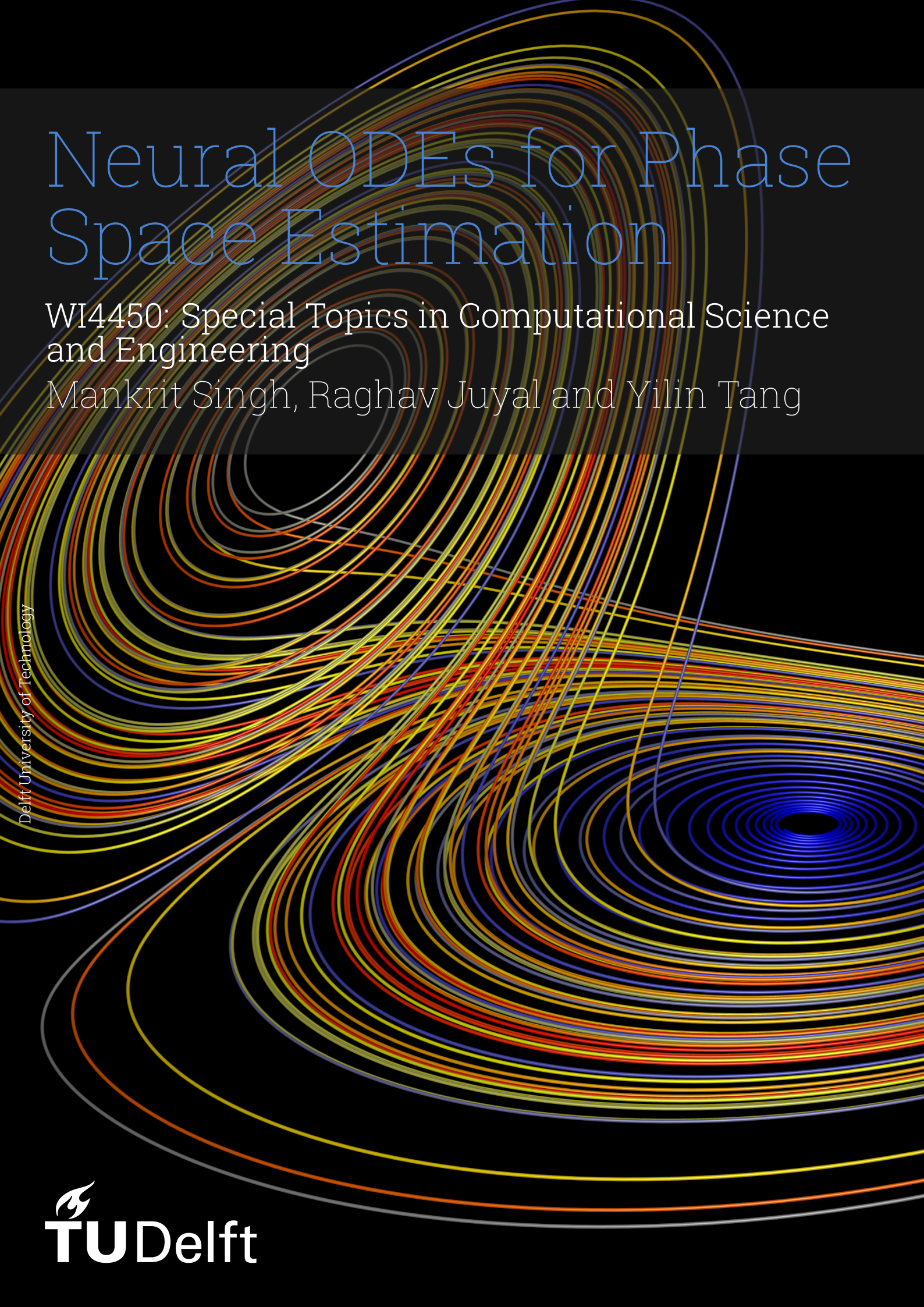
Neural ODEs for Phase Space Estimation

WI4450: Special Topics in Computational Science and Engineering

Mankrit Singh, Raghav Juyal and Yilin Tang



Delft University of Technology

TU Delft

Neural ODEs for Phase Space Estimation

by

Mankrit Singh, Raghav Juyal and Yilin Tang

Instructor: A. Heinlein
Project Duration: April, 2025 - June, 2025
Faculty: Faculty of Electrical Engineering, Mathematics and Computer Science

Cover: The Lorenz Attractor by Paul Bourke
Style: TU Delft Report Style, with modifications by Daan Zwaneveld

TU Delft

Contents

1. Introduction

Understanding dynamical systems is at the heart of several fields of science and engineering. Any system whose state evolves over time according to a set of differential equations can be modeled as a continuous-time dynamical system [1]. Dynamical systems can be linear or nonlinear, and they are often studied in their phase space, which highlights the evolution of the system's state over time. Essentially, if a system starts at an arbitrary initial point in the phase space, it follows a predefined trajectory that depends only on the governing equations and its parameters [1].

Traditionally, dynamical systems were studied by deriving their governing differential equations from first principles, which is challenging or intractable depending on the complexity of the system. Due to the challenging nature of this task in the general case, where we do not know the underlying dynamics and/or equation for a system, the more modern way to tackle this problem is using data-driven dynamics [2]. While using this approach, we typically do not assume the governing equations of the system and use techniques from Machine Learning to better model the system. Further details can be found in Section 2.

Neural Ordinary Differential Equations (Neural ODEs) [3] offer a continuous-time framework to learn the underlying vector field governing a system's dynamics, without the need for explicit equations. This serves as a promising means to study the phase space of different dynamical systems, and a complete pipeline can be built in the following manner: we train the Neural ODE with trajectories from the system of our interest, then we use the trained Neural ODE add more trajectories in the phase space. Finally, with a better picture of the phase space (within the training range), we explore the dynamical features of the system such as fixed points, limit cycles etc. This is precisely what we attempt to do in our project, and our exact research question is formalised in Section 3.

