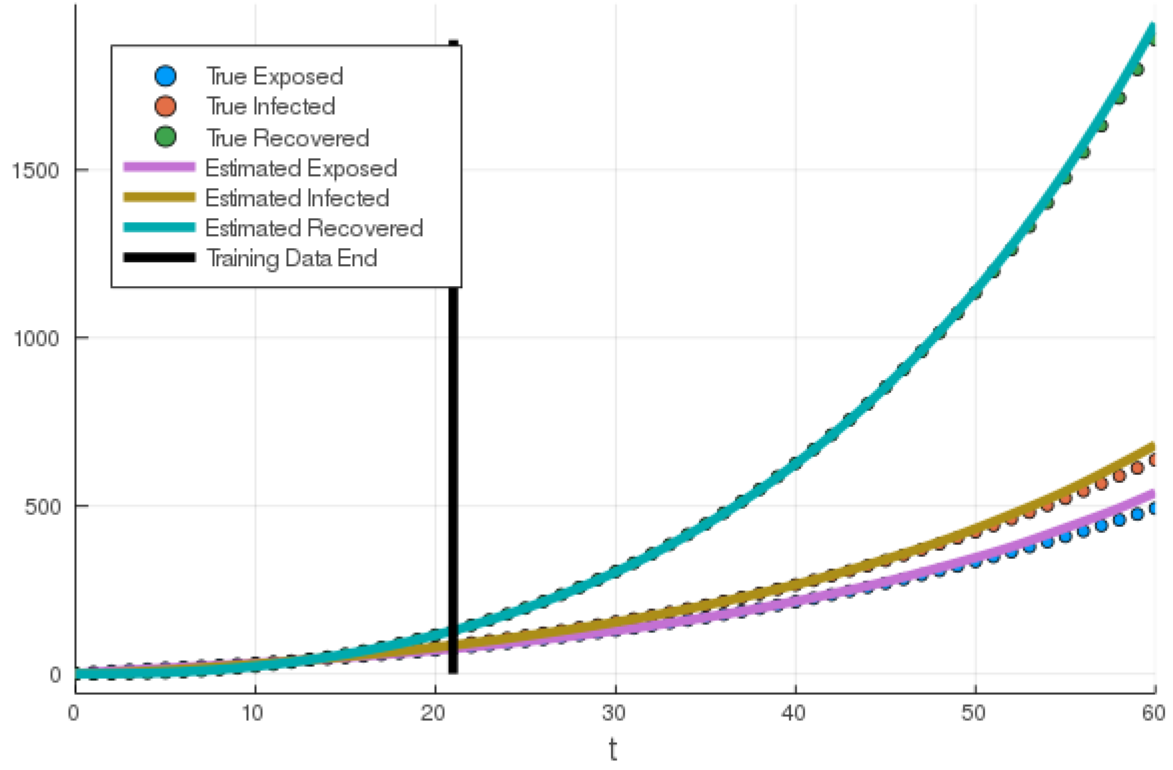
Universal ODE -> Internal Sparse Regression

Sparse Identification on only the missing term
Operation[u_2 * 0.10234428543435758 + u_1 * u_2 * 0.11371750552005416 +
u_1 ^ 2 * u_2 * 0.12635459799855597] of u=(S/N,I,D/N)

$$S' = -\frac{\beta_0 S F}{N} - \text{[Replace Unknown Portion]} - \mu S,$$
$$E' = \frac{\beta_0 S F}{N} + \text{[Replace Unknown Portion]} - (\sigma + \mu) E,$$
$$I' = \sigma E - (\gamma + \mu) I,$$
$$R' = \gamma I - \mu R,$$
$$N' = -\mu N,$$
$$D' = d\,\gamma I - \lambda D, \quad \text{and}$$
$$C' = \sigma E,$$

Replace Unknown Portion

Replace Unknown Portion



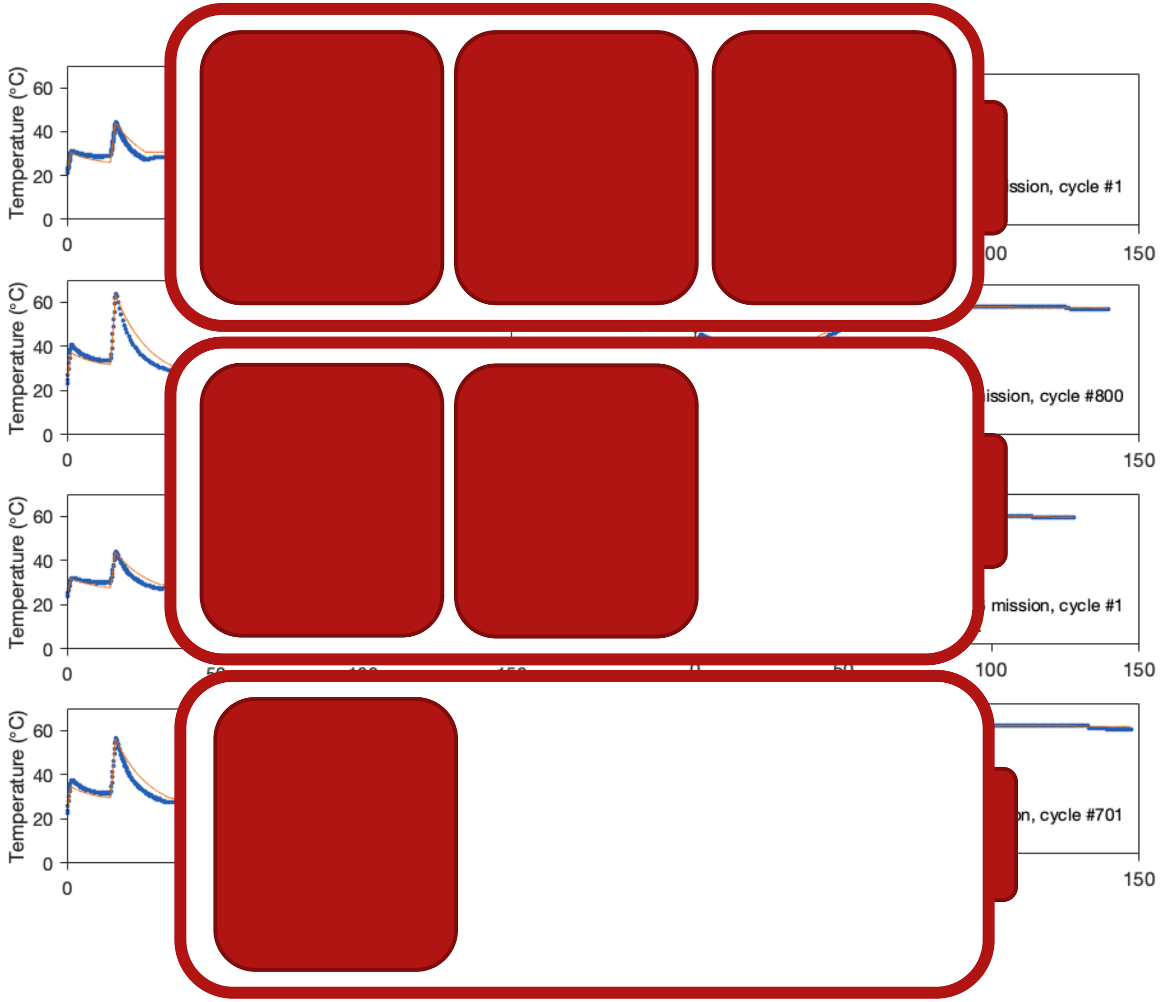
ML-Augmented Scientific Modeling

1. IDENTIFY KNOWN PARTS OF A MODEL, BUILD A UODE
2. TRAIN A NEURAL NETWORK (OR OTHER APPROXIMATOR) TO CAPTURE THE MISSING MECHANISMS
3. SPARSE IDENTIFY THE MISSING TERMS TO MECHANISTIC TERMS
4. VERIFY THE MECHANISMS ARE SCIENTIFICALLY PLAUSIBLE
5. EXTRAPOLATE, DO ASYMPTOTIC ANALYSIS, PREDICT BIFURCATIONS
6. GET MORE DATA TO VERIFY THE NEW TERMS

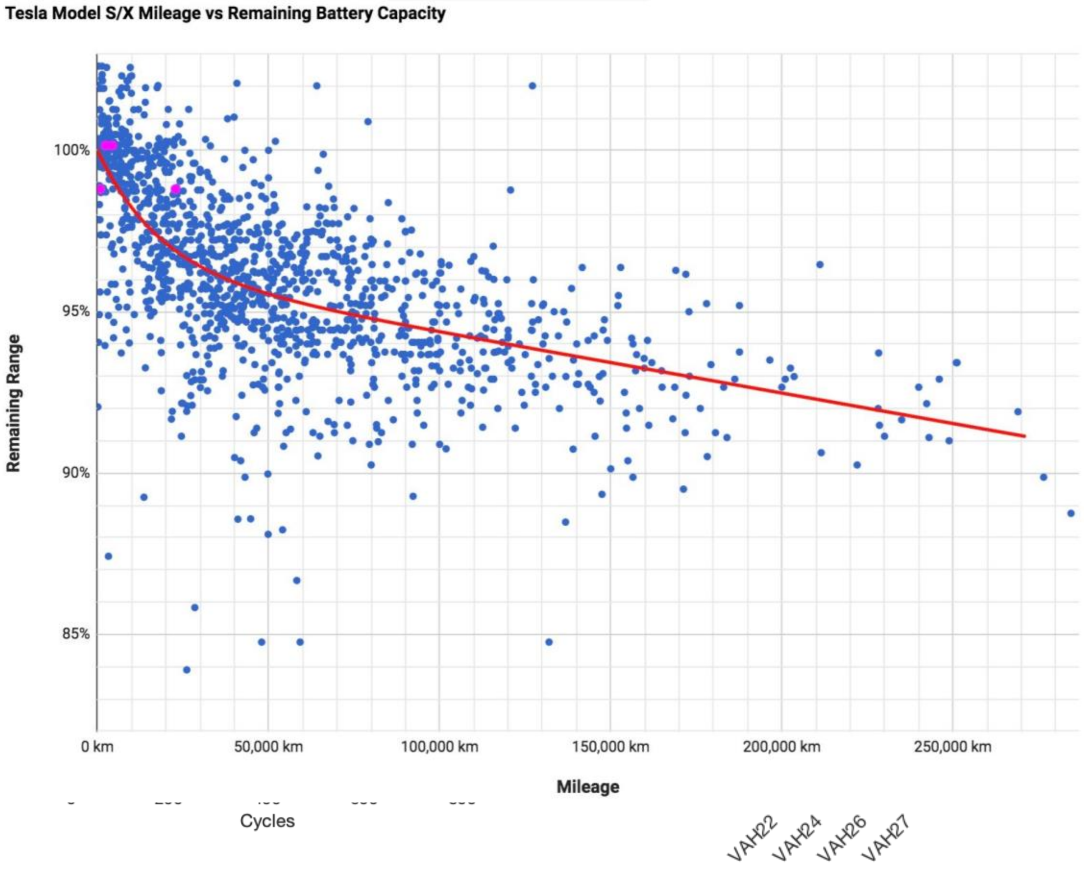
UTILIZE ALL ADVANCED NUMERICAL METHODS WITH ML!

