Neural Ordinary Differential Equations

Prithvi Dake
Department of Chemical Engineering, University of California, Santa Barbara

March 20, 2025
ME255NN: Neural Networks for modeling, optimization and control
Winter 2025

1 Introduction

Conventional neural network architectures, such as recurrent neural networks (RNNs) and residual networks (ResNets), can be viewed as discrete approximations of ordinary differential equations (ODEs), specifically resembling the explicit Euler scheme. Neural ODEs (NODEs) extend these architectures into a continuous framework, providing a more principled way of modeling dynamical systems.

One major challenge in training NODEs is gradient propagation during backpropagation. Unlike traditional architectures, where intermediate states are stored for gradient computation, NODEs require solving an augmented ODE system, involving adjoint sensitivity equations, to compute gradients efficiently. This adjoint sensitivity method significantly reduces memory requirements compared to brute-force backpropagation & forward sensitivity equations. Overcoming this training limitation enables NODEs to be applied in broader contexts, such as continuous normalizing flows and state-space models.

In this review, we cover fundamental concepts and recent advancements in NODEs. We begin with a discussion on artificial neural networks (ANNs) implemented in a physics-informed manner that does not involve recurrence or differential modeling as shown by Raissi et al. (2019). Next, we explore discrete data-driven models for process systems using ResNets and RNNs. Building on these foundations, we introduce sensitivity and adjoint equation methods referring Stapor et al. (2018), and then introduce the formal development of NODEs by Chen et al. (2018).

We then review advanced applications, including generative latent models by Rubanova et al. (2019), continuous normalizing flows by Grathwohl et al. (2018), and robustness analyses of NODEs by Yan et al. (2019). Finally, we discuss efficient NODE training techniques compiled in Finlay et al. (2020).

The structure of this report is as follows: We first introduce conventional neural network architectures, such as RNNs and ResNets, and their connection to NODEs. Next, we explain how backpropagation is handled in NODEs and discuss applications in continuous normalizing flows and NODE-based state-space models. Most of the sections give exemplar implementations. The report concludes with a summary of key findings and an overview of ongoing developments in the field.
Notation: We keep our notation consistent with the one used in Rawlings et al. (2020).

2 Background

2.1 ANNs for Physics-Informed Learning

In most engineering systems, we only have access to noisy measurements of input-output data, typically originating from high-dimensional, nonlinear dynamic systems. While such systems can be approximated using finite impulse response (FIR) or autoregressive exogenous (ARX) models (Ljung, 1999), these approaches often fail to respect fundamental physical laws. One way to address this limitation is by penalizing violations of these laws—a key idea behind physics-informed neural networks (PINNs), as introduced by Raissi et al. (2019).

However, it is important to view PINNs not merely as data-driven models but rather as a solution strategy for PDEs and ODEs. Fundamentally, they operate as *methods of weighted residuals* (Villadsen and Stewart, 1967)—specifically, collocation schemes—where the residual is minimized at selected collocation points. While PINNs can be used to learn certain model parameters, their results can be difficult to interpret without a fundamental understanding of the underlying physics.

Recent research has leveraged the PINN framework to enforce simple mass and energy balance equations at steady-state, as this is often the only available information in practical scenarios. Algorithm 1 outlines the basic algorithm for training a PINN applied to an ODE system.

Algorithm 1 Training a Physics-Informed Neural Networks (PINNs)

```
1: **Input:** Neural network architecture $f_{NN}$
2: **Physical Model:** $Lx = f \quad B_1 x = 0 \quad B_2 x = 0$                ▷ True or known physics
3: **Initialize** neural network parameters $\theta$
4: **repeat**
5:     $x = f_{NN}(x, t; \theta)$
6:     $V_{ODE} = \|Lx - f\|^2$                                 ▷ Prediction error minimization
7:     $V_{BC} = \|B_1 x\|^2 + \|B_2 x\|^2$                        ▷ Enforce boundary conditions
8:     $V = \lambda_1 V_{PDE} + \lambda_2 V_{BC}$                            ▷ Combine loss terms
9:     $\theta \leftarrow \theta - \eta \nabla_\theta V$                        ▷ Update $\theta$ using gradient descent
10: **until** convergence criteria is met
11: **Return** trained model $f_{NN}$
```