In a real-life situation, we can expect to measure the different observables related to the state of the dynamical system in set intervals of time, and assuming these measurements are non-invasive to the dynamics of the system, we can observe the system's trajectory in phase space. Suppose we wish to model a particular closed thermodynamic system, and note how certain initial states (ie. a defined pressure, temperature etc.) evolve with time. Using the data points during the evolution of the system and feeding them into our proposed model, we can learn essential dynamics like fixed points (points in the phase space where the system does not evolve) and their stability. With this information, we may then initialise our system near these fixed points depending on our use case.

In summary, this project explores a data-driven approach to modelling the phase space of dynamical systems using Neural Ordinary Differential Equations (Neural ODEs), trained directly on observed trajectories. We benchmark our model against the Sparse Identification of Nonlinear Dynamics (SINDy), a well-established tool for equation discovery. We focus on identifying fixed points and evaluating their stability in 2D systems, as well as demonstrating bifurcation behaviour in the Van der Pol oscillator. Additionally, we compare the effect of different activation functions on model performance. Finally, we examine the limitations of Neural ODEs—particularly in chaotic regimes—and propose directions for improvement.

2. Literature Review

In this section, we review relevant literature on state-of-the-art data-driven methods for modeling dynamical systems (specifically SINDy), and Neural Ordinary Differential Equations (Neural ODEs).

A widely used tool to tackle the problem described above, is the Sparse Identification of Nonlinear Dynamics (SINDy) algorithm [4], which, like Neural ODEs, models a system using its observed trajectories. However, SINDy outputs an explicit symbolic form of the governing equations by selecting sparse terms from a predefined dictionary. In our results (Section 5), we compare the phase portraits and metrics generated by our model with SINDy (and fourth-order Runge–Kutta method on its output).

One limitation for Neural ODEs is the common assumption that the system's time derivatives are directly observable, which is often unrealistic in real-world scenarios. Recent advances propose provably stable latent dynamical systems that follow the Neural ODE framework but eliminate the need for derivative information [5]. These models leverage trajectory-based losses, such as the Average Hausdorff Distance, and are benchmarked on datasets like LASA.

ControlSynth Neural ODEs (CSODEs) extend the standard Neural ODE framework to handle highly nonlinear systems across diverse domains. These models ensure convergence and are evaluated using metrics like Mean Squared Error (MSE), Mean Absolute Error (MAE), R^2 score, and Chamfer Distance [6].

Finally, chaotic systems remain a persistent challenge for Machine Learning algorithms in general. While these models can reasonably capture stable local Lyapunov exponents, their ability to learn unstable or neutral exponents is significantly weaker, limiting their effectiveness in highly sensitive dynamical regimes [7].

3. Research Question

Can Neural ODEs accurately learn the underlying vector field of a dynamical system from trajectory data, and can the learned dynamics be used to reliably identify and analyse fixed points and their stability?

Additionally:

- How well do Neural ODEs interpolate within the training region, and to what extent do they generalise when extrapolating beyond it?
- How do different activation functions affect the ability of Neural ODEs to learn complex dynamics?
- How does the performance of Neural ODEs compare to equation discovery methods such as SINDy?

4. Methodology

4.1. Theory of Dynamical Systems

A dynamical system describes the evolution of a state over time according to a fixed rule. Formally, a continuous-time dynamical system is given by the ordinary differential equation (ODE):

$$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), t) \tag{1}$$

where $\mathbf{x}(t) \in \mathbb{R}^n$ represents the state at time t, and $f : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ is a function describing the system dynamics.

A trajectory is the solution $\mathbf{x}(t)$ with a certain initial condition $\mathbf{x}(t_0) = \mathbf{x}_0$ plotted in the phase space, where the axes are the elements of $\mathbf{x}$.

4.1.1. Equilibrium Point and Stability

An equilibrium point (also called fixed point) $\mathbf{x}^* \in \mathbb{R}^n$ is a constant solution to an ODE, which satisfies

$$f(\mathbf{x}^*, t) = 0, \quad \forall t \tag{2}$$

An equilibrium point $\mathbf{x}^*$ is stable, if $\forall \epsilon > 0$, there exists a $\delta > 0$ such that any trajectory starting within a δ-neighbourhood of $\mathbf{x}^*$ remains within ϵ-neighbourhood of $\mathbf{x}^*$ as $t \to \infty$. If additionally, these trajectories converge to $\mathbf{x}^*$ as $t \to \infty$, $\mathbf{x}^*$ is asymptotically stable.

For nonlinear systems, an equilibrium point $\mathbf{x}^*$ can be analysed by linearising the system:

$$\frac{d\mathbf{y}}{dt} = J_f(\mathbf{x}^*)\mathbf{y} \tag{3}$$

where $J_f(\mathbf{x}^*)$ is the Jacobian matrix of f evaluated at $\mathbf{x}^*$. The stability of $\mathbf{x}^*$ can be determined by the sign of the real parts of the eigenvalues of $\mathbf{x}^*$. If all eigenvalues have negative real parts, $\mathbf{x}^*$ is asymptotically stable. If any eigenvalue has a positive real part, $\mathbf{x}^*$ is unstable. If neither of the above two cases applies, the stability cannot be determined by the linearisation technique.

Especially, in two dimensions, the equilibrium point can be classified into different types according to the eigenvalues of the Jacobian matrix [8]:

- Node: Two eigenvalues are real and have the same signs. Given the equilibrium point is stable or not, trajectories approach or move away from the equilibrium point as $t \to \infty$.
- Focus: Two eigenvalues are complex conjugate. Given the equilibrium point is stable or not, trajectories spirally approach or move away from the equilibrium point as $t \to \infty$.
- Saddle: Two eigenvalues are real and have different signs. Trajectories approach along one direction and move away along another.
- Center: Two eigenvalues are purely imaginary. Trajectories form closed orbits around the equilibrium point.

4.1.2. Other Dynamical Behaviours

Other than equilibrium points, some dynamical systems show many interesting behaviours. Some are listed as follows [1]:

- Limit Cycle: An isolated periodic solution, which is a closed trajectory in the phase plane, having the property that at least one other trajectory spirals into it either as $t \to \infty$, or as $t \to -\infty$. If all the neighbouring trajectories approach the limit cycle as $t \to \infty$, the limit cycle is stable. If all the neighbouring trajectories approach the limit cycle as $t \to -\infty$, the limit cycle is unstable.
- Heteroclinic Orbit: A path joins two different equilibrium points in the phase space.
- Homoclinic Orbit: A path joins a saddle equilibrium point to itself in the phase space.
- Chaos: The system exhibits extreme sensitivity to initial conditions, meaning that tiny differences in starting points can lead to vastly different trajectories over time.

