Chaos
An Interdisciplinary Journal of Nonlinear Science

RESEARCH ARTICLE | SEPTEMBER 20 2021

Stiff neural ordinary differential equations FREE

Suyong Kim ; Weiqi Ji ; Sili Deng ; Yingbo Ma; Christopher Rackauckas

Chaos 31, 093122 (2021)
https://doi.org/10.1063/5.0060697
CHORUS

AIP Publishing

Stiff neural ordinary differential equations

Cite as: Chaos **31**, 093122 (2021); doi: 10.1063/5.0060697
Submitted: 21 June 2021 · Accepted: 31 August 2021 ·
Published Online: 20 September 2021

Suyong Kim,[1] Weiqi Ji,[1] Sili Deng,[1,a)] Yingbo Ma,[2] and Christopher Rackauckas[3,4,5,a)]

AFFILIATIONS

[1]Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, USA
[2]Julia Computing Inc., Cambridge, Massachusetts 02144, USA
[3]Department of Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, USA
[4]School of Pharmacy, University of Maryland, Baltimore, Maryland 21201, USA
[5]Pumas AI, Baltimore, Maryland 21201, USA

[a)]**Authors to whom correspondence should be addressed:** silideng@mit.edu and contact@chrisrackauckas.com

ABSTRACT

Neural Ordinary Differential Equations (ODEs) are a promising approach to learn dynamical models from time-series data in science and engineering applications. This work aims at learning neural ODEs for stiff systems, which are usually raised from chemical kinetic modeling in chemical and biological systems. We first show the challenges of learning neural ODEs in the classical stiff ODE systems of Robertson's problem and propose techniques to mitigate the challenges associated with scale separations in stiff systems. We then present successful demonstrations in stiff systems of Robertson's problem and an air pollution problem. The demonstrations show that the usage of deep networks with rectified activations, proper scaling of the network outputs as well as loss functions, and stabilized gradient calculations are the key techniques enabling the learning of stiff neural ODEs. The success of learning stiff neural ODEs opens up possibilities of using neural ODEs in applications with widely varying time-scales, such as chemical dynamics in energy conversion, environmental engineering, and life sciences.

Neural Ordinary Differential Equations (ODEs) have been emerged as a powerful tool to describe a dynamical system using an artificial neural network. Despite its many advantages, there are many scientific and engineering examples where neural ODEs may fail due to stiffness. This study demonstrates how instability arises in neural ODEs during the training with the benchmark stiff system. Furthermore, we apply scaling to differential equations and loss functions to mitigate stiffness. The proposed technique in adjoint handling and equation scaling for stiff neural ODEs can be used in many biological, chemical, and environmental systems subject to stiffness.

I. INTRODUCTION

Neural Ordinary Differential Equations (ODEs) are elegant approaches for data-driven modeling of dynamical systems from time-series data in science and engineering applications. Neural ODEs offer various advantages over physics-based modeling and traditional neural network modeling. Compared to physics-based modeling where a presumed functional form is required, the neural ODEs approach precludes the need for expert knowledge in identifying such prior information.[1,2] Compared to traditional neural network modeling, such as recurrent neural networks, neural ODEs enable flexibility in the modeling of irregularly and incomplete sampled time series[3] and can be more efficient by taking advantage of modern ODE solvers and adjoint methods.[4,5]

Despite the success of neural ODEs[4] in many sciences and engineering problems,[6–9] it has been recognized that it is very challenging to learn neural ODEs for stiff dynamics,[10] characterized by dynamics with widely separated time scales. Part of the challenge is due to the high computational cost of solving stiff ODEs as well as handling the ill-conditioned gradients of loss functions with respect to neural network weights.[11] In addition, it has been shown that stiffness could lead to failures in many data-driven modeling approaches, such as reduced order of modeling[12] and physics-informed neural networks.[13] It was suggested that the stiffness could lead to gradient pathologies and ill-conditioned optimization problems, which leads to the failure of stochastic gradient descent based optimization.[14]

This work is motivated to elucidate the challenges of learning neural ODEs for stiff systems and proposes strategies to overcome these obstacles. We study learning neural ODEs on data generated

from two classical stiff systems, ROBER[15] and POLLU,[16] which are extensively used for benchmarking stiff ODE integrators. Both ROBER and POLLU describe the dynamics of species concentrations in stiff chemical reaction systems. We propose strategies to mitigate the challenges associated with the scale separations in time as well as the magnitudes of states. We are thus able to successfully learn stiff neural ODEs for those two benchmark problems. We demonstrate how the application of adjoint methods to stiff systems leads to numerical issues and changes in the computational complexity that are overcome by proposing new adjoint techniques. Those strategies not only demonstrate successful numerical methods and architectures for handling time scale separations in neural ODEs but also elucidate general theoretical issues of data-driven modeling of stiff dynamical systems, which can be transferred to other techniques.

II. BACKGROUND

A. Neural ordinary differential equations

Many science and engineering problems can be formulated as ODEs; that is,

$$\frac{dy(t)}{dt} = f\left(y(t), \theta, t\right), \tag{1}$$

where t is time, $y(t)$ is the state variables, and the function f models the dynamics. Identifying the model f is one of the central tasks in many scientific discoveries, which usually involves proposing functional forms and then performing parameter inference against observation data. Proposing functional forms of f is a very challenging task for many complex systems, such as modeling the gene regulatory networks and cell signaling networks in life science.[17,18] Discovering the functional forms of f could require decades of effort with expert knowledge, and it is often the case that a lot of unknown dynamics are yet to discover in those complex systems.[19,20] With the help of the universal approximation theorem, one can approximate the model f using a neural network without worrying about missing important dynamics as in physics-based modeling; i.e.,

$$\frac{dy(t)}{dt} = NN_\theta\left(y(t)\right). \tag{2}$$

If one can measure the data pair of $\left(y(t), \frac{dy(t)}{dt}\right)$, one can train the neural network straightforwardly as a normal supervised learning problem. However, we usually only have access to the time-series data of $y(t)$. On the contrary, neural ODEs train the NN by integrating the ODEs,

$$\begin{aligned} y(t_1) &= y(t_0) + \int_{t_0}^{t_1} NN\left(y(t)\right) dt \\ &= ODESolve\left(y(t_0), NN, t_0, t_1, \theta\right), \end{aligned} \tag{3}$$

where $y(t_0)$ are the initial conditions of the integration and $t \in (t_0, t_1)$ is the range of the integration. We can define the loss functions as the difference between the observed state variables and the predicted state variables,

$$L(\theta) = MAE\left(y(t)^{model}, y(t)^{obs}\right), \tag{4}$$

where the mean absolute error (MAE) is as an example metric. Using modern differentiable ODE solvers,[5] one can compute the gradient of the loss function to the model parameters efficiently. This work employs Julia's[21] Scientific Machine Learning ecosystem, mainly consisting of DifferentialEquations.jl,[22] Flux.jl,[23] and ForwardDiff.jl[24] to integrate the neural ODEs and do the backpropagation. We can thus use stochastic gradient descent algorithms to optimize the neural network.

