Universal Differential Equations for Scientific Machine Learning

Christopher Rackauckas (crackauc@mit.edu)
Massachusetts Institute of Technology https://orcid.org/0000-0001-5850-0663

Yingbo Ma
Julia Computing

Julius Martensen
University of Bremen https://orcid.org/0000-0003-4143-3040

Collin Warner
Massachusetts Institute of Technology

Kirill Zubov
Saint Petersburg State University https://orcid.org/0000-0003-0441-449X

Rohit Supekar
Massachusetts Institute of Technology

Dominic Skinner
Massachusetts Institute of Technology https://orcid.org/0000-0002-2698-041X

Ali Ramadhan
Massachusetts Institute of Technology

Alan Edelman
Massachusetts Institute of Technology

Article

Keywords: machine learning, generalized approaches, modeling methodology

Posted Date: August 31st, 2020

DOI: https://doi.org/10.21203/rs.3.rs-55125/v1

Universal Differential Equations for Scientific Machine Learning

Christopher Rackauckas[a,b], Yingbo Ma[c], Julius Martensen[d], Collin Warner[a], Kirill Zubov[e], Rohit Supekar[a], Dominic Skinner[a], Ali Ramadhan[a], and Alan Edelman[a]

[a]Massachusetts Institute of Technology
[b]University of Maryland, Baltimore
[c]Julia Computing
[d]University of Bremen
[e]Saint Petersburg State University

August 26, 2020

Abstract

In the context of science, the well-known adage "a picture is worth a thousand words" might well be "a model is worth a thousand datasets." Scientific models, such as Newtonian physics or biological gene regulatory networks, are human-driven simplifications of complex phenomena that serve as surrogates for the countless experiments that validated the models. Recently, machine learning has been able to overcome the inaccuracies of approximate modeling by directly learning the entire set of nonlinear interactions from data. However, without any predetermined structure from the scientific basis behind the problem, machine learning approaches are flexible but data-expensive, requiring large databases of homogeneous labeled training data. A central challenge is reconciling data that is at odds with simplified models without requiring "big data".

In this work demonstrate how a mathematical object, which we denote universal differential equations (UDEs), can be utilized as a theoretical underpinning to a diverse array of problems in scientific machine learning to yield efficient algorithms and generalized approaches. The UDE model augments scientific models with machine-learnable structures for scientifically-based learning. We show how UDEs can be utilized to discover previously unknown governing equations, accurately extrapolate beyond the original data, and accelerate model simulation, all in a time and data-efficient manner. This advance is coupled with open-source software that allows for training UDEs which incorporate physical constraints, delayed interactions, implicitly-defined events, and intrinsic stochasticity in the model. Our examples show how a diverse set of computationally-difficult modeling issues across scientific disciplines, from automatically discovering biological mechanisms to accelerating the training of physics-informed neural networks and large-eddy simulations, can all be transformed into UDE training problems that are efficiently solved by a single software methodology.

Recent advances in machine learning have been dominated by deep learning which utilizes readily available "big data" to solve previously difficult problems such as image recognition [1, 2, 3] and natural language processing [4, 5, 6]. While some areas of science have begun to generate the large amounts of data required to train deep learning models, notably bioinformatics [7, 8, 9, 10, 11], in many areas the expense of scientific experiments has prohibited the effectiveness of these ground breaking techniques. In these domains, such as aerospace engineering, quantitative systems pharmacology, and macroeconomics, mechanistic models which synthesize the knowledge of the scientific literature are still predominantly deployed due to the inaccuracy of deep learning techniques with small training datasets. While these mechanistic models are constrained to be predictive by utilizing prior structural knowledge conglomerated throughout the scientific literature, the data-driven approach of machine learning can be more flexible and allows one to drop the simplifying assumptions required to derive theoretical models. The purpose of this work is to bridge the gap by merging the best of both methodologies while mitigating the deficiencies.

It has recently been shown to be advantageous to merge differential equations with machine learning. Physics-Informed Neural Networks (PINNs) utilize partial differential equations in the cost functions of neural networks to incorporate prior scientific knowledge [12]. While this has been shown to be a form of data-efficient machine learning for some scientific applications, the resulting model does not have the interpretability of mechanistic models. On the other end of the spectrum, machine learning practitioners have begun to make use of scientific structures as a modeling basis for machine learning. For example, neural ordinary differential equations are initial value problems of the form [13, 14, 15, 16]:

$$u' = \mathrm{NN}_\theta(u,t), \tag{1}$$

defined by a neural network NN_θ where θ are the weights. As an example, a neural network with two hidden layers can be written as

$$NN_\theta(u,t) = W_3\sigma_2(W_2\sigma_1(W_1[u;t]+b_1)+b_2)+b_3, \tag{2}$$

where $\theta = (W_1, W_2, W_3, b_1, b_2, b_3)$ where W_i are matrices and b_i are vectors of weights, and (σ_1, σ_2) are the choices of activation functions. Because the embedded function is a universal approximator (UA), it follows that NN_θ can learn to approximate any sufficiently regular differential equation. However, the resulting model is defined without direct incorporation of known mechanisms. The Universal Approximation Theorem (UAT) demonstrates that sufficiently large neural networks can approximate any nonlinear function with a finite set of parameters [17, 18, 19]. Our approach extends the previous data-driven neural ODE approaches to directly utilize mechanistic modeling simultaneously with UAs in order to allow for arbitrary data-driven model extensions. The objects of this semi-mechanistic approach, which we denote as Universal Differential Equations (UDEs) for universal approximators in differential equations, are differential equation models where part of the differential equation contains an embedded UA, such as a neural network, Chebyshev expansion, or a random forest.

As a motivating example, the universal ordinary differential equation (UODE):

$$u' = f(u, t, U_\theta(u, t)), \tag{3}$$

denotes a known mechanistic model form f with missing terms defined by some UA U_θ. While the utility of this object has been seen before in optimal control [20] and model augmentation [21], here we demonstrate that this object can be used to solve a greatly expanded scope of problems and is thus able to be the basis of much research in scientific machine learning, from accelerating models to being a stepping stone in symbolic algorithms. We demonstrate generalizations to incorporate process noise, delayed interactions, and physics-based constraints are given by embedding UAs into stochastic, delay, and differential-algebraic equations respectively. As a fundamental object underlying so many algorithms, it then becomes essential to be able to efficiently train UDEs in any context in which they arise. In Section 1 we describe our methodology and software implementation for efficiently training UDEs of any of these forms in a way that covers stiffness, nonlinear algebraic constraints, stochasticity, delays, parallelism, and more. We then demonstrate that the following abilities within the UDE framework:

- In Section 2.1 we recover governing equations from much lesser data than prior methods and demonstrate the ability to accurately extrapolate from a short time series.
- In Section 2.2 we demonstrate the ability to utilize arbitrary conservation laws as prior knowledge in the discovery of dynamical systems.
- In Section 2.3 we discover the differential operator and nonlinear reaction term of a biological partial differential equation (PDE) from spatiotemporal data, demonstrating the interpretability of trained UDEs.
- In Section 3 We derive an adaptive method for automated solving of a 100-dimensional nonlinear Hamilton-Jacobi-Bellman PDE, the first adaptive method for this class of problems that the authors are aware of.
- In Section 4.1 we automate the discovery of fast, accurate, and physically-consistent surrogates to accelerate a large-eddy simulation commonly used in the voxels of a climate simulation.
- In Section 4.2 we approximate closure relations in viscoelastic fluids to accelerate the simulation of a system of 6 differential-algebraic equations by 2x, showing that this methodology is also applicable to small-scale problems.

- In Section 4.3 we demonstrate that discrete physics-informed neural networks fall into a subclass of universal ODEs and extend previous methods directly through this formalism.

1 Efficient Training of Universal Differential Equations via Differentiable Programming

Training a UDE amounts to minimizing a cost function $C(\theta)$ defined on the current solution $u_\theta(t)$, the current solution to the differential equation with respect to the choice of parameters θ. One choice is the Euclidean distance $C(\theta) = \sum_i \|u_\theta(t_i) - d_i\|$ at discrete data points (t_i, d_i). When optimized with local derivative-based methods, such as stochastic gradient decent, ADAM [22], or L-BFGS [23], this requires the calculation of $\frac{dC}{d\theta}$ which by the chain rule amounts to calculating $\frac{du}{d\theta}$. Thus the problem of efficiently training a UDE reduces to calculating gradients of the differential equation solution with respect to parameters.

In certain special cases there exist efficient methods for calculating these gradients called adjoints [24, 25, 26, 27, 28]. The asymptotic computational cost of these methods does not grow multiplicatively with the number of state variables and parameters like numerical or forward sensitivity approaches, and thus it has been shown empirically that adjoint methods are more efficient on large parameter models [29, 30]. However, given the number of different families of UDE models we wish to train, we generalize to a differentiable programming framework with reverse-mode accumulation in order to allow for deriving on-the-fly approximations for the wide range of possible differential equation types.

Given a function $f(x) = y$, the pullback at x is the function:

$$B_f^x(y) = y^T f'(x), \tag{4}$$

where $f'(x)$ is the Jacobian J. We note that $B_f^x(1) = (\nabla f)^T$ for a function f producing a scalar output, meaning the pullback of a cost function computes the gradient. A general computer program can be written as the composition of discrete steps:

$$f = f^L \circ f^{L-1} \circ \ldots \circ f^1, \tag{5}$$

and thus the vector-Jacobian product can be decomposed:

$$v^T J = (\ldots((v^T J_L) J_{L-1}) \ldots) J_1, \tag{6}$$

which allows for recursively decomposing a the pullback to a primitively known set of $\mathcal{B}_{f^i}^x$:

$$\mathcal{B}_f^x(A) = \mathcal{B}_{f^1}^x\left(\ldots\left(\mathcal{B}_{f^{L-1}}^{x_{L-2}}\left(\mathcal{B}_{f^L}^{x_{L-1}}(A)\right)\right)\ldots\right), \tag{7}$$

where $x_i = \left(f^i \circ f^{i-1} \circ \ldots \circ f^1\right)(x)$. Implementations of code generation for the backwards pass of an arbitrary program in a dynamic programming language can vary. For example, building a list of function compositions (a tape) is provided by libraries such as Tracker.jl [31] and PyTorch [32], while other libraries perform direct generation of backward pass source code such as Zygote.jl [33], TAF [34], and Tapenade [35].

