Stirring Up Neural ODEs: A Chemical Engineer's perspective

Prithvi Dake

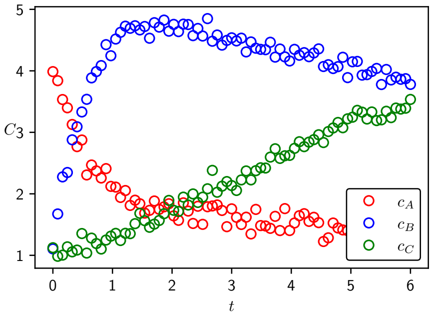
Department of Chemical Engineering

ME255NN Report
March 20, 2025

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:

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

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



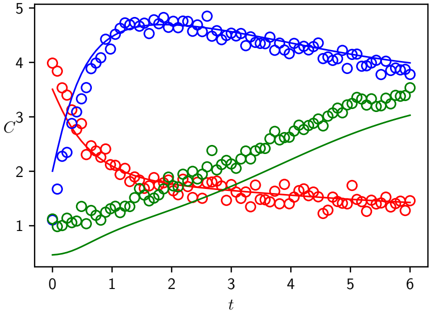
Background: If PINNs why NODEs?

Training a Physics-Informed Neural Network (PINN) [1]

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

[1] Raissi et al. (2019)

Background: If PINNs why NODEs?



- I view PINNs as a softer version[2] of *method of weighted residuals*. [3]
- Mostly used as surrogate numerical solver for PDEs.
- Can learn some physics but has no *structure*.
- Causality problem (no sequential model, future depends on past). [4]

Recent advances

Physics-informed neural ODE (PINODE) [5]
Lift-and-embed: Koopman operators [6]

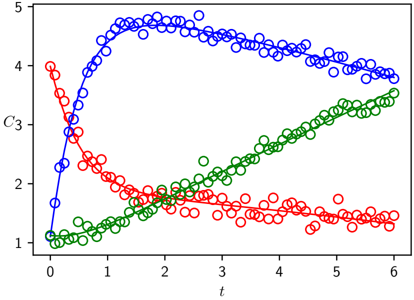
[2]By softer I mean PINNs minimize rather than exactly satisfy equality constraints
[3]Villadsen and Stewart (1967)
[4]Wang et al. (2022)
[5]Sholokhov et al. (2023)
[6]Liu et al. (2024)

ResNets and RNNs

A state-space representation: Euler discretized system

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

Parallels with FIR and ARX models in system identification [7].
Backprop strategy: Brute-force chain rule (store intermediate states).



If $x \in \mathbb{R}^n, \theta \in \mathbb{R}^p$,
$$S = \frac{\partial x}{\partial \theta^T} \in \mathbb{R}^{n \times p}$$

[7] Ljung (1999)

Don't want to store intermediate states? Forward sensitivities for IVPs [9]

- Consider a non-linear state-space model ($y = x$):

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

- Sensitivity is defined as and evolves as [8]:

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

- Augmented system of ODEs:

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

- Notice again: $S \in \mathbb{R}^{n \times p}$ i.e sensitivities we need to solve for increase linearly with p. Imagine how bad this could get for FNN with millions of parameters and states!

[8] Using Clairaut's Theorem on Mixed Partials
[9] Derivation in Rawlings and Ekerdt (2020)

How to avoid the linear cost? Pontryagin's adjoint method

Lev Semenovich Pontryagin P. S. Alexandrov Andrey N. Kolmogorov

Backward adjoint method

- Consider following optimization problem and the Lagrange functional given data $\tilde{x}$:

$$\min_{\theta} V(x) = \int_0^t |x - \tilde{x}|^2 \, dt = \int_0^t g(x) dt$$

$$\min_{\theta,\lambda} \mathcal{L} = \int_0^t g dt + \int_0^t \lambda(t)^T \left(f - \frac{dx}{dt} \right) dt$$

- The augmented system of ODEs is [10]:

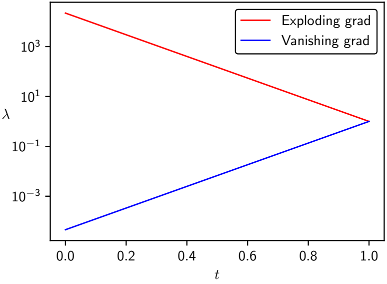
$$\begin{bmatrix} \dot{x} \\ \dot{\lambda}^T \end{bmatrix} = \begin{bmatrix} f_{NN} \\ -\lambda^T \dfrac{\partial f_{NN}}{\partial x^T} \end{bmatrix} \qquad \begin{bmatrix} x(t) \\ \lambda^T(t) \end{bmatrix} = \begin{bmatrix} x_t \\ -\dfrac{\partial g}{\partial x^T}|_t \end{bmatrix}$$

