Deep Learning for Partially Observed Dynamical Systems

Thibault Monsel

université PARIS-SACLAY

Deep Learning for Partially Observed Dynamical Systems

Apprentissage profond pour les systèmes dynamiques partiellement observables

Thèse de doctorat de l'université Paris-Saclay

École doctorale n°580, Science et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat : Informatique
Graduate School : Informatique et sciences du numérique
Référent : Faculté des sciences d'Orsay

Thèse préparée dans les unités de recherche Laboratoire Interdisciplinaire des Sciences du Numérique (Université Paris-Saclay, CNRS) et INRIA Saclay-Ile-de-France (Université Paris-Saclay, INRIA), sous la direction de **Alexandre ALLAUZEN**, professeur, avec le co-encadrement de **Guillaume CHARPIAT** chargé de recherche et **Onofrio SEMERARO**, chargé de recherche.

Thèse soutenue à Paris-Saclay, le 20 décembre 2024, par

Thibault MONSEL

Composition du jury

Membres du jury avec voix délibérative

Mathilde MOUGEOT Professeur, ENS Paris Saclay	Presidente du jury
Benjamin SANDERSE Associate Professor (équiv.HDR), Head of the Scientific Computing group of Centrum Wiskunde, Informatica (CWI)	Rapporteur & Examinateur
Luca MAGRI Full Professor of Scientific Machine Learning Department of Aeronautics (équiv.HDR) - Faculty of Engineering, Imperial College London	Rapporteur & Examinateur
Nicolas THOME Professeur, Sorbonne Université	Examinateur
Patrick KIDGER Researcher, Cradel.bio, Imperial College London	Examinateur
Camilla FIORINI Maître de conférences, CNAM	Examinatrice

THESE DE DOCTORAT
NNT : 2024UPASG113

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Apprentissage profond pour des systèmes dynamiques partiellement observables
Mots clés : Apprentissage profond, Systèmes dynamiques, Systèmes partiellement observés, Équations Différentielles Ordinaires neuronales, Équations Différentielles à Retardement

Résumé : Les équations différentielles partielles (EDP) sont la pierre angulaire de la modélisation des systèmes dynamiques dans diverses disciplines scientifiques. Traditionnellement, les scientifiques utilisent une méthodologie rigoureuse pour interagir avec les processus physiques, collecter des données empiriques et dériver des modèles théoriques. Cependant, même lorsque ces modèles correspondent étroitement aux données observées, ce qui n'est souvent pas le cas, les simplifications nécessaires à l'étude et à la simulation peuvent obscurcir notre compréhension des phénomènes sous-jacents.

Cette thèse explore la manière dont les données acquises à partir de systèmes dynamiques peuvent être utilisées pour améliorer et/ou dériver de meilleurs modèles. Le manuscrit se concentre particulièrement sur les dynamiques partiellement observées, où l'état complet du système n'est pas complètement mesuré ou observé. Grâce à la théorie des systèmes partiellement observés, y compris le formalisme de Mori–Zwanzig et le théorème de Takens, nous motivons une structure non-markovienne, en particulier les équations différentielles à retardement (EDR).

En combinant le pouvoir d'expression des réseaux neuronaux avec les EDR, nous proposons de nouveaux modèles pour les systèmes partiellement observés. Comme les EDP basées sur les réseaux neuronaux (EDP neuronales) en sont encore à leurs débuts, nous étendons l'état actuel de l'art dans ce domaine en étudiant et en comparant les modèles d'EDP neuronales avec des types de retard arbitraires connus a-priori à travers une variété de systèmes dynamiques. Ces références incluent des systèmes avec des retards dépendant du temps et de l'état. Sur la base de ces études, nous explorons ensuite la paramétrisation des retards constants dans les EDP neuronales. Nos résultats démontrent que l'introduction de retards constants pouvant être appris, par opposition à des configurations de retards fixes, permet d'améliorer les performances globales de la modélisation et de l'ajustement des systèmes dynamiques.

Nous appliquons ensuite les EDP neurales non markoviennes avec des retards constants pouvant être appris à la modélisation de la fermeture et de la correction des systèmes dynamiques, en démontrant une meilleure précision à long terme par rapport aux termes des équations différentielles ordinaires. Enfin, nous explorons l'utilisation des EDR neuronales dans le contexte de la commande prédictive de modèle pour le contrôle des systèmes dynamiques.

Title : Deep Learning for Partially Observed Dynamical Systems
Keywords : Deep Learning, Dynamical Systems, Partially observed systems, Neural ODEs, Delay Differential Equations

Abstract : Partial Differential Equations (PDEs) are the cornerstone of modeling dynamical systems across various scientific disciplines. Traditionally, scientists employ a rigorous methodology to interact with physical processes, collect empirical data, and derive theoretical models. However, even when these models align closely with observed data, which is often not the case, the necessary simplifications made for study and simulation can obscure our understanding of the underlying phenomena.

This thesis explores how data acquired from dynamical systems can be utilized to improve and/or derive better models. The manuscript focuses particularly on partially observed dynamics, where the system's full state is not completely measured or observed. Through the theory of partially observed systems, including the Mori–Zwanzig formalism and Takens' theorem, we motivate a non-Markovian structure, specifically Delay Differential Equations (DDEs).

By combining the expressive power of neural networks with DDEs, we propose novel models for partially observed systems. As neural network-based DDEs (Neural DDEs) are still in their infancy, we extend the current state of the art in this field by studying and benchmarking Neural DDE models with a-priori known arbitrary delay types across a variety of dynamical systems. These benchmarks include systems, with time-dependent and state-dependent delays. Building upon these investigations, we then explore the parameterization of constant delays in Neural DDEs. Our findings demonstrate that introducing learnable constant delays, as opposed to fixed delay configurations, results in improved overall performance in dynamical system modeling and fitting.

We then apply the non-Markovian Neural DDEs with learnable constant delays to dynamical system closure and correction modeling, demonstrating improved long-term accuracy compared to Ordinary Differential Equation terms. Lastly, we explore the use of Neural DDEs in the context of Model Predictive Control (MPC) for controlling dynamical systems.

Table des matières

Table des figures

Nomenclature

Symbols

$\lambda(t)$	The adjoint state
$[\![1, N]\!]$,	Set of integers between 1 and N
$\mathbb{R}^+$	The set of strictly positive real numbers
$\mathbb{R}^{d_1 \times \cdots \times d_k}$	The space of tensors of shape $d_1 \times \cdots \times d_k$
$\mathcal{A}, \mathcal{U}$	Banach spaces
$\mathcal{B}$	Operator associated with the boundary and/or initial conditions of a PDE
$\mathcal{D}$	Differential operator associated to a PDE
$\mathcal{F}$	Fourier Transform
$\mathcal{G}$	infinitesimal generator of a dynamical system
$\mathcal{K}$	Koopman operator or Kernel integral operator
$\mathcal{L}$	Liouville operator
$\mathcal{M}_d(\mathbb{R})$	Set of $d \times d$ matrices with real entries
$\mathcal{P}$	Projector operator
$\mathcal{Q}$	Orthogonal projector operator
$\left.\frac{\partial f_\theta}{\partial x}\right\|_t$	The partial derivative of f_θ with respect to x evaluated at time t
$\frac{\partial f_\theta}{\partial x}$	The partial derivative of f_θ with respect to x
$\psi(t)$ or $\phi(t)$	Delay Differential Equation's history function
τ	The time delay
θ	The neural network's parameters
$g(t)$	The observable function
$x(t)$ or $y(t)$	The full dynamical system's state

Abbreviations

AD	Autodifferentiation
ANODE	Augmented Neural ODE
CEM	Cross Entropy Method
DAE	Differential Algebraic Equations
DDE	Delay Differential Equations
DMD	Dynamics Mode Decomposition
GLE	Generalized Langevin Equation
GRU	Gated Recurrent Unit
IDE	Integro Differential Equations
JVP	Jacobian Vector Product
KS	Kuramoto Sivashinsky
LSTM	Long Short Term Memory
MLP	Multi Layer Perceptron
MPC	Model Predictive Control
MZ	Mori–Zwanzig
NCDE	Neural Controlled Differential Equations
NDDE	Neural DDE
NHH	Hamiltonian neural networks
NO	Neural Operator
NODE	Neural ODE
NPCDDE	Neural Piecewise Constant DDE
NSDDDE	Neural State dependent DDE
ODE	Ordinary Differential Equations
PINN	Physics Informed Neural Network
RNN	Recurrent Neural Networks
VJP	Vector Jacobian Product
w.r.t	with respect to

1 - Introduction

The early 20th century saw quantum mechanics revolutionize our understanding of the physical world, especially at atomic and subatomic scales. This advancement brought forth the challenge of accurately modeling open quantum systems, which interact with their environment. A significant breakthrough in this field occurred in the 1960s with Felix Bloch and his colleagues' work on nuclear magnetic resonance (NMR) [65]. NMR spectroscopy had become a powerful tool for studying molecular structure and dynamics, but researchers noticed that observed relaxation times of nuclear spins often deviated significantly from simple quantum model predictions.

Initial spin relaxation models were based on Markovian assumptions, which posited that a system's future state depended only on its present state, not its history [53]. These models treated the interaction between the spin system and its environment as a series of uncorrelated, random perturbations. However, these Markovian models failed to accurately predict the behavior of many real quantum systems, particularly in complex molecular environments.

Bloch and others realized that these discrepancies arose because they were dealing with a partially observed system [19]. While they could measure nuclear spin behavior, they couldn't directly observe all the complex interactions with the surrounding environment. Moreover, they discovered that the history of these interactions – information not captured in Markovian models – was crucial for accurate predictions.

This realization led to the development of more sophisticated models incorporating non-Markovian effects. These models accounted for memory effects in the system-environment interaction, recognizing that past states could influence future evolution in ways not captured by purely present-state dependence. The inclusion of non-Markovian terms dramatically improved prediction accuracy in NMR spectroscopy and other quantum systems. It allowed researchers to better model phenomena such as spectral line shapes, coherence transfer, and relaxation processes in complex molecular systems.

This introductory example effectively demonstrates the critical need to acknowledge and account for the constraints of our observational abilities when constructing models. It underscores the significance of integrating these limitations into our theoretical frameworks to ensure more accurate and comprehensive representations of complex systems.

Thus, the aim of this doctoral thesis is to design methods that can accurately model complex dynamical systems, particularly those with partially observed dynamics. To this end, we propose different data-driven approaches that can capture both Markovian and non-Markovian contributions to system dynamics. We focus on developing Delay Differential Equations (DDEs) and their applications in physics. DDEs provide a natural framework for capturing non-Markovian effects, as they explicitly account for delays in system evolution. By integrating Delay Differential Equations (DDEs) with the power and flexibility of neural networks, we open up new possibilities for creating sophisticated, data-driven models. This synergistic approach allows us to construct highly adaptable frameworks capable of capturing and predicting the intricate dynamics of a diverse array of systems. The marriage of DDEs and neural networks enables us to leverage the strengths of both methodologies : the explicit time-delay modeling of DDEs and the complex pattern recognition abilities of neural networks.

Prerequistes

This thesis is aimed at a broad audience coming from a STEM background but certain prerequisites are strongly recommended, if not required. Among those, the reader should have some familiarity with Ordinary Differential Equations (ODEs), their numerical integration and the basics of modern deep learning. Apart from these assumptions, other concepts will be introduced and based off these requisites. Several Appendices are included to provide an introduction or review on these introduced topics.

Main Contributions

The material in this thesis is either (a) original work conducted during the PhD with or without collaborators, or (b) where relevant prior or concurrent work included for reference, to provide a survey of the field.

The primary contributions of this thesis are centered on the comprehensive development of delay differential equations and their applications within machine learning and physics.

- **Time and State Dependent Neural DDEs :** This contribution focuses on enhancing Neural DDEs through the incorporation of delays that depend on time and/or state. This enhancement enables the representation of systems characterized by diverse types of delays, which are prevalent in practical applications such as biology and engineering. The

resulting general-purpose Neural DDEs are compared with other existing models, demonstrating greater expressiveness and accuracy.

- **Neural DDEs with Learnable Delays for Partially Observed Systems :** This contribution expands Neural DDEs by introducing learnable delays for dynamical systems that are only partially observed. The adjoint method is meticulously derived for Neural DDEs featuring learnable delays. Moreover, the Mori–Zwanzig formalism establishes a connection between Neural DDEs and partially observed systems. The ability to learn delays adds a layer of complexity and expressivity to the model. These Neural DDEs with learnable parameters are evaluated against other models in literature, proving to be more accurate. Additionally, an open-source library named `torchdde` is developed for the implementation of Neural DDEs.

- **Non Markovian modeling :** This contribution mainly functions as an application of the research outlined in the thesis. Neural DDEs with learnable delays are employed to model non-Markovian closure (or correction) terms in physical surrogate models. When compared to ODE-type closure (or correction) models, the proposed model shows enhanced accuracy in representing the system's dynamics. This application also demonstrates that the package `torchdde` can be used to model complex physical systems and scale with large networks (≈ 8 million parameters were used for an experiment).

Outline

The thesis is organized around the scientific contributions that have been produced over the course of the PhD. The first two chapters serve as introductions to the field of Neural Differential Equations and dynamical systems' modeling. The following chapters are dedicated to the main contributions of the thesis. The final chapter concludes the thesis and provides a discussion on future work. We provide below a summary of each chapter :

- **Chapter 2** introduces Neural Differential Equations and offers a survey of the field. Neural ODEs, DDEs and Integro-Differential Equations (IDEs) are introduced along with their training procedures (regular backpropagation and the adjoint method).

- **Chapter 3** provides an introduction to dynamical systems and their modeling. The chapter covers the theoretical basics of the Mori–Zwanzig formalism, Takens' theorem and the Koopman operator. These concepts

are essential for the understanding of the motivations of the following chapters. All the concepts are introduced in a self-contained manner and introduced in their continuous-time form.

- **Chapter 4** is based on the following publication :

  T.Monsel, O.Semeraro, L.Mathelin, and G.Charpiat, "Time and State Dependent Neural Delay Differential Equations", Proceedings of Machine Learning Research 255 :1–20, 2024 ML-DE Workshop at ECAI 2024 https://arxiv.org/pdf/2306.14545.

  This publication presents the initial contribution of the thesis with the goal of adapting Neural DDEs to incorporate known arbitrary types of delays a priori. This enhancement is integrated into the existing literature and is demonstrated to address a more general test case than the previously introduced Neural DDEs. The model is assessed using a series of benchmarks and compared to established models.

- **Chapter 5** is based on the following pre-print submitted for publication :

  T.Monsel, E.Menier, O.Semeraro, L.Mathelin, and G.Charpiat, "Neural DDEs with Learnable Delays for Partially Observed Dynamical Systems", pre-print, 2024. [121]

  This pre-print presents the extension of Neural DDEs to include learnable delays for partially observed systems. The adjoint method is derived for Neural DDEs with learnable delays, and the Mori–Zwanzig formalism is used to establish a connection between Neural DDEs and partially observed systems. The model is evaluated against other models in the literature, demonstrating its competitive performance.

- **Chapter 6** is based on the undergoing research to be submitted soon :

  T.Monsel, O.Semeraro, L.Mathelin, and G.Charpiat, "Non-Markovian closure or correction modeling for dynamical systems", in progress, 2024.

  This chapter presents the application of Neural DDEs with learnable delays to model non-Markovian closure (or correction) terms in physical surrogate models. The chapter demonstrates the model's ability to accurately represent the system's dynamics and outperform ODE-type closure (or correction) models. Theoretical links are made with the Mori–Zwanzig formalism.

- **Chapter 7** Concluding remarks of the thesis.

Acknowledgements

The completion of my PhD thesis would not have been possible without the support and guidance of many individuals and institutions. I am deeply grateful to all who have assisted me throughout this journey.

First, I would like to express my deepest gratitude to my advisors, Guillaume, Onofrio and Lionel for their invaluable guidance, support, and encouragement. Their expertise and insights have greatly enriched my work, and their belief in my abilities has been a constant source of motivation.

My heartfelt thanks go to my colleagues and friends in the lab, particularly Remy, Amine, Alice, Matthieu, Emmanuel, Lucas, Manon and Michele. Their camaraderie and the stimulating discussions we shared have been an essential part of my academic journey.

I am profoundly grateful to my family for their unwavering support and encouragement. To my parents, Veronique and Eric, thank you for your constant love and sacrifice. To my siblings, Pierre and Guillaume, thank you for your understanding, patience and support. Lastly, to all my friends, Guillaume, Gregoire, Lena, Clement C, Clement T, Astrid, Paul, Mathilde, Nina, Aubry, Thomas, Julien, Benoit, Arnaud, Nicolas and many others that have been there for these three years, thank you.

Finally, I would like to dedicate this thesis to LISN lab and TAU team, whose influence and inspiration have driven me to undertake and complete this challenging work.

Thank you all.

2 - Neural Differential Equations

This chapter introduces the concept of Neural Differential Equation (synonymous to continuous-depth models), a class of neural networks that leverages continuous time transformations to model data. We begin by defining the Neural ODE model, the pioneering Neural Differential Equation (DE), along with various adaptations. Next, we explain the process of backpropagation for these models. Then, we highlight the original contributions of this thesis, focusing on the comprehensive development and establishment of Neural DDEs. Finally, we present another type of Neural DE, Neural Integro-Differential Equation (IDE) and other deep learning models that are loosely related to Neural DEs.

Remark 2.0.1. *This thesis does not cover any type of Stochastic Differential Equations (SDEs [182]) since no stochastic systems were studied.*

2.1 . Introduction

Continuous-depth models represent a type of neural network that stands apart from "traditional" Deep Learning models through their use of continuous transformations. The specific term "continuous-depth model" was introduced in the seminal Neural ODE paper [29], marking the inception of the first *Neural DE* model of its kind. A Neural ODE is defined as :

$$\frac{dx(t)}{\mathrm{d}t} = f_\theta(t, x(t)), \quad x(0) = x_0 \tag{2.1}$$

where θ represent the parameters, $f_\theta : \mathbb{R} \times \mathbb{R}^{d_1 \times \cdots \times d_k} \to \mathbb{R}^{d_1 \times \cdots \times d_k}$ is an arbitrary neural network, $x_0 \in \mathbb{R}^{d_1 \times \cdots \times d_k}$ the initial condition and $x : [0, T] \to \mathbb{R}^{d_1 \times \cdots \times d_k}$ is the equation's solution.

In 2015, the groundbreaking *Residual Network* (Resnet) model introduced the use of residual connections in neural network layers [68]. Resnet's discrete transformations are :

$$x_{j+1} = x_j + f_{\theta_j}(x_j) \tag{2.2}$$

where f_{θ_j} is the $j - th$ residual block. These iterative updates can be seen as an explicit Euler discretization of a continuous transformation [61, 157, 109]. If we discretize Equation 2.1 with an explicit Euler scheme at time t_j uniformly separated by Δt, we obtain the following update rule :

$$x(t_{j+1}) = x(t_j) + \Delta t f_\theta(t_j, x(t_j)).$$

Incorporating Δt with f_θ, we get the Resnet's residual block Equation 2.2. With this in mind, we clearly see that neural ODEs are the continuous limit of a residual network. Moreover, many popular deep learning architectures like the GRU [31] (Remark 2.1.1 provides the derivation of GRUs's continuous-time formulation), the LSTM [70] or invertible coupling layers [11] can be seen as discretized differential equations.

Remark 2.1.1. *Recall that a GRU is defined with the following equations with the matrix weights* $\{W_i\}_{i=1}^6$ *and biases* $\{b_i\}_{i=1}^4$ *:*

$$\begin{aligned} i_j &= \sigma(W_1 x_j + W_2 h_j + b_1) \\ r_j &= \sigma(W_3 x_j + W_4 h_j + b_2) \\ n_j &= \tanh(W_5 x_j + b_3 + r_j \odot (W_6 h_j + b_4)) \\ h_{j+1} &= n_j + i_j \odot (h_j - n_j) \end{aligned} \tag{2.3}$$

for the input data x_j *and its associated hidden state* h_j*. Here* $\odot$ *denotes the element-wise product and* σ *is the sigmoid function.*

The GRU cell is an explicit Euler discretization of the following continuous-time differential equation :

$$\begin{aligned} i(t) &= \sigma(W_1 x(t) + W_2 h(t) + b_1) \\ r(t) &= \sigma(W_3 x(t) + W_4 h(t) + b_2) \\ n(t) &= \tanh(W_5 x(t) + b_3 + r(t) \odot (W_6 h(t) + b_4)) \\ \frac{dh(t)}{\mathrm{d}t} &= (1 - i(t)) \odot (n(t) - h(t)) \end{aligned} \tag{2.4}$$

2.2 . The Neural ODE family

In this subsection, we will delve into several notable and widely-used Neural ODE architectures, including Augmented Neural ODE [44], Latent ODE [153], Hamiltonian Neural Networks [57]. It is important to note that there are many specific variations of Neural ODEs and will not cover all of them.

2.2.1 . Vanilla Neural ODE

A Neural ODE, introduced in Chen et al. [29] (the first instance of it was actually in [148]), is defined as :

$$\frac{dx(t)}{\mathrm{d}t} = f_\theta(t, x(t)), \quad x(0) = x_0 \tag{2.5}$$

where θ represent the parameters, $f_\theta : \mathbb{R} \times \mathbb{R}^{d_1 \times \cdots \times d_k} \to \mathbb{R}^{d_1 \times \cdots \times d_k}$ is an arbitrary neural network, $x_0 \in \mathbb{R}^{d_1 \times \cdots \times d_k}$ is an arbitrary tensor and $x : [0, T] \to \mathbb{R}^{d_1 \times \cdots \times d_k}$ is the equation's solution.

The model's existence and uniqueness is guaranteed by the Cauchy–Lipschitz theorem given that f_θ is Lipschitz continuous, something that is often true for neural networks [186].

Theorem 2.2.1 (Cauchy–Lipschitz Theorem)**.** *Let $f : [0, T] \times \mathbb{R}^d \to \mathbb{R}^d$ be continuous in t and uniformly Lipschitz in x . Let $x_0 \in \mathbb{R}^d$. Then, there exist a unique differentiable function $x : [0, T] \to \mathbb{R}^d$ such that :*

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = f(t, x(t)), \quad x(0) = x_0.$$

This is a well established result and the proof can be found in any standard textbook on differential equations [23].

