Generalizing Scientific Machine Learning through Differentiable Simulation

Chris Rackauckas

VP of Modeling and Simulation,
Julia Computing

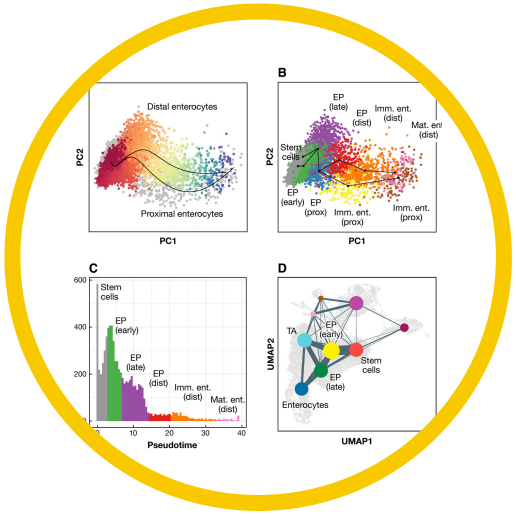
Research Affiliate, Co-PI of Julia Lab,
Massachusetts Institute of Technology, CSAIL

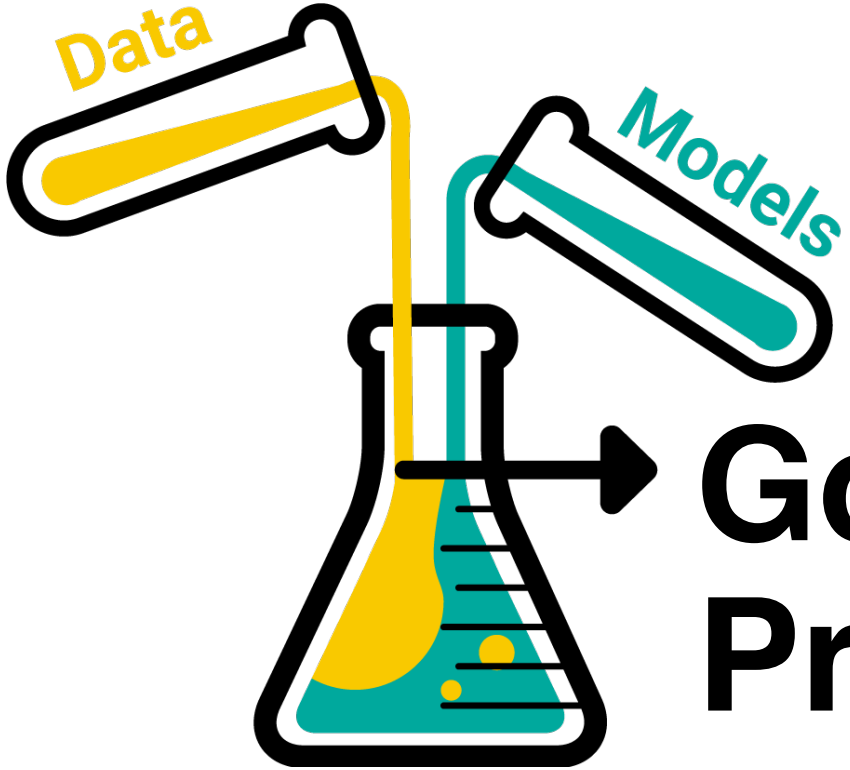
Director of Scientific Research,
Pumas-AI

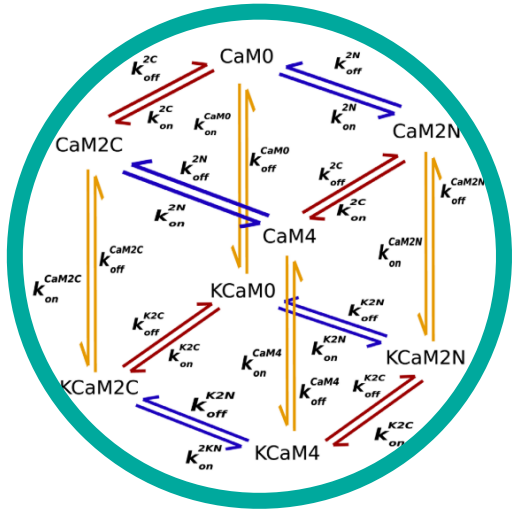
*CTO, Neuroblox**

Scientific Machine Learning is model-based data-efficient machine learning

How do we simultaneously use both sources of knowledge?





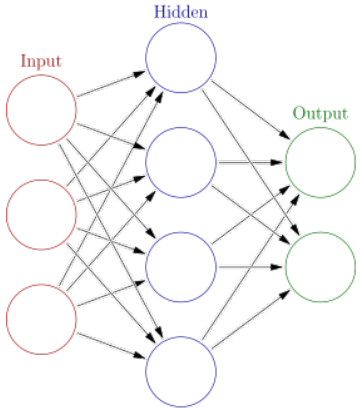


Good Predictions

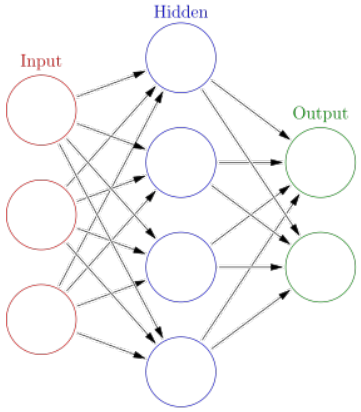
Core Point: Differentiable Simulation is hard, but it's worth investing more development time in.

Universal (Approximator) Differential Equations

$$u' = f(u, \quad , t)$$

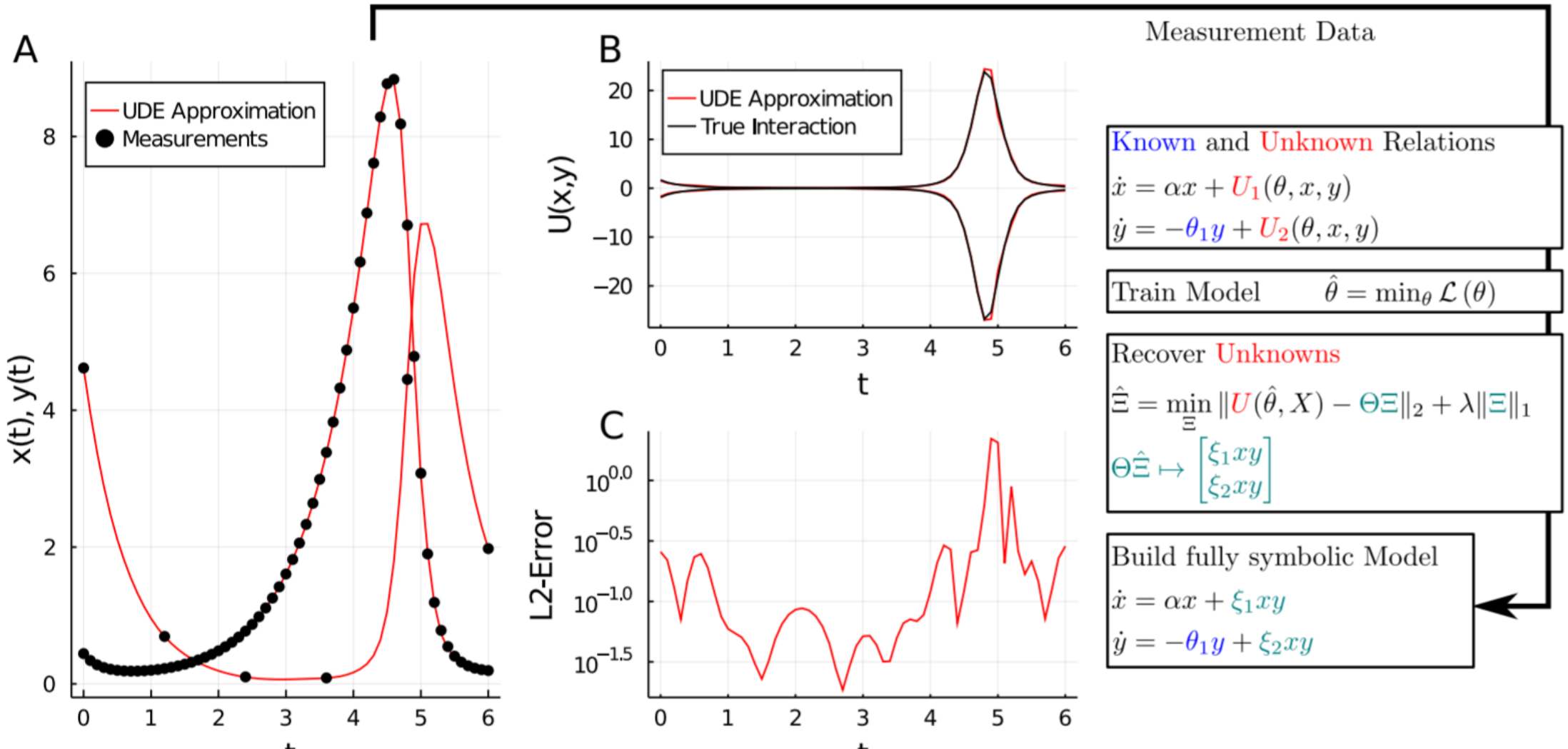


$$x' = \alpha x \quad +$$
$$y' = -\beta y +$$



Rackauckas, Christopher, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, Ali Ramadhan, and Alan Edelman. "Universal differential equations for scientific machine learning." *arXiv preprint arXiv:2001.04385* (2020).

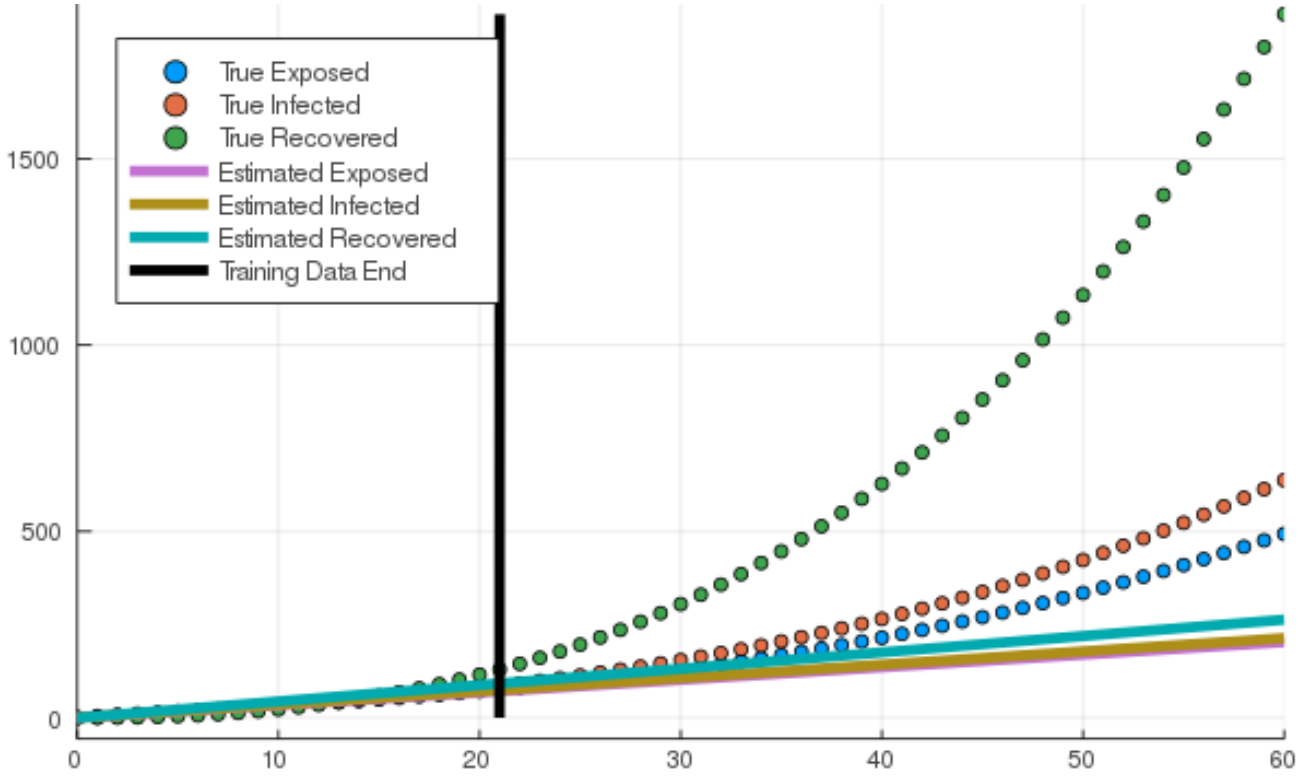
Universal (Approximator) Differential Equations



Julia SIM

Let's dive in a bit!

Standard Machine Learning: Learn the whole model



u'=NN(u) trained on 21 days of data

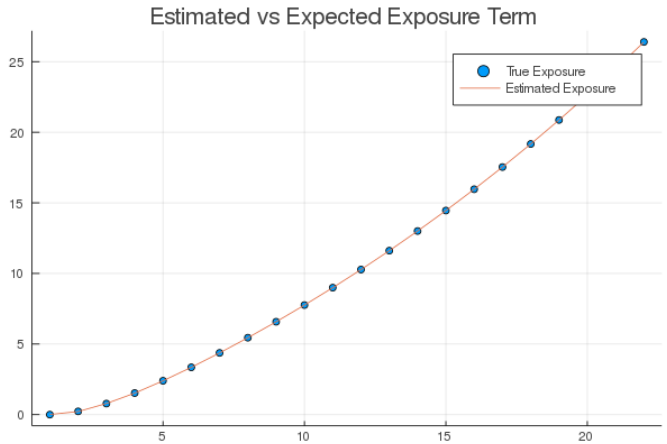
Can fit, but not enough information to accurately extrapolate

Does not have the correct asymptotic behavior

More examples of this issue:

Ridderbusch et al. "Learning ODE Models with Qualitative Structure Using Gaussian Processes."

Universal ODE



$$
\begin{aligned}
S' &= -\frac{\beta_0 S F}{N} - \mu S, \\
E' &= \frac{\beta_0 S F}{N} - \left(\sigma + \mu\right) E, \\
I' &= \sigma E - (\gamma + \mu) I, \\
R' &= \gamma I - \mu R, \\
N' &= -\mu N, \\
D' &= d\,\gamma I - \lambda D, \quad \text{and} \\
C' &= \sigma E,
\end{aligned}
$$

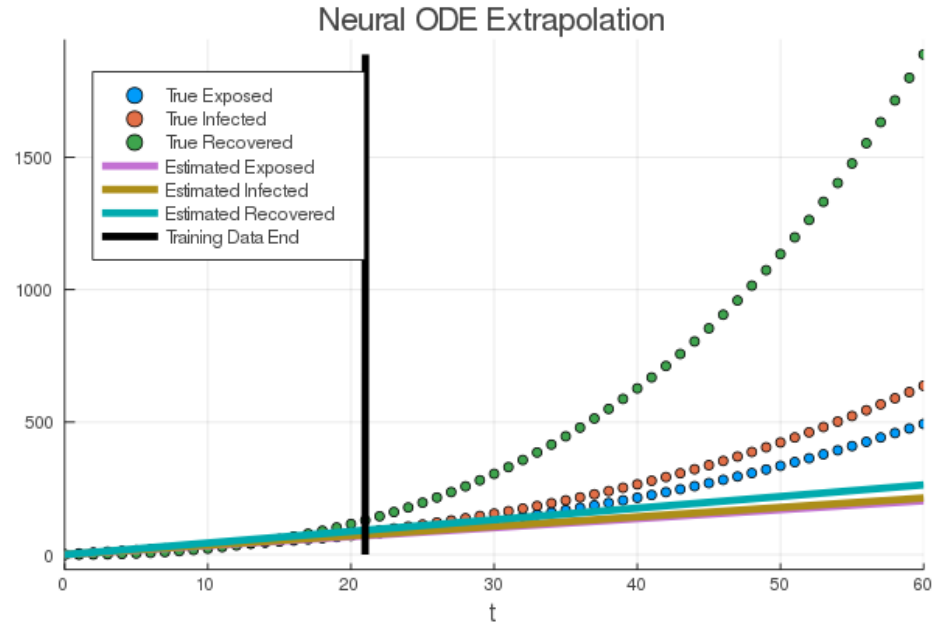
Replace Unknown Portion

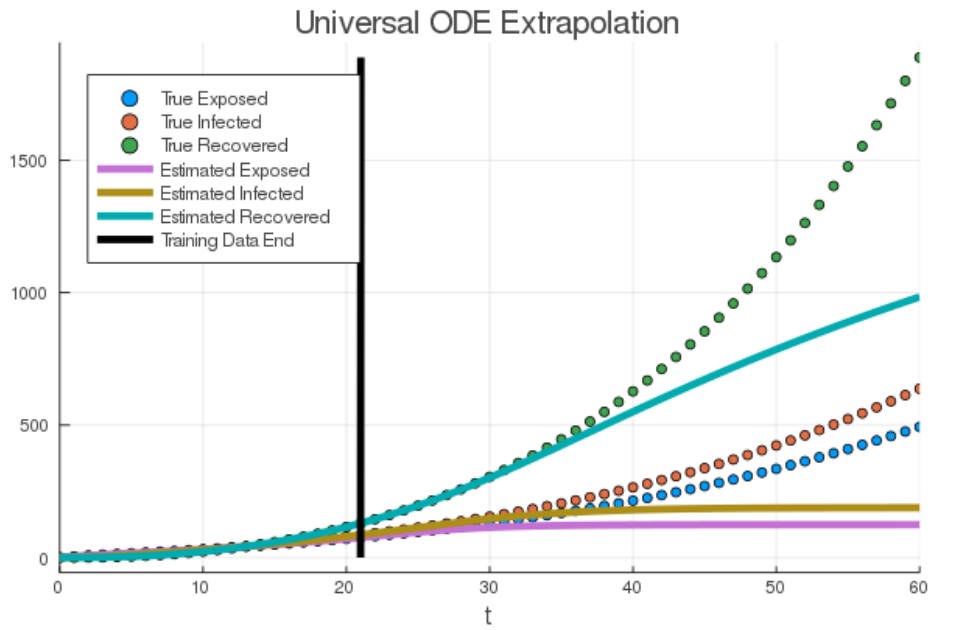
Replace Unknown Portion

Exposure: **Unknown**

Infection rates: known From disease quantities

Percentage of cases known to be severe, can be estimated





Universal ODE -> Internal Sparse Regression

Sparse Identification on only the missing term:

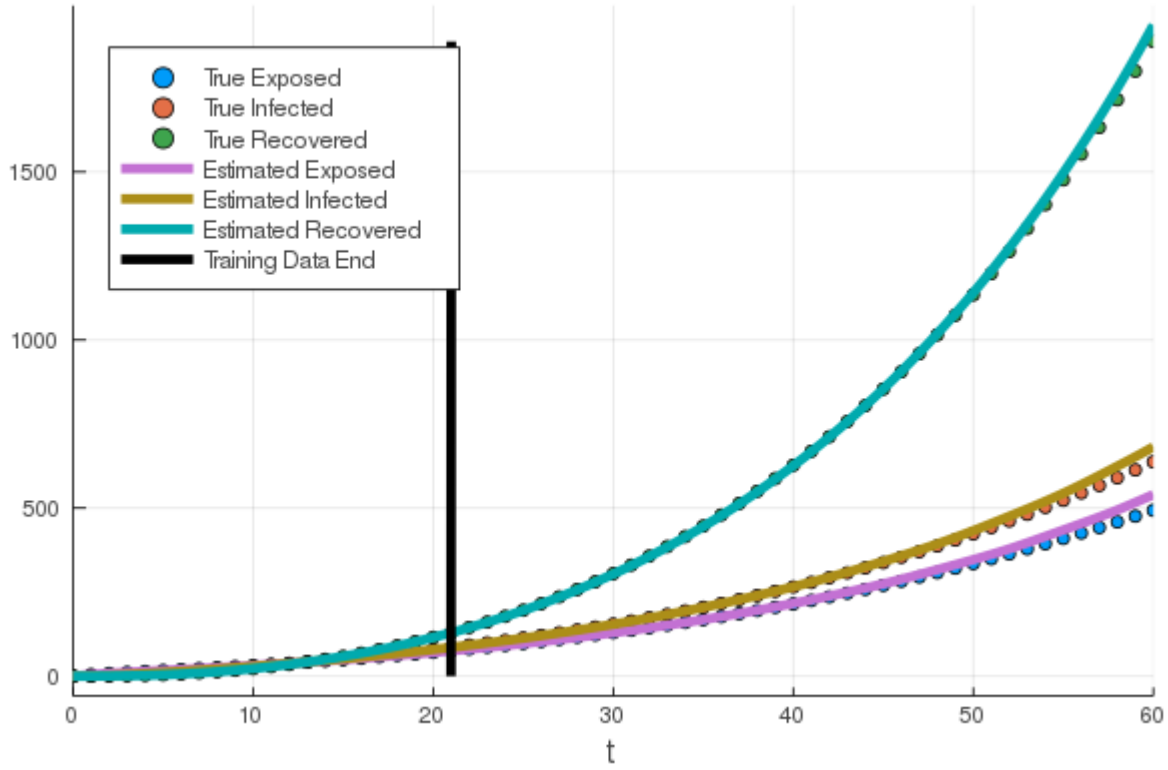
I * 0.10234428543435758 + S/N * I * 0.11371750552005416 + (S/N) ^ 2 * I * 0.1263545979985597

$$
\begin{aligned}
S' &= -\frac{\beta_0 SF}{N} - \mu S, \\
E' &= \frac{\beta_0 SF}{N} + \left(\sigma + \mu\right) E, \\
I' &= \sigma E - (\gamma + \mu) I, \\
R' &= \gamma I - \mu R, \\
N' &= -\mu N, \\
D' &= d\,\gamma I - \lambda D, \quad \text{and} \\
C' &= \sigma E,
\end{aligned}
$$

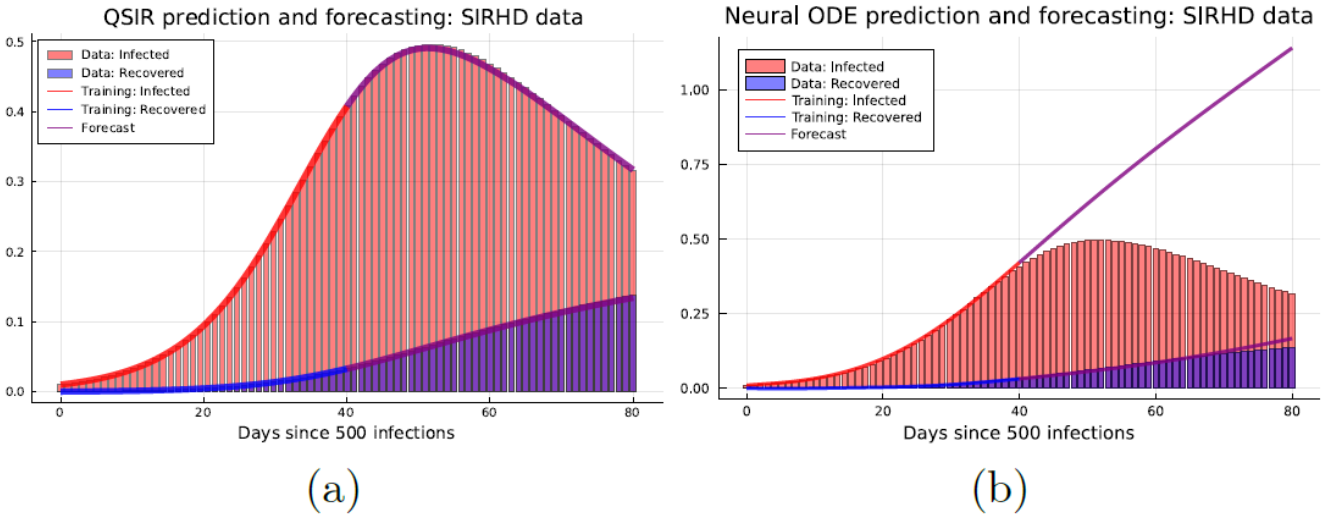
Replace Unknown Portion

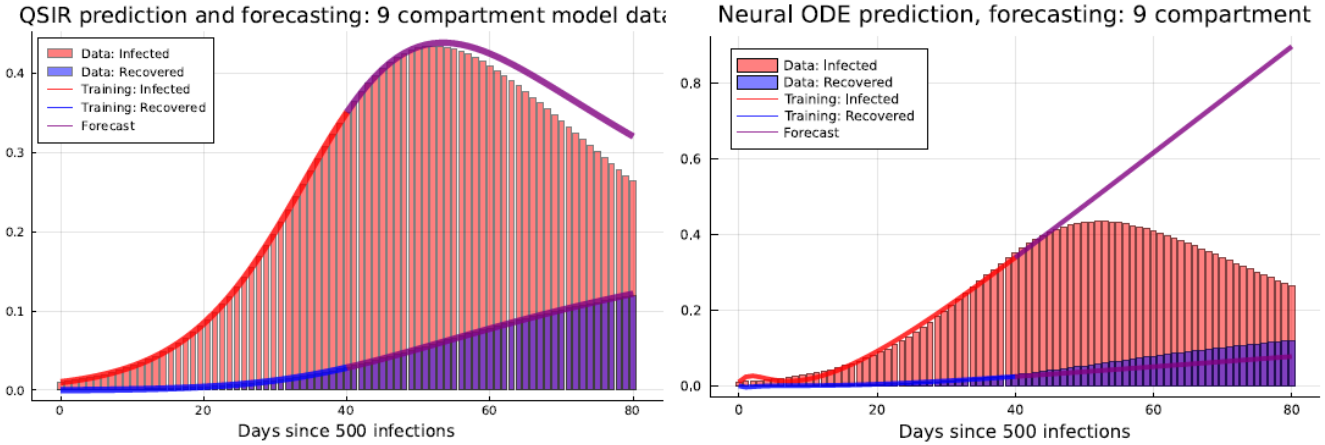
Replace Unknown Portion

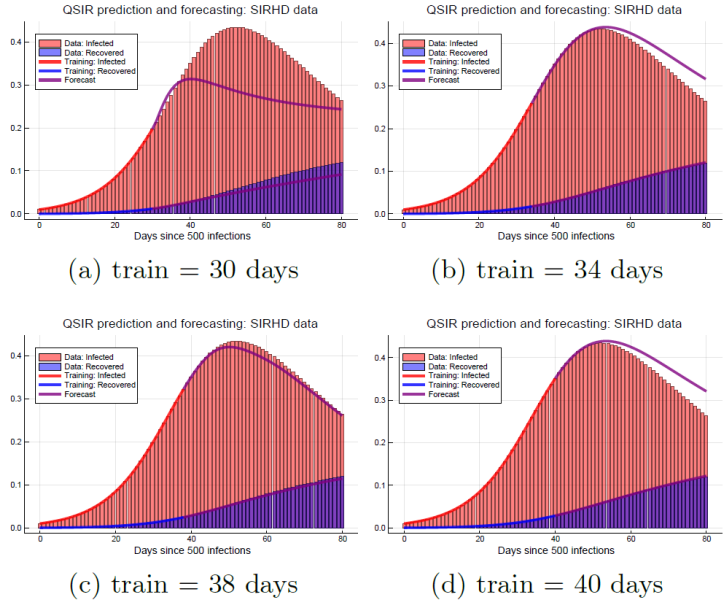
Sparsity improves generalizability!



Scientific Machine Learning: Improving Predictions with Less Data







Acquesta, E., Portone, T., Dandekar, R., Rackauckas, C., Bandy, R., & Huerta, J. (2022). Model-Form Epistemic Uncertainty Quantification for Modeling with Differential Equations: Application to Epidemiology (No. SAND2022-12823). Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

Dandekar, R., Rackauckas, C., & Barbastathis, G. (2020). A machine learning-aided global diagnostic and comparative tool to assess effect of quarantine control in COVID-19 spread. Patterns, 1(9), 100145.

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:

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

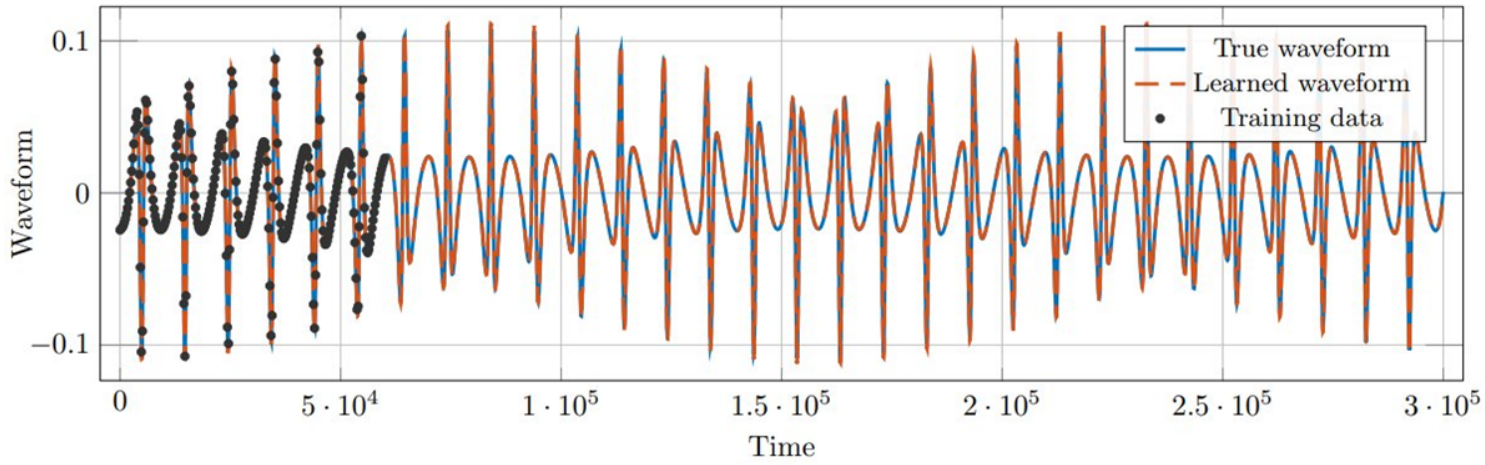
$$\dot{\chi} = \frac{(1 + e\cos(\chi))^2}{Mp^{3/2}}\big(1 + \mathcal{F}_2(\cos(\chi), p, e)\big), \tag{5b}$$

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

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

Automated discovery of geodesic equations from LIGO black hole data: run the code yourself!

https://github.com/Astroinformatics/ScientificMachineLearning/blob/main/neuralode_gw.ipynb



Keith, B., Khadse, A., & Field, S. E. (2021). Learning orbital dynamics of binary black hole systems from gravitational wave measurements. Physical Review Research, 3(4), 043101.