- The gradient of loss wrt parameters is calculated as:

$$\frac{\partial V}{\partial \theta^T} = \lambda^T(0) \frac{\partial x}{\partial \theta^T}|_0 + \int_0^t \lambda^T \frac{\partial f}{\partial \theta^T} dt$$

[10] Refer Sengupta et al. (2014) for paper and `https://youtu.be/k6s2G5MZv-I?si=ZIyP7MvgzTK7D9kn` for a walkthrough. Additionally, I have worked out the derivation step-by-step in my report.

An ultimate panacea?



- Consider $\dot{\lambda} = \pm 10\lambda$ and $\lambda(0) = 0$
- We see that the adjoint variable will either *explode* or *vanish*. This heavily impacts the learning problem.
- The Jacobian $\dfrac{\partial f}{\partial x^T}$ can be ill-conditioned.
- We solve the adjoint equation by quadrature, thus numerical noise can still be a problem.

Remedies

- Don't solve adjoint over long horizons, but rather small windows and store the initial conditions for those intermediate windows.
- Popularly called as *checkpointing* [11].

[11] Zhuang et al. (2020)

NODE

Input: Neural network architecture f_{NN}

ODE: $\begin{bmatrix} \dot{x} \\ \dot{\lambda}^T \end{bmatrix} = \begin{bmatrix} f_{NN} \\ -\lambda^T \dfrac{\partial f_{NN}}{\partial x^T} \end{bmatrix} \qquad \begin{bmatrix} x(t) \\ \lambda^T(t) \end{bmatrix} = \begin{bmatrix} x_t \\ -\dfrac{\partial g}{\partial x^T}\Big|_t \end{bmatrix}$ ▷ Construct the augmented ODE using `PyTorch`

Initialize neural network parameters θ

repeat

  $V = |\hat{x} - x|^2$ ▷ Forward solve to get the loss V

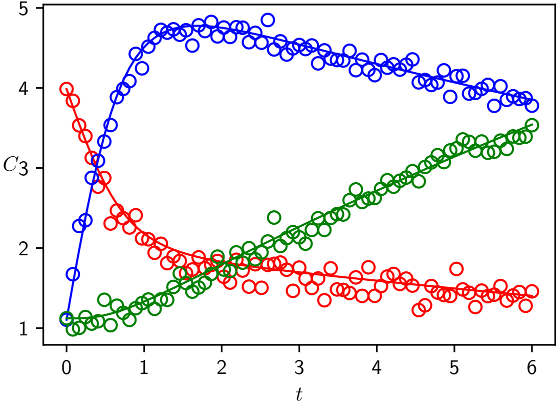
  $\hat{x}_{aug} = \text{ODESolve}(f_{aug}, x_{aug}(0), T, t_0)$

  $\dfrac{\partial V}{\partial \theta^T} = \lambda^T(0) \dfrac{\partial x}{\partial \theta^T}\Big|_0 + \int_0^t \lambda^T \dfrac{\partial f}{\partial \theta^T} dt$ ▷ Construct the gradient wrt loss

  $\theta \leftarrow \theta - \eta \nabla_\theta V$

until convergence criteria is met

Return trained model f_{NN}



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 [12].

[12] All the case studies in this report are built using `Torchdiffeq` by Chen (2018)

Generative latent model using CNF

- **Latent** = States in state-space model, since we usually cannot measure them directly. Thus, we construct state-estimators.
- **Generative latent** = State-space model that also propagates uncertainty in estimates i.e. you can sample and *generate* multiple trajectories after learning.
- Surprise! Good old Kalman filter can be called a generative latent model! [13]
- Consider a simple linear transform:

$$
\begin{aligned}
&x \in \mathcal{N}(m, \sigma^2 I) \\
&y = Ax \\
&y \in \mathcal{N}(Am, A\sigma^2 A^T)
\end{aligned}
$$

- Bayesian approach: $p(y|x) \sim p(x|y)p(y)$. Assuming an uniform prior the likelihood problem is

$$
\theta = \arg\max_{\theta} \log p(x|y;\theta) \qquad p(x|y) = A^{-1}y \sim \mathcal{N}(m, \sigma^2 I)
$$

- Ofcourse, A is assumed to be non-singular (i.e. unique mapping between x and y; *bijective mapping*).