Remark 2.2.2. *In the Augmented Neural ODE paper [44] introduced hereafter, authors present a counter-example demonstrating that Neural ODEs cannot represent certain types of functions (Section 3 of [44]), indicating that NODEs are not a universal approximator. They design a problem where ODE trajectories must intersect, something impossible due to Theorem 2.2.1.*

2.2.2 . Augmented Neural ODEs

Augmented NODEs (ANODEs) were able to alleviate NODEs' expressivity bottleneck by augmenting the dimension of the space allowing the model to learn more complex functions using simpler flows [44]. Let $a(t) \in \mathbb{R}^p$ denotes a point in the augmented space, the ODE problem is formulated as

$$\frac{d}{\mathrm{d}t}\begin{bmatrix} x(t) \\ a(t) \end{bmatrix} = f_\theta\left(t, \begin{bmatrix} x(t) \\ a(t) \end{bmatrix}\right), \quad \begin{bmatrix} x(0) \\ a(0) \end{bmatrix} = \begin{bmatrix} x_0 \\ 0 \end{bmatrix}. \tag{2.6}$$

Regarding their expressive power, Augmented Neural ODEs are universal approximators, even if their vector fields themselves are not necessarily universal approximators ([80] App C.1).

At times, ANODEs are regarded as being non Markovian. I do not fully agree with this perspective and believe it's essential to clarify this matter. When considering the augmented full state $[x(t), a(t)]$, the model calculates the next state by only using the current state $[x(t), a(t)]$ and the present time t. This exemplifies a Markovian characteristic. While the augmented variable $a(t)$ may be somewhat opaque, it does not alter the Markovian nature of the model. The opposing point of view sees that lifting into a higher-dimensional space may be regarded as a relaxation of the Markov property because for $s < t$ the output $x(s)$ does not completely determine $x(t)$ (because it is determined by $[x(t), a(t)]$).

2.2.3 . Latent ODEs

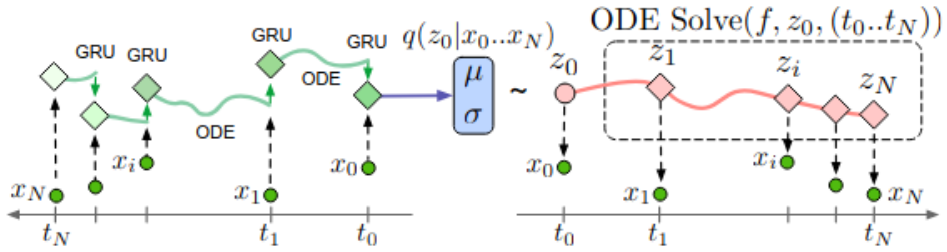

Figure 2.1 – Latent ODE model with an ODE-RNN encoder. The ODE-RNN encoder runs backward in time to produce an approximate posterior $q(\mathbf{z}_0|\{(\mathbf{x}(t_i), t_i\}_{i=0}^N)$ over the initial latent state $\mathbf{z}_0$ conditioned on the dataset $\{(\mathbf{x}(t_i), t_i\}_{i=0}^N$. Given a sample $\mathbf{z}_0$, the ODE decoder integrates the latent state forward in time to produce the output that is then projected into the space where the dataset lives in. Figure taken from [153].

The variational-autoencoder model uses an ODE-RNN encoder and ODE decoder architecture to construct a continuous time model with a latent state defined at all times [153]. The ODE-RNN encoder runs backward in time to produce h_i hidden state features at each time t_i that combines the ODE integration and RNN cell operation which is outlined in Algorithm 1 and displayed in Figure 2.1.

Algorithm 1 ODE-RNN

Input : Data points and their timestamps $\{(x_i, t_i)\}_{i=1..N}$
$h_0 = \mathbf{0}$
for i in N, N-1, . . ., 1 **do**
    $h'_i = \text{ODESolve}(f_\theta, h_{i-1}, (t_{i-1}, t_i))$ {Solve ODE to get state at t_i}
    $h_i = \text{RNNCell}(h'_i, x_i)$ {Update hidden state given current observation x_i}
end for
$o_i = \text{OutputNN}(h_i)$ for all $i = 1..N$
Return : $\{o_i\}_{i=1..N}; h_N$

The mean and standard deviation of the approximate posterior $q(\mathbf{z}_0|\{(\mathbf{x}(t_i), t_i\}_{i=0}^N)$ are function of the final hidden state of the ODE-RNN :

$$q(\mathbf{z}_0|\{(\mathbf{x}(t_i), t_i\}_{i=0}^N) = \mathcal{N}(\mu_{\mathbf{z}_0}, \sigma_{\mathbf{z}_0})$$
$$\text{where} \quad \mu_{\mathbf{z}_0}, \sigma_{\mathbf{z}_0} = g(\text{ODE-RNN}_\phi(\{(\mathbf{x}(t_i), t_i\}_{i=0}^N)$$

where g is a neural network translating the final hidden state of the ODE-RNN encodes into the mean and variance of $\mathbf{z}_0$. Then, the sample $\mathbf{z}_0$ is integrated in time and projected back onto the original space to get the output.

Compared to ANODEs, Latent ODE is clearly a non-Markovian model because of the latent state $\mathbf{z}_0$ and the ODE-RNN.

2.2.4 . Hamiltonian Neural Networks

Hamiltonian neural networks (HNN) aim to learn the Hamiltonian operator $\mathcal{H}$ of a system [57]. This approach is particularly relevant as numerous physical systems adhere to Hamiltonian dynamics. In this framework, generalized coordinates $p \in \mathbb{R}^d$ and $q \in \mathbb{R}^d$, representing momentum and position respectively, evolve according to the following ODE :

$$\begin{aligned} \frac{\mathrm{d}p(t)}{\mathrm{d}t} &= -\frac{\partial \mathcal{H}(p,q)}{\partial q} \\ \frac{\mathrm{d}q(t)}{\mathrm{d}t} &= +\frac{\partial \mathcal{H}(p,q)}{\partial p} \end{aligned} \tag{2.7}$$

Various adaptations of the HNN concept have been proposed. For instance, Duong and Atanasov [43] developed a version of HNN on the $SE(3)$ manifold to model rigid body dynamics. Desai et al. [38] extended Hamilton's equations to incorporate energy dissipation and external control inputs in dynamical systems. Finally, David and Méhats [37] leveraged the symplectic structure inherent to Hamiltonian systems, introducing a novel loss function that theoretically ensures the existence of an exact Hamiltonian function, which the HNN can then learn.

2.3 . Neural ODEs and Backpropagation

Kidger's thesis *on Neural Differential Equations* [80] is the most comprehensive work on the subject of Neural Differential Equations. Therefore, we decide to adopt the vocabulary used in his manuscript, and more specifically *discretize-then-optimize* and *optimize-then-discretize* jargon.

In order to train a Neural ODE, we need to compute the gradient of the loss with respect to the parameters θ. Two options are available. On the one hand, *discretize-then-optimize* is fast and accurate but memory intensive. On the other hand, *optimize-then-discretize* is slow and less accurate but memory efficient.

For an unfamiliar audience, we provide some background on neural network backpropagation in Appendix A.1. Backpropagation is the cornerstone of Deep

Learning and is used to compute the gradient of the loss function with respect to the parameters of the model. For Neural DEs, backpropagation can be a bit more involved computationally, leading to the development of an alternative approach known as the *optimize-then-discretize*, or adjoint method. This is seen and explained in later subsequent sections of this chapter for each introduced Neural DE model.

Remark 2.3.1. *Some empirical studies have been done to compare both methods mentioned above for Neural ODE [114, 115].*

2.3.1 . Discretize-then-optimize

This corresponds to the "regular backpropagation" we are used to. Given the requirement, that the ODE solver is written with an auto-differentiable package like JAX or PyTorch, we backpropagate through the internal operations of the solver. Advantages and drawbacks of this method are found in Kidger's thesis but we quickly enumerate them [80].

Pros :

- **Accuracy of gradients** The computed gradients will be accurate for the discrete model that is actually being used. This is in contrast to optimize then discretize which computes only approximate gradients.
- **Speed** This is often the quickest way to backpropagate. One reason for this is that the full computation graph is known prior to performing the backpropagation, and so the underlying autodifferentiation library may better exploit parallelism.
- **Ease of implementation** The implementation of discretize-then-optimize is generally straightforward : provided the differential equation solver is written in an AD framework, then gradients may automatically be computed.

Cons :

- **Memory requirements** This approach must store every internal operation of the solver. If the memory cost of recording the operations of a single differential equation step is H, and recalling that T is the time horizon, then this approach consumes $O(HT)$ memory. This is in contrast to the optimize then discretize approach which reduces this to only $O(H)$. However, there are ways to lower the memory footprint of this method, such as using recursive checkpointing [58, 80]. Using a checkpointing strategy, the memory cost can be reduced to $O(H \log T)$ and is in fact the de facto method used in practice (in the `diffrax` library for example).

2.3.2 . Optimize-then-discretize

This approach was first mentioned in the Machine Learning community in Neural ODE paper [29] but is a widely used technique in many scientific communities, notably to do PDE-constrained optimization [143, 62, 18]. Instead of backpropagating through the ODE solver, we differentiate the idealized continuous-time model.

Theorem 2.3.2 (ODE adjoint method)**.** *Let us consider the continuous-depth ODE model below defined with the same hypothesis as in Theorem 2.2.1 :*

$$\frac{dx(t)}{\mathrm{d}t} = f_\theta(t, x(t)), \quad x(0) = x_0 \tag{2.8}$$

and the following loss function L :

$$L(x(T)) = \int_0^T l(x(t))\,\mathrm{d}t.$$

The gradient's loss w.r.t. the parameters is given by :

$$\frac{\mathrm{d}L}{\mathrm{d}\theta} = -\int_0^T \lambda(t)\frac{\partial f_\theta(t, x(t))}{\partial \theta}\mathrm{d}t. \tag{2.9}$$

where the adjoint dynamics $\lambda(t)$ are given by another ODE :

$$\frac{d\lambda(t)}{\mathrm{d}t} = -\lambda(t)\frac{\partial f_\theta(t, x(t))}{\partial x}, \quad \lambda(T) = -\frac{\partial L(x(T))}{\partial x} \tag{2.10}$$

We provide proof of the ODE adjoint theorem 2.3.2 in Appendix A.2. Vector matrix multiplications found in Equation 2.10 and 2.9 can be efficiently computed with Vector Jacobian products (VJPs) in auto-differentiable libraries. For a more in depth discussion on the adjoint method, we refer the reader to Pontryagin [143] and for a more ML context to Kidger [80] and Menier et al. [116].

2.4 . Neural Delay Differential Equations

Before diving into the section, the reader must be familiar with the concept of Delay Differential Equations (DDEs). If this isn't the case, we provide the following literature [13, 12, 206] or our "quick" DDE introduction in Appendix A.6 that aims at showing the common challenges encountered in integration with DDE solvers and how it differs from ODEs.

2.4.1 . Introduction

This new type of neural differential equations aims at modeling DDEs with neural networks. The first appearance of such an instance dates back to 2021

with the formulation of a single constant delay DDE [200], defined as :

$$\begin{aligned} \frac{dx(t)}{\mathrm{d}t} &= f_\theta(t, x(t), x(t-\tau)) \\ x(t \leq 0) &= \psi(t) \end{aligned} \tag{2.11}$$

where θ represent the parameters, $f_\theta : \mathbb{R} \times \mathbb{R}^{d_1 \times \cdots \times d_k} \times \mathbb{R}^{d_1 \times \cdots \times d_k} \to \mathbb{R}^{d_1 \times \cdots \times d_k}$ is a neural network, $\psi : \mathbb{R} \to \mathbb{R}^{d_1 \times \cdots \times d_k}$ the DDE's history function, $\tau \in \mathbb{R}^+$ the constant delay and $x : [0, T] \to \mathbb{R}^{d_1 \times \cdots \times d_k}$ is the equation's solution.

Remark 2.4.1. *The history function ψ in a DDE is analogous to the initial condition x_0 in an ODE. Specifically, $x(t-\tau)$ for $t \in [-\tau, 0]$ must be provided as input for f_θ to be well-posed.*

Hereafter, the same authors introduced a new model, Neural Piecewise-Constant DDEs (NPCDDEs) [201]. This model is a special case of Neural DDEs where piece-wise constant delays are used [27, 34]. Formally, NPCDDEs is defined as :

$$\begin{aligned} &\frac{dx(t)}{\mathrm{d}t} = f_\theta(t, x(t), x(\lfloor \frac{t}{\tau} \rfloor \tau), \ldots, x(\lfloor \frac{t-n\tau}{\tau} \rfloor \tau)), n \in \mathbb{N}, \tau \in \mathbb{R}^+ \\ &x(-n\tau) = \cdots = x(-\tau) = x(0) = x_0 \end{aligned} \tag{2.12}$$

Remark 2.4.2. *The reader might notice that there isn't any history function here for the piecewise constant DDEs. This is perfectly normal because of the floor function $t \mapsto \lfloor t \rfloor$ properties. Indeed, $t \mapsto (\lfloor \frac{t-n\tau}{\tau} \rfloor \tau)$ is a step function constant on intervals of length $n\tau$. Therefore, the DDE's history function $x(t \leq 0)$ only needs to be defined at multiples of τ.*

Remark 2.4.3. *In the paper's experimental section, the authors only use a single piece-wise constant delay even though NPCDDEs is defined with multiple delays. To the best of our knowledge (since no code is associated to the paper), experimentation of Neural DDEs with several delays has not been done so far in the literature.*

Remark 2.4.4. *To the best of our knowledge, the first implementation appearance of Neural DDE*[1]*. Upon examining it, we discovered that the authors' claim of fitting DDEs with neural networks was inaccurate. This is evident in the code, where the training data is interpolated before integrating the DDE*[2]*. Hence, their modeling choice is the following :*

$$\begin{aligned} \frac{dx(t)}{\mathrm{d}t} &= f(t, x(t), \phi(t-\tau)) \\ x(t \leq 0) &= \psi(t) \end{aligned}$$

1. https://github.com/zhuqunxi/NDDE/blob/main/Examples/Mackey$_Glass/MGlass.py$
2. https://github.com/zhuqunxi/NDDE/blob/1e30916fc7e39f75f9a02fe36571064ab3b5b25d/Examples/Mack$L50C43$

where $\phi(t)$ is a cubic spline interpolator constructed from the training data. Essentially, this reduces to an ODE since ϕ is interpolated. This becomes evident by incorporating $\phi(t-\tau)$ into the time dependence of the vector field, i.e., defining $f_{new}(t, x(t)) = f(t, x(t), \phi(t-\tau))$.

2.4.2 . Neural State and Time Dependent DDEs

In this subsection, we aim to expand the scope of Neural DDEs by ensuring compatibility with a variety of delay types and accommodating multiple delays. We call this model Neural State Dependent DDEs (SDDDEs) and is one of the thesis main contribution [122]. The model is defined as :

$$\begin{aligned} &\frac{dx(t)}{\mathrm{d}t} = f_\theta(t, x(t), x(\alpha_1(t)), \ldots, x(\alpha_k(t))) \\ &\alpha_i(t) = t - \tau_i(t, x(t)), \quad \forall i \in [\![1, k]\!] \\ &x(t \leq 0) = \psi(t), \end{aligned} \tag{2.13}$$

where θ represent the parameters, $f_\theta : \mathbb{R} \times \mathbb{R}^{d_1 \times \cdots \times d_k} \times \ldots \times \mathbb{R}^{d_1 \times \cdots \times d_k} \to \mathbb{R}^{d_1 \times \cdots \times d_k}$ is an arbitrary neural network, $\psi : \mathbb{R} \to \mathbb{R}^{d_1 \times \cdots \times d_k}$ the DDE's history function, $\tau_i : \mathbb{R} \times \mathbb{R}^{d_1 \times \cdots \times d_k} \to \mathbb{R}^+$ the delay functions and $x : [0, T] \to \mathbb{R}^{d_1 \times \cdots \times d_k}$ the equation's solution.

Remark 2.4.5. *What is meant by any type of delays is that the delay function τ_i can be any function of time and/or state or none. This includes time-dependent delays, state-dependent delays, etc. In many applications, such delays arise. Delays feature prominently in real-world scenarios, giving rise to a multitude of applications such as modeling molecular kinetics [152] and diffusion processes [45], as well as in physics for semiconductor laser modeling [187], climate research for El Niño current descriptions [52, 78], infectious disease studies [35], and tsunami forecasting applications [192], among others.*

It is important to note that this Neural SDDDE model is not capable of handling delays that are continuous in the sense of delays that are expressed with integrals as can be found in integro-differential equations. The types of delays that can be handled are listed in Table 2.1.

Table 2.1 – Comparison of DDE implementations with Neural Differential Equations

Delay types	Neural DDE [200]	NPCDDEs [201]	Neural SDDDE [This thesis]
Constant	$\surd$	$\times$	$\surd$
Piece-wise constant	$\times$	$\surd$	$\surd$
Time-dependent	$\times$	$\times$	$\surd$
State-dependent	$\times$	$\times$	$\surd$

Similar to ODEs, the establishment of existence and uniqueness theorems for DDEs relies on the functions' continuity concerning t and Lipschitz continuity concerning x and its delayed counterparts $x(t-\tau)$. Demonstrating such a theorem becomes intricate due to the varied types of delays encountered (such as time-dependent, state-dependent, etc.) and their multiplicity, making it a less practical endeavor in our scenario. In the following theorems, we provide the existence and uniqueness theorem for the single (constant and state dependent) delay DDE.

Theorem 2.4.6 (Constant Delay : Existence and Uniqueness)**.** *Let $f : [0,T] \times \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ be continuous in t and Lipschitz continuous with respect to its other arguments. Let $\psi : \mathbb{R} \to \mathbb{R}^d$ be continuous and let $\tau \in \mathbb{R}^+$. Then, there exist a unique continuous solution $x : [0,T] \to \mathbb{R}^d$ such that :*

$$\begin{aligned} &\frac{dx(t)}{\mathrm{d}t} = f(t, x(t), x(t-\tau)) \\ &x(t \leq 0) = \psi(t). \end{aligned}$$

Remark 2.4.7. *The existence and uniqueness proof for such a single constant DDE is a direct application of the Cauchy–Lipschitz theorem 2.2.1 on subintervals of $[0,T]$. Indeed, in the interval $[0,\tau]$, the DDE reduces to an ODE and Cauchy–Lipschitz can be applied. By iteratively applying such a result on the intervals $[i\tau,(i+1)\tau], i = 0,1,\ldots$, we can establish the global existence and uniqueness of the DDE solution on $[0,T]$. This method is known in the literature as the method of steps introduced by Bellman in the 1960s [13].*

In the scenario of state-dependent delays, when the delays τ disappear at certain points t^*, proving existence and uniqueness becomes more challenging. Driver [42] proved the following theorem, where the version presented here is sourced and adapted from [12].

Theorem 2.4.8 (State Dependent Delay : Local Existence and Uniqueness)**.** *Let $U \subset \mathbb{R}^d$ and $V \subset \mathbb{R}^d$ be neighborhoods of $\psi(0)$ and $\psi(0-\tau(0,\psi(0)))$ respectively, and assume that $f : [0,T] \times \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ be continuous in t and Lipschitz continuous with respect to its other arguments in $[0,h] \times U \times V$ for some $h > 0$. Moreover, let the history function $\psi : \mathbb{R} \to \mathbb{R}^d$ be Lipschitz continuous in t and the delay function $\tau(t,x) \geq 0$ be continuous in t and Lipschitz continuous in x in $[0,T] \times U$. Then, for some $\delta > 0$, there exist a unique continuous solution $x : [0,\delta] \to \mathbb{R}^d$ that depends continuously on the initial data such that :*

$$\begin{aligned} &\frac{dx(t)}{\mathrm{d}t} = f(t, x(t), x(t-\tau(t,x(t)))) \\ &x(t \leq 0) = \psi(t). \end{aligned} \tag{2.14}$$

In practice, theorem 2.4.8 can be extended by using the same ideas mentioned in Remark 2.4.7. If the condition (H) below holds :

$$(H) : \inf_{(t,x)\in[0,T]\times\mathbb{R}^d} \tau(t,x) = \tau_0 > 0 \tag{2.15}$$

then Equation 2.14 in the interval $[0, \tau_0]$ reduces to an ODE.

Remark 2.4.9. *Driver also proved an analogous result for equations with multiple state dependent delays [42].*

2.5 . Neural DDEs and Backpropagation

In order to train a Neural DDE, we need to compute the gradient of the loss with respect to the parameters θ. As detailed in Section 2.3, we mentioned the two options : the *discretize-then-optimize* and *optimize-then-discretize* approaches. In the specific instance of *optimize-then-discretize* i.e. the adjoint method will vary based on the types of delays employed.

2.5.1 . Discretize-then-optimize

This corresponds to the "regular backpropagation" we are used to. Given the requirement, that the DDE solver is written with an auto-differentiable package like JAX or PyTorch, we backpropagate through the internal operations of the solver. Advantages and drawbacks of this method are the same as in the ODE case, which can be found in Kidger's thesis [80]. To simplify the discussion, we will focus on a single constant delay DDE. Although extending the concept to multiple delays or time- and state-dependent delays is more complex, it does not alter the fundamental idea.

2.5.2 . Optimize-then-discretize

This approach was initially introduced in the Neural DDE paper by Zhu et al. [200] for the case of a single constant delay, where the delay isn't in the model's parameters θ. This section and thesis aim to extend the adjoint methods for Neural DDEs to accommodate multiple delays as well as various types of delays, such as time-dependent or state-dependent delays. Let us first, tend our focus on the adjoint method associated to multiple constant delays. Without loss of generality, we will present the adjoint method for a single constant delay, and then discuss how it can be extended to multiple constant delays. Lastly, we will briefly discuss the adjoint method for general type delays.

Theorem 2.5.1 (DDE's adjoint method for constant delays)**.** *Let us consider the Neural DDE model below where τ can appear in the parameters vector θ. For*

convenience, we use the following notation $y(t) = x(t-\tau)$.

$$\begin{aligned} &\frac{dx(t)}{\mathrm{d}t} = f_\theta(t, x(t), x(t-\tau)), \quad \tau \in \mathbb{R}^+ \\ &x(t \leq 0) = \psi(t) \end{aligned} \tag{2.16}$$

and the following loss function :

$$L(x(T)) = \int_0^T l(x(t))\,\mathrm{d}t.$$

Then, the gradient's loss w.r.t. the parameters θ *is given by :*

$$\frac{\mathrm{d}L}{\mathrm{d}\theta} = -\int_0^T \lambda(t) \left(\frac{\partial f_\theta(t, x(t), x(t-\tau))}{\partial \theta} - \frac{\partial f_\theta(t, x(t), x(t-\tau))}{\partial y} x'(t-\tau) \right) \mathrm{d}t \tag{2.17}$$

where the adjoint dynamics $\lambda(t)$ *are given by another DDE :*

$$\begin{aligned} &\frac{\mathrm{d}\lambda(t)}{\mathrm{d}t} = \frac{\partial l(x(t))}{\partial x} - \lambda(t)\frac{\partial f_\theta(t, x(t), x(t-\tau))}{\partial x} - \lambda(t+\tau)\frac{\partial f_\theta(t+\tau, x(t+\tau), x(t))}{\partial y}, \\ &\lambda(t \geq T) = 0. \end{aligned} \tag{2.18}$$

We provide Theorem 2.5.1's proof in the Appendix A.3.