B. Stiff ODE systems

Ernst Hairer's classic working definition for the field of numerical ODE solving is "stiff equations are problems for which explicit methods do not work."[25,26] In other words, it is the case where numerical ODE solvers such as the `dopri`method, chosen in the original neural ODE paper,[4] fail to adequately solve the differential equation. Many definitions have been proposed to explain this phenomenon. A commonly used definition is the stiffness index,

$$S = \frac{Re(\lambda_{\max})}{Re(\lambda_{\min})}(t_1 - t_0), \tag{5}$$

where λ_i are the eigenvalues of the Jacobian. While these types of stiffness can be helpful in identifying essential analytical properties for numerical methods to capture, every stiffness index does not capture all of the numerically difficult problems. Shampine famously stated: "Indeed, the classic Robertson chemical kinetics problem typically used to illustrate stiffness has two zero eigenvalues. Any definition of stiffness that does not apply to this standard test problem is not very useful."[27] In addition, the role of the time span is often overlooked. Shampine notes that a system may not seem stiff if one is only solving on the timescale of the fast process: stiffness requires that one is investigating the effects on the time scale of the slow process.

Due to the ambiguity of the definition of stiff systems, we will choose two differential equations studied extensively by the stiff ODE solver literature as representative highly stiff systems:[25] ROBER[15] and POLLU.[16] Specifically, the canonical ROBER problem consists of three species and five reactions, and the POLLU problem consists of 20 species and 25 reactions describing the air pollution formation in atmospheric chemistry. The formula of the ROBER and POLLU problem is presented in Sec. IV and the supplementary material. We will investigate the difficulty of fitting data sampled from these systems, thus illuminating the unique features of the training process that arise in the discovery of stiff systems and the numerical methods for handling them.

III. METHOD

A. Stability of adjoints on stiff ODEs

The core problem of training neural ODEs is calculating the gradient of the ODE's solution. In the method proposed by Chen *et al.*,[4] the ODE is reversed and augmented with the adjoint

equations as

$$\frac{d}{dt}\begin{bmatrix} z \\ \omega \\ \frac{dL}{d\theta} \end{bmatrix} = -\begin{bmatrix} 1 & \omega^T & \omega^T \end{bmatrix}\begin{bmatrix} f & \frac{\partial f}{\partial z} & \frac{\partial f}{\partial \theta} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{6}$$

However, previous work has called into question the stability of reversing the ODE equation; i.e.,

$$z' = -f(z, \theta, -t) \tag{7}$$

when solving from t_1 to t_0. While theoretically this reverses the ODE exactly, in practice, numerical issues can cause errors that propagate into the gradient. Gholami *et al.*[28] noted that if the Lipschitz constant is too large in an absolute value, then either the forward pass or reverse solve is numerically unstable. This can be seen by using a standard ODE solver on the linear ODE $\frac{dy}{dt} = -\lambda y$ on $t \in (0, 1)$ with $y(0) = 1$, $\lambda = 100$ imparts about 1% error, while $\lambda = 10\,000$ cannot be reversed numerically in double precision.

To show how this result extends to stiff ODEs, recall that the Lipschitz constant of a nonlinear function is the infimum of values Λ such that $\left\| \frac{f(x)-f(y)}{x-y} \right\| \leq \Lambda$ for some domain $x, y \in \Omega$, which directly implies that the global Lipschitz constant satisfies $\| \frac{df}{dy} \|_\infty \leq \Lambda$ over the whole domain. Thus, because the maximum eigenvalues of the Jacobian give a lower bound on the Lipschitz constant, stiff ODEs fall into the class of problems that are numerically difficult to reverse. Figure 1 demonstrates this on the ROBER stiff ODE test problem[13,15] and demonstrates that even with imperceptible errors in the forward pass, we can receive noticeable errors in the reverse solve even when plotted in a log scale. Using the well-known CVODE BDF integrator from Sundials,[29] we see that with $tol = abstol = reltol = 10^{-6}$, we receive 72% error, $tol = 10^{-7}$ gives 38% error, and $tol = 10^{-8}$ gives 5% error, with 0.1% error only reached at $tol < 10^{-10}$, i.e., below the threshold of single precision arithmetic.

The adjoint calculation only amplifies this error. With a test loss function $L(\theta) = \sum y_1(t_i)$ at evenly spaced points in the log-space, using the adjoint as described caused CVODE to exit with a Newton divergence near the start of the reverse pass for all $tol = 10^{-i}$ for integers $i = 6, 7, 8, 9, 10, 11, 12, 13, 14$. Manually inspecting the solution process reveals the problem. While the reverse solution only has an approximately 10^{-10} error in y_2 at $t = 99\,999.989\,99$ as shown in Fig. 2, the large k_2 amplifies this term by 6×10^7 imparting approximately $\mathcal{O}(10^{-3})$ error into $\frac{d}{dt}\frac{dL}{d\theta}$. This is because while in the original ODE, the middle term is of the form $k_2 y_2^2$; in the Jacobian, the term is $2k_2 y_2$; and without the squaring, the small error is not diminished. This sends $\frac{dL}{d\theta}$ negative in a way that linearly explodes.

The simple linear ODE $u'(t) = \lambda u(t), u(0) = 1$ on the domain $t \in [0, 1]$ is sufficient to demonstrate the blow-up behavior of the reverse solve after the forward solve. In the presence of truncation error and round-off error, the forward solution is $u_f(t) = e^{t\lambda} + \varepsilon_f(t)$, where ε_f denotes the error of the forward solve. It follows that the reverse problem has the initial condition $u_b(1) = e^{\lambda} + \varepsilon_f(1)$. Thus, at $t = 0$, the backward solution is $u_b(0) = 1 + e^{-\lambda}\varepsilon_f(1) + \varepsilon_b(0)$,