SciML Shows how to build Earthquake-Safe Buildings

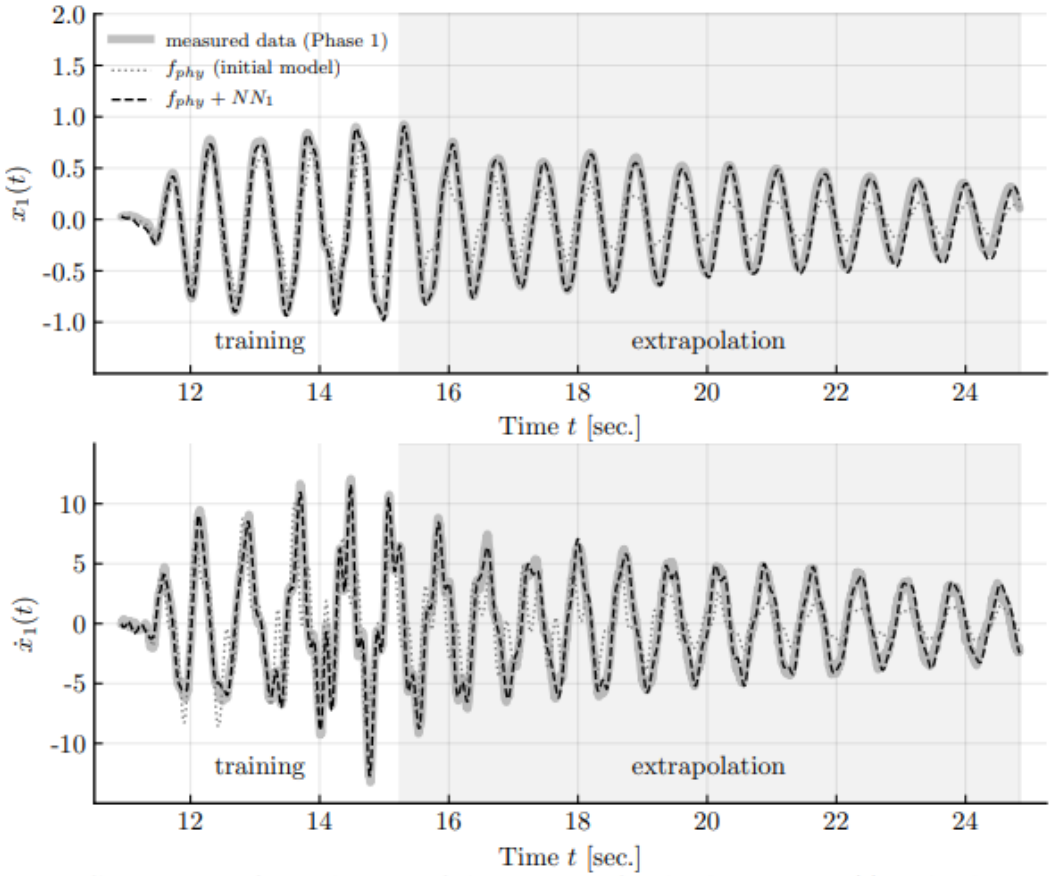


Figure 12: Comparison of time history of the response for displacement $x_1(t)$ and velocity $\dot{x}_1(t)$ for the NSD experiment (Phase 1).

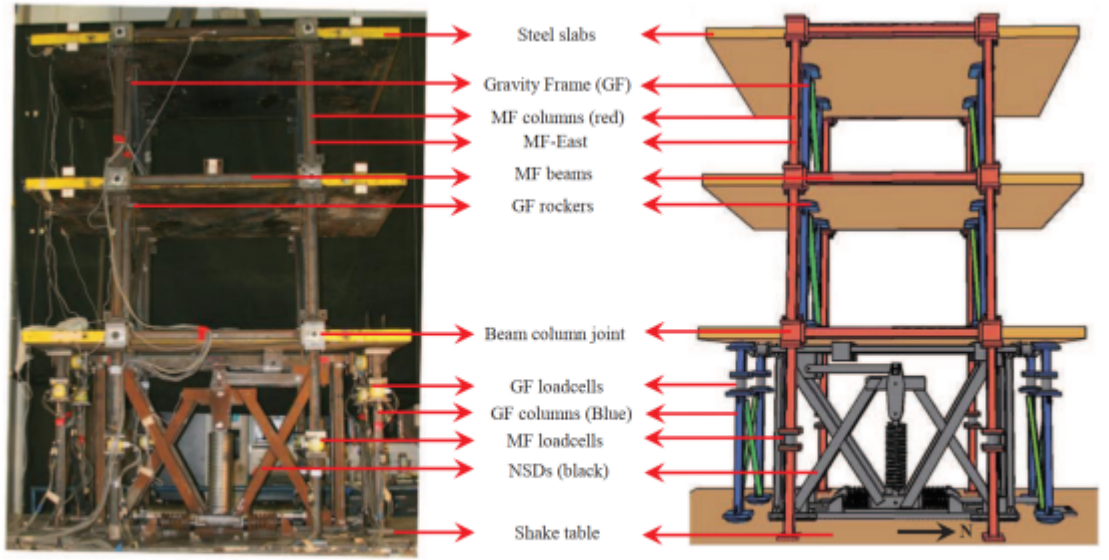


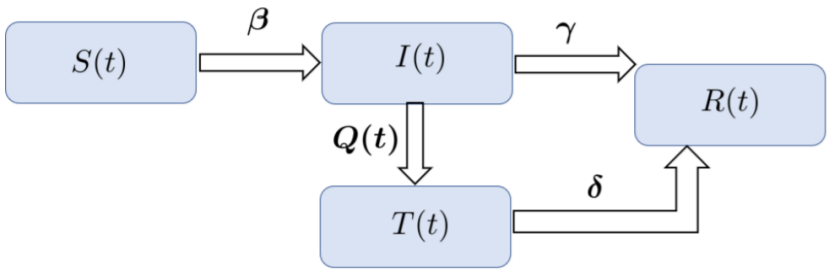
Figure 10: The structural system equipped with a negative stiffness device in between the first floor and the shake table.

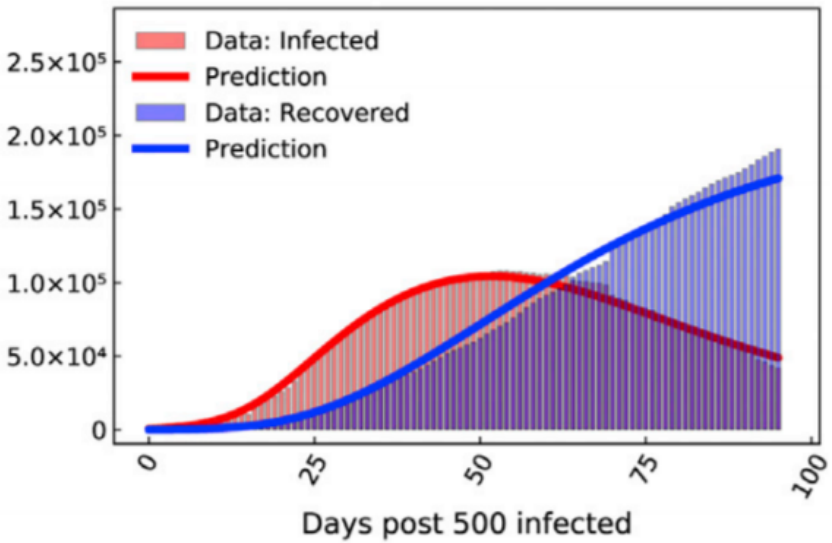
Structural identification with physics-informed neural ordinary differential equations
Lai, Zhilu, Mylonas, Charilaos, Nagarajaiah, Satish, Chatzi, Eleni

For a detailed walkthrough of UDEs and applications watch on Youtube:

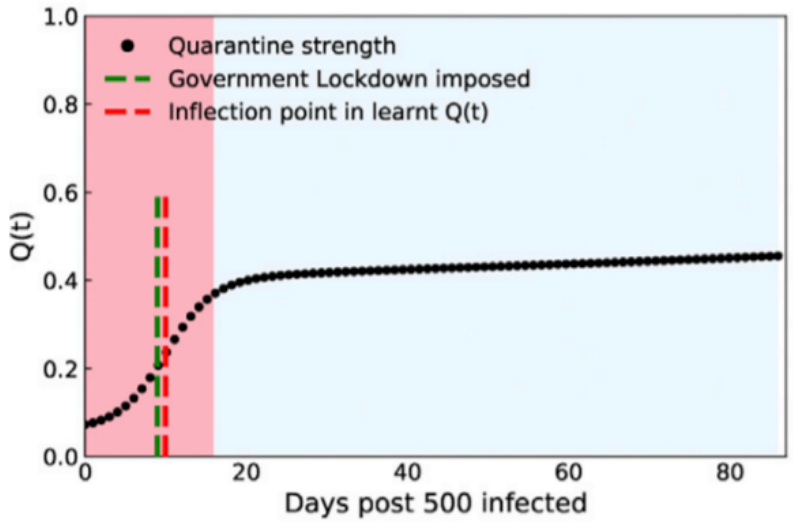
Chris Rackauckas: Accurate and Efficient Physics-Informed Learning Through Differentiable Simulation

QSIR Predicts Quarantine Measure Evolution

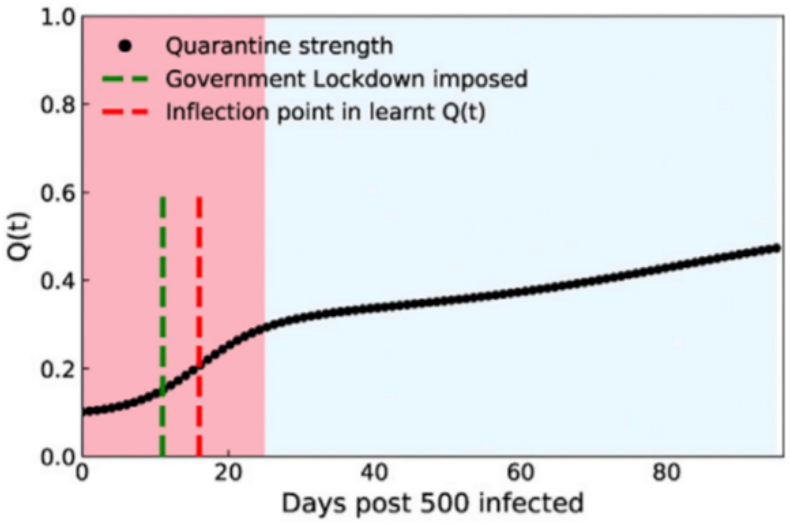




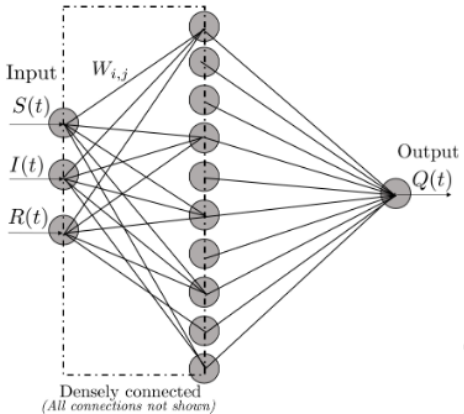
Italy



Spain



Italy



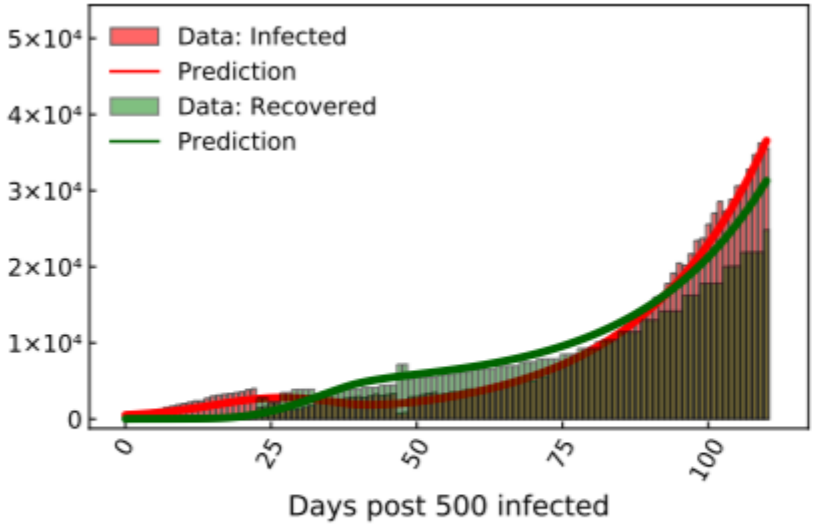
The QSIR Learns A Simplified SIR With Quarantine, and Quarantine Predictions are Within Days of Reported Changes

QSIR Counterfactuals: How Many Unnecessary Deaths in the Southern US?

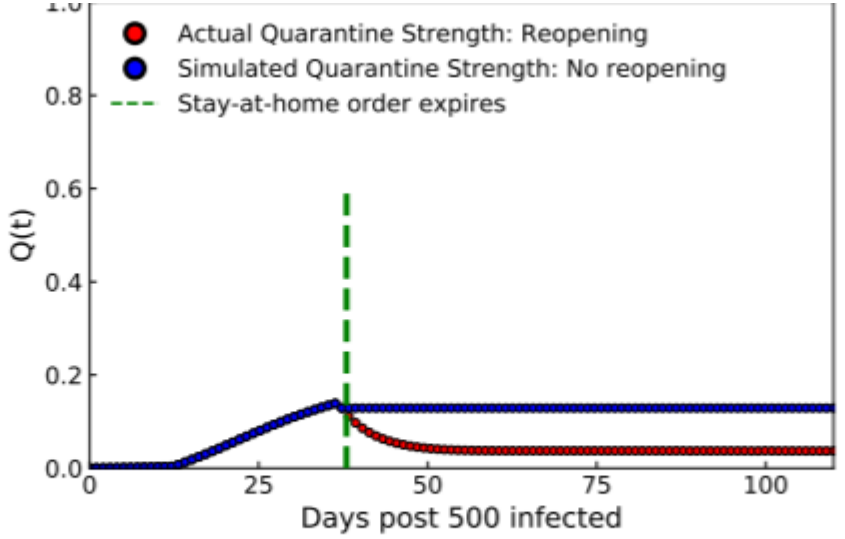
Verified QSIR now let's us ask questions: what if Q(t) didn't change?

Table 2. Infected count reduction by 14 July, 2020, if states had not reopened early, as estimated by our model.

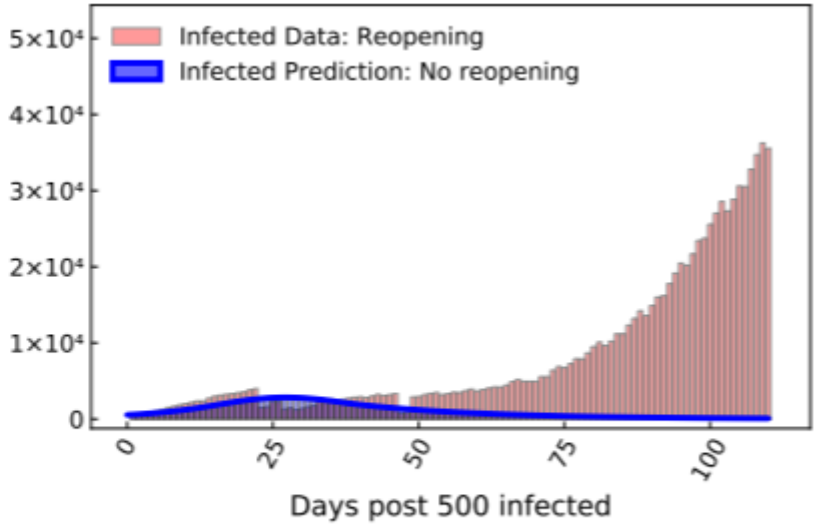
State	% decrease range (5% - 95% quantiles)	Mean % decrease	Case reduction range	Mean case reduction
1. Arizona	35 – 62	49	44000 – 79000	63000
2. Florida	20 – 75	49	57000 – 218000	144000
3. Louisiana	37 – 50	44	31000 – 41000	36000
4. Nevada	32 – 68	51	10000 – 20000	15000
5. Oklahoma	46 – 69	58	10000 – 15000	13000
6. South Carolina	83 – 86	84	50000 – 52000	51000
7. Tennessee	41 – 53	47	27000 – 36000	31000
8. Texas	41 – 51	46	115000 – 143000	129000
9. Utah	35 – 47	41	11000 – 14000	12000



(g) South Carolina

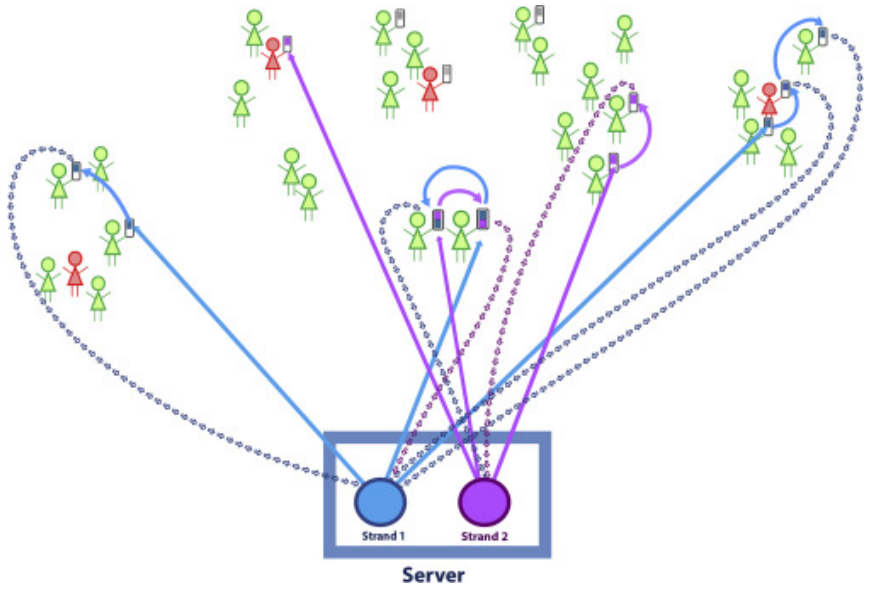


(h)



(i)

SciML for Real-Time Prediction: Powering SafeBlues



Patterns

PERSPECTIVE | VOLUME 2, ISSUE 3, 100220, MARCH 12, 2021

Safe Blues: The case for virtual safe virus spread in the long-term fight against epidemics

Raj Dandekar • Shane G. Henderson • Hermanus M. Jansen • ... Christopher Rackauckas • Peter G. Taylor • Aapeli Vuorinen • Show all authors

GIZMODO | We come from the future

PRIVACY AND SECURITY

Researchers Are Working on a Virtual Phone Virus That 'Mimics' the Spread of Covid-19

Lucas Ropek
Yesterday 5:25PM

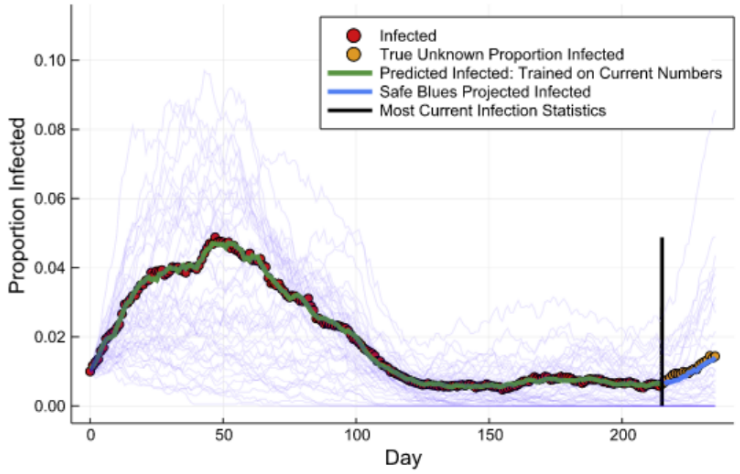
Photo: Sean Gallup (Getty Images)

Joshua Ne
HP's Josephine Ta
contributed to their
Spectre x360 14. C

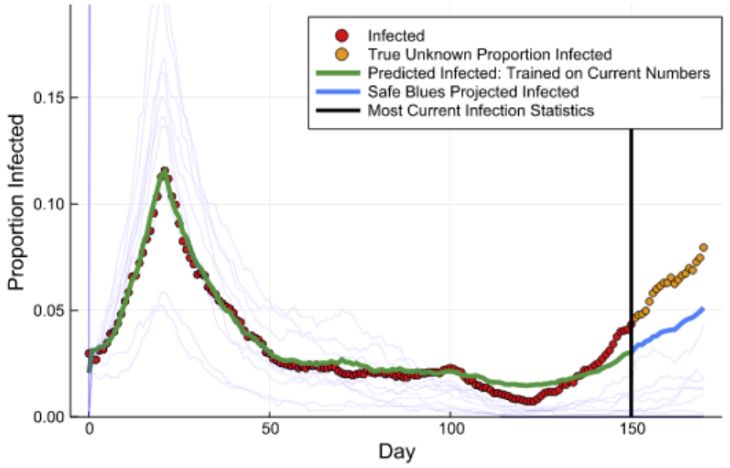
18

Recent Video

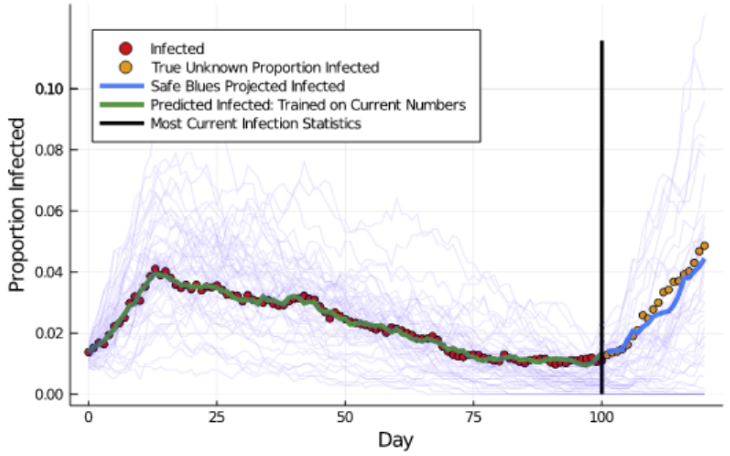
SciML for Real-Time Prediction: Powering SafeBlues



Model I.



Model II.



Model III.

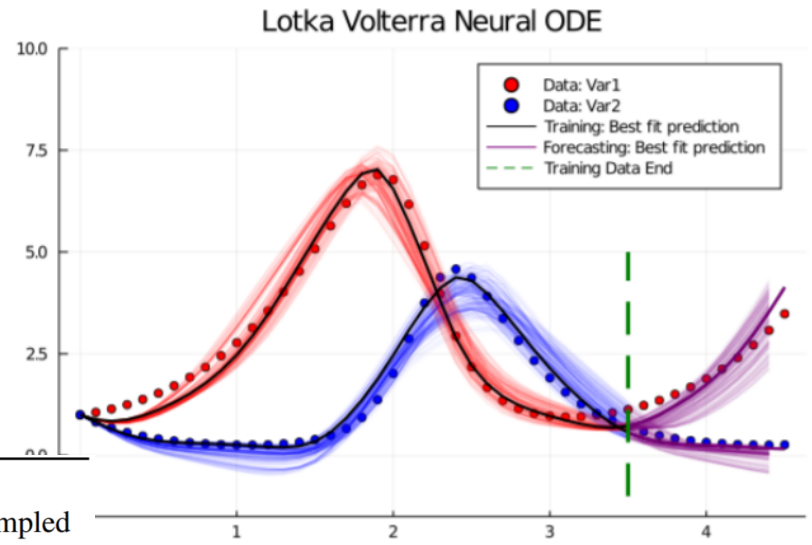
Idea: use neural networks to learn a mapping between fake viral strands to the infection statistics.

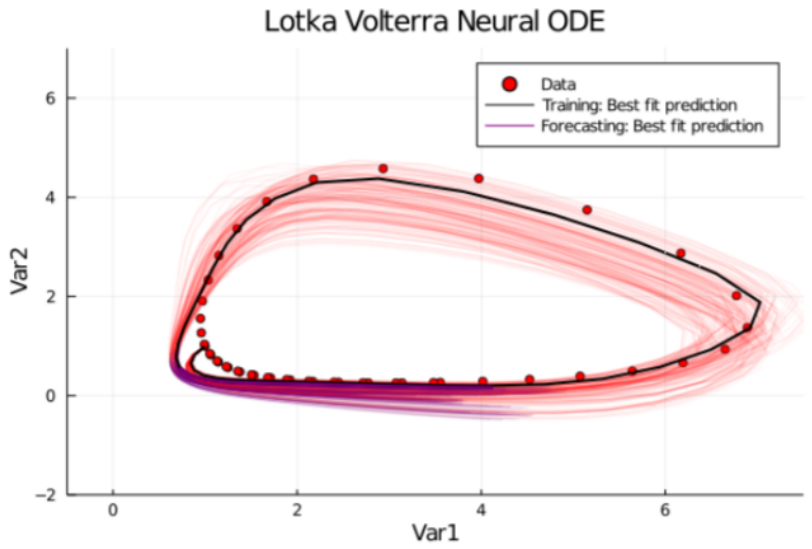
Current state: published proof of concept showing it works on many models, **on-campus experiment in process**

Bayesian UODEs: Knowledge-Enhanced Model Discovery with UQ

Result: Probability of Missing Mechanisms

λ	Number of Active terms	Dominant terms	Error	Mean AIC score	% sampled
0.01	9	u_1^2, u_2^2, u_1u_2 $u_1^2u_2^2, u_1^2u_2, u_2^2u_1$ u_1u_2, const	0.765	40.4	100
0.1	9	u_1^2, u_2^2, u_1u_2 $u_1^2u_2^2, u_1^2u_2, u_2^2u_1$ u_1u_2, const	0.764	35	100
1	5	u_1^2, u_2^2, u_2 $u_1^2u_2, u_1u_2$	0.764	21.6	100
2	2	$u_1^2u_2, u_1u_2$	0.634	7.2	100
3	**1**	u_1u_2	**0.7**	**4.1**	**100**
5	1	$u_1^2u_2$	2.49	-1	100





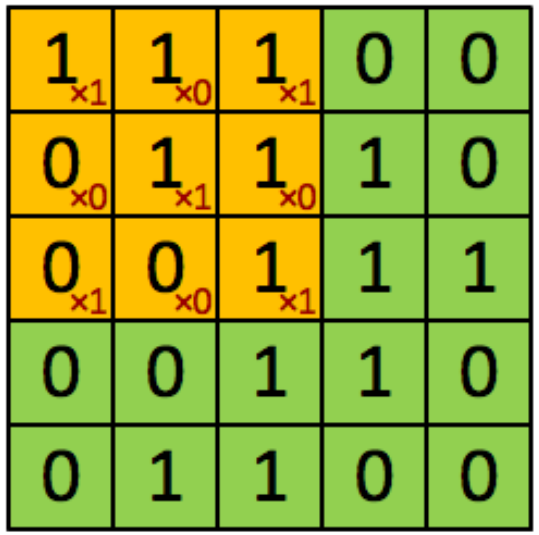

```
function lotka_volterra!(du, u, p, t)
  x, y = u
  α, β, δ, γ = p
  du[1] = dx = α*x -
  du[2] = dy = -δ*y
end
```

Dandekar, R., Dixit, V., Tarek, M., Garcia-Valadez, A., & Rackauckas, C. (2020). Bayesian Neural Ordinary Differential Equations. Languages for Inference (LAFI) 2021 - POPL 2021

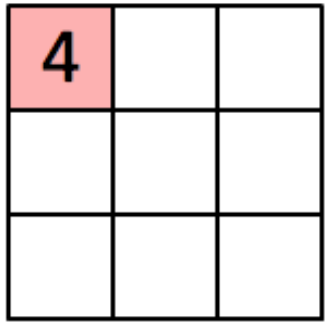
The UDE formulation fairly generally allows for imposing prior known structure