Remark 2.5.2. *Learning the delay* τ *and the associated vector field* f *seems to be ongoing popular topic, since some groups are deriving the same adjoint equation [167, 166]. However, these works only deal with DDEs with single constant delays and model that do not exceed* 10 *parameters. To our best of our knowledge, no work has taken into account the adjoint method for Neural DDEs with multiple constant delays along with an implementation that scales for "large" neural networks.*

In the case of multiple delays τ_i, where $i \in 1, \ldots, k$, the equations in Theorem 2.5.1 are extended as follows. For each i, we define $y_i(t) = x(t-\tau_i)$.

The second term $\lambda(t)\frac{\partial f_\theta(t, x(t), x(t-\tau))}{\partial x}$ in the adjoint dynamics (Eq. 2.18) is replaced by :

$$\lambda(t)\frac{\partial f_\theta(t, x(t), x(t-\tau_1), \ldots, x(t-\tau_k)))}{\partial x} \tag{2.19}$$

and the last term $\lambda(t+\tau)\frac{\partial f_\theta(t+\tau, x(t+\tau), x(t))}{\partial y}$ in the adjoint dynamics (Eq. 2.18) is replaced by the following :

$$\sum_{i=1}^{k} \lambda(t+\tau_i) \frac{\partial f_\theta(t+\tau_i, x(t+\tau_i), x(t-\tau_0+\tau_i), \ldots, x(t-\tau_n+\tau_i))}{\partial y_i} \tag{2.20}$$

For the gradient $\frac{\partial f_\theta(t,x(t),x(t-\tau))}{\partial \theta}$ in Equation 2.17 is replaced by the following :

$$\frac{\partial f_\theta(t, x(t), x(t-\tau_1), \ldots, x(t-\tau_k))}{\partial \theta} \tag{2.21}$$

and $\frac{\partial f_\theta(t,x(t),x(t-\tau))}{\partial y} x'(t-\tau)$ in Equation 2.17 is replaced by the following :

$$\sum_{i=1}^{k} \frac{\partial f_\theta(t, x(t), x(t-\tau_1), \ldots, x(t-\tau_n))}{\partial y_i} x'(t-\tau_i) \tag{2.22}$$

Remark 2.5.3. *If the delay τ isn't a learnable parameter, then the gradient's loss w.t.r to the parameters θ (i.e. Equation* (2.17)*) is simplified to the following equation (please see the proof of the theorem 2.5.1 and more Specifically Eq. A.29) :*

$$\frac{\mathrm{d}L}{\mathrm{d}\theta} = -\int_0^T \lambda(t) \frac{\partial f_\theta(t, x(t), x(t-\tau))}{\partial \theta} \, \mathrm{d}t.$$

Due to the thesis' time considerations and the complexity of the adjoint method for general type (i.e. time and state dependent) delays, this approach wasn't explored further, but we refer to the work of Zivari-Piran and Enright [205], providing the adjoint method.

Remark 2.5.4. *In a private communication with the authors from Zivari-Piran and Enright [205], they provided the adjoint method for general type delays.*

2.6 . Neural Integro Differential Equations

To the best of the authors' knowledge, the most recent "generic" Neural Differential Equation model introduced is the Neural Integro Differential Equation (IDE) in 2022 [196]. Neural IDE is defined as :

$$\frac{dx(t)}{\mathrm{d}t} = f_\theta(t, x(t)) + \int_{\alpha(t)}^{\beta(t)} K_\theta(t,s) F_\theta(x(s)) ds, \quad x(0) = x_0 \tag{2.23}$$

where θ represent the parameters, $f_\theta : \mathbb{R} \times \mathbb{R}^{d_1 \times \cdots \times d_k} \to \mathbb{R}^{d_1 \times \cdots \times d_k}$, $K_\theta : \mathbb{R} \times \mathbb{R} \to \mathcal{M}_{d_1 \times \cdots \times d_k}(\mathbb{R})$ and $F_\theta : \mathbb{R}^{d_1 \times \cdots \times d_k} \to \mathbb{R}^{d_1 \times \cdots \times d_k}$ are arbitrary neural networks, α, β be scalar-valued functions, and $x : [0,T] \to \mathbb{R}^{d_1 \times \cdots \times d_k}$ is the equation's solution.

Remark 2.6.1. *The Neural IDE can model specific IDEs like Volterra (with $\alpha(t) = 0$ and $\beta(t) = t$) [20, 180] and Fredholm (with $\alpha(t) = a$ and $\beta(t) = b$) [180], depending on the integration bound functions used.*

Remark 2.6.2. *The code can be found on this link here* [3].

Similarly, the existence and uniqueness problem of IDEs resembles that of ODEs and DDEs. Given reasonable regularity property of the integrand, it can be showed that Equation 2.23 admits a unique solution. We refer to Chapter 1 of Lakshmikantham [95].

Neural IDE is a non Markovian model by construction due to its integral form.

Remark 2.6.3. *Authors from Neural IDE have also worked on improvements of such a model [197, 195]. However, these cited papers have yet to be accepted in peer-reviewed journals/conferences.*

2.7 . Neural IDE and Backpropagation

2.7.1 . Discretize-then-optimize

Once again, this corresponds to the "regular" backpropagating. Provided that the IDE solver is implemented using an auto-differentiable package such as JAX or PyTorch, we can perform backpropagation through the solver's internal operations. By combining well established Neural ODE libraries with the `torchquad` package—a tool for multidimensional numerical integration optimized for GPUs, regardless of the backend [54]—we can compute an IDE's right-hand side using auto-differentiable operations. While IDEs differ in certain respects, the benefits and limitations of this method are analogous to those in the ODE context, as discussed in earlier sections.

Remark 2.7.1. *One way of seeing IDEs is to simple treat them as (expensive) ODEs where the right hand side is function of t and $x(t)$. In the PDE scientific community, this is also referred to the numerical method of the lines, which is a technique for solving partial differential equations by discretizing in all but one dimension and then integrating the semi-discrete problem as a system of ODEs or differential-algebraic equations (DAEs). Our IDE formulation becomes if $\beta(t) \leq t$:*

$$\frac{dx(t)}{\mathrm{d}t} = F_\theta(t, x(t)), \quad x(0) = x_0. \tag{2.24}$$

if we set :

$$F_\theta(t, x(t)) = f_\theta(t, x(t)) + \int_{\alpha(t)}^{\beta(t)} K_\theta(t, s) F_\theta(x(s)) ds. \tag{2.25}$$

This is one way of dealing with IDEs numerically, although this thesis isn't focused on numerically integrating such equations.

3. https://github.com/emazap7/NIDE/tree/1885c8bef46c3a19609c0847cbc97ff7e99c42c3

2.7.2 . Optimize-then-discretize

This adjoint method was first introduced in the Neural IDE [196] and we recall and adapt its theorem.

Theorem 2.7.2 (IDE's adjoint method)**.** *Let us consider the Neural IDE model below :*

$$\frac{dx(t)}{\mathrm{d}t} = f_\theta(t, x(t)) + \int_{\alpha(t)}^{\beta(t)} K_\theta(t,s) F_\theta(x(s)) ds, \quad x(0) = x_0 \tag{2.26}$$

and the following loss function :

$$L(x(T)) = \int_0^T l(x(t))\,\mathrm{d}t$$

The gradient's loss w.r.t. the parameters θ is given by :

$$\begin{aligned}\frac{\mathrm{d}L}{\mathrm{d}\theta} = &+ \int_0^T -\lambda(t) \frac{\partial f_\theta(t, x(t))}{\partial \theta} \mathrm{d}t \\ &+ \int_0^T \int_{\alpha(t)}^{\beta(t)} -\lambda(t) \left[\frac{\partial K_\theta(t,s)}{\partial \theta} F_\theta(x(s)) + + K_\theta(t,s) \frac{\partial F_\theta(x(s))}{\partial \theta} \right] ds \mathrm{d}t,\end{aligned} \tag{2.27}$$

where the adjoint dynamics $\lambda(t)$ are given by the following equation :

$$\begin{aligned}&\dot{\lambda}(t) = \frac{\partial l(x(t))}{\partial x} - \lambda(t) \left(\frac{\partial f_\theta(t, x(t))}{\partial x} + K_\theta(t,t) \frac{\partial F_\theta(x(t))}{\partial x} \right) \\ &\lambda(T) = 0.\end{aligned} \tag{2.28}$$

We provide Theorem 2.7.2's proof in the Appendix A.4.

Remark 2.7.3. *In the original Neural IDE paper [196], the authors presented mainly the adjoint method for an IDE with a parameterless function f and briefly discussed the parameterized case. We provided explicitly here the full adjoint method to handle the case where f is parameterized by θ.*

Remark 2.7.4. *If the IDE's integral term is null then its adjoint method boils back down to the ODE's one.*

2.8 . Other Deep Learning Models loosely related to Neural DEs

In this section, we present other deep learning models that are related to Neural DEs to some extent. These models are not necessarily part of the Neural DEs family per se but share some similarities with them.

2.8.1 . Physics Informed Neural Networks

Physics Informed Neural Networks (PINNS) [147, 145, 146] represent the solution u of a Partial Differential Equation (PDE) as a neural network $u_\theta(x)$ where θ are the model's parameters. Similar to prior work [108, 67], we consider the following system of PDEs :

$$\begin{aligned} \mathcal{D}[u][x] &= 0, \quad x \in \Omega \\ \mathcal{B}[u][x] &= 0, \quad x \in \partial\Omega \end{aligned} \tag{2.29}$$

where $\mathcal{D}$ is a differential operator defining the PDE, $\mathcal{B}$ is an operator associated with the boundary and/or initial conditions, and $\Omega \subseteq \mathbb{R}^d$. To solve Equation 2.29 the neural network is usually trained by minimizing the following loss $\mathcal{L}$:

$$\mathcal{L} = \frac{1}{2n_{\text{res}}} \sum_{i=1}^{n_{\text{res}}} \left(\mathcal{D}\left[u_\theta\right]\left[x_r^i\right]\right)^2 + \frac{1}{2n_{\text{bc}}} \sum_{i=1}^{n_{\text{bc}}} \left(\mathcal{B}\left[u_\theta\right]\left[x_b^j\right]\right)^2$$

Here $\left\{x_r^i\right\}_{i=1}^{n_{\text{res}}}$ are the residual points and $\left\{x_b^j\right\}_{j=1}^{n_{\text{bc}}}$ are the boundary/initial points. The first loss term measures how much $u_\theta(x)$ fails to satisfy the PDE, while the second term measures how much $u_\theta(x)$ fails to satisfy the boundary/initial conditions.

Remark 2.8.1. *PINNs are distinct from Neural Differential Equation models. The latter group, which includes Neural ODEs, DDEs, and IDEs, utilizes neural networks to define differential equations. On the other hand, as shown in Equation 2.29, PINNs employ neural networks to find solutions to predefined differential equations.*

Remark 2.8.2. *PINNs main drawback comes from their computational cost which can sometimes be comparable to the cost of directly solving the problem in a standard PDE solver. These types of model are prone to overfitting.*

2.8.2 . Neural Operators

Neural Operators (NO) propose to learn directly the mapping between function spaces on bounded domains by using a finite collection of observations of input-output pairs from this mapping [88]. Let $\mathcal{A}$ and $\mathcal{U}$ be Banach spaces of functions defined on bounded domains $D \subset \mathbb{R}^d, D \subset \mathbb{R}^{d'}$ respectively and $\mathcal{G}^\dagger : \mathcal{A} \to \mathcal{U}$ be a non-linear map. Let $\{a^{(i)}, u^{(i)}\}_{i=1}^N$ be the PDE's observations. Neural Operators aims to construct a map

$$\mathcal{G}_\theta : \mathcal{A} \to \mathcal{U}, \theta \in \mathbb{R}^p \tag{2.30}$$

such that for particular $\theta^* \in \mathbb{R}^p, \mathcal{G}^\dagger \approx \mathcal{G}_{\theta^*}$. This is done by solving the empirical-risk minimization problem :

$$\min_\theta \frac{1}{N} \sum_{i=1}^{N} \left\| u^{(i)} - \mathcal{G}_\theta(a^{(i)}) \right\|_{\mathcal{U}}^2 \tag{2.31}$$

The first NO of its kind is the *DeepONet* model [107], followed by the major contribution of the Fourier Neural Operator [100] from where now many types of NOs derive from. For illustration purpose, let's introduce the renowned Fourier Neural Operator (FNO) model.

The FNO is formulated as an iterative architecture $v_0 \mapsto v_1 \mapsto \cdots \mapsto v_T$ where v_j for $j = 0, 1, \ldots, T-1$ is a sequence of functions each taking values in $\mathbb{R}^{d_v}$. The first transformation v_0 consists of projecting the input a in a higher dimensional space, $v_0(x) = P(a(x))$ where P is a fully connected network. Then several transformations $v_{t-1} \mapsto v_t$ are applied (defined below). The output $u(x) = Q(v_T(x))$ is obtained by projection v_T with a parameterised Q.

Definition 2.8.3 (Iterative updates)**.** Define the update to representation $v_t \mapsto v_{t+1}$ as follows :

$$v_{t+1}(x) := \sigma\left(W v_t(x) + \left(\mathcal{K}(a;\phi)v_t\right)(x)\right), \quad \forall x \in D \tag{2.32}$$

where $\mathcal{K} : \mathcal{A} \times \Theta_{\mathcal{K}} \to \mathcal{L}\left(\mathcal{U}\left(D;\mathbb{R}^{d_v}\right), \mathcal{U}\left(D;\mathbb{R}^{d_v}\right)\right)$ maps to bounded linear operators on $\mathcal{U}\left(D;\mathbb{R}^{d_v}\right)$ and is parameterized by $\phi \in \Theta_{\mathcal{K}}, W : \mathbb{R}^{d_v} \to \mathbb{R}^{d_v}$ is a linear transformation, and $\sigma : \mathbb{R} \to \mathbb{R}$ is a non-linear activation function whose action is defined component-wise.

Definition 2.8.4 (Kernel integral operator $\mathcal{K}$)**.** Let the kernel integral operator mapping in 2.32 be defined as follows :

$$\left(\mathcal{K}(a;\phi)v_t\right)(x) := \int_D \kappa(x, y, a(x), a(y);\phi)v_t(y)\mathrm{d}y, \quad \forall x \in D \tag{2.33}$$

where $\kappa_\phi : \mathbb{R}^{2(d+d_a)} \to \mathbb{R}^{d_v \times d_v}$ is a neural network parameterized by $\phi \in \Theta_{\mathcal{K}}$.

The kernel integral operator $\mathcal{K}$ is to be a convolution operator, this allows to simplify Equation 2.33. Then, $\mathcal{K}$ is parameterized in the Fourier Space. Writing such operator in the Fourier space we get :

$$\left(\mathcal{K}(a;\phi)v_t\right)(x) = \mathcal{F}^{-1}\left(\mathcal{F}\left(\kappa_\phi\right) \cdot \mathcal{F}\left(v_t\right)\right)(x), \quad \forall x \in D \tag{2.34}$$

By parametrizing $\mathcal{F}\left(\kappa_\phi\right)$ with a linear layer R_ϕ, we obtain the introduced Fourier Neural Operator.

Remark 2.8.5. *Neural Operators still face challenges in long-range integration, achieved via rollout in the literature. However, efforts have been made to mitigate this issue. The PDE Refiner [103] treats the Neural Operator as a diffusion process, aiming to denoise its current prediction for improved results. Another limitation is that although Neural Operators are discretization-invariant over the spatial domain of a PDE, this does not hold true for time. Typically, Neural Operators model the mapping $u(t) \mapsto u(t+\Delta t)$, to evaluate the state u in between time t and $t+\Delta t$ is to either by training a new NO with a new Δt or to interpolate, which can lead to inaccuracies.*

2.8.3 . Neural CDE

Kidger et al. [81] introduced Neural CDE, another variant of Neural DEs, which can be viewed as the continuous equivalent of RNNs. Given its continuous time nature, Neural CDE can process irregular sampled time series and its hidden state possesses a continuous dependence on the observed data (due to its Riemann–Stieltjes integral).

Given irregularly sampled time series $\mathbf{x} = ((t_0, x_0), (t_1, x_1), \ldots, (t_n, x_n))$, with each $t_i \in \mathbb{R}$ the timestamp of the observation $x_i \in \mathbb{R}^v$, and $t_0 < \cdots < t_n$.

Let $X : [t_0, t_n] \to \mathbb{R}^{v+1}$ be the natural cubic spline with knots at $t_0, \ldots, t_n$ such that $X_{t_i} = (x_i, t_i)$. Let $f_\theta : \mathbb{R}^w \to \mathbb{R}^{w \times (v+1)}$ be any neural network model depending on parameters θ. The value w is a hyperparameter describing the size of the hidden state. Let $\zeta_\theta : \mathbb{R}^{v+1} \to \mathbb{R}^w$ be any neural network model depending on parameters θ. Then we define the *neural controlled differential equation* model as the solution of the CDE :

$$z_t = z_{t_0} + \int_{t_0}^{t} f_\theta(z_s) \mathrm{d}X_s \quad \text{for } t \in (t_0, t_n], \tag{2.35}$$

where $z_{t_0} = \zeta_\theta(x_0, t_0)$.

Remark 2.8.6. *In the context of modeling dynamical systems, Neural CDEs are not particularly suitable. Consider a dataset $((t_0, x_0), (t_1, x_1), \ldots, (t_n, x_n))$, where each $t_i \in \mathbb{R}$ represents the timestamp for the observation $x_i \in \mathbb{R}^v$. Our objective is to forecast $x_1, x_2, \ldots, x_n$ based on the initial value x_0. The fact that Neural CDEs rely on the Riemann-Stieltjes integral, which requires the data (t_i, x_i) we are interested in predicting, makes their use difficult. With the Riemann-Stieltjes integral, the model would take as input the data (i.e., $x_1, x_2, \ldots, x_n$) to predict.*

Remark 2.8.7. *In a private communication, authors' of Neural CDEs recommend against using natural cubic splines for neural CDEs, as they are noncausal and as such allow information to be leak backward-in-time / cannot be used in online settings at inference time. In later work they found causal splines that can be used instead.*

2.9 . Tips and tricks for Neural DEs

In this section, we offer practical guidance and strategies for effectively training Neural DEs in regression tasks, specifically focusing on fitting trajectories of dynamical systems derived from PDEs. We will discuss the importance of the training data, the choice of the solver, the learning rate, the regularization, and the delay clipping. This section is intended to communicate some of

the best practices and lessons learned from our experience in training Neural DEs, particularly Neural DDEs.

Training data In a data-driven context, you usually have at your disposal a dataset of observations generated by an unknown system, which may consist of either several trajectory expressed as $\mathcal{D}_1 = \{(t_0^l, x_0^l), \ldots, (t_N^l, x_N^l)\}_{l=1}^L$, or one very long time series $\mathcal{D}_2 = \{(t_0, x_0), \ldots, (t_M, x_M)\}$, where $M \gg N$ and $M \approx LN$. Several options are possible to process the datasets $\mathcal{D}_1$ and $\mathcal{D}_2$.

Dataset $\mathcal{D}_1$ Traditionally, the model is trained using whole trajectories. Alternatively, the trajectories can be divided into smaller segments of length m and these segments are used for training the model. During the testing phase, even though the model was trained on small segments, it predicts the longer testing trajectories. This method balances integration computation time with an increased number of batches needed to process the entire dataset. If the results from the test trajectories are not satisfactory, which is often the case, one can incrementally increase m until it reaches the maximum length of the dataset. This approach is also known as curriculum learning [15, 59] (the model is progressively fed longer trajectories where the trajectory length can be associated to the task's difficulty). This is a standard trick to attain good performance more quickly, or to converge to a better local optimum if the global optimum is not found.

Dataset $\mathcal{D}_2$ By dividing dataset $\mathcal{D}_2$ into L segments, you end up in the same dataset formulation as $\mathcal{D}_1$.

Choice of the solver Choosing the right solver is essential for training Neural DEs. Generally, high-order solvers such as Dormand–Prince 5(4) [41] are recommended for the training. However, if lower-order solvers achieve satisfactory results, they can be employed to save computational resources and time. An alternative strategy is to use low-order solvers for initial training and then switch to high-order solvers for fine-tuning. Although this hasn't been seen in practice much.

Other relevant information In the context of curriculum learning, the learning rate should be gradually adjusted as longer trajectories are introduced to the model. This approach helps prevent sudden changes in the loss function and stabilizes the training process. For Neural DDEs, applying any form of regularization to the delays is not advisable. The delays should be allowed to adjust freely during training because, for instance, L1 regularization might push the delays towards zero, which is undesirable. Lastly, clipping the

delays after each optimizer step can be necessary to avoid numerical instabilities and maintain the constraints that delays must be positive and not exceed the trajectory length. Typically, delays exceeding the trajectory length are not encountered because initialized delays (often sampled from uniform distribution $\mathcal{U}(0.1, 1)$) would require an excessive number of optimizer steps to reach such values. Using separate optimizers and learning rates for delays and the parameterized vector field showed some promise. However, this approach is not yet fully explored and requires further investigation.

3 - Modeling Dynamical Systems

In this chapter, we will adopt the following setup : let us consider a deterministic continuous-time dynamical system defined by $F : \mathbb{R}^d \to \mathbb{R}^d$, where d indicates the dimension of the state space. We are limited to a single observable function $g : \mathbb{R}^d \to \mathbb{R}^m$, with $m < d$, and our main concern is to efficiently predict the dynamics of g[1]. The following sections will introduce the Mori–Zwanzig (MZ) formalism, Takens' theorem, the Koopman theory and its link with MZ, which are all powerful distinct methods for tackling this issue of partially observed dynamical systems.

3.1 . Introduction

Learning dynamical systems from data is essential in fields such as science, control theory, robotics, and machine learning. Often, real-world systems are only partially observed, meaning that observers have access to only a portion of the system's state, which makes the Markovian assumption invalid, i.e., the next state does not depend on the previous one only. To handle this, observers need to infer the unobserved portions from the available data to predict future behaviors. This process frequently involves introducing latent variables that function as memory proxies to enhance prediction accuracy. These latent variables can be obtained through data-driven methods, analytically, or through a combination of both. While discrete systems are often preferred for their simplicity and ease of implementation, our focus in this thesis is exclusively on systems within a continuous time framework.

Partially observed systems are ubiquitous, particularly within complex systems. Climate systems, epidemiology, and financial markets are instances where only a portion of the information is accessible. Addressing these challenges often involves dealing with partially observed systems, highlighting the importance of formalizing this task both theoretically and practically.

1. Throughout this thesis, we will assume that the measured system is observable. As a crucial concept in dynamical system theory, observability is closely related to controllability and is essential for successful dynamics fitting. Observability isn't the focus of this thesis and will not be discussed further.