2.2 ResNets and RNNs for Process Systems

Most engineering systems operating near a steady state can be effectively modeled using discrete linear time-invariant (DLTI) systems. These state-space models are particularly well-suited for systems experiencing small perturbations. For highly nonlinear systems, ResNets and RNNs offer a similar state-space modeling approach. In the system identification community, ResNet and RNN models are commonly employed for this purpose, as illustrated in (1) on left and right respectively.

$$\begin{aligned} x^+ &= x + f_{NN}(x, u; \theta) \\ y &= g_{NN}(x, u; \theta) \end{aligned} \qquad \begin{aligned} x^+ &= f_{NN}(x, u; \theta) \\ y &= g_{NN}(x, u; \theta) \end{aligned} \tag{1}$$

There are two approaches to learning the parameters θ in (1): *multi-step prediction error minimization* and *single-step prediction error minimization*. The single-step approach minimizes the prediction error at each time step independently, while the multi-step approach considers the accumulated error over multiple time steps. In practice, the multi-step method tends to be more stable, especially in the presence of noisy measurements.

Consider following *true* model of linear chemical kinetics taking place in a well-mixed batch reactor with $c = (c_A, c_B, c_C)^T$ and initial concentration c_0:

$$\mathrm{A} \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} 2\mathrm{B}, \qquad \mathrm{B} \xrightarrow{k_2} \mathrm{C}$$

$$\begin{bmatrix} \dot{c}_A \\ \dot{c}_B \\ \dot{c}_C \end{bmatrix} = \begin{bmatrix} -k_1 & k_{-1} & 0 \\ 2k_1 & 2k_{-1} - k_2 & 0 \\ 0 & k_2 & 0 \end{bmatrix} \begin{bmatrix} c_A \\ c_B \\ c_C \end{bmatrix} \tag{2}$$

Assume we have full-state measurement i.e.we measure all the three concentrations. Thus, a ResNet model can be easily trained to predict the concentration profile over time as shown in figure 1. A similar example can be reproduced using an RNN model. Note we used multi-step prediction error minimization. One disadvantage of such iterative models is that they require large memory to store intermediate states for gradient propagation (since we basically apply chain-rule while differentiating through the solution). Another disadvantage is they are discrete in nature and thus, can struggle when the data is not sampled at regular intervals.

2.3 Forward and backward sensitivity/adjoint methods

2.3.1 Forward Sensitivity

Here we give a quick overview of both the forward and backward sensitivity/adjoint equations for ODEs. Consider following non-linear state-space model:

$$\dot{x} = f(x, u; \theta) \qquad x(0) = g(x_0; \theta)$$

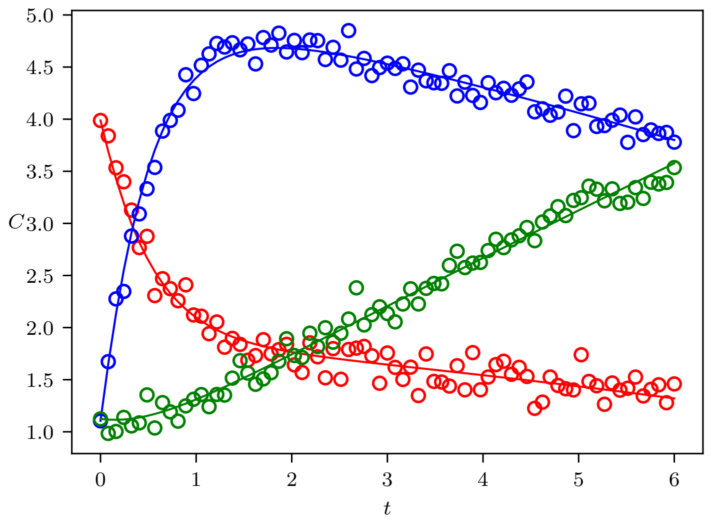

Fig. 1: Concentration profiles of species A, B, and C in a batch reactor. The discrete ResNet model does quite a good job to predict the concentration profiles.