Discretized PDE Operators are Convolutions

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



Image



Convolved Feature

Is equivalent to the stencil

$$\begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}$$

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

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

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

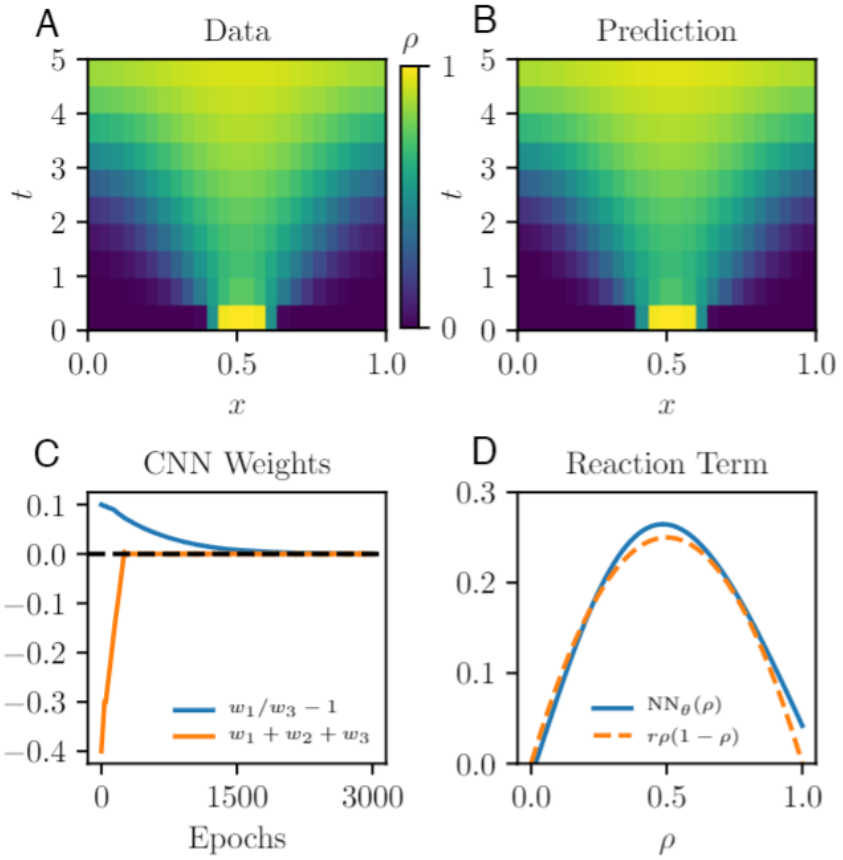
Truth: Fisher-KPP Equations

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

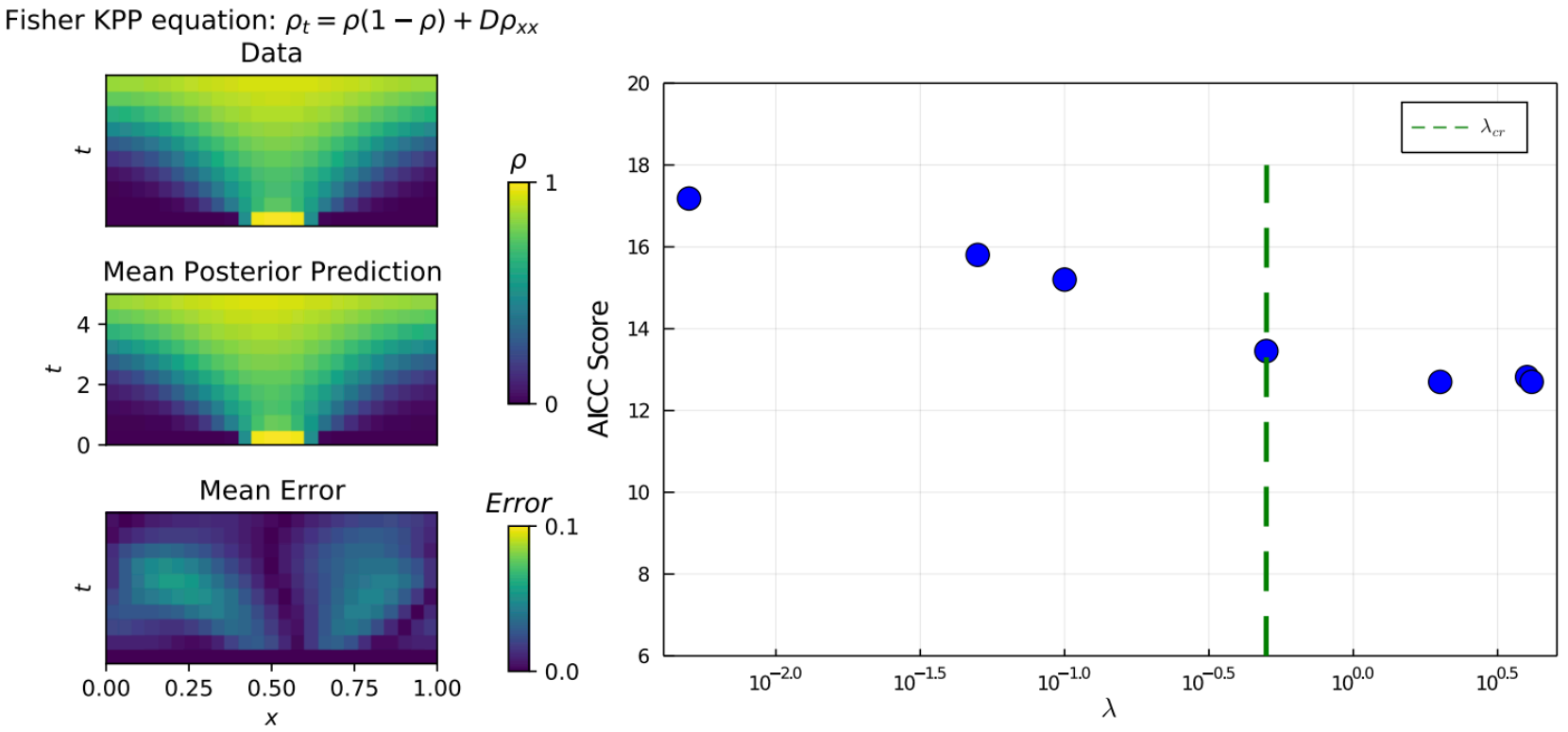
Truth: Universal Differential Equation

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

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



Bayesian Universal Differential Equations for PDEs



λ_{cr}	Number of Active terms	Dominant terms	% of samples
0.5	2	ρ, ρ^2	73
0.5	3	ρ, ρ^2, ρ^3	27

UDEs Effectively Recover Nonlinearities of Epidemic Models

The baseline case:

$$\frac{\mathrm{d}S(t)}{\mathrm{d}t} = -\frac{\tau_{SI}\, S(t)\, I(t)}{N}$$
$$\frac{\mathrm{d}I(t)}{\mathrm{d}t} = \frac{\tau_{SI}\, S(t)\, I(t)}{N} - \tau_{IR} I(t) - \tau_{ID} I(t)$$
$$\frac{\mathrm{d}R(t)}{\mathrm{d}t} = \tau_{IR} I(t)$$
$$\frac{\mathrm{d}D(t)}{\mathrm{d}t} = \tau_{ID} I(t).$$

Replacement of all terms with neural networks:

$$\frac{\mathrm{d}S(t)}{\mathrm{d}t} = -NN_{SI}$$
$$\frac{\mathrm{d}I(t)}{\mathrm{d}t} = NN_{SI} - NN_{IR} - NN_{ID}$$
$$\frac{\mathrm{d}R(t)}{\mathrm{d}t} = NN_{IR}$$
$$\frac{\mathrm{d}D(t)}{\mathrm{d}t} = NN_{ID}$$

Use SciML knowledge to constrain the interaction graph, but learn the nonlinearities!

	Actual Equations	SINDY Active terms	SINDY Equations	Minimum AICC
NN_{SI}	0.85 S I	1: SI	0.74 S I	14
NN_{IR}	0.1 I	1: I	0.097 I	19
NN_{ID}	0.05 I	1: I	0.049 I	21

Table 4: SIRD: SINDY Recovered terms

Universal Differential Equations Predict Chemical Processes

$$\frac{\partial c}{\partial t^*} = -\frac{1-\varepsilon}{\varepsilon}\mathrm{ANN}(q, q^*, \theta) - \frac{\partial c}{\partial x^*} + \frac{1}{Pe}\frac{\partial c^2}{\partial x^{*2}},$$

$$\frac{\partial q}{\partial t^*} = \mathrm{ANN}(q, q^*, \theta),$$

$$\frac{\partial c(x^* = 1, \forall t)}{\partial x^*} = 0,$$

$$\frac{\partial c(x^* = 0, \forall t)}{\partial x^*} = Pe(c - c_{inlet}),$$

$$c(x^* \in (0,1), t^* = 0) = c_0,$$

$$q(x^* \in (0,1), t^* = 0) = q^*(c_0),$$

$$q^* = f(c, p),$$

Santana, V. V., Costa, E., Rebello, C. M., Ribeiro, A. M., Rackauckas, C., & Nogueira, I. B. (2023). Efficient hybrid modeling and sorption model discovery for non-linear advection-diffusion-sorption systems: A systematic scientific machine learning approach. *arXiv preprint arXiv:2303.13555*.

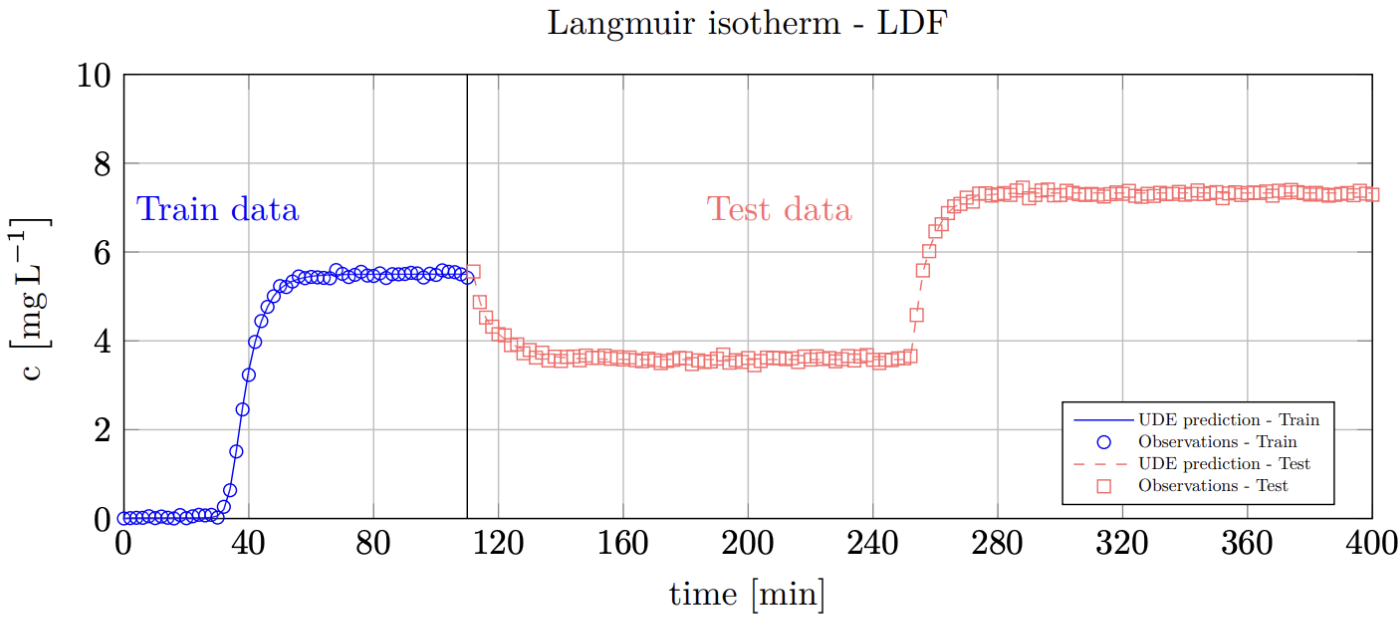


Figure 2: Schematic representation of the proposed hybrid model.

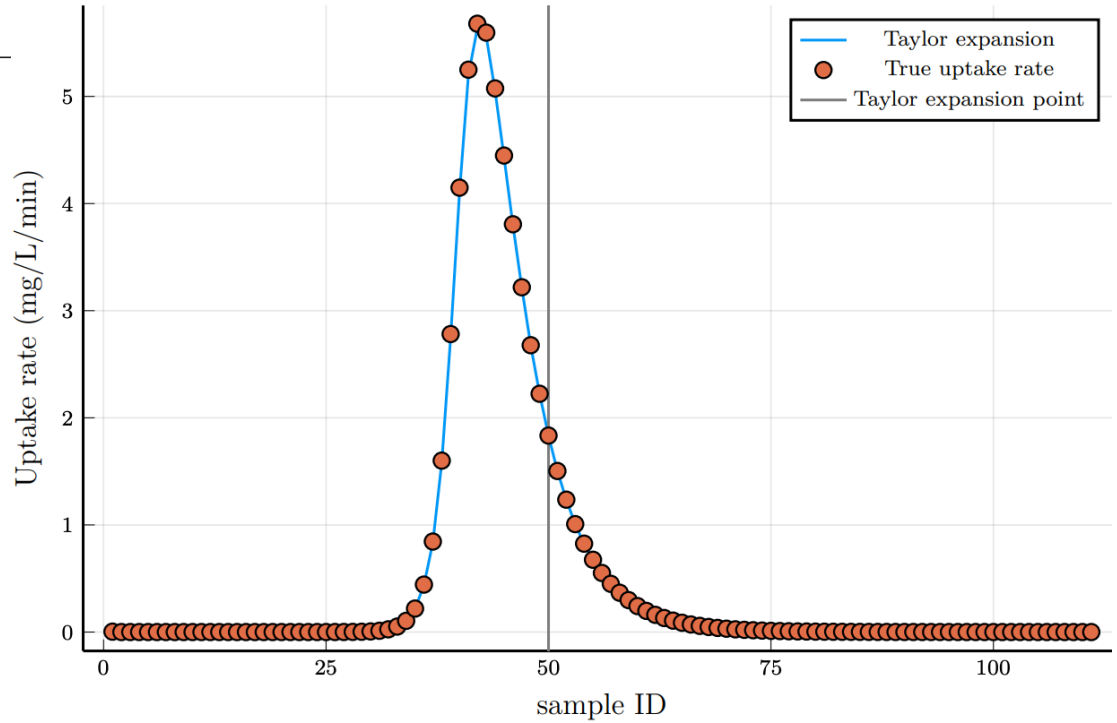
Universal Differential Equations Predict Chemical Processes

Table 5: Symbolic regression learned polynomials.

Isotherm	Kinetic	True kinetics	Learned kinetics
Langmuir	LDF	$0.22q^* - 0.22q$	$-0.535 - 0.225q + 0.234(q^*)$
Langmuir	improved LDF	$0.22(q^* + 0.2789q^* e^{\frac{-q}{2q^*}} - q)$	$-0.554 - 0.234q + 0.281(q^*)$
Langmuir	Vermeulen's	$0.22\frac{q^{*2}-q^2}{2.0q}$	$-0.6098 + 0.0122q + 0.263q^*$ $-0.00526qq^*$
Sips	LDF	$0.22q^* - 0.22q$	$0.198q^* - 0.200q$
Sips	improved LDF	$0.22(q^* + 0.2789q^* e^{\frac{-q}{2q^*}} - q)$	$0.277q^* - 0.241q$
Sips	Vermeulen's	$0.22\frac{q^{*2}-q^2}{2.0q}$	$-0.003557q^{*2} - 0.216q + 0.395q^*$

$$0.22(q^* + 0.2789q^* e^{\frac{-q}{2q^*}} - q)(49.23, 49.22) \approx 1.834 + 0.275q^* - 0.238q + \mathcal{O}(\|x^2\|)$$

Recovers equations with the same 2nd order Taylor expansion



Santana, V. V., Costa, E., Rebello, C. M., Ribeiro, A. M., Rackauckas, C., & Nogueira, I. B. (2023). Efficient hybrid modeling and sorption model discovery for non-linear advection-diffusion-sorption systems: A systematic scientific machine learning approach. *arXiv preprint arXiv:2303.13555*.

Universal Differential-Algebraic Equations: Encoding Physical Constraints

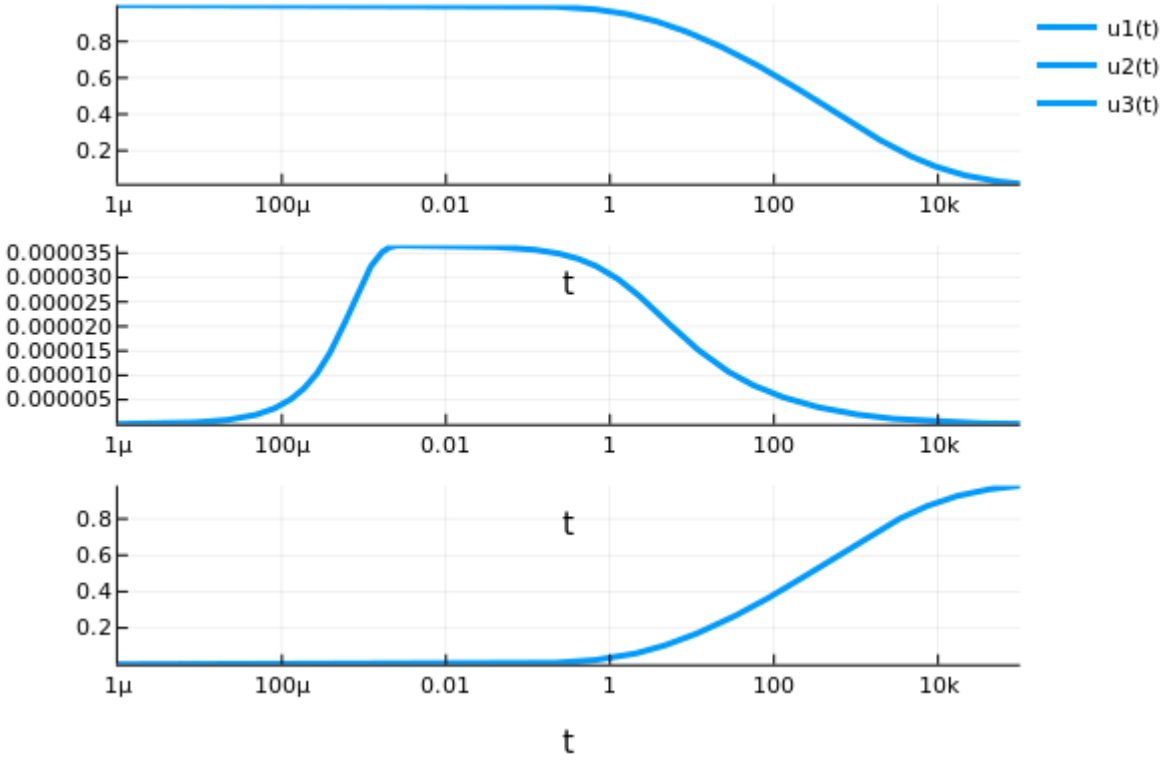
Utilize known chemical kinetics

$$y_1' = -0.04y_1 + \mathrm{NN}_1(y_1, y_2, y_3)$$
$$y_2' = 0.04y_1 + \mathrm{NN}_2(y_1, y_2, y_3)$$
$$1 = y_1 + y_2 + y_3$$

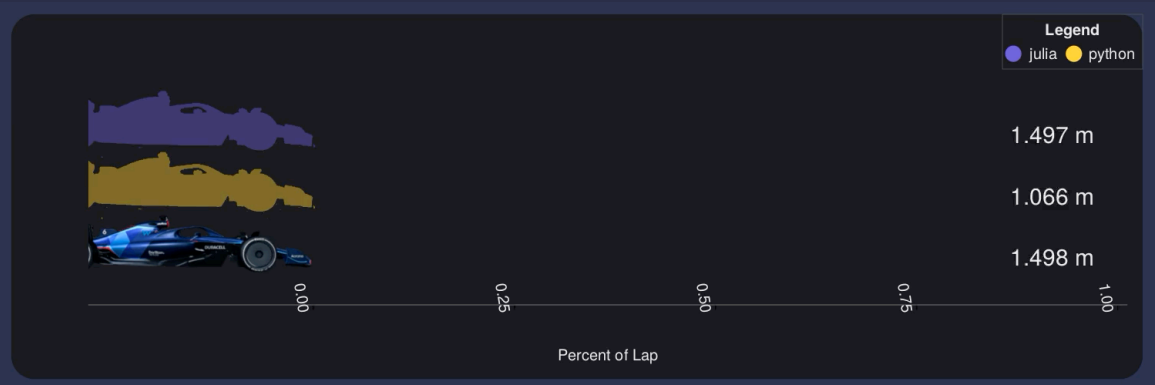
With known conservation laws

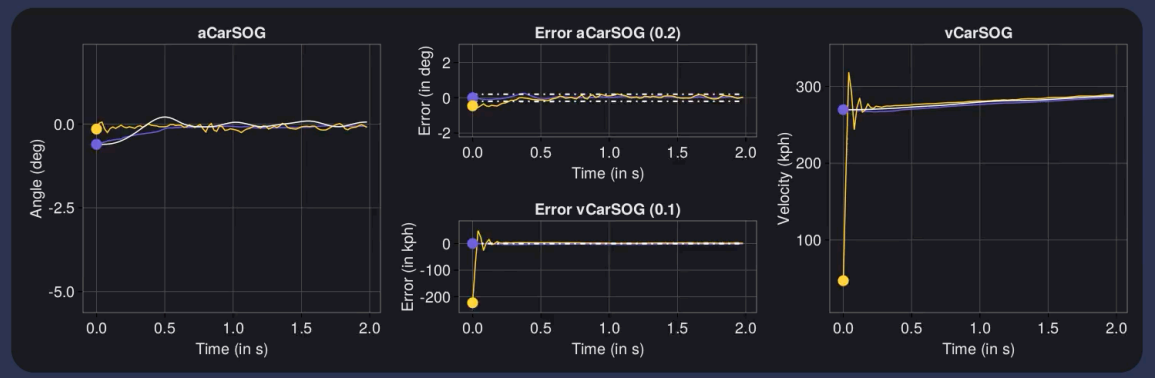
$$Mu' = f(u) + NN(u)$$

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



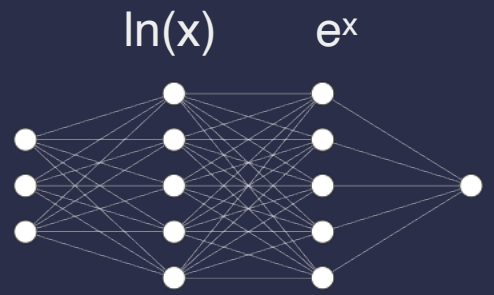
Learn highly stiff equations: **Hessian condition number 10^{13}**





Physically-Informed Machine Learning

$$\dot{\beta} \approx \frac{a_y}{u} - \frac{a_x}{u} - r$$

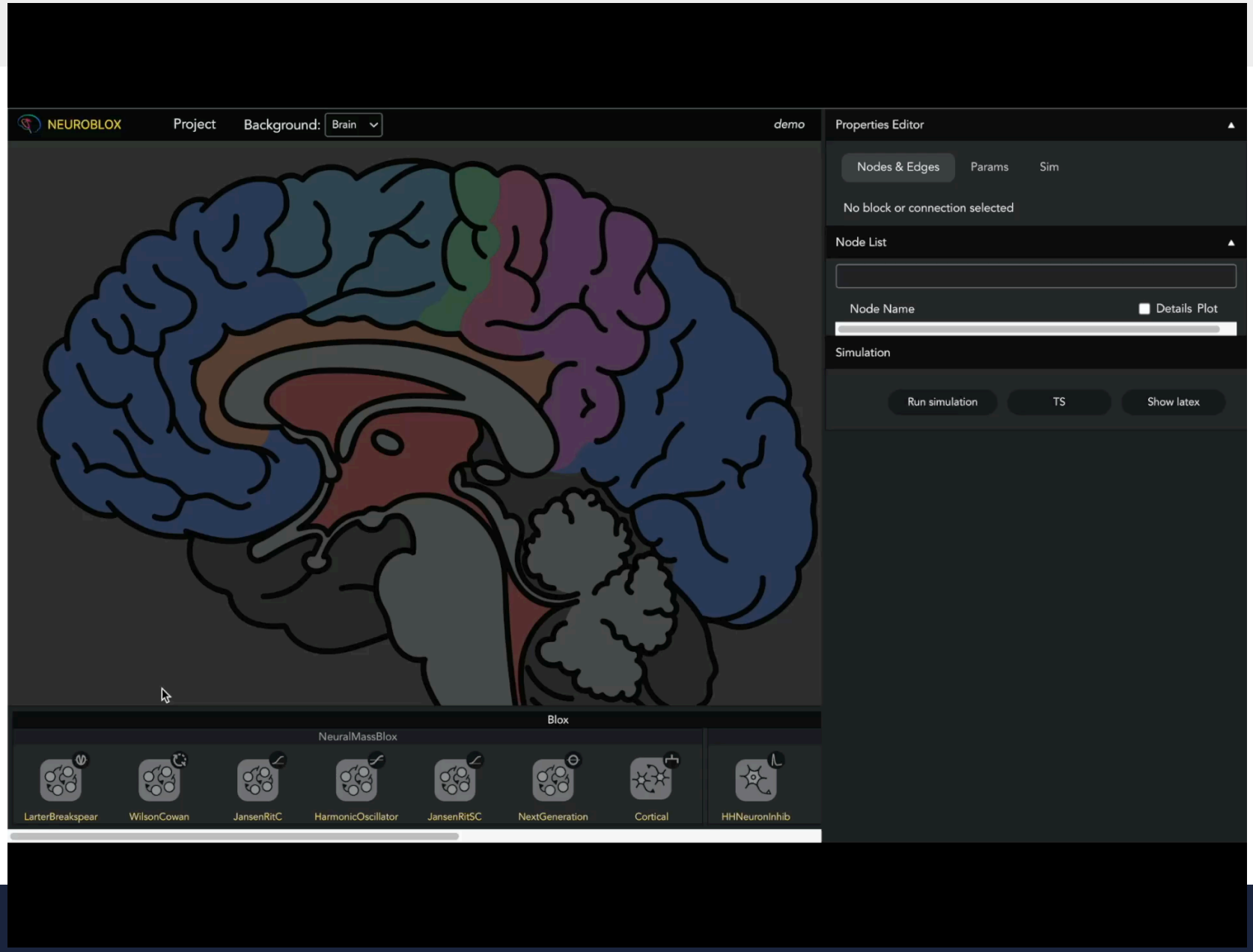


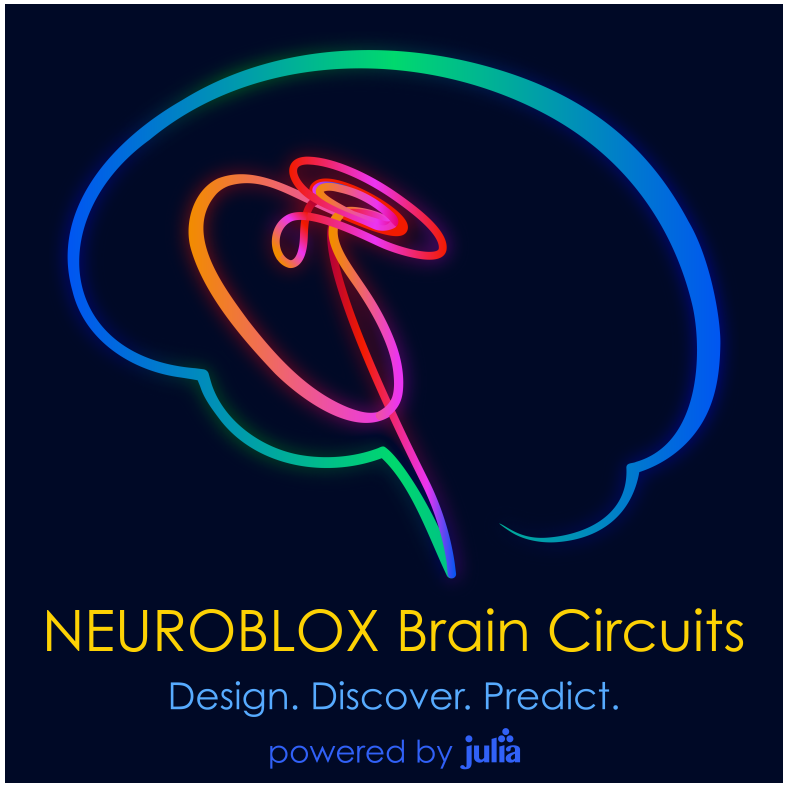
Using knowledge of the physical forms as part of the design of the neural networks.

Formulation: Koopman + SIREN + Physics

Smoother, more accurate results

Neuroblox: UDEs in Psychopharmacology





Bayesian Chemical Reaction Neural Network

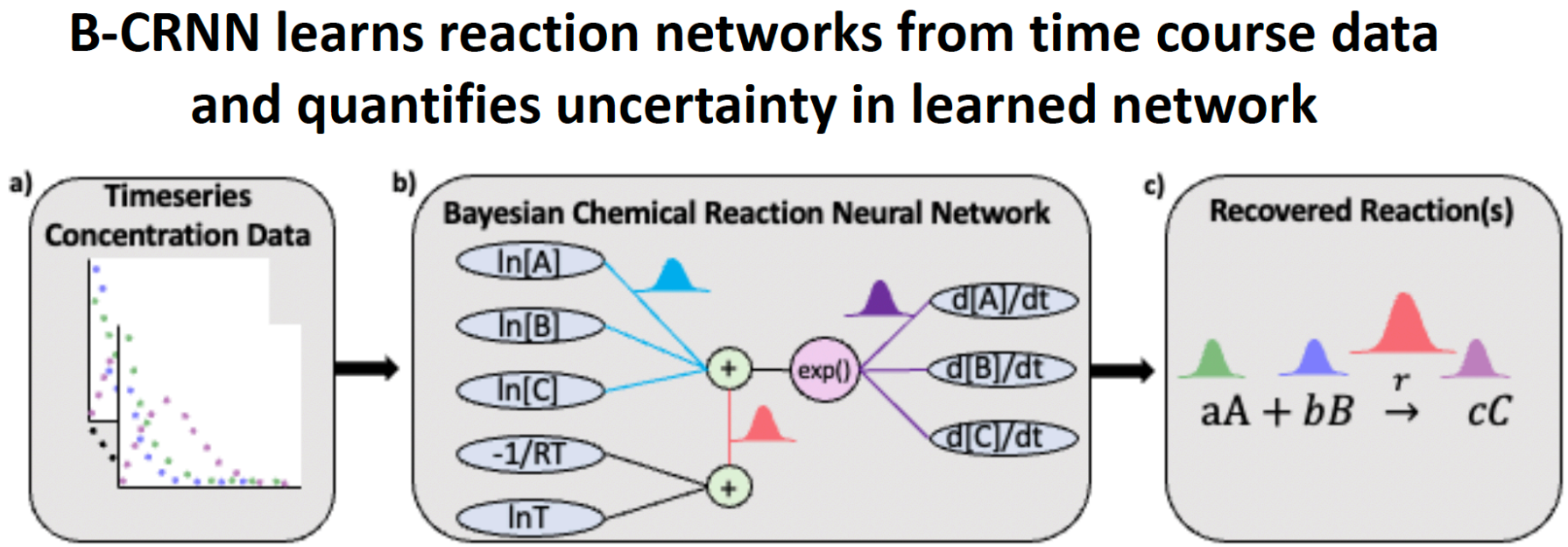


Figure 2. Overview of the B-CRNN which uses time course concentration data (a) to train a constrained neural network (b) that uses a preconditioned SGLD optimizer to reconstruct the reaction network and estimate the uncertainty in the learned stoichiometry and reaction rates (c).