U-ODE's for eVTOL Battery Modeling: 19% Increase in Degradation Modeling Accuracy

Coupled Electrochemical-Thermal Performance Model



U-ODE Degradation Model



A. Bills, S. Sripad, W. L. Fredericks, M. Guttenberg, D. Charles E. Frank, V. Viswanathan, Universal Battery Performance and Degradation Model for Electric Aircraft, DOI: 10.26434/chemrxiv.12616169.v1

https://electrek.co/2018/04/14/tesla-battery-degradation-data/

Data-Driven Quantification of Quarantine Strength

$$\frac{\mathrm{d}S(t)}{\mathrm{d}t} = -\frac{\beta\, S(t)\, I(t)}{N}$$

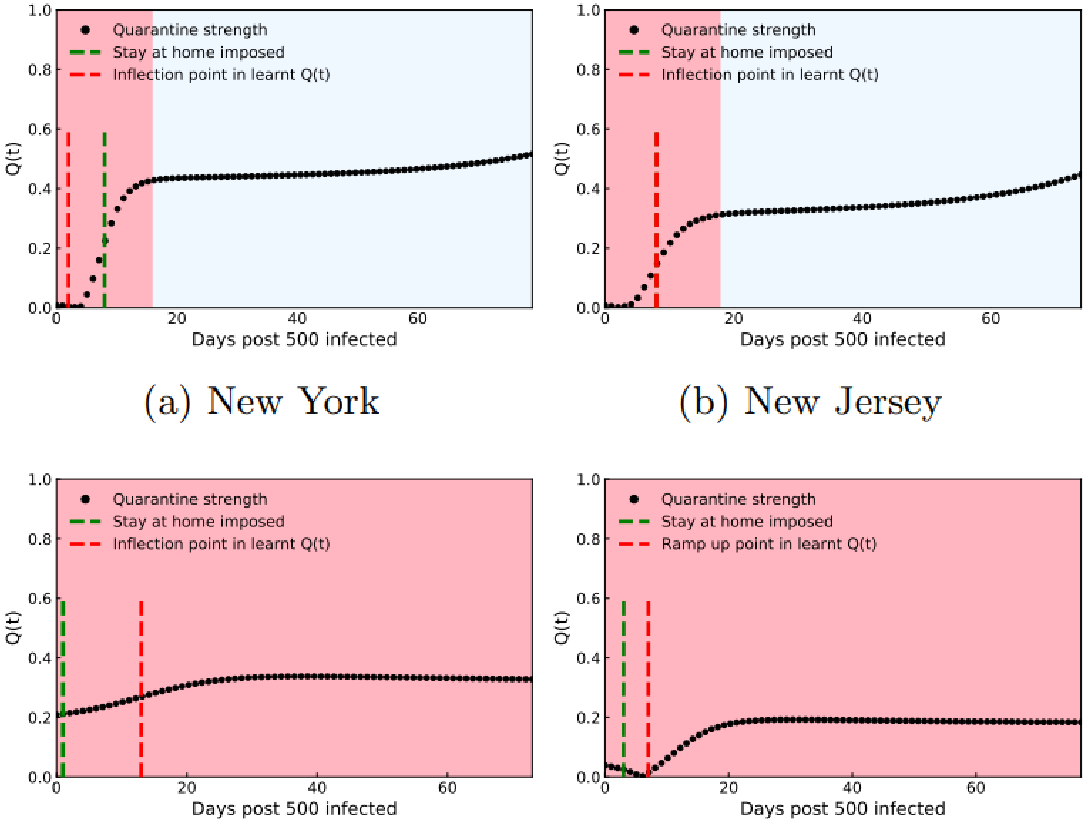
$$\frac{\mathrm{d}I(t)}{\mathrm{d}t} = \frac{\beta\, S(t)\, I(t)}{N} - \big(\gamma + Q(t)\big)\, I(t) = \frac{\beta\, S(t)\, I(t)}{N} - \big(\gamma + \mathrm{NN}(W,U)\big)\, I(t)$$

$$\frac{\mathrm{d}R(t)}{\mathrm{d}t} = \gamma I(t) + \delta T(t)$$

$$\frac{\mathrm{d}T(t)}{\mathrm{d}t} = Q(t)\, I(t) = \mathrm{NN}(W,U)\, I(t) - \delta T(t).$$

A machine learning aided global diagnostic and comparative tool to assess effect of quarantine control in Covid-19 spread

Raj Dandekar,[1] Chris Rackauckas,[2] and George Barbastathis[3,4]†



(a) New York (b) New Jersey

(c) Illinois (d) California

Diagnostics En Masse Reveal Interesting Trends

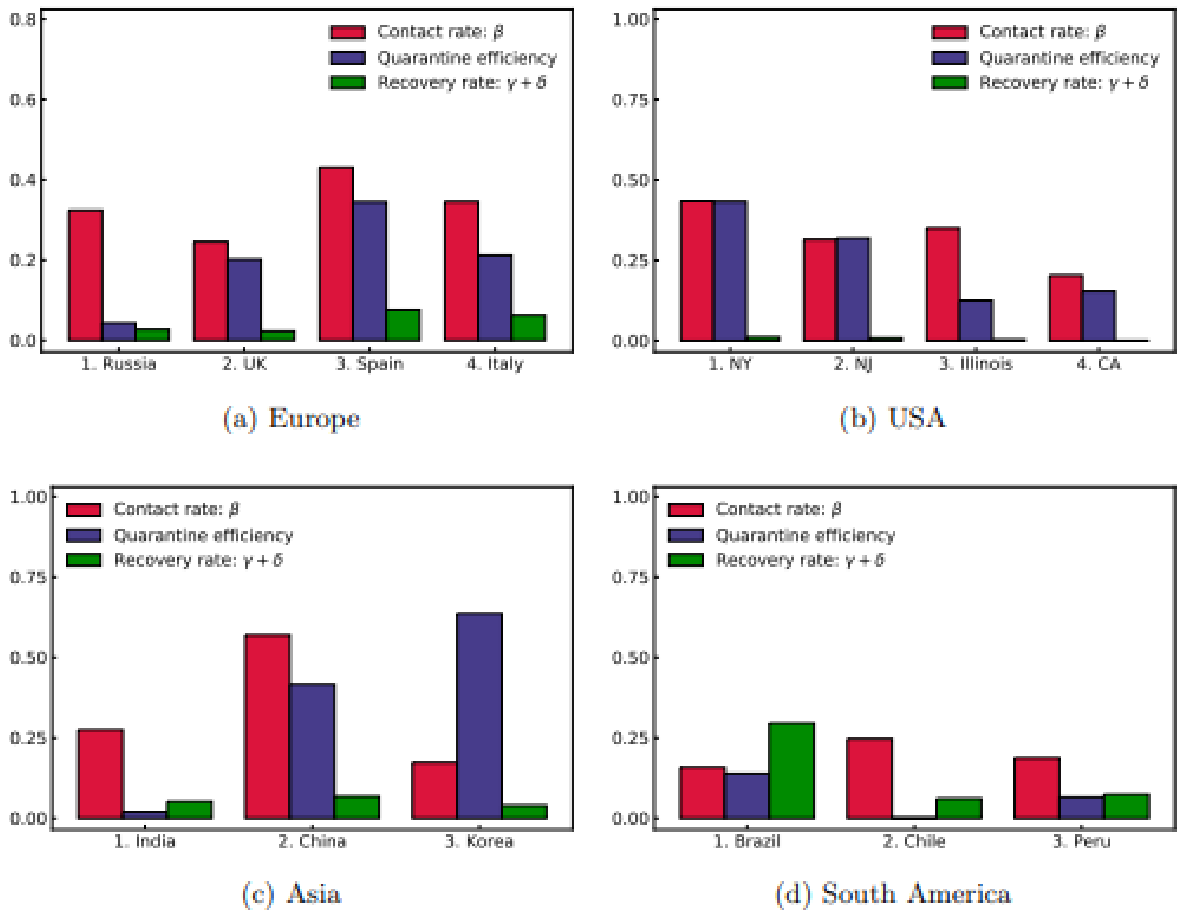


Figure 15: Global comparison of infection, recovery rates and quarantine efficiency.

But ODEs are simple, lets move to more difficult equations

Warning: these next few slides may be information overload if you're not familiar with scientific computing. That's okay! Take in what you can.

Universal Differential-Algebraic Equations: Encoding Physical Constraints

Utilize known chemical kinetics

$$y_1' = -0.04y_1 + \mathrm{NN}_1(y_1, y_2, y_3)$$

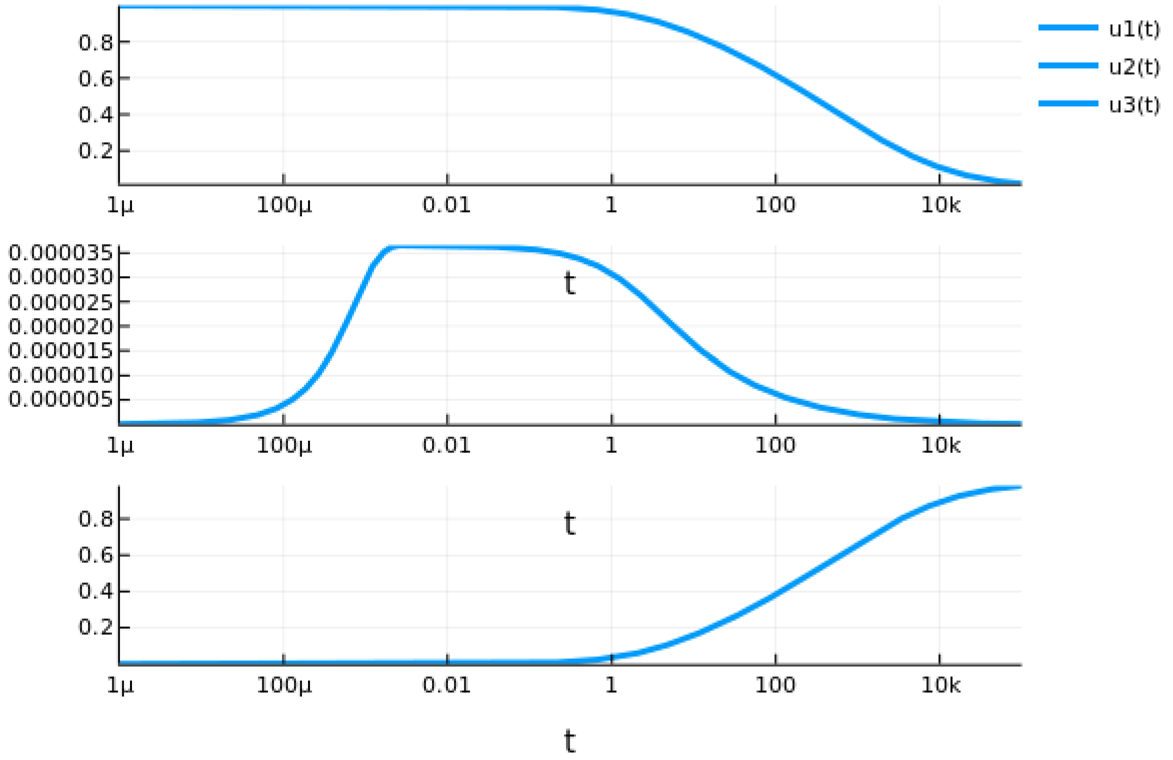
$$y_2' = 0.04y_1 + \mathrm{NN}_2(y_1, y_2, y_3)$$