4.1.3. Discussed Systems

To examine the function of Neural ODE on simulating different kinds of dynamical systems, we will discuss the following systems, which incorporate most of the interesting behaviour mentioned before.

Van der Pol Equation The Van der Pol equation is

$$\begin{cases} \dot{x} = y \\ \dot{y} = \mu\left(1 - x^2\right) y - x \end{cases} \tag{4}$$

When $\mu > 0$, $(0,0)$ is an unstable equilibrium point and there exists a stable limit cycle around $(0,0)$ [1]. When $\mu = 0$, $(0,0)$ is a center, which means all neighbouring trajectories around $(0,0)$ are periodic solutions. When $\mu < 0$, $(0,0)$ is a stable equilibrium point. As μ moves from negative to positive, the stability of $(0,0)$ changes from stable to unstable, and the orbits are attracted to a limit cycle, which is known as a Hopf bifurcation [1].

A Nonlinear Homoclinic System We consider the following 2D dynamical system (from [8], p.17)

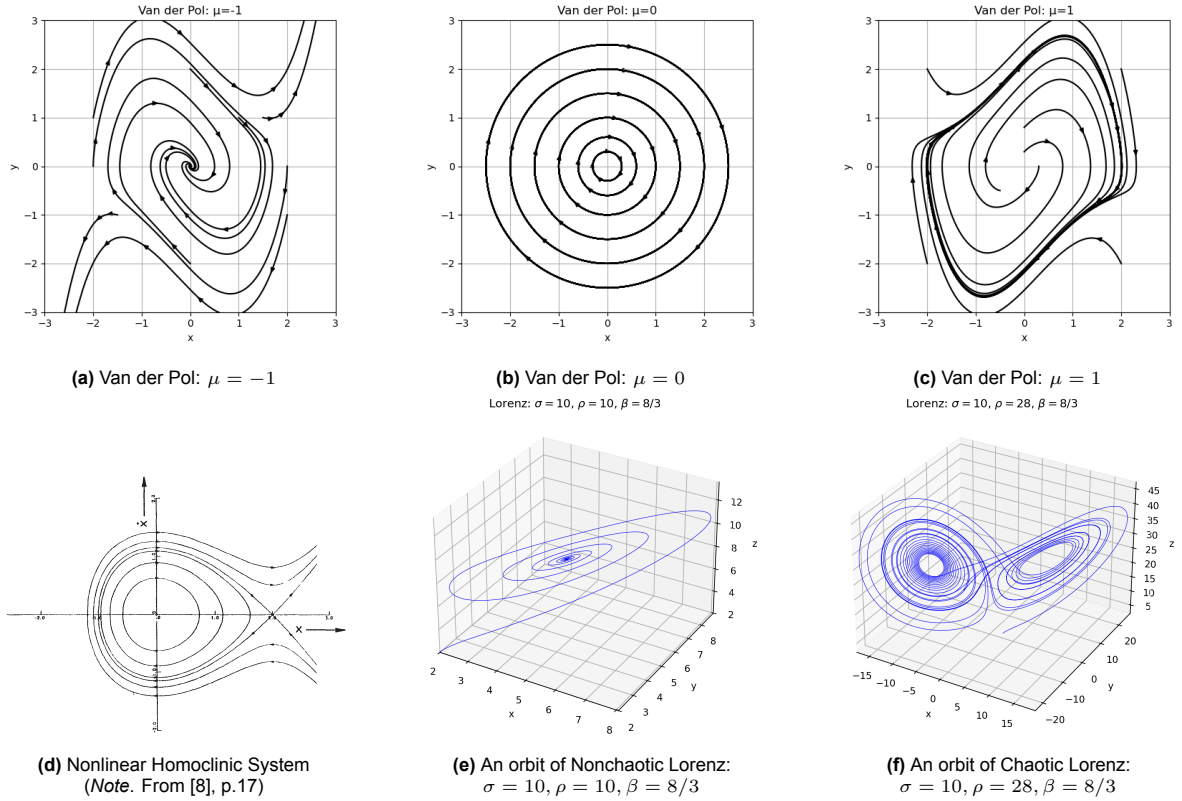
$$\begin{cases} \dot{x} = y \\ \dot{y} = \frac{1}{2}x^2 - x \end{cases} \tag{5}$$

It has two equilibrium points, a center $(0,0)$ and a saddle $(2,0)$. There is a homoclinic orbit joins the saddle $(2,0)$ with the center $(0,0)$ inside it.

Lorenz System The Lorenz system is

$$\begin{cases} \dot{x} = \sigma(y - x) \\ \dot{y} = x(\rho - z) - y \\ \dot{z} = xy - \beta z \end{cases} \tag{6}$$

The Lorenz system with $\sigma = 10, \rho = 28, \beta = \frac{8}{3}$ is a classic example of chaotic systems. If $\rho < 1$, the only equilibrium point is the origin, which is stable. For $\rho > 1$, there are three equilibrium points: the origin and $(\pm\sqrt{\beta(\rho-1)}, \pm\sqrt{\beta(\rho-1)}, \rho-1)$. $(\pm\sqrt{\beta(\rho-1)}, \pm\sqrt{\beta(\rho-1)}, \rho-1)$ are stable for $1 < \rho < \sigma\frac{\sigma+\beta+3}{\sigma-\beta-1}$ [1].



(a) Van der Pol: $\mu = -1$

(b) Van der Pol: $\mu = 0$

(c) Van der Pol: $\mu = 1$

(d) Nonlinear Homoclinic System (*Note*. From [8], p.17)

(e) An orbit of Nonchaotic Lorenz: $\sigma = 10, \rho = 10, \beta = 8/3$

(f) An orbit of Chaotic Lorenz: $\sigma = 10, \rho = 28, \beta = 8/3$

Figure 1: Dynamical Behaviours of the Three Systems

4.2. Neural ODEs

For some complicated equations, solving the ODE analytically is not possible. This is why we need numerical solvers. The Runge-Kutta methods are a family of methods used to find approximate solutions to initial value problems. The simplest and most intuitive method in this family is the Euler method.