We assume full state measurement (i.e. $y = x$). Consider a least-squares objective to measure the fit to the data ($\tilde{x}$):

$$\min_{\theta} V(x) = \|\tilde{x} - x\|^2 \tag{3}$$

We refer Rawlings and Ekerdt (2020) for following derivation. To propagate the gradient we need to solve the sensitivity equations:

$$\begin{aligned} \frac{\partial V}{\partial \theta^T} &= -2(\tilde{x} - x)^T \frac{\partial x}{\partial \theta^T} \\ S &= \frac{\partial x}{\partial \theta^T} \end{aligned} \tag{4}$$

where S is the *sensitivity* matrix of the model solutions with parameters. We differentiate S to obtain:

$$\frac{dS}{dt} = \frac{d}{dt}\left(\frac{\partial x}{\partial \theta^T}\right) = \frac{\partial}{\partial \theta^T}\left(\frac{dx}{dt}\right) = \frac{\partial}{\partial \theta^T} f \tag{5}$$

Using chain rule on f, we get following matrix equation:

$$\frac{dS}{dt} = \frac{\partial f}{\partial x^T} S + \frac{\partial f}{\partial \theta^T} \qquad S(0) = \frac{\partial g}{\partial \theta^T} \tag{6}$$

Thus, we can construct following augmented ODE system:

$$\begin{bmatrix} \dot{x} \\ \dot{S} \end{bmatrix} = \begin{bmatrix} f \\ \dfrac{\partial f}{\partial x^T} S + \dfrac{\partial f}{\partial \theta^T} \end{bmatrix} \qquad \begin{bmatrix} x(0) \\ S(0) \end{bmatrix} = \begin{bmatrix} x_0 \\ \dfrac{\partial g}{\partial \theta^T} \end{bmatrix} \tag{7}$$

This augmented system can be solved using any ODE solver, allowing gradient propagation. However, if $\theta \in \mathbb{R}^p$ and $x \in \mathbb{R}^n$, we need to solve $S \in \mathbb{R}^{n+p}$ equations i.e. the cost increases with p. Thus, forward sensitivity is not an efficient method to propagate gradients. However, we can circumvent the problem of linear increase in sensitivity equations using backward adjoint equations. This key advantage of backward adjoint method contributed to the feasibility of Neural ODEs (NODEs), beyond their interpretation as a continuous counterpart to Euler-step based architectures like ResNets. We next show a short example on solving the augmented ODE system for forward sensitivity. Consider, following kinetics:

$$\begin{aligned} \dot{c}_A &= -k c_A \\ c_A(0) &= c_{A0} \end{aligned} \tag{8}$$

Say we want to calculate sensitivities wrt k and c_{A0}. With reference to (7), for $S_1 = \dfrac{\partial c_A}{\partial k}$ and $S_2 = \dfrac{\partial c_A}{\partial c_{A0}}$, we have following augmented system:

$$\begin{bmatrix} \dot{c}_A \\ \dot{S}_1 \\ \dot{S}_2 \end{bmatrix} = \begin{bmatrix} -kc_A \\ -kS_1 - c_A \\ -kS_2 \end{bmatrix} \qquad \begin{bmatrix} c_A(0) \\ S_1(0) \\ S_2(0) \end{bmatrix} = \begin{bmatrix} c_{A0} \\ 0 \\ 1 \end{bmatrix} \tag{9}$$

If we were learning the parameter k and initial condition c_{A0}, we could show a gradient descent update as follows:

$$\begin{bmatrix} k \\ c_{A0} \end{bmatrix} \leftarrow \begin{bmatrix} k \\ c_{A0} \end{bmatrix} - \eta \begin{bmatrix} -2(\hat{c}_A - c_A)^T S_1 \\ -2(\hat{c}_A - c_A)^T S_2 \end{bmatrix} \tag{10}$$

2.3.2 Backward Sensitivity

In this section we redefine the objective function as an integral over time. The integral formulation allows us to derive the adjoint equation. We refer Sengupta et al. (2014) that gives a straightforward proof of the adjoint equation than the one in Chen et al. (2018). Consider following least-squares objective for (3) and the Lagrange functional:

$$\begin{aligned} \min_{\theta} V(x) &= \int_0^t \|x - \hat{x}\|^2 dt = \int_0^t g(x) dt \\ \min_{\theta,\lambda} \mathcal{L} &= \int_0^t g dt + \int_0^t \lambda(t)^T \left(f - \frac{dx}{dt} \right) dt \end{aligned} \tag{11}$$