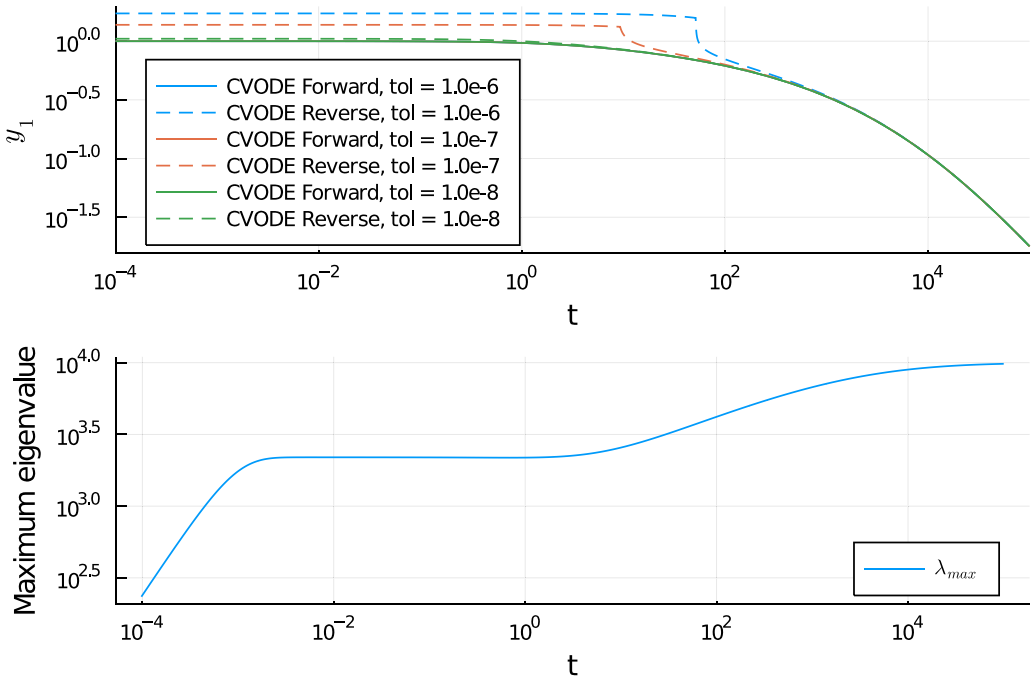

FIG. 1. Reversability of the ROBER[15] equation. Top: Solution forward and backward of y_1 in the ROBER test problem with the CVODE integrator at the shown tolerance using the BDF tableau. Bottom: Maximum eigenvalue of the Jacobian over the solution trajectory.

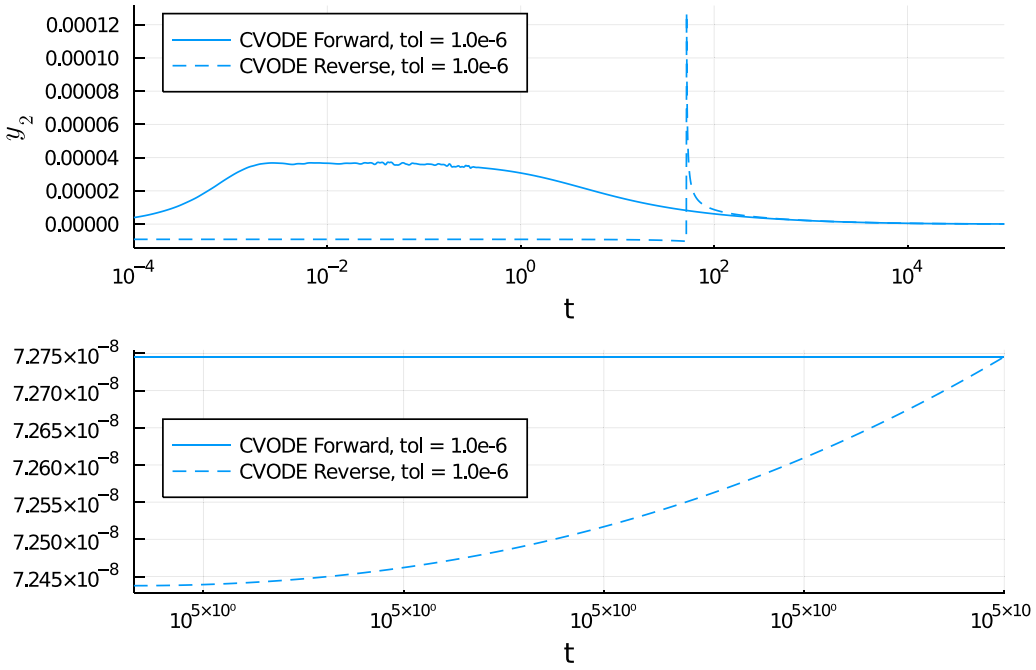

FIG. 2. Reversability instability inspection on the ROBER equation. Top: Solution forward and backward of y_2 in the ROBER test problem with the CVODE integrator at the shown tolerance using the BDF tableau. Bottom: Zoomed in solution on $t \in (999\,99.989\,99, 10^5)$.

where ε_b denotes the error of the backward solve. Therefore, the total error at $t = 0$ from solving the linear equation forward and backward is $e^{-\lambda}\varepsilon_f(1) + \varepsilon_b(0)$, and it exhibits inherit exponential blow-up behavior when $\lambda < 0$. A stable forward problem implies the exponential divergence of the reverse problem. To demonstrate the exponential diverge, the above simple linear ODE is solved forward and backward, and the error at $t = 0$ is plotted with various parameters λ in Fig. 3.

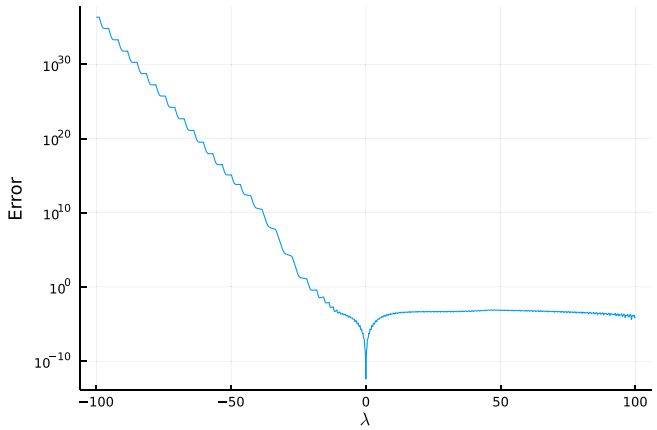

FIG. 3. Reversability of the linear ODE. The total error of the solution produced by the Tsit5 solver at time $t = 0$ is plotted with varying λ. Note that the error is plotted in a logarithmic scale to illustrate the exponential blow-up behavior.

Avoiding this issue requires one does not attempt to reverse the ODE. To this effect, we tested the interpolated checkpointing method of `CVODES`[29] (known as `InterpolatingAdjoint` in DiffEqFlux) and discrete adjoint sensitivities via automatic differentiation of the solver,[30] which both successfully solved the equation and agreed on all derivatives to at least three decimal places when solved with a $tol = 10^{-6}$. We note that one can prove the consistency of the adjoint discretization in the case of discrete adjoint sensitivities ensuring the stability.[31]

B. Linear scaling adjoints

Being limited to non-reversing adjoints, we investigate the computational cost of derivative calculations on stiff ODEs. To illustrate the computational cost and motivate new formulations, consider the implicit Euler method,

$$y_{n+1} = y_n + hf(y_{n+1}, t + h), \tag{8}$$

which is solved via the root finding problem $G(y_{n+1}) = y_{n+1} - y_n + hf(y_{n+1}, t + h) = 0$ for the next step. The instability of fixed point solvers[25] makes it necessary to solve the root finding problem by Newton's method; i.e., $y_{i+1} = y_i - \frac{dG(y_i)}{dy_i}^{-1} G(y_i)$ so that $y_i \to y_{n+1}$. The core computational cost is thus the solving of the linear system $\frac{dG(y_i)}{dy_i} x = G(y_i)$. The condition number of the matrix $\frac{dG(y_i)}{dy_i}$ is precisely the ratios of the maximum and minimum eigenvalues of the system's Jacobian, meaning that the linear system is ill-conditioned

for stiff equations. Krylov subspace methods such as GMRES have slow convergence when the condition number of the matrix is high[32] and thus using these methods can require problem-specific preconditioners to make convergence computationally acceptable. For this reason, methods for solving stiff ODEs generally attempt direct methods, either dense or sparse, on the Jacobian. The general cost of performing such a linear solve via LU-factorization is $\mathcal{O}(k^3)$, where k is the number of columns in the Jacobian, which is equivalent to the number of states in the ODE.