Consider

$$\dot{x} = f(x,t),\ x(t_0) = x_0$$

where f and x_0 are given. We then pick a step-size $h > 0$, a number of steps N and define the following method:

$$y_{n+1} = y_n + hf(y_n, t_n)$$
$$t_{n+1} = t_n + h$$

Many deep learning architectures, like ResNet, employ residual connections:

$$x_{l+1} = x_l + f(x_l, \theta_l)$$

where f is the neural network with parameters θ_l and hidden states y_l and y_{l+1} and l denotes the layer. This residual connection can be seen as an Euler method approximation of a continuous transformation, which can be seen in Fig.2. This led to interest in trying to model the transition function of ODEs f with a neural network, and because neural networks are universal approximators, they can approximate it to arbitrary precision. In neural ODEs, the neural network is used to represent f and then the numerical solver is used to evaluate the dynamics necessary to determine the solution with the required accuracy. For more details on the theory of Neural ODEs, one can refer to [3, 9, 10].

One of the key challenges with Neural ODEs is their sensitivity to the choice of numerical integration scheme. Most implementations rely on non-symplectic solvers, which can violate physical conservation

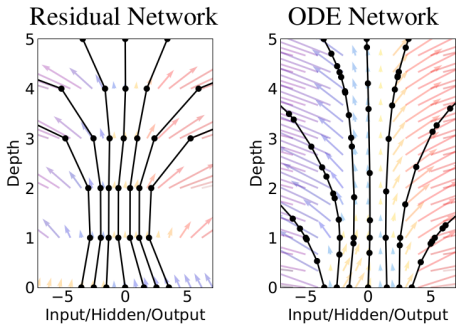


Figure 2: Comparison between ResNet and ODE-Net architectures showing how ResNet discretises ODEs [3].

laws and degrade performance [11]. To address this, physics-informed variants like PINODE incorporate known governing equations and collocation points into the training process, resulting in improved accuracy on classical systems such as the Duffing oscillator and Burgers' equation [12].

4.3. Learning system dynamics with Neural ODEs

In some cases, we may not know the ODE beforehand, which prevents us from using the previously mentioned numerical solvers, but we may still have data for certain trajectories. To model the phase space of this unknown ODE, we first train the Neural ODE on the known trajectories. To generate the data, we use known ODEs and use a numerical solver (4th order Runge-Kutta) to generate solution trajectories up to a certain timestep for a given set of starting points. In this way, we have trajectories with no knowledge of the true function given to the model. We then give the model the set of initial values and ask it to predict the entire trajectory for them. After training, we use the network to model the phase space and look at the fixed points.

4.4. Architecture Details

In this project, we tried using the following models

Model 1: Neural network with architecture $d \to 64 \to d$ with activation function between layers

Model 2: Neural network with architecture $d \to 64 \to 64 \to d$ with activation functions between layers

Here, d is the dimension of the data. Model 1 is mostly used for the 3D system and Model 2 is mostly used for the 2D systems. The activation function is something we varied following [13] to find which performed the best. The numerical solver used in the Neural ODE is DOPRI5 method (Dormand-Prince 5th order Runge-Kutta with 4th order method for adaptive sizing).

4.5. Finding Fixed Points and Stability Analysis

To identify fixed points of the learned vector field, we find the minimum squared magnitude of the vector field at multiple random points within a defined domain. This helps locate points where the vector field vanishes, which could be potential fixed points. We then perform minimisation using the `scipy.optimize.fmin` function, which implements the Nelder-Mead simplex algorithm for local optimisation. We use a tolerance threshold to filter and ensure uniqueness by discarding nearby duplicates.

Once the fixed points are identified, we analyse their stability by computing the Jacobian matrix via finite differences at each point. The eigenvalues of this Jacobian determine the local stability characteristics, as discussed in Section 4.1.1. This analysis enables classification of dynamical features such as sinks, sources, and saddle points, which are essential to understanding the system's long-term behaviour.

4.6. Error Metrics

We evaluate the similarity between predicted and ground truth trajectories using the following metrics, which were identified during Literature Review (Section 2):

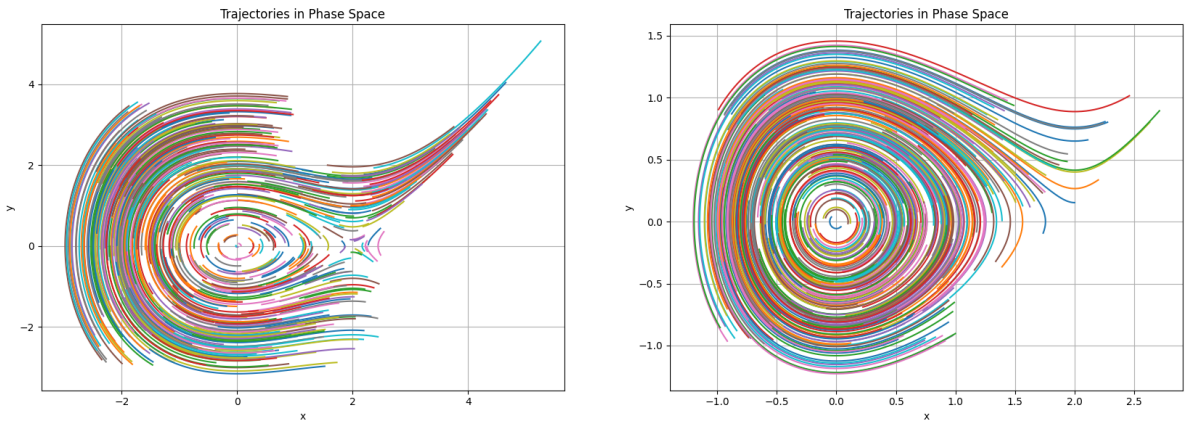
- Mean Squared Error (MSE): Measures the average squared difference between predicted and true trajectory points.
- Mean Absolute Error (MAE): Computes the average absolute difference between corresponding points.
- Chamfer Distance: A set-based metric that averages the distance from each point in one trajectory to its nearest neighbour in the other.
- Hausdorff Distance: Measures the maximum distance from a point in one trajectory to the nearest point in the other, capturing worst-case deviation.