Hence,

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \theta^T} &= \int_0^t \frac{\partial g}{\partial x^T} \frac{\partial x}{\partial \theta^T} dt + \int_0^t \lambda^T \left(\frac{\partial f}{\partial x^T} \frac{\partial x}{\partial \theta^T} + \frac{\partial f}{\partial \theta^T} - \frac{\partial}{\partial \theta^T} \frac{dx}{dt} \right) dt \\ \frac{\partial \mathcal{L}}{\partial \theta^T} &= \int_0^t \frac{\partial g}{\partial x^T} \frac{\partial x}{\partial \theta^T} dt + \int_0^t \lambda^T \left(\frac{\partial f}{\partial x^T} \frac{\partial x}{\partial \theta^T} + \frac{\partial f}{\partial \theta^T} - \frac{d}{dt} \frac{\partial x}{\partial \theta^T} \right) dt \\ \frac{\partial \mathcal{L}}{\partial \theta^T} &= \int_0^t \lambda^T \frac{\partial f}{\partial \theta^T} dt + \int_0^t \left(\frac{\partial g}{\partial x^T} + \lambda^T \frac{\partial f}{\partial x^T} - \lambda^T \frac{d}{dt} \right) \frac{\partial x}{\partial \theta^T} dt \end{aligned} \tag{12}$$

The term $\int_0^t \lambda^T \dfrac{d}{dt} \dfrac{\partial x}{\partial \theta^T} dt$ can be solved using integration by parts. Thus, we have:

$$\frac{\partial \mathcal{L}}{\partial \theta^T} = \int_0^t \lambda^T \frac{\partial f}{\partial \theta^T} dt + \int_0^t \left(\frac{\partial g}{\partial x^T} + \lambda^T \frac{\partial f}{\partial x^T} + \frac{d\lambda^T}{dt} \right) \frac{\partial x}{\partial \theta^T} dt + \lambda^T(0) \frac{\partial x}{\partial \theta^T}|_0 - \lambda^T(T) \frac{\partial x}{\partial \theta^T}|_t \tag{13}$$

Now, the whole objective of backward sensitivity is to avoid calculating $\dfrac{\partial x}{\partial \theta^T}$. Also note that the equality constraint in the Langrange functional is always satisfied since we *solve* the ODE system. Thus we get to choose λ such that the terms associated with $\dfrac{\partial x}{\partial \theta^T}$ vanish. Also, we can easily calculate $\dfrac{\partial x}{\partial \theta^T}|_0$ but we would like to set $\dfrac{\partial x}{\partial \theta^T}|_t = 0$. Thus, we can construct following adjoint equations:

$$\begin{aligned} \frac{d\lambda^T}{dt} &= -\frac{\partial g}{\partial x^T} - \lambda^T \frac{\partial f}{\partial x^T} \\ \lambda(T) &= 0 \end{aligned} \tag{14}$$

Referring to (13), we see $\dfrac{\partial V}{\partial \theta^T} = \dfrac{\partial \mathcal{L}}{\partial \theta^T}$. Chen et al. (2018) uses objective function of the form:

$$\min_{\theta} V(x) = \int_0^t \|x - \hat{x}\|^2 \delta(t) dt \tag{15}$$

Hence, the adjoint equations for their case are:

$$\begin{aligned}
\frac{d\lambda^T}{dt} &= -\lambda^T \frac{\partial f}{\partial x^T} \\
\lambda(t)^T &= -\frac{\partial g}{\partial x^T}|_t = -\frac{\partial V}{\partial x^T}|_t \\
\frac{\partial V}{\partial \theta^T} &= \lambda^T(0)\frac{\partial x}{\partial \theta^T}|_0 + \int_0^t \lambda^T \frac{\partial f}{\partial \theta^T} dt
\end{aligned} \tag{16}$$

Hence, $\lambda = \dfrac{\partial V}{\partial x}$ as we see from the terminal condition in (16). Thus, we need to solve the augmented ODE system once forward and once backward in each optimal iteration. We need a forward pass to store the value of $\dfrac{\partial g}{\partial x^T}|_t$ and x_t. Also, note that the number of adjoint equations do not increase with parameters (i.e $\lambda \in \mathbb{R}^n$ unlike $S \in \mathbb{R}^{n\times p+1}$). We can evaluate the gradient in one go as shown in (16). However, such method is not an ultimate panacea. Such systems are still vulnerable to the problems commonly faced by numerical ODE solvers like stiffness, accuracy, etc. One can analyze the stiffness of the adjoint ODE by simply inspecting the eignevalues of the Jacobian matrix $\dfrac{\partial f}{\partial x^T}$ at different time points. Sometimes we may end up with noisy gradients which may lead to poor convergence. Some recent remedies is solving adjoint equation in multiple time-intervals rather than a single long time-interval and storing intermediate points, often called as *checkpointing.* (Zhuang et al., 2020).