Bayesian Chemical Reaction Neural Network

B-CRNN describes uncertainty in learned reaction rates

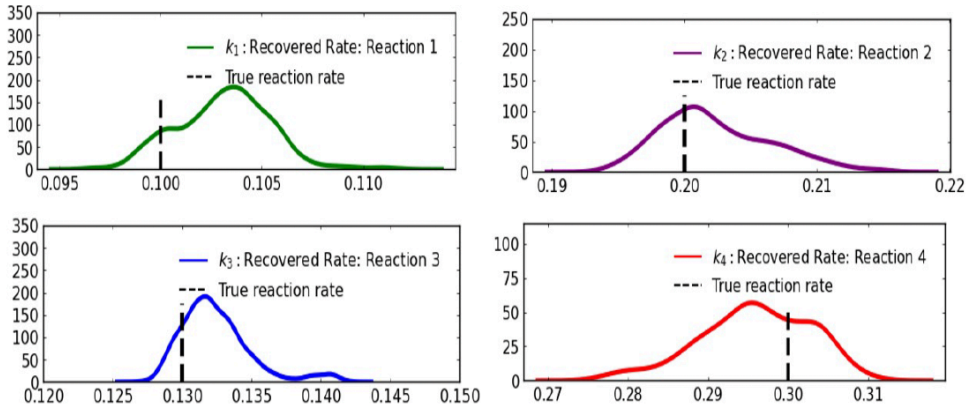
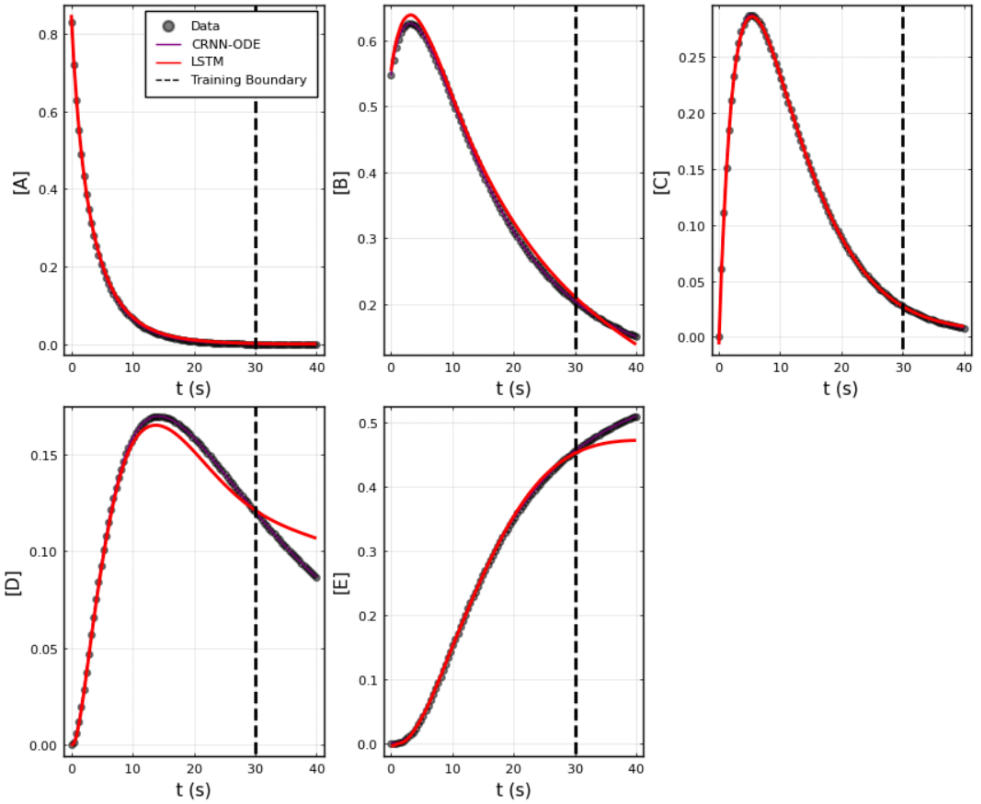


Figure 3. Posterior distribution of learned reaction rates for the four reactions included in table 1. Vertical dashed lines are true rates.

B-CRNN can extrapolate beyond training region, purely data-driven ML cannot

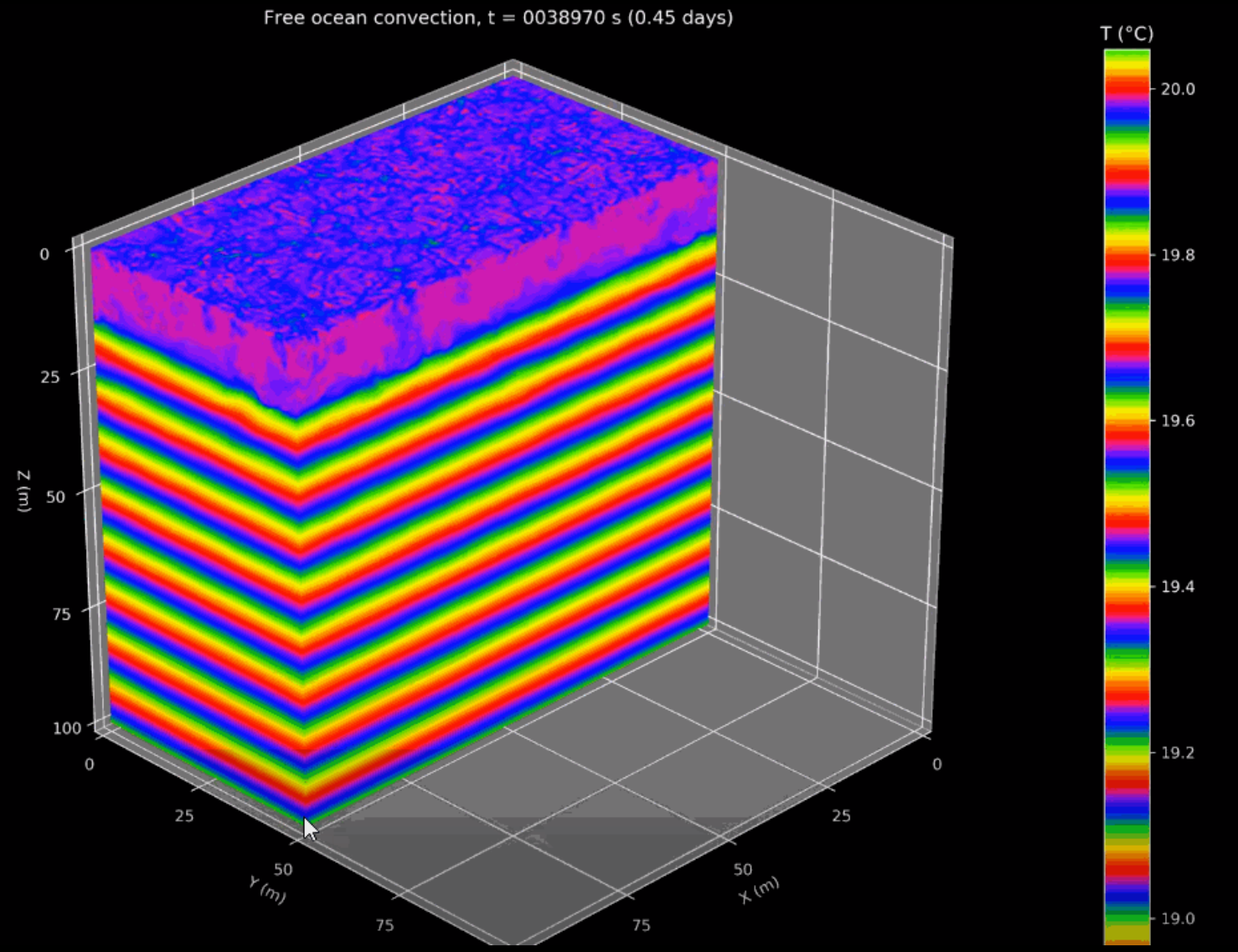


Does doing such methods require differentiation of the simulator?

High fidelity surrogates of ocean columns for climate models

3D simulations are high resolution but too expensive.

Can we learn faster models?



Ramadhan, Ali, John Marshall, Andre Souza, Gregory LeClaire Wagner, Manvitha Ponnapati, and Christopher Rackauckas. "Capturing missing physics in climate model parameterizations using neural differential equations." *arXiv preprint arXiv:2010.12559* (2020).

(Update going online in the next month)

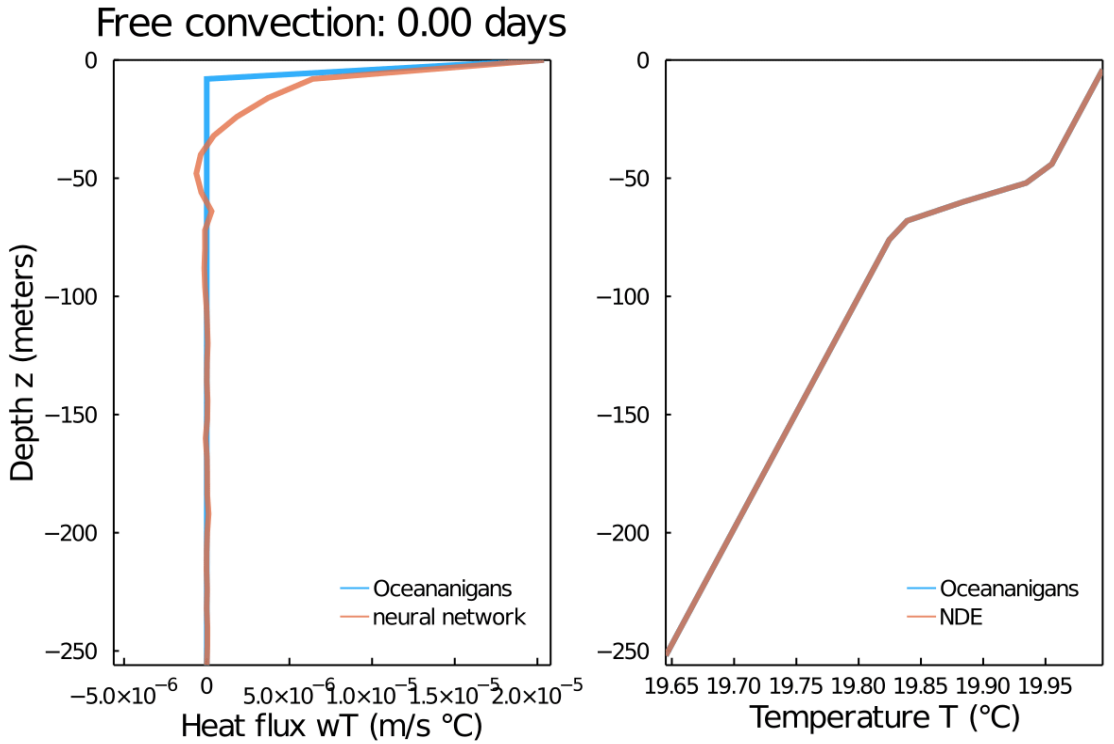
Neural Networks Infused into Known Partial Differential Equations

Derive a 1D approximation to the 3D model

$$\frac{\partial T}{\partial t} = -\frac{\partial}{\partial z}\underbrace{\left(\text{NN}(T) - K\frac{\partial T}{\partial z}\right)}_{\overline{w'T'}}$$

Incorporate the "convective adjustment"

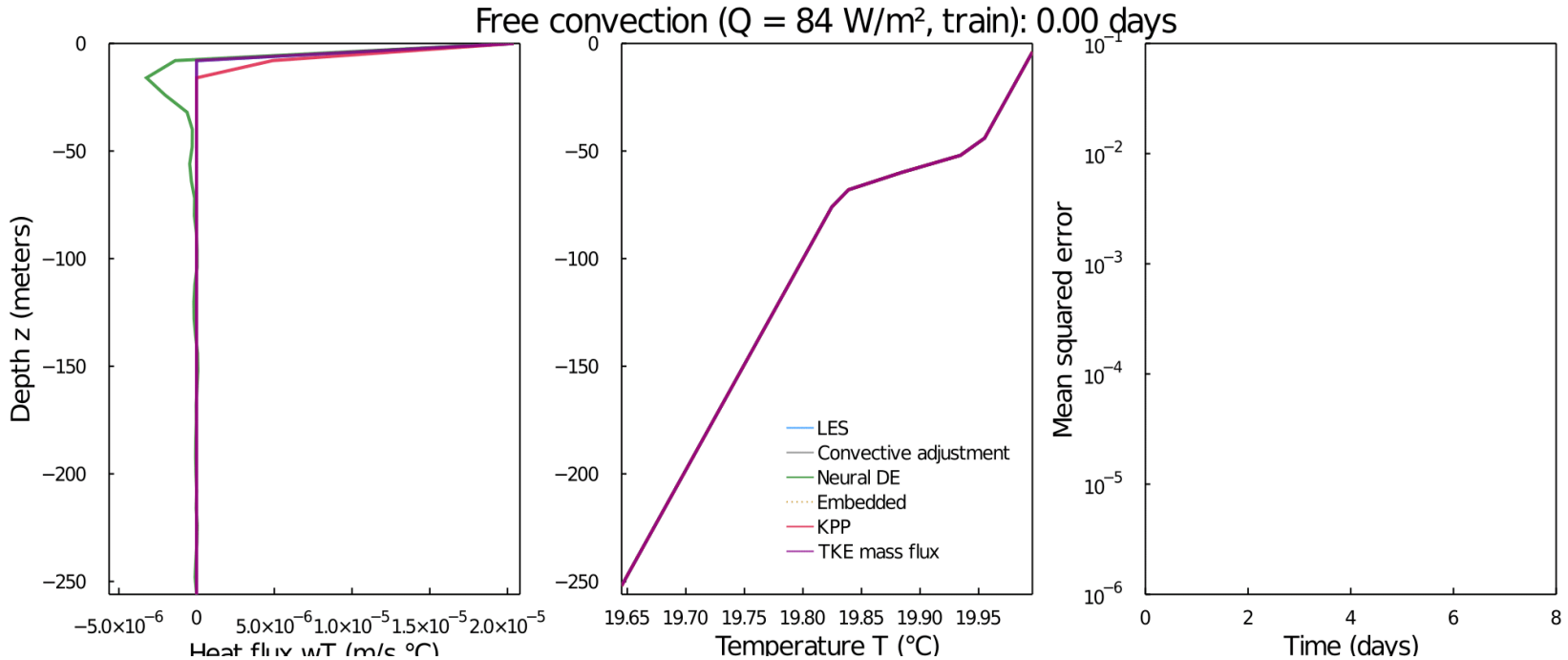
$$K = \begin{cases} 0 & \text{if } \partial_z T > 0 \\ 100\ \text{m}^2/\text{s} & \text{if } \partial_z T < 0 \end{cases}$$



$$loss(T, wT) = |NN(T) - wT|^2$$

Only okay, but why?

Good Engineering Principles: Integral Control!



$$\frac{\partial T}{\partial t} = -\frac{\partial}{\partial z}\underbrace{\left(\text{NN}(T) - K\frac{\partial T}{\partial z}\right)}_{\overline{w'T'}}$$

$$loss(T_{NN}, T) = |T_{NN}(z,t) - T(z,t)|^2$$

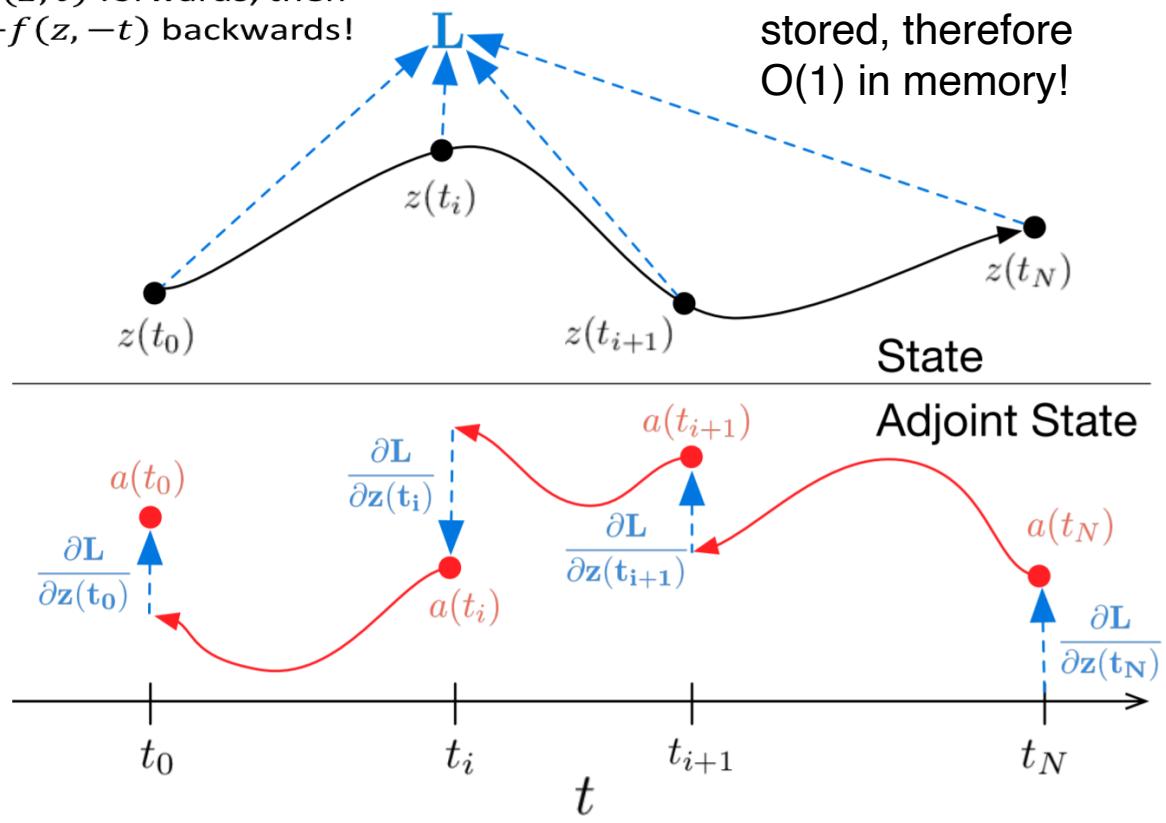
The adjoint methods (start skipping from here)

Method	Stability	Stiff Performance Scaling	Memory Usage
BacksolveAdjoint	Poor	O((n+p)^3)	Low. O(1)
InterpolatingAdjoint	Good	O((n+p)^3)	High. Requires full continuous solution of forward
QuadratureAdjoint	Good	O(n^3 + p)	Higher. Requires full continuous solution of forward and Lagrange multiplier
BacksolveAdjoint (Checkpointed)	Okay	O((n+p)^3)	Medium. O(c) where c is the number of checkpoints
InterpolatingAdjoint (Checkpointed)	Good	O((n+p)^3)	Medium. O(c) where c is the number of checkpoints
ReverseDiffAdjoint	Best	O(n^3 + p)	Highest. Requires full forward and reverse AD of solve
TrackerAdjoint	Best	O(n^3 + p)	Highest. Requires full forward and reverse AD of solve
ForwardLSS/AdjointLSS/ NILSS	Chaos	Not even comparable: expensive.	Super duper high OMG.

Machine Learning Neural Ordinary Differential Equations

$u' = f(z, t)$ forwards, then
$u' = -f(z, -t)$ backwards!

Timeseries is not stored, therefore O(1) in memory!



The adjoint equation is an ODE!

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}}$$

How do you get z(t)? One suggestion: Reverse the ODE

$$\frac{d\mathbf{a}_{aug}(t)}{dt} = -\begin{bmatrix}\mathbf{a}(t) & \mathbf{a}_\theta(t) & \mathbf{a}_t(t)\end{bmatrix} \frac{\partial f_{aug}}{\partial[\mathbf{z}, \theta, t]}(t)$$

Chen, Ricky TQ, et al. "Neural ordinary differential equations." Advances in neural information processing systems. 2018.

“Adjoints by reversing” also is unconditionally unstable on some problems!

Advection Equation:

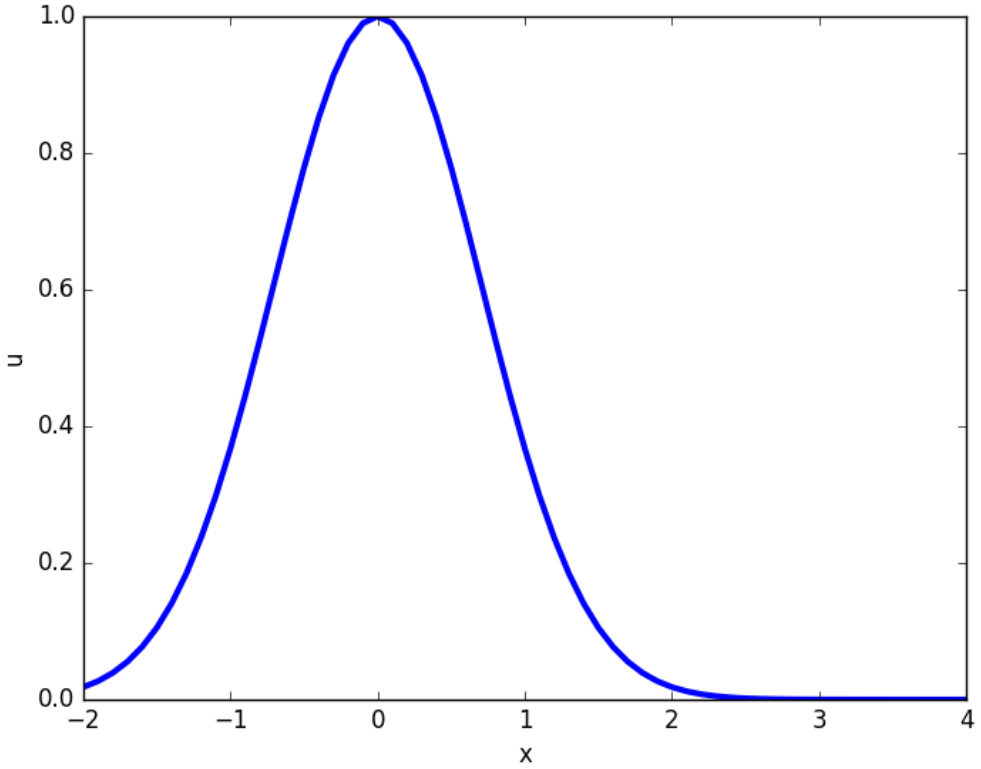
$$\frac{\partial u}{\partial t} + \frac{a(\partial u)}{\partial x} = 0$$

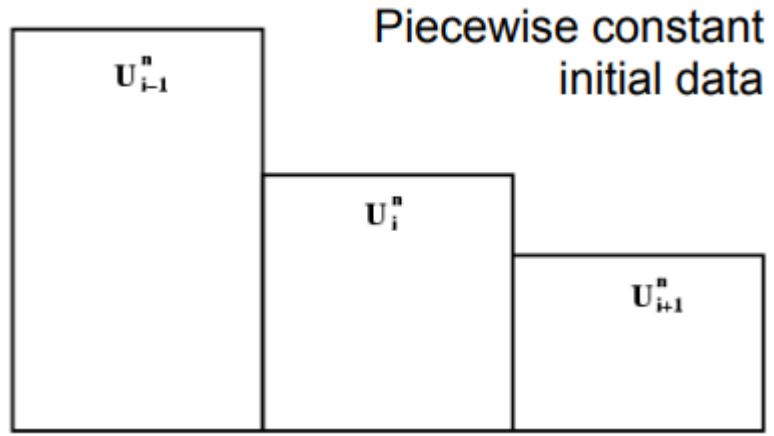
Approximating the derivative in ***X*** has two choices: forwards or backwards

$$u_i' = -\frac{a(u_i - u_{i-1})}{\Delta x} \text{ or } u_i' = -\frac{a(u_{i+1} - u_i)}{\Delta x}?$$

If you discretize in the wrong direction you get **unconditional instability**

You need to understand the engineering principles and the numerical simulation properties of domain to make ML stable on it.

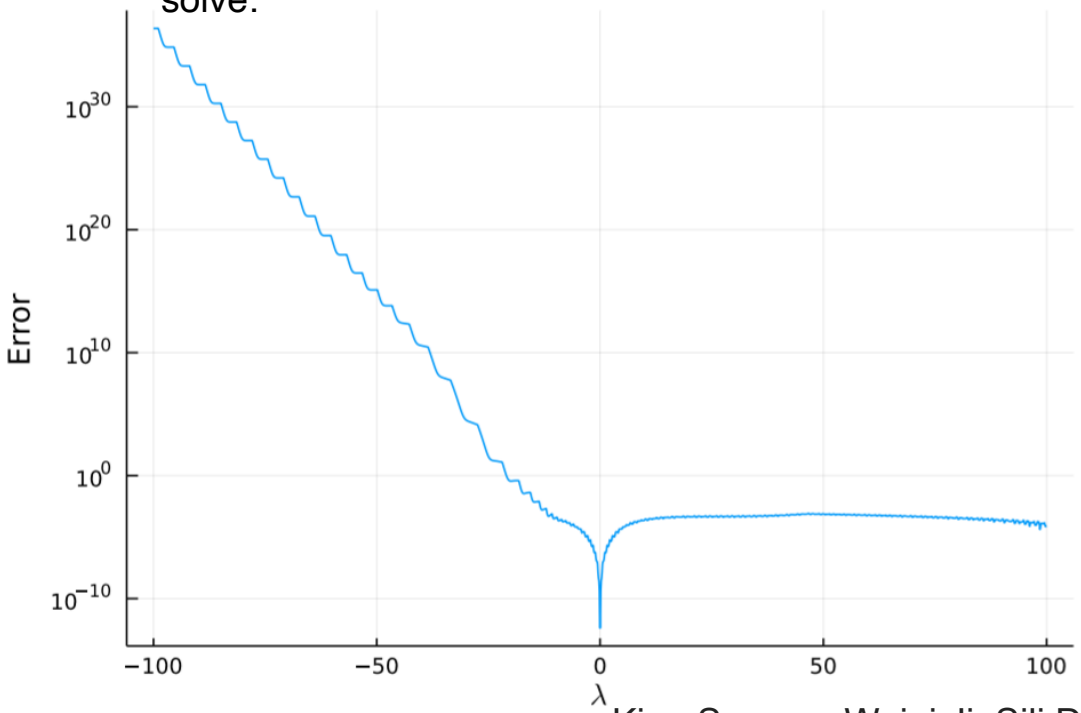




Problems With Naïve Adjoint Approaches On Stiff Equations

Error grows exponentially…

$u'(t) = \lambda u(t)$, plot the error in the reverse solve:

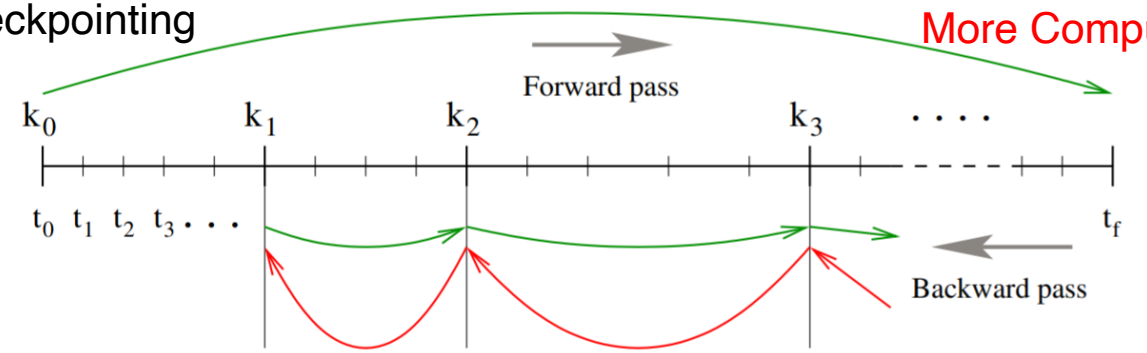


How do you get u(t) while solving backwards?
3 options!

1. $u' = f(z,t)$ forwards, then $u' = -f(z,-t)$ backwards! Unstable

2. Store u(t) while solving forwards (dense output) High memory

3. Checkpointing More Compute



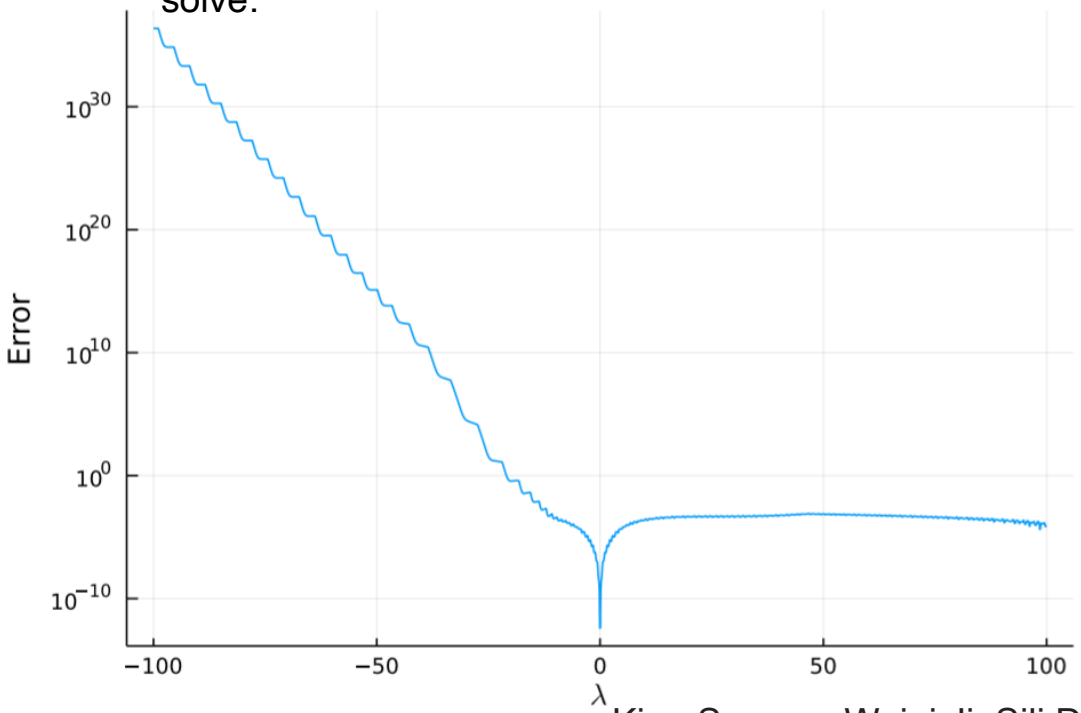
Each choices has an engineering trade-off!

Kim, Suyong, Weiqi Ji, Sili Deng, and Christopher Rackauckas. "Stiff neural ordinary differential equations." *Chaos* (2021).