The open-source differential equation solvers of DifferentialEquations.jl [36] were developed in a manner such that all steps of the programs have a well-defined pullback when using a Julia-based backwards pass generation system. Our software allows for automatic differentiation to be utilized over differential equation solves without any modification to the user code. This enables the simulation software already written with DifferentialEquations.jl, including large software infrastructures such as the MIT-CalTech CLiMA climate modeling system [37] and the QuantumOptics.jl simulation framework [38], to be compatible with all of the techniques mentioned in the rest of the paper. Thus while we detail our results in isolation from these larger simulation frameworks, the UDE methodology can be readily used in full-scale simulation packages which are already built on top of the Julia SciML ecosystem.

The full set of adjoint options, which includes continuous adjoint methods and pure reverse-mode AD approaches, is described in Supplement S1. Methods via solving ODEs in reverse [16] are the common adjoint utilized in neural ODE software such as torchdiffeq and are O(1) in memory, but are known to be unstable under certain conditions such as on stiff equations [39]. Checkpointed interpolation adjoints [27] and continuous quadrature approaches are available which do not require stable reversibility of the ODEs while retaining a relatively low-memory implementation via checkpointing (in particular Section 2.2 and 4.1 are noted as a cases which are not stable under the reversed adjoint but stable under the checkpointing adjoint approach). These adjoint methods fall under the continuous optimize-then-discretize approach. Through the differentiable programming integration, discrete adjoint sensitivity analysis [40, 41] is implemented through both tape-based reverse-mode [42] and source-to-source translation [33], with computational trade-offs between the two approaches. The former can be faster on scalarized heterogeneous differential equations while the latter is more optimized for homogeneous vectorized functions calls like are demonstrated in neural networks and discretizations of partial differential equations. Full discretize-then-optimize is implemented using this package by utilizing the step-wise integrator interface in conjunction with these discrete adjoints of the steps. Continuous and discrete forward mode sensitivity analysis approaches are also provided and optimized for equations with smaller numbers of parameters.

Previous research has shown that the discrete adjoint approach is more stable than continuous adjoints in some cases [43, 39, 44, 45, 46, 47] while continuous adjoints have been demonstrated to be more stable in others [48, 45] and can reduce spurious oscillations [49, 50, 51]. This trade-off between discrete and continuous adjoint approaches has been demonstrated on some equations as a trade-off between stability and computational efficiency [52, 53, 54, 55, 56, 57, 58, 59, 60]. Care has to be taken as the stability of an adjoint approach can be dependent on the chosen discretization method [61, 62, 63, 64, 65], and our software contribution helps researchers switch between all of these optimization approaches in combination with hundreds of differential equation solver methods with a single line of code change.

As described in Supplement S1.1, these adjoints utilize reverse-mode automatic differentiation for vector transposed Jacobian products within the adjoint definitions to reduce the computational complexity while supporting advanced features like constraint and conservation equations. In addition, the module DiffEqFlux.jl handles compatibility with the Flux.jl neural network library so that these vector Jacobian products are automatically replaced with efficient pullback implementations for embedded deep neural networks (also known as backpropogation) wherever neural networks are encountered in the right hand side of any differential equation definitions. This allows for common deep architectures, such as convolutional neural networks and recurrent neural networks, to be efficiently used as the basis for a UDE without any Jacobians being calculated in the full adjoint and without requiring any intervention from users.

Using this approach, the solvers are capable of building efficient gradient calculations for training ML-embedded UDEs of the classes:

- Universal Ordinary Differential Equations (UODEs)
- Universal Stochastic Differential Equations (USDEs), or universal differential equations with continuous process noise
- Universal Delay Differential Equations (UDDEs), or universal differential equations with delayed interactions
- Universal Differential-Algebraic Equations (UDAEs), or universal differential equations with constraint equations and conservation laws
- Universal Boundary Value Problems (UBVPs), or universal differential equations with final time point constraints
- Universal Partial Differential Equations (UPDEs)
- Universal Hybrid (Event-Driven) Differential Equations

as well as the combinations, such as stochastic delay differential equations, jump diffusions, and stochastic partial differential equations. A combination of over 300 solver methods cover the efficient training of stiff and non-stiff versions of each of these equations, with support for adaptivity, high-order, automatic stiffness detection, sparse differentiation with automatic sparsity detection and coloring [66], Newton-Krylov implicit handling, GPU compatibility, and multinode parallelism via MPI compatibility. Thus together, semi-mechanistic UDEs of any form can embed machine learning models and be trained using this open-source library with the most effective differential equation solvers for that class of equations.

Feature	Stiff	DAEs	SDEs	DDEs	Stabilized	DtO	GPU	Dist	MT	Sparse
SciML	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
torchdiffeq	0	0	0	0	0	✓	✓	0	0	0
torchsde	0	0	✓	0	0	0	✓	0	0	0
tfdiffeq	0	0	0	0	0	0	✓	0	0	0

Table 1: Feature comparison of ML-augmented differential equation libraries. First first column corresponds to support for stiff ODEs, then DAEs, SDEs, DDEs, stabilized non-reversing adjoints, discretize-then-optimize methods, distributed computing, and multithreading. Sparse refers to automated sparsity handling in Jacobian calculations of implicit methods.

1.1 Features and Performance

We assessed the viability of alternative differential equation libraries for universal differential equation workflows by comparing the features and performance of the given libraries. Table 1 demonstrates that the Julia SciML ecosystem is the only differential equation solver library with deep learning integration that supports stiff ODEs, DAEs, DDEs, stabilized adjoints, distributed and multithreaded computation. We note the importance of the stabilized adjoints in Section 4.1 as many PDE discretizations with upwinding exhibit unconditional instability when reversed, and thus this is a crucial feature when training embedded neural networks in many PDE applications. Table 2 demonstrates that the SciML ecosystem exhibits more than an order of magnitude performance when solving ODEs against torchdiffeq of up to systems of 1 million equations. Because the adjoint calculation itself is a differential equation, this also corresponds to increased training times on scientific models. To reinforce this result, Supplement S2 demonstrates a 100x performance difference over torchdiffeq when training the spiral neural ODE from [16, 43]. We note that the author of the tfdiffeq library has previous concluded “speed is almost the same as the PyTorch (torchdiffeq) codebase (±2%)”. Additionally, Supplement S2 demonstrates a 1,600x performance advantage for the SciML ecosystem over torchsde using the geometric Brownian motion example from the torchsde documentation [67]. Given the computational burden, the mix of stiffness, and non-reversibility of the examples which follow in this paper, these results demonstrate that the SciML ecosystem is the first deep learning integrated differential equation software ecosystem that can train all of the equations necessary for the results of this paper. Note that this does not infer that our solvers will demonstrate more than an order of magnitude performance difference on all equations, for example very non-stiff ODEs dominated by large dense matrix multiplications like in image classification neural ODEs, but it does demonstrate that on the equations generally derived from scientific models (ODEs derived from PDE discretizations, heterogeneous differential equation systems, and neural networks in sufficiently small systems) that an order of magnitude or more performance difference can exist.

# of ODEs	3	28	768	3,072	12,288	49,152	196,608	786,432
SciML	1.0x	1.0x	1.0x	1.0x	1.0x	1.0x	1.0x	1.0x
SciML DP5	1.0x	1.9x	2.9x	2.9x	2.9x	2.9x	3.4x	2.6x
torchdiffeq dopri5	5,850x	1700x	420x	280x	120x	31x	41x	38x
torchdiffeq adams	7,600x	1100x	710x	490x	170x	44x	47x	43x

Table 2: Relative time to solve for ML-augmented differential equation libraries (smaller is better). Standard non-stiff solver benchmarks from representative scientific systems were taken from [68] as described in Supplement S2. SciML stands for the optimal method choice out of the 300+ from the SciML, which for the first is DP5, for the second is VCABM, and for the rest is ROCK4.

2 Knowledge-Enhanced Model Reconstruction of Biological Models

Automatic reconstruction of models from observable data has been extensively studied. Many methods produce non-symbolic representations by learning functional representations [69, 70] or through dynamic mode decomposition (DMD, eDMD) [71, 72, 73, 74]. Symbolic reconstruction of equations has utilized symbolic regressions which require a prechosen basis [75, 76], or evolutionary methods to grow a basis [77, 78]. However, a common thread throughout much of the literature is that added domain knowledge constrains the problem to allow for more data-efficient reconstruction [79, 80]. Here we detail how a UA embedded workflow can augment existing symbolic regression frameworks to allow for reconstruction from partially known models in a more data-efficient manner.

2.1 Automated Identification of Nonlinear Interactions with Universal Ordinary Differential Equations

The SInDy algorithm [81, 82, 83] finds a sparse basis $\mathbf{\Xi}$ over a given candidate library $\mathbf{\Theta}$ minimizing the objective function $\left\|\dot{\mathbf{X}} - \mathbf{\Theta}\mathbf{\Xi}\right\|_2 + \lambda \left\|\mathbf{\Xi}\right\|_1$ using data for $\dot{\mathbf{X}}$ generated by interpolating the trajectory data $\mathbf{X}$. Here we describe a UDE approach to extend SInDy in a way that embeds prior structural knowledge.