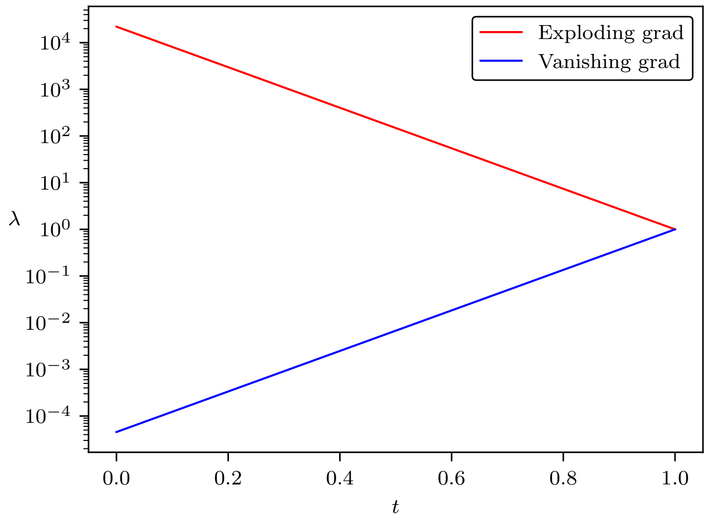

Fig. 2: Consider $\dot{\lambda} = -e\lambda$ for $e = \pm 10$. We see that the adjoint variable will either explode or vanish. Thus $\dfrac{\partial V}{\partial \theta} = \lambda(0) = 10^{\pm 5}$. A simple example to show how an ill-conditioned adjoint system can lead to poor convergence.

2.4 Neural Ordinary Differential Equations

With the background established in section 2.3, we can now introduce the algorithm for training NODEs. The key idea is to replace the right-hand side of the ODE system with a neural network. With auto-differentiation we can easily construct the augmented ODE system and solve it using any ODE solver.

We close the section with the same example as shown in (2) but now we use a NODE to predict the concentration profiles. The results are shown in figure 3.

2.5 Generative latent models based on continuous normalizing flows

The NODE paper applies the same motivation to develop continuous normalizing flows (Grathwohl et al., 2018). Interested reader can refer the paper for proof of theorem on instantaneous change of variables (Chen et al., 2018). The key idea is the neural ODE rhs is trained to conserve the probability density while maximizing log-likelihood to

Algorithm 2 Training a Neural Ordinary Differential Equation (NODE)

```
1:  Input: Neural network architecture $f_{NN}$
2:  ODE: $\begin{bmatrix}\dot{x}\\ \dot{\lambda}^T\end{bmatrix} = \begin{bmatrix} f_{NN} \\ -\lambda^T \frac{\partial f_{NN}}{\partial x^T}\end{bmatrix} \quad \begin{bmatrix}x(t)\\ \lambda(t)\end{bmatrix} = \begin{bmatrix} x_t \\ -\frac{\partial g}{\partial x^T}|_t\end{bmatrix}$    ▷ Construct the augmented ODE using PyTorch
3:  Initialize neural network parameters $\theta$
4:  repeat
5:      $V = \|\hat{x} - x\|^2$    ▷ Forward solve to get the loss $V$
6:      $\hat{x}_{aug} = \text{ODESolve}(f_{aug}, x_{aug}(0), T, t_0)$
7:      $\frac{\partial V}{\partial \theta^T} = \lambda^T(0)\frac{\partial x}{\partial \theta^T}|_0 + \int_0^t \lambda^T \frac{\partial f}{\partial \theta^T} dt$    ▷ Construct the gradient wrt loss
8:      $\theta \leftarrow \theta - \eta \nabla_\theta V$
9:  until convergence criteria is met
10: Return trained model $f_{NN}$
```

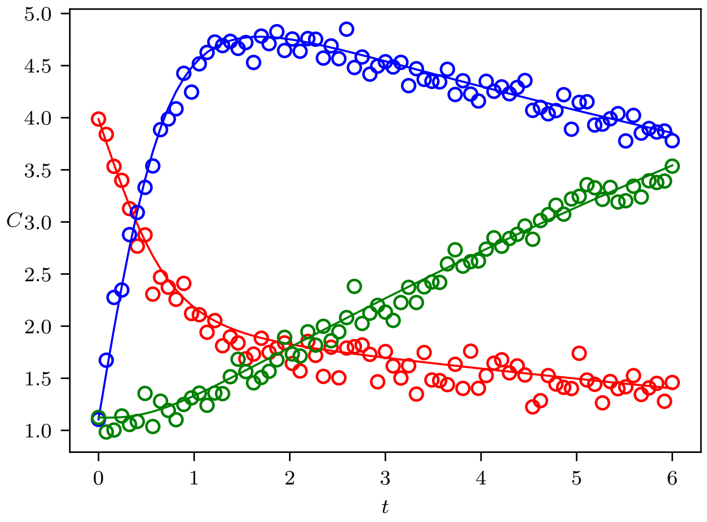

Fig. 3: Concentration profiles of species A, B, and C in a batch reactor. The NODE model also does quite a good job to predict the concentration profiles. In fact for such simple problem, we hardly see any difference between ResNet and NODE.