$$1 = y_1 + y_2 + y_3$$

With known conservation laws

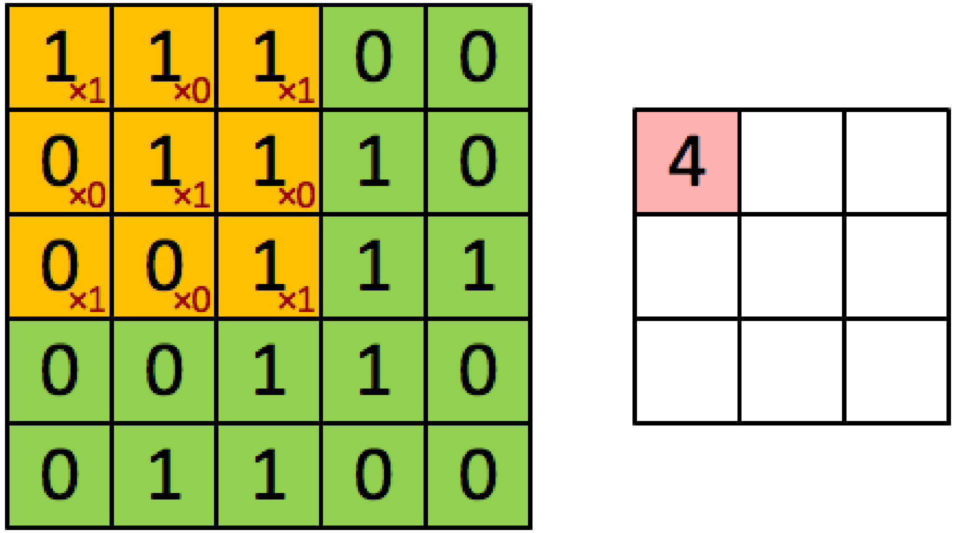
$$Mu' = f(u) + NN(u)$$

Convert to a mass-matrix DAE (singular mass matrix) and fit



Learn highly stiff equations: **Hessian condition number 10^{13}**

Discretized PDE Operators are Convolutions



Image

Convolved Feature

$$\frac{u(x+\Delta x,y)-2u(x,y)+u(x-\Delta x,y)}{\Delta x^2}+\frac{u(x,y+\Delta y)-2u(x,y)+u(x-x,y-\Delta y)}{\Delta y^2}$$

Is equivalent to the stencil

0	1	0
1	-4	1
0	1	0

$$\frac{u(x+\Delta x)-2u(x)+u(x-\Delta x)}{\Delta x^2}=u''(x)+\mathcal{O}\left(\Delta x^2\right)$$

$$\Delta u = u_{xx}+u_{yy}$$

Automatically Learning PDEs from Data: Universal PDEs for Fisher-KPP

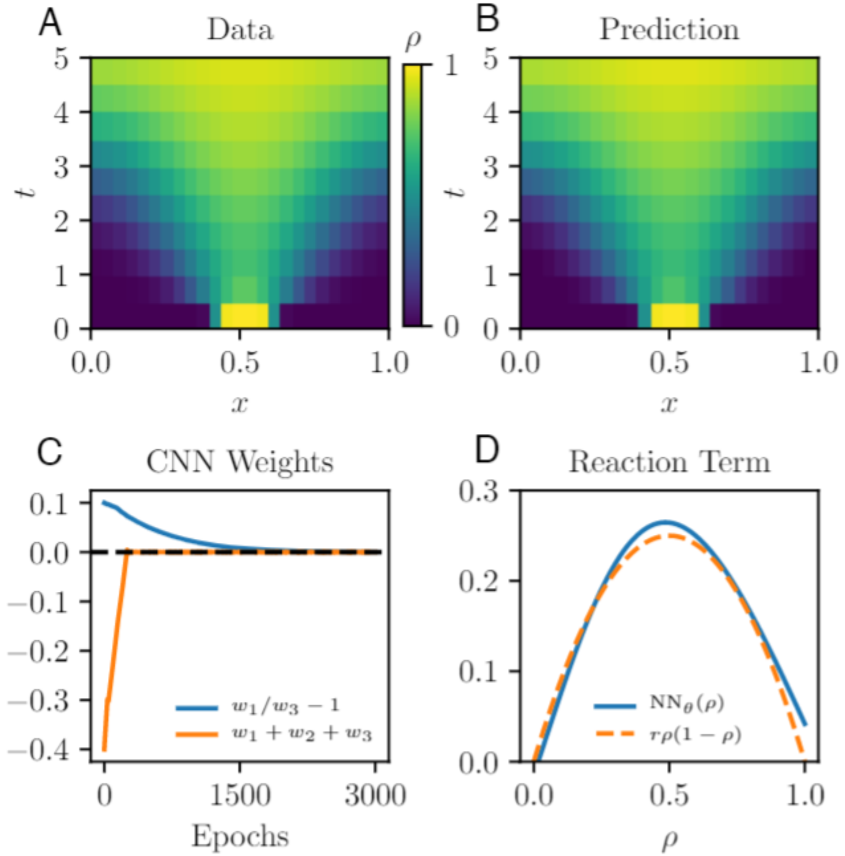
Truth: Fisher-KPP Equations

$$\rho_t = r\rho(1-\rho) + D\rho_{xx},$$

Truth: Universal Differential Equation

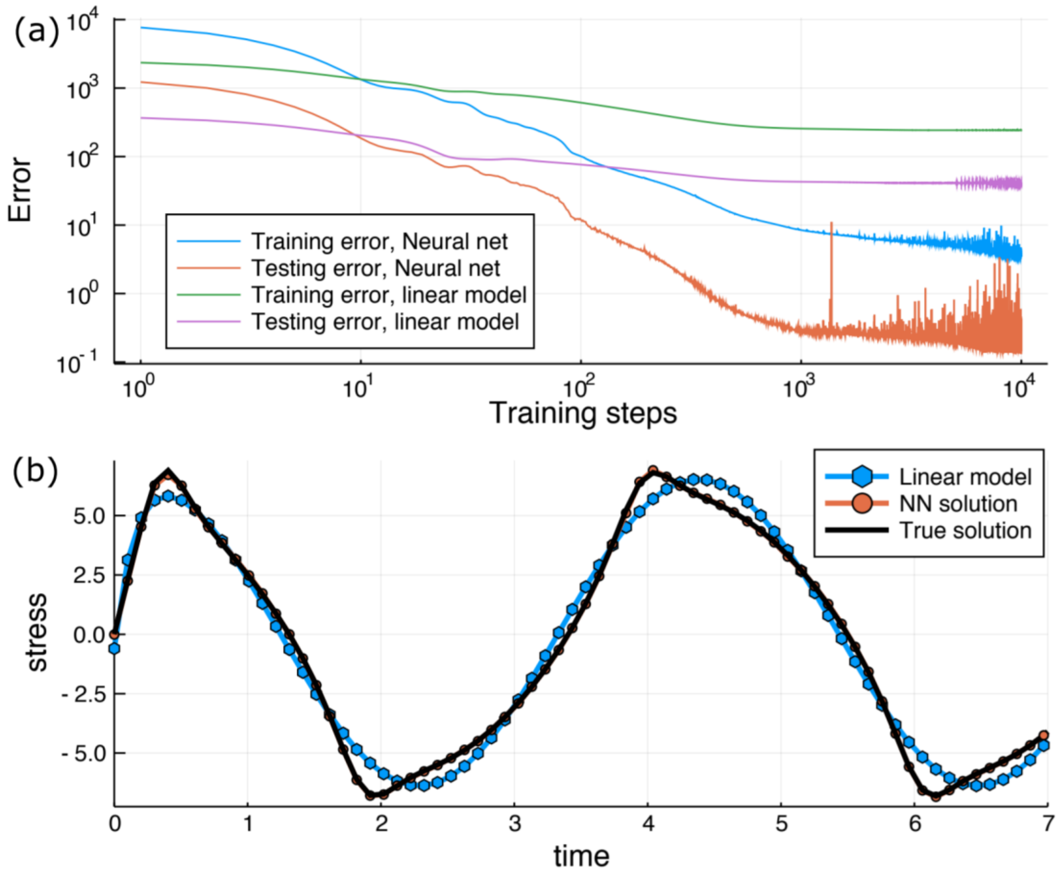
$$\rho_t = \mathrm{NN}_\theta(\rho) + D\,\mathrm{CNN}(\rho),$$

Automatically recover that the dynamical system has a diffusion operator and a quadratic reaction term!



Embedding Neural Networks into Scientific Simulation Can Also Be Used To Accelerate!

Universal ODEs Accelerate Non-Newtonian Fluid Simulations



$$\sigma(t) = \int_{-\infty}^{t} G(t-s)F(\dot{\gamma}(s))\,\mathrm{d}s, \qquad [15]$$

which depends on the history of deformation, with some memory function G (55). This is equivalent to the following instantaneous form:

$$\sigma(t) = \phi_1(t), \qquad [16]$$

$$\frac{\mathrm{d}\phi_1}{\mathrm{d}t} = G(0)F(\dot{\gamma}) + \phi_2, \qquad [17]$$

$$\frac{\mathrm{d}\phi_2}{\mathrm{d}t} = \frac{\mathrm{d}G(0)}{\mathrm{d}t}F(\dot{\gamma}) + \phi_3, \qquad [18]$$

$$\vdots$$

Transform a system
Of DAEs into
Parameterized
system of ODEs,
2x acceleration

$$\sigma(t) = U_0(\dot{\gamma}, \phi_1, \ldots, \phi_N),$$

$$\frac{d\phi_1}{dt} = U_1(\dot{\gamma}, \phi_1, \ldots, \phi_N),$$

$$\vdots$$

$$\frac{d\phi_N}{dt} = U_N(\dot{\gamma}, \phi_1, \ldots, \phi_N),$$

Universal PDEs for Acceleration: Automated Climate Parameterizations

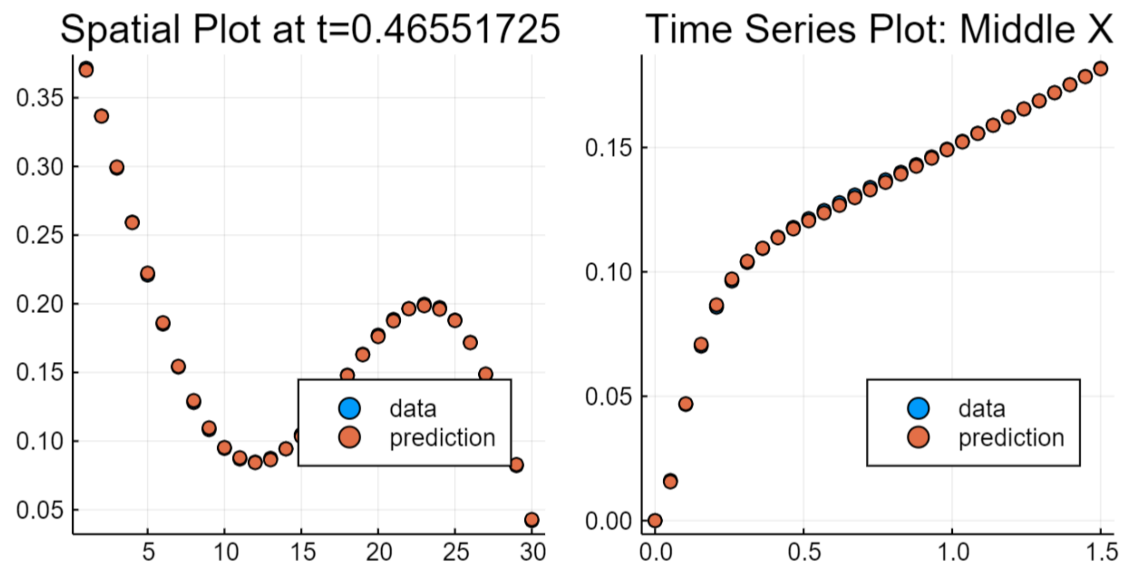


Fig. S1. Reduction of the Boussinesq equations. On the left is the comparison between the training data (blue) and the trained UPDE (ornage) over space at the 10th fitting time point, and on the right is the same comparison shown over time at spatial midpoint.