The cubic cost growth of solving a stiff system changes the efficiency calculus of methods for computing the derivative of the ODE solution and thus for calculating the gradient of the neural ODE's loss function. In the traditional argument, forward-mode automatic differentiation of an ODE is equivalent to solving the forward sensitivity equations,

$$z' = f(z, p, t), \tag{9}$$

$$\frac{d}{dt}\frac{dz}{dp} = \frac{\partial f}{\partial z}\frac{dz}{dp} + \frac{\partial f}{\partial p}, \tag{10}$$

where there exists one sensitivity $\frac{dz}{dp}$ for each parameter. Thus, given k state equations and m parameters, the total size of the ODE system is $\mathcal{O}(km)$. In contrast, the adjoint method works by only solving the original ODE forward and solves a system of size $\mathcal{O}(m+k)$ in reverse, greatly reducing the computational cost when the size of the ODE and the parameters is sufficiently large.

However, the total computational cost for implicit methods is based on the cube of the system size, meaning standard forward-mode sensitivities scale as $\mathcal{O}(k^3m^3)$ while the adjoint method of Chen *et al.*[4] or Serban and Hindmarsh[33] scales as $\mathcal{O}((k+m)^3)$. Given the large number of parameters in a neural ODE, the cubic scaling adds a tremendous computational cost. To mitigate this issue, we developed three separate strategies. For the first, note that many solvers contain a scheme known as dense output for generating a high order interpolation by reusing the internal computations of the steps.[34] This interpolation scheme in many cases requires zero additional f calculations by the ODE solver to construct and thus is computationally efficient. However, because it uses the intermediate steps for the calculation, it requires an increase in the memory by s for an s-stage Runge–Kutta or Rosenbrock method.[35] Using the dense output schemes, we can instead calculate Eq. (6) by sequentially solving $z' = f(z, \theta, t)$ forward, then solving $\omega' = -\omega^T\frac{\partial f}{\partial z}$ in reverse [using the dense output of $z(t)$ for any Jacobian computation], and finally solve $\frac{dL}{d\theta} = \int_{t_0}^{t_1}\omega^T(t)\frac{\partial f}{\partial \theta}dt$ independently using the dense output for $\omega(t)$. Using quadrature techniques such as Clenshaw–Curtis can converge exponentially fast, using less steps than the ODE solver. Thus, it potentially further decreases the computational cost of this step of the calculation, which is $\mathcal{O}(m)$. However, most importantly, this splits the computation into two stiff ODE solves matching the size of z plus an explicit quadrature matching the size of θ. Because of the cubic cost scaling of LU-factorizations in a stiff ODE solve, this trades computation for memory and brings the cost down to $\mathcal{O}(k^3+m)$. We termed this the `QuadratureAdjoint` method.

A core fact that is exploited in that formulation is that the final equation is explicit and thus does not necessarily have similar stability properties to the forward or adjoint equations. It would only be stiff if the Jacobian of $\omega^T(t)\frac{\partial f}{\partial \theta}$ is ill-conditioned, which does not necessarily follow from the conditioning of $\frac{\partial f}{\partial z}$. Thus, another formulation that could reduce the memory cost while achieving a similar effect would be to use an implicit–explicit (IMEX) ODE solver. IMEX ODE solvers allow for solving the differential equation as $f = f_i + f_e$, where f_i is treated implicitly, while f_e is treated explicitly. In this sense, Eq. (6) could be split so that

$$f_i = -\begin{bmatrix}\omega^T & 0\end{bmatrix}\begin{bmatrix}\frac{\partial f}{\partial z} & 0\\ 0 & 0\end{bmatrix}, \tag{11}$$

$$f_e = -\begin{bmatrix}0 & \omega^T\end{bmatrix}\begin{bmatrix}0 & \frac{\partial f}{\partial \theta}\\ 0 & 0\end{bmatrix}. \tag{12}$$

As an IMEX solver only requires factorizing the Jacobian of the f_i term, specializing the factorization on the block structure reduces the LU-factorization cost to $\mathcal{O}(k^3)$, and factoring in the linear cost of the explicit Runge–Kutta handling, we arrive at another $\mathcal{O}(k^3+m)$ scheme but with reduced peak memory requirements.

These two approaches highlight that the main difficulty is due to the inclusion of m parameter derivatives into the stiff solve. Thus, the last way to achieve a similar effect is to simply do any of the aforementioned schemes but only calculate the derivative of $l \ll m$ parameters at a time. In this case, one would need to repeat the adjoint solve $\frac{m}{l}$ times. However, because each solve has a cubic computational cost with respect to the states, this form of splitting brings the complexity down to $\mathcal{O}(mk^3l^2)$ for forward sensitivities and $\mathcal{O}(\frac{m}{l}(k+l)^3)$ for adjoint. Note that this technique also applies to direct differentiation via forward and reverse automatic differentiation of the solver. Additionally, the separate differentiation passes can all be solved in parallel, further reducing the real-world compute cost. This is the technique that we used to make the experiments of Sec. IV computationally viable.

C. Equation scaling

Armed with computationally stable and efficient gradient calculations, we noticed that these techniques were still not enough to stabilize the training process. To further tackle the difficulties in training stiff neural ODEs introduced by scale separation, we propose to normalize the neural network outputs and the loss components for different species. Specifically, we learn the following normalized form of neural ODEs:

$$\begin{aligned}\frac{dy(t)}{dt} &= NN\left(y(t), t\right)\frac{y_{scale}}{t_{scale}},\\ y_{scale} &= y_{\max} - y_{\min},\\ t_{scale} &= t_1 - t_0,\end{aligned} \tag{13}$$

where y_{scale} is a vector that consists of the characteristic scales for each species and t_{scale} refers to the characteristic time scale. We further re-balance the loss components by normalizing the loss

functions as

$$L(\theta) = MAE\left(\frac{y(t)^{model}}{y_{scale}}, \frac{y(t)^{obs}}{y_{scale}}\right). \tag{14}$$