describe the data. The framework can be described as follows:

$$\begin{bmatrix} x_0 \\ \log p(x) - \log p(x_0)\end{bmatrix} = \int_t^0 \begin{bmatrix} f_{NN}(x, u; \theta) \\ -\text{tr}\left(\frac{\partial f_{NN}}{\partial x^T}\right)\end{bmatrix} dt; \qquad \begin{bmatrix} x(t) \\ \log p(x) - \log p(x_t)\end{bmatrix} = \begin{bmatrix} x_t \\ 0 \end{bmatrix} \tag{17}$$

We assume, $x_0 \sim \mathcal{N}(0, \sigma^2 I)$ and thus the objective function is:

$$\min_\theta V(x) = -\log p(x) \tag{18}$$

That's a powerful framework to model distributions (say a human face composed of pixels). However, we are more interested in using similar generative framework for chemical kinetics. Thus, we again use the same example as shown in (2) but we also model how the uncertainty in the initial conditions decreases as we add more measurement points. The results are shown in figure 4. The system we solve is:

$$\begin{bmatrix} \dot{c}_A \\ \dot{c}_B \\ \dot{c}_C \\ \dot{\log p} \end{bmatrix} = \begin{bmatrix} -k_1 c_A + k_{-1} c_B \\ 2k_1 c_A + 2k_{-1} c_B - k_2 c_B \\ k_2 c_B \\ -\mathrm{tr}\left(\frac{\partial f_{NN}}{\partial c^T}\right) \end{bmatrix}; \qquad \begin{bmatrix} c_A(0) \\ c_B(0) \\ c_C(0) \\ \log p(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \log \mathcal{N}([c_A, c_B, c_C]^T, \sigma^2 I) \end{bmatrix}$$

$$\min_{\theta} V(c) = -\log p(c) + \|c - \tilde{c}\|^2 \tag{19}$$

The author would like to draw parallels to the classical Kalman filter (Kalman, 1960) that also does the similar job, but the continuous normalizing flow can be applied to any non-linear dynamical system.

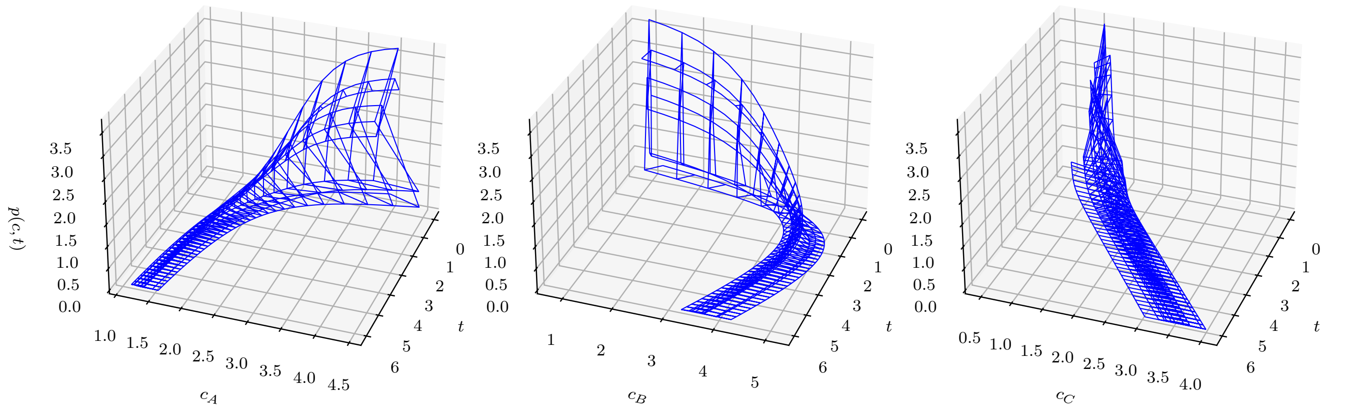

Fig. 4: Results for chemical kinetics modeled as continuous normalizing flow. We start with a high uncertainty in the initial conditions. The spread in probability decreases as we add more measurement points. Readers would like to draw similarities to Kalman filter, only in this case we can apply it to any non-linear continuous dynamical system.