5. Results

5.1. 2D Systems

5.1.1. Nonlinear Homoclinic System

We use the ODE in section 4.1.3 to generate train data points evolved for 1 timestep with the starting points taking x values in the range [-2.5,2.5] and y values in the range [-3,2], and test data points evolved for 3 timesteps with the starting points taking x and y values in the range [-1,1]. This can be seen in Fig. 3.



(a) Training data **(b)** Test data

Figure 3: Homoclinic trajectory datasets used for training and testing.

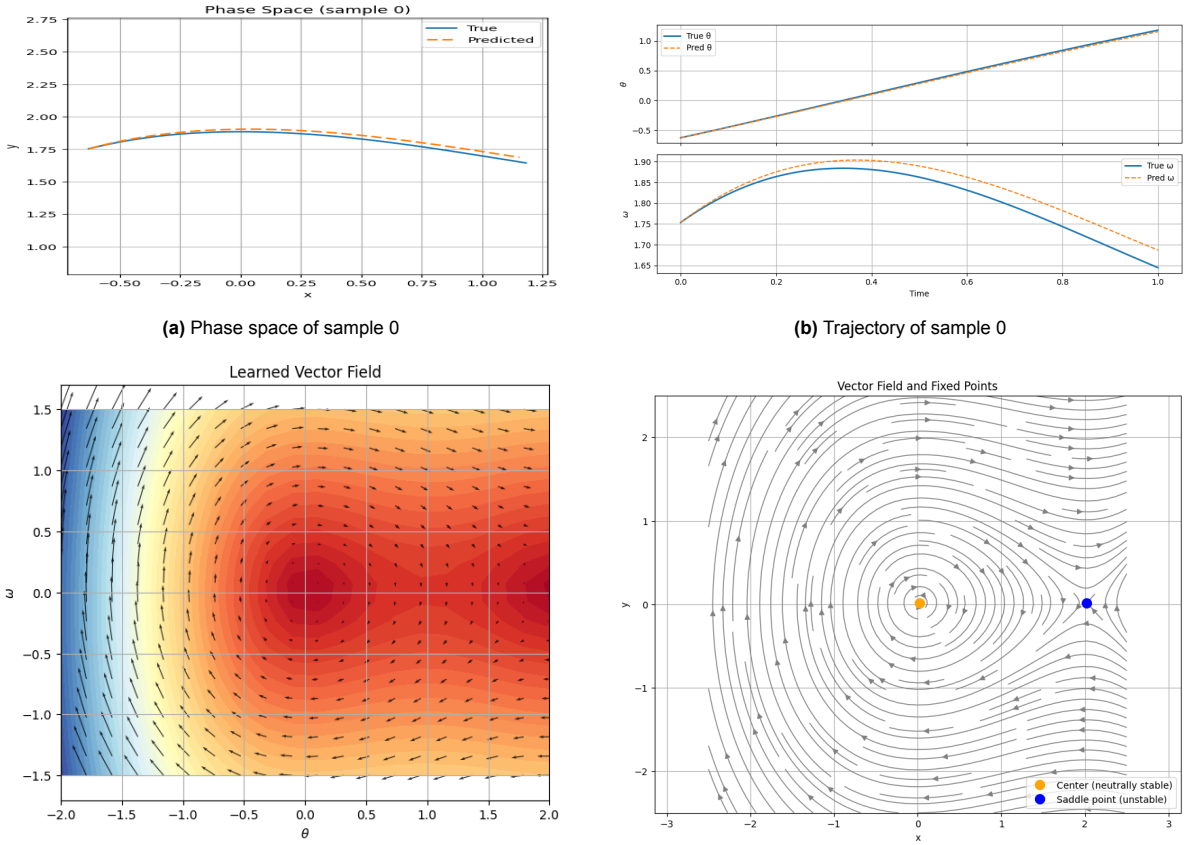
In Fig.4, we show the phase space and trajectory of the first sample, the learned vector field and the fixed point learned by the Neural ODE. We can see that the Neural ODE is in fact able to find both the stable center at the origin as well as the saddle point at (2,0). The error metrics for the Neural ODE can be seen in Table 2.

5.1.2. Van der Pol System

We use the ODE in section 4.1.3 to generate train points and test points for various values of μ. From Fig. 5 we can see that for $\mu = 1, 3$ we see the unstable equilibrium point at (0,0) and a stable limit cycle around (0,0). For $\mu = -0.1, 0, 0.1$ we see a stable center. While the center is only expected for $\mu = 0$, because we approximate eigenvalues, $\mu = -0.1, 0.1$ also show the center. For $\mu = -1, -3$ we see a stable focus at (0,0) with no limit cycle. This change in stability of the origin gives us Hopf bifurcation at $\mu = 0$, which is expected in these systems, so in this case, the Neural ODE has replicated this expected behaviour.

5.2. Interpolation and Extrapolation

We used the same Nonlinear Homoclinic system in section 4.1.3 to train the model. We then varied the test dataset to see how the Neural ODE would perform. The results can be see in Table 1. We notice



(a) Phase space of sample 0

(b) Trajectory of sample 0

(c) Learned vector field

(d) Fixed point analysis

Figure 4: Results on the homoclinic dataset using the trained Neural ODE model.