As a motivating example, take the Lotka-Volterra system:

$$\begin{aligned} \dot{x} &= \alpha x - \beta x y, \\ \dot{y} &= \gamma x y - \delta y. \end{aligned} \tag{8}$$

Assume that a scientist has a short time series from this system but knows the birth rate of the prey x and the death rate of the predator y. With only this information, a scientist can propose the knowledge-based UODE as:

$$\begin{aligned} \dot{x} &= \alpha x + U_1(x, y), \\ \dot{y} &= -\delta y + U_2(x, y), \end{aligned} \tag{9}$$

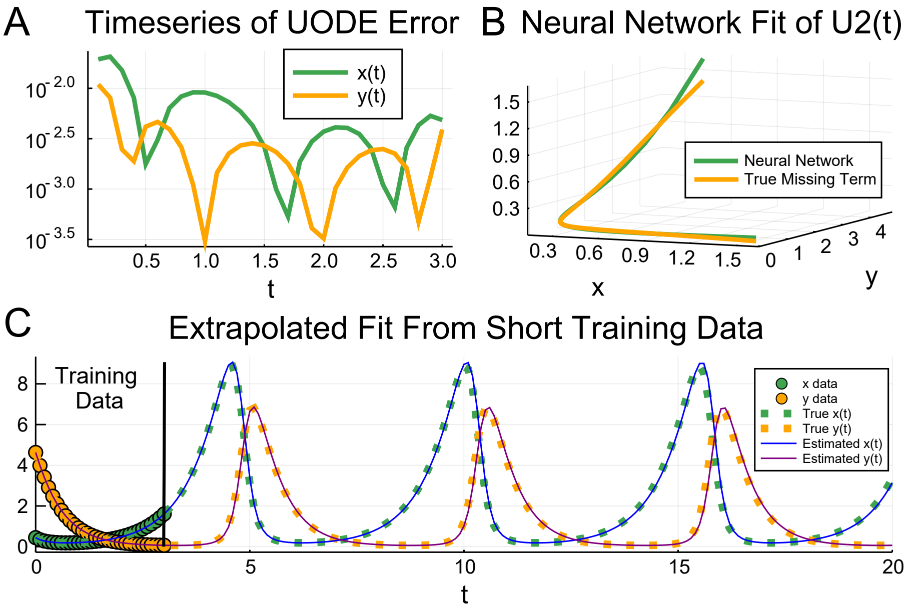

Figure 1: Automated Lotka-Volterra equation discovery with UODE-enhanced SInDy. (A) The error in the trained UODE against $x(t)$ and $y(t)$ in green and yellow respectively. (B) The measured values of the missing term $U_2(x, y)$ throughout the time series, with the neural network approximate in green and the true value γxy in yellow. (C) The extrapolation of the knowledge-enhanced SInDy fit series. The green and yellow dots show the data that was used to fit the UODE, and the dots show the true solution of the Lotka-Volterra Equations 8 beyond the training data. The blue and purple lines show the extrapolated solution how the UODE-enhanced SInDy recovered equations.

which is a system of ordinary differential equations that incorporates the known structure but leaves room for learning unknown interactions between the the predator and prey populations. Learning the unknown interactions corresponds training the UA $U : \mathbb{R}^2 \to \mathbb{R}^2$ in this UODE.

While the SInDy method normally approximates derivatives using a spline over the data points or similar numerical techniques, here we have $U_\theta(x, y)$ as an estimator of the derivative for only the missing terms of the model and we can perform a sparse regression on samples from the trained $U_\theta(x, y)$ to reconstruct only the unknown interaction equations. As described in Supplement S3.1, we trained $U_\theta(x, y)$ as a neural network against the simulated data for $t \in [0, 3]$ and utilized a sparse regression techniques [81, 84, 85] on the neural network outputs to reconstruct the missing dynamical equations. Using a 10-dimensional polynomial basis extended with trigonometric functions, the sparse regression yields 0 for all terms except for the missing quadratic terms, directly learning the original equations in an interpretable form. Even though the original data did not contain a full period of the cyclic solution, the resulting fit is then able to accurately extrapolate from the short time series data as shown in Figure 1. Supplement S3.1 further demonstrates the robustness of the discovery approach to noise in the data. Likewise, when attempting to learn full ODE with the original SInDy approach on the same trained data with the analytical derivative values, we were unable to recover the exact original equations from the sparse regression, indicating that the knowledge-enhanced approach increases the robustness equation discovery.

We note that collaborators using the preprint of this manuscript have successfully demonstrated the ability to construct UODE models which improve the prediction of Li-ion battery performance [86] and for automated discovery of droplet physics directly from imaging data, effectively replicating the theoretical results of a one and a half year study with a UODE discovery process which trains in less than an hour [87].

2.2 Incorporating Prior Knowledge of Conservation Equations

The extra features of the SciML ecosystem can be utilized to encode more information into the model. For example, when attempting to discover a biological chemical reaction network or a chemical combustion network, one may only have prior knowledge of the conservation laws between the constituent substrates. As a demonstration, in the Robertson equation

$$\frac{dy_1}{dt} = -0.04y_1 + 10^4 y_2 y_3 \tag{10}$$

$$\frac{dy_2}{dt} = 0.04y_1 - 10^4 y_2 y_3 - 3 * 10^7 y_2^2 \tag{11}$$

$$1 = y_1 + y_2 + y_3 \tag{12}$$

one might only have prior knowledge of the conservation equation $1 = y_1 + y_2 + y_3$. In this case, a universal DAE of the form:

$$\frac{d[y_1, y_2]}{dt} = U_\theta(y_1, y_2, y_3) \tag{13}$$

$$1 = y_1 + y_2 + y_3 \tag{14}$$

can be utilized to encode this prior knowledge. This can then be trained by utilizing a singular mass matrix in the form $Mu' = f(u, p, t)$. Supplement S1's derivation of the adjoint method describes a new initialization scheme for index-1 DAEs in mass matrix form which directly solves a linear system for new consistent algebraic variables in the adjoint pass without requiring the approximate nonlinear iterations of [88], thus further demonstrating the efficiency and accuracy of the SciML software's methods for UDE workflows. Supplement S3.2 demonstrates the ability to learn this system utilizing the SciML tools through this universal DAE approach.

2.3 Reconstruction of Spatial Dynamics with Universal Partial Differential Equations

To demonstrate discovery of spatiotemporal equations directly from data, we consider data generated from the one-dimensional Fisher-KPP (Kolmogorov–Petrovsky–Piskunov) PDE [89]:

$$\frac{\partial \rho}{\partial t} = r\rho(1-\rho) + D\frac{\partial^2 \rho}{\partial x^2}, \tag{15}$$

with $x \in [0,1]$, $t \in [0,T]$, and periodic boundary condition $\rho(0,t) = \rho(1,t)$. Here ρ represents population density of a species, r is the local growth rate and D is the diffusion coefficient. Such reaction-diffusion equations appear in diverse physical, chemical and biological problems [90]. To learn the generated data, we define the UPDE:

$$\rho_t = \mathrm{NN}_\theta(\rho) + \hat{D}\,\mathrm{CNN}(\rho), \tag{16}$$

where NN_θ is a neural network representing the local growth term. The derivative operator is approximated as a convolutional neural network CNN, a learnable arbitrary representation of a stencil while treating the coefficient $\hat{D}$ as an unknown. We encode in the loss function extra constraints to ensure the learned equation is physically realizable, i.e. the derivative stencil must be conservative (the coefficients sum to zero), as described in Supplement S4. Figure 2 shows the result of training the UPDE against the simulated data, which recovers the canonical $[1,-2,1]$ stencil of the one-dimensional Laplacian and the diffusion constant while simultaneously finding a neural representation of the unknown quadratic growth term. We note that the differentiable programming integration in conjunction with the Flux.jl deep learning framework allows for the adjoints to automatically utilize efficient backpropogation of the embedded convolutional neural networks and automatically utilizes the fast kernels provided by cudnn when trained using GPUs.

3 Computationally-Efficient Solving of High-Dimensional Partial Differential Equations

It is impractical to solve high dimensional PDEs with mesh-based techniques since the number of mesh points scales exponentially with the number of dimensions. Given this difficulty, mesh-free methods based on universal approximators such as neural networks have been constructed to allow for direct solving of high dimensional PDEs [91, 92]. Recently, methods based on transforming partial differential equations into alternative forms, such as backwards stochastic differential equations (BSDEs), which are then approximated by neural networks have been shown to be highly efficient on important equations such as the nonlinear Black-Scholes and Hamilton-Jacobi-Bellman (HJB) equations [93, 94, 95, 96].