2.6 Robustness analysis of NODEs

Well-posedness of an IVP is given by following theorem (Ascher and Petzold, 1998):

Theorem 1. *Let $f(t, x)$ be continuous for all (t, x) in a region $\mathcal{D} = 0 \le t \le b, |x| \le \infty$. Moreover assume Lipschitz contnuity in x, then there exists a constant L such that:*

$$\|f(t, x) - f(t, \bar{x})\| \le L\|x - \bar{x}\|$$

Thus, for ODEs with slightly perturbed initial conditions, the solution should not intersect though they can approach the same trajectory. Thus, the solution of an ODE can be sandwhiched between two other perturbed solutions. Refer to figure 5 for a simple example that again follows the kinetics in (2). Thus, following above idea, Yan et al. (2019) developed robust NODEs. The idea is the perturbed solution can be seen as time-shifted solution of the original ODE sytem. Using uniqueness of solutions, time-invariant NODEs can be regularized by adding a penalty term to the loss function. The penalty term is given by:

$$\|x(t + T) - x(t)\| \le \|\int_t^{t+T} f_{NN}(x, u; \theta) dt\|$$

Finlay et al. (2020) recommend to penalize the Jacobian of ODE rhs function ($\nabla_x f$) to damp oscillations that happen due to overfitting in neural ODE. A simple example using polynomial fit is show in figure 6.

3 Conclusions

In this review, we explored the mathematical foundations and practical applications of Neural Ordinary Differential Equations (NODEs), highlighting their connection to classical ODE-based modeling and modern machine learning

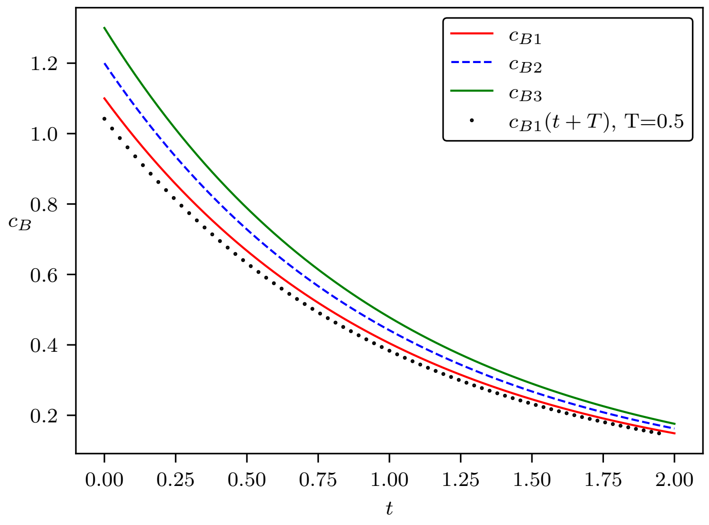

Fig. 5: Robustness analysis of ODEs. We see that the solution c_{B2} is sandwiched between two perturbed solutions c_{B1}, c_{B3}, though all approach the same steady-state. Also note that the slightly time-shifted solution appears as a perturbed solution

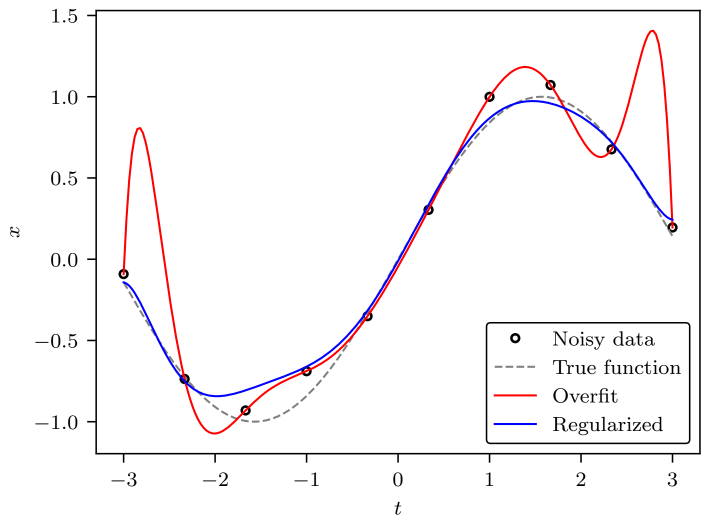

Fig. 6: If we train based only on prediction loss, the model can overfit. However, if we penalize the jacobian of the loss, we can regularize the model to damp the oscillations.

approaches. We first established how NODEs extend traditional discrete architectures such as ResNets and RNNs by formulating neural networks as continuous dynamical systems. This formulation provides a more principled way to model physical and process-driven data while reducing memory constraints through adjoint-based sensitivity analysis.