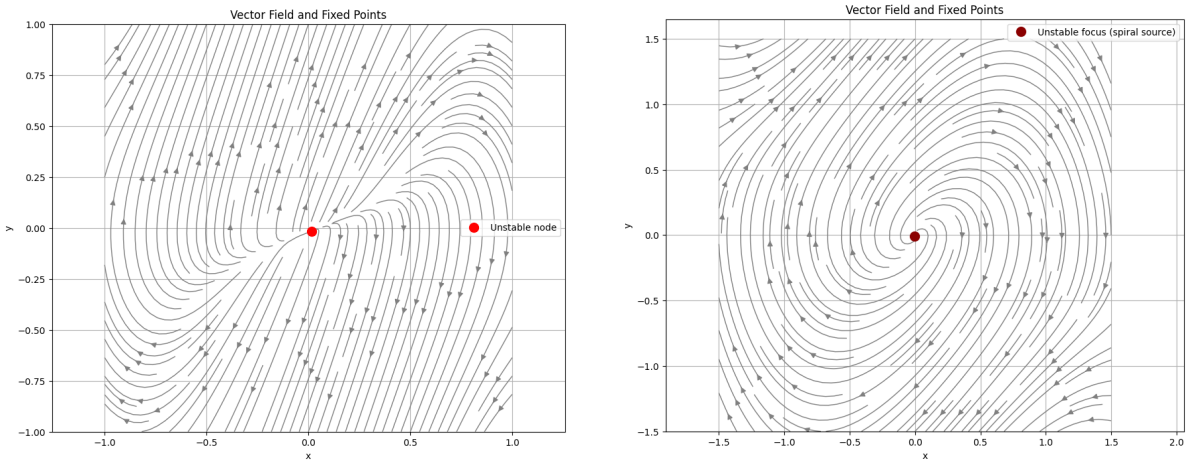
that it is able to interpolate well. In particular, we see that near the origin, because the system changes slowly, the performance of interpolation is in fact better than even the performance on the train data itself. The performance of interpolation in the region where the system changes quickly, beyond the saddle point, is worse. Evolving for a longer timestep or extrapolating beyond the train data means that the trajectories will go into the fast-changing parts of the system, and thus, the performance is worse.

Dataset	Step-size	Range x	Range y	Timestep	MSE	MAE	Chamfer	Hausdorff
Train Data	0.01	$[-2.5, 2.5]$	$[-3, 2]$	1	0.00039	0.01223	0.00069	0.04405
Interpolation(slow)	0.001	$[-1, 1]$	$[-1, 1]$	1	0.00015	0.00907	0.00033	0.02792
Interpolation(fast)	0.001	$[2, 2.5]$	$[1.5, 2]$	1	0.00423	0.04486	0.00107	0.25788
Long Evolution	0.01	$[-1, 1]$	$[-1, 1]$	5	0.08819	0.04677	0.13818	0.32874
Extrapolation	0.01	$[-5, 5]$	$[-5, 5]$	1	0.69298	0.16305	0.96269	1.05056

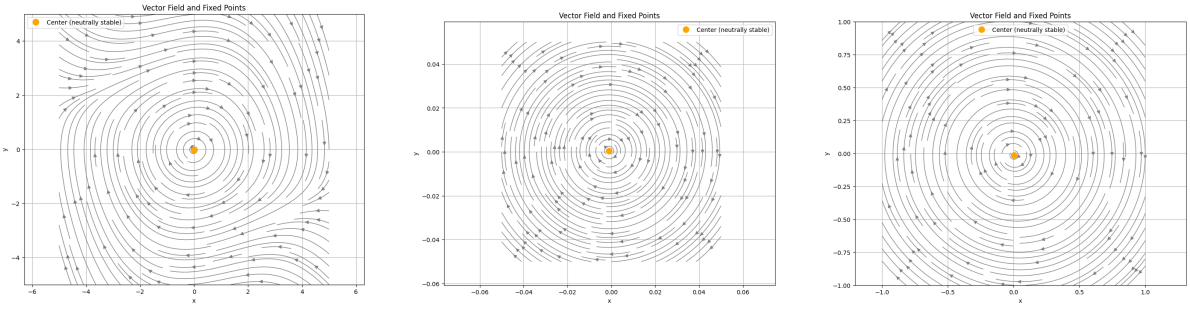
Table 1: Performance of the model on train, interpolation, long evolution, and extrapolation data.

5.3. 3D System: Lorenz

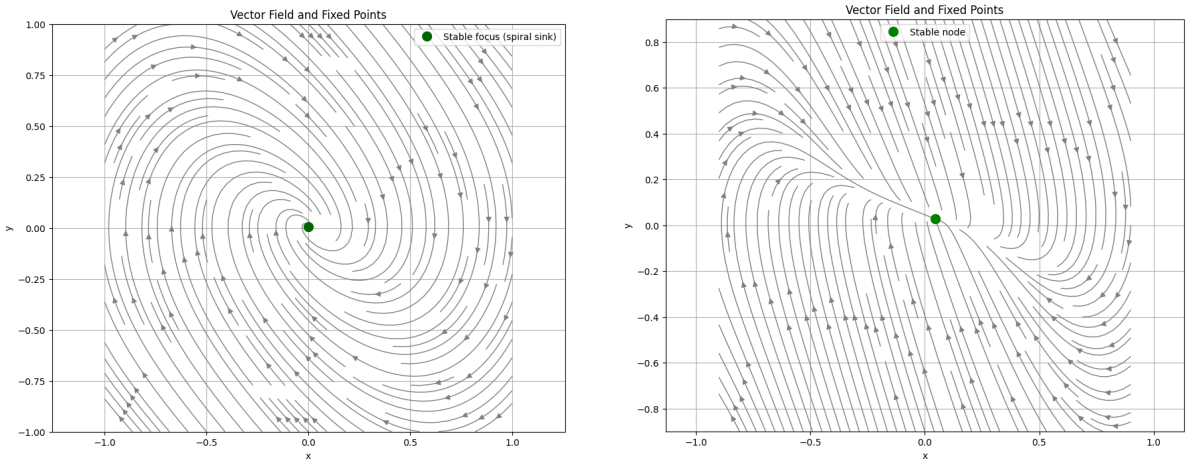
We also ran our model on the Lorenz system described in Section 4.1.3, which is a 3-dimensional system known for exhibiting chaos in certain regimes. Our goal was to assess whether the Neural ODE could capture the structure of trajectories for this system. As shown in Table 2, the model performs reasonably well in the non chaotic regime but struggles to reproduce the trajectory accurately in the chaotic setting. This is expected, as chaotic systems exhibit extreme sensitivity to initial conditions, leading to divergence in predicted trajectories over time. The significant gap in Hausdorff and Chamfer distances in the chaotic regime confirms that even small inaccuracies in learning the vector field lead to visibly diverging paths. In Fig 6, we can see the original phase space and the one generated by Neural ODE.