- Boussinesq Equations (Navier-Stokes) are used in climate models

$$\nabla \cdot \mathbf{u} = 0$$

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \mathrm{Pr}\nabla^2 \mathbf{u} + b\hat{z}$$

$$\frac{\partial b}{\partial t} + \mathbf{u} \cdot \nabla b = \nabla^2 b + Fe^z$$

- People attempt to solve this by “parameterizing”, i.e. getting a 1-dimensional approximation through averaging:

$$\left(\frac{\partial}{\partial t} + \mathbf{u}.\nabla - \nabla^2\right) c' - \frac{\partial}{\partial z}\overline{w'c'} = -w'\frac{\partial \bar{c}}{\partial z}$$

where $\overline{w'c'}$ is unknown.

- Instead of picking a form for $\overline{w'c'}$(the current method), replace it with a neural network and learn it from small scale simulations!

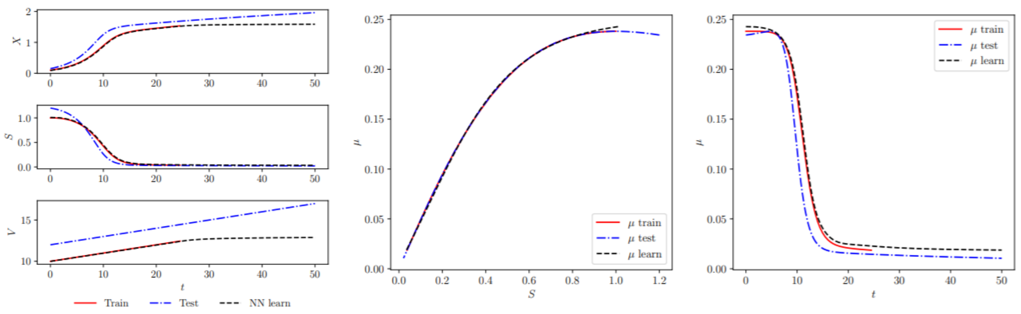
Universal Differential Equations extend previous physics-informed neural network and deep BSDE algorithms

UDE Methods Cover Accelerated Physics-Informed Neural Network Methods

$$l_n = \sum_{m=0}^{M} [\alpha_m y_{n-m} + \Delta t \beta_m f^{NN}(y_{n-m})], \quad n = M, \cdots, N.$$

$$loss(f^{NN}(y^{NN}(t))) = \frac{1}{N_y} \sum_{n=1}^{N_y} \left(y^{NN}(t_n) - y^*(t_n)\right)^2$$

$$+ \frac{1}{N_f} \sum_{n=1}^{N_f} \left(\frac{dy^{NN}}{dt}(t_n) - f^{NN}(y^{NN}(t_n)) \right)^2,$$



This methodology can be seen as a universal differential equation with a multistep integrator where adaptive=false

The UDE methodology thus gives an generalization to:

- Implicit methods, SSP methods
- Runge-Kutta-Chebyshev methods
- SDEs, DAEs, DDEs, etc.

A comparative study of physics-informed neural network models for learning unknown dynamics and constitutive relations Ramakrishna Tipireddy, Paris Perdikaris, Panos Stinis and Alexandre Tartakovsky

Multistep Neural Networks for Data-driven Discovery of Nonlinear Dynamical Systems Maziar Raissi, Paris Perdikaris , and George Em Karniadakis

Our results indicate that the accuracy of the trained neural network models is much higher for the cases where we only have to learn a constitutive relation instead of the whole dynamics.

Solving 1000 dimensional Hamilton-Jacobi-Bellman via Universal SDEs

- Semilinear Parabolic Form (Diffusion-Advection Equations, Hamilton-Jacobi-Bellman, Black-Scholes)

$$\frac{\partial u}{\partial t}(t,x)+\frac{1}{2}\mathrm{Tr}\left(\sigma\sigma^{\mathrm{T}}(t,x)(\mathrm{Hess}_x u)(t,x)\right)+\nabla u(t,x)\cdot\mu(t,x)$$
$$+f\left(t,x,u(t,x),\sigma^{\mathrm{T}}(t,x)\nabla u(t,x)\right)=0 \qquad \textbf{[1]}$$

Then the solution of Eq. **1** satisfies the following BSDE (cf., e.g., refs. 8 and 9):

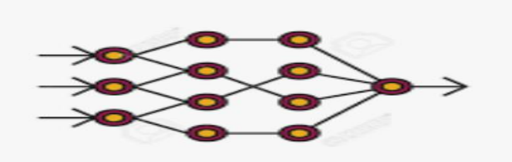
$$u(t,X_t)-u(0,X_0)$$
$$=-\int_0^t f\left(s,X_s,u(s,X_s),\sigma^{\mathrm{T}}(s,X_s)\nabla u(s,X_s)\right)ds \qquad \textbf{[3]}$$
$$+\int_0^t [\nabla u(s,X_s)]^{\mathrm{T}}\sigma(s,X_s)\,dW_s.$$

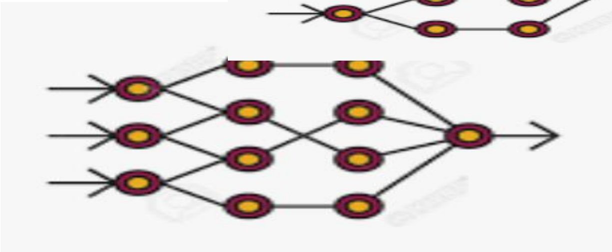
- Make $(\sigma^T \nabla \mathrm{u})(t, X)$ a neural network.
- Solve the resulting SDEs and learn $\sigma^T \nabla \mathrm{u}$ via:

$$l(\theta)=\mathbb{E}\left[\left|g(X_{t_N})-\hat{u}\left(\{X_{t_n}\}_{0\le n\le N},\{W_{t_n}\}_{0\le n\le N}\right)\right|^2\right].$$

Forward-Backward Stochastic Neural Networks: Deep Learning of High-dimensional Partial Differential Equations
Maziar Raissi

$$dX_t = \mu(t, X_t)dt + \sigma(t, X_t)dW_t,$$

$$dU_t = f(t, X_t, U_t,$$  $$)dt$$

$$+$$  $$dW_t,$$

Use high order, implicit, adaptive SDE solvers
Train a solution in minutes

Using non-adaptive explicit 0.5th order Euler-Maruyama matches the state-of-the-art deep BSDE methods from the literature

Solving high-dimensional partial differential equations using deep learning
Jiequn Han, Arnulf Jentzen, and Weinan E

UDEs are a BLAS/LAPACK of SciML

Scientific Machine Learning requires efficient and accurate training of UDEs

Efficient and robust software for UDEs in the Julia language result in efficient and robust implementations for many algorithms

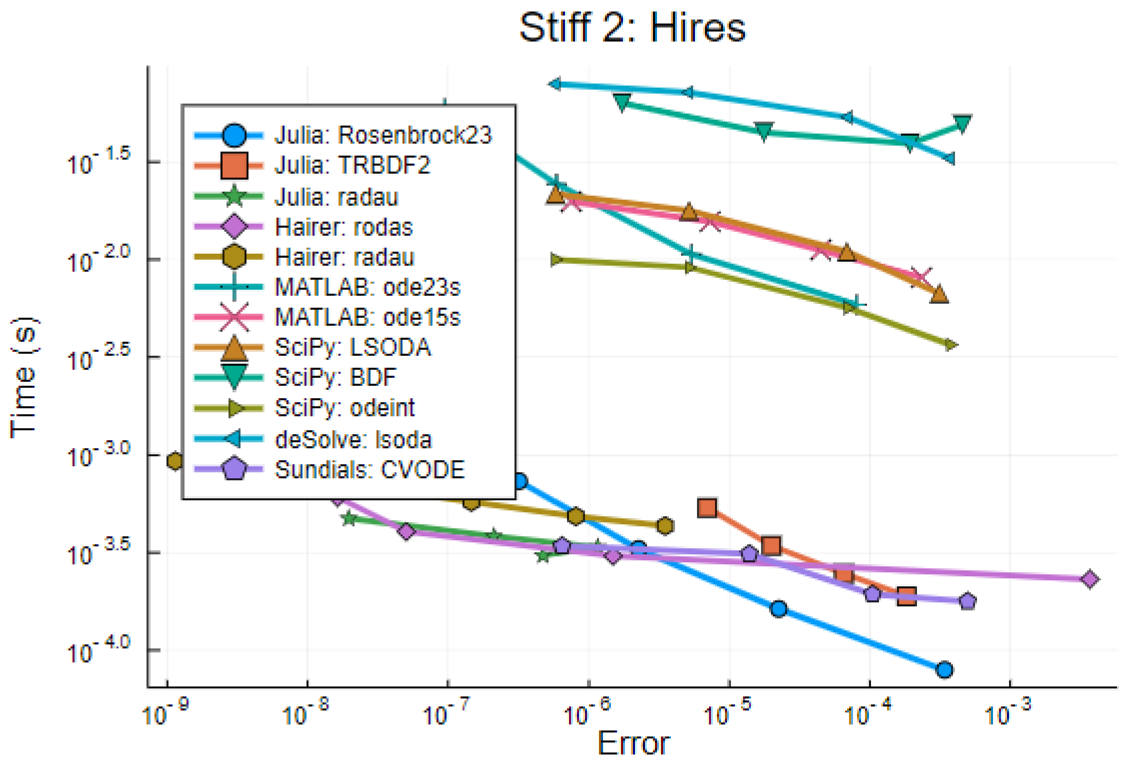
SciML Open Source Software Orgnaization

sciml.ai

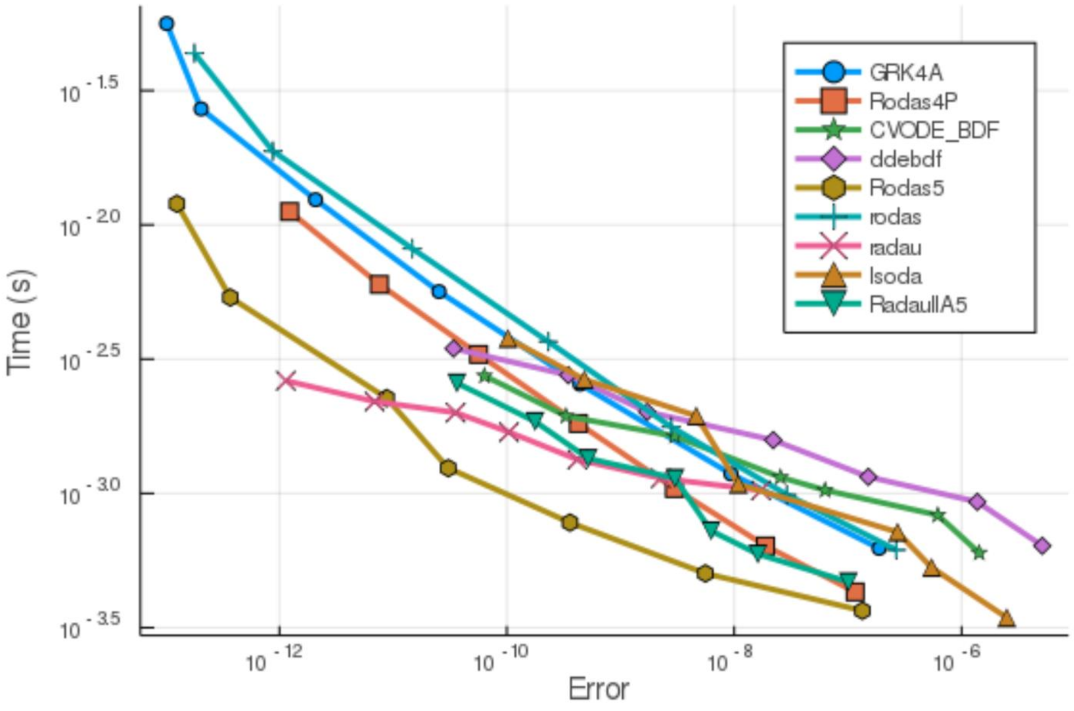
- DifferentialEquations.jl: high-performance differential equation solvers
- DiffEqFlux.jl: universal differential equation training optimizers, sensitivity analysis, and layer functions
- ModelingToolkit.jl: symbolic-numeric optimizations and automated parallelism
- NeuralPDE.jl: neural network solvers for PDEs, including automated physics-informed neural networks and deep BSDE methods for high dimensional PDEs
- Catalyst.jl: high-performance differentiable modeling of chemical reaction networks
- NBodySimulator.jl: high-performance differentiable molecular dynamics
- DataDrivenDiffEq.jl: Koopman Dynamic mode decomposition (DMD) methods and sparse identification (SInDy)

And 50 more libraries that cannot be fit!