A key focus of this review was the gradient computation techniques for NODEs. We discussed the forward sensitivity method, which requires solving an augmented ODE system but suffers from an increase in computational cost with the number of parameters. The adjoint sensitivity method, in contrast, offers a more efficient alternative by solving a backward ODE, making NODEs feasible for large-scale learning problems. While adjoint methods significantly improve efficiency, they are not free from challenges, as stiffness and numerical instabilities can still affect convergence.

We also explored continuous normalizing flows (CNFs), where NODEs are utilized to construct generative models that explicitly model probability density evolution. The application of CNFs to chemical kinetics demonstrated how this framework can be extended to physical modeling while drawing conceptual parallels with classical filtering techniques such as the Kalman filter. These methods provide powerful tools for uncertainty quantification and data-driven discovery in dynamical systems.

A critical aspect of NODE-based learning is the robustness of solutions. We examined how NODEs, as solutions to differential equations, inherit the stability properties of the underlying ODEs. In particular, well-posedness ensures that small perturbations in initial conditions do not lead to solution intersections, reinforcing the interpretability and reliability of NODE-based models. This insight was leveraged to propose robust NODEs, where regularization techniques enforce time-invariance by penalizing deviations between time-shifted trajectories.

Overall, NODEs present a compelling framework for modeling dynamical systems in a continuous, memory-efficient, and theoretically grounded manner.

4 Data availability and Code

All the figures in the review have been generated by the author. The code for the figures can be found at `https://github.com/dakeprithvi/2025a_cnf`.

References

U. N. Ascher and L. R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations.* SIAM, Philadelphia, 1998.

R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. *Adv. Neural Inf. Process. Syst.*, 31, 2018.

C. Finlay, J.-H. Jacobsen, L. Nurbekyan, and A. Oberman. How to train your neural ode: the world of jacobian and kinetic regularization. In *International conference on machine learning*, pages 3154–3164. PMLR, 2020.

W. Grathwohl, R. T. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

R. E. Kalman. A new approach to linear filtering and prediction problems. *Trans. ASME, J. Basic Engineering*, pages 35–45, Mar 1960.

L. Ljung. *System Identification: Theory for the User.* Prentice Hall, New Jersey, second edition, 1999.

M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.*, 378: 686–707, 2019.

J. B. Rawlings and J. G. Ekerdt. *Chemical Reactor Analysis and Design Fundamentals.* Nob Hill Publishing, Santa Barbara, CA, 2nd, paperback edition, 2020. 664 pages, ISBN 978-0-9759377-4-7.

J. B. Rawlings, D. Q. Mayne, and M. M. Diehl. *Model Predictive Control: Theory, Computation, and Design.* Nob Hill Publishing, Santa Barbara, CA, 2nd, paperback edition, 2020. 770 pages, ISBN 978-0-9759377-5-4.

Y. Rubanova, R. T. Chen, and D. K. Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019.

B. Sengupta, K. J. Friston, and W. D. Penny. Efficient gradient computation for dynamical models. *NeuroImage*, 98:521–527, 2014.

P. Stapor, F. Froehlich, and J. Hasenauer. Optimization and uncertainty analysis of ODE models using 2nd order adjoint sensitivity analysis. *bioRxiv*, page 272005, 2018.

J. V. Villadsen and W. E. Stewart. Solution of boundary-value problems by orthogonal collocation. *Chem. Eng. Sci.*, 22:1483–1501, 1967.

H. Yan, J. Du, V. Y. Tan, and J. Feng. On robustness of neural ordinary differential equations. *arXiv preprint arXiv:1910.05513*, 2019.

J. Zhuang, N. Dvornek, X. Li, S. Tatikonda, X. Papademetris, and J. Duncan. Adaptive checkpoint adjoint method for gradient estimation in neural ode. In *International Conference on Machine Learning*, pages 11639–11649. PMLR, 2020.