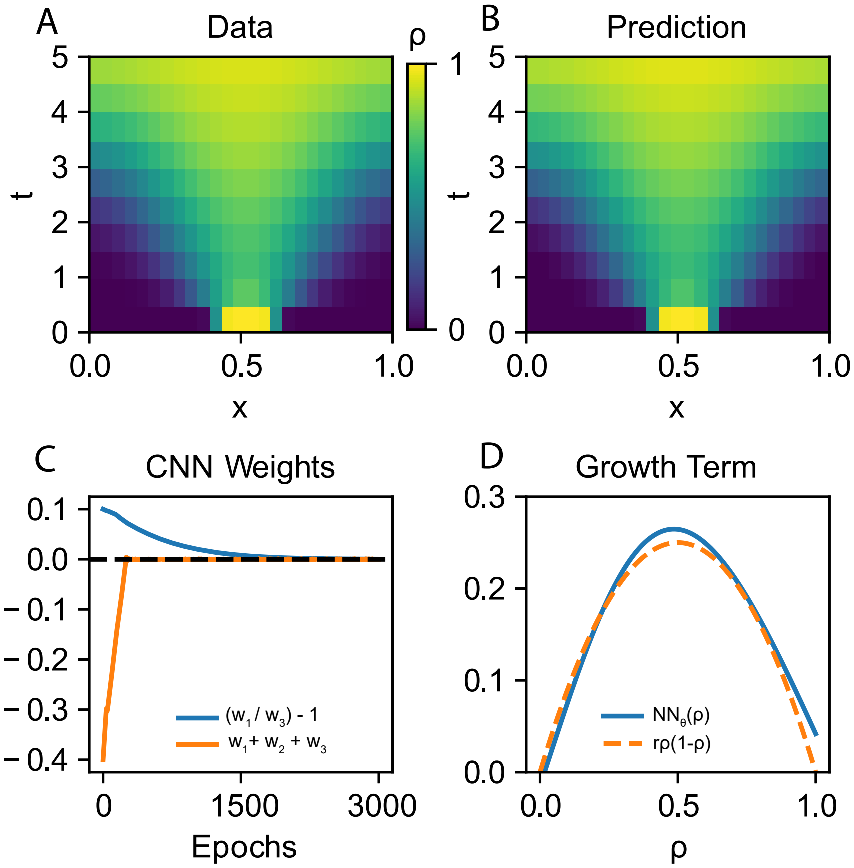

Figure 2: Recovery of the UPDE for the Fisher-KPP equation. (A) Training data and (B) prediction of the UPDE for $\rho(x, t)$. (C) Curves for the weights of the CNN filter $[w_1, w_2, w_3]$ indicate the recovery of the $[1, -2, 1]$ stencil for the 1-dimensional Laplacian. (D) Comparison of the learned (blue) and the true growth term (orange) showcases the learned parabolic form of the missing nonlinear equation.

Here we will showcase how one of these methods, a deep BSDE method for semilinear parabolic equations [94], can be reinterpreted as a universal stochastic differential equation (USDE) to generalize the method and allow for enhancements like adaptivity, higher order integration for increased efficiency, and handling of stiff driving equations through the SciML software.

Consider the class of semilinear parabolic PDEs with a finite time span $t \in [0, T]$ and d-dimensional space $x \in \mathbb{R}^d$ that have the form:

$$\begin{aligned}\frac{\partial u}{\partial t}(t,x) + \frac{1}{2}\operatorname{Tr}\left(\sigma\sigma^T(t,x)\left(\operatorname{Hess}_x u\right)(t,x)\right)& \\ + \nabla u(t,x)\cdot\mu(t,x)& \\ + f\left(t,x,u(t,x),\sigma^T(t,x)\nabla u(t,x)\right) = 0,&\end{aligned} \tag{17}$$

with a terminal condition $u(T, x) = g(x)$. Supplement S5 describes how this PDE can be solved by approximating by approximating the FBSDE:

$$\begin{aligned}dX_t &= \mu(t,X_t)dt + \sigma(t,X_t)dW_t,\\ dU_t &= f(t,X_t,U_t,U^1_{\theta_1}(t,X_t))dt + \left[U^1_{\theta_1}(t,X_t)\right]^T dW_t,\end{aligned} \tag{18}$$

where $U^1_{\theta_1}$ and $U^2_{\theta_2}$ are UAs and the loss function is given by the requiring that the terminating condition $g(X_T) = u(X_T, W_T)$ is satisfied.

3.1 Adaptive Solution of High-Dimensional Hamilton-Jacobi-Bellman Equations

A fixed time step Euler-Maryumana discretization of this USDE gives rise to the deep BSDE method [94]. However, this form as a USDE generalizes the approach in a way that makes all of the methodologies of our USDE training library readily available, such as higher order methods, adaptivity, and implicit methods for stiff SDEs. As a motivating example, consider the classical linear-quadratic Gaussian (LQG) control problem in 100 dimensions:

$$dX_t = 2\sqrt{\lambda}c_t dt + \sqrt{2}dW_t, \tag{19}$$

with $t \in [0, T]$, $X_0 = x$, and with a cost function $C(c_t) = \mathbb{E}\left[\int_0^T \|c_t\|^2 dt + g(X_t)\right]$ where X_t is the state we wish to control, λ is the strength of the control, and c_t is the control process. Minimizing the control corresponds to solving the 100-dimensional HJB equation:

$$\frac{\partial u}{\partial t} + \nabla^2 u - \lambda\|\nabla u\|^2 = 0 \tag{20}$$

We solve the PDE by training the USDE using an adaptive Euler-Maruyama method [97] as described in Supplement S5. Supplementary Figure 2 showcases that this methodology accurately solves the equations, effectively extending recent algorithmic advancements to adaptive forms simply be reinterpreting the equation as a USDE. While classical methods would require an amount of memory that is exponential in the number of dimensions making classical adaptively approaches infeasible, this approach is the first the authors are aware of to generalize the high order, adaptive, highly stable software tooling to the high-dimensional PDE setting.

4 Accelerated Scientific Simulation with Automatically Constructed Closure Relations

4.1 Automated Discovery of Large-Eddy Model Parameterizations

As an example of directly accelerating existing scientific workflows, we focus on the Boussinesq equations [98]. The Boussinesq equations are a system of 3+1-dimensional partial differential equations acquired through simplifying assumptions on the incompressible Navier-Stokes equations, represented by the system:

$$\begin{aligned} \nabla \cdot \mathbf{u} &= 0, \\ \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} &= -\nabla p + \nu \nabla^2 \mathbf{u} + b\hat{z}, \\ \frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T &= \kappa \nabla^2 T, \end{aligned} \tag{21}$$

where $\mathbf{u} = (u, v, w)$ is the fluid velocity, p is the kinematic pressure, ν is the kinematic viscosity, κ is the thermal diffusivity, T is the temperature, and b is the fluid buoyancy. We assume that density and temperature are related by a linear equation of state so that the buoyancy b is only a function $b = \alpha g T$ where α is the thermal expansion coefficient and g is the acceleration due to gravity.

This system is commonly used in climate modeling, especially as the voxels for modeling the ocean [99, 100, 101, 98] in a multi-scale model that approximates these equations by averaging out the horizontal dynamics $\overline{T}(z,t) = \iint T(x,y,z,t)\,dx\,dy$ in individual boxes. The resulting approximation is a local advection-diffusion equation describing the evolution of the horizontally-averaged temperature $\overline{T}$:

$$\frac{\partial \overline{T}}{\partial t} + \frac{\partial \overline{wT}}{\partial z} = \kappa \frac{\partial^2 \overline{T}}{\partial z^2}. \tag{22}$$

This one-dimensional approximating system is not closed since $\overline{wT}$ is unknown. Common practice closes the system by manually determining an approximating $\overline{wT}$ from ad-hoc models, physical reasoning, and scaling laws. However, we can utilize a UDE-automated approach to learn such an approximation from data. Let

$$\overline{wT} = U_\theta\left(P, \overline{T}, \frac{\partial \overline{T}}{\partial z}\right) \tag{23}$$

where P are the physical parameters of the Boussinesq equation at different regimes of the ocean, such as the amount of surface heating or the strength of the surface winds [102]. We can accurately capture the non-locality of the convection in this term by making the UDE a high-dimensional neural network. Using data from horizontal average temperatures $\overline{T}$ with known physical parameters P, we can directly reconstruct a nonlinear P-dependent parameterization by training a universal diffusion-advection partial differential equation. Supplementary Figure 3 demonstrates the accuracy of the approach using a deep UPDE with high order stabilized-explicit Runge-Kutta (ROCK) methods where the fitting is described in Supplement S6. To contrast the trained UPDE, we directly simulated the 3D Boussinesq equations under similar physical conditions and demonstrated that the neural parameterization results in around a 15,000x acceleration. This demonstrates that physical-dependent parameterizations for acceleration can be directly learned from data utilizing the previous knowledge of the averaging approximation and mixed with a data-driven discovery approach.

4.2 Data-Driven Nonlinear Closure Relations for Model Reduction in Non-Newtonian Viscoelastic Fluids

All continuum materials satisfy conservation equations for mass and momentum. The difference between an elastic solid and a viscous fluid comes down to the constitutive law relating the stresses and strains. In a one-dimensional system, an elastic solid satisfies $\sigma = G\gamma$, with stress σ, strain γ, and elastic modulus G, whereas a viscous fluid satisfies $\sigma = \eta\dot{\gamma}$, with viscosity η and strain rate $\dot{\gamma}$. Non-Newtonian fluids have more complex constitutive laws, for instance when stress depends on the history of deformation,

$$\sigma(t) = \int_{-\infty}^{t} G(t-s)F(\dot{\gamma}(s))\,\mathrm{d}s, \tag{24}$$

alternatively expressed in the instantaneous form [103]:

$$\begin{aligned}
\sigma(t) &= \phi_1(t),\\
\frac{\mathrm{d}\phi_1}{\mathrm{d}t} &= G(0)F(\dot{\gamma}) + \phi_2,\\
\frac{\mathrm{d}\phi_2}{\mathrm{d}t} &= \frac{\mathrm{d}G(0)}{\mathrm{d}t}F(\dot{\gamma}) + \phi_3,\\
&\vdots
\end{aligned} \tag{25}$$

where the history is stored in ϕ_i. To become computationally feasible, the expansion is truncated, often in an ad-hoc manner, e.g. $\phi_n = \phi_{n+1} = \cdots = 0$, for some n. Only with a simple choice of $G(t)$ does an exact closure condition exist, e.g. the Oldroyd-B model. For a fully nonlinear approximation, we train a UODE according to the details in Supplement S7 to learn a closure relation:

$$\sigma(t) = U_0(\dot{\gamma}, \phi_1, \ldots, \phi_N), \tag{26}$$

$$\frac{d\phi_i}{dt} = U_i(\dot{\gamma}, \phi_1, \ldots, \phi_N), \quad \text{for } i = 1 \text{ to } N \tag{27}$$

from the numerical solution of the FENE-P equations, a fully non-linear constitutive law requiring a truncation condition [104]. Figure 3 compares the neural network approach to a linear, Oldroyd-B like, model for σ and showcases that the nonlinear approximation improves the accuracy by more than 50x. We note that the neural network approximation accelerates the solution by 2x over the original 6-state DAE, demonstrating that the universal differential equation approach to model acceleration is not just applicable to large-scale dynamical systems like PDEs but also can be effectively employed to accelerate small scale systems.