SciML tools outperform ecosystems in high and low level languages



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



DifferentialEquations.jl's stiff ODE solvers can outperform SUNDIALS CVODE (C++) and Fortran methods like Radau

Speed drives researchers to Julia's SciML

Test problem: Lorenz equation

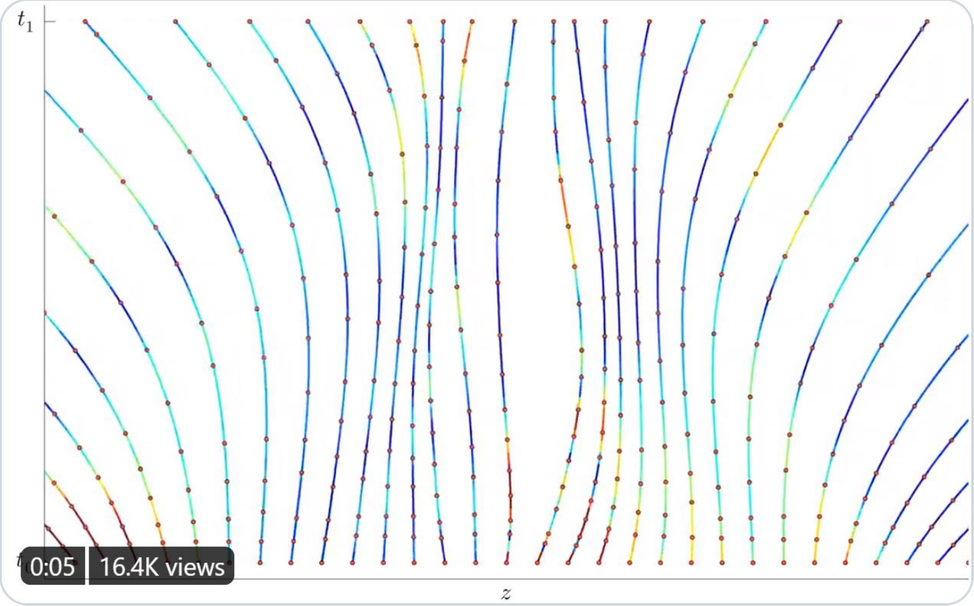
- DifferentialEquations.jl: 1.675 ms
- Jax: 3.66ms (*from author of Jax)
- Torchscript torchdiffeq: 48 seconds

Simple neural ODE training (2-dimensional neural ODE from Neural Ordinary Differential Equations Chen et al.):

- DifferentialEquations.jl: ~3 seconds (will show live!)
- Torchdiffeq: ~300 seconds

David Duvenaud @DavidDuvenaud · Jul 17

Neural ODEs are slow. We speed them up by regularizing their higher derivatives, learning ODEs that are easy to solve:
arxiv.org/pdf/2007.04504...
with @jacobjinkelly @jessebett @SingularMattrix



6 124 622

Jesse Bettencourt
@jessebett

Replying to @jessebett @TheyCallMeMr_ and 3 others

Here the training/evaluation of the dynamics neural network and its higher order derivatives are in JAX (python).
Not a problem in Julia, I used PyCall.jl to import the conda-environment + JAX code, calling my JAX neural nets, vmaps, and jets from within the Julia ODE solver!

4:12 PM · Jul 20, 2020 · Twitter Web App

Feature	SciML	Sundials (C++)	PETSc TS (C++)	torchdiffeq	Jax
Stiff ODEs and DAEs	Hundreds of methods tested and tuned on hundreds of problems	Yes (CVODE_BDF and IDA)	Yes (Rosenbrock-W methods, BDFs, etc.)	None	None (one in progress, ~200 times slower than SciPy according to the author!)
Adjoint Methods	8 choices tuned for different scenarios, including stabilized checkpointing, differentiate the solver, reversing adjoint	Stabilized checkpointing	Discrete sensitivity analysis (equivalent to differentiate through the solver)	Requires reversing the ODE or differentiate the solver	Requires reversing the ODE
Parallelism	GPU, MPI, multithreading	GPU, MPI, multithreading	GPU, MPI, and multithreading	GPU	GPU
Event handing	Yes	Yes	Yes	None	None
SDEs	Lots of methods, including stabilized, methods for stiff equations, high strong order, high weak order	None	None	torchsde, only diagonal noise (or order 0.5), requires reversing the SDE	None
Delays	All ODE methods	None	None	None	None

What do these features mean?

SciML is not just for speed

SCIML IS FOR FLEXIBILITY, ACCURACY, AND CORRECTNESS

WARNING: SOME “EXPERT” TALK HERE

SciML is meticulously tested

✓ master Merge pull request #1215 from utkarsh530/extplmethods

Multi-threading for Implicit Extrapolation Methods

Commit a09a464
Compare 96c3949..a09a464
Branch master

Christopher Rackauckas

#5442 passed

Restart build

Ran for 1 hr 26 min 58 sec
Total time 6 hrs 16 min 8 sec

18 hours ago

Build jobs | View config

✓	# 5442.1	AMD64	Xenial	Julia: 1	GROUP=InterfaceI	44 min 59 sec
✓	# 5442.2	AMD64	Xenial	Julia: 1	GROUP=InterfaceII	31 min 34 sec
✓	# 5442.3	AMD64	Xenial	Julia: 1	GROUP=Integrators_I	34 min 53 sec
✓	# 5442.4	AMD64	Xenial	Julia: 1	GROUP=Integrators_II	41 min 23 sec
✓	# 5442.5	AMD64	Xenial	Julia: 1	GROUP=Regression_I	31 min 45 sec
✓	# 5442.6	AMD64	Xenial	Julia: 1	GROUP=Regression_II	26 min 47 sec
✓	# 5442.7	AMD64	Xenial	Julia: 1	GROUP=AlgConvergence_I	37 min 57 sec
✓	# 5442.8	AMD64	Xenial	Julia: 1	GROUP=AlgConvergence_II	44 min 16 sec
✓	# 5442.9	AMD64	Xenial	Julia: 1	GROUP=AlgConvergence_III	28 min 43 sec
✓	# 5442.10	AMD64	Xenial	Julia: 1	GROUP=Downstream	41 min 42 sec
✓	# 5442.11	AMD64	Xenial	Julia: 1	GROUP=ODEInterfaceRegression	12 min 9 sec

Full test suite is over a day of events, gradients, GPUs, convergence, stochastic distributions, etc.

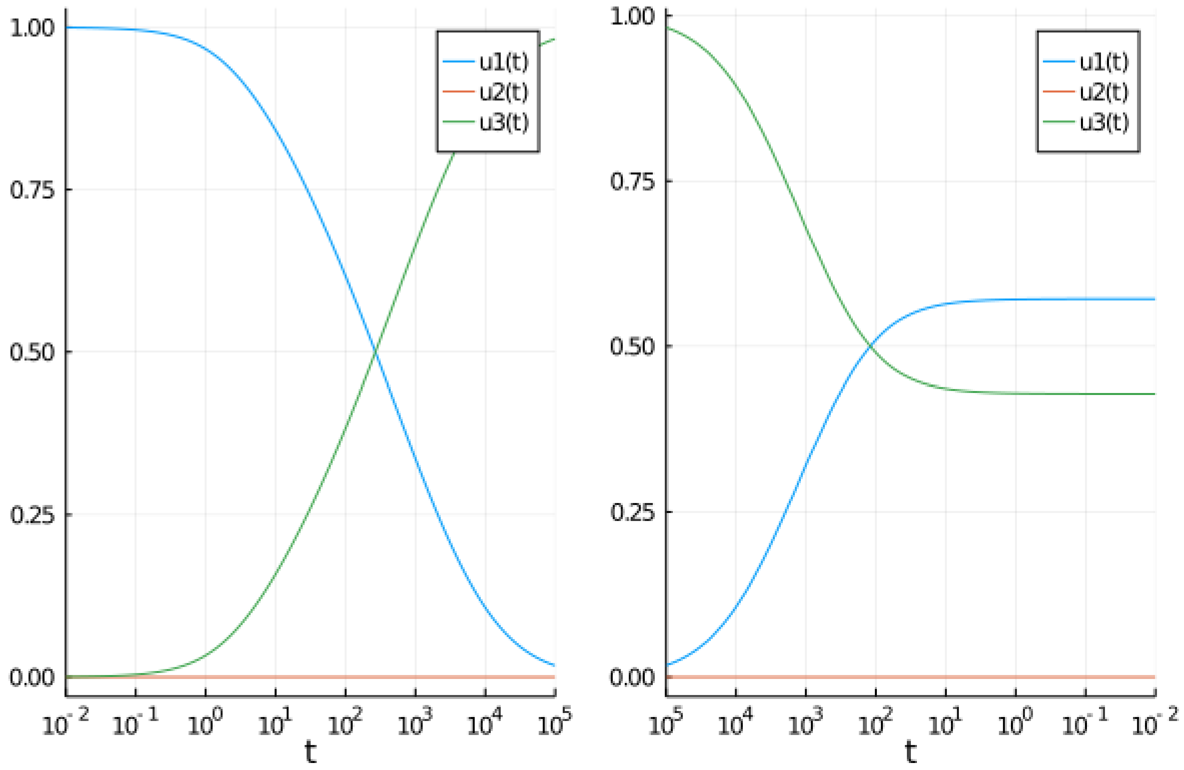
Match behaviors exactly in pure Julia, and **fix bugs from the widely used Fortran code (deSolve, SciPy)**

```
sol1 =solve(prob,DP5(),dt=1/8)
sol2 =solve(prob,dopri5(),dt=1/8)
@test sol1.t ≈ sol2.t
```