3.2 . Mori Zwanzig formalism

3.2.1 . Derivation of the MZ equation

The Mori–Zwanzig formalism, rooted in statistical mechanics, provides a method to construct accurate evolution equations for relevant quantities (often called observables), such as macroscopic observables, within high-dimensional dynamical systems [124, 207, 209]. This framework is instrumental in situations where the full state $x(t)$ is unavailable, and one can only access lower dimensional observations. Consequently, the MZ formalism is relevant for addressing dimension reduction problems [203]. No matter which scale we are interested in, the evolution equation of low-dimensional observables can be formally derived as an operator equation, which is now known as the Mori–Zwanzig equation. The formalism goal is to find an appropriate projector $\mathcal{P}$ and $\mathcal{Q} = \mathcal{I} - \mathcal{P}$ that splits the dynamics of the original high-dimensional system into resolved variables, unresolved variables and the interaction between these two. Below we provide the derivation of the MZ equation.

If we consider a nonlinear system evolving on a smooth manifold $\mathcal{S}$ (for simplicity we set $\mathcal{S} \subset \mathbb{R}^d$) :

$$\frac{dx(t)}{\mathrm{d}t} = F(x), \quad x(0) = x_0 \tag{3.1}$$

The system can be seen through the lens of an arbitrary number of scalar-valued observables $\forall i, g_i : \mathcal{S} \to \mathbb{C}$ (or $\mathbb{R}$). The dynamics of any scalar-valued observable $g_i(x)$ (quantity of interest) can be expressed with the Koopman operator $\mathcal{K}(t,s)$ [86] :

$$g_i(x(t)) = [\mathcal{K}(t,s)g_i](x(s)) \tag{3.2}$$

$$\mathcal{K}(t,s) = e^{(t-s)\mathcal{L}}, \quad \mathcal{L}g_i(x) = F(x) \cdot \nabla g_i(x) \tag{3.3}$$

with $\mathcal{L}$ the Liouville operator. Often rather than not, instead of computing the dynamics of all observables, it is better to compute the evolution of a subset of quantities of interest. This subspace can be modelled with a bounded linear operator $\mathcal{P}$ (projector) and its orthogonal projector $\mathcal{Q} = \mathcal{I} - \mathcal{P}$. Since we are only considering a subset of observable we seek to get the dynamics of $\mathcal{P}\mathcal{K}(t,s)$. With the definition of the Koopman operator and the Dyson identity [185] :

$$e^{t\mathcal{L}} = e^{t\mathcal{Q}\mathcal{L}} + \int_0^t e^{s\mathcal{L}}\mathcal{P}\mathcal{L}e^{(t-s)\mathcal{Q}\mathcal{L}}ds \tag{3.4}$$

we can write the evolution of $e^{t\mathcal{L}}$ as :

$$\frac{d}{dt}e^{t\mathcal{L}} = e^{t\mathcal{Q}\mathcal{L}}\mathcal{Q}\mathcal{L} + \frac{\mathrm{d}}{\mathrm{d}t}\left(\int_0^t e^{s\mathcal{L}}\mathcal{P}\mathcal{L}e^{(t-s)\mathcal{Q}\mathcal{L}}ds\right). \tag{3.5}$$

and thus obtain the Mori–Zwanzig operator equation

$$\frac{d}{dt}e^{t\mathcal{L}} = e^{t\mathcal{L}}\mathcal{P}\mathcal{L} + e^{t\mathcal{Q}\mathcal{L}}\mathcal{Q}\mathcal{L} + \int_0^t e^{s\mathcal{L}}\mathcal{P}\mathcal{L}e^{(t-s)\mathcal{Q}\mathcal{L}}\mathcal{Q}\mathcal{L}ds \tag{3.6}$$

Remark 3.2.1. *In order to differentiate the integral term of the Dyson identity from Equation 3.5, we use the differentiation under the integral sign theorem in its general form (or 'Leibniz Integral Rule').*

The three terms at the right-hand side are respectively the streaming (or Markovian) term, the fluctuation (or noise) term and the memory term.

One can rewrite also the MZ's functional form equation [179] by applying equation 3.6 to all observables g_i at their initial condition $g_i(t = 0) = g_{i0}$, concatenating all observable g_i together into $g = [g_1, \ldots, g_m]$:

$$\frac{dg(t)}{\mathrm{d}t} = M(g(t)) + F(t) - \int_0^t K(g(t-s), s)ds \tag{3.7}$$

where

$$M(g(t)) = e^{t\mathcal{L}}\mathcal{P}\mathcal{L}g_0 \tag{3.8}$$

$$K(g(t-s), s) = -e^{(t-s)\mathcal{L}}\mathcal{P}\mathcal{L}e^{s\mathcal{Q}\mathcal{L}}\mathcal{Q}\mathcal{L}g_0 \tag{3.9}$$

$$F(t) = e^{t\mathcal{Q}\mathcal{L}}\mathcal{Q}\mathcal{L}g_0 \tag{3.10}$$

Equation 3.7, derived within the framework of the Mori–Zwanzig theory, is sometimes referred to as the Generalized Langevin Equation (GLE) [208]. It provides a rigorous governing equation for the observable g. This elegant formulation of partially observed dynamics yields an exact evolution equation that takes the form of an Integro-Differential Equation.

Beginning with a general dynamical system Equation 3.1 and considering that we only have access to an observable g, the MZ formalism offers a method to derive an equation for the evolution of g characterized by the Markovian M, memory K, and fluctuation F components. The Markovian term represents the immediate contribution of the observable at time t, the memory term accounts for the interaction between the observable and the unresolved variables, and the fluctuation term describes the orthogonal dynamics which is often modeled with noise. MZ provides a way to formulate g's dynamics with a memory term that captures the system's history, which is crucial for accurate predictions. MZ research's primary focus is on the approximation and quantification of the memory term. Common data-driven approaches used in this endeavor include the NARMAX technique [106] and the rational function approximation proposed by Lei et al. [97].

3.2.2 . MZ's projection operators

The MZ Equation 3.7 is rather general, by specifying the projection operator $\mathcal{P}$ one can obtain different forms of the MZ equation. Authors Zhu [203] derive many forms of the MZ equation by choosing different projection operators. Here we only present the most common and easy to use projection operator.

A choice for $\mathcal{P}$ is the orthogonal projection onto the span of linearly independent set of observables $\{g_1, \ldots, g_m\}$. Such a finite rank projection operator is also called the Mori projector [125], widely used in statistical physics. $\mathcal{P}$ relies on the inner product defined as :

$$\langle k, l \rangle = \int k(x) l(x) d\mu(x), \quad k, l \in L^2(\mu) \tag{3.11}$$

where k, l are L^2 integrable functions and x is drawn from the measure μ. The measure μ is conventionally set as a natural measure that is specific to the dynamics. For example, it is natural to adopt the canonical equilibrium distribution (Gibbs' measure) for equilibrium Hamiltonian systems [125]. For non-equilibrium systems, we can adopt the stationary measure [73]. Thus, the Mori projector is defined as, given the observables $g = [g_1, \ldots, g_m]$:

$$[\mathcal{P}k](g) = \sum_{i,j=1}^{m} \langle k, g_i \rangle [C_0^{-1}]_{i,j} g_j \tag{3.12}$$

where C_0^{-1} is the inverse of C_0 whose (i, j) entry is $\langle g_i, g_j \rangle$.

Hence, the MZ equation 3.7 reduces to :

$$\frac{dg(t)}{\mathrm{d}t} = M(g(t)) + F(t) - \int_0^t K(t-s) g(s) ds \tag{3.13}$$

where

$$M_{ij} = \sum_{k=1}^{m} [C_0^{-1}]_{jk} \langle g_k(0), \mathcal{L} g_i(0) \rangle \tag{3.14}$$

$$K_{ij}(t) = \sum_{k=1}^{m} [C_0^{-1}]_{jk} \langle g_k(0), \mathcal{L} e^{t\mathcal{Q}\mathcal{L}} \mathcal{Q}\mathcal{L} g_i(0) \rangle \tag{3.15}$$

$$F(t) = e^{t\mathcal{Q}\mathcal{L}} \mathcal{Q}\mathcal{L} g_0 \tag{3.16}$$

$$\tag{3.17}$$

Remark 3.2.2. *Compared to the general MZ equation 3.7, the MZ equation with the Mori projector 3.13 has a simpler form. Notably, all terms (M, K, F) have analytical expressions. Moreover, the memory integrand is now the product of the kernel and the observable.*

3.2.3 . A simple example of MZ in practice

This example illustrates the MZ formalism in practice. We consider an harmonic oscillator :

$$\frac{\mathrm{d}^2x(t)}{\mathrm{d}t^2} + \frac{1}{\tau}\frac{\mathrm{d}x(t)}{\mathrm{d}t} + \omega_0^2 x(t) = 0 \tag{3.18}$$

where $1/\tau$ is the damping coefficient and ω_0 the natural frequency. Given the state $X(t) := [x(t), \frac{\mathrm{d}x(t)}{\mathrm{d}t}]^T$, the dynamics of the system can be written as :

$$\frac{dX(t)}{\mathrm{d}t} = \begin{bmatrix} 0 & 1 \\ -\omega_0^2 & -\frac{1}{\tau} \end{bmatrix} \begin{bmatrix} x(t) \\ \frac{\mathrm{d}x(t)}{\mathrm{d}t} \end{bmatrix} \tag{3.19}$$

We choose our observable to be $g(X) = \frac{\mathrm{d}x(t)}{\mathrm{d}t}$. Since Equation 3.19 is written in the same form as Equation 3.1, and our observable is one dimensional (this implies that we have a Mori projector because of the linearly independent set $\{\dot{x}\}$), we can apply the MZ equation 3.13 which yields :

$$\frac{d\frac{\mathrm{d}x(t)}{\mathrm{d}t}}{\mathrm{d}t} = -\frac{1}{\tau}\frac{\mathrm{d}x(t)}{\mathrm{d}t} - \omega_0^2 \int_0^t \frac{\mathrm{d}x(s)}{\mathrm{d}s} ds - \omega_0^2 x(0) \tag{3.20}$$

Remark 3.2.3. *Alternatively we can get Equation 3.20 by manipulating Equation 3.19 :*

$$\begin{aligned} &\frac{d\frac{\mathrm{d}x(t)}{\mathrm{d}t}}{\mathrm{d}t} = -\omega_0^2 x(t) - \frac{1}{\tau} dv x(t) t \\ &c + x(t) = \int_0^t \frac{\mathrm{d}x(s)}{\mathrm{d}s} ds \implies c = -x(0) \end{aligned} \tag{3.21}$$

By using elementary calculus and definitions,

$$\begin{aligned} &c + x(t) = \int_0^t \frac{\mathrm{d}x(s)}{\mathrm{d}s} ds \implies c = -x(0) \\ &x(0) + x(t) = \int_0^t \frac{\mathrm{d}x(s)}{\mathrm{d}s} ds \end{aligned} \tag{3.22}$$

We ultimately end up with the same Equation 3.20.

Remark 3.2.4. *We provide the exhaustive derivation of Equation 3.20. Given any linear system of dimension d in the matrix form alongside with the operator MZ Equation 3.6 :*

$$\frac{dX(t)}{\mathrm{d}t} = A(t)X(t), \quad A(t) := A$$

$$\frac{d}{dt}e^{t\mathcal{L}} = e^{t\mathcal{L}}\mathcal{P}\mathcal{L} + e^{t\mathcal{Q}\mathcal{L}}\mathcal{Q}\mathcal{L} + \int_0^t e^{s\mathcal{L}}\mathcal{P}\mathcal{L}e^{(t-s)\mathcal{Q}\mathcal{L}}\mathcal{Q}\mathcal{L}ds \tag{3.23}$$

Let us suppose that only m variables are observed. Hence, the matrix A can be split into blocks as follows. We have respectively $n_r = d - m$ and $n_u = m$ resolved

and unresolved variables and each operator corresponds to a certain matrix of the system. The projector $\mathcal{P} = P$ projects onto the subspace where the resolved variable live in and its orthogonal counterpart $\mathcal{Q} = Q$ on the unresolved ones.

$$A = \begin{bmatrix} A_{rr} & A_{ru} \\ A_{ur} & A_{uu} \end{bmatrix}, \quad X = \begin{bmatrix} X_r \\ X_u \end{bmatrix} \tag{3.24}$$

$$PX = X_r, \quad QX = X_u \tag{3.25}$$

The MZ equation 3.23 applied to the resolved variables $X_r(t)$ gives

$$\frac{dX_r(t)}{\mathrm{d}t} = A_{rr}X_r(t) + A_{ru}\int_0^t e^{(t-s)A_{uu}}A_{ur}X_r(s)ds + A_{ru}e^{tA_{uu}}X_u(0) \tag{3.26}$$

The integral $\int_0^t e^{(t-s)A_{uu}}A_{ur}X_r(s)ds$ is then the memory term, A_{rr} the Markovian one and the fluctuation was canceled out by hypothesis.

In this specific instance, we can observe the contribution of each term in the MZ equation : $-\frac{1}{\tau}\frac{\mathrm{d}x(t)}{\mathrm{d}t}$ represents the streaming term, $-\omega_0^2 x(0)$ denotes the fluctuation term, and $-\omega_0^2\int_0^t \frac{\mathrm{d}x(s)}{\mathrm{d}s}ds$ corresponds to the memory term. Regardless of the system's initial conditions, the fluctuation term maintains a constant contribution, while the memory and Markovian term have the greatest impact on the dynamics. We present several examples of the decomposition of the observable $\dot{x}$ into Markovian, memory, and fluctuation terms using the MZ equation with varying values of ω_0 and τ in Figures 3.1, 3.2, and 3.3. Depending on the value of $\omega_0^2\tau$, the memory term can have varying magnitude. When $\omega_0^2\tau \ll 1$, the Markovian term predominates the dynamics, as demonstrated in Figure 3.2. Conversely, in the opposite extreme, the memory term significantly influences the dynamics, as shown in Figure 3.3. Lastly, Figure 3.1 illustrates the balanced influence of the Markovian, memory, and fluctuation terms when $\omega_0^2\tau = 1$.

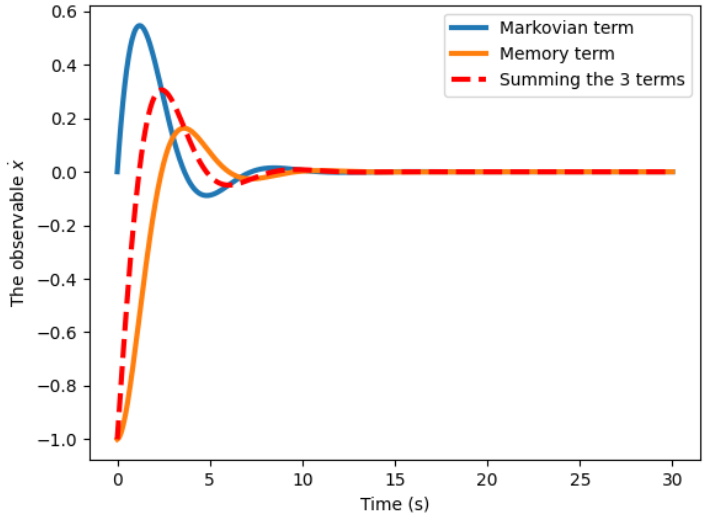

Figure 3.1 – Decomposition of the observable $\frac{\mathrm{d}x(t)}{\mathrm{d}t}$ into Markovian, Memory and Fluctuation terms with the MZ equation for $\omega_0 = 1, \tau = 1.0$

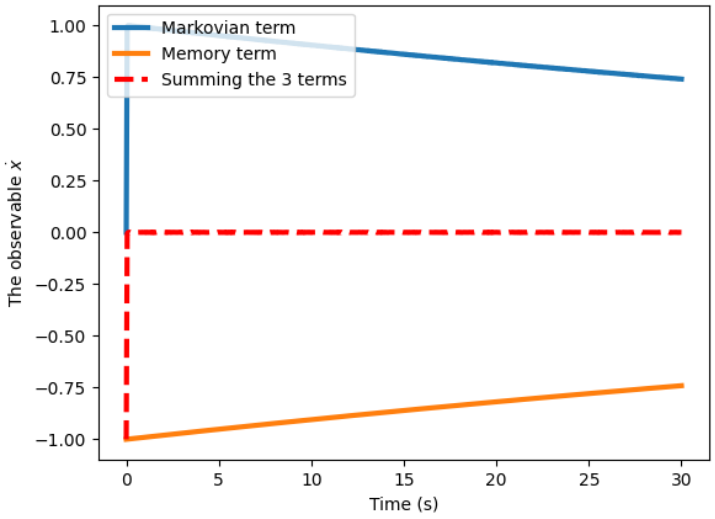

Figure 3.2 – Decomposition of the observable $\frac{\mathrm{d}x(t)}{\mathrm{d}t}$ into Markovian, Memory and Fluctuation terms with the MZ equation for $\omega_0 = 1, \tau = 0.01$

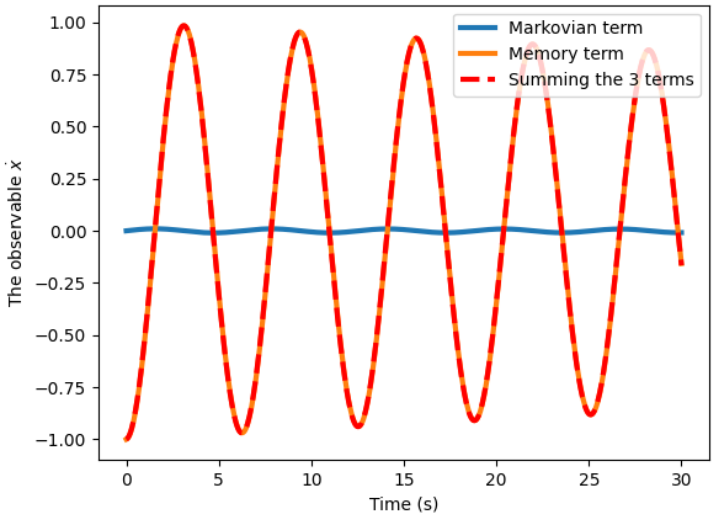

Figure 3.3 – Decomposition of the observable $\frac{\mathrm{d}x(t)}{\mathrm{d}t}$ into Markovian, Memory and Fluctuation terms with the MZ equation for $\omega_0 = 1, \tau = 100$

Remark 3.2.5. *With linear systems we can obtain analytical forms for the 3 terms of the MZ equation, which is not the case most of the time for more general systems (i.e. non-linear) and is an active research areas to get numerical approximations of each term [55, 137].*

3.3 . Takens' Theorem

3.3.1 . Introduction

Takens' work provided a groundbreaking method for analyzing dynamical systems from a geometrical perspective. He showed that, under specific and mild conditions, it is possible to reconstruct the dynamics of systems, even when observations are limited to some variables, using a time series of delayed observations. These conditions require the system to evolve on a smooth manifold M, which serves as an invariant set of the system, meaning $G(M) \subset M$ where G represents the dynamical system's vector field. This manifold M is often called an *attractor* in dissipative systems. An additional condition is the absence of periodic orbits (or cycle) with a period equal to the chosen delay.

In summary, employing a delay-coordinate map (defined in Definition 3.3.3) allows for the construction of a diffeomorphic shadow manifold M' from univariate observations of the original system. This method makes it possible to reconstruct a "shadow" dynamical system that is topologically equivalent to

the original. Being topologically equivalent means that both the original and shadow dynamical systems possess the same dynamical properties, Lyapunov exponents, number and nature of its fixed points, and the dimension of the manifold. Thus, we formally introduce Takens' theorem along with the necessary definitions as described by sources such as Noakes [130], Takens [173].

Definition 3.3.1 (Attractor)**.** Let (X, d) be a compact metric space and $f : X \to X$ a continuous map. An attractor is a nonempty, closed set $A \subset X$ such that $f(A) = A$ and, for every $\epsilon > 0$, there is a $\delta > 0$ such that for every point x such that $d(x, A) < \delta$ will stay within distance ϵ and verify $d(f^n(x), A) < \epsilon$ when $n \to \infty$.

Definition 3.3.2 (Immersion & Embedding)**.** An immersion is a differentiable function between two differentiable manifolds whose derivative is injective at every point in the manifold. An embedding is an injective immersion.

Definition 3.3.3. If Φ is a flow on a manifold M, τ is a positive number (called the *delay*), and $g : M \to \mathbb{R}$ (usually called an observable) is a smooth function, define the delay-coordinate map $F(g, \Phi, \tau) : M \to \mathbb{R}^n$ by :

$$F(g, \Phi, \tau)(x) = (g(x), g(\Phi_{-\tau}(x)), g(\Phi_{-2\tau}(x)), \ldots, g(\Phi_{-(n-1)\tau}(x))) \tag{3.27}$$

Here, Φ_τ stands for the operator that advances the dynamical system by a time step τ, i.e. that sends $x(t)$ to $x(t+\tau)$, and g is the observable operator, that sends a full state $x(t)$ to actual observables $g(x(t)) =: g(t)$.

Theorem 3.3.4 (Takens' embedding theorem)**.** *Let M be compact of dimension m. There is an open dense subset $\mathcal{D}$ of $\mathrm{Diff}(M) \times C^k(M, \mathbb{R})$ with the property that delay coordinate map $F(g, \Phi, \tau) : M \to \mathbb{R}^{2m+1}$ is an embedding of C^k manifolds, when $(\Phi, g) \in \mathcal{D}$.*

Remark 3.3.5. *In practice, this delay coordinate map is the simple expression $F(g, \Phi, \tau)(x) = (g(t), g(t-\tau), g(t-2\tau), \ldots, g(t-(n-1)\tau))$.*

Several generalizations of such theorem were established [134, 47], with the latest being from Deyle and Sugihara [39] that extended the reach of Takens' theorem by providing proofs for a generalized theorem for non-linear state space reconstruction (SSR). Theorems from Deyle and Sugihara [39] provide proof of principle for modeling attempts of nonlinear dynamics involving multiple time series, and provides the rather non-restrictive assumptions required in such applications for building models from multiple time series variables. Additionally, they propose multiple embeddings as a potentially effective method for extracting information from time series data with restricted length, especially when there are numerous simultaneous observations of dynamics occurring on the same attractor manifold [159].

Remark 3.3.6. *The delay τ in the delay-coordinate map must not be equal to the period of any periodic orbit because it will result in :*

$$\Phi(x) = \Phi_{-\tau}(x) \tag{3.28}$$

This implies that the delay coordinate map will not be able to distinguish between the two states $x(t)$ and $x(t-\tau)$, and thus the reconstruction of the attractor will ultimately fail. This is clearly seen with the Takens map from theorem 3.3.3 :

$$\begin{aligned} F(g,\Phi,\tau)(x) &= (g(x), g(\Phi_{-\tau}(x)), g(\Phi_{-2\tau}(x)), \dots, g(\Phi_{-(n-1)\tau}(x))) \\ &= (g(x), \dots, g(x)) \end{aligned} \tag{3.29}$$

In the next subsection, we will discuss the delay-coordinate map and how many coordinates are needed to reconstruct the attractor, along with the choice of the delay τ.