IV. EXPERIMENTS

A. ROBER problem

Robertson's equations, denoted as ROBER, are one of the prominent stiff ODEs that model a reaction network with three chemicals.[15] The species concentrations $[y_1, y_2, y_3]$ are governed by Eq. (15) with reaction rate constants of $k_1 = 0.04$, $k_2 = 3 \times 10^7, k_3 = 10^4$,

$$\begin{aligned} \frac{dy_1}{dt} &= -k_1 y_1 + k_3 y_2 y_3, \\ \frac{dy_2}{dt} &= k_1 y_1 - k_2 y_2^2 - k_3 y_2 y_3, \\ \frac{dy_3}{dt} &= k_2 y_2^2. \end{aligned} \tag{15}$$

1. Baseline model

We first study baseline neural ODEs to elucidate the challenges of training stiff neural ODEs. The baseline model consists of one hidden layer with 50 hidden nodes and a hyperbolic tangent activation function, which is similar to the demo code in DiffEqFlux.jl and torchdiffeq. The stiff ODE integrator `Rosenbrock23`, an Order 2/3 L-Stable Rosenbrock-W method, is employed, and we hybrid Rosenbrock23 with Tsit5 via auto-switching to accelerate the integration and training. ForwardDiff.jl is adopted for the backpropagation, which we found is more efficient than adjoint methods in the current case due to the small number of states. The training data are generated by integrating Eq. (15) with the initial conditions of $[y_1, y_2, y_3] = [1, 0, 0]$ and a time span of $[10^{-5}, 10^5]$. The 50 sample data points are uniformly spaced in a logarithmic time scale. The neural ODE is trained with the ADAM[36] optimizer with a learning rate of 0.005. The training ended at 10 000 epochs where the loss function flatlines, as shown in Fig. 4.

Figure 5 shows the predictions of the learned baseline model. The baseline model failed to learn the actual dynamics and consequently failed to reproduce the species profiles. In addition, the gradient norm highly fluctuates and training losses do not decrease, which implies training instability (Fig. 4). We hypothesize that the failure is attributed to the time scale separations and stiffness induced scale separations in the neural network outputs as well as the imbalance of different loss function components. Specifically, after the initial short induction period, the changes of y_1 and y_3 are in the order of unity while y_2 in the order of 10^{-5}, which implies that one has to approximate such widely separated scales in a single neural network. In addition, since the magnitude of y_2 is 5 orders of magnitude smaller than y_1 and y_3, such imbalance in the three loss components could lead to gradient pathologies, which was anticipated that stochastic gradient descent is unstable in the presence of such gradient pathologies.[14]

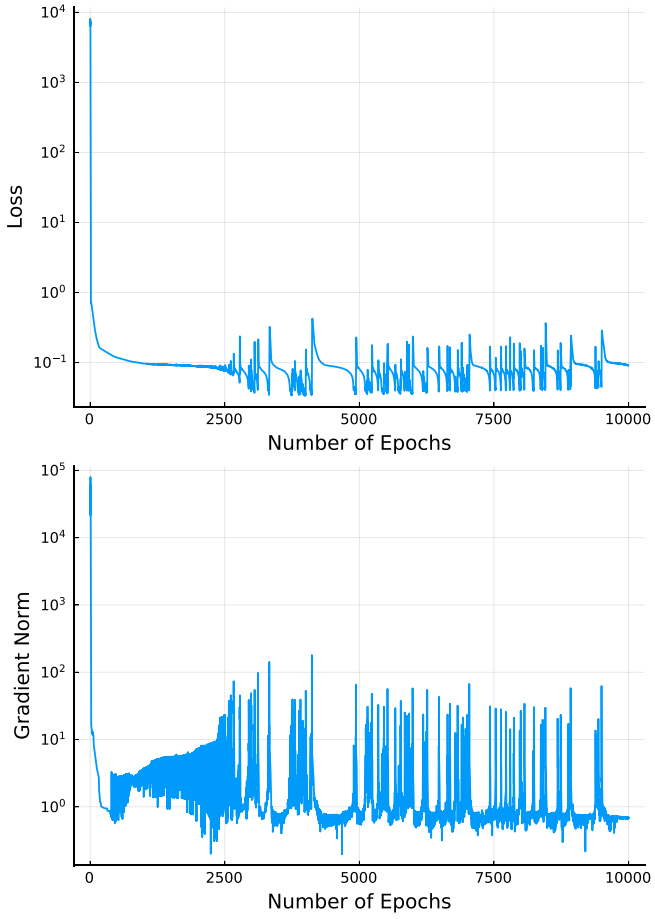

FIG. 4. Training of the baseline model. Top: Loss profile along epochs. Bottom: Gradient norm along epochs.

2. Mitigation of gradient pathologies via scaling

Figure 6 shows the results of a successful learning of stiff ROBER problems with equation scaling. The neural networks consist of six hidden layers and five nodes per layer, with an activation function of GELU.[37] GELU was chosen to mitigate potential issues with vanishing gradients typically seen in recurrent neural networks with saturating activation functions.[38] Figure 7 shows the history of loss functions and the L_2 norm of the gradient. The loss functions steadily decrease, and the gradient norm is stable during the training. The training in a workstation using a single CPU processor took 2 h.

We carried out sensitivity analyses to better understand the importance of proposed techniques, i.e., the equation scaling and the deep networks, on the training of stiff ODEs. Our test shows that scaling is essential for training stiff neural ODEs, as the training with the same neural network structure but without equation scaling failed to learn the ROBER problem. Regarding the neural network

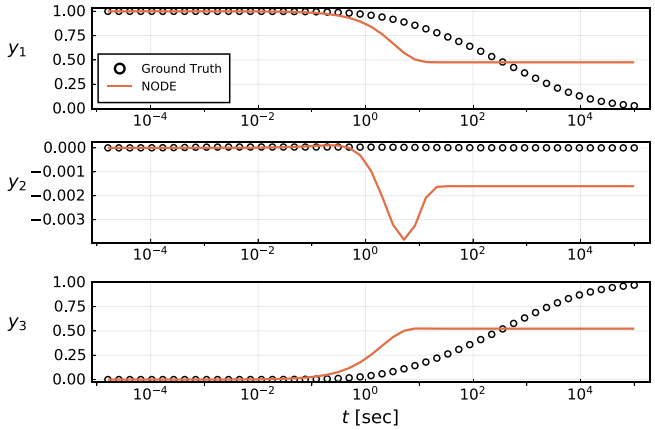

FIG. 5. The ground truth data (symbols) generated using the ROBER model (solid line) and the predictions of the learned baseline model.

structure, we found that shallow neural networks with the hyperbolic tangent activation function (tanh) can also successfully learn the ROBER problem but takes a longer training time, which could be attributed to the potential gradient issues. The results of the training without equation scaling and the training with the shallow network are provided in the supplementary material.

We can also accelerate the training with temporal regularization[10] and annealing of learning rates, although such techniques were not required for successful learning of stiff neural ODEs. The temporal regularization, STEER,[10] proceeds as randomly truncate the training to an integration time ahead of the entire time-space and in such a way introduce stochastic into the optimization to accelerate the training. The annealing of a learning rate is a widely adopted technique for accelerating the training of neural networks at later stages.