[13] Refer Rawlings et al. (2020) for derivation of the Kalman filter using Bayesian estimation principles.

Extending the bayesian framework for any non-linear transform

- Consider following series of transformations (*planar-flow*):

$$x^{+} = f_{NN}(x, u; \theta) \qquad x(0) \sim \mathcal{N}(m, \sigma^2 I) \qquad x(t) \sim ?$$

- Use *change of variables*, compute the determinant of Jacobian ($\sim N(o^3)$) and conserve the probability mass as well as maximize the likelihood.
- Instead use continuous framework (*continuous normalizing flow*) [14]:

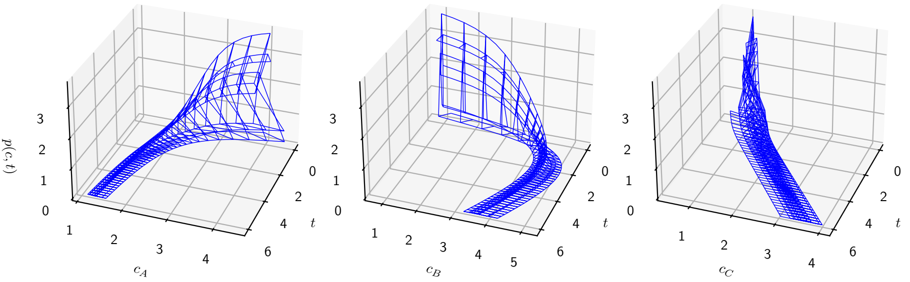
$$\frac{dx}{dt} = f_{NN}(x, u; \theta) \qquad x(0) \sim \mathcal{N}(m, \sigma^2 I) \qquad x(t) \sim ?$$

$$\frac{\partial p(x)}{\partial t} = -\mathrm{tr} \nabla_x f_{NN}$$

- Cost to get the trace of Jacobian is now linear $\sim N(o)$

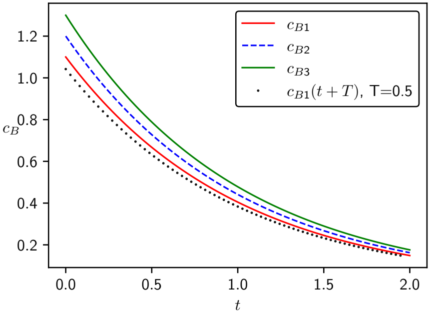
[14] Refer Chen et al. (2018) for proof

Generative latent model for kinetics



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

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



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 [15]

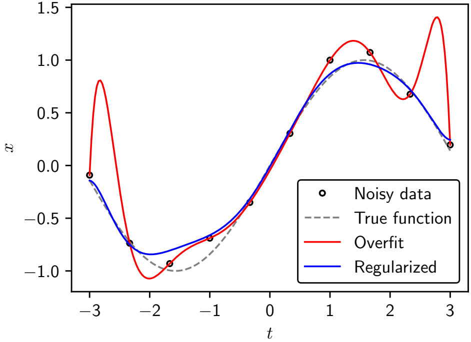
Theoretical bound

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

[15] Refer Ascher and Petzold (1998) for well-posedness theorem of IVPs.

[16] Refer Yan et al. (2019)

How to train your NODE?



If we train based only on prediction loss, the model can overfit. However, if we penalize the Jacobian term ($\frac{\partial f_{NN}}{\partial x^T}$), we can regularize the model to damp the oscillations [17]

[17] Finlay et al. (2020)

Access the code for all figures at `https://github.com/dakeprithvi/2025a_cnf`

References I

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.

R. T. Q. Chen. torchdiffeq, 2018. URL `https://github.com/rtqichen/torchdiffeq`.

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.

Y. Liu, A. Sholokhov, H. Mansour, and S. Nabi. Physics-informed koopman network for time-series prediction of dynamical systems. In *ICLR 2024 Workshop on AI4DifferentialEquations In Science*, 2024.

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.

References II

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.

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

A. Sholokhov, Y. Liu, H. Mansour, and S. Nabi. Physics-informed neural ode (pinode): embedding physics into models using collocation points. *Scientific Reports*, 13(1): 10166, 2023.

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

S. Wang, S. Sankaran, and P. Perdikaris. Respecting causality is all you need for training physics-informed neural networks, 2022. URL `https://arxiv.org/abs/2203.07404`.

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.