3.3.2 . Embedding dimension

By definition, the embedding dimension refers to the number of components in the delay-coordinate map $F(g,\Phi,\tau)$. According to Takens' theorem $3.3.4$, the number of delayed terms necessary for time delay embedding $F(g,\Phi,\tau)$ should be larger than twice the dimension of the manifold M. Although this theorem provides an upper bound for the embedding dimension, in some instances, the embedding dimension can be reduced. This typically occurs when the dynamics operate within a set rather than a high-dimensional manifold. Consequently, the upper bound specified by Takens' theorem is influenced by the geometry of the set, rather than the manifold itself. Determining the dimension of a set is out of scope for this thesis, but we refer to the *box counting dimension* for further information [113]. The embedding dimension is linked to many concepts in the physics fields like the intrinsic dimension, that says that the number of degrees of freedom of a system is not necessarily as large as the number of variables used to describe it [26]. A classic example is the case of the laminar cylinder flow which requires a high number of degrees of freedom to be simulated within a finite element solver, but can be described using only three, well-chosen, observables [178].

Another example is the Lorenz system, which is a simplified model of atmospheric processes based on fluid dynamics. This model demonstrates how systems can exist in lower-dimensional spaces. This model is a three-dimensional dynamical system that exhibits chaotic behavior under specific conditions and ultimately settles within a defined spatial area that looks like butterfly wings, creating an attractor. This attractor is known as a strange attractor due to its fractal (non-integer) dimension, which is approximately 2.07. The Lorenz system is defined by the following ODEs [105] :

$$\begin{aligned}\frac{dx}{\mathrm{d}t} &= \sigma(y-x)\\ \frac{dy}{\mathrm{d}t} &= x(\rho-z)-y\\ \frac{dz}{\mathrm{d}t} &= xy-\beta z\end{aligned} \tag{3.30}$$

with ρ, σ, β are constants. It has been shown that an embedding dimension of 3 is enough for the delay-coordinate maps of x and y coordinates to give embeddings. This isn't the case for z coordinate [83, 1] because of Lorenz's fixed point inside the lobe of each of the butterfly-shaped attractor. When using a delay-coordinate map for the z coordinate, the map will not be able to distinguish between each lobe of the attractor, making the reconstruction of the attractor impossible. This is seen in Figure 3.4 the x and y still possess the topological information of the system (for example the two lobes of the attractor), while the z coordinate does not.

This example demonstrates that the embedding dimension suggested by Takens' theorem is not always useful in either theory or practice. Utilizing a greater number of delayed terms in the delay-coordinate map can be advantageous. Authors of [159] indicated that selecting a larger embedding dimension may diminish the impact of noise in the data, a concept associated with the *Filtered Delay Embedding Prevalence Theorem* [193]. In this thesis and our results, we have indeed observed this numerically.

Remark 3.3.7. *We show quantitatively here that the Lorenz system can be embedded in a three-dimensional delay coordinate map. We will focus on the x coordinate, although the same concept applies to y. Starting with the Taylor expansion for x :*

$$\begin{aligned}\frac{x(t+\tau)-x(t-\tau)}{2\tau} &= \dot{x}(t)+\frac{\tau^2}{6}\dddot{x}+\mathcal{O}(\tau^3)\\ \frac{x(t+\tau)-2x(t)+x(t-\tau)}{\tau^2} &= \ddot{x}(t)+\frac{\tau^2}{12}x^{(4)}(t)+\mathcal{O}(\tau^5)\end{aligned}$$

Incorporating the Taylor expansion into the Lorenz system gives us y and z in terms of x :

$$\begin{cases}\dot{x}=\sigma(y-x)\\ y=x+\frac{\dot{x}}{\sigma}\end{cases} \qquad \begin{cases}\dot{y}=x(\rho-z)-y\\ z=\rho-\frac{\dot{y}}{x}+y\\ z=\rho-\frac{\dot{x}+\frac{\ddot{x}}{\sigma}}{x}+x+\frac{\dot{x}}{\sigma}\end{cases} \tag{3.31}$$

By combining the results, we obtain :

$$y = x + \frac{\dot{x}}{\sigma}$$
$$z = \rho - \frac{\dot{x} + \frac{\ddot{x}}{\sigma}}{x} + x + \frac{\dot{x}}{\sigma}$$

By substituting the derivatives of x with their Taylor expansions, it becomes clear that y and z can be expressed using a three-dimensional delay coordinate map of x.

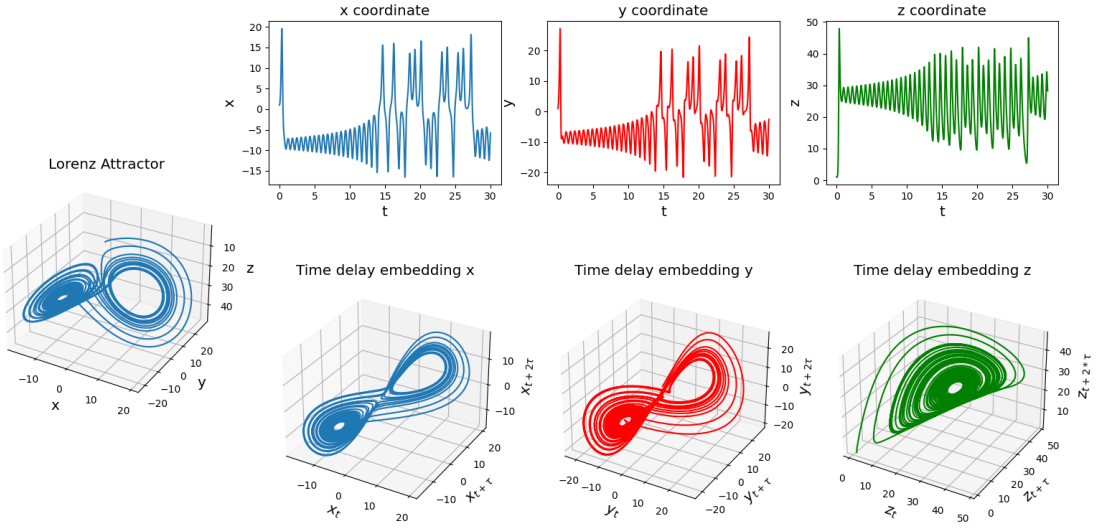

Figure 3.4 – The Lorenz attractor and its three individual and delay-coordinate map for the x, y, z coordinates with $\tau = 0.06$ and $(\sigma, \rho, \beta) = (10.0, 28.0, 8/3)$.

3.3.3 . The delay τ in the delay-coordinate map

Delay-coordinate maps can be classified into two primary categories : *uniform time delay embedding* and *non-uniform time delay embedding*. We will first address the uniform embedding before moving on to the non-uniform ones. The concept of time delay embedding was first introduced using evenly spaced measurements of a scalar time series $x(t)$, expressed as :

$$(x(t), x(t-\tau), x(t-2\tau), \ldots, x(t-(n-1)\tau)) \tag{3.32}$$

where n denotes the embedding dimension and τ refers to the delay [135]. When n is sufficiently large, the delay τ can be selected without restriction, as

long as it does not match the period of any periodic orbit, according to Takens' theorem 3.3.4. However, this is not the case in practice; the choice of both the delay and the embedding dimension is critical for accurately reconstructing the dynamics. This indicates that not all delay-coordinate maps are equivalent or of the same quality. Attempts have been made to develop relevant metrics for evaluating embedding quality, particularly in studies such as [28, 144], but no conclusive method has been established. Information theory and topological concepts have primarily been considered [48, 183, 128]. For instance, a commonly used heuristic is to set the delay to one-quarter of the system's period. Additionally, techniques like measuring the mutual information (or auto-correlation) between the time series and its delayed version have been used to find the optimal delay [48]. Regardless of the chosen method for setting delays, several general principles are widely accepted. If the delay is too short, the components of the delay-coordinate map will be similar, causing all points to cluster around the bisector of the embedding space [28]. Conversely, if the delay is too long, the different coordinates may become nearly uncorrelated. For a more thorough discussion, we refer to the survey by [175]. Another class of uniform embedding, known as *derivatives embedding*, utilizes the derivatives of the time series :

$$(x(t), \dot{x}(t), \ddot{x}(t), \ldots, x^{(n-1)}(t)) \tag{3.33}$$

This approach is not commonly employed due to the need for calculating high-order derivatives, which can introduce noise and be computationally intensive. We also briefly mention the *Integral-Differential embedding*, which faces similar practical challenges as the derivatives embedding [77].

The appeal of uniform delay embedding lies in its simplicity, requiring only two parameters to be selected, τ and n. In contrast, non-uniform delay embedding introduces additional complexity and can overcome the limitations associated with uniform embeddings [183]. Indeed, when a system exhibits different time scales, such as slow and fast dynamics, a uniform time delay embedding can only capture one of these scales. By incorporating multiple delays $[\tau_1, \tau_2, \ldots, \tau_n]$, the delay-coordinate map takes the form of :

$$(x(t), x(t-\tau_1), x(t-\tau_2), \ldots, x(t-\tau_n)) \tag{3.34}$$

Several methods have been proposed to determine the optimal delays, including continuity statistics [141], PECUZAL [90], maximizing derivatives on projections (MDOP) [128], and Monte Carlo decision tree search (MTCDS) [89]. Recently, methods from persistent homology have been used to determine optimal non-uniform embedding [175]. Going into the details of the methods mentioned just before isn't the focus of this thesis, but we refer to the cited papers for more information.

3.4 . The Koopman Operator

The Koopman operator theory, introduced in 1931 [87], is a promising and well-established approach for modeling dynamical systems. In this section, we will present an overview of the theory and recommend more comprehensive sources such as [22, 191, 101] for deeper insights.

3.4.1 . Introduction

Consider a nonlinear system evolving on a smooth manifold $\mathcal{S} \subset \mathbb{R}^n$:

$$\frac{dx(t)}{\mathrm{d}t} = F(x), \quad x(0) = x_0 \tag{3.35}$$

where x represents the system's state, F is the vector field, and x_0 the initial condition. We assume the availability of a scalar-valued observable function g. The dynamics of $g(x)$ can be expressed through the Koopman operator $\mathcal{K}(t, s)$ (Equation 3.2.1) :

$$\begin{aligned} &g(x(t)) = [\mathcal{K}(t,0)g](x_0) \\ &\mathcal{K}(t,0) = e^{t\mathcal{L}}, \quad \mathcal{L}g(x) = F(x) \cdot \nabla g(x) \end{aligned} \tag{3.36}$$

where $\mathcal{L}$ is the Liouville operator associated to the dynamics. In the rest of this section, we will as a notation abuse write $\mathcal{K}(t)$ instead of $\mathcal{K}(t, 0)$ for simplicity. The Koopman operator $\mathcal{K}$ advances the observable function g in time, thus making it a linear operator, enabling the application of linear algebra. Given that dynamics vector field F is a C^1-manifold, the Koopman eigenfunction satisfies the eigenvalue equation :

$$\mathcal{L}\phi_\lambda = F \cdot \nabla \phi_\lambda = \lambda \phi_\lambda \tag{3.37}$$

A Koopman eigenfunction ϕ_λ associated to the eigenvalue λ satisfies :

$$\mathcal{K}(t)\phi_\lambda = e^{\lambda t}\phi_\lambda \tag{3.38}$$

Equation 3.38 & 3.37 shows that the Koopman operator $\mathcal{K}$ and the Liouville operator $\mathcal{L}$ share the same eigenfunction ϕ_λ, given an eigenfunction ϕ_λ of $\mathcal{L}$ with eigenvalue λ is an eigenfunction of $\mathcal{K}(t)$ with eigenvalue $e^{\lambda t}$. Thus, this property shows that Koopman eigenfunctions are directly related to the dynamics of the system, i.e. knowing them is equivalent to knowing the trajectories of the system. The Koopman eigenfunctions and eigenvalues possess many properties but the one we wish to highlight is the following :

Theorem 3.4.1. *Suppose that ϕ_{λ_1} and ϕ_{λ_2} are two Koopman eigenfunctions associated with the eigenvalues λ_1 and λ_2 respectively. If $\phi_{\lambda_1}^{k_1}, \phi_{\lambda_2}^{k_2} \in \mathcal{S}$, with $k_1, k_2 \in \mathbb{R}$ then it is an eigenfunction associated with the eigenvalue $k_1\lambda_1 + k_2\lambda_2$.*

Proof of theorem 3.4.1

$$\mathcal{K}(t)(\phi_{\lambda_1}^{k_1}\phi_{\lambda_2}^{k_2}) = (\mathcal{K}(t)\phi_{\lambda_1})^{k_1}(\mathcal{K}(t)\phi_{\lambda_2})^{k_2} = e^{(k_1\lambda_1+k_2\lambda_2)t}\phi_{\lambda_1}^{k_1}\phi_{\lambda_2}^{k_2} \tag{3.39}$$

Theorem 3.4.1 implies that there is an infinity of Koopman eigenfunctions (some or all of them could be linearly dependent) [112]. Therefore, using the Koopman's eigenfunctions basis on the observable function g, could lead to a much simpler representation of the dynamics but at the expense of being infinite dimensional. This means that using the Koopman operator in practice, a cut-off must be made on the number of eigenfunctions to consider. For this purpose, it is useful to use a set of observable functions g_i for $i = 1, \ldots, m$. If the Koopman operator has a pure point spectrum, we can write the concatenated observables $\mathbf{g} = [g_1, \ldots, g_m]$, in the Koopman eigenfunction basis as [76, 40] :

$$\mathbf{g}(x(t)) = \mathcal{K}(t)\mathbf{g}(x_0) = \sum_{j=0}^{\infty} \mathbf{c}_j \phi_{\lambda_j}(x_0) e^{\lambda_j t} \tag{3.40}$$

When the Koopman operator is unitary, the eigenfunctions ϕ_{λ_j} are orthogonal, and the coefficients $\mathbf{c}_j = \langle \phi_{\lambda_j}, \mathbf{g} \rangle$ constitute what is known as the Koopman mode decomposition [117]. Based on Equation 3.40, it is possible to apply a cut-off in the basis to approximate the system's dynamics within a finite-dimensional space. Several practical algorithms have been devised to estimate a finite dimensional approximation of the Koopman operator from data, including Dynamic Mode Decomposition (DMD) [161] and Extended Dynamic Mode Decomposition (EDMD) [191]. The DMD algorithm will be covered in the next subsection.

3.4.2 . Dynamic Mode Decomposition (DMD)

DMD, initially introduced by Schmid [161], is a foundational technique for dimensional reduction, inspired by the Koopman theory and is categorized under modal decomposition methods [10]. It decomposes high-dimensional data into three components : spatial modes, scalar amplitudes, and temporal signals. Given a sequential set of data vector $\{z_0, \ldots, z_m\}$ where each $z_k \in \mathbb{R}^n$, the DMD algorithm finds the linear dynamics (i.e. A) assumed to have generated the data :

$$z_{k+1} = Az_{k+1}. \tag{3.41}$$

The DMD modes and eigenvalues are intended to approximate the eigenvectors and eigenvalues of A. The algorithm proceeds as stated in Algorithm 2 :

Algorithm 2 Standard DMD Algorithm

1: Define $X = [z_0, \ldots, z_{m-1}]$ and $Y = [z_1, \ldots, z_m]$
2: Compute the SVD of X : $X = U\Sigma V^*$
3: Define $\tilde{A} = U^* Y V \Sigma^{-1}$
4: Compute $\tilde{A}$'s eigenvalues and eigenvectors : $\tilde{A}w = \lambda w$
5: The DMD modes corresponding to the DMD eigenvalue λ are given by : $\hat{\phi} = Uw$

Since its inception, various adaptations of the original DMD have been developed to address and enhance its limitations; we mention only a select few here but direct readers to a more extensive review for further details [162]. One such variant, Extended DMD (EDMD) [190], incorporates a more complex set of observables that include nonlinear functions of the measurements, facilitating a more effective capture of dynamics. However, selecting the most appropriate and effective dictionary of observables for EDMD remains an unresolved issue. Advancements in machine learning have spurred initiatives to identify such dictionaries using neural networks [98]. Moreover, Multiresolution DMD, as detailed by Kutz et al. [94], addresses challenges posed by multiple timescales within the data, utilizing a technique where the core DMD algorithm is applied recursively to data that has been filtered across progressively lower frequencies.

3.4.3 . The Mori–Zwanzig framework within the Koopman theory

To the best of our knowledge, the authors of [101] were the first to formally connect the Generalized Langevin Equation (also known as the MZ equation) with the Koopman operator. As discussed in the section 3.4.1; consider a nonlinear system evolving on a smooth manifold $\mathcal{S} \subset \mathbb{R}^n$:

$$\frac{dx(t)}{\mathrm{d}t} = F(x), \quad x(0) = x_0 \tag{3.42}$$

where x represents the system's state, F is the vector field, and x_0 the initial condition. We assume the availability of several scalar-valued observable functions g_i for $i \in 1, \ldots, m$.

Koopman theory seeks to describe the evolution of a collection of linearly independent observables $\mathcal{M} = \{g_i\}_{i=1}^m$, which we will concatenate as $\mathbf{g} = [g_1, \ldots, g_m]$ for future reference. Assuming that the Koopman operator has a pure point spectrum (in the case of a continuous spectrum, the computations are more complex, though the overall concept remains the same [118]), the evolution of the observables $\mathbf{g}$ can be expressed, as shown in Equation 3.40

from Section 3.4.1, in the form of an operator :

$$\mathbf{g} = \sum_{j=0}^{\infty} \mathbf{c}_j \phi_{\lambda_j}$$

Koopman theory decomposes the observable $\mathbf{g}$ into a sum of Koopman eigenfunctions ϕ_{λ_j} and its coefficients $\mathbf{c}_j$. In contrast, the Mori–Zwanzig formalism uses the inner product in the Hilbert space $\mathcal{H}$ to decompose the space into the subspace linearly spanned by the set of observables $\mathcal{H}_{\mathbf{g}} = \mathrm{span}(\mathcal{M})$ and its orthogonal subspace $\mathcal{H}_{\bar{\mathbf{g}}} = \{\bar{g} \in \mathcal{S} : \langle \bar{g}, g_i \rangle = 0,\ g_i \in \mathcal{M}\}$. The Gram–Schmidt process can be used to construct the basis functions of the orthogonal space $\mathcal{H}_{\bar{\mathbf{g}}}$ with the Koopman eigenfunctions $\{\phi_{\lambda_i}\}_{i=1}^{\infty}$. We denote this infinite set of basis functions as $\bar{\mathcal{M}} = \{\bar{g}_i\}_{i=1}^{\infty}$. Similarly, with $\mathbf{g}$ represented in the Koopman basis, we can represent $\bar{\mathbf{g}}$ as $\sum_{j=0}^{\infty} \bar{\mathbf{c}}_j \phi_{\lambda_j}$, where the coefficients are given by $\bar{\mathbf{c}}_j = \langle \phi_{\lambda_j}, \bar{\mathbf{g}} \rangle$. Thanks to the Gram–Schmidt process, we also have $\langle g_i, \bar{g}_j \rangle = 0$ for $j \in \mathbb{N}$ and $i \in \{1, \ldots, m\}$. Since the basis functions are linearly independent, the combination of the two sets $\mathcal{M}$ and $\bar{\mathcal{M}}$ creates a complete basis for $\mathcal{H}$. Therefore, the Koopman eigenfunctions ϕ_{λ_j} can be expressed within this new basis :

$$\phi_{\lambda_i} = \sum_{j=1}^{m} \omega_{ij} g_j + \sum_{j=1}^{\infty} \bar{\omega}_{ij} \bar{g}_j \tag{3.43}$$

By applying the Koopman operator $\mathcal{K}$ that advances the observables in time, we get :

$$\phi_{\lambda_i}(t) = \sum_{j=1}^{m} \omega_{ij} g_j(t) + \sum_{j=1}^{\infty} \bar{\omega}_{ij} \bar{g}_j(t) \tag{3.44}$$

In order to link the Koopman operator with the Mori–Zwanzig equation **??**, we need to derive the evolution of the observables $\mathbf{g}$:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} g_i(t) &= \sum_{j=1}^{\infty} c_{ij} \phi_{\lambda_j}(x_0) \lambda_j e^{\lambda_j t} \\ &= \sum_{i=0}^{\infty} c_{ij} \lambda_j \phi_{\lambda_j}(t) \\ &= \sum_{l=1}^{m} \left(\sum_{j=1}^{\infty} c_{ij} \lambda_j \omega_{jl} \right) g_l(t) + \sum_{l=1}^{\infty} \left(\sum_{j=1}^{\infty} c_{ij} \lambda_j \bar{\omega}_{jl} \right) \bar{g}_l(t) \end{aligned} \tag{3.45}$$

and $\bar{\mathbf{g}}$:

$$\frac{\mathrm{d}}{\mathrm{d}t} \bar{g}_i(t) = \sum_{l=1}^{m} \left(\sum_{j=1}^{\infty} \bar{c}_{ij} \lambda_j \omega_{jl} \right) g_l(t) + \sum_{l=1}^{\infty} \left(\sum_{j=1}^{\infty} \bar{c}_{ij} \lambda_j \bar{\omega}_{jl} \right) \bar{g}_l(t) \tag{3.46}$$

Given the evolution of the observables $\mathbf{g}$ and $\bar{\mathbf{g}}$, we can express them in a more compact matrix form :

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\mathbf{g}(\mathbf{t})\\ \bar{\mathbf{g}}(\mathbf{t})\end{bmatrix} = \mathbf{L}\cdot\begin{bmatrix}\mathbf{g}(t)\\ \bar{\mathbf{g}}(t)\end{bmatrix} = \begin{bmatrix}\mathbf{L}_{gg} & \mathbf{L}_{g\bar{g}}\\ \mathbf{L}_{\bar{g}g} & \mathbf{L}_{\bar{g}\bar{g}}\end{bmatrix}\cdot\begin{bmatrix}\mathbf{g}\\ \bar{\mathbf{g}}\end{bmatrix} \tag{3.47}$$

where we employ a slight notation abuse $\mathbf{L}_{ij}, i, j \in \{\mathbf{g}, \bar{\mathbf{g}}\}$. Although Equation 3.45 & 3.46 are explicit, knowing the Koopman eigenfunctions ϕ_{λ_j} is not an easy feat, making this method difficult to apply in practice. In order, to get a closed form our dynamics $\mathbf{g}$, we need to integrate the orthogonal dynamics $\bar{\mathbf{g}}$. Given the orthogonal initial conditions $\bar{\mathbf{g}}(0)$, we can write the solution of the orthogonal dynamics as :

$$\bar{\mathbf{g}}(t) = \int_0^t e^{(t-s)\mathbf{L}_{\bar{g}\bar{g}}}\mathbf{L}_{\bar{g}g}\mathbf{g}(s)ds + e^{t\mathbf{L}_{\bar{g}\bar{g}}}\cdot\bar{\mathbf{g}}(0) \tag{3.48}$$

The implicit solution of $\bar{\mathbf{g}}$ allows us to express the evolution of the observables $\mathbf{g}$ as :

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{g}(t) = \mathbf{L}_{gg}\mathbf{g}(t) + \mathbf{L}_{g\bar{g}}\int_0^t e^{(t-s)\mathbf{L}_{\bar{g}\bar{g}}}\mathbf{L}_{\bar{g}g}\mathbf{g}(s)ds + \mathbf{L}_{g\bar{g}}e^{t\mathbf{L}_{\bar{g}\bar{g}}}\cdot\bar{\mathbf{g}}(0) \tag{3.49}$$

The derived Equation 3.49 is equal to Equation 3.26 in Section 3.2.3. Starting from Koopman theory, we have derived the Mori–Zwanzig equation that utilizes the Mori projection, introduced in Section 3.2.2. This link between the two theories is a significant step towards understanding the relationship between the two theories, and showcases that the MZ formalism supersets the Koopman theory.