4.3 Efficient Discrete Physics-Informed Neural Networks as Universal ODEs

To further demonstrate the breadth of computational problems covered by the UODE framework, we note that the discrete physics-informed neural networks can be cast into the framework of UODEs. A physics-informed neural network is the representation of a PDE's solution via a neural network, allowing machine learning training techniques to solve the equation [12]. These works note that the continuous PDE can be discretized in a single dimension to give rise to the discrete physics-informed neural network, simplified as:

$$u^{n+c_i} = u^n - \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}[u^{n+c_j}] \tag{28}$$

$$u^{n+1} = u^n - \Delta t \sum_{j=1}^{q} b_j \mathcal{N}[u^{n+c_j}] \tag{29}$$

These results have demonstrated that the discrete form can enhance the computational efficiency of training physics-informed neural networks. However, we note that this directly corresponds to training the universal ODE $u' = \mathcal{N}(u)$ using an explicit or implicit Runge-Kutta method in the SciML ecosystem. This directly gives rise to the further work on multistep discrete physics-informed neural networks [70, 80] by training the UODE via a multistep method, but also immediately gives the generalization to Runge-Kutta-Chebyshev, Rosenbrock, exponential integrator, and more formalizations which all are available via the SciML tools.

5 Discussion

While many attribute the success of deep learning to its blackbox nature, the key advances in deep learning applications have come by developing new architectures which directly model the structures that are attempting to be learned.

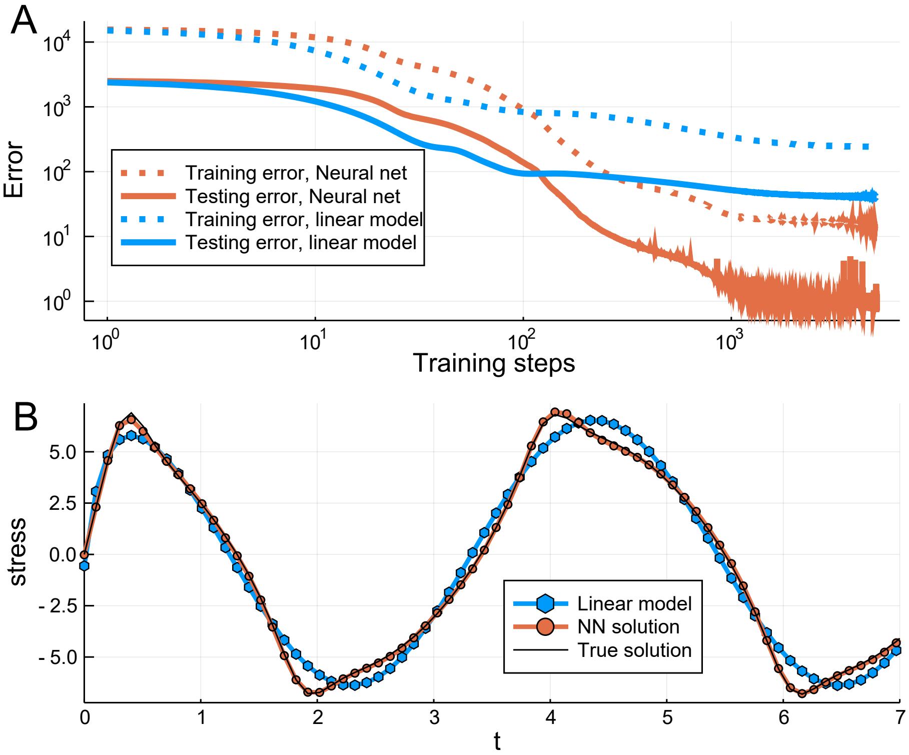

Figure 3: Convergence of neural closure relations for a non-Newtonian Fluid. (A) Error between the approximated σ using the linear approximation Equation 7 and the neural network closure relation Equation 26 against the full FENE-P solution. The error is measured for the strain rates $\dot{\gamma} = 12\cos\omega t$ for $\omega = 1, 1.2, \ldots, 2$ and tested with the strain rate $\dot{\gamma} = 12\cos 1.5t$. (B) Predictions of stress for testing strain rate for the linear approximation and UODE solution against the exact FENE-P stress.

For example, deep convolutional neural networks for image processing directly utilized the local spatial structure of images by modeling convolution stencil operations. Similarly, recurrent neural networks encode a forward time progression into a deep learning model and have excelled in natural language processing and time series prediction. Here we present a software that allows for combining existing scientific simulation libraries with neural networks to train and augment known models with data-driven components. Our results show that by building these hybrid mechanistic models with machine learning, we can arrive at similar efficiency advancements by utilizing all known prior knowledge of the underlying problem's structure. While we demonstrate the utility of UDEs in equation discovery, we have also demonstrated that these methods are capable of solving many other problems, and many methods of recent interest, such as discrete physics-informed neural networks, fall into the class of UDE methods and can thus be analyzed and efficiently computed as part of this formalization.

Our software implementation is the first deep learning integrated differential equation library to include the full spectrum of adjoint sensitivity analysis methods that is required to both efficiently and accurately handle the range of training problems that can arise from universal differential equations. We have demonstrated orders of magnitude performance advantages over previous machine learning enhanced adjoint sensitivity ODE software in a variety of scientific models and demonstrated generalizations to stiff equations, DAEs, SDEs, and more. While the results of this paper span many scientific disciplines and incorporate many different modeling approaches, together all of the examples shown in this manuscript can be implemented using the SciML software ecosystem in just hundreds of lines of code each, with none of the examples taking more than half an hour to train on a standard laptop. This both demonstrates the efficiency of the software and its methodologies, along with the potential to scale to much larger applications.

The code for reproducing the computational experiments can be found at:

https://github.com/ChrisRackauckas/universal_differential_equations

6 Acknowledgements

We thank Jesse Bettencourt, Mike Innes, and Lyndon White for being instrumental in the early development of the DiffEqFlux.jl library, Tim Besard and Valentin Churavy for the help with the GPU tooling, and David Widmann and Kanav Gupta for their fundamental work across DifferentialEquations.jl. Special thanks to Viral Shah and Steven Johnson who have been helpful in the refinement of these ideas. We thank Charlie Strauss, Steven Johnson, Nathan Urban, and Adam Gerlach for enlightening discussions and remarks on our manuscript and software. We thank Stuart Rogers for his careful read and corrections. We thank David Duvenaud for extended discussions on this work. We thank the author of the torchsde library, Xuechen Li, for optimizing the SDE benchmark code.

References

[1] Boukaye Boubacar Traore, Bernard Kamsu-Foguem, and Fana Tangara. Deep convolution neural network for image recognition. *Ecological informatics*, 48:257–268, 2018.

[2] M. T. Islam, B. M. N. Karim Siddique, S. Rahman, and T. Jabid. Image recognition with deep learning. In *2018 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, volume 3, pages 106–110, Oct 2018.

[3] Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan K Su. This looks like that: deep learning for interpretable image recognition. In *Advances in Neural Information Processing Systems*, pages 8928–8939, 2019.

[4] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75, 2018.

[5] Daniel W Otter, Julian R Medina, and Jugal K Kalita. A survey of the usages of deep learning in natural language processing. *arXiv preprint arXiv:1807.10854*, 2018.

[6] Y Tsuruoka. Deep learning and natural language processing. *Brain and nerve= Shinkei kenkyu no shinpo*, 71(1):45, 2019.

[7] Yu Li, Chao Huang, Lizhong Ding, Zhongxiao Li, Yijie Pan, and Xin Gao. Deep learning in bioinformatics: Introduction, application, and perspective in the big data era. *Methods*, 2019.

[8] Binhua Tang, Zixiang Pan, Kang Yin, and Asif Khateeb. Recent advances of deep learning in bioinformatics and computational biology. *Frontiers in Genetics*, 10, 2019.

[9] James Zou, Mikael Huss, Abubakar Abid, Pejman Mohammadi, Ali Torkamani, and Amalio Telenti. A primer on deep learning in genomics. *Nature genetics*, 51(1):12–18, 2019.

[10] Christof Angermueller, Tanel Pärnamaa, Leopold Parts, and Oliver Stegle. Deep learning for computational biology. *Molecular systems biology*, 12(7), 2016.

[11] Davide Bacciu, Paulo JG Lisboa, José D Martín, Ruxandra Stoean, and Alfredo Vellido. Bioinformatics and medicine in the era of deep learning. *arXiv preprint arXiv:1802.09791*, 2018.

[12] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[13] Mauricio Alvarez, David Luengo, and Neil D Lawrence. Latent force models. In *Artificial Intelligence and Statistics*, pages 9–16, 2009.

[14] Yueqin Hu, Steve Boker, Michael Neale, and Kelly L Klump. Coupled latent differential equation with moderators: Simulation and application. *Psychological Methods*, 19(1):56, 2014.

[15] Mauricio Alvarez, Jan R Peters, Neil D Lawrence, and Bernhard Schölkopf. Switched latent force models for movement segmentation. In *Advances in neural information processing systems*, pages 55–63, 2010.

[16] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

[17] Hongzhou Lin and Stefanie Jegelka. Resnet with one-neuron hidden layers is a universal approximator. In *Advances in Neural Information Processing Systems*, pages 6169–6178, 2018.