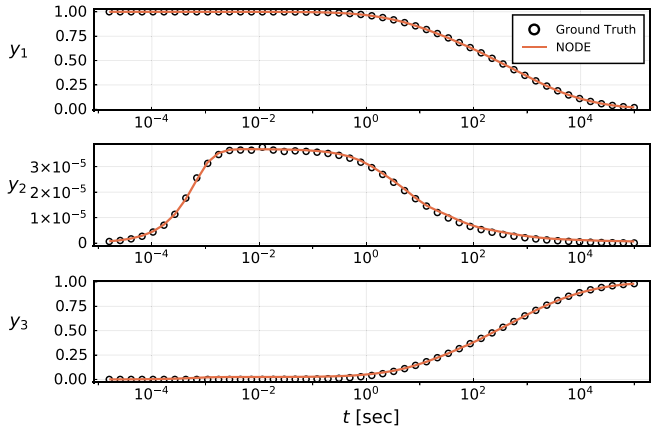

FIG. 6. Ground truth of Robertson equations (circle mark) and results of the scaled neural ordinary differential equation with six hidden layers of five nodes and GELU activation function (orange line).

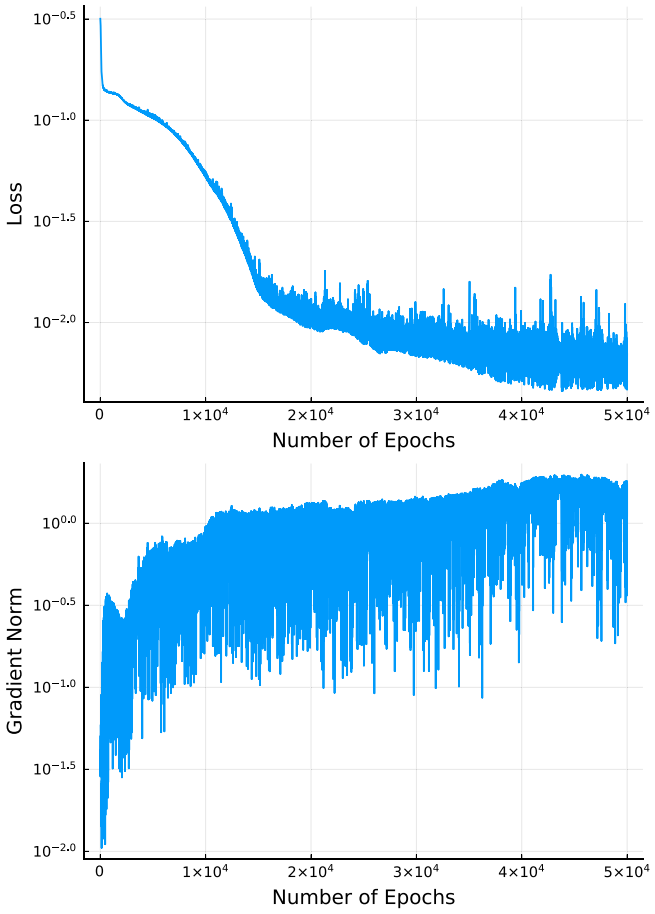

FIG. 7. Training of a scaled neural ODE. Top: Loss profile along epochs. Bottom: Gradient norm along epochs.

B. POLLU problem

To show the generality of the technique, we demonstrate the effectiveness of the proposed methods in the POLLU problem, which is more complex than the ROBER problem and consists of 20 species and 25 reactions. The POLLU is an air pollution model developed at The Dutch National Institute of Public Health and Environmental Protection. It can be described mathematically by the following 20 non-linear ODEs shown in Eq. (16), and full details about the model can be found in Ref. 16 and the supplementary material,

$$\frac{dy(t)}{dt} = f\big(y(t)\big), \qquad y(0) = y_0, y \in \mathbb{R}^{20}, 0 \le t \le 60. \tag{16}$$

The neural ODEs consist of three hidden layers and ten nodes per hidden layer. The label data are sampled uniformly in $[0, 60]$ on a linear scale. The rest of the hyper-parameter settings are the same as the ones for the ROBER problem. Figure 8 presents the label