Problems With Naïve Adjoint Approaches On Stiff Equations

Error grows exponentially…

$u'(t) = \lambda u(t)$, plot the error in the reverse solve:



Compute cost is cubic with parameter size when stiff

Size of reverse ODE system is:

$$2states + parameters$$

Linear solves inside of stiff ODE solvers, ~cubic

Thus, adjoint cost:

$$O\left(\left(states + parameters\right)^3\right)$$

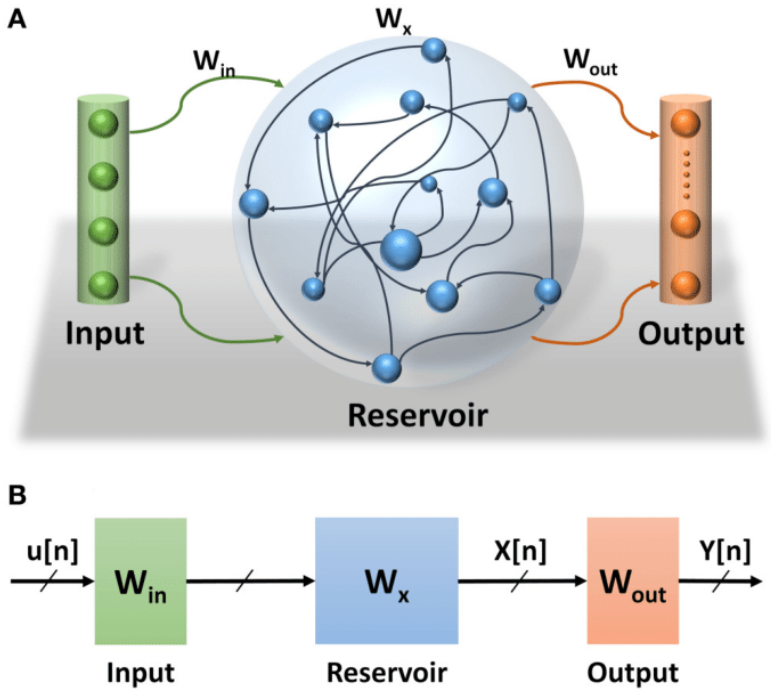
Kim, Suyong, Weiqi Ji, Sili Deng, and Christopher Rackauckas. "Stiff neural ordinary differential equations." *Chaos* (2021).

Though note that if you just want to speed things up, there might be easier things you can do

Continuous-Time Echo State Networks: Avoid Gradients and Use an Implicit Fit

One way to visualize: reservoir computing

Fix a random dynamical process and find a projection to fit the system



Accelerating Simulation of Stiff Nonlinear Systems using Continuous-Time Echo State Networks

Ranjan Anantharaman, Yingbo Ma, Shashi Gowda, Chris Laughman, Viral Shah, Alan Edelman, Chris Rackauckas

Another interpretation: Semi-Neural ODE

Fix the parameters of the first layer and only train the last layer. By doing so, you can transform the training problem into a linear solve via SVD.

Fix $r' = \sigma(Ar + W_x x)$
Predict $x(t) = W_{out} r(t)$

Turns into a linear solve
Solve the linear system via SVD
(to manage the growth factor)

Get W_{out} at many parameters of the system

Predict behavior at new parameters via:

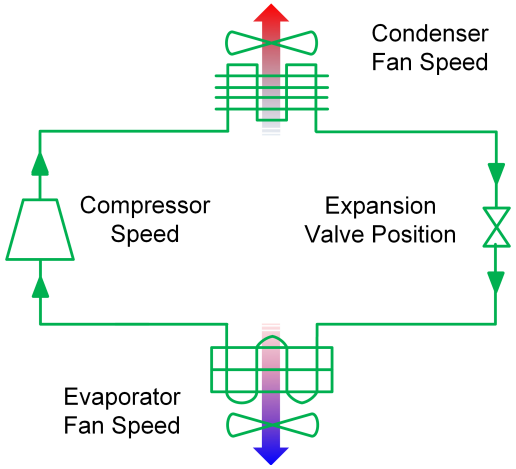
$$x(t) = W_{out}(p) r(t)$$

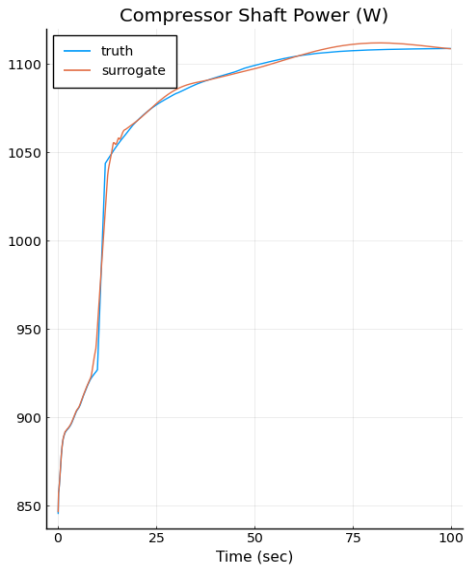
Using a Radial Basis Function constructed from the W_{out} training data

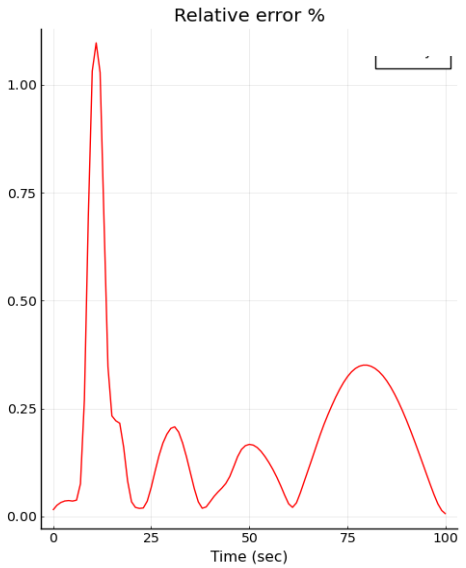
ARPA-E: Accelerated Simulation of Building Energy Efficiency

8,000 ODE Highly stiff vapor-compression cycle model

MITSUBISHI ELECTRIC
Changes for the Better







The Julia implementation is 6x faster than Dymola for the full cycle simulation.

- Dymola reference model: 35.3 s
- Julia (as close to) equivalent model: 5.8 s
- Could be due to details such as the linear solvers, the refrigerant property libraries, etc. More benchmarking to come.

Using CTESNs as surrogates improves simulation times between 10x-95x over the Julia baseline. Acceleration depends on the size of the reservoir in the CTESN. **The surrogate approximates 20 of the observables.**

Training set size	Reservoir size	Prediction time	Speedup over baseline
100	1000	0.06 s	95x
1000	2000	0.56 s	10x

Error is < 5% in all cases.

Total speedup over Dymola: 60-570x

Take Arbitrary Large Models and Automatically Accelerate with CTESNs

1265 ODE model of spatial cell signaling in Arabidopsis

BioModels

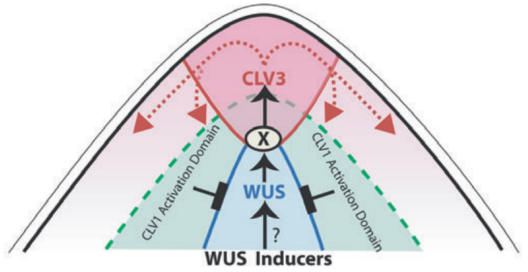
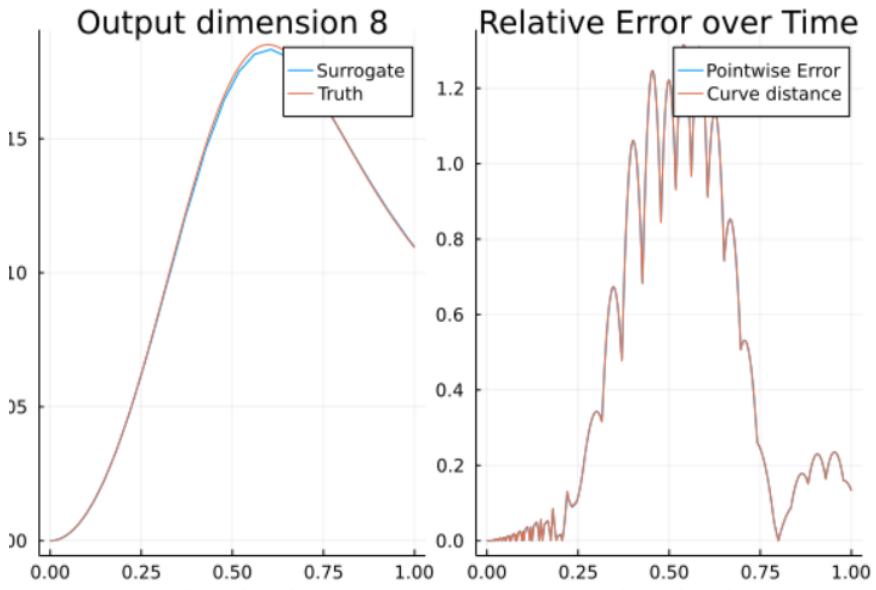


Fig. 1. A schematic of the expression domains of *CLAVATA1*, *CLAVATA3* and *WUSCHEL*. The solid arrows indicate the regulatory (indirect) interactions and the dashed arrows show the movement of the CLV3 protein.



- COPASI simulation: crashed upon reading (“not responding”)
- MATLAB SBMLToolbox: 870s to read, 1.13s to simulate
- Julia vanilla: 60s to read, 0.6s to simulate
- Julia surrogatized simulation: ~instant to read, 0.062s to simulate

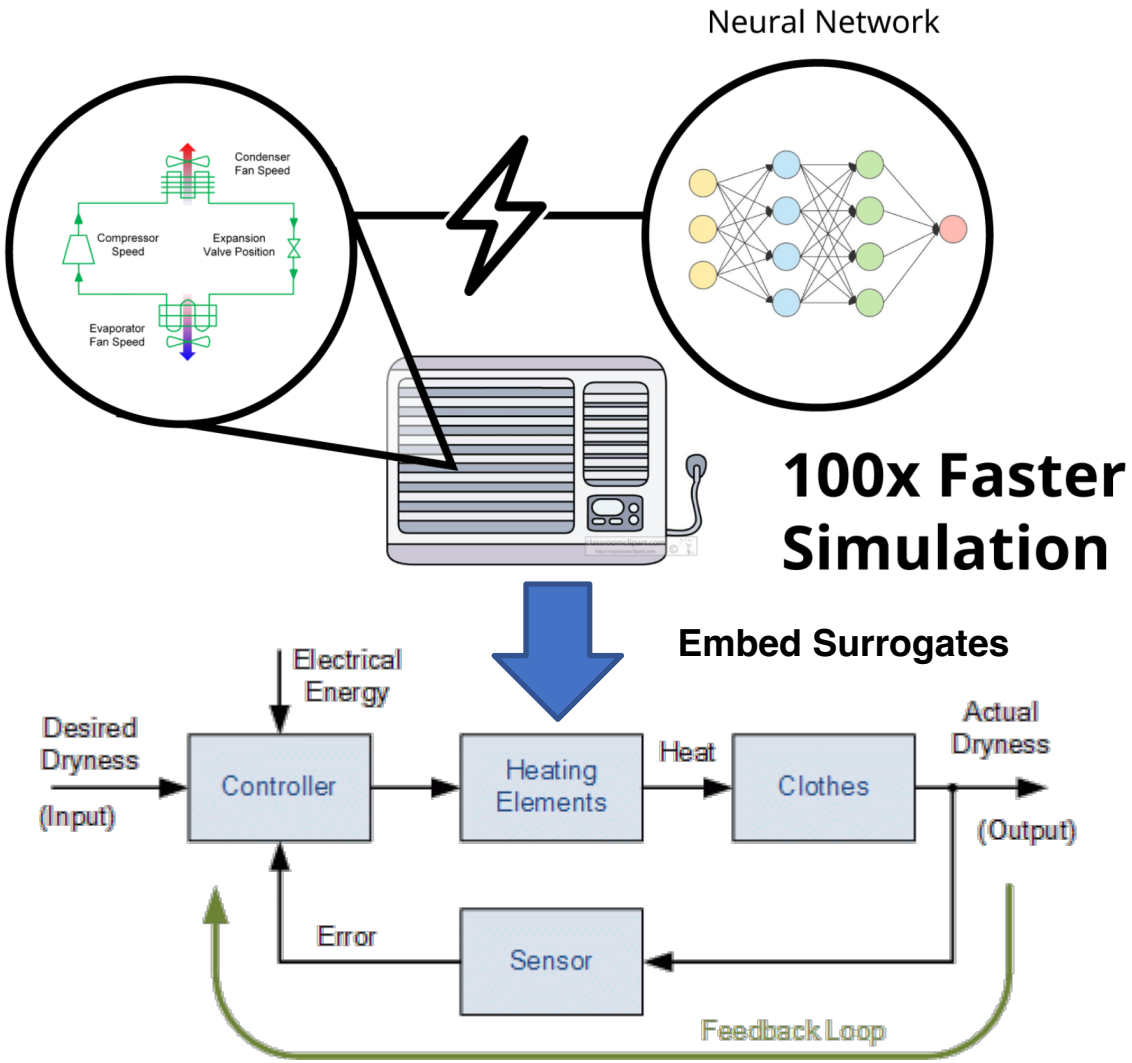
Julia vanilla outperforms MATLAB’s SBMLToolbox

CTESN predictions at new parameters have < 5% error, are almost instant to read and 100x faster to simulate

(Julia SBML reader is incomplete: full Jacobians right now and no e-graph simplification. Probably ~10x performance left on the table)

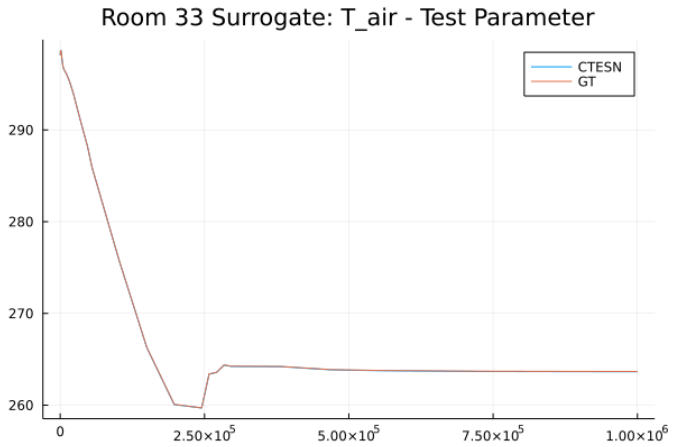
Total speedup: 100x vs MATLAB SBMLToolbox

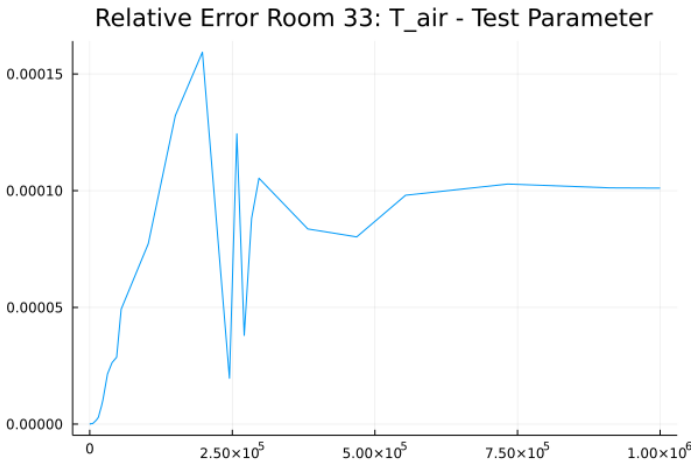
The Transformed Models are Just Components: Compose As Normal



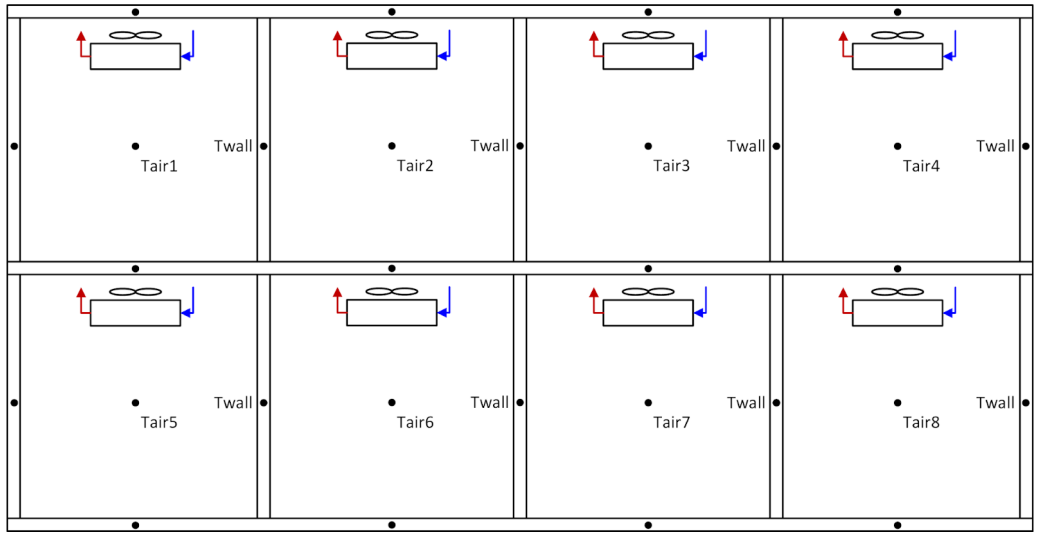
Accelerate large (100,000 ODE) simulations without retraining by using an accelerated HVAC component inside of different building models

Large Building Models 100K Equations, 80x Acceleration

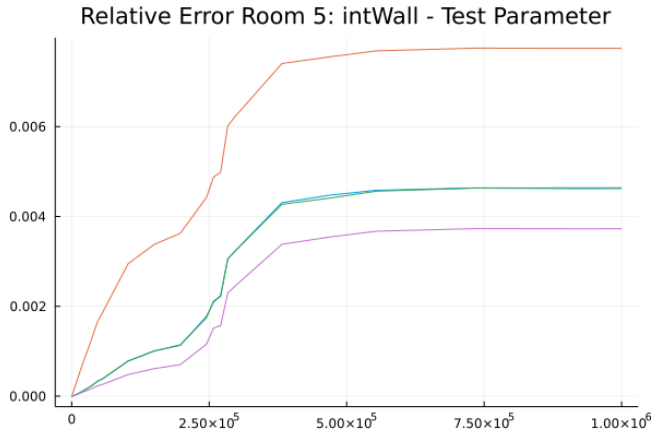


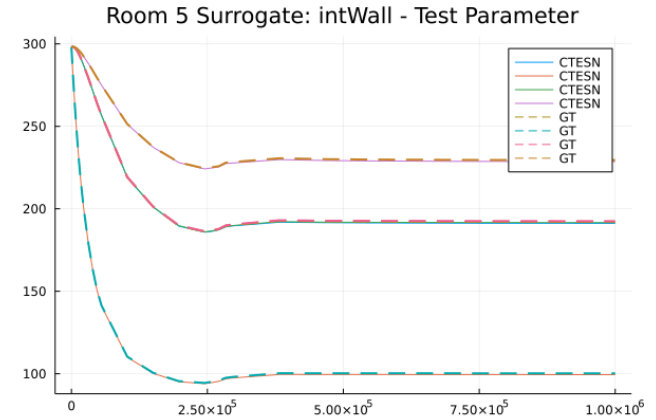


Rooms Disturbed	Training set size	Reservoir size	Prediction time	Speedup over baseline
1	100	200	0.2597 s	77x
3	100	200	0.413s	80x



Scalable building model with equipment





Total speedup over original : 80x

Therefore, any solver you can differentiate can do UDE things

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

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

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

$$\vdots$$

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

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

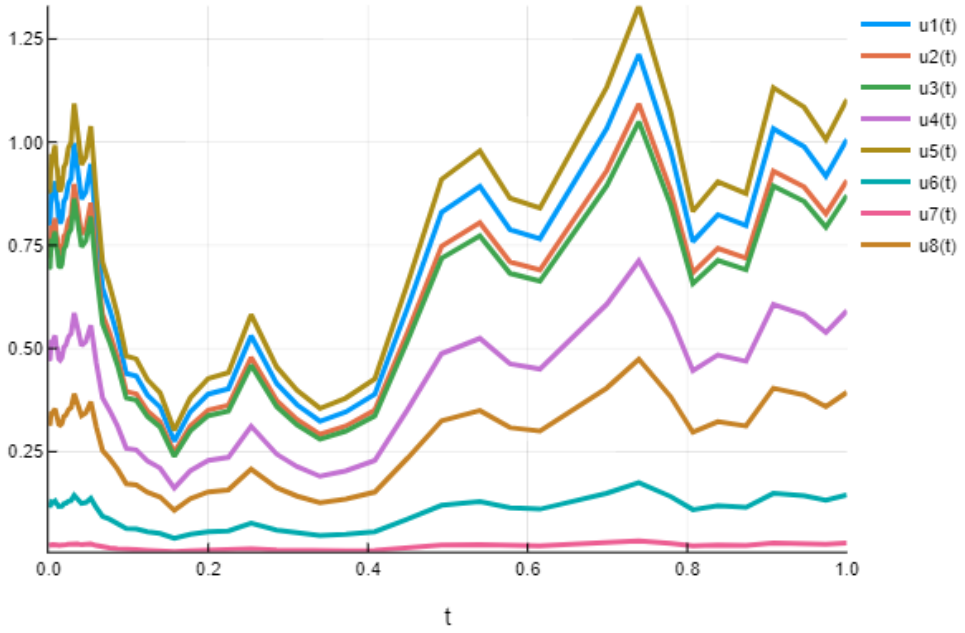
The SciML Common Interface for Julia Equation Solvers

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

Differential Equations Go Beyond ODEs

- Discrete equations (function maps, discrete stochastic (Gillespie/ Markov) simulations)
- Ordinary differential equations (ODEs)
- Split and Partitioned ODEs (Symplectic integrators, IMEX Methods)
- Stochastic ordinary differential equations (SODEs or SDEs)
- Stochastic differential-algebraic equations (SDAEs)
- Random differential equations (RODEs or RDEs)
- Differential algebraic equations (DAEs)
- Delay differential equations (DDEs)
- Neutral, retarded, and algebraic delay differential equations (NDDEs, RDDEs, and DDAEs)
- Stochastic delay differential equations (SDDEs)
- Experimental support for stochastic neutral, retarded, and algebraic delay differential equations (SNDDEs, SRDDEs, and SDDAEs)
- Mixed discrete and continuous equations (Hybrid Equations, Jump Diffusions)
- (Stochastic) partial differential equations ((S)PDEs) (with both finite difference and finite element methods)

…



But if you keep adding solver choices, then you're okay?

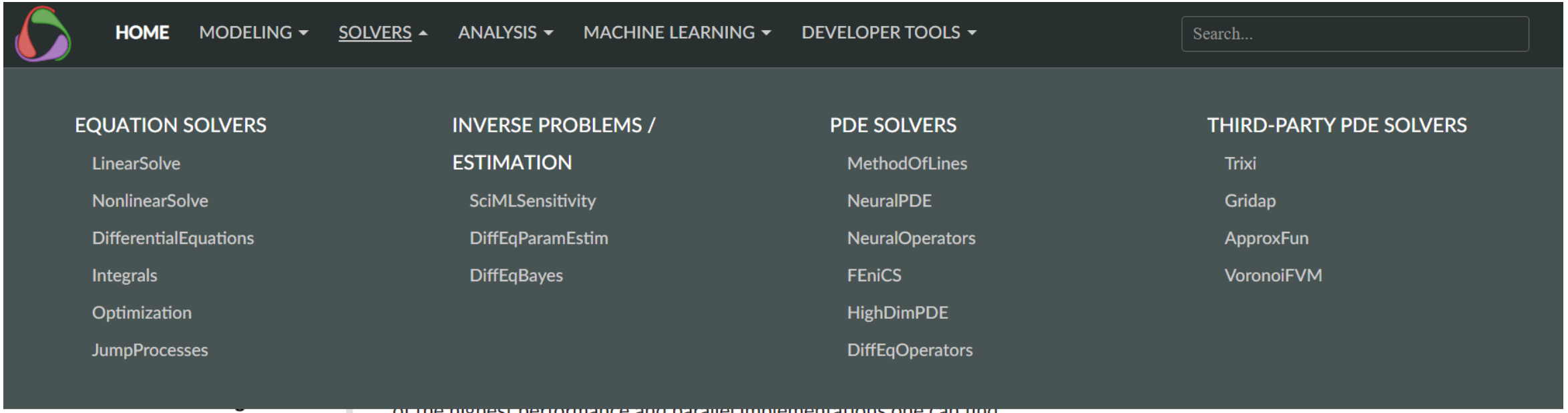
Unified Interfaces to Partial Differential Tooling

```
using ModelingToolkit
import ModelingToolkit: Interval, infimum, supremum

@parameters x y
@variables u(..)
Dxx = Differential(x)^2
Dyy = Differential(y)^2

# 2D PDE
eq  = Dxx(u(x,y)) + Dyy(u(x,y)) ~ -sin(pi*x)*sin(pi*y)

# Boundary conditions
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)]
# Space and time domains
domains = [x ∈ Interval(0.0,1.0),
           y ∈ Interval(0.0,1.0)]
pde_system = PDESystem(eq,bcs,domains,[x,y],[u])
```

Lots of Auto-Discretizers:

- Physics-Informed NNs: NeuralPDE.jl
- Finite Difference / WENO: MethodOfLines.jl
- Neural Operators: NeuralOperators.jl
- Finite Volume: Trixi.jl
- Finite Element: Gridap.jl
- Pseudospectral: ApproxFun.jl
- High Dimension: HighDimPDE.jl

New SciML Docs: Comprehensive Documentation of Differentiable Simulation



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.

Version v0.2

SciML Interface Coverage is Growing: Bringing AD to All Solvers by Default

LinearSolve.jl

1. SuiteSparse.jl (KLU, UMFPACK)
2. RecursiveFactorization.jl
3. Base.LinearAlgebra
4. FastLapackInterface.jl
5. Pardiso.jl
6. CUDA.jl (automated GPU offloading)
7. IterativeSolvers.jl
8. Krylov.jl
9. KrylovKit.jl

…

More keep being added (PETSc, Magma, HSL, Hypre, Elemental, CuSolverRF, …)

NonlinearSolve.jl

1. NLsolve.jl (KLU, UMFPACK)
2. SteadyStateDiffEq.jl
3. MINPACK
4. SUNDIALS (KINSOL)
5. New methods

…

More keep being added (PETSc, SpeedMapping.jl, etc.)

SciML Interface Coverage is Growing: Bringing AD to All Solvers by Default

Integrals.jl

1. QuadGK.jl
2. Cuba.jl
3. Cubature.jl
4. Hcubature.jl
5. MonteCarloIntegration.jl

Optimization.jl

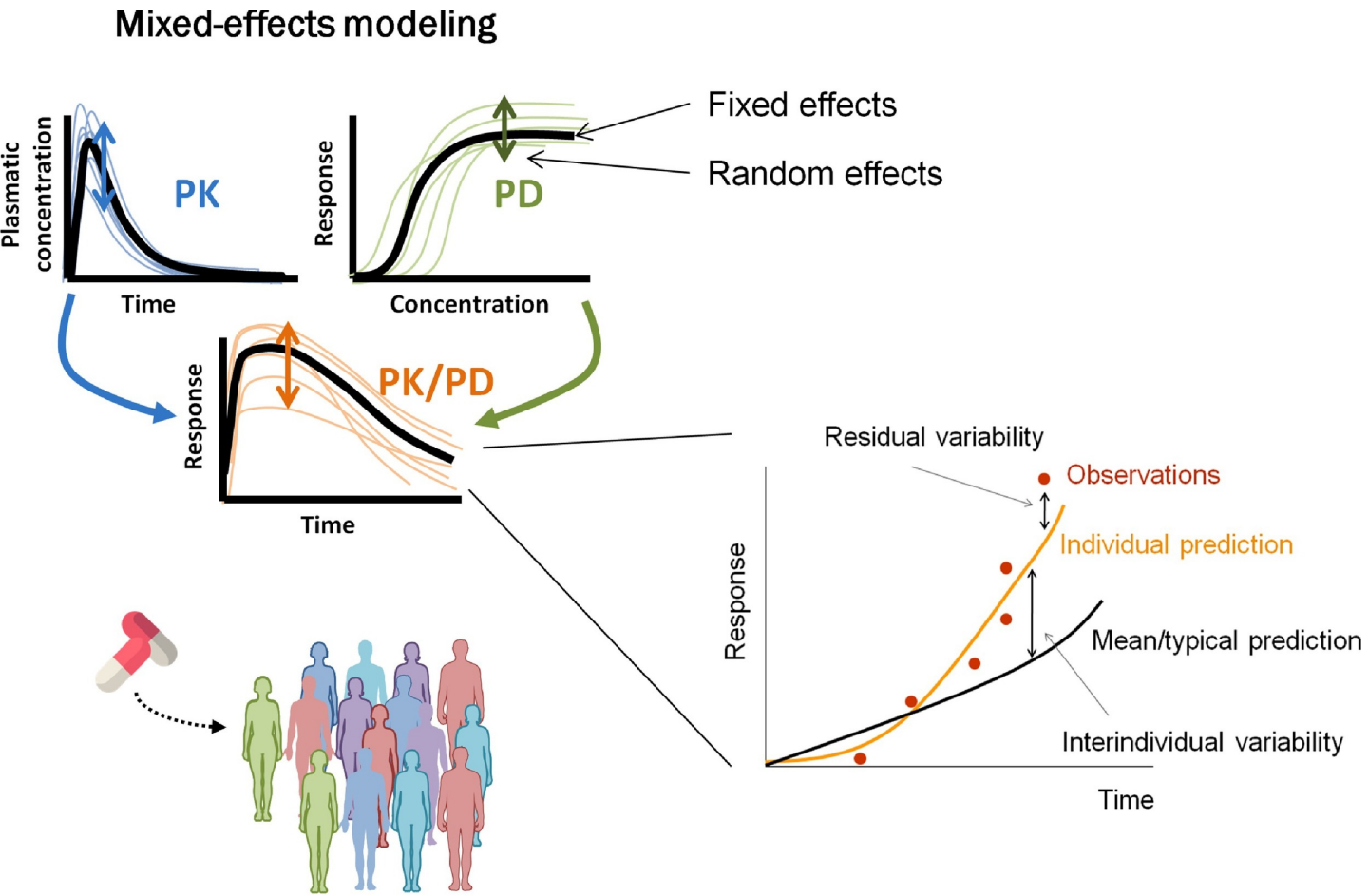
Overview of the Optimizers

Package	Local Gradient-Based	Local Hessian-Based	Local Derivative-Free	Local Constrained	Global Unconstrained	Global Constrained
BlackBoxOptim	✗	✗	✗	✗	✓	✗
CMAEvolutionaryStrategy	✗	✗	✗	✗	✓	✗
Evolutionary	✗	✗	✗	✗	✓	●
Flux	✓	✗	✗	✗	✗	✗
GCMAES	✗	✗	✗	✗	✓	✗
MathOptInterface	✓	✓	✓	✓	✓	●
MultistartOptimization	✗	✗	✗	✗	✓	✗
Metaheuristics	✗	✗	✗	✗	✓	●
NOMAD	✗	✗	✗	✗	✓	●
NLopt	✓	✗	✓	●	✓	●
Nonconvex	✓	✓	✓	●	✓	●
Optim	✓	✓	✓	✓	✓	✓
QuadDIRECT	✗	✗	✗	✗	✓	✗

What's something that's not quite "just an ODE" where the UDE technique can give an equation discovery method?