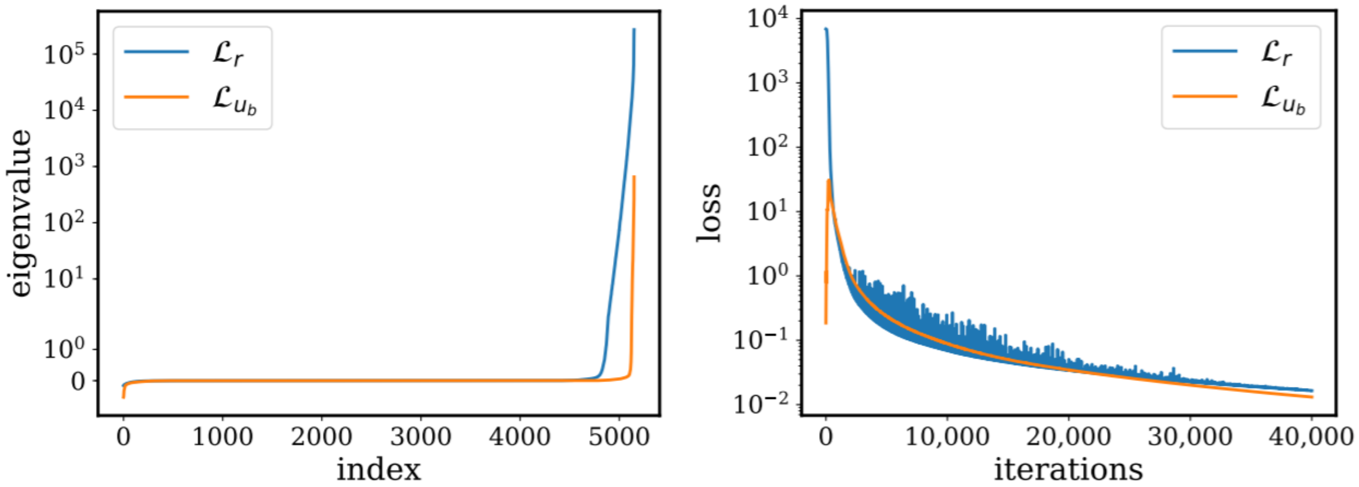
SciML's tools do not rely on properties which can fail to hold

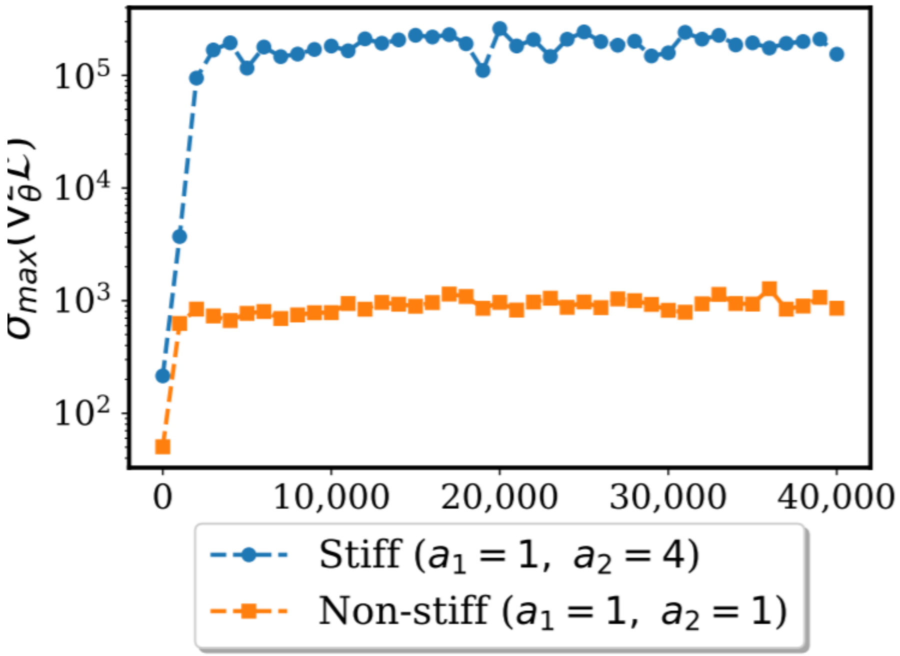
```
using DifferentialEquations, ODEInterfaceDiffEq, Plots
function rober(du,u,p,t)
  y₁,y₂,y₃ = u
  k₁,k₂,k₃ = p
  du[1] = -k₁*y₁+k₃*y₂*y₃
  du[2] =  k₁*y₁-k₂*y₂^2-k₃*y₂*y₃
  du[3] =  k₂*y₂^2
  nothing
end
prob = ODEProblem(rober,[1.0,0.0,0.0],(0.0,1e5),(0.04,3e7,1e4))
sol = solve(prob,DP5(),reltol=1e-10,abstol=1e-10) # Fails!

sol = solve(prob,Rosenbrock23())
p1 = plot(sol,tspan=(1e-2,1e5),xscale=:log10)
prob_reverse = ODEProblem(rober,sol[end],(1e5,0.0),(0.04,3e7,1e4))
sol2 = solve(prob_reverse,Rosenbrock23())
p2 = plot(sol2,tspan=(1e-2,1e5),xscale=:log10)
plot(p1,p2)
```



Ill-Conditioned Gradients Cause Difficulties in Scientific Machine Learning





Understanding and mitigating gradient pathologies in physics-informed neural networks
Sifan Wang, Yujun Teng, Paris Perdikaris

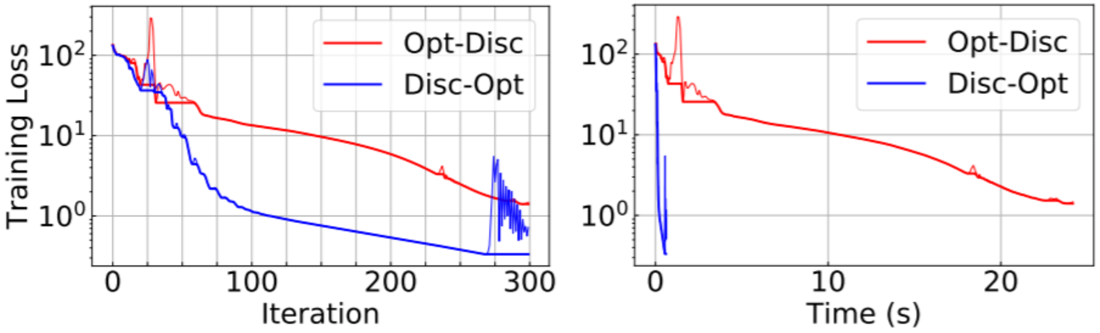
Off-the-shelf ML tools will not work on stiff scientific machine learning problems!

DiffEqFlux has the features to handle stiff ill-conditioned scientific problems

- 300 highly optimized differential equation solvers for highly stiff equations:
  - ROCK methods
  - Implicit methods (ODEs, SDEs, DAEs, DDEs)
  - Multistep methods
  - SSP methods (hyperbolic PDEs)
  - Adaptive SDE solvers (implicit, high order)
  - Event handling
- And these implementations are well-optimized:
  - DiffEqFlux trains the neural ODE from the original neural ODE paper in ~3 seconds
  - torchscript torchdiffeq : ~300 seconds

Hessian condition number 10^{13} effectively trained in tutorials

- Mixed AD Hessian-free Newton-Krylov for robust second order optimization
- Discrete and continuous sensitivity analysis (checkpointed, stabilized, etc.)



Discretize-Optimize vs. Optimize-Discretize for Time-Series Regression and Continuous Normalizing Flows
Derek Onken, Lars Ruthotto

DiffEqFlux.jl has the bells and whistles to solve "real" problems

Neural ODE with batching on the GPU (without internal data transfers) with high order adaptive implicit ODE solvers for stiff equations using matrix-free Newton-Krylov via preconditioned GMRES and trained using checkpointed adjoint equations.

```
using OrdinaryDiffEq, Flux, DiffEqFlux, DiffEqOperators, CuArrays
x = Float32[2.; 0.]|>gpu
tspan = Float32.((0.0f0,25.0f0))
dudt = Chain(Dense(2,50,tanh),Dense(50,2))|>gpu
p = DiffEqFlux.destructure(dudt)
dudt_(du,u::TrackedArray,p,t) =  u .= DiffEqFlux.restructure(dudt,p)(u)
dudt_(du,u::AbstractArray,p,t) = u .= Flux.data(DiffEqFlux.restructure(dudt,p)(u))
ff = ODEFunction(dudt_,jac_prototype = JacVecOperator(dudt_,x))
prob = ODEProblem(ff,x,tspan,p)
diffeq_adjoint(p,prob,KenCarp4(linsolve=LinSolveGMRES());u0=x,
               saveat=0.0:0.1:25.0,backsolve=false)
```

Workshop Outline

- ~~Overview of scientific machine learning and SciML~~
  - ~~What is scientific machine learning?~~
  - ~~What makes the SciML ecosystem unique?~~
- Modeling with differential equations
  - Solving differential equations with DifferentialEquations.jl
  - Adding stochasticity, delays, events
- Introduction to challenge and learning problems
  - Workshop exercises (with answers!)
  - HelicopterSciML Challenge Problem
  - **Magnetic Navigation Challenge Problem**
- Automated model discovery via universal differential equations
  - Parameter inference on differential equations
    - Local and global optimization
    - Bayesian optimization
  - Mixing DiffEqFlux.jl and DataDrivenDiffEq.jl!
- Solving differential equations with neural networks (physics-informed neural networks)

Now let's get to coding

SOLVING DIFFERENTIAL EQUATIONS WITH STOCHASTICITY, DELAYS, AND EVENTS

AND THEN ADD SOME PARALLELISM

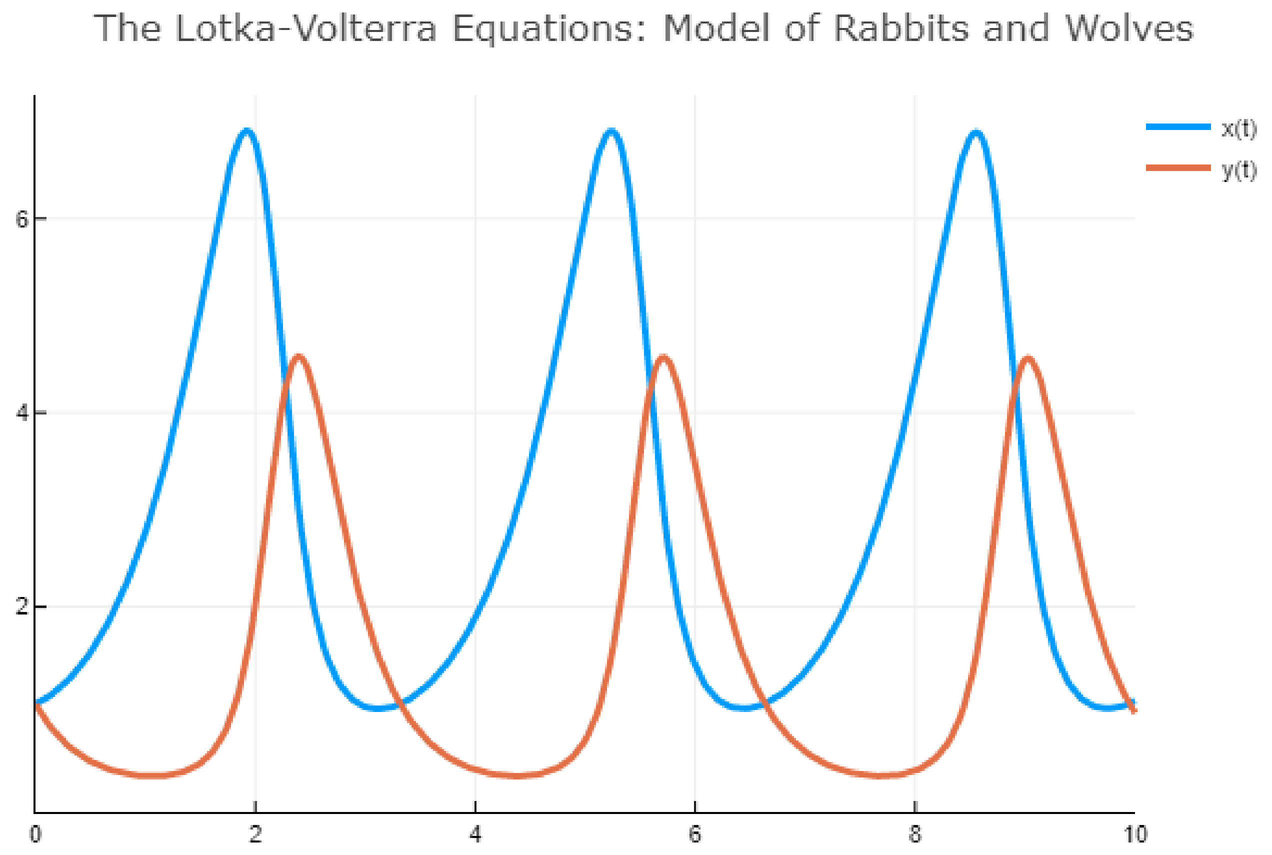
Let’s start coding some models: Lotka-Volterra Equations