4 - Time and State dependent Delay Differential Equations

Discontinuities and delayed terms are encountered in the governing equations of a large class of problems ranging from physics and engineering to medicine and economics. These systems cannot be properly modelled and simulated with standard ODE, or data-driven approximations such as Neural ODE. To circumvent this issue, latent variables are typically introduced to solve the dynamics of the system in a higher dimensional space and obtain the solution as a projection to the original space. However, this solution lacks physical interpretability. In contrast, DDEs, and their data-driven approximated counterparts, naturally appear as good candidates to characterize such systems. In this work we revisit the recently proposed Neural DDE by introducing Neural State-Dependent DDE (SDDDE), a general and flexible framework that can model multiple and state- and time-dependent delays, as discussed earlier in the Neural DE section 2.4.2. We show that our method is competitive and outperforms other continuous-class models on a wide variety of delayed dynamical systems. Code is available at https://github.com/thibmonsel/Time-and-State-Dependent-Neural-Delay-Differential-Equations.

4.1 . Introduction

In many applications, one assumes the time-dependent system under consideration satisfies a Markov property ; that is, future states of the system are entirely defined from the current state and are independent of the past. In this case, the system is satisfactorily described by an ordinary or a partial differential equation. However, the property of Markovianity is often only a first approximation to the true situation and a more realistic model would include past states of the system. Describing such systems has fueled the extensive development of the theory of DDEs [119, 126, 64]. This development has given rise to many practical applications : in the modelling of molecular kinetics [152] as well as for diffusion processes [45], in physics for modeling semiconductor lasers [187], in climate research for describing the El Ninõ current [52, 78], infectious diseases [35] and tsunami forecasting [192], to list only a few.

At the same time, the blooming of machine learning in recent years boosted the development of new algorithms aimed at modelling and predicting the behavior of dynamical systems governing phenomena commonly found in a wide variety of fields. Among these novel strategies, the introduction of NODEs [29] has contributed to further deepening the analysis of continuous

dynamical systems modelling based on neural networks. NODEs are a family of neural networks that can be seen as the continuous extension of Residual Networks [156], where the dynamics of a vector $x(t) \in \mathbb{R}^d$ at time t – hereafter often identified with the state of a physical system – is given by the parameterized network f_θ and the system's initial condition x_0 :

$$\frac{dx(t)}{\mathrm{d}t} = f_\theta(t, x(t)), \quad x(0) = x_0. \tag{4.1}$$

NODEs have been successfully applied to various tasks, such as normalizing flows [79, 56], handling irregularly sampled time data [153, 82], and image segmentation [142].

Starting from this groundbreaking work, numerous extensions of the NODE framework enabled to widen the range of applications. Among them, ANODEs [44] were able to alleviate NODEs' expressivity bottleneck by augmenting the dimension of the space allowing the model to learn more complex functions using simpler flows [44]. Let $a(t) \in \mathbb{R}^p$ denotes a point in the augmented space, the ODE problem is formulated as

$$\frac{d}{\mathrm{d}t}\begin{bmatrix} x(t) \\ a(t) \end{bmatrix} = f_\theta\left(t, \begin{bmatrix} x(t) \\ a(t) \end{bmatrix}\right), \quad \begin{bmatrix} x(0) \\ a(0) \end{bmatrix} = \begin{bmatrix} x_0 \\ 0 \end{bmatrix}. \tag{4.2}$$

By introducing this new variable $a(t)$, ANODE overcomes the inability of NODE to represent particular classes of systems. However, this comes with the cost of augmenting the data into a higher dimensional space, hence losing physical interpretability. Among the alternative techniques proposed for circumventing the limitations rising from the modelling of non-Markovian systems, the Neural Laplace model [72] proposes a unified framework that solves differential equations (DE) : it learns DE solutions in the Laplace domain. The Neural Laplace model cascades 3 steps : first, a network $_\gamma$ encodes the trajectory, then the so-called Laplace representation network g_β learns the dynamics in the Laplace domain to finally map it back to the temporal domain with an inverse Laplace transform (ILT). With the state y sampled T times at arbitrary time instants, $_\gamma$ gives a latent initial condition representation vector $p \in \mathbb{R}^K$:

$$p = h_\gamma((x(t_1), t_1), \ldots, (x(t_T), t_T)), \tag{4.3}$$

that is fed to the network g_β to get the Laplace transform

$$f(s) = v\left(g_\beta(p, u(s))\right), \tag{4.4}$$

with u a stereographic projector and v its inverse. Ultimately, an ILT step is applied to reconstruct state estimate $\widehat{y}$ from the learnt $f(s)$.

As an alternative to the aforementioned techniques, one may directly address the DDE problem by working within the framework of neural network-based DDEs. Despite the success of the NODEs philosophy, the extension to DDEs has barely been studied yet, possibly owing to the challenges of using general purpose DDE solvers. DDEs extend ODEs by incorporating additional terms into their vector fields, which are states delayed by a certain time τ. Recently, [200] introduced a neural network based DDE with one single constant delay :

$$\begin{aligned} &\frac{dx(t)}{\mathrm{d}t} = f_\theta(t, x(t), x(t-\tau)), \quad \tau \in \mathbb{R}^+ \\ &x(t \leq 0) = \psi(t), \end{aligned} \tag{4.5}$$

where $\psi(t)$ is the system's history function, τ a constant delay and f_θ a parameterized network. This work was next extended in the Neural Piece-Wise Constant Delays Differential Equations (NPCDDEs) model in [201]. Compared to NODE and its augmented counterpart, neural network-based DDEs do not require an augmentation to a higher dimensional space in order to be a universal approximator, thus preserving physical interpretability of the state vector and allowing the identification of the time delays. Nonetheless, the current variants of Neural DDE models only deals with a single constant delay or several piece-wise constant delays, thus lacking the generalization to arbitrary delays. Moreover, to the best of our knowledge, no machine learning library or open-sourced code exists to model not only these very specific types of DDEs but also any generic DDEs.

Contributions We propose the Neural State-Dependent DDE (SDDDE) model : an open-source, robust python DDE solver compatible with neural networks. Neural SDDDE is based on a general framework that pushes the envelope of Neural DDEs by handling DDEs with several delays in a more generic way. The implementation further encompasses general time- and state-dependent delay systems which extends the reach of [201].

In the remainder of this section, we briefly re-introduce the model SDDDE (mentioned in the theoretical section 2.4.2). Then, Implementation and methodologies are further discussed in Sec. 4.2. Experiments and comparisons with the state-of-the-art techniques are detailed in Sec. 4.3 & 4.4, using as benchmark numerous time-delayed models of incremental complexity. Our model is shown below to compare favorably with the current models on DDEs systems . Conclusions and outlook are finalized in Sec. 4.5.

4.2 . Methods

In the following, we discuss similarities, drawbacks and benefits of Neural SDDDE compared with the following models : NODE, ANODE and Neural Laplace. We recall that Neural SDDDE is a direct method for solving DDEs, specifically designed to handle delayed systems. This is not the case for ANODE where flexibility is obtained by introducing a higher dimensional space. Lastly, Neural Laplace can solve a broader class of differential equations, although with some limitations that we pinpoint in the following.

From the theoretical viewpoint, the Laplace transformation is often a tool used in proofs on DDEs [13] since it allows transforming linear functional equations in $f(x(t))$ involving derivative and differences into linear equations involving only $F(s)$. Thus, time-dependent and constant delay DDEs are transformed into linear equations of $F(s)$ using the Laplace transform. This transformation enables Neural Laplace to bypass the explicit definition of delays, whereas Neural SDDDE needs the delays to be specified unless the vector flow f and delays are learned jointly. These observations tie Neural Laplace to Neural SDDDE as they can be seen as similar models but living in different domains. However, a limitation of Laplace transformation-based approaches is that they are not defined for DDEs with state-dependent delays, thus restricting the class of time-delayed equations that one can solve with this technique.

Neural Laplace is a model that needs memory initialized latent variables, i.e., a long portion of the solution trajectory needs to be fed in order to get a reasonable representation of the latent variable. In contrast, by design, NODE and ANODE require information only at the initial time to predict the system dynamics. From this viewpoint, Neural SDDDE lies between these two frameworks as it requires a history function $\phi(t)$ to be provided for $t \in [-\tau_{\max}, 0]$, where $\tau_{\max}$ is the maximum delay encountered during integration. This is more demanding than solely relying on an initial condition $x(0)$ but far less than memory-based latent variables methods. Moreover, the training and testing schemes for Neural Laplace is constrained by this very same observation since future events can only be predicted after a certain observation time. This also makes the length of the trajectory to feed to Neural Laplace a hyperparameter to tune. This is not the case with NODE, ANODE and our approach.

4.3 . Experiments

4.3.1 . Description of the test cases

We evaluate and compare the Neural SDDDE on several dynamical systems (time, state-dependent and constant delays) listed below coming from biology and population dynamics. We show that Neural SDDDE outperforms

a variety of continuous-depth models and demonstrates its capabilities in simulating delayed systems. For all the systems listed in this section, data generation information is gathered in Appendix A.7.3.

Time-dependent delay system Here, we study a time-dependent delayed logistic equation [4] :

$$\frac{dx(t)}{\mathrm{d}t} = x(t)\big[1 - x(t - \tau(t))\big], \tag{4.6}$$

with $\tau(t) = 2 + \sin(t)$. We integrate in the time range $[0, 20]$ and define the constant history function $\psi(t) = x_0$, where x_0 is sampled uniformly from $[0.1, 2.0]$.

State-dependent time-delay system In this example, we consider the 1-D state-dependent Mackey Glass system from Dads et al. [36] with a state-dependent delay :

$$\begin{aligned} \frac{dx(t)}{\mathrm{d}t} &= -\alpha(t)x(t) + \beta(t)\frac{x(t-\tau(x))^2}{1 + x(t-\tau(x))^2} + \gamma(t) \\ \text{with} \quad \alpha(t) &= 4 + \sin(t) + \sin\Big(\sqrt{2t}\Big) + \frac{1}{1+t^2} \\ \beta(t) &= \gamma(t) = \sin(t) + \sin\Big(\sqrt{2t}\Big) + \frac{1}{1+t^2} \end{aligned} \tag{4.7}$$

where the delay function is $\tau(x) = \frac{1}{2}\cos(x(t))$. The model is defined on the time range $[0, 10]$ and the constant history function is $\psi(t) = x_0$ with x_0 sampled uniformly from $[0.1, 1]$.

Delayed Diffusion Equation Finally, we choose the delayed PDE taken from Arino et al. [5]. Such dynamics can for example model single species growth in a food-limited environment.

$$\frac{\partial u}{\partial t}(x,t) = D\frac{\partial^2 u}{\partial x^2}(x,t) + ru(x,t)\left(1 - u(x, t-\tau)\right), \tag{4.8}$$

where $D = 0.01, r = 0.9$ and $\tau = 2$. We integrate in the time range $[0, 4]$, the spatial domain is $\mathcal{D}_x = [0, 1]$ with periodic boundary conditions and define the history function $\psi(x,t) = a\sin(x)e^{-0.01t}$ where a is uniformly sampled from $[0.1, 4.0]$. The spatial domain is discretized with a uniform grid of resolution $\Delta x = 0.01$.

4.3.2 . Evaluation

We assess the performance of the models with their ability to predict future states of a given system. The metric used is the mean square error (MSE)

in all cases. Neural Laplace predicts only after a burn-in time since a part of the observed trajectory is used to learn a latent initial condition vector p. Since NODE, ANODE and Neural SDDDE can be seen as initial value problems (IVPs) we produce trajectories from initial conditions and compute the MSE with respect to the whole trajectory. On each DDE system, to judge the quality of each model, we elaborate two additional experiments alongside with the *test set predictions*. By modifying the history function $\psi(t)$ we change the behavior of the system and hence generate new trajectories. The first experiment puts each model in a *pure extrapolation regime*; the constant value of the history function $\psi(t) = x_0$ is sampled outside the range of the training and testing data (see Appendix A.7.4 for more details). This allows to see the models' extrapolation capabilities. The second assessment is a more hybrid approach where the *history function is a step function* :

$$\psi(t) = \begin{cases} x_0 & t \leq t_{\text{jump}} \\ x_1 & \text{otherwise} \end{cases}$$
$$t_{\text{jump}} \sim \mathcal{U}(-\tau_{\text{max}}, 0), \quad x_0, x_1 \sim \mathcal{U}(c_0, c_1)$$

where t_{jump} is the largest delay in the system and c_0, c_1 are system specific randomly sampled values (see Appendix A.7.4 for more details). Not only can the nature of the history step function change but can also have its domain function outside the training and test data (extrapolation regime).

As a reminder, to produce outputs, NODE and ANODE need an initial condition, Neural SDDDE the history function and Neural Laplace a portion of the trajectory. To ensure comparison in our experiments, we opt to give Neural Laplace the same information as Neural SDDDE, specifically the history function. This could be seen as a restriction of Neural Laplace since in its authors give 50% of the trajectory to produce outputs. Therefore, we propose another small experiment : on one dynamical system, Neural SDDDE is compared with Neural Laplace which is given more than just the history function.

4.4 . Results

Test errors for each dynamical system are reported in Table 4.1. Complementary information are included in the Appendix A.7.2 for what concerns the training process; model and training hyperparameters.

	Time Dependent DDE	State-Dependent DDE	Delay Diffusion
NODE	$.72 \pm .086$	$.0355 \pm .00064$	$.0029 \pm .0014$
ANODE	$.00962 \pm .00368$	$.00011 \pm .000071$	$.00087 \pm .00035$
Neural Laplace	$.00191 \pm .0006$	$.00049 \pm .00078$	$\mathbf{.00064 \pm .00016}$
Neural SDDDE	$\mathbf{.000989 \pm .00017}$	$\mathbf{.0000215 \pm .00001}$	$.00075 \pm .00019$

Table 4.1 – Test MSE averaged over 5 runs (random model initialization seed) of each experiment with their standard deviation. Best result bolded.

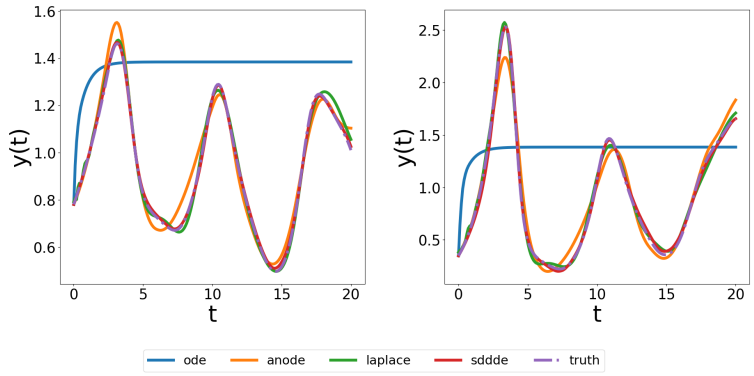

Figure 4.1 – Time Dependent DDE randomly sampled test trajectory plots

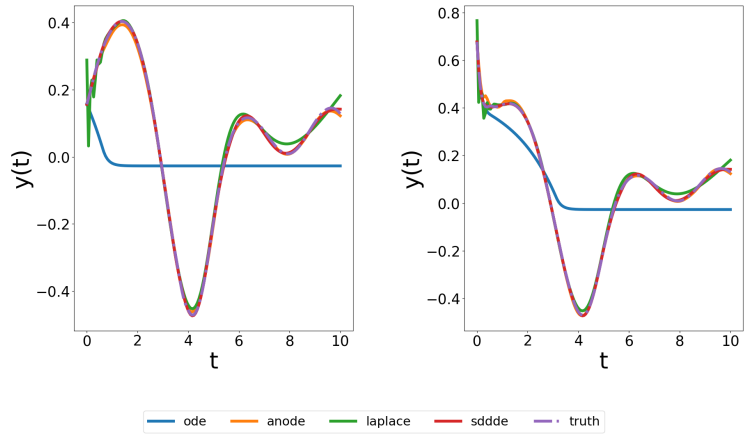

Figure 4.2 – State Dependent DDE randomly sampled test trajectory plots

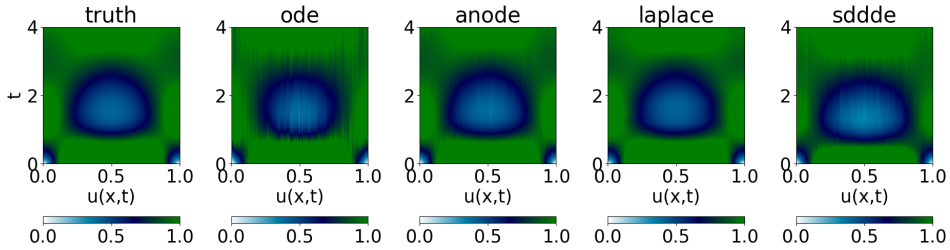

Figure 4.3 – Diffusion Delay PDE randomly sampled from the test set

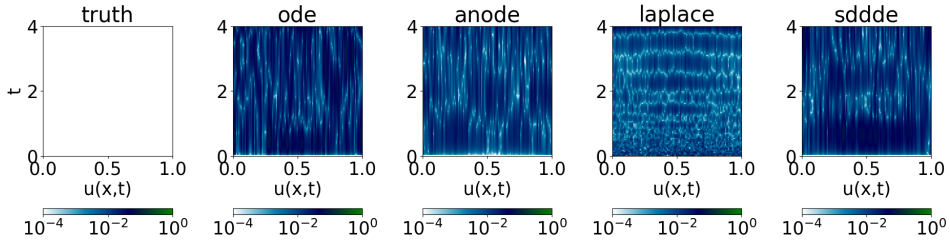

Figure 4.4 – Absolute error of Diffusion Delay PDE randomly sampled from the test set

Testset prediction Neural SDDDE almost consistently outperforms all other models across the DDE systems discussed in Section 4.3, as demonstrated in Figures 4.1, 4.2, 4.3, and 4.4. Neural Laplace appears to suffer from the Runge phenomenon (such a phenomenon is a problem of oscillation at the edges of an interval that occurs when using polynomial interpolation with polynomials of high degree over a set of equispaced interpolation points), particularly evident in the State-Dependent DDE (Figure 4.2). This issue likely stems from the ILT algorithm providing too few query points. As expected, Neural ODE is the most limited model, generally predicting only the mean trajectory of the dynamical systems being considered. ANODE yields satisfactory results, except for the Time Dependent DDE (Figure 4.1). For the Diffusion Delay PDE, all models predict the PDE's evolution with an absolute error reaching up to 10^{-2}, as shown in Figure 4.3. The absolute error, depicted in Figure 4.4, illustrates the discrepancies between the models. Neural Laplace produces more errors across the entire spatial domain for given time steps, while IVP models have errors localized in specific spatial regions.

Increasing trajectory fed for Neural Laplace Instead of providing the same history as for Neural SDDDE, Neural Laplace is now provided 50% of the trajectory to build its latent initial condition representation vector p. To that pur-

pose, the first half of the trajectory is used in Neural Laplace to predict the second half. We choose to train the model on the Time Dependent DDE along with the same training procedure (see Appendix A.7.2). Given the test MSE of Table 4.1, we choose to only compare the test MSE of Neural SDDDE and Neural Laplace in Table 4.2. By comparing Figure 4.5 and 4.1 one can clearly note that the Runge phenomenon is almost absent and predictions are almost as good as Neural SDDDE. This confirms that, in general, Neural Laplace needs more than the history function in order to correctly simulate DDEs.

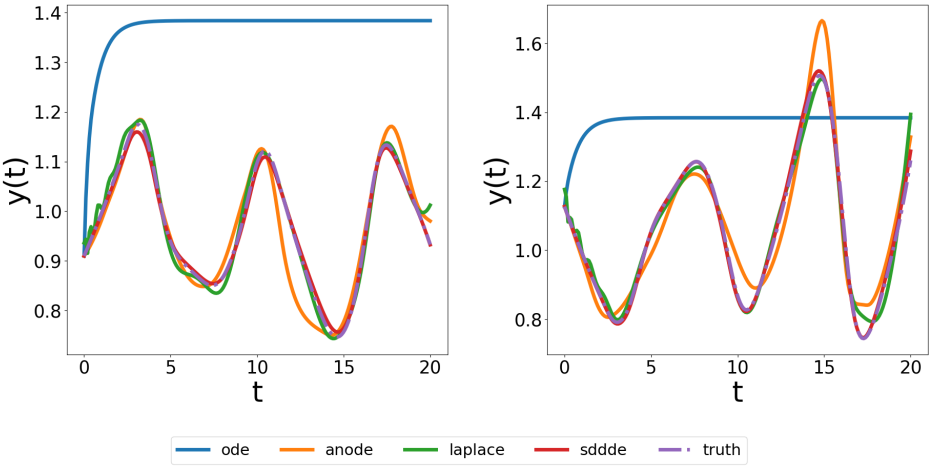

Figure 4.5 – Time-dependent DDE randomly sampled test set trajectories where 50% of data is fed to Neural Laplace

	Test MSE
Neural Laplace	$.00125 \pm .000798$
Neural SDDDE	$\mathbf{.000989 \pm .00017}$

Table 4.2 – Time Dependent test MSE averaged over 5 runs with their standard deviation. Best result bolded.

Extrapolation regime prediction This experiment really challenges model generalization capabilities. Overall, on certain datasets, some models can extrapolate with new constant history functions that are not too far out from the function domain of history functions used during training; more in details, trajectories were generated with $\psi(t) = x_0 \in [a, b]$: some models are able to exhibit adequate predictions for history functions that have a value near the bounds of $[a, b]$. For the Time Dependent system (Figure 4.6), Neural SDDDE yields better results compared to the other models. NODE produces the trajectory's mean field while ANODE captures the dynamics main trend

but with amplitude discrepancies. For the State-Dependent DDE (Figure 4.7), Neural SDDDE once again efficiently captures the dynamics while the other models fail. For the Diffusion Delay PDE displayed in Figure 4.8 & 4.9, overfitting is observed for Neural Laplace. Out of all the IVP models, Neural SDDDE predicts the best possible outcome compared to NODE and ANODE.