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

DeepNLME is SciML-enhanced modeling for clinical trials



- 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

From Dynamics to Nonlinear Mixed Effects (NLME) Modeling

Goal: Learn to predict patient behavior (dynamics) from simple data (covariates)

$$Z_i = \begin{bmatrix} wt_i, \\ sex_i, \end{bmatrix}$$

Covariates

Math: Find (θ, η) such that $\mathrm{E}[\eta] = 0$

Requires special fitting procedures (Pumas)

$$g_i = \begin{bmatrix} Ka \\ CL \\ V \end{bmatrix} = \begin{bmatrix} \theta_1 e^{\eta_{i,1} \kappa_{i,k,1}}, \\ \theta_2 (\frac{wt_i}{70})^{0.75} \theta_4^{sex_i} e^{\eta_{i,2}}, \\ \theta_3 e^{\eta_{i,3}}, \end{bmatrix}$$

Structural Model (pre)

$$\frac{d[\text{Depot}]}{dt} = -Ka[\text{Depot}],$$
$$\frac{d[\text{Central}]}{dt} = Ka[\text{Depot}] - \frac{CL}{V}[\text{Central}].$$

Dynamics

Intution: η (the random effects) are a fudge factor

Find θ (the fixed effect, or average effect) such that you can predict new patient dynamics as good as possible

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

From Dynamics to Nonlinear Mixed Effects (NLME) Modeling

Goal: Learn to predict patient behavior (dynamics) from simple data (covariates)

$$Z_i = \begin{bmatrix} wt_i, \\ sex_i, \end{bmatrix}$$

Covariates

Math: Find (θ, η) such that $\mathrm{E}[\eta] = 0$

$$g_i = \begin{bmatrix} Ka \\ CL \\ V \end{bmatrix} = \begin{bmatrix} \theta_1 e^{\eta_{i,1}\kappa_{i,k,1}}, \\ \theta_2 (\frac{wt_i}{70})^{0.75} \theta_4^{sex_i} e^{\eta_{i,2}}, \\ \theta_3 e^{\eta_{i,3}}, \end{bmatrix}$$

Structural Model (pre)

How can we find these models?

$$\frac{d[\text{Depot}]}{dt} = -Ka[\text{Depot}],$$
$$\frac{d[\text{Central}]}{dt} = Ka[\text{Depot}] - \frac{CL}{V}[\text{Central}].$$

Dynamics

Intution: η (the random effects) are a fudge factor

Find θ (the fixed effect, or average effect) such that you can predict new patient dynamics as good as possible

From Dynamics to Nonlinear Mixed Effects (NLME) Modeling

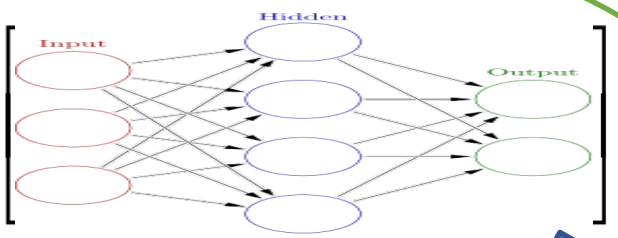
Goal: Learn to predict patient behavior (dynamics) from simple data (covariates)

$$Z_i = \begin{bmatrix} wt_i, \\ sex_i, \end{bmatrix}$$

Covariates

Math: Find (θ, η) such that $\mathrm{E}[\eta] = 0$

$$g_i = \begin{bmatrix} Ka \\ CL \\ V \end{bmatrix} =$$

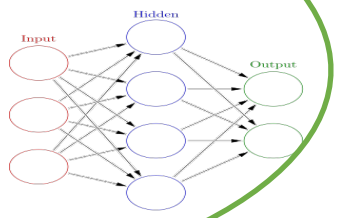


Structural Model (pre)

How can we find these models?

$$\frac{d[\text{Depot}]}{dt} = -Ka[\text{Depot}],$$

$$\frac{d[\text{Central}]}{dt} = Ka[\text{Depot}] -$$

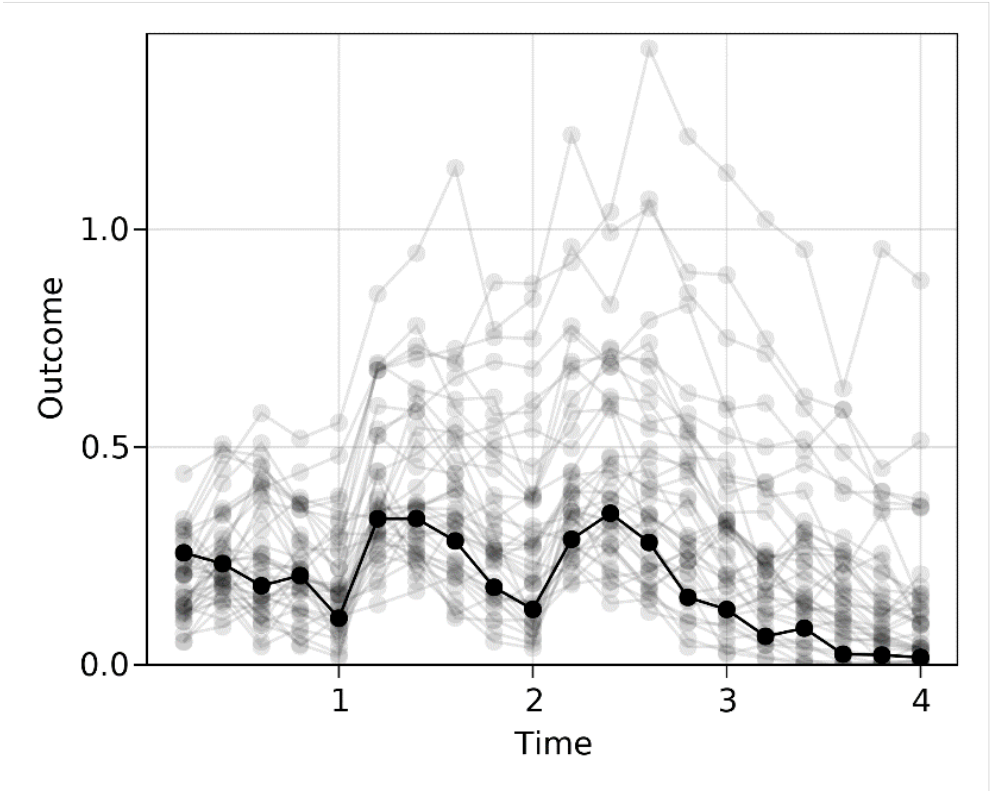


Dynamics

Idea: Parameterize the model such that the models can be neural networks, where the weights of the neural networks are fixed effects!

Indirect learning of unknown functions!

From Dynamics to Nonlinear Mixed Effects (NLME) Modeling



Typical values

$$\theta \in \mathbb{R}^3_+$$
$$\Omega \in \mathbb{R}^3_+$$

Patient data

Age
Weight

Random effects

$$\eta \sim \text{MvNormal}\,(\Omega)$$

Individual parameters

$$Ka_i = \theta_1 \cdot e^{\eta_{i,1}} + c_1 \cdot Age_i \quad +$$
$$CL_i = \theta_2 \cdot e^{\eta_{i,2}}$$
$$V_i = \theta_3 \cdot e^{\eta_{i,3}} + c_2 \cdot Weight_1^{c_3} \quad +$$

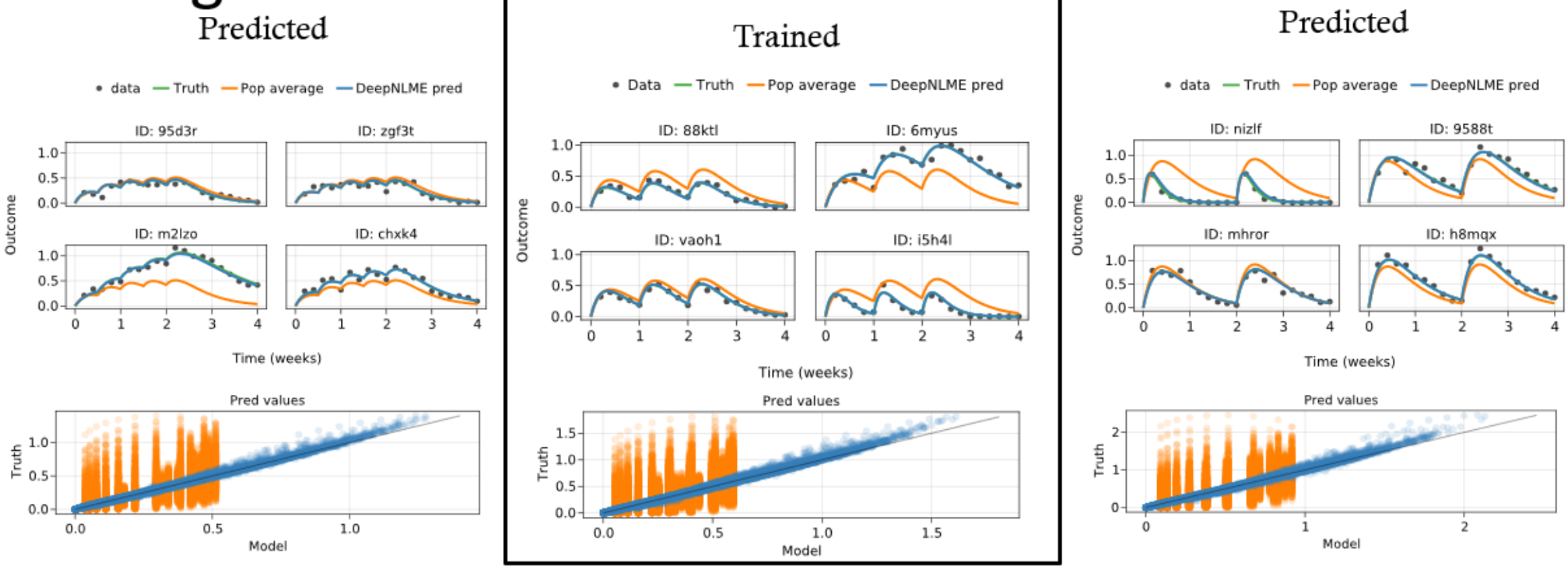
Dynamics

$$\frac{d[\text{Depot}]}{dt} = -Ka[\text{Depot}],$$
$$\frac{d[\text{Central}]}{dt} = Ka[\text{Depot}] -$$

Error model

$$Outcome \sim \text{Normal}\left(Central, \sqrt{Central} \cdot \sigma\right)$$

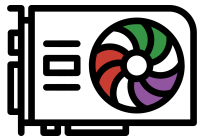
DeepNLME: Automated Construction of Patient-Specific Pharmacological Models for Individualized Dosing



Won International Society of Pharmacology (ISoP) Mathematics and Computing Special Interest Group Award at ACoP 2021 (Top Pharmacology Conference)

DeepNLME in Practice: Data Mining for Predictive Covariates

```
model = @model begin
  @param begin
    θ ∈ VectorDomain(lower=[0.1,0.0008,0.0040.1],upper=[5.0,0.5,0.9,5.0])
    Ω ∈ PSDDomain(3)
    σ²_add ∈ RealDomain(lower=0.001, init=sqrt(0.388))
    p1 ∈ NeuralDomain(FastChain(FastDense(2,50,tanh),FastDense(50,1),(x,p)->x.^2))
    p2 ∈ NeuralDomain(FastChain(FastDense(2,50,tanh),FastDense(50,1),(x,p)->x.^2))
  end

  @random begin η ~ MvNormal(Ω) end

  @pre begin
    Ka = SEX == 0 ? θ[1] + η[1] : θ[4] + η[1]
    K = nn1([θ[2],η[2]],p1)[1]
    CL = nn2([θ[3]*WT,η[3]],p2)[1]
    Vc = CL/K
    SC = CL/K/WT
  end

  @covariates SEX WT
  @vars begin conc = Central / SC end
  @dynamics Depots1Central1
  @derived begin dv ~ @. Normal(conc, sqrt(σ²_add)) end
end
```



Utilize GPU acceleration for neural networks

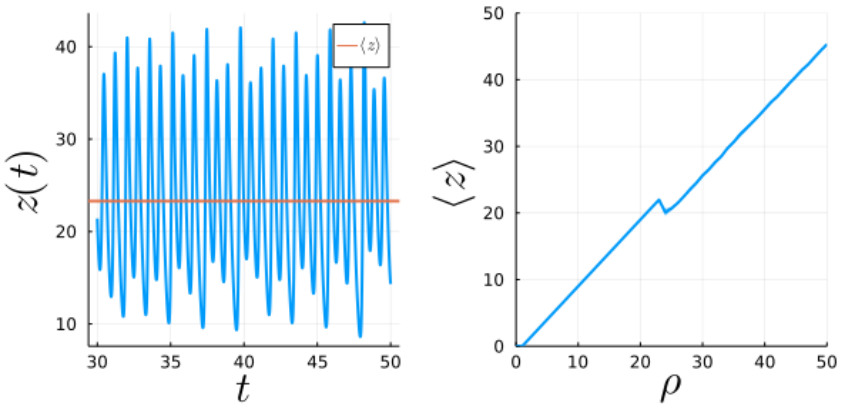
Automate the discovery of covariate models

- Train convolutional neural networks to incorporate images as covariates
- Train transformer models to utilize natural language processing on electronic health records
- Utilize automated model discovery to prune genomics data to find the predictive subset

Currently being tested on clinical trial data

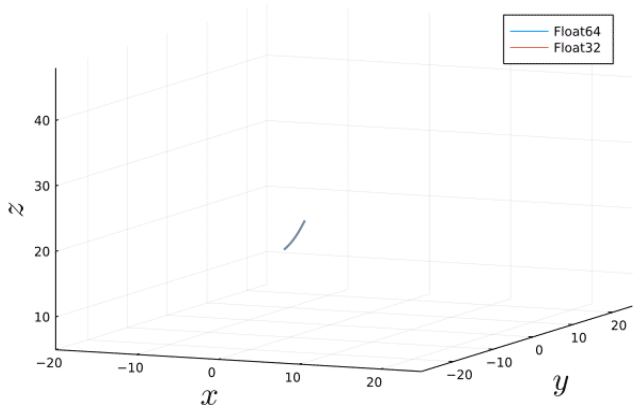
Can we generalize Differentiable Simulation beyond continuous models?

Differentiation of Chaotic Systems: Shadow Adjoints



$$\frac{\mathrm{d}}{\mathrm{d}\rho}\langle z \rangle_\infty \neq \lim_{T\to\infty} \frac{\partial}{\partial \rho}\langle z \rangle_T$$

chaotic systems: trajectories diverge to o(1) error … but shadowing lemma guarantees that the solution lies on the attractor



- AD and finite differencing fails!

$$\left.\frac{\mathrm{d}\langle z \rangle_\infty}{\mathrm{d}\rho}\right|_{\rho=28} \approx -49899 \text{ (ForwardDiff)}$$

$$\left.\frac{\mathrm{d}\langle z \rangle_\infty}{\mathrm{d}\rho}\right|_{\rho=28} \approx 472 \text{ (Calculus)}$$

- Shadowing methods in DiffEqSensitivity.jl

$$\left.\frac{\mathrm{d}\langle z \rangle_\infty}{\mathrm{d}\rho}\right|_{\rho=28} \approx 1.028 \text{ (LSS/AdjointLSS)}$$

$$\left.\frac{\mathrm{d}\langle z \rangle_\infty}{\mathrm{d}\rho}\right|_{\rho=28} \approx 0.997 \text{ (NILSS)}$$

https://frankschae.github.io/post/shadowing/

Differentiation of Chaotic Systems: Shadow Adjoints

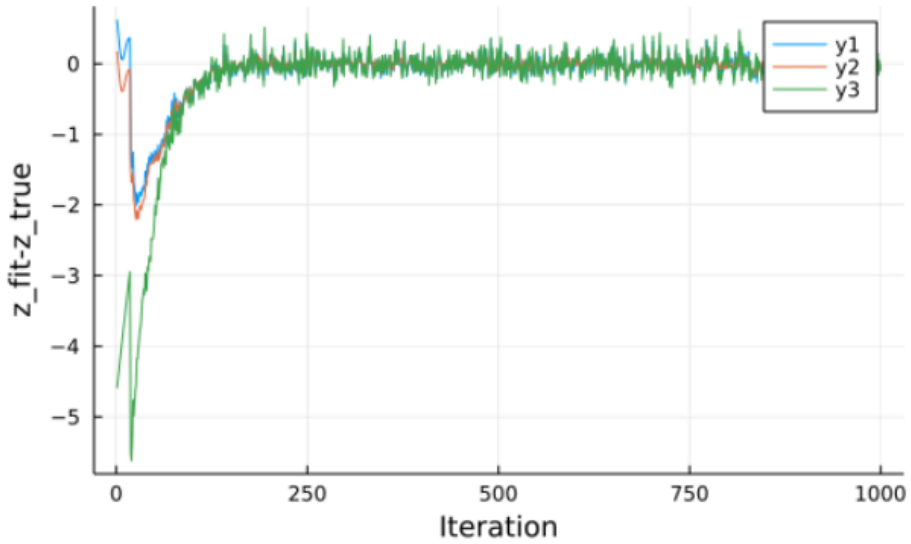


Figure 18: Summary statistic fitting corresponding to Figure 17. Fitting with all 3 Lorenz parameters free using 3 summary statistics ($< |x| >, < |y| >, < |z| >$). ADAM optimizer learning rate is 1e-1.

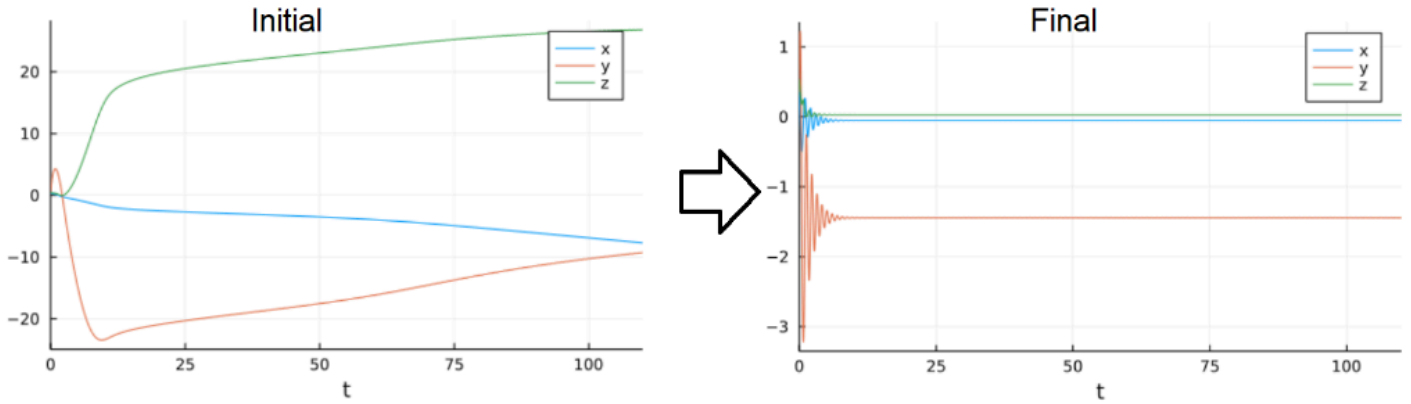
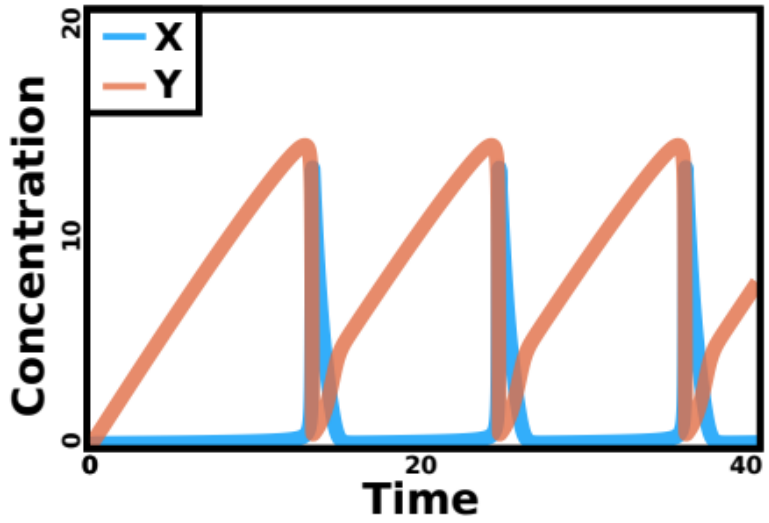


Figure 7: UODE fit with ADAM optimizer 1e-1 and re-initialized ADAM(1e-2) every 50 iterations. Summary statistics are all combinations of 2 summary statistics and standard deviations of each parameter xyz. Left is before and right is after training.

Poisson Jump Equations: Stochastic Chemical Reactions

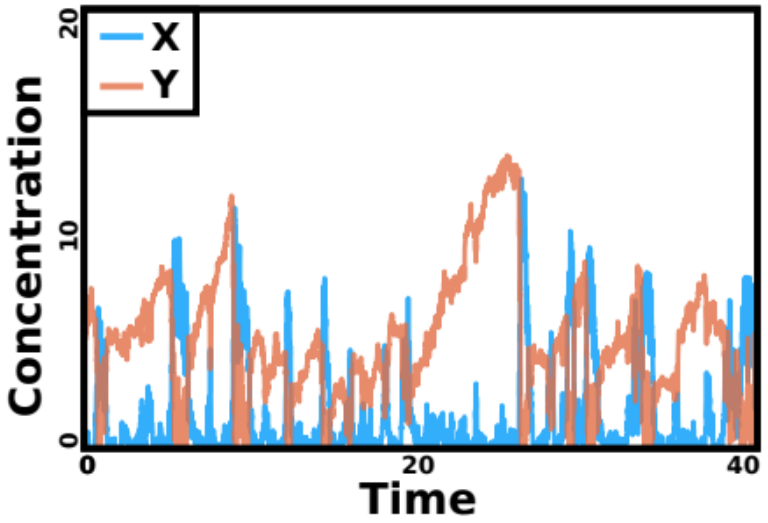
A

```
using OrdinaryDiffEq
u0 = [:X => 0., :Y => 0.]
tspan = (0., 60.)
p = [:A => 1.0, :B => 4.0]
oprob = ODEProblem(brusselator,u0,tspan,p)
sol = solve(oprob,Rosenbrock23())
plot(sol)
```



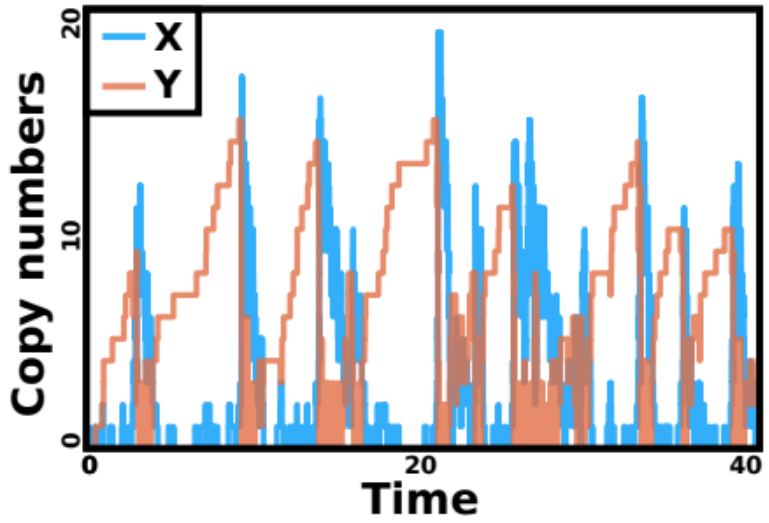
B

```
using StochasticDiffEq
u0 = [:X => 0., :Y => 5.]
tspan = (0., 60.)
p = [:A => 1.0, :B => 4.0]
sprob = SDEProblem(brusselator,u0,tspan,p)
sol = solve(sprob,ImplicitEM())
plot(sol)
```



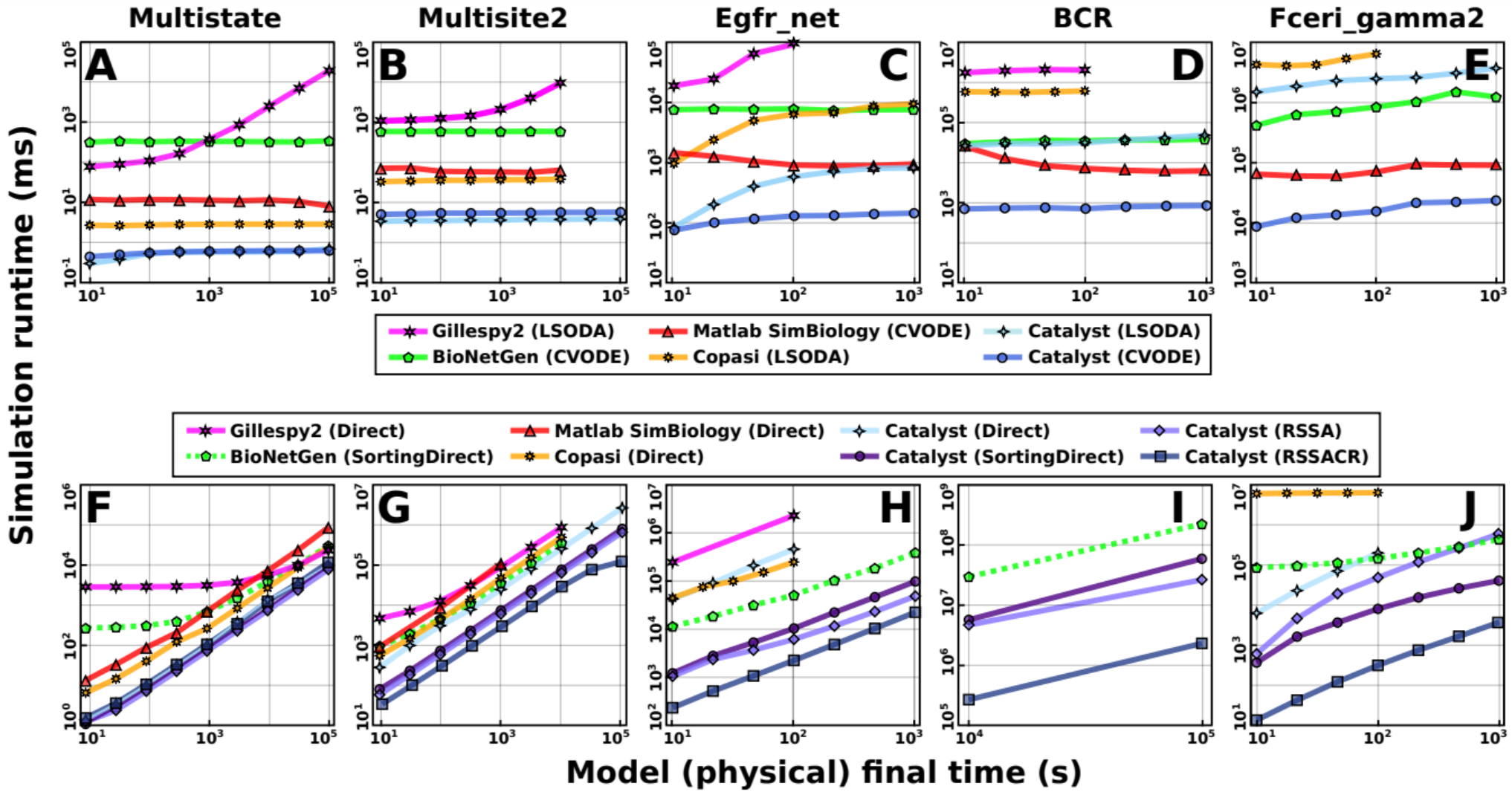
C

```
using DiffEqJump
u0 = [:X => 0, :Y => 0]
tspan = (0., 60.)
p = [:A => 3.0, :B => 4.0]
dprob = DiscreteProblem(brusselator,u0,tspan,p)
jprob = JumpProblem(brusselator,dprob,Direct())
sol = solve(jprob,SSAStepper())
plot(sol)
```



Loman, Torkel, Yingbo Ma, Vasily Ilin, Shashi Gowda, Niklas Korsbo, Nikhil Yewale, Christopher Vincent Rackauckas, and Samuel A. Isaacson. "Catalyst: fast biochemical modeling with Julia." *bioRxiv* (2022).

~100x performance improvements in Gillespie SSAs with Catalyst.jl



New Automatic Differentiation of Discrete Stochastic Spaces?