$$\frac{d\text{🐇}}{dt} = \alpha\text{🐇} - \beta\text{🐇}\text{🐺}$$

Exponential growth — Gets eaten by wolves

$$\frac{d\text{🐺}}{dt} = \delta\text{🐇}\text{🐺} - \gamma\text{🐺}$$

Increases with more food — Decreases with competition



ODE Solver Packages on the Common Interface

You do not need to change your code to use solves on the common interface! Julia might have the largest number of active developers in the field! Other great solvers you should check out:

- OrdinaryDiffEq.jl: The workhorse
- Sundials.jl: CVODE_BDF is a great stiff ODE solver
- ODEInterfaceDiffEq.jl: radau is great for stiff ODEs at low tolerances (<1e-8)
- LSODA.jl: lsoda is all-around good for smaller ODEs (<100)
- IRKGaussLegendre.jl: IRKGL16 is 16^{th} order and symplectic, great for physical problems at really low tolerances (<1e-12)
  - **JuliaCon: Implicit RK solver for high precision numerical integration**
- TaylorIntegration.jl: Great at low tolerances, can give error bounds
- NeuralPDE.jl: parallized-in-time physics-informed neural network methods
  - **JuliaCon: Julia Track Google Code In and Beyond**
  - **JuliaCon: Minisymposium on Partial Differential Equations**
- GeometricIntegratorsDiffEq.jl: Great fixed time-step methods for small ODEs (symplectic)
- QuDiffEq.jl: Great ODE solvers if you have a quantum computer and need to output QASM
- TimeMachine.jl: A priori time stepping from Clima, great for multi-node MPI problems

(Differentiable) Modeling Frameworks

- ModelingToolkit.jl: symbolic-numerics for accelerated modeling
  - **JuliaCon: Auto-Optimization and Parallelism in DifferentialEquations.jl**
- Catalyst.jl: chemical reaction networks
- Petri.jl and AlgebraicPetri.jl: Petri networks and applied category theory
- NetworkDynamics.jl: dynamics on networks
  - **JuliaCon: NetworkDynamics.jl - Modeling dynamical systems on networks**
- PowerSimulationsDynamics.jl: dynamics of power grids
  - **JuliaCon: Crash Course in Energy Systems Modeling and Analysis with Julia**
- JuSDL.jl: causal modeling that can mix the various differential equations
  - **JuliaCon: Jusdl.jl - Julia Based System Description Language**
- BioEnergeticFoodWebs.jl: simulations of biomass flows
- QuantumOptics.jl: simulations of quantum systems
- DynamicalSystems.jl: dynamical systems and chaos analysis
- RigidBodySim.jl: simulations of rigid-body dynamics and robotics

And so many more!

github.com/epirecipes/sir-julia
Various implementations of the classical SIR model in Julia

Workshop Outline

- ~~Overview of scientific machine learning and SciML~~
  - ~~What is scientific machine learning?~~
  - ~~What makes the SciML ecosystem unique?~~
- ~~Modeling with differential equations~~
  - ~~Solving differential equations with DifferentialEquations.jl~~
  - ~~Adding stochasticity, delays, events~~
- Introduction to challenge and learning problems
  - Workshop exercises (with answers!)
  - HelicopterSciML Challenge Problem
  - **Magnetic Navigation Challenge Problem**
- Automated model discovery via universal differential equations
  - Parameter inference on differential equations
    - Local and global optimization
    - Bayesian optimization
  - Mixing DiffEqFlux.jl and DataDrivenDiffEq.jl!
- Solving differential equations with neural networks (physics-informed neural networks)

SciML challenge and learning problems

Workshop Exercise Sheet

- https://tutorials.sciml.ai/html/exercises/01-workshop_exercises.html
  - Lots of exercises, from beginner to advanced
  - Problems on performance optimization, parameter inference, neural ODEs

Solutions: https://tutorials.sciml.ai/html/exercises/02-workshop_solutions.html

HelicopterSciML Challenge Problem: Learn missing physics!

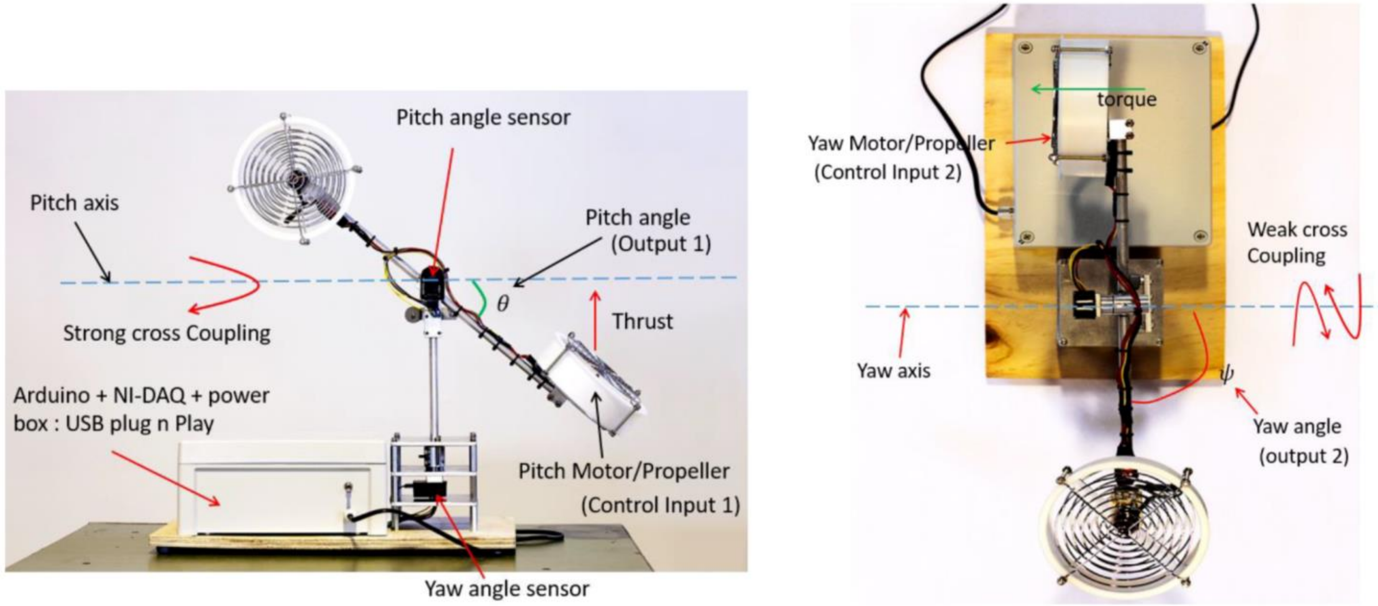


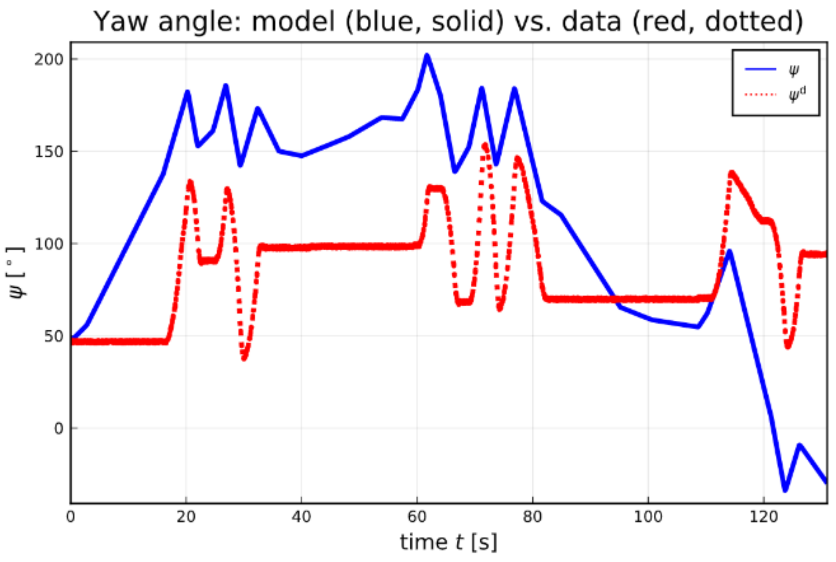
Figure 1: Laboratory helicopter, Sharma (2020).

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

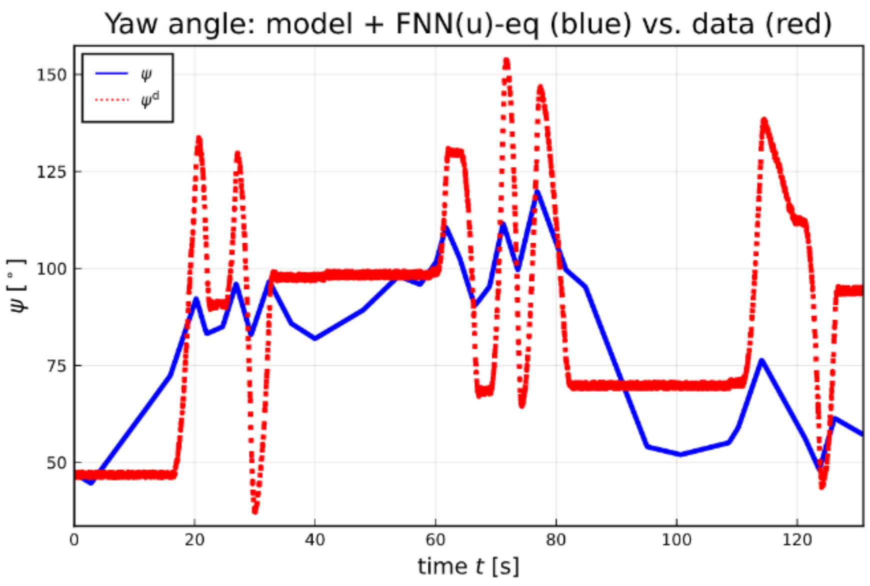
Goal: Discover the unexplained physics of this system