[18] David A Winkler and Tu C Le. Performance of deep and shallow neural networks, the universal approximation theorem, activity cliffs, and qsar. *Molecular informatics*, 36(1-2):1600118, 2017.

[19] Alexander N Gorban and Donald C Wunsch. The general approximation theorem. In *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98CH36227)*, volume 2, pages 1271–1274. IEEE, 1998.

[20] M.R. Arahal and E.F. Camacho. Neural network adaptive control of nonlinear plants. *IFAC Proceedings Volumes*, 28(13):239 – 244, 1995. 5th IFAC Symposium on Adaptive Systems in Control and Signal Processing 1995, Budapest, Hungary, 14-16 June, 1995.

[21] Wannes De Groote, Edward Kikken, Erik Hostens, Sofie Van Hoecke, and Guillaume Crevecoeur. Neural network augmented physics models for systems with partially unknown dynamics: Application to slider-crank mechanism. *arXiv preprint arXiv:1910.12212*, 2019.

[22] Diederik P Kingma and Jimmy Ba. Adam A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[23] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[24] Ronald M Errico. What is an adjoint model? *Bulletin of the American Meteorological Society*, 78(11):2577–2592, 1997.

[25] Grégoire Allaire. A review of adjoint methods for sensitivity analysis, uncertainty quantification and optimization in numerical codes. *Ingenieurs de l'Automobile*, 836:33–36, July 2015.

[26] Gilbert Strang. *Computational science and engineering*, volume 791. Wellesley-Cambridge Press Wellesley, 2007.

[27] Alan C Hindmarsh, Peter N Brown, Keith E Grant, Steven L Lee, Radu Serban, Dan E Shumaker, and Carol S Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.

[28] Steven G Johnson. Notes on adjoint methods for 18.335.

[29] Biswa Sengupta, Karl J Friston, and William D Penny. Efficient gradient computation for dynamical models. *NeuroImage*, 98:521–527, 2014.

[30] Christopher Rackauckas, Yingbo Ma, Vaibhav Dixit, Xingjian Guo, Mike Innes, Jarrett Revels, Joakim Nyberg, and Vijay Ivaturi. A comparison of automatic differentiation and continuous sensitivity analysis for derivatives of differential equation solutions. *arXiv preprint arXiv:1812.01892*, 2018.

[31] Michael Innes, Elliot Saba, Keno Fischer, Dhairya Gandhi, Marco Concetto Rudilosso, Neethu Mariya Joy, Tejan Karmali, Avik Pal, and Viral Shah. Fashionable modelling with flux. *CoRR*, abs/1811.01457, 2018.

[32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[33] Mike Innes, Alan Edelman, Keno Fischer, Chris Rackauckus, Elliot Saba, Viral B Shah, and Will Tebbutt. Zygote: A differentiable programming system to bridge machine learning and scientific computing. *arXiv preprint arXiv:1907.07587*, 2019.

[34] Ralf Giering, Thomas Kaminski, and Thomas Slawig. Generating efficient derivative code with taf: adjoint and tangent linear euler flow around an airfoil. *Future generation computer systems*, 21(8):1345–1355, 2005.

[35] Laurent Hascoet and Valérie Pascual. The tapenade automatic differentiation tool: Principles, model, and specification. *ACM Transactions on Mathematical Software (TOMS)*, 39(3):20, 2013.

[36] Christopher Rackauckas and Qing Nie. Differentialequations.jl – a performant and feature-rich ecosystem for solving differential equations in julia. *The Journal of Open Research Software*, 5(1), 2017. Exported from https://app.dimensions.ai on 2019/05/05.

[37] Tapio Schneider, Shiwei Lan, Andrew Stuart, and Joao Teixeira. Earth system modeling 2.0: A blueprint for models that learn from observations and targeted high-resolution simulations. *Geophysical Research Letters*, 44(24):12–396, 2017.

[38] Sebastian Krämer, David Plankensteiner, Laurin Ostermann, and Helmut Ritsch. Quantumoptics.jl: A julia framework for simulating open quantum systems. *Computer Physics Communications*, 227:109 – 116, 2018.

[39] Amir Gholami, Kurt Keutzer, and George Biros. Anode: Unconditionally accurate memory-efficient gradients for neural odes. *arXiv preprint arXiv:1902.10298*, 2019.

[40] Hong Zhang, Shrirang Abhyankar, Emil Constantinescu, and Mihai Anitescu. Discrete adjoint sensitivity analysis of hybrid dynamical systems with switching. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64(5):1247–1259, 2017.

[41] Thomas Lauß, Stefan Oberpeilsteiner, Wolfgang Steiner, and Karin Nachbagauer. The discrete adjoint method for parameter identification in multibody system dynamics. *Multibody system dynamics*, 42(4):397–410, 2018.

[42] J. Revels, M. Lubin, and T. Papamarkou. Forward-mode automatic differentiation in julia. *arXiv:1607.07892 [cs.MS]*, 2016.

[43] Derek Onken and Lars Ruthotto. Discretize-optimize vs. optimize-discretize for time-series regression and continuous normalizing flows. *arXiv preprint arXiv:2005.13420*, 2020.

[44] Feby Abraham, Marek Behr, and Matthias Heinkenschloss. The effect of stabilization in finite element methods for the optimal boundary control of the oseen equations. *Finite Elements in Analysis and Design*, 41(3):229 – 251, 2004.

[45] John T Betts and Stephen L Campbell. Discretize then optimize. *Mathematics for industry: challenges and frontiers*, pages 140–157, 2005.

[46] Geng Liu, Martin Geier, Zhenyu Liu, Manfred Krafczyk, and Tao Chen. Discrete adjoint sensitivity analysis for fluid flow topology optimization based on the generalized lattice boltzmann method. *Computers & Mathematics with Applications*, 68(10):1374 – 1392, 2014.

[47] Alfonso Callejo, Valentin Sonneville, and Olivier A Bauchau. Discrete adjoint method for the sensitivity analysis of flexible multibody systems. *Journal of Computational and Nonlinear Dynamics*, 14(2), 2019.

[48] S Scott Collis and Matthias Heinkenschloss. Analysis of the streamline upwind/petrov galerkin method applied to the solution of optimal control problems. 2002.

[49] Jun Liu and Zhu Wang. Non-commutative discretize-then-optimize algorithms for elliptic pde-constrained optimal control problems. *Journal of Computational and Applied Mathematics*, 362:596–613, 2019.

[50] E Huntley. A note on the application of the matrix riccati equation to the optimal control of distributed parameter systems. *IEEE Transactions on Automatic Control*, 24(3):487–489, 1979.

[51] Ziv Sirkes and Eli Tziperman. Finite difference of adjoint or adjoint of finite difference? *Monthly weather review*, 125(12):3373–3378, 1997.

[52] Guojun Hu and Tomasz Kozlowski. Assessment of continuous and discrete adjoint method for sensitivity analysis in two-phase flow simulations. *arXiv preprint arXiv:1805.08083*, 2018.

[53] JOHANNES Kepler. Sensitivity analysis: The direct and adjoint method.

[54] F Van Keulen, RT Haftka, and NH Kim. Review of options for structural design sensitivity analysis. part 1: Linear systems. *Computer methods in applied mechanics and engineering*, 194(30-33):3213–3243, 2005.

[55] M Kouhi, G Houzeaux, F Cucchietti, M Vázquez, and F Rodriguez. Implementation of discrete adjoint method for parameter sensitivity analysis in chemically reacting flows.

[56] Siva Nadarajah and Antony Jameson. A comparison of the continuous and discrete adjoint approach to automatic aerodynamic optimization. In *38th Aerospace Sciences Meeting and Exhibit*, page 667.

[57] Tianyi Gou and Adrian Sandu. Continuous versus discrete advection adjoints in chemical data assimilation with cmaq. *Atmospheric environment*, 45(28):4868–4881, 2011.

[58] Nicolas R Gauger, Michael Giles, Max Gunzburger, and Uwe Naumann. Adjoint methods in computational science, engineering, and finance (dagstuhl seminar 14371). In *Dagstuhl Reports*, volume 4. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[59] G. Hu and T. Kozlowski. Development and assessment of adjoint sensitivity analysis method for transient two-phase flow simulations. pages 2246–2259, January 2019. 18th International Topical Meeting on Nuclear Reactor Thermal Hydraulics, NURETH 2019 ; Conference date: 18-08-2019 Through 23-08-2019.

[60] Dacian N. Daescu, Adrian Sandu, and Gregory R. Carmichael. Direct and adjoint sensitivity analysis of chemical kinetic systems with kpp: Ii—numerical validation and applications. *Atmospheric Environment*, 37(36):5097 – 5114, 2003.

[61] A Schwartz and E Polak. Runge-kutta discretization of optimal control problems. *IFAC Proceedings Volumes*, 29(8):123–128, 1996.

[62] Kimia Ghobadi, Nedialko S Nedialkov, and Tamas Terlaky. On the discretize then optimize approach. *Preprint for Industrial and Systems Engineering*, 2009.

[63] Alain Sei and William W Symes. A note on consistency and adjointness for numerical schemes. 1995.

[64] William W Hager. Runge-kutta methods in optimal control and the transformed adjoint system. *Numerische Mathematik*, 87(2):247–282, 2000.

[65] Adrian Sandu, Dacian N Daescu, Gregory R Carmichael, and Tianfeng Chai. Adjoint sensitivity analysis of regional air quality models. *Journal of Computational Physics*, 204(1):222–252, 2005.

[66] Shashi Gowda, Yingbo Ma, Valentin Churavy, Alan Edelman, and Christopher Rackauckas. Sparsity programming: Automated sparsity-aware optimizations in differentiable programming. 2019.