Traditional AD

$$f(p)$$

$$\frac{\mathrm{d}f(p)}{\mathrm{d}p}$$

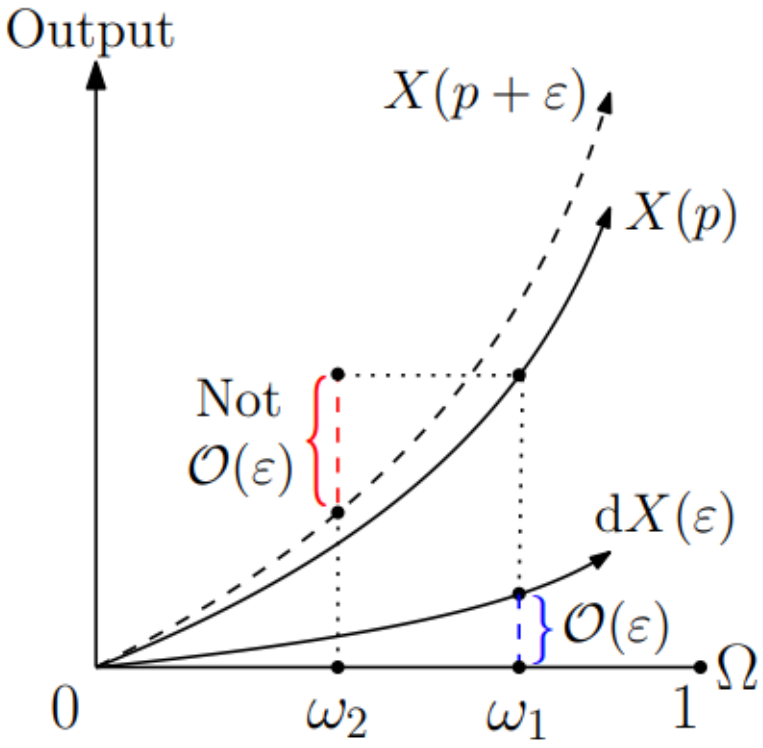
Discrete Stochastic AD

$$X(p)$$

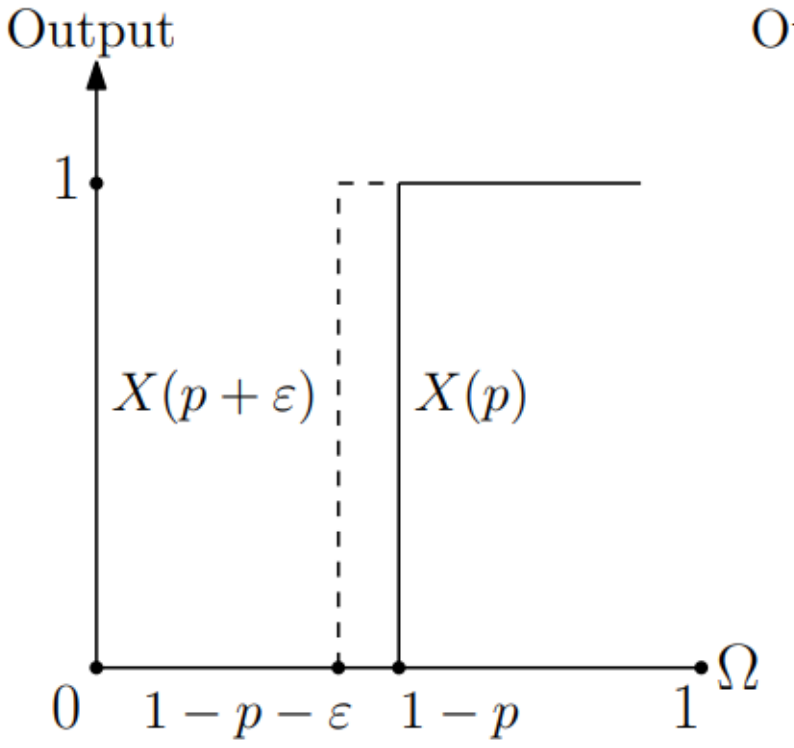
$$\mathbb{E}[\tilde{X}(p)] = \frac{\mathrm{d}\mathbb{E}[X(p)]}{\mathrm{d}p}$$

Automatic Differentiation of Programs with Discrete Randomness
Accepted to NeurIPS 2022!

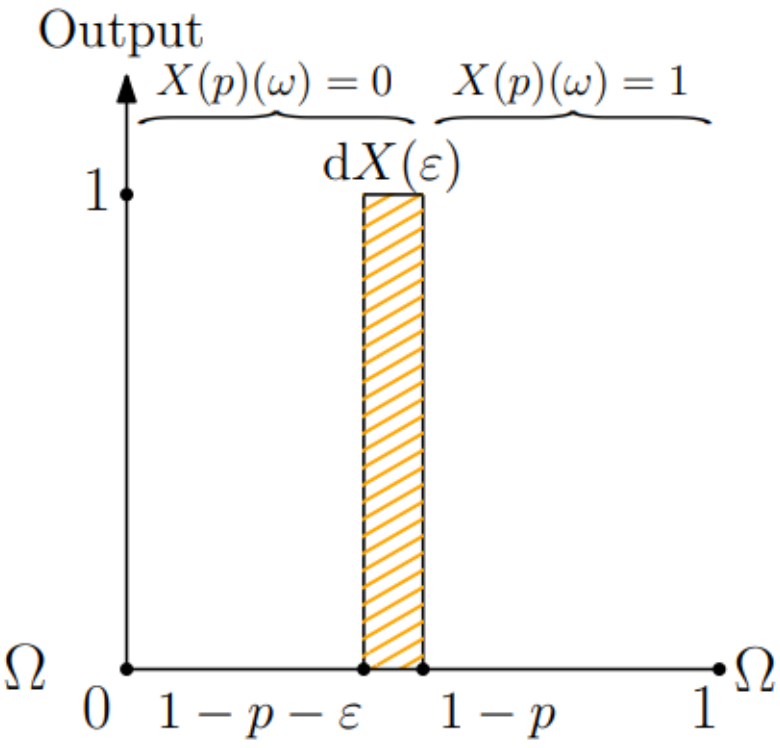
Infinitesimal Changes -> O(1) Changes



(a) $X(p) \sim \mathrm{Exp}(p)$.

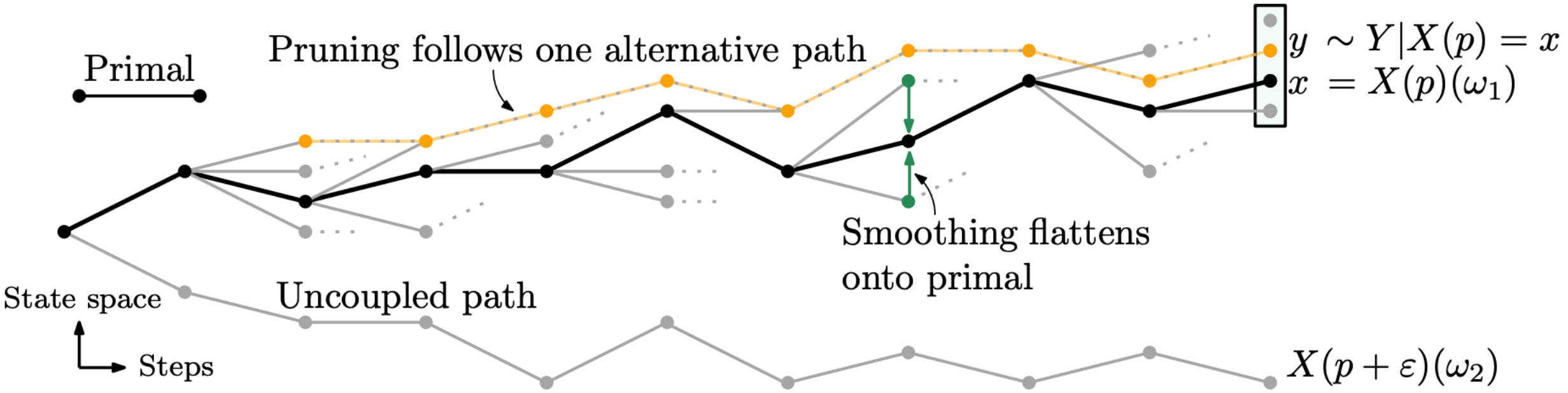


(b) $X(p) \sim \mathrm{Ber}(p)$.



(c) $X(p) \sim \mathrm{Ber}(p)$.

O(1) changes with infinitesimal probability



StochasticAD.jl: Unbiased Stochastic Derivatives with Linear Cost

Perform AD on Agent-Based models, particle filters, chemical reactions, and more!

```julia
struct StochasticTriple
    value # primal evaluation
    δ # "infinitesimal" component
    Δs # component of discrete change
       # with "infinitesimal"
       # probability
end
```

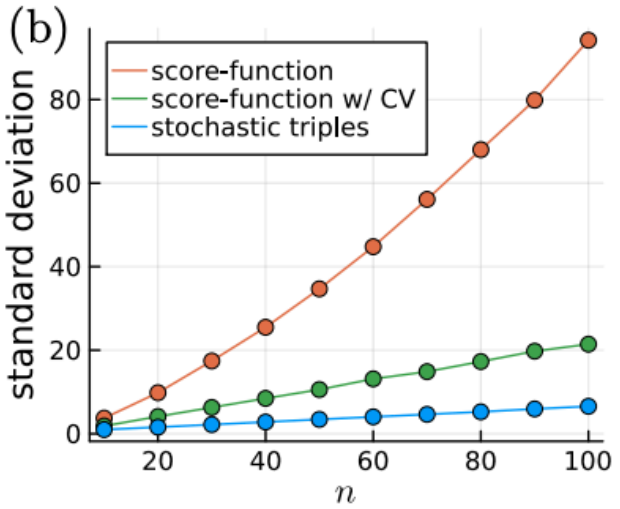
```julia
using Distributions
function X(p)  p = 0.6 + ε
    a = p^2  0.36 + 1.2ε
    b = rand(Binomial(10, p))
    6 + (1 with probability 10.0ε)
    c = 2 * b + 3 * rand(Bernoulli(p))
    12 + (3 with probability 12.5ε)
    return a * c * rand(Normal(p, p^2))
end
```

```
julia> using StochasticAD
julia> st = stochastic_triple(X, 0.6) # sample a single stochastic triple at p = 0.6
2.16 + 10.05ε + (0.64 with probability 12.5ε)
julia> derivative_contribution(st) # which produces a single derivative estimate...
16.79
julia> samples = [derivative_estimate(X, 0.6) for i in 1:1000] # take many estimates!
julia> println("d/dp of E[X(p)]: $(mean(samples)) ± $(std(samples) / sqrt(1000))")
d/dp of E[X(p)]: 19.39 ± 0.48
```

StochasticAD.jl on Agent-Based Models

(a)

```
...
X = 0
for step in 1:n
  i = rand(Categorical(probs(X)))
  X += steps[i]
end
return f(X)
```



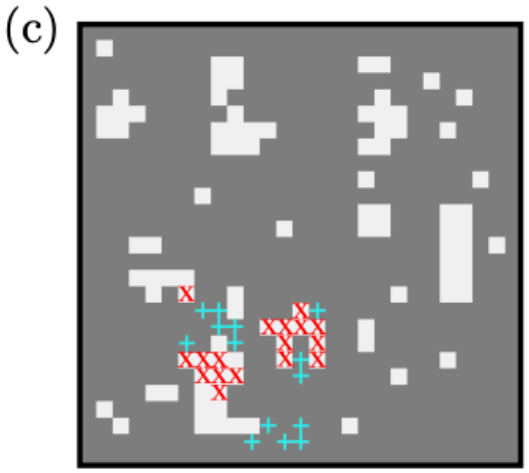
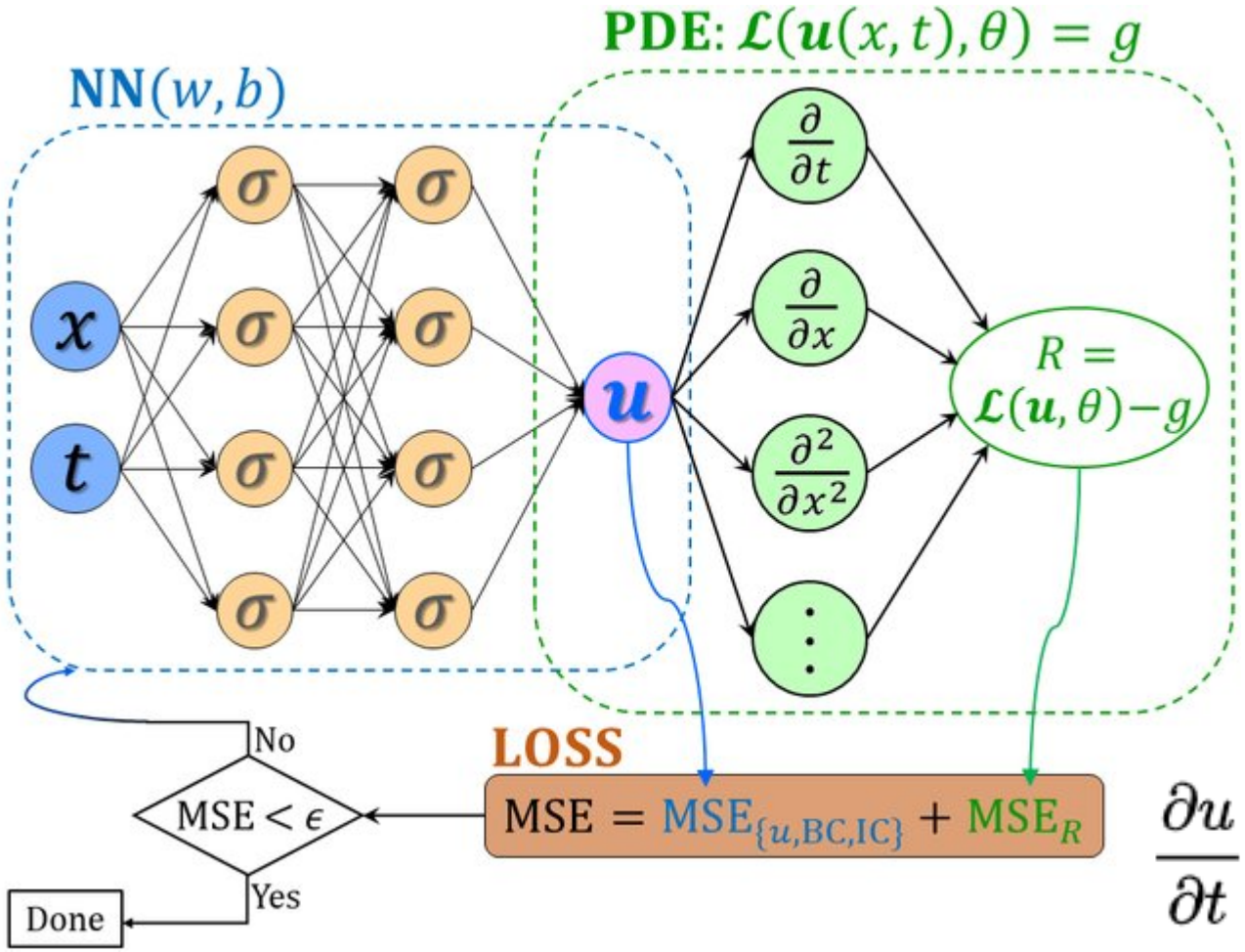


Figure 4: Automatic differentiation of discrete-time Markov processes. **(a)** Code snipped for a 1D random walk, which can be automatically differentiated by `StochasticAD.jl`; `probs(X)` gives the transition probabilities at value `X` and `steps[i]` gives the step size for the `i`th transition. **(b)** The variance of unbiased gradient estimates of the random walk program using stochastic triples and the score function, which is applied both without a control variate (CV) and with a pre-computed batch-average CV. **(c)** The final board for one run of the stochastic Game of Life with $N = 25$ and $T = 10$, where the "+" signs represent additional living cells (white) in the stochastic alternative path, and the "X" signs represent additional dead cells (grey).

But wait, why not PINNs?

How does differentiable simulation compare to PINNs?

Physics-Informed Neural Networks



Approximate the PDE solution u as a neural network

Make a loss function be that its derivatives must solve the PDE

Use gradient descent

…

Now the neural network is u which solves the PDE!

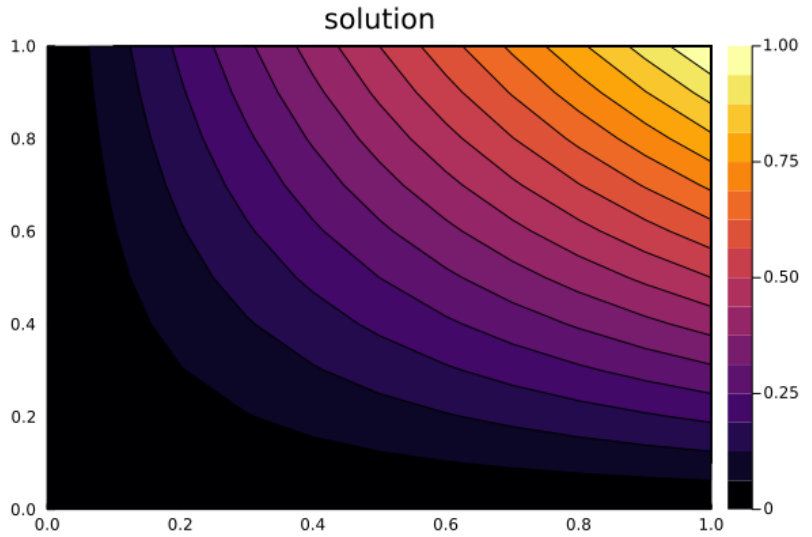
$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x_1^2} + \cdots + \frac{\partial^2 u}{\partial x_n^2},$$

ModelingToolkit's General PDE Solver: Physics-Informed Neural Networks

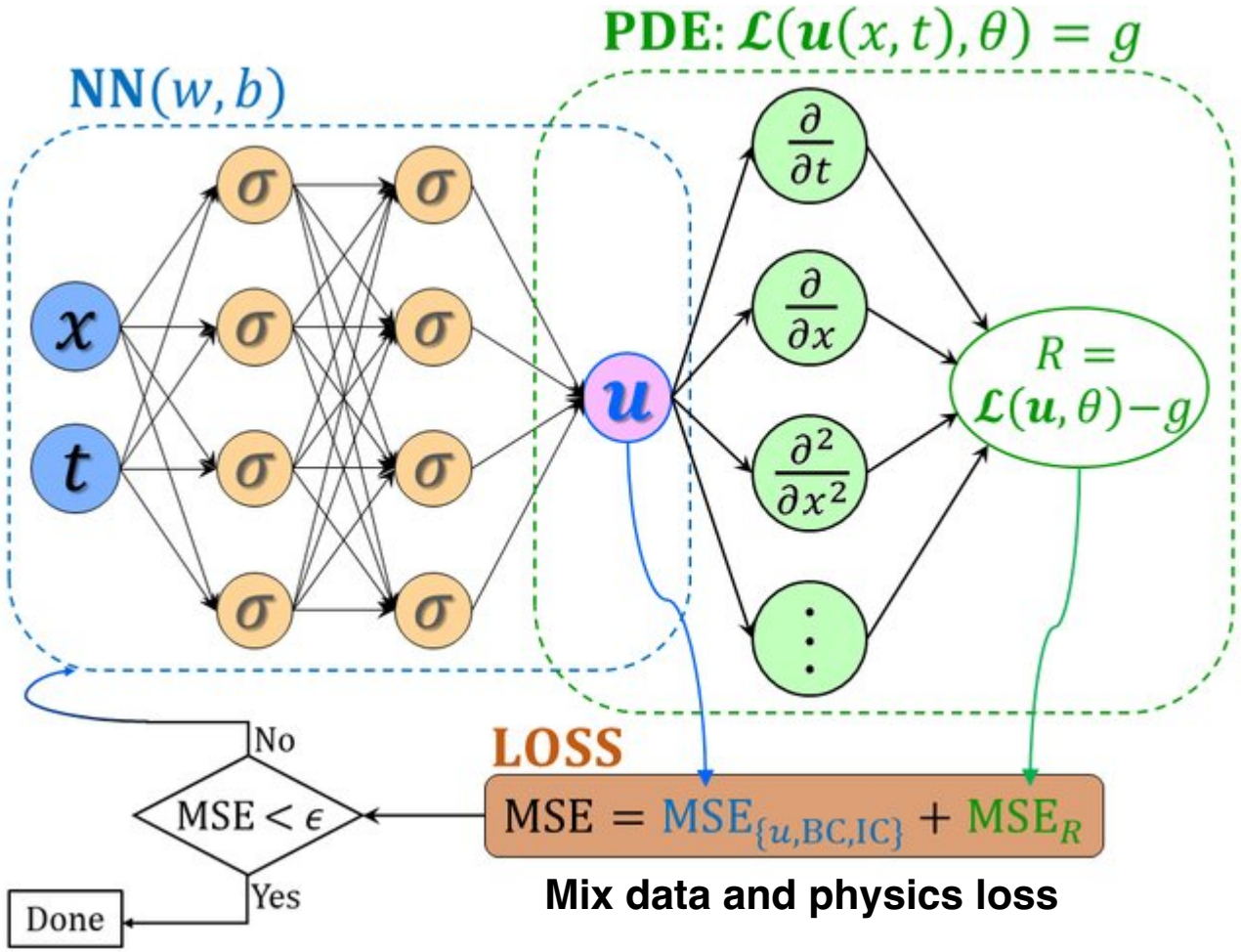
```
using ModelingToolkit
import ModelingToolkit: Interval, infimum, supremum

@parameters x y
@variables u(..)
Dxx = Differential(x)^2
Dyy = Differential(y)^2

# 2D PDE
eq  = Dxx(u(x,y)) + Dyy(u(x,y)) ~ -sin(pi*x)*sin(pi*y)

# Boundary conditions
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)]
# Space and time domains
domains = [x ∈ Interval(0.0,1.0),
           y ∈ Interval(0.0,1.0)]
pde_system = PDESystem(eq,bcs,domains,[x,y],[u])
```

Easy and Customizable PINN PDE Solving with NeuralPDE.jl, JuliaCon 2021

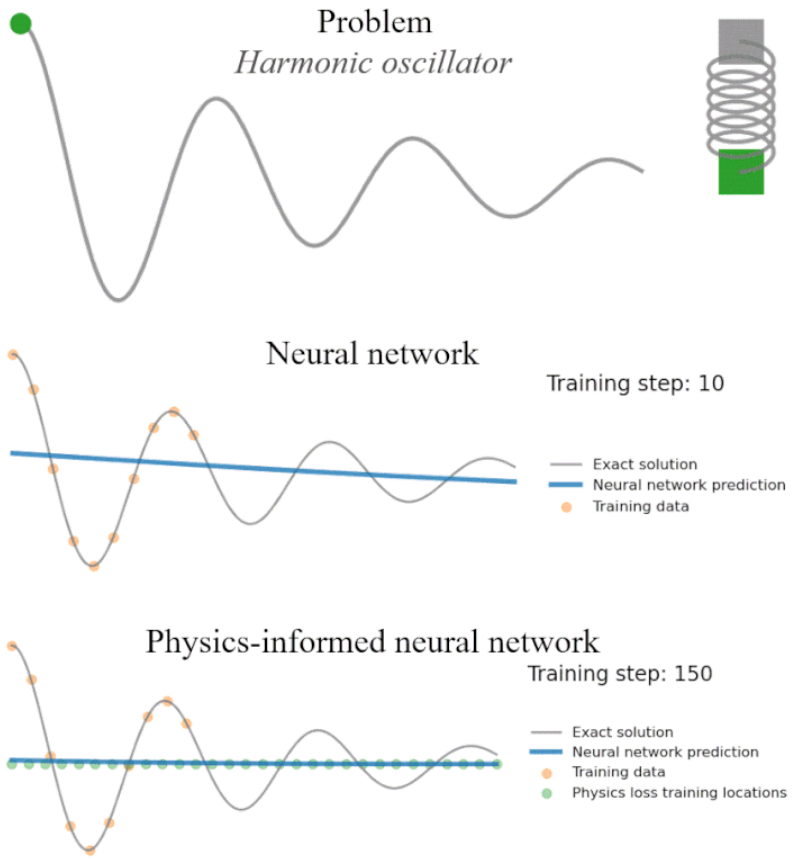
```
# Neural network
dim = 2 # number of dimensions
chain = FastChain(FastDense(dim,16,Flux.σ),
                  FastDense(16,16,Flux.σ),FastDense(16,1))
# Discretization
dx = 0.05
discretization = PhysicsInformedNN(chain,GridTraining(dx))
prob = discretize(pde_system,discretization)
#Optimizer
opt = Optim.BFGS()
res = GalacticOptim.solve(prob, opt, maxiters=1000)
```



Why Use Physics-Informed Neural Networks?

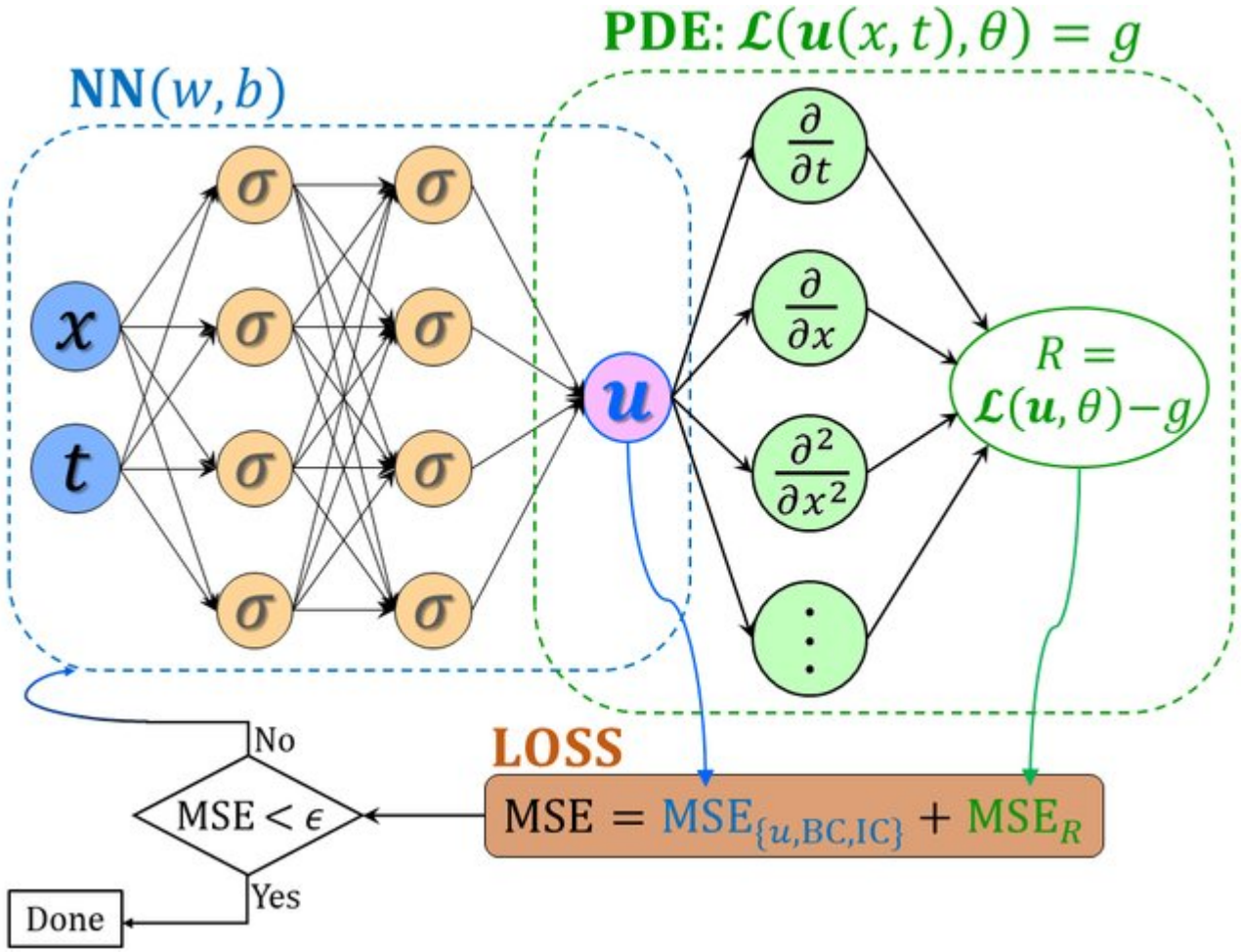


Mix data and physics loss



Outperforms standard machine learning

Why Use Physics-Informed Neural Networks?



Generality: Easily works for any PDE

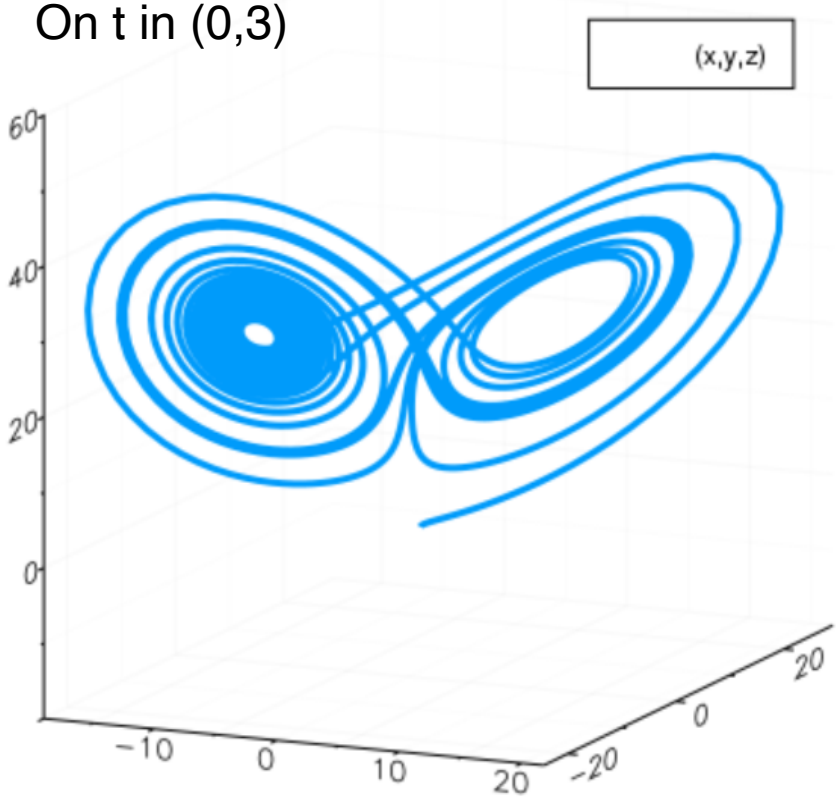
Simplicity: Doesn't require any simulation methods, just a neural network and automatic differentiation

Why not use a PINN?

Performance.