HelicopterSciML Challenge Problem Example Solution

Before Augmentation



After Discovery



Discovered missing higher order friction terms

$$\mathrm{FNN}(u;p)_1 \approx -4.37 \cdot 10^{-4} \cos(u_\psi) + 4.02 \cdot 10^{-4} \sin(u_\psi)$$

$$\mathrm{FNN}(u;p)_2 \approx -1.35 \cdot 10^{-2} \cos(u_\psi) + 7.74 \cdot 10^{-3} u_\psi^2.$$

Magnetic Navigation Challenge Problem

HTTPS://GITHUB.COM/MIT-AI-ACCELERATOR/MAGNAV.JL

Workshop Outline

- ~~Overview of scientific machine learning and SciML~~
  - ~~What is scientific machine learning?~~
  - ~~What makes the SciML ecosystem unique?~~
- ~~Modeling with differential equations~~
  - ~~Solving differential equations with DifferentialEquations.jl~~
  - ~~Adding stochasticity, delays, events~~
- ~~Introduction to challenge and learning problems~~
  - ~~Workshop exercises (with answers!)~~
  - ~~HelicopterSciML Challenge Problem~~
  - **~~Magnetic Navigation Challenge Problem~~**
- Automated model discovery via universal differential equations
  - Parameter inference on differential equations
    - Local and global optimization
    - Bayesian optimization
  - Mixing DiffEqFlux.jl and DataDrivenDiffEq.jl!
- Solving differential equations with neural networks (physics-informed neural networks)

Now let's do some model inference

LEARN THE PARAMETERS OF A DIFFERENTIAL EQUATION

THEN LEARN THE MISSING PIECES OF A DIFFERENTIAL EQUATION

Universal ODEs learn and extrapolate other dynamical behaviors

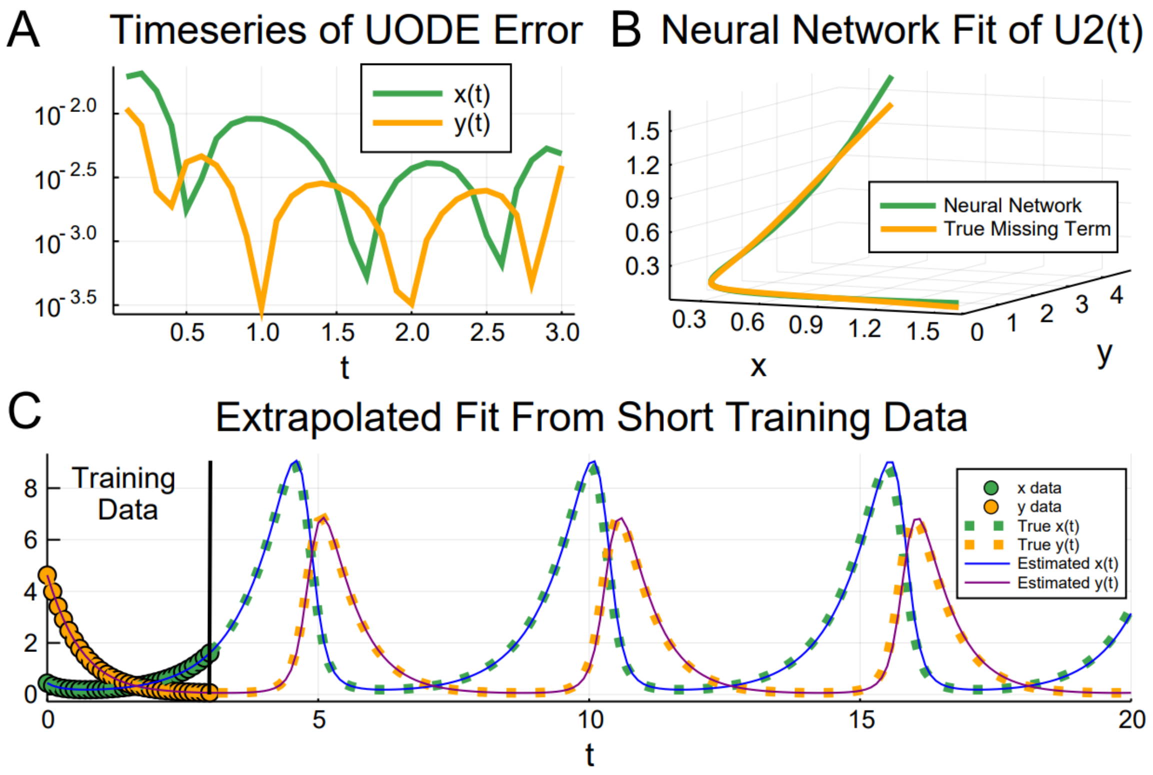
Truth

$$\dot{x} = \alpha x - \beta x y,$$
$$\dot{y} = \gamma x y - \delta y.$$

Partially-known neural embedded equations

$$\dot{x} = \alpha x - U_1(x, y),$$
$$y = -\delta y + U_2(x, y),$$

Automatically recover the long-term behavior from less than half of a period in a cyclic time series!



Turn neural networks back into equations with SInDy. Let's do this example!

Packages for model inference

- DiffEqFlux.jl: helpers for performing inference on models. Interface over:
  - Optim.jl: workhorse optimizers like BFGS
  - Flux.jl: specialized neural network optimizers like ADAM
  - BlackBoxOptim.jl: very robust global optimizers
  - Evolutionary.jl: genetic algorithms and CMA
  - And many more!
- DataDrivenDiffEq.jl: methods for Koopman DMD and SInDy (turning data into equations!)
- Turing.jl: Bayesian estimation
- Gen
- GalacticOptim.jl: differentiable local+global optimizer interface. Coming soon!

Workshop Outline

- ~~Overview of scientific machine learning and SciML~~
  - ~~What is scientific machine learning?~~
  - ~~What makes the SciML ecosystem unique?~~
- ~~Modeling with differential equations~~
  - ~~Solving differential equations with DifferentialEquations.jl~~
  - ~~Adding stochasticity, delays, events~~
- ~~Introduction to challenge and learning problems~~
  - ~~Workshop exercises (with answers!)~~
  - ~~HelicopterSciML Challenge Problem~~
  - **~~Magnetic Navigation Challenge Problem~~**
- ~~Automated model discovery via universal differential equations~~
  - ~~Parameter inference on differential equations~~
    - ~~Local and global optimization~~
    - ~~Bayesian optimization~~
  - ~~Mixing DiffEqFlux.jl and DataDrivenDiffEq.jl!~~
- Solving differential equations with neural networks (physics-informed neural networks)

NeuralPDE.jl: Automated PDE Solving via Neural Networks

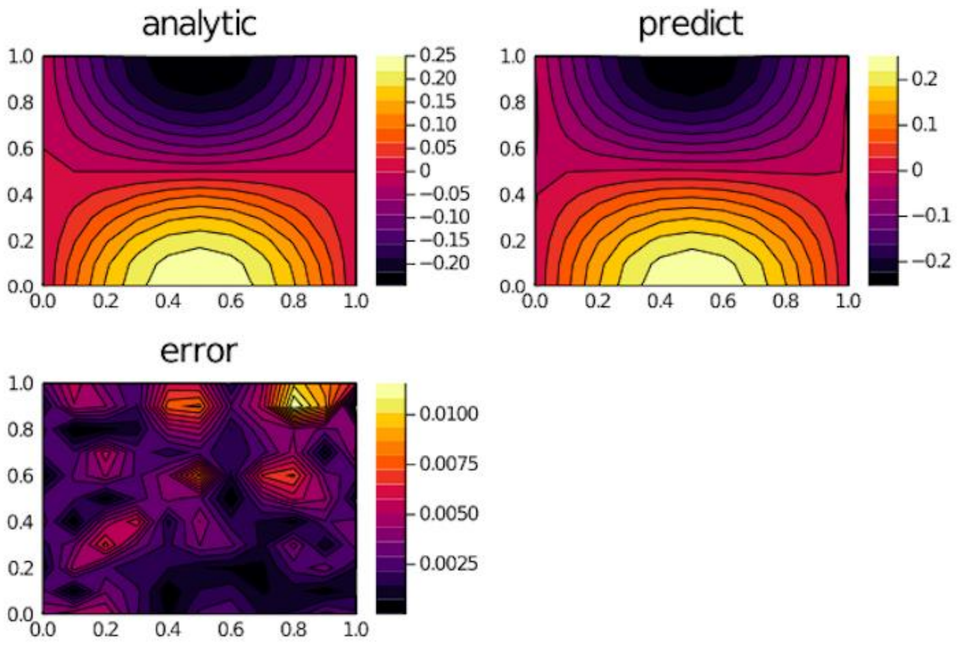
```
@parameters x y θ
@variables u(..)
@derivatives Dxx''~x Dyy''~y

eq  = Dxx(u(x,y,θ)) + Dyy(u(x,y,θ)) ~ -sin(pi*x)*sin(pi*y)
bcs = [u(0,y) ~ 0.f0, u(1,y) ~ -sin(pi*1)*sin(pi*y),
       u(x,0) ~ 0.f0, u(x,1) ~ -sin(pi*x)*sin(pi*1)]
domains = [x ∈ IntervalDomain(0.0,1.0),y ∈ IntervalDomain(0.0,1.0)]
discretization = PhysicsInformedNN(0.1)

opt = Flux.ADAM(0.02)
chain = FastChain(FastDense(2,16,Flux.σ),FastDense(16,16,Flux.σ),FastDense(16,1))

pde_system = PDESystem(eq,bcs,domains,[x,y],[u])
prob = discretize(pde_system,discretization)
alg = NNDE(chain,opt,autodiff=false)

phi,res  = solve(prob,alg,verbose=true, maxiters=5000)
```



What is this library doing?

- The deep BSDE method
  - Mentioned earlier: can be transformed into a universal stochastic differential equation and solved via DiffEqFlux.jl
- Physics-informed neural networks

For understanding, let's build the simplest physics-informed neural network from scratch!

Solve an ODE with a neural network

- Let $u' = f(u,t)$ with $u(0) = u_0$. We want to build a neural network $\mathrm{NN}(t)$ that is the solution to this differential equation.
- By definition then, we must have that $NN'(t) = f(NN(t),t)$ and $NN(0) = u_0$
- Define $C(\theta) = \sum_t ||NN'(t) - f(NN(t),t)||$ for θ the parameters of the ODE
  - Then this cost is zero when $NN(t)$ is the solution to the ODE
  - Therefore minimize this loss to get the solution!
- Extra trick: $g(t) = tNN(t) - u_0$ is an approximator that always satisfies the boundary condition

Why Physics-Informed Neural Networks?

- $C(\theta) = C_{pde}(\theta) + C_{boundary}(\theta) + C_{data}(\theta)$ can nudge a model towards data
  - Equivalent to regularizing the neural network by a scientific equation
- Can train fast continuous surrogates by making the neural network parameter dependent

Time to build a physics-informed neural network in Flux!

Thank you! For more information, check out JuliaCon starting next week!

Probabilistic Optimization with the Koopman Operator, July 29th
SciML: Automatic Discovery of droplet fragmentation Physics, July 29th
Exploring Disease Vector Dynamics Under Environmental Change, July 29th
NetworkDynamics.jl - Modeling dynamical systems on networks, July 30th
Automated optimization and parallelism with DifferentialEquations.jl, July 31st

all feature SciML tools, along with many, many more!

Mix neural networks with FDM, FVM, FEM, pseudospectral methods, implicit ODE solvers, high order adaptive SDE solver, …

Julia's SciML software ecosystem is built to handle the sparse, stiff, and ill-conditioned problems of real science