[67] Xuechen Li, Ting-Kam Leonard Wong, Ricky T. Q. Chen, and David Duvenaud. Scalable gradients for stochastic differential equations. *International Conference on Artificial Intelligence and Statistics*, 2020.

[68] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I (2nd Revised. Ed.): Nonstiff Problems*. Springer-Verlag, Berlin, Heidelberg, 1993.

[69] Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. Pde-net: Learning pdes from data. *arXiv preprint arXiv:1710.09668*, 2017.

[70] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Multistep neural networks for data-driven discovery of nonlinear dynamical systems. *arXiv preprint arXiv:1801.01236*, 2018.

[71] Peter J Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of fluid mechanics*, 656:5–28, 2010.

[72] Matthew O Williams, Ioannis G Kevrekidis, and Clarence W Rowley. A data–driven approximation of the koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25(6):1307–1346, 2015.

[73] Qianxiao Li, Felix Dietrich, Erik M Bollt, and Ioannis G Kevrekidis. Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the koopman operator. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(10):103111, 2017.

[74] Naoya Takeishi, Yoshinobu Kawahara, and Takehisa Yairi. Learning koopman invariant subspaces for dynamic mode decomposition. In *Advances in Neural Information Processing Systems*, pages 1130–1140, 2017.

[75] Hayden Schaeffer. Learning partial differential equations via data discovery and sparse optimization. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 473(2197):20160446, 2017.

[76] Markus Quade, Markus Abel, Kamran Shafi, Robert K Niven, and Bernd R Noack. Prediction of dynamical systems by symbolic regression. *Physical Review E*, 94(1):012214, 2016.

[77] Hongqing Cao, Lishan Kang, Yuping Chen, and Jingxian Yu. Evolutionary modeling of systems of ordinary differential equations with genetic programming. *Genetic Programming and Evolvable Machines*, 1(4):309–337, 2000.

[78] Khalid Raza and Rafat Parveen. Evolutionary algorithms in genetic regulatory networks model. *CoRR*, abs/1205.1986, 2012.

[79] Hayden Schaeffer, Giang Tran, and Rachel Ward. Extracting sparse high-dimensional dynamics from limited data. *SIAM Journal on Applied Mathematics*, 78(6):3279–3295, 2018.

[80] Ramakrishna Tipireddy, Paris Perdikaris, Panos Stinis, and Alexandre M. Tartakovsky. A comparative study of physics-informed neural network models for learning unknown dynamics and constitutive relations. *CoRR*, abs/1904.04058, 2019.

[81] Steven L Brunton, 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):3932–3937, 2016.

[82] Niall M Mangan, Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Inferring biological networks by sparse identification of nonlinear dynamics. *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, 2(1):52–63, 2016.

[83] Niall M Mangan, J Nathan Kutz, Steven L Brunton, and Joshua L Proctor. Model selection for dynamical systems via sparse regression and information criteria. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 473(2204):20170009, 2017.

[84] Peng Zheng, Travis Askham, Steven L. Brunton, J. Nathan Kutz, and Aleksandr Y. Aravkin. A unified framework for sparse relaxed regularized regression: SR3. 7:1404–1423. Conference Name: IEEE Access.

[85] Kathleen Champion, Peng Zheng, Aleksandr Y. Aravkin, Steven L. Brunton, and J. Nathan Kutz. A unified sparse optimization framework to learn parsimonious physics-informed models from data.

[86] Alexander Bills, Shashank Sripad, William Leif Fredericks, Matthew Guttenberg, Devin Charles, Evan Frank, and Venkatasubramanian Viswanathan. Universal Battery Performance and Degradation Model for Electric Aircraft. 7 2020.

[87] Raj Dandekar and Lydia Bourouiba. Splash upon impact on a deep pool. In preparation.

[88] Yang Cao, Shengtai Li, Linda Petzold, and Radu Serban. Adjoint sensitivity analysis for differential-algebraic equations: The adjoint dae system and its numerical solution. *SIAM journal on scientific computing*, 24(3):1076–1089, 2003.

[89] R. A. Fisher. The wave of advance of advantageous genes. *Annals of Eugenics*, 7(4):355–369, 1937.

[90] P. Grindrod. *The Theory and Applications of Reaction-diffusion Equations: Patterns and Waves*. Oxford applied mathematics and computing science series. Clarendon Press, 1996.

[91] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, Dec 2018.

[92] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

[93] E Weinan, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, 2017.

[94] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[95] Yaohua Zang, Gang Bao, Xiaojing Ye, and Haomin Zhou. Weak adversarial networks for high-dimensional partial differential equations. *arXiv preprint arXiv:1907.08272*, 2019.

[96] Côme Huré, Huyên Pham, and Xavier Warin. Some machine learning schemes for high-dimensional nonlinear pdes. *arXiv preprint arXiv:1902.01599*, 2019.

[97] H Lamba. An adaptive timestepping algorithm for stochastic differential equations. *Journal of computational and applied mathematics*, 161(2):417–430, 2003.

[98] Benoit Cushman-Roisin and Jean-Marie Beckers. Chapter 4 - equations governing geophysical flows. In Benoit Cushman-Roisin and Jean-Marie Beckers, editors, *Introduction to Geophysical Fluid Dynamics*, volume 101 of *International Geophysics*, pages 99 – 129. Academic Press, 2011.

[99] Zhihua Zhang and John C. Moore. Chapter 11 - atmospheric dynamics. In Zhihua Zhang and John C. Moore, editors, *Mathematical and Physical Fundamentals of Climate Change*, pages 347 – 405. Elsevier, Boston, 2015.

[100] Section 1.3 - governing equations. In Lakshmi H. Kantha and Carol Anne Clayson, editors, *Numerical Models of Oceans and Oceanic Processes*, volume 66 of *International Geophysics*, pages 28–46. Academic Press, 2000.

[101] Stephen M Griffies and Alistair J Adcroft. Formulating the equations of ocean models. 2008.

[102] Stephen M. Griffies, Michael Levy, Alistair J. Adcroft, Gokhan Danabasoglu, Robert W. Hallberg, Doug Jacobsen, William Large, , and Todd Ringler. Theory and Numerics of the Community Ocean Vertical Mixing (CVMix) Project. Technical report, 2015. Draft from March 9, 2015. 98 + v pages.

[103] F.A. Morrison and A.P.C.E.F.A. Morrison. *Understanding Rheology*. Raymond F. Boyer Library Collection. Oxford University Press, 2001.

[104] P.J. Oliveira. Alternative derivation of differential constitutive equations of the oldroyd-b type. *Journal of Non-Newtonian Fluid Mechanics*, 160(1):40 – 46, 2009. Complex flows of complex fluids.

Figures

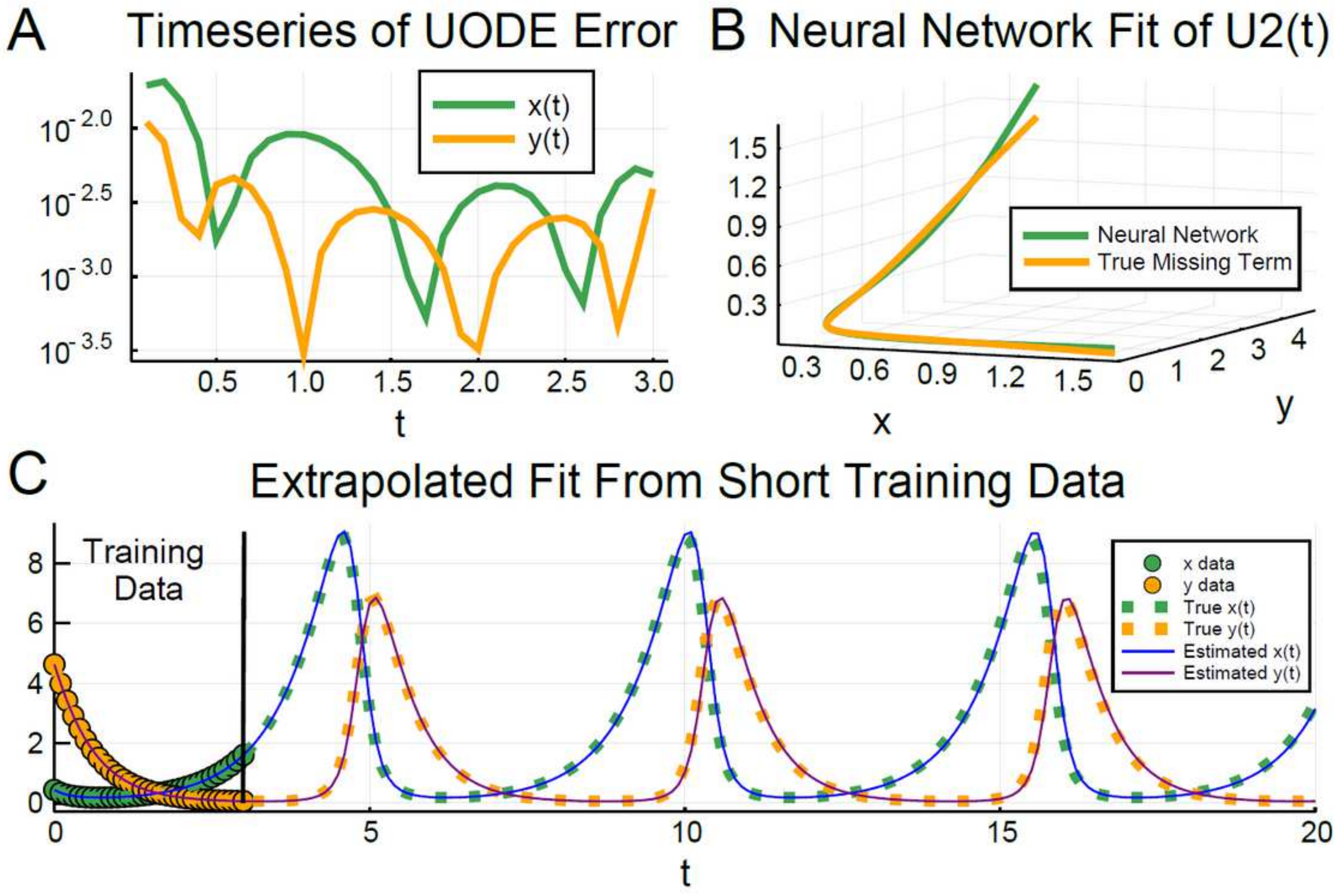

Figure 1

Automated Lotka-Volterra equation discovery with UODE-enhanced SInDy. (A) The error in the trained UODE against x(t) and y(t) in green and yellow respectively. (B) The measured values of the missing term U2(x; y) throughout the time series, with the neural network approximate in green and the true value xy in yellow. (C) The extrapolation of the knowledge-enhanced SInDy fit series. The green and yellow dots show the data that was used to fit the UODE, and the dots show the true solution of the Lotka-Volterra Equations 8 beyond the training data. The blue and purple lines show the extrapolated solution how the UODE-enhanced SInDy recovered equations.