Keeping Neural Networks Small Keeps Speed For Inverse Problems

Problem: parameter estimation
of Lorenz equation from data
On t in (0,3)



DeepXDE (TensorFlow Physics-Informed NN)

```
Best model at step 57000:
  train loss: 5.91e-03
  test loss: 5.86e-03
  test metric: []

'train' took 362.351454 s
```

DiffEqFlux.jl (Julia UDEs)

```
opt = Opt(:LN_BOBYQA, 3)
lower_bounds!(opt,[9.0,20.0,2.0])
upper_bounds!(opt,[11.0,30.0,3.0])
min_objective!(opt, obj_short.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 10000)
@time (minf,minx,ret) = NLopt.optimize(opt,LocIniPar) # 0.1 seconds
```

```
0.032699 seconds (148.87 k allocations: 14.175 MiB)
(2.7636309213683456e-18, [10.0, 28.0, 2.66], :XTOL_REACHED)
```

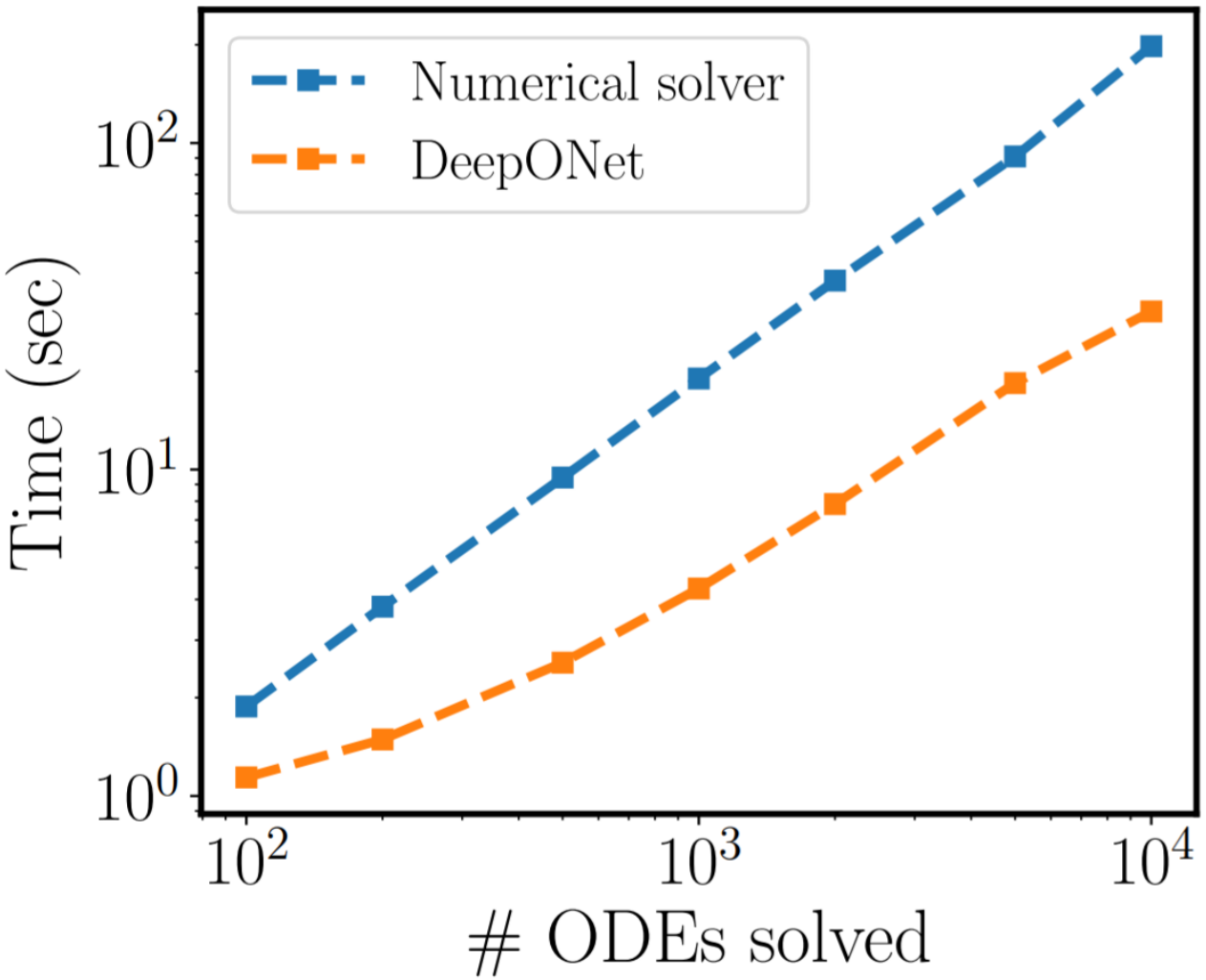
Note on Neural Networks “Outperforming” Classical Solvers

Long-time integration of parametric evolution equations with physics-informed DeepONets

Sifan Wang, Paris Perdikaris

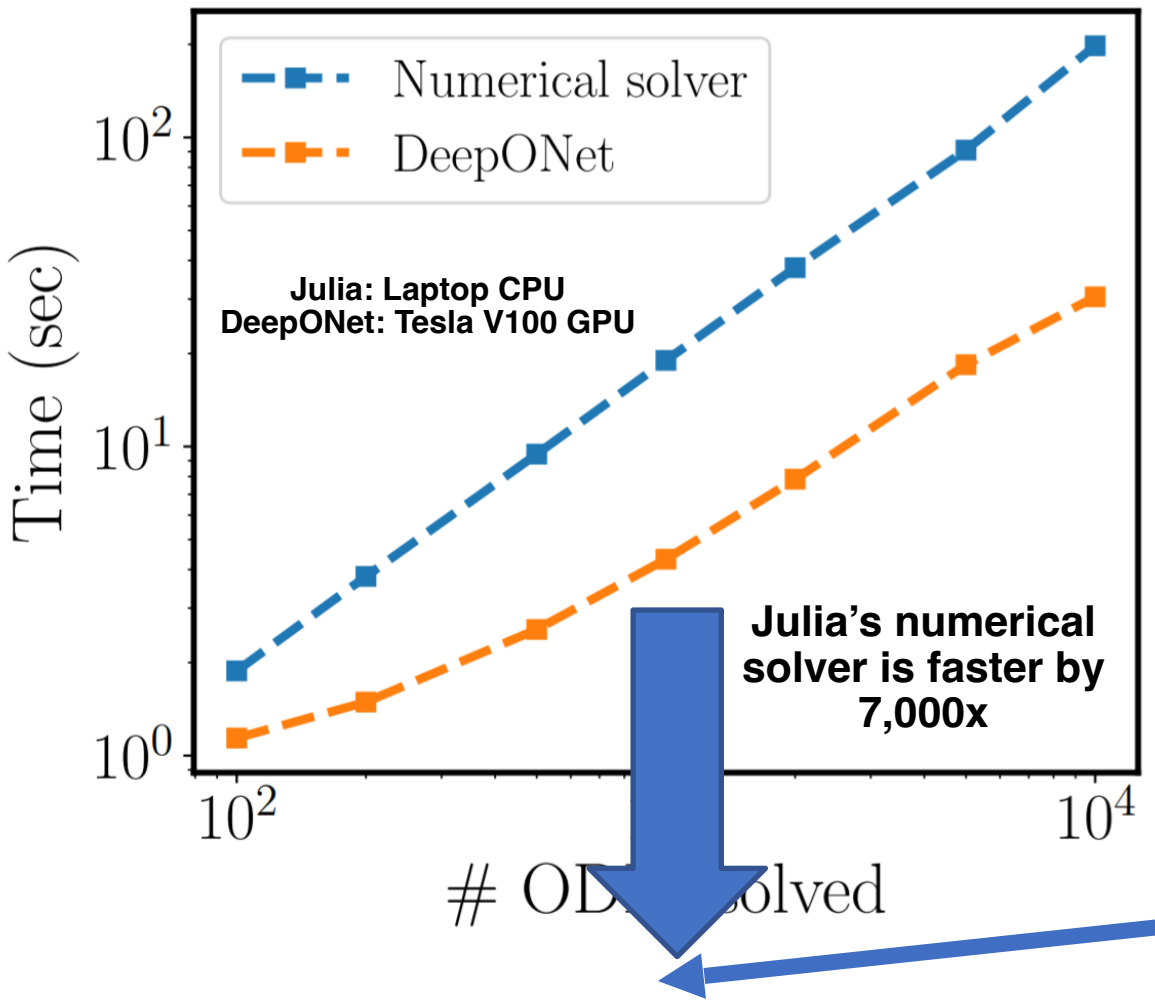
Ordinary and partial differential equations (ODEs/PDEs) play a paramount role in analyzing and simulating complex dynamic processes across all corners of science and engineering. In recent years machine learning tools are aspiring to introduce new effective ways of simulating PDEs, however existing approaches are not able to reliably return stable and accurate predictions across long temporal horizons. We aim to address this challenge by introducing an effective framework for learning infinite-dimensional operators that map random initial conditions to associated PDE solutions within a short time interval. Such latent operators can be parametrized by deep neural networks that are trained in an entirely self-supervised manner without requiring any paired input-output observations. Global long-time predictions across a range of initial conditions can be then obtained by iteratively evaluating the trained model using each prediction as the initial condition for the next evaluation step. This introduces a new approach to temporal domain decomposition that is shown to be effective in performing accurate long-time simulations for a wide range of parametric ODE and PDE systems, from wave propagation, to reaction-diffusion dynamics and stiff chemical kinetics, all at a fraction of the computational cost needed by classical numerical solvers.

Note on Neural Networks "Outperforming" Classical Solvers



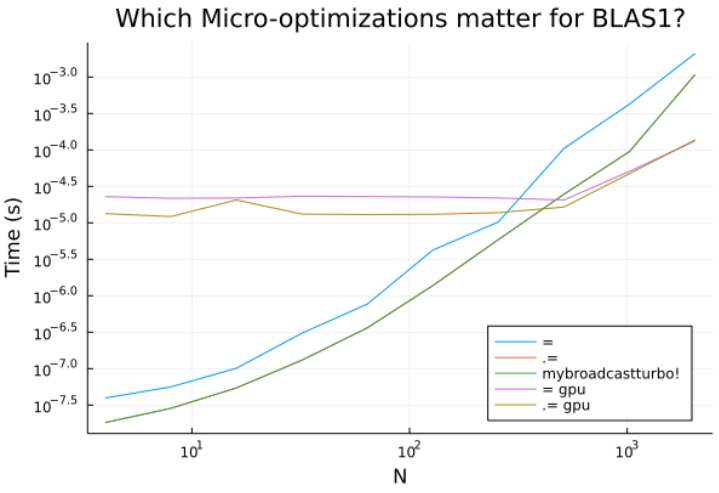
Oh no, we're doomed!

Stay tuned…




```julia
using ModelingToolkit, OrdinaryDiffEq, StaticArrays

@variables t y₁(t) y₂(t) y₃(t)
@parameters k₁ k₂ k₃
D = Differential(t)
eqs = [D(y₁) ~ -k₁*y₁+k₃*y₂*y₃
       D(y₂) ~  k₁*y₁-k₂*y₂^2-k₃*y₂*y₃
       D(y₃) ~  k₂*y₂^2]

sys = ODESystem(eqs, t)
prob = ODEProblem{false}(sys,SA[y₁=>1f0,y₂=>0f0,y₃=>0f0],(0f0,500f0),
                                SA[k₁=>4f-2,k₂=>3f7,k₃=>1f4],jac=true)

N = 1000
y₁s = rand(Float32,N)
y₂s = 1f-4 .* rand(Float32,N)
y₃s = rand(Float32,N)

function prob_func(prob,i,repeat)
    remake(prob,p=SA[y₁s[i],y₂s[i],y₃s[i]])
end

monteprob = EnsembleProblem(prob, prob_func = prob_func, safetycopy=false)
solve(monteprob,Rodas5(),EnsembleThreads(),trajectories=1000)

@time solve(monteprob,Rodas5(),EnsembleThreads(),trajectories=1000)

#0.006486 seconds (172.26 k allocations: 16.740 MiB)
#0.006024 seconds (172.26 k allocations: 16.740 MiB)
#0.007074 seconds (172.26 k allocations: 16.740 MiB)
```

Code Optimization in Machine Learning vs Scientific Computing

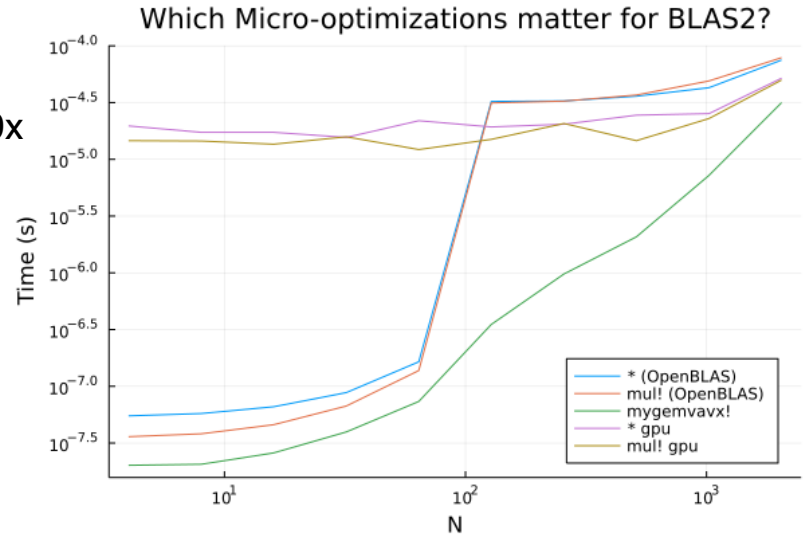


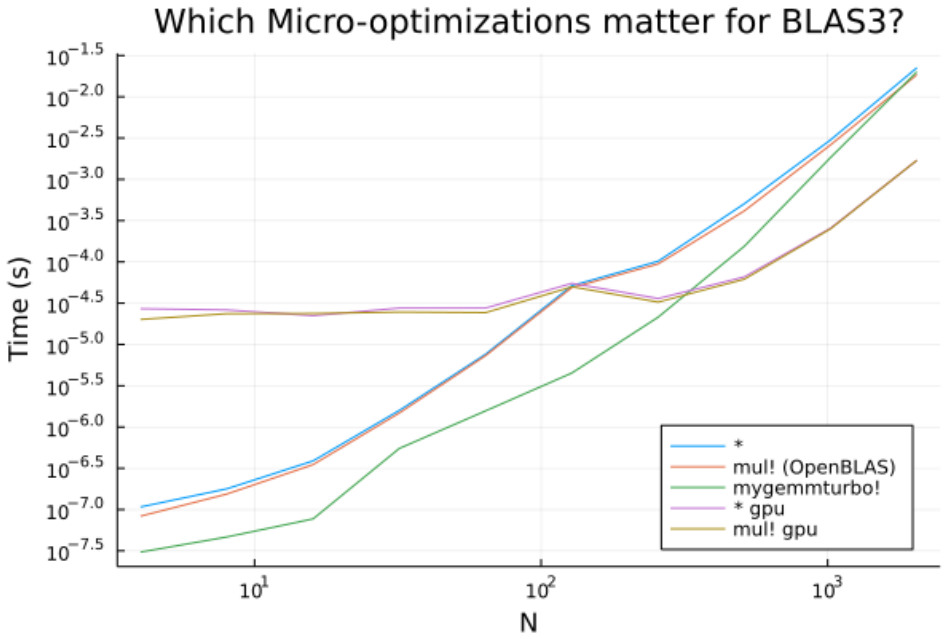
Scientific codes
O(n) and O(n^2)
operations

Mutation and
Memory management: 10x

Manual SIMD: 5x

…





Big O(n^3) operations?
Just use a GPU
Don’t worry about overhead
You’re fine!

Simplest code is ~3x from optimized

SimpleChains + StaticArray Neural ODEs

```
sc = SimpleChain(
                static(2),
                Activation(x -> x.^3),
                TurboDense{true}(tanh, static(50)),
                TurboDense{true}(identity, static(2))
            )

p_nn = SimpleChains.init_params(sc)

f(u,p,t) = sc(u,p)
```

his function is plugged into an ODE solver and the L2 loss is calculated from
ıe numerical solution and the NeuralODE output.

```
prob_nn = ODEProblem(f, u0, tspan)

function predict_neuralode(p)
    Array(solve(prob_nn,
Tsit5();p=p,saveat=tsteps,sensealg=QuadratureAdjoint(autojacvec=ZygoteVJP())))
end
```

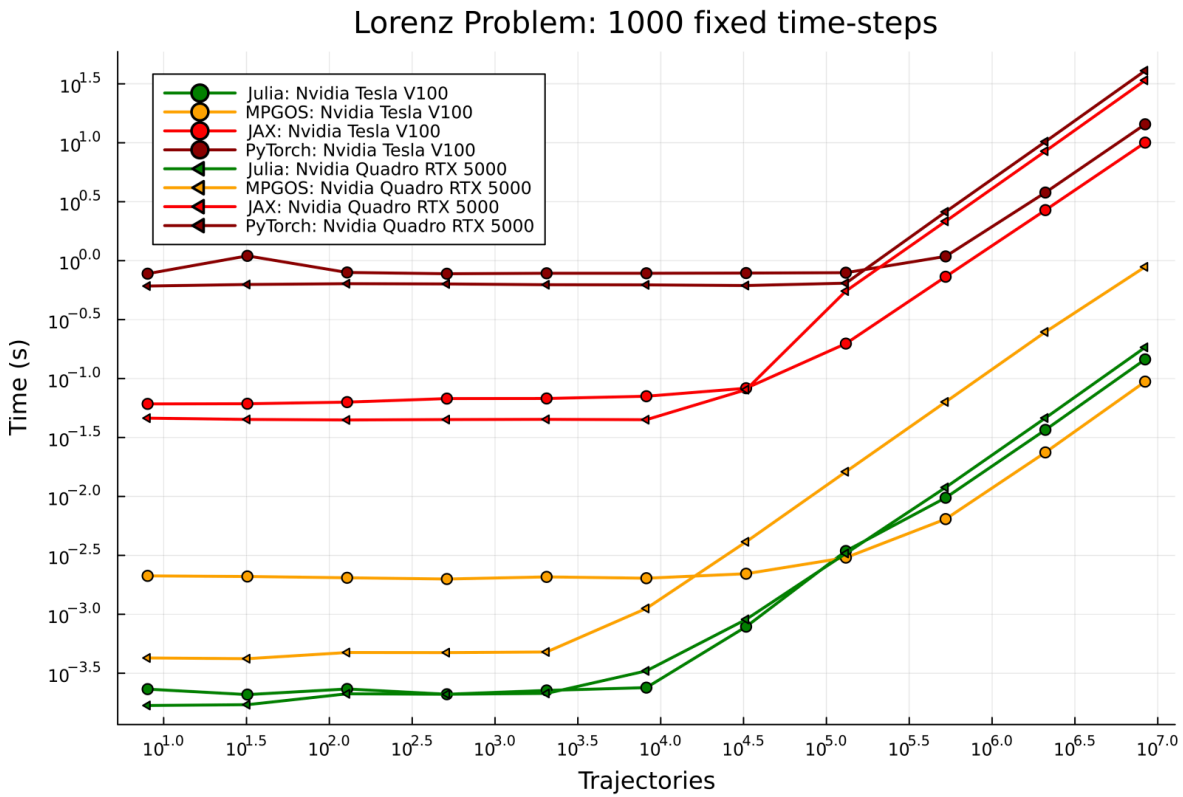
About a 5x improvement

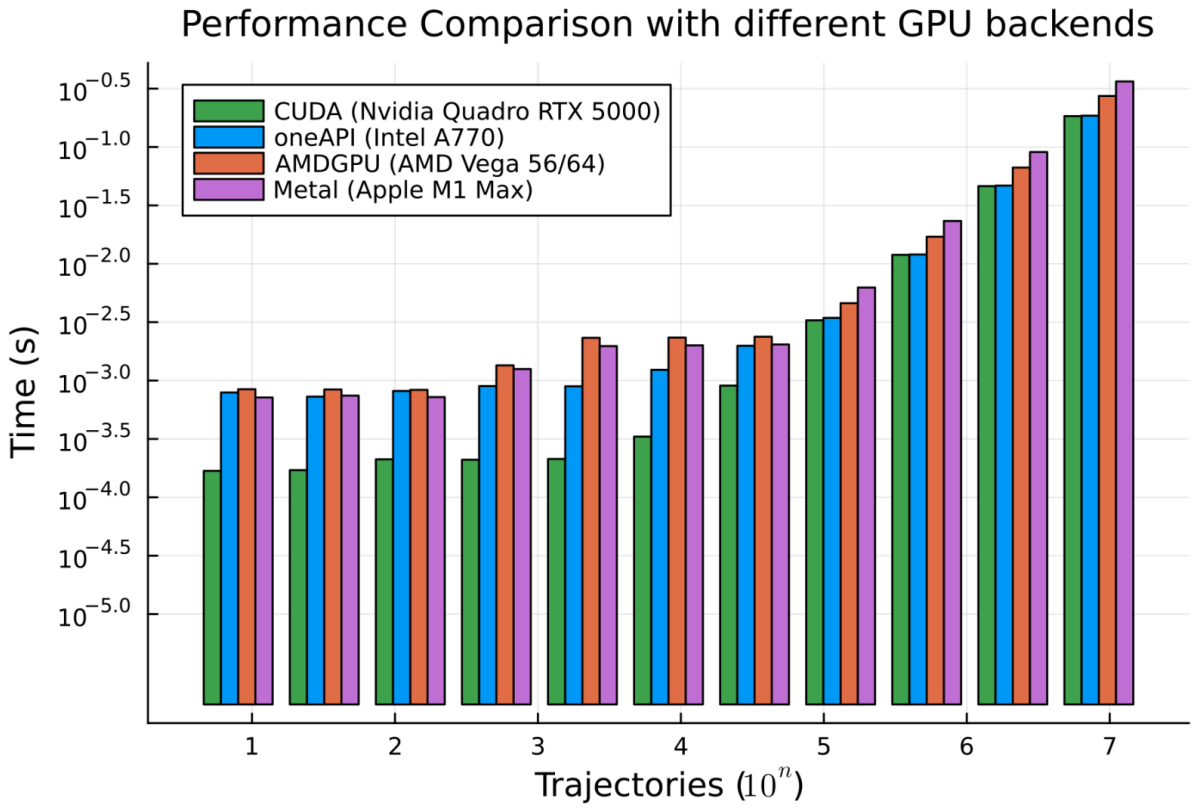
~1000x in a nonlinear mixed effects context

Caveat: Requires sufficiently small ODEs (<20)

And simulation processes are still improving rapidly.

New Parallelized GPU ODE Parallelism: 20x-100x Faster than Before





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

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*.

Foundation: Fast Differential Equation Solvers

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

DifferentialEquations.jl is generally:

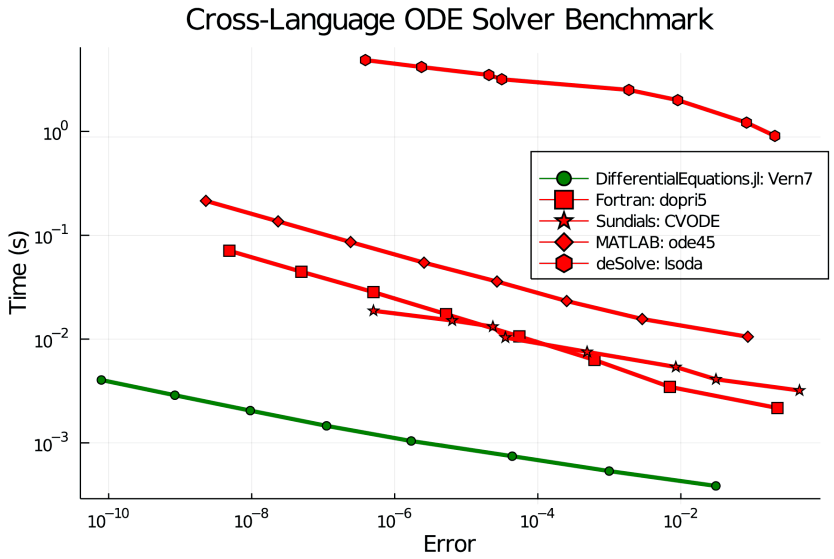
- 50x faster than SciPy
- 50x faster than MATLAB
- 100x faster than R's deSolve

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

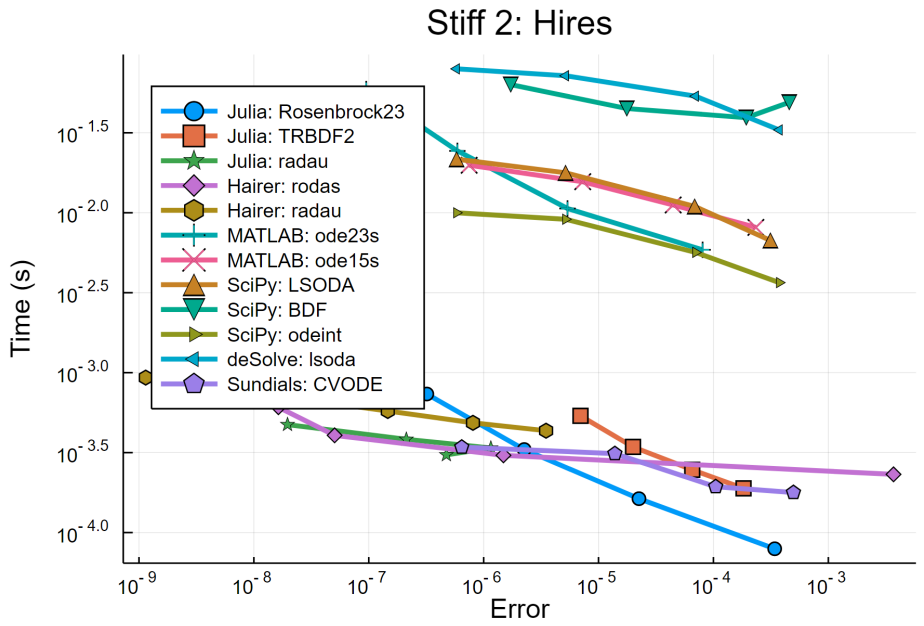
Rackauckas, Christopher, and Qing Nie. "Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia." Journal of Open Research Software 5.1 (2017).

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 Extrapolation Methods for Stiff ODEs: 2x-4x improvements

Elrod, Chris, Yingbo Ma, and Christopher Rackauckas. "Parallelizing Explicit and Implicit Extrapolation Methods for Ordinary Differential Equations." *arXiv preprint arXiv:2207.08135* (2022).

Accepted to HPEC 2022!

Caveat: only good for 5-200 ODEs with no implicit parallelism in f

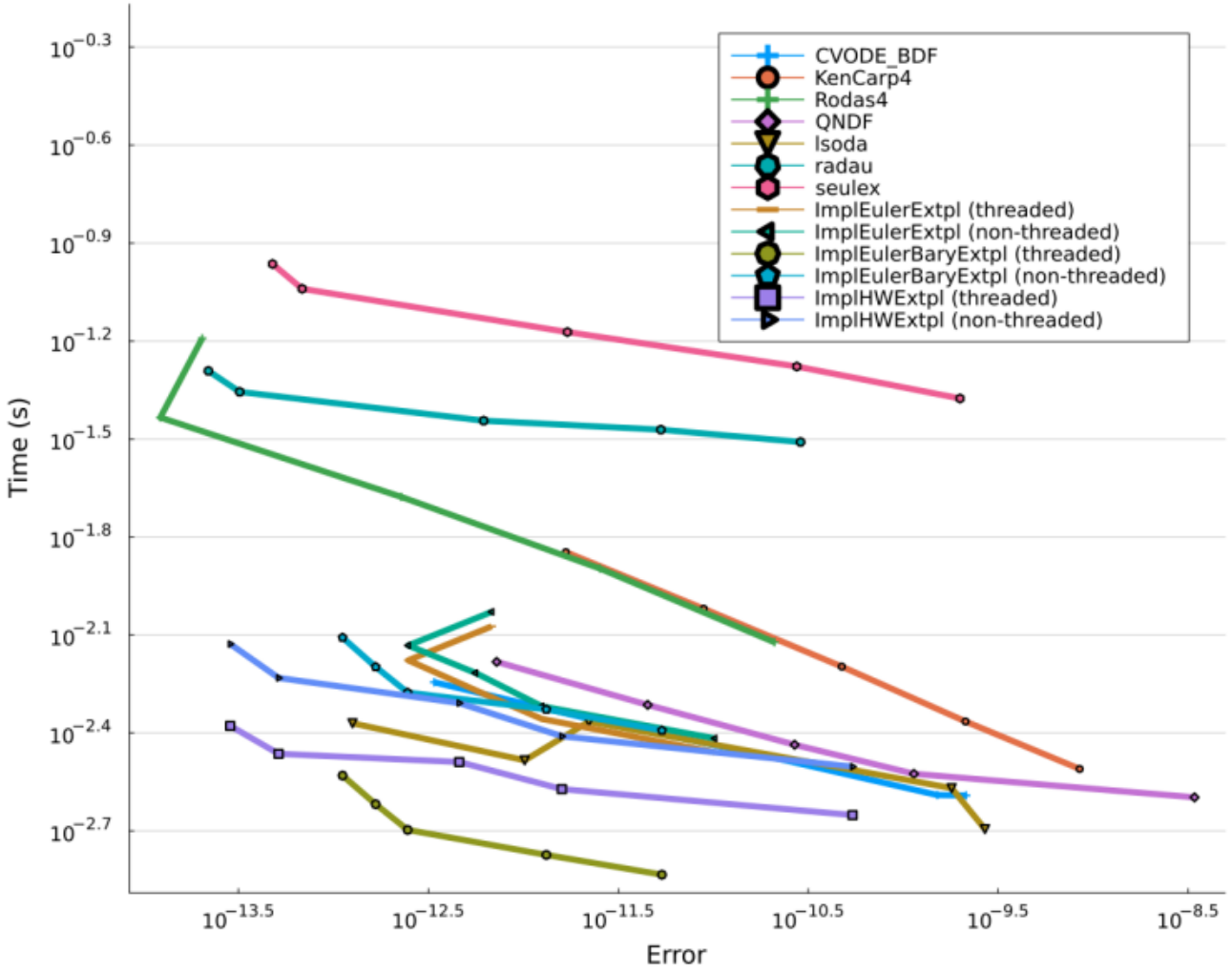


Fig. 6. Benchmark on QSP model with low tolerances [VI-B6].

SciML Open Source Software Organization
sciml.ai

- DifferentialEquations.jl: 2x-10x Sundials, Hairer, …
- DiffEqFlux.jl: adjoints outperforming Sundials and PETSc-TS
- ModelingToolkit.jl: 15,000x Simulink
- Catalyst.jl: >100x SimBiology, gillespy, Copasi
- DataDrivenDiffEq.jl: >10x pySindy
- NeuralPDE.jl: ~2x DeepXDE* (more optimizations to be done)
- NeuralOperators.jl: ~3x original papers (more optimizations required)
- ReservoirComputing.jl: 2x-10x pytorch-esn, ReservoirPy, PyRCN
- SimpleChains.jl: 5x PyTorch GPU with CPU, 10x Jax (small only!)
- DiffEqGPU.jl: Some wild GPU ODE solve speedups coming soon

And 100 more libraries to mention…

If you work in SciML and think optimized and maintained implementations of your method would be valuable, please let us know and we can add it to the queue.

Democratizing SciML via pedantic code optimization
Because we believe full-scale open benchmarks matter