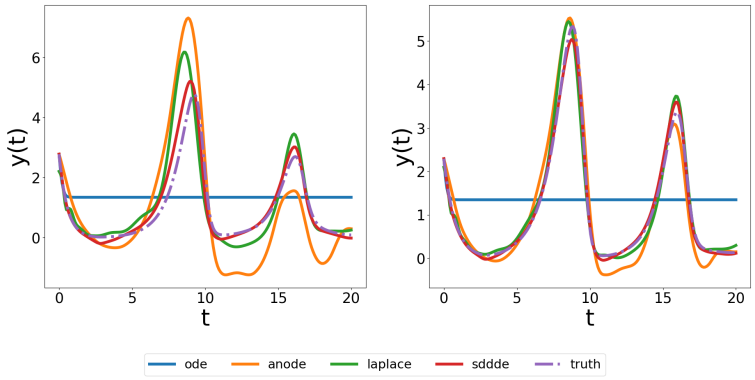

Figure 4.6 – Time Dependent DDE randomly sampled extrapolated trajectory plots

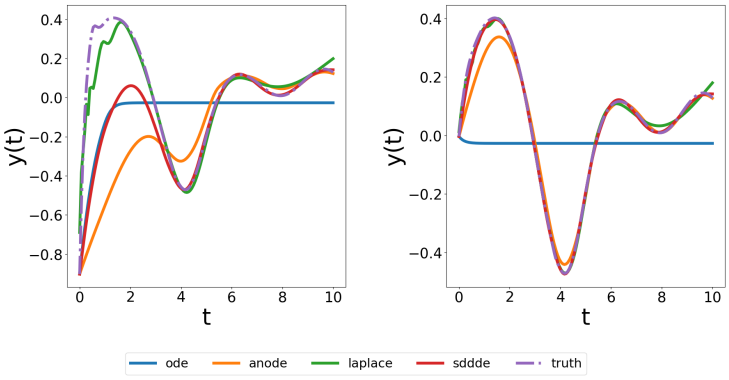

Figure 4.7 – State Dependent DDE randomly sampled extrapolated trajectory plots

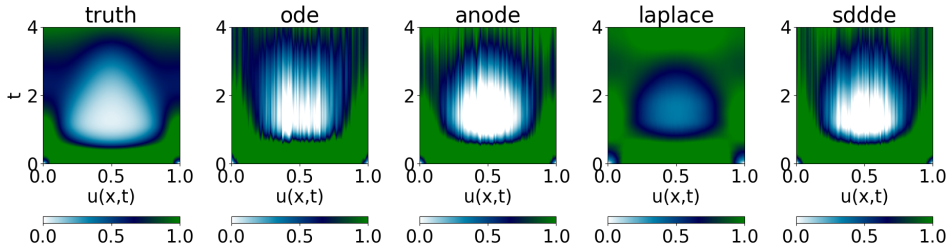

Figure 4.8 – Diffusion Delay PDE randomly sampled from the extrapolated test set

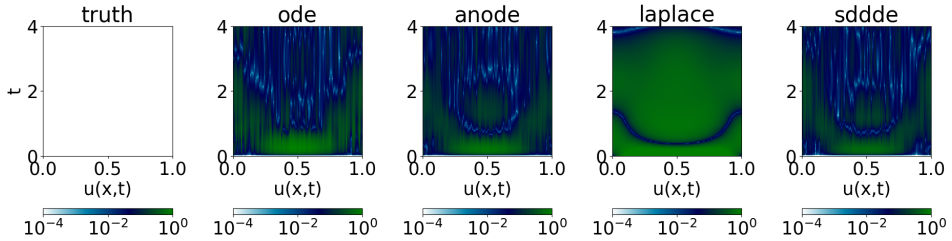

Figure 4.9 – Absolute error of Diffusion Delay PDE randomly sampled from the extrapolated test set

Step history function prediction This third experiment also demonstrates how the modification of the history function leads to changes in the transient regime and impacts later dynamics. Neural Laplace fails to generate adequate trajectories for the Time-dependent DDE system (Figure 4.10) and the State-Dependent DDE (Figure 4.11). NODE and ANODE do not generalize well compared to Neural SDDDE that accurately predicts the dynamics. By studying the effect of such a new history function on the Diffusion Delay PDE, we saw that the system's dynamics is not changed substantially, therefore, we decided to omit this system's comparison.

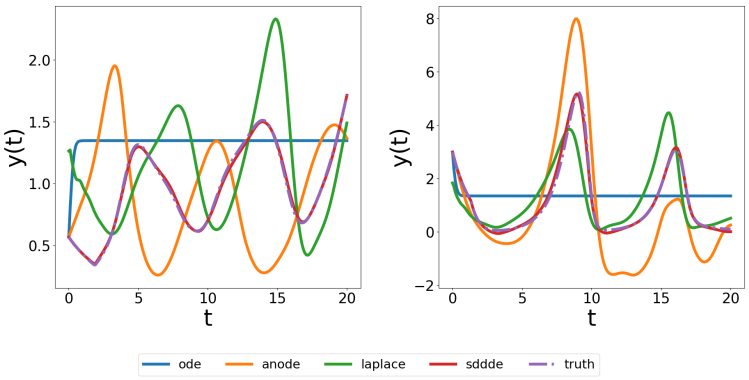

Figure 4.10 – Time Dependent DDE randomly sampled from history step function

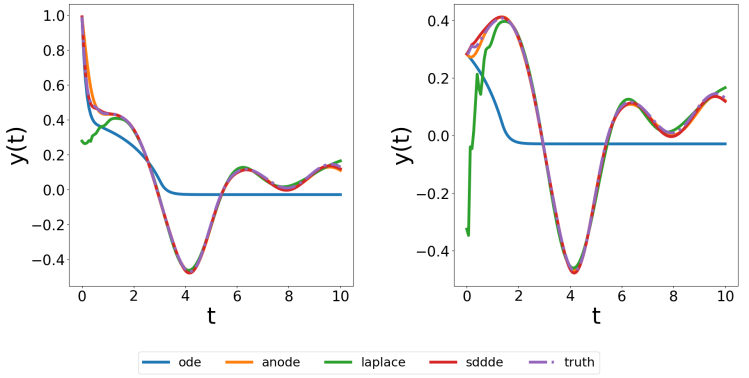

Figure 4.11 – State Dependent DDE randomly sampled from history step function

Noise analysis Finally, we also conduct a noise study on one of the datasets, the Time Dependent DDE system. Each data point is added Gaussian noise that is scaled with a certain factor α of the trajectory's variance. The model is then trained with this noisy data and evaluated on the noiseless test set. In our experiment, we selected 4 scaling factors $\alpha : 0.02, 0.05, 0.1$ and 0.2. Results in Table 4.3 show that our model is robust to noisy data and almost consistently outperforms other models. Additionally, results from Table 4.3 show that adding a small amount of noise (here $\alpha = 0.02$) makes the learning process more robust, a common result in Machine Learning [123, 194].

	NODE	ANODE	Neural Laplace	Neural SDDDE
$\alpha = 0$	$1.01 \pm .435$	$.00729 \pm .00235$	$.0014 \pm .00046$	$\mathbf{.00148 \pm .000872}$
$\alpha = 0.02$	$.720 \pm .00254$	$.0128 \pm .002377$	$.00881 \pm .00254$	$\mathbf{.000906 \pm .000441}$
$\alpha = 0.05$	4.032 ± 4.225	$.03655 \pm .0349$	$.00977 \pm .00146$	$\mathbf{.00250 \pm .000951}$
$\alpha = 0.1$	1.597 ± 1.100	$.0223 \pm .00634$	$.0154 \pm .00501$	$\mathbf{.0121 \pm .00534}$
$\alpha = 0.2$	$1.02 \pm .282$	$.0321 \pm .00319$	$.0273 \pm .00704$	$\mathbf{.0186 \pm .00524}$

Table 4.3 – Test MSE with the noiseless data averaged over 5 runs of each Time Dependent DDE noise experiments with their standard deviation. Best result bolded.

4.5 . Conclusion and Future Work

In this thesis, we introduced Neural State-Dependent Delays Differential Equations (Neural SDDDE) capable of solving DDEs with any type of delays via neural networks. This open-source, robust python DDE solver compatible with neural networks pushes the current envelope of Neural DDEs by handling the delays in a more generic way. To the best of our knowledge, no machine learning library or open-sourced code is available to model such a large class of DDEs.

To validate the effectiveness of Neural SDDDE, we conducted a series of benchmark tests, comparing it against NODEs, the augmented version ANODE, and Neural Laplace. These numerical experiments covered a diverse range of models, including time- and state-dependent scenarios, as well as a delayed Partial Differential Equation (PDE). Our findings revealed that Neural SDDDE accurately reproduced the dynamics across all scenarios tested. Furthermore, Neural SDDDE demonstrated superior performance in terms of accuracy and reliability when compared to the other established methods.

We believe this flexible and versatile tool may provide a valuable contribution to several fields such as control theory where time delays are often considered. In particular, it may prove useful in learning a model for partially observed systems whose dynamics of observables can be learned, under mild conditions, from their time-history.

5 - Neural DDEs with Learnable Delays for Partially Observed Dynamical Systems

Many successful methods to learn dynamical systems from data have recently been introduced. Such methods often rely on the availability of the system's full state. However, this underlying hypothesis is rather restrictive as it is typically not confirmed in practice, leaving us with partially observed systems. Utilizing the Mori–Zwanzig (MZ) formalism from statistical physics, we demonstrate that Constant Lag Neural Delay Differential Equations (NDDEs) naturally serve as suitable models for partially observed states. In empirical evaluation, we show that such models outperform existing methods on both synthetic and experimental data. Code is available here.

5.1 . Introduction

Learning system dynamics is essential in many domains such as biology [152, 45], climate research [52, 78] or finance [2]. In a data-driven context, given a dataset $\{(t_0^l, x_0^l), \ldots, (t_N^l, x_N^l)\}_{l=1}^L$ of observations of a unknown system :

$$\begin{aligned} \frac{\mathrm{d}x(t)}{\mathrm{d}t} &= f(t, x(t)) \\ x(0) &= x_0 \end{aligned} \tag{5.1}$$

we wish to learn a model for the dynamics of $x(t)$ with $x : \mathbb{R} \to \mathbb{R}^n$ and $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$.

In section 4.1, we have seen that such an approach is valid if the system's state is fully observed and the system is Markovian. Neural network based solutions like NODE, ANODE and RNNs were mentioned [30, 44, 74, 155, 69]. However, several applications involve dealing with partially observable states and non-Markovian dynamics, which makes the aforementioned models less optimal. Ultimately, it was highlighted that Neural DDE approaches, as defined in Equation 2.13, incorporate non-Markovian terms through delayed arguments, thereby offering a more realistic modeling alternative for such systems.

The modeling capabilities of NDDEs vary based on the chosen delay type. Inherently, NDDEs, with their delays, incorporate and leverage information from preceding time points, effectively converting the delay term into a dynamic memory mechanism. Initially proposed by Zhu et al. [200] to learn NDDEs with a single constant delay, subsequent work by Zhu et al. [202] and Schla-

ginhaufen et al. [160] explored piece-wise constant delays and developed a stabilizing loss for NDDEs, respectively. Additionally, Oprea et al. [133] focused on learning a single delay within a small network.

In this thesis, we extend these previous contributions by embedding NDDEs in the general framework of the MZ formalism. We explore the possibility of learning the values of the delays at the same time as neural flows for realistic network sizes and extend the state of the art to complex physical applications.

Our main contributions are the following :

- Demonstrate that constant lag NDDEs can model partially observed dynamics with numerical examples and experimental data from fluid mechanics. Code is available here [1].
- Provide an open source package for constant lag DDEs compatible with neural networks, implemented in PyTorch. This implementation allows learning jointly the delay and the DDE's dynamics. Code is available at [2].

5.2 . Modelling Partially Observed Dynamical Systems

In this section, we examine the challenges associated with the typical methods for implementing the Mori-Zwanzig formalism through Integro-Differential Equations and introduce our new approach, which is grounded in Takens' theorem, to address these concerns.

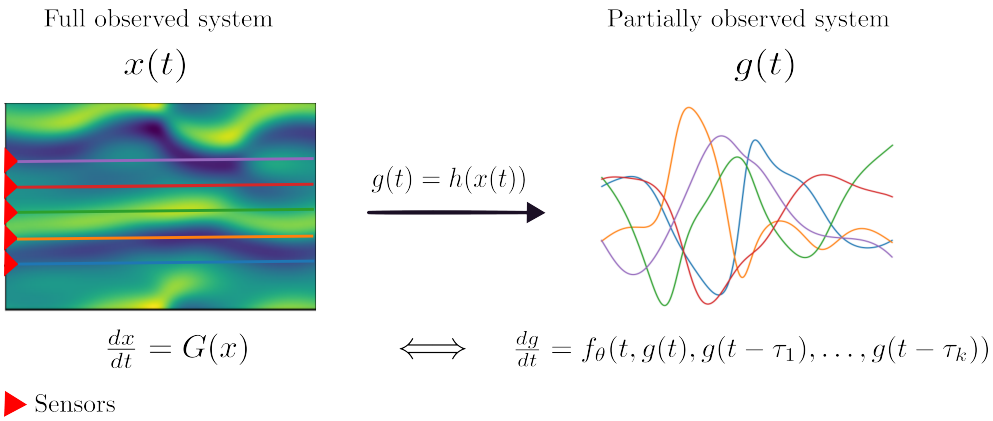

Figure 5.1 – The MZ equation DDE approximation used to model partially observed systems. Here $h(\cdot)$ is a measurement sensing operator.

1. https://github.com/thibmonsel/learnable_delays
2. ttps://github.com/thibmonsel/torchdde

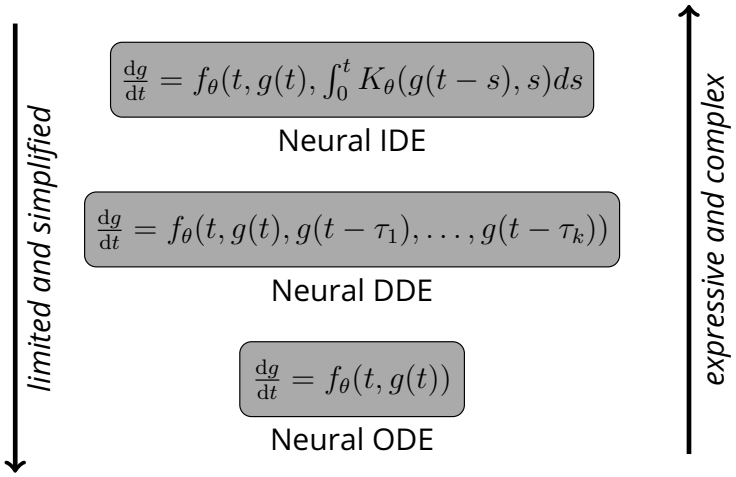

Figure 5.2 – Modelling possibilities

5.2.1 . Approximations of Integro-Differential Equations (IDE)

The Mori–Zwanzig equation (Equation 3.7) outlines the structure of the vector field dynamics for any observable $g(t) := g(x(t))$. However, solving this dynamical equation can prove challenging due to the complexity implied by the integral term and the noise term. Appendix A.8.1 demonstrates some cases where the noise term F can be null, a scenario we set ourselves in. We now discuss several approaches from the literature to estimate this integral.

A simplistic approximation consists in just disregarding the integral term, focusing solely on the impact of the observable g at the current time step t. This corresponds to NODE [29], where the dynamics are approximated as :

$$\frac{\mathrm{d}g}{\mathrm{d}t} \approx f_\theta(t, g(t)).$$

where f_θ is a neural network with parameters θ.

Instead of neglecting it, an approach to approximate the integral consists in studying particular asymptotic regimes. Among the many different models proposed in the literature [168, 32, 169], one of the most popular consists in approximating the integral under assumptions of very short memory or very long memory regimes. For example, the *t-model*, also commonly called *slowly decaying memory approximation* [33] leads to Markovian equations with time-dependent coefficients (see Appendix A.8.2 for full derivation). However, these remain asymptotic approximations, and, more problematically, they are not extendable to intermediate-range memory in general.

Another approach consists in performing Monte Carlo integration [149]. This

work has been extended in a neural network-based formulation (Neural IDE, [196]), where the memory integrand is decomposed as a product of type $K(t,s)F(g(s))$. However, the number of function evaluations required to accurately integrate Equation 3.7 scales with the increasing value of t, making the process computationally intensive. In practice, experiments indicate that Neural IDE is at *least 150 times slower* compared to other models introduced subsequently (see Appendix A.8.4). In a similar spirit, under assumptions of short memory, one can restrict the integral to a short past and discretize it in time, leading to an equation of the form [50] :

$$\frac{dg}{\mathrm{d}t} \approx M(g(t)) - \frac{1}{k}\sum_{i=1}^{k} K(g(t-\tau_i), \tau_i) \tag{5.2}$$

using k delays τ_i uniformly spaced instead of sampling them by Monte Carlo. Such approximations can be improved using high-order discretization schemes, yet, as for Neural IDE, they require an unaffordable number of delays τ_i if the integrand varies quickly or if the interval is too large.

5.2.2 . Exact representation with Neural DDE

While Equation (5.2) only provides an approximation of the true dynamics and requires many delays, we show that using a more complex function of a few delays it is actually possible to represent the dynamics *exactly* :

Proposition 5.2.1 (**Exact representation with delays**). *For any smooth dynamical system (C^2 is enough), and differentiable observables g, using the same notations as for Equation 3.7, there exists almost surely a function M of the current observables, a finite number k of delays $\tau_1, ..., \tau_k > 0$ and a function f such that the observables* exactly *follow the dynamics :*

$$\frac{dg}{dt} \;=\; M(g(t)) \;+\; f\big(t, g(t), g(t-\tau_1), g(t-\tau_2), ..., g(t-\tau_k)\big). \tag{5.3}$$

The proof, deferred to Appendix A.8.5, is based on the application of Takens' theorem, which also provides a bound on the required number of delays : twice the intrinsic dimension of the manifold $\mathcal{S}$ in which the full state x lives, plus one. Note that this evolution equation is exact : approximations may arise from the optimization or the expressivity of the neural networks estimating M and f, but not from the number of delays provided they reach Takens' bound. This is in contrast with the discretization of the IDE integral as in Equation (5.2), which becomes asymptotically precise only when the number of delays becomes large compared to the complexity of the integrand. Note that integral discretization as in Equation (5.2) are doable with a single linear layer of a neural network taking past observables $g(t-\tau_i)$ as input,

and that Proposition 5.2.1 actually states that by stacking more layers one can reach exact representation of the dynamics.

Additionally, this framework is also motivated in practice with climate models from [52, 46] that utilize the MZ formalism to derive DDE structures.

Experiments in Section 5.4 will illustrate that NDDEs where *both the delays and their dynamics are learned jointly* can effectively capture the dynamics of partially observed systems. Before this, we detail how to perform such a training in the next section. In Figure 5.1, the general scenario is highlighted, wherein users have access solely to the system's observables, and we wish to learn their dynamics by using the *MZ equation* approximation (Eq. 5.3). The different models seen in this section, namely Neural-{IDE, DDE, ODE}, are summarized in Figure 5.2.

5.3 . Neural Delay Differential Equations with Learnable Delays

A constant lag NDDE is part of the larger family of continuous depth models that emerged with NODE [29], it is defined by :

$$\begin{aligned} \frac{\mathrm{d}x(t)}{\mathrm{d}t} &= f_\theta(t, x(t), x(t-\tau_1), \ldots, x(t-\tau_k)) \\ x(t \leq 0) &= \psi(t) \end{aligned} \tag{5.4}$$

where $\psi : \mathbb{R} \to \mathbb{R}^n$ be the history function, $\forall i, \tau_i \in \mathbb{R}^+$ be a delay constant and $f_\theta : [0, T] \times \mathbb{R}^n \times \cdots \times \mathbb{R}^n \to \mathbb{R}^n$ be neural network. We refer the reader to Section 2.5 for the training procedure of Neural DDEs with learnable delays.

Learning the delays τ_i within NDDEs is crucial for accurately modeling partially observed dynamics. Indeed, certain delays may not provide relevant enough information (if too small, entries of the delay vector data are too similar ; if too large, the entries tend to be completely uncorrelated and cannot be numerically linked to a consistent dynamical system), therefore delays need to be adapted during training. This is illustrated in the second **Cavity** experiment in Section 5.4.2, where the training of NDDEs with fixed delays remains stuck, while concurrently learning the delays converges to a satisfactory solution. Appendix A.8.6 briefly illustrates how suitable delays (and the information contained within these delayed observables) can impact the model's learning process.

Substantial efforts have been expanded to design a user-friendly API, developing a numerically robust DDE solver, and implementing the adjoint method (referenced as Theorem 2.5.1) in the `torchdde` package. These advancements

ensure a seamless integration of DDEs for future users, enhancing reproducibility. Further details about `torchdde` are available in the supplementary materials and benchmarks are found in Section A.5.

5.4 . Experiments

Numerous experiments have been carried out, categorically addressing two aspects : firstly, validating the existing adjoint approach and assessing the benefits of incorporating learnable constant delays; secondly, examining how neural DDEs with learnable delays is essential to effectively approximate partially observed systems, demonstrated on both synthetic data and experimental data. Additional experiments are also provided in Appendix A.8.7.

In a nutshell, the Brusselator and KS System experiments introduced below showcase multiple delays learning. NDDE is the only model that accurately captures the statistics of the KS System. In the Cavity experiment presented below, learning delays is crucial, as fixed delays fail to capture the system's dynamics correctly.

5.4.1 . Dynamical systems

Toy Dataset We demonstrate that the current approach with the adjoint method can learn jointly the delay and the dynamics of a system used to model population dynamics in biology [6, 8]. Such a described system is formulated through the following DDE :

$$\begin{aligned} &\frac{\mathrm{d}x(t)}{\mathrm{d}t} = x(t)\left(1 - x(t-\tau)\right), \\ &x(t \leq 0) = \psi(t) \end{aligned} \tag{5.5}$$

where we integrate from $t \in [0, 10]$, $\tau = 1$, $\psi(t) = x_0$ and $x(0)$ is sampled from the uniform distribution $\mathcal{U}(2.0, 3.0)$.

The following experiments showcase how NDDEs can effectively model partially observed systems with the systems past state values rather than with opaque latent variables.