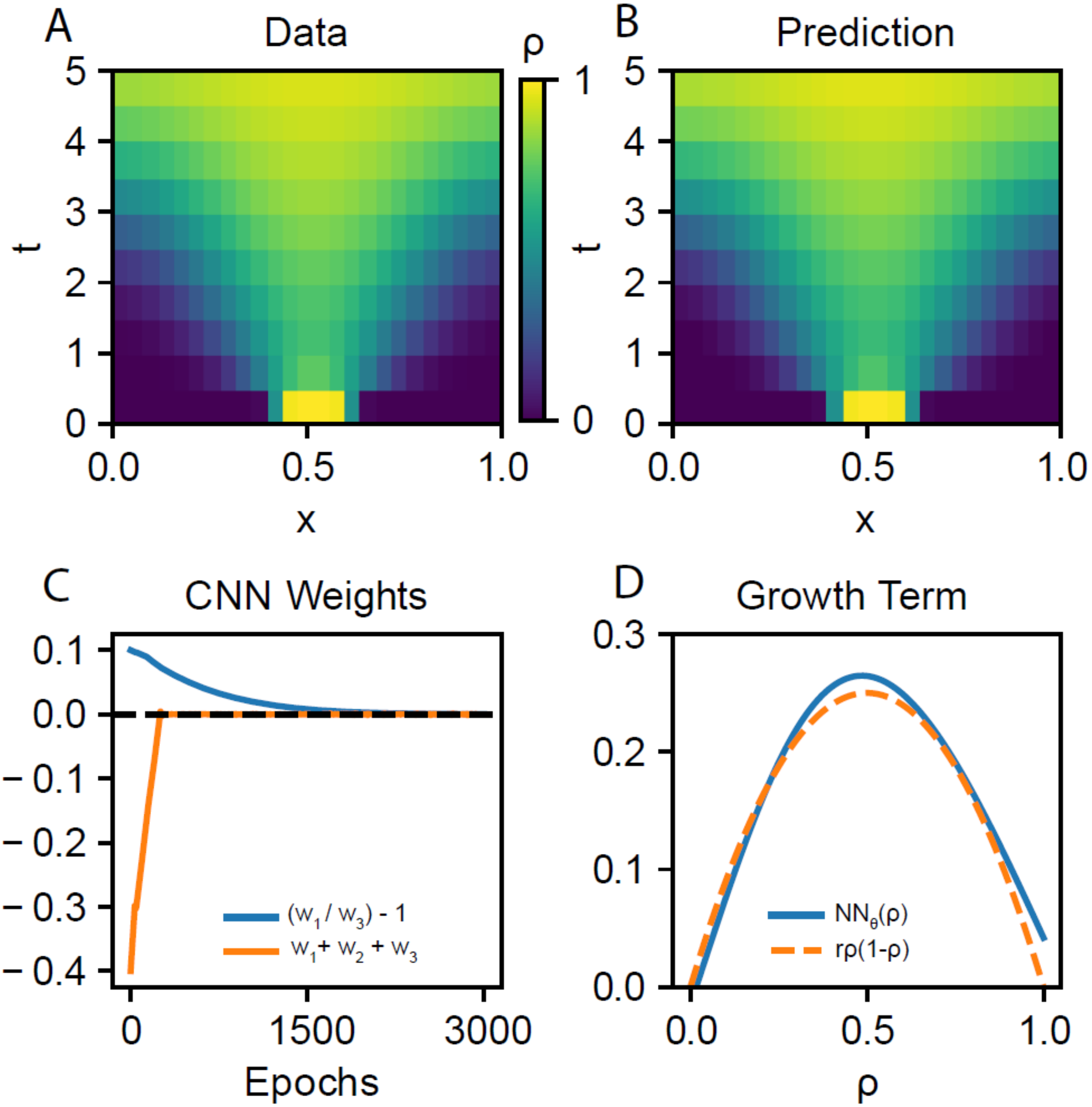

Figure 2

Recovery of the UPDE for the Fisher-KPP equation. (A) Training data and (B) prediction of the UPDE for ρ(x, t). (C) Curves for the weights of the CNN filter [w1,w2,w3] indicate the recovery of the [1,−2,1] stencil for the 1-dimensional Laplacian. (D) Comparison of the learned (blue) and the true growth term (orange) showcases the learned parabolic form of the missing nonlinear equation.

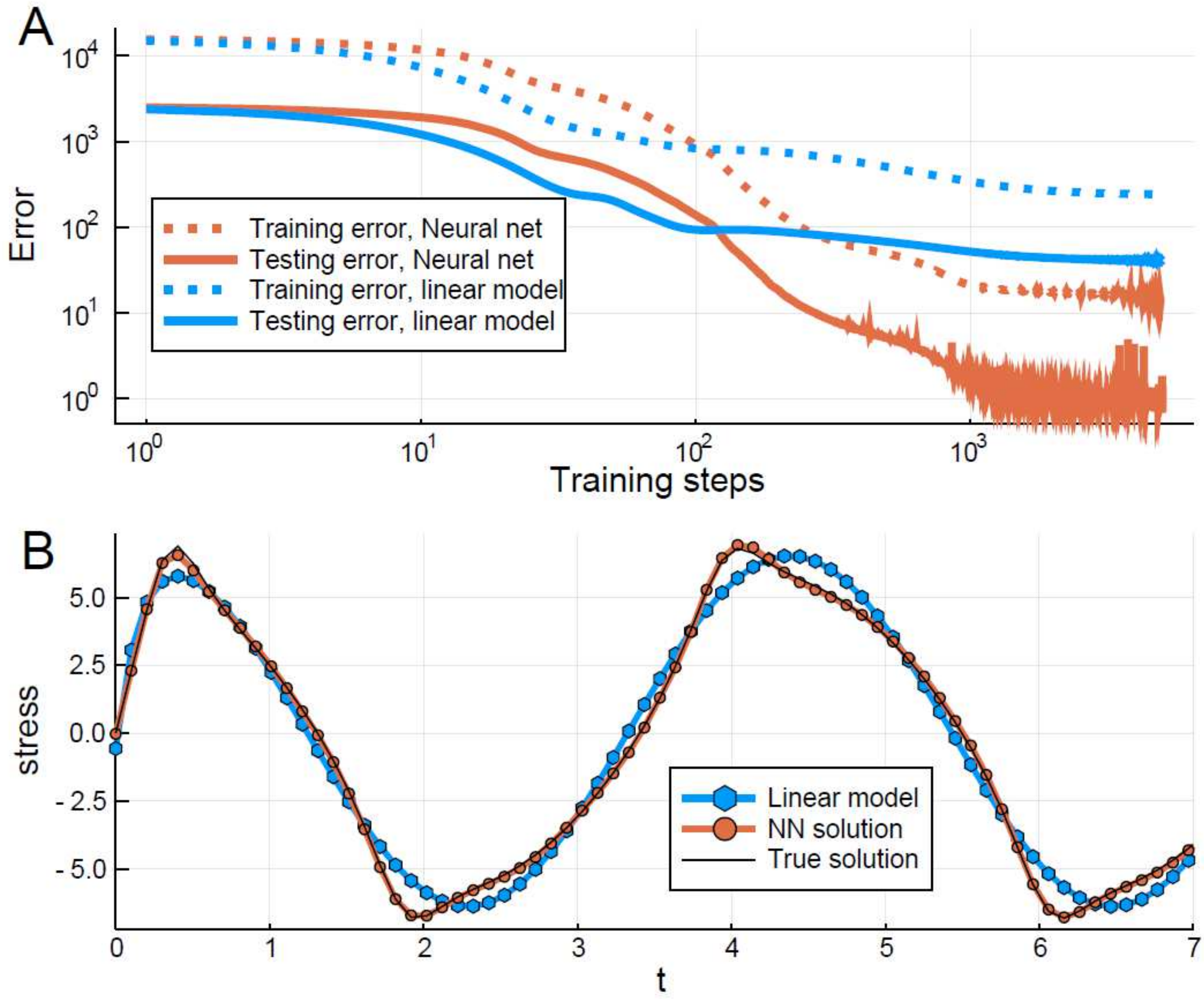

Figure 3

Convergence of neural closure relations for a non-Newtonian Fluid. (A) Error between the approximated σ using the linear approximation Equation 7 and the neural network closure relation Equation 26 against the full FENE- P solution. The error is measured for the strain rates γ = 12cos ωt for ω = 1,1.2,...,2 and tested with the strain rate γ = 12 cos 1.5t. (B) Predictions of stress for testing strain rate for the linear approximation and UODE solution against the exact FENE-P stress.

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- UniversalDifferentialEquationsSupplement.pdf