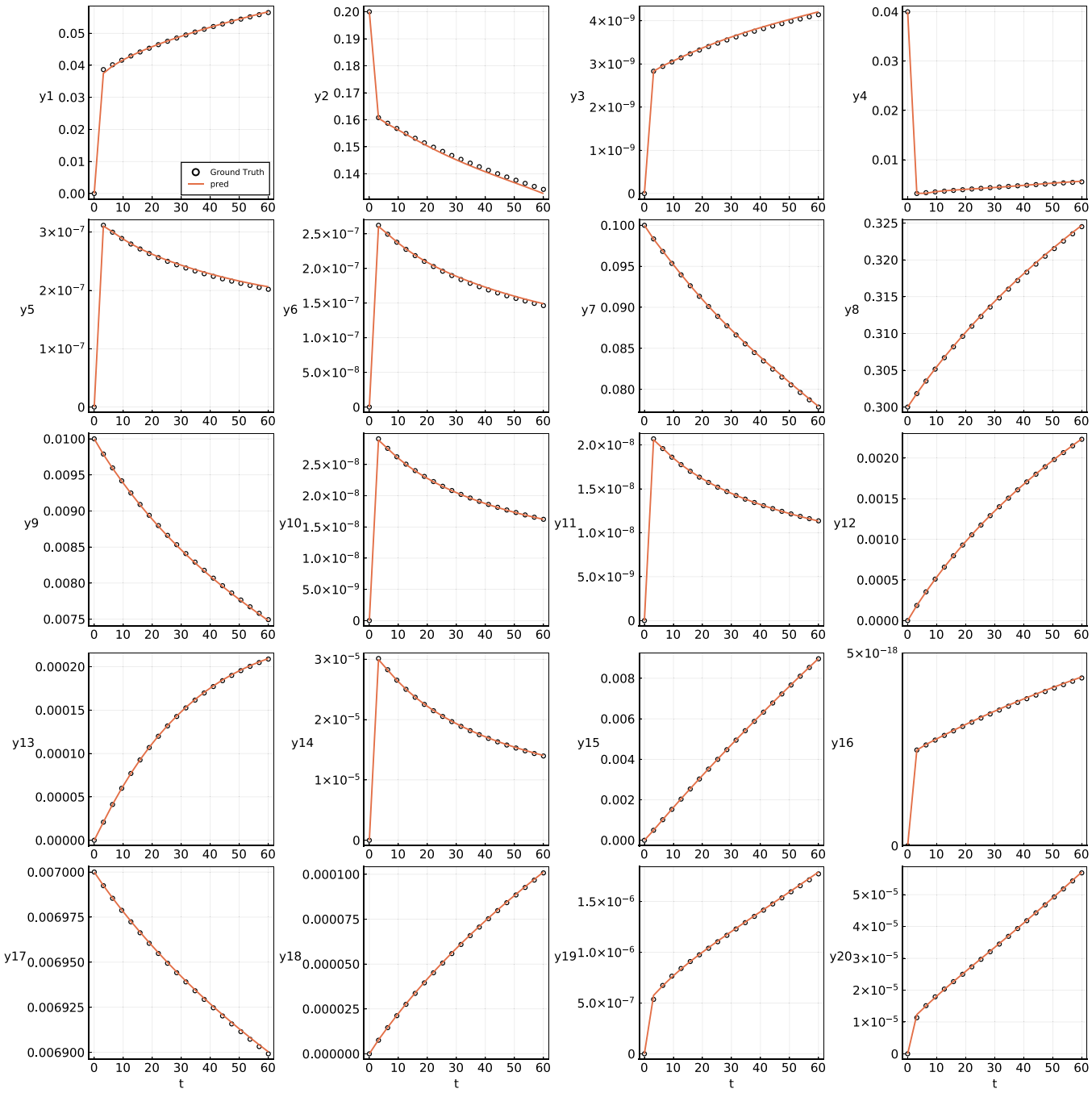

FIG. 8. Ground truth of an air pollution model POLLU (symbols) and results of the scaled neural ordinary differential equation with three hidden layers of ten nodes and GELU activation function (solid line).

data generated with Eq. (16) and the predictions of the learned neural ODEs. Note that the baseline model again failed for the POLLU problem similarly to the ROBER problem, and it is presented in the supplementary material. The trained neural ODEs can capture both the dynamics during the induction period and the near-equilibrium states. It also well captures both the dynamics of the major species, such as y_1, y_2, with relatively large concentration and slow time scales, and the dynamics of transient intermediate species, such as

y_3, y_{10}, with relatively low concentrations and fast time scales. Therefore, the demonstration of the POLLU problem further shows the robustness of proposed approaches in learning stiff neural ODEs where previous techniques had failed.

V. CONCLUSION AND DISCUSSION

This work proposed new derivative calculation techniques for the stability of stiff ODE systems while reducing the computational complexity. It is coupled with equation scaling and deeper neural networks with rectified activation functions to learn neural ODEs in stiff systems. The effectiveness of the proposed approaches is demonstrated in two classical stiff systems: the ROBER and the POLLU. While the baseline models even failed to learn the overall trend of the dynamics, the proposed approaches accurately reproduced all of the species profiles including both species with slow and fast time scales.

While we have shown the effectiveness of the proposed techniques, there is a demand for further theoretical analysis of the challenges of learning stiff neural ODE, such as how stiffness affects the gradient dynamics during training and leads to training failures. In addition, more theoretical analysis on equation scaling relations to the gradient flow of neural ODEs would better elucidate why it was important in the training process. It is also interesting to explore other equation scaling and neural network scaling techniques, such as scaling the neural network inputs and batch-normalization.[39] Given the first order optimizers are solving an ordinary differential equation on the gradient flow themselves, one likely could define the stiffness index on the Jacobian of the gradient of the optimization, which is the Hessian with respect to the parameters. Handling the optimization itself with stiffly-aware methods could be beneficial in cases where gradient pathologies are not mitigated.

Importantly, this paper demonstrates the impact of errors in the continuous adjoint handling on the training of a neural ODE. Such an analysis could be required in other "implicit layer" methods as well. For example, we referenced how a nonlinear solver destabilizes when its Jacobian exhibits ill-conditioning, equivalent to stiffness in the ODE system when viewing it as a solver to steady state.[25] This would suggest that on highly stiff systems, methods such as the DEQ,[40] similar behavior would apply to the forward pass. However, importantly, similar instabilities in the adjoint problem likely occur due to the reliance on the adjoint's assumption that $G(x) = 0$ exactly, which is more difficult to satisfy as the condition number rises. Likely, similar stabilization may be required for this case. Similarly, differentiable optimization as a neural network layer[41,42] requires satisfaction of the KKT equations for its adjoint, which are likely to be less satisfied when the Hessian of the optimization problem is ill-conditioned, therefore likely requires a similar stability analysis.

We end by noting that such developments in handling highly stiff equations can allow for machine learning architectures that satisfy arbitrary constraint equations during their evolution. Rackauckas *et al.*[1] proposed using differential-algebraic equations (DAEs) in a mass-matrix form for this purpose; i.e.,

$$Mz' = f(z, p, t), \tag{17}$$

where M is singular. For example, it is well-known that the Rosenbrock equation can be written in its DAE form,

$$\frac{dy_1}{dt} = -k_1 y_1 + k_3 y_2 y_3, \tag{18}$$

$$\frac{dy_2}{dt} = k_1 y_1 - k_2 y_2^2 - k_3 y_2 y_3, \tag{19}$$

$$0 = y_1 + y_2 + y_3 - 1, \tag{20}$$

where the mass matrix is of the $M = [100; 010; 000]$. Similarly, an arbitrary system constrained to have dynamical states sum to 1 could be imposed by

$$\frac{d[y_1, y_2]}{dt} = NN(y), \tag{21}$$

$$0 = y_1 + y_2 + y_3 - 1, \tag{22}$$

and more generally,

$$\frac{dy}{dt} = NN([y, z]), \tag{23}$$

$$0 = g(y, z, t), \tag{24}$$

the constraint equation can be set or fit using domain information as necessary. While currently able to be specified and trained in differentiable DAE solver systems, we noticed fitting instabilities similar to stiff neural ODEs arise in such cases. Indeed, Wanner and Hairer[25] notes that DAEs are a form of infinite stiffness, being posed as the limit of singularly perturbed stiff ODEs. Petzold[43] famously showcased how "DAEs are not ODEs," showing how they exhibit additional behavior that must be appropriately treated; thus, an analysis of such systems will be required for fully stabilizing neural network DAE formulations including the relationship of training techniques to a differential index.

SUPPLEMENTARY MATERIAL

See the supplementary material for the sensitivity analysis to the neural network structures and the details in POLLU equations.[16]

ACKNOWLEDGMENTS

The information, data, or work presented herein was funded in part by the Advanced Research Projects Agency-Energy (ARPA-E), U.S. Department of Energy, under Award No. DE-AR0001222. The views and opinions of the authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DATA AVAILABILITY

The data that support the findings of this study are openly available in GitHub at https://github.com/DENG-MIT/StiffNeuralODE, Ref. 44.

REFERENCES

[1] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan, and A. Edelman, "Universal differential equations for scientific machine learning," arXiv:2001.04385 (2020).

[2] W. Ji and S. Deng, "Autonomous discovery of unknown reaction pathways from data by chemical reaction neural network," arXiv:2002.09062 (2020).

[3] Y. Rubanova, R. T. Chen, and D. Duvenaud, "Latent ordinary differential equations for irregularly-sampled time series," in *33rd Conference on Neural Information Processing Systems* (Curran Associates, Inc., 2019), available at https://papers.nips.cc/paper/2019/hash/42a6845a557bef704ad8ac9cb4461d43-Abstract.html.

[4] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," arXiv:1806.07366 (2018).