(a) $\mu = 3$ **(b)** $\mu = 1$



(c) $\mu = 0.1$ **(d)** $\mu = 0$ **(e)** $\mu = -0.1$

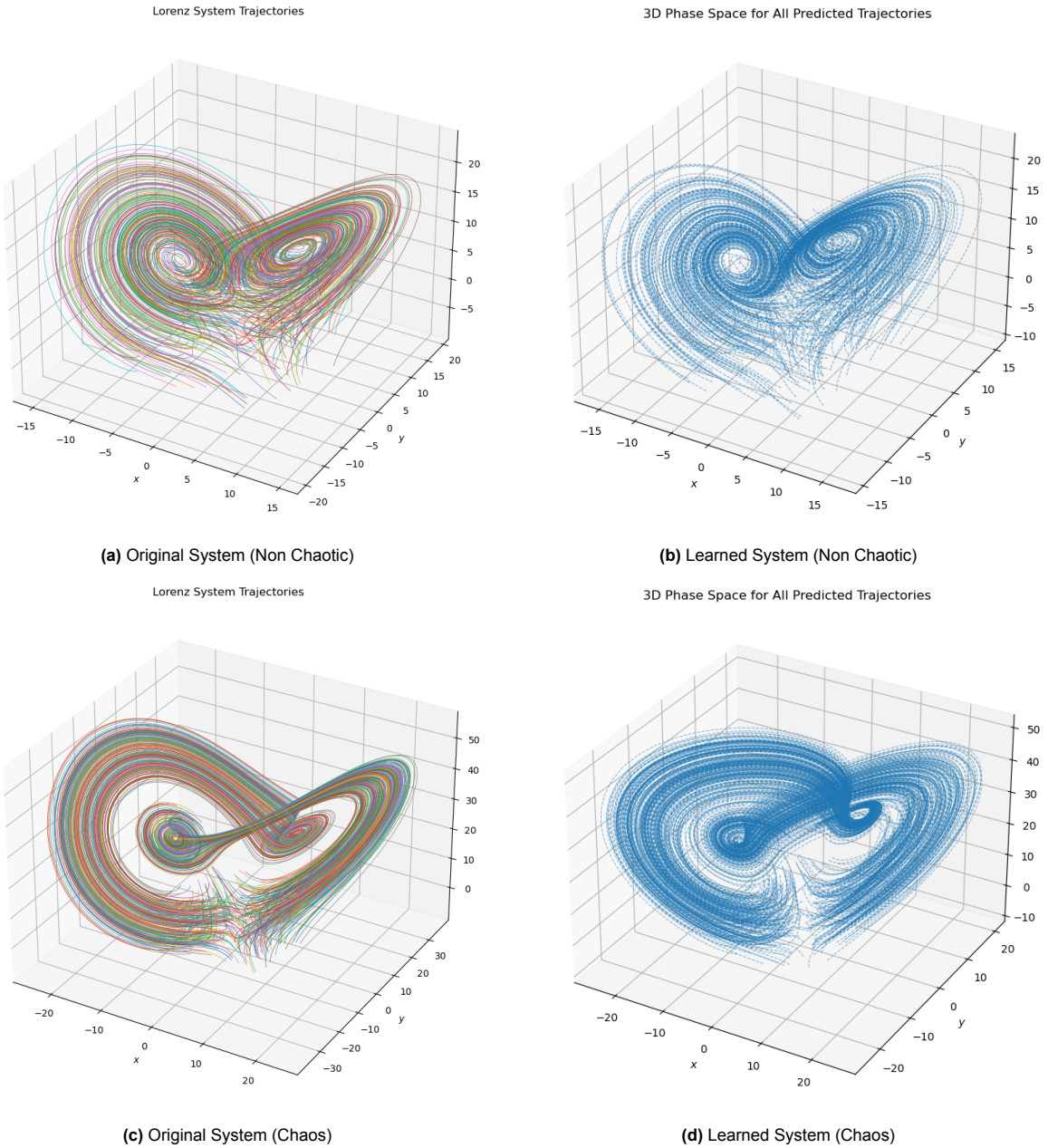


(f) $\mu = -1$ **(g)** $\mu = -3$

Figure 5: Fixed points of the Van der Pol system for various values of μ.

5.4. Benchmarking with SINDy

To benchmark our results against a state-of-the-art method to solve such problems, we ran the Sparse Identification of Nonlinear Dynamics (SINDy) [4], followed by numerical integration using a 4th-order Runge-Kutta solver. SINDy produces interpretable symbolic equations of the system dynamics from trajectory information. As seen in Table 2, SINDy outperforms Neural ODEs across all systems and metrics — especially in the chaotic regime of the Lorenz system — and does so with orders of magnitude less compute time. However, the trade-off is its reliance on a predefined library of base functions, which may limit its ability to model more complex, highly nonlinear, or noisy systems. Also, since the model



(a) Original System (Non Chaotic)

(b) Learned System (Non Chaotic)

(c) Original System (Chaos)

(d) Learned System (Chaos)

Figure 6: Original and Learned phase space for the Lorenz system.

is sparse, there's a tendency to prefer simple dynamics, which might bias the system away from more accurate but complex models.

5.5. Activation Functions

We used the Lorenz system from Section 4.1.3 in the chaotic regime and varied the activation function. While varying the activation function, we made sure to keep the data and the model architecture the same. As we can see from Table 3, SiLU performed the best in all of our error metrics which is why we used this activation in all of our results in this report.

6. Conclusion

In this work, we investigated the use of Neural Ordinary Differential Equations (Neural ODEs) as a data-driven approach to learning the underlying vector field of dynamical systems directly from tra-

System	Model	MSE	MAE	Chamfer	Hausdorff	Time Taken(s)
Homoclinic	Neural ODE	0.0008	0.02	0.001	0.07	2514
	SINDy + RK4	8.9e-10	2.06e-05	3.58e-09	6.64e-05	0.16
Lorenz (Non Chaotic)	Neural ODE	0.55	0.41	1.52	1.38	1689
	SINDy + RK4	0.0005	0.015	0.003	0.05	0.19
Lorenz (Chaos)	Neural ODE	18.09	2.74	54.63	11.13	1738
	SINDy + RK4	0.28	0.19	0.29	0.81	0.18

Table 2: Comparison of SINDy (with RK4) and Neural ODE for different systems using trajectory-matching metrics.

Name	MSE	MAE	Chamfer	Hausdorff
SiLU	19.01	2.75	49.47	11.19
LeakyReLU	37.55	4.12	102.39	14.81
Softplus	40.53	4.52	114.37	15.64
GELU	47.90	4.75	130.29	16.59
ELU	49.57	4.79	128.33	16.50
ReLU	49.93	4.71	124.72	15.99
Tanh	68.03	5.45	206.03	28.84
Sigmoid	81.09	6.31	255.10	31.64

Table 3: Comparison of activation functions on different error metrics (run on Lorenz in the chaotic regime).

jectory data. We assessed whether these learned dynamics could reliably identify and analyse fixed points and their stability. Our results show that Neural ODEs can effectively recover fixed points and local phase portraits in low-dimensional systems such as the 2D homoclinic system. In these settings, they demonstrate strong interpolation capabilities within the training region. However, performance degrades during extrapolation, especially in fast-changing regions of the phase space or where training data is sparse, which is consistent with known limitations of neural integrators.

We also evaluated the model on more challenging systems like the 3D Lorenz attractor, where Neural ODEs struggled to capture global structure and long-term behaviour in the chaotic regime, highlighting limitations in modelling chaotic dynamics. In contrast, Sparse Identification of Nonlinear Dynamics (SINDy), paired with RK4 integration, consistently outperformed Neural ODEs in terms of accuracy and computational efficiency, even in chaotic regimes. It is important to note that our study used relatively simple neural architectures and did not extensively tune hyperparameters. This likely contributed to some of the observed limitations.

To understand how activation functions influence performance, we benchmarked several options and found that SiLU provided the most robust results in the chaotic setting. We also demonstrated that Neural ODEs are capable of qualitatively capturing bifurcation behaviour, as evidenced in the Van der Pol oscillator.

Future work could employ deeper or more expressive networks, have systematic hyperparameter optimisation, and evaluate performance on real-world experimental data. Moreover, extensions such as autoencoder-based dimensionality reduction, GPU acceleration, or learning one-step transitions rather than full trajectories could further improve performance and scalability.

Overall, our study highlights both the potential and the limitations of Neural ODEs in learning the vector fields of dynamical systems, and points to future avenues for improving model fidelity in challenging regimes, expanding their applicability in nonlinear dynamics modelling.

7. Code Availability and AI Usage

Code is available at Gitlab. Other systems can be seen in the appendix folder. In this report, we use AI tools (Grammarly) to correct grammatical errors. For the code, we used AI tools (ChatGPT) to help make the structure and to help with comments.

References

[1] S. H. Strogatz. *Nonlinear Dynamics and Chaos: with Applications to Physics, Biology and Chemistry*. 2nd ed. CRC Press, 2015.

[2] S.L. Brunton and J.N. Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. 2nd ed. Cambridge: Cambridge University Press, 2022.

[3] Ricky T. Q. Chen et al. *Neural Ordinary Differential Equations*. 2019. arXiv: 1806.07366 [cs.LG]. URL: https://arxiv.org/abs/1806.07366.

[4] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". In: *Proceedings of the National Academy of Sciences* 113.15 (Mar. 2016), pp. 3932–3937. ISSN: 1091-6490. DOI: 10.1073/pnas.1517384113. URL: http://dx.doi.org/10.1073/pnas.1517384113.

[5] Andreas Sochopoulos, Michael Gienger, and Sethu Vijayakumar. *Learning Deep Dynamical Systems using Stable Neural ODEs*. 2024. arXiv: 2404.10622 [cs.RO]. URL: https://arxiv.org/abs/2404.10622.

[6] Wenjie Mei, Dongzhe Zheng, and Shihua Li. *ControlSynth Neural ODEs: Modeling Dynamical Systems with Guaranteed Convergence*. 2024. arXiv: 2411.02292 [cs.LG]. URL: https://arxiv.org/abs/2411.02292.

[7] Daniel Ayers et al. "Supervised machine learning to estimate instabilities in chaotic systems: Estimation of local Lyapunov exponents". In: *Quarterly Journal of the Royal Meteorological Society* 149.753 (2023), pp. 1236–1262. DOI: https://doi.org/10.1002/qj.4450. eprint: https://rmets.onlinelibrary.wiley.com/doi/pdf/10.1002/qj.4450. URL: https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.4450.

[8] Ferdinand Verhulst. *Nonlinear Differential Equations and Dynamical Systems*. 2nd ed. Universitext. Springer Berlin, Heidelberg, Sept. 1996, pp. X, 306. ISBN: 978-3-540-60934-6. DOI: 10.1007/978-3-642-61453-8. URL: https://doi.org/10.1007/978-3-642-61453-8.

[9] Patrick Kidger. *On Neural Differential Equations*. 2022. arXiv: 2202.02435 [cs.LG]. URL: https://arxiv.org/abs/2202.02435.

[10] Phillip Lippe. *UvA Deep Learning Tutorials*. https://uvadlc-notebooks.readthedocs.io/en/latest/. 2024.

[11] Aiqing Zhu et al. *On Numerical Integration in Neural Ordinary Differential Equations*. 2022. arXiv: 2206.07335 [cs.LG]. URL: https://arxiv.org/abs/2206.07335.

[12] A. Sholokhov, Y. Liu, and H. et al. Mansour. "Physics-informed neural ODE (PINODE): embedding physics into models using collocation points". In: *Nature* (2023).

[13] Tianxiang Gao et al. "Global Convergence in Neural ODEs: Impact of Activation Functions". In: *The Thirteenth International Conference on Learning Representations*. 2025.