[5] C. Rackauckas, M. Innes, Y. Ma, J. Bettencourt, L. White, and V. Dixit, "DiffEqFlux.jl—A Julia library for neural differential equations," arXiv:1902.02376 (2019).

[6] A. Bills, S. Sripad, W. L. Fredericks, M. Guttenberg, D. Charles, E. Frank, and V. Viswanathan, "Universal battery performance and degradation model for electric aircraft," arXiv:2008.01527 (2020).

[7] G. D. Portwood, P. P. Mitra, M. D. Ribeiro, T. M. Nguyen, B. T. Nadiga, J. A. Saenz, M. Chertkov, A. Garg, A. Anandkumar, A. Dengel *et al.*, "Turbulence forecasting via neural ode," arXiv:1911.05180 (2019).

[8] R. Maulik, A. Mohan, B. Lusch, S. Madireddy, P. Balaprakash, and D. Livescu, "Time-series learning of latent-space dynamics for reduced-order model closure," Physica D **405**, 132368 (2020).

[9] O. Owoyele and P. Pal, "A neural ordinary differential equations approach for chemical kinetics solvers," arXiv:2101.04749 (2020).

[10] A. Ghosh, H. S. Behl, E. Dupont, P. H. Torr, and V. Namboodiri, "Steer: Simple temporal regularization for neural odes," arXiv:2006.10711 (2020).

[11] R. Anantharaman, Y. Ma, S. Gowda, C. Laughman, V. Shah, A. Edelman, and C. Rackauckas, "Accelerating simulation of stiff nonlinear systems using continuous-time echo state networks," arXiv:2010.04004 (2020).

[12] C. Huang, C. R. Wentland, K. Duraisamy, and C. Merkle, "Model reduction for multi-scale transport problems using structure-preserving least-squares projections with variable transformation," arXiv:2011.02072 (2020).

[13] W. Ji, W. Qiu, Z. Shi, S. Pan, and S. Deng, "Stiff-pinn: Physics-informed neural network for stiff chemical kinetics," arXiv:2011.04520 (2020).

[14] S. Wang, Y. Teng, and P. Perdikaris, "Understanding and mitigating gradient pathologies in physics-informed neural networks," arXiv:2001.04536 (2020).

[15] H. Robertson, "The solution of a set of reaction rate equations," in *Numerical Analysis: Introduction* (Thompson, 1966), pp. 178–182.

[16] J. G. Verwer, "Gauss–Seidel iteration for stiff ODEs from chemical kinetics," SIAM J. Sci. Comput. **15**, 1243–1250 (1994).

[17] M. Hoffmann, C. Fröhner, and F. Noé, "Reactive SINDy: Discovering governing reactions from concentration data," J. Chem. Phys. **150**, 025101 (2019).

[18] B. Yuan, C. Shen, A. Luna, A. Korkut, D. S. Marks, J. Ingraham, and C. Sander, "CellBox: Interpretable machine learning for perturbation biology with application to the design of cancer combination therapy," Cell Syst. **12**, 128–140 (2020).

[19] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," Proc. Natl. Acad. Sci. U.S.A. **113**, 3932–3937 (2016).

[20] N. M. Mangan, S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Inferring biological networks by sparse identification of nonlinear dynamics," IEEE Trans. Mol. Biol. Multi-Scale Commun. **2**, 52–63 (2016).

[21] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," SIAM Rev. **59**, 65–98 (2017).

[22] C. Rackauckas and Q. Nie, "DifferentialEquations.jl—A performant and feature-rich ecosystem for solving differential equations in Julia," J. Open Res. Softw. **5**, 15 (2017).

[23] M. Innes, "Flux: Elegant machine learning with Julia," J. Open Source Softw. **3**, 602 (2018).

[24] J. Revels, M. Lubin, and T. Papamarkou, "Forward-mode automatic differentiation in Julia," arXiv:1607.07892 (2016).

[25] G. Wanner and E. Hairer, *Solving Ordinary Differential Equations II* (Springer, Berlin, 1996), Vol. 375.

[26] L. F. Shampine and C. W. Gear, "A user's view of solving stiff ordinary differential equations," SIAM Rev. **21**, 1–17 (1979).

[27] L. F. Shampine and S. Thompson, "Stiff systems," Scholarpedia **2**, 2855 (2007).

[28] A. Gholami, K. Keutzer, and G. Biros, "ANODE: Unconditionally accurate memory-efficient gradients for neural ODEs," arXiv:1902.10298 (2019).

[29] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward, "SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers," ACM Trans. Math. Softw. **31**, 363–396 (2005).

[30] C. Rackauckas, A. Edelman, K. Fischer, M. Innes, E. Saba, V. B. Shah, and W. Tebbutt, "Generalized physics-informed learning through language-wide differentiable programming," in *AAAI Spring Symposium: MLPS 2020* (AAAI, 2020).

[31] H. Zhang and A. Sandu, "FATODE: A library for forward, adjoint, and tangent linear integration of ODEs," SIAM J. Sci. Comput. **36**, C504–C523 (2014).

[32] J. Liesen and P. Tich`y, "Convergence analysis of Krylov subspace methods," GAMM Mitt. **27**, 153–173 (2004).

[33] R. Serban and A. C. Hindmarsh, "CVODES: An ODE solver with sensitivity analysis capabilities," Technical Report No. UCRL-JP-200039, Lawrence Livermore National Laboratory, 2003.

[34] L. F. Shampine and L. O. Jay, "Dense output," in *Encyclopedia of Applied and Computational Mathematics* (Springer, Berlin, 2015), pp. 339–345.

[35] Note that many methods do not need to store all stages. Additionally, some higher order Runge–Kutta methods require few additional f evaluations in order to generate an interpolation matching the order of the solver and thus can default to interpolants of an order less or more. For a full discussion of dense output, see Ref. 34.

[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv:1412.6980 (2014).

[37] D. Hendrycks and K. Gimpel, "Gaussian error linear units (GELUs)," arXiv:1606.08415 (2016).

[38] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International Conference on Machine Learning* (PMLR, 2013), pp. 1310–1318.

[39] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning* (PMLR, 2015), pp. 448–456.

[40] S. Bai, J. Z. Kolter, and V. Koltun, "Deep equilibrium models," arXiv:1909.01377 (2019).

[41] B. Amos and J. Z. Kolter, "OptNet: Differentiable optimization as a layer in neural networks," in *International Conference on Machine Learning* (PMLR, 2017), pp. 136–145.

[42] B. Amos, "Differentiable optimization-based modeling for machine learning," Ph.D. thesis (Carnegie Mellon University, 2019).

[43] L. Petzold, "Differential/algebraic equations are not ODE's," SIAM J. Sci. Stat. Comput. **3**, 367–384 (1982).

[44] S. Kim, W. Ji, S. Deng, Y. Ma, and C. Rackauckas, "Stiff neural ordinary differential equations," (2021), available at see https://github.com/DENG-MIT/StiffNeuralODE.