Brusselator The Belousov-Zhabotinsky kinetic equation [14, 198] can be modelled by the Brusselator system :

$$\begin{cases} \frac{\mathrm{d}\phi_1(t)}{\mathrm{d}t} = A - B\phi_1(t) - \phi_1(t) + \phi_1(t)^2\phi_2(t) \\ \frac{\mathrm{d}\phi_2(t)}{\mathrm{d}t} = B\phi_1(t) - \phi_1(t)^2\phi_2(t). \end{cases} \tag{5.6}$$

where we integrate in the time domain $t \in [0, 25]$, the initial condition $\phi_1(0)$ is sampled from the uniform distribution $\mathcal{U}(0, 2.0)$ and $\phi_2(0) = 0.0$. We set ourselves in the partially observable case where we only have ϕ_1's dynamics and wish to reconstruct its dynamics.

Kuramoto–Sivashinsky (KS) System We set ourselves in another experiment with the chaotic Kuramoto–Sivashinsky System whose 1D dynamics $u(t, x)$ is :

$$\frac{\partial u}{\partial t} + \frac{\partial^2 u}{\partial x^2} + \frac{\partial^4 u}{\partial x^4} + \frac{1}{2}\frac{\partial u^2}{\partial x} = 0$$

The system is integrated over the time domain $t \in [0, 30]$, and its spatial domain $D_x = [0, 22]$ is discretized into 128 points. To put ourselves in the partially observed setting we choose to observe k features uniformly spread across the spatial domain (here $k = 5$).

Incompressible open cavity flow We consider here as experimental demonstrator the modelling based on time-series derived from wind tunnel experiments of an open cavity flow represented in Figure 5.3 ; the facility is described in [181], where the data is provided in open access with this work are discussed. Open cavity flow attracted numerous research efforts in the last decades for the interesting dynamics at work : the flow is characterized by an impinging shear layer activating a centrifugal instability in a cavity ; this interplay, reminiscent of the feedback acoustic mechanisms described in [151], leads to a self-sustained oscillation. A broad range of dynamics is observed, ranging from limit cycles, to toroidal and chaotic dynamics. The data obtained is for a Reynolds number $Re = 9190$. More details on the experimental setup is given in Tuerke et al. [181].

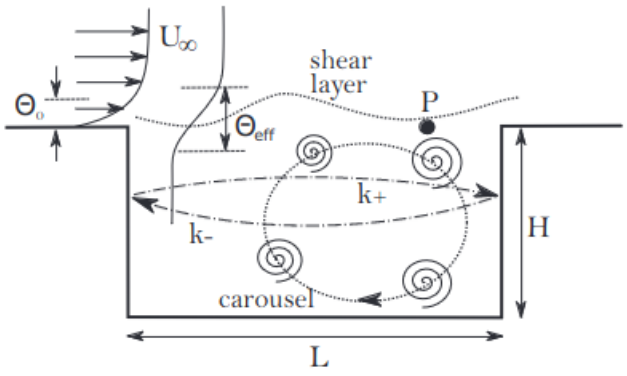

Figure 5.3 – Sketch of open cavity flow. Sensor placed in P.

2. Figure taken from Tuerke et al. [181]

5.4.2 . Results

In this section, we assess the performance of the models with their ability to predict future states of a given partially observed system along with the toy dataset experiment. In this study, LSTM, NODE, ANODE, Latent ODE and NDDE were selected for comparison, and Table 5.1 displays the test MSE loss over each experiment. Appendix A.8.8 goes in more details about each model's architecture and the training and testing procedure. Every model incorporates a form of 'memory' into its architecture, except NODE. While LSTM and Latent ODE utilizes hidden units and ANODE employs its augmented state $a(t)$, NDDE leverages past states such as $x(t-\tau)$. Finally, Table 5.2 outlines the number of delays employed in NDDE for each experiment. We provide in Appendix A.8.3 a general discussion on the delays learned and their evolution during the training process of each experiment. In all subsequent figures, the y-axis $y(t)$ represents our observables, defined as $y(t) = g(x(t))$.

	Brusselator	KS	Cavity
LSTM	$\mathbf{0.0051 \pm 0.0031}$	0.77 ± 0.041	0.75 ± 0.46
NODE	0.77 ± 0.00080	0.71 ± 0.10	0.96 ± 0.0011
ANODE	$\mathbf{0.0050 \pm 0.0050}$	0.55 ± 0.027	0.65 ± 0.0090
NDDE	0.016 ± 0.0076	$\mathbf{0.28 \pm 0.024}$	$\mathbf{0.13 \pm 0.012}$

Table 5.1 – Test loss experiments averaged over 5 runs

	KS	Cavity	Brusselator
NDDE	5	1	2

Table 5.2 – Number of delays used in NDDE for each experiment

Toy dataset Figure 5.4 & 5.5 respectively depict the model's robust convergence to accurate dynamics and the delay evolution during training over many seeds, showcasing a consequence of Takens' theorem [173], that is, by using a delay-coordinate map, one can construct a diffeomorphic shadow manifold M' from univariate observations of the original system in the generic sense. In our case such a lag variable is $(x(t), x(t-\tau))$ with $\tau \in \mathbb{R}^+$. The result of Figure 5.5 may seem surprising as the underlying DDE has a unique delay $\tau = 1$. However, we are not approximating the exact dynamics from the DDE itself but rather from the shadow manifold M'. In classical approaches (see Tan et al. [176, 177]), the selected delay with Takens' theorem for SSR corresponds

to the time series' minimum delayed mutual information which measures the general dependence of two variables [48].

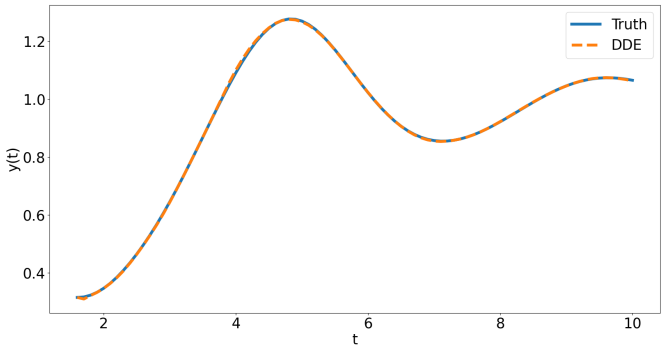

Figure 5.4 – Toy dataset random test sample

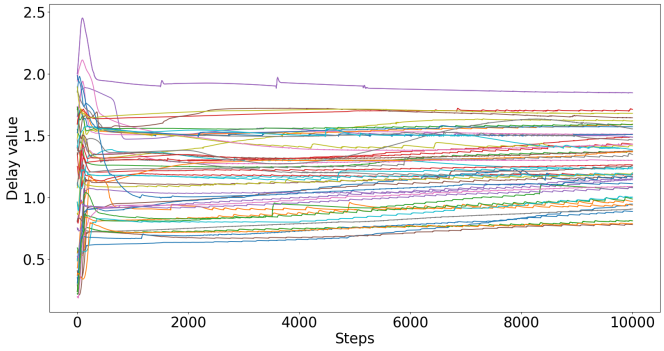

Figure 5.5 – Toy dataset delay evolution during training

Brusselator In the case of this highly stiff and periodic dataset, all models demonstrate satisfactory performance except for NODE. NODE predicts the trajectory's mean thus highlighting the importance of incorporating memory terms. Remarkably, both LSTM and ANODE perform equally well, with NDDE and Latent ODE slightly trailing by a narrow margin as shown in Figure 5.6.

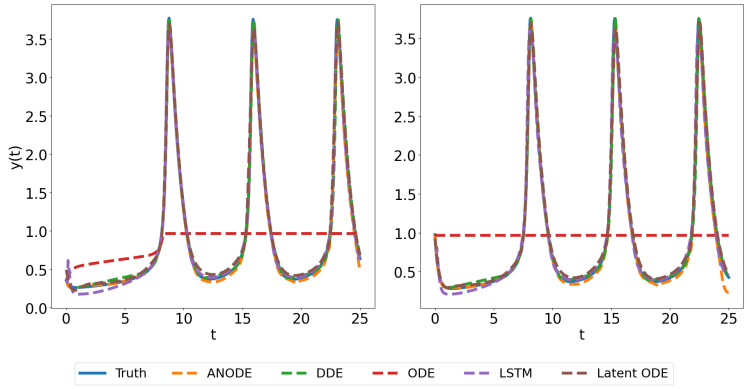

Figure 5.6 – Brusselator random test sample

KS System This experiment deals with a chaotic setting of the partially observed system. By observing periodically k features, the hopes of estimating the high order spatial partial derivatives of $u(x,t)$ is in vain making such formulation of the problem even more challenging. Figure 5.7 & 5.8 showcases random test samples from two different training runs, highlighting how NDDEs outperform other models struggling with the dynamics of the selected features. Furthermore, in a chaotic setting, the statistics of the dynamics prove more informative than the trajectory itself. Figure 5.9 demonstrates that the DDE formulation is distinctive in its adherence to the density distribution of the KS System's test set. This outcome highlights that utilizing past states $x(t-\tau)$ as memory buffers, as opposed to opaque latent variables, can lead to superior results.

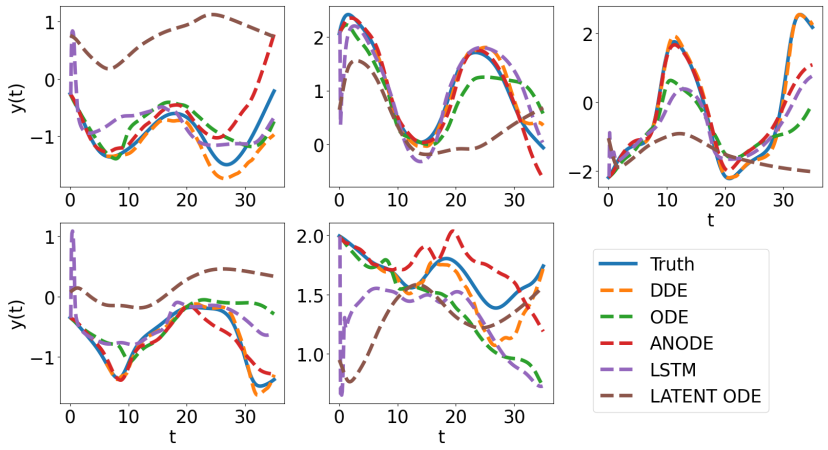

Figure 5.7 – KS random test sample (Part 1)

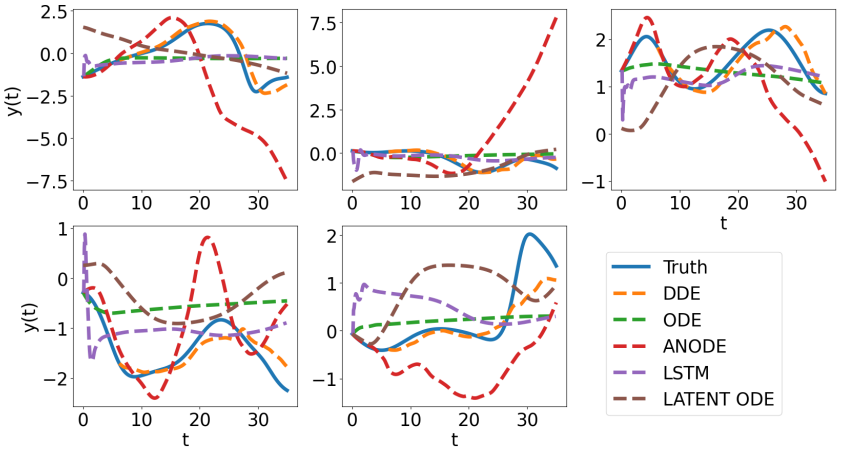

Figure 5.8 – KS random test sample (Part 2)

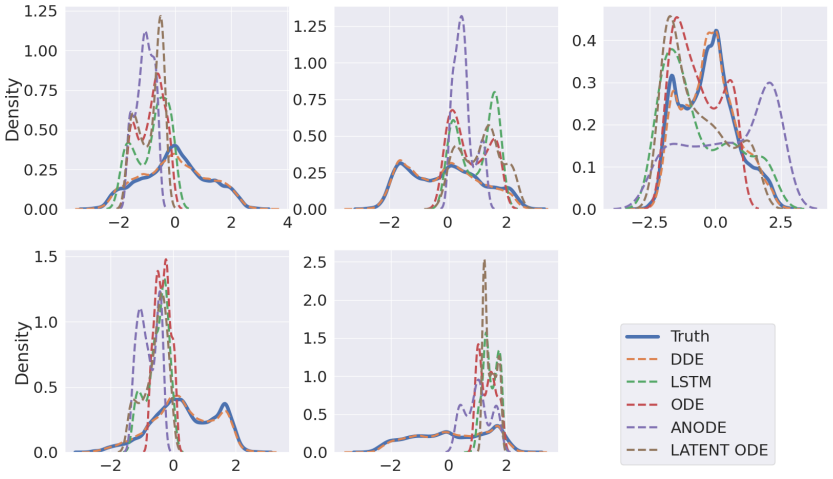

Figure 5.9 – Example of KS test set density plots

Cavity Once more, the NDDE formulation distinguishes itself from other models as seen in Figure 5.10, and this can be attributed to various factors. The experimental setup encourages a delayed formulation of the problem, with the vortex-induced flow originating from the cavity, coupled with partial observability issues. Latent ODE yields acceptable results compared to NODE that generates the system's average trajectory, while LSTM and ANODE capture vague oscillations, albeit occasionally in conflicting phases. Finally, these

experiments demonstrate that NDDE can effectively model trajectories even in the presence of noise in the data.

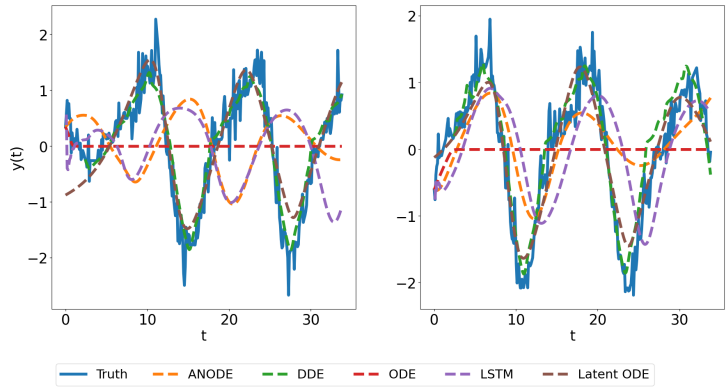

Figure 5.10 – Cavity random test samples

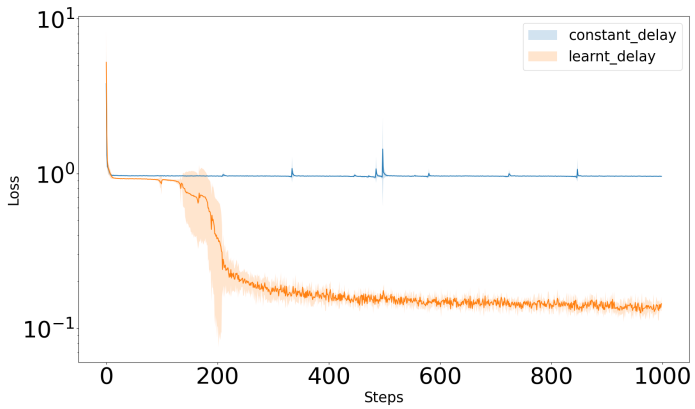

Figure 5.11 – NDDE's with constant and learnable delays MSE train loss averaged over 5 runs

Lastly, we compare NDDE with learnable and fixed delays, initialized at a random value, on our Cavity dataset. Figure 5.11 illustrates a significant difference in the MSE loss magnitude between NDDEs with fixed delays and those with learnable delays, emphasizing the significance of training these delays.

5.5 . Conclusion

In this study, we showcased the capability of constant lag neural delay differential equations (NDDEs) to effectively represent partially observed systems. The theoretical support for this assertion comes from the Mori–Zwanzig

formalism and with its simplification that introduces DDE dynamics. We applied NDDEs to synthetic, chaotic, and real-world noisy data, and conducted comparisons with other continuous-depth and memory based models. The performed experiments revealed two key insights : firstly, the essential role of memory in accurately capturing dynamics; secondly, it was demonstrated that LSTMs' and Latent ODEs' hidden latent states or ANODEs' latent variables are not the exclusive means or sometimes come short to achieve optimal performance, emphasizing the efficacy of delayed terms as an efficient dynamic memory mechanism.

NDDEs come with inherent limitations, such as the linear scaling of its adjoint method with the number of delays (refer to Section 2.5.2 for the case of multiple constant delays). Another question is how to determine the optimal number of delays to consider; this said, overestimating the number of delays does not hurt the final performance. Promising directions for future research involve exploring an equivalent version of ANODEs with NDDEs to assess whether simpler flows can be learned. Additionally, there is a research opportunity to investigate regularization terms that could enhance the NDDE training process. Specifically, we are contemplating the inclusion of a penalty term resembling of a delayed mutual information, inspired by the work of [48].

6 - Non-Markovian closure or correction modelling for dynamical systems

6.1 . Introduction

In many domains, learning and simulating system dynamics is of paramount importance. Many scientific and engineering applications, such as weather forecasting, ocean modeling, and cardiovascular flow simulation, can often be represented by multiscale systems of partial differential equations (PDE) in high-dimensional spaces [136, 189]. In a data-driven context, when dealing with high-dimensional data, reduced order models (ROMs) present a viable alternative; however, fitting the entirety of available data might also be feasible in certain situations. Regardless of the approach taken, it is crucial to recognize that artifacts and errors will inevitably arise in any surrogate model. ROMs are inherently incapable of capturing the interaction between the selected modes and the discarded ones, leading to inaccuracies [9, 7]. Similarly, parametric full-order models (FOMs) in the machine learning field have demonstrated that these models often capture only the dominant Fourier modes of the data, leading to long term discrepancies. [103]. This highlights the significance of incorporating closure terms for ROMs and correction terms for parametric FOMs to ensure the long-term accuracy of the studied system.

In this realm, highly parameterized neural networks can enhance predictive accuracy by introducing Markovian terms to complete the dynamics of ROMs and parametric FOMs. However, a limitation of many closure and correction terms experiments is that they are typically used within a time-discrete framework, which compromises the attractive continuous representation of the system. Furthermore, while Markovian formulations may falter in capturing dynamics not represented by the ROM or parametric FOM, non-Markovian terms have the potential to fill this gap. Non-Markovian terms in the discrete-time formulation have been extensively used with LSTMs [92, 93] but has not been studied in its continuous formulation to the best of our knowledge.

In this work, we propose a novel approach that involves non-Markovian closure and correction term modeling in a continuous time setting through the use of neural networks. We will provide a theoretical context that outlines the challenges we aim to address and highlight the necessity for an additional model alongside a ROM or parametric model to achieve accurate predictions. Following this, we will introduce the non-Markovian modeling framework utilizing Neural Delay Differential Equations (DDEs) and demonstrate its relation

to the MZ formalism. Finally, we will illustrate the capabilities of Neural DDEs through various examples, beginning with an exploration of Markovian and non-Markovian closure terms applied to a ROM for the Kuramoto–Sivashinsky (KS) system, followed by using a parametric model to approximate the full state of the KS system and subsequently for the Kolmogorov flow.

6.2 . Theoretical context

6.2.1 . Dynamical system modelling

We will consider a time-dependent PDE of the form :

$$\begin{aligned} &\frac{\partial u(t,x)}{\partial t} = F(t,x,u,\frac{\partial u}{\partial x},\frac{\partial^2 u}{\partial x^2},\dots) \quad (t,x) \in [0,T]\times\mathbb{X} \\ &u(0,x) = u^0(x), \quad B[u](t,x) = 0 \quad x \in \mathbb{X}, (t,x) \in [0,T]\times\partial\mathbb{X} \end{aligned} \tag{6.1}$$

where $u : [0,T]\times\mathbb{X} \to \mathbb{R}^n$ is the solution, with initial condition $u^0(x)$ at time $t = 0$ and boundary conditions $B[u](t,x) = 0$ when x is on the boundary $\partial\mathbb{X}$ of the domain $\mathbb{X}$. We consider Dirichlet boundary conditions, where the boundary operator $B_{\mathcal{D}}[u] = u - \mathbf{b}_{\mathcal{D}}$ for a fixed function $\mathbf{b}_{\mathcal{D}}$ and Neumann boundary conditions, where $B_{\mathcal{N}}[u] = \mathbf{n}^T ux - \mathbf{b}_{\mathcal{N}}$ for scalar-valued u, where $\mathbf{n}$ is an outward facing normal on $\partial\mathbb{X}$.

Equation 6.1, commonly referred to as the *full order model*, is computationally intensive to simulate when resulting in a high dimensional problem. To address this challenge, our objective is to accurately approximate the high-dimensional solution $u(t,x)$ using one of two approaches. The first approach involves transforming the full order solution u into a reduced field $\bar{u}(t,x)$ through a reduction operator $\mathcal{A}$. The second approach entails fitting a parameterized model F_θ of the full dynamics F. We will first discuss the former approach, followed by the latter. The reduction operator $\mathcal{A}$ can employ various techniques such as filtering, averaging, projection, or truncation. It's worth noting that $\mathcal{A}$ can be either local or non-local in space and time, with convolutions being an example of a non-local operation. The reduced field $\bar{u}$ is defined as follows :

$$\bar{u}(t,x) := \mathcal{A}(u(t,x)), \tag{6.2}$$

where the main issue is that the reduced variable $\bar{u}$ isn't a solution of the FOM Equation 6.1. Thus, the common approach is to use a parameterized reduced model f_{ROM} such that :

$$\frac{\partial \bar{u}(t,x)}{\partial t} \approx f_{ROM}(t,\bar{u}) \tag{6.3}$$

The reduced order model f_{ROM} is usually computationally faster than the original PDE presented in Equation 6.1, while still providing reliable predictions.

This performance advantage makes ROMs a practical alternative to high-fidelity models. Initially developed in the field of fluid mechanics [110], ROMs have evolved over time, giving rise to a variety of innovative techniques. ROMs can be classified into two primary categories : *intrusive* and *non-intrusive* methods. Intrusive ROMs are fundamentally *physics-based*; they utilize the governing equations to project the high-fidelity model onto a reduced order space, allowing for the resolution of lower-dimensional dynamics. In contrast, non-intrusive ROMs are predominantly *data-driven*, making them particularly suitable for modern machine learning-based approaches [111, 24, 51, 60].

The reduction operator $\mathcal{A}$ is often not optimal, which can lead to information loss in the system being studied. When computationally feasible, one approach to address this is to utilize the full state and attempt to develop a parameterized model of the unknown PDE. This approach aligns with the model discovery paradigm, as discussed in various studies [21, 154, 104, 184, 49]. In this scenario, the aim would be to develop a parameterized model F_θ that accurately represents the FOM described in Equation 6.1.

6.2.2 . Bridging the gap with closure and correction terms

By applying the reduction operator $\mathcal{A}$[1] to the FOM Equation 6.1 :

$$\begin{aligned} \mathcal{A}(\frac{\partial u(t,x)}{\partial t}) &= \mathcal{A}(F(t,x,u,\frac{\partial u}{\partial x},\frac{\partial^2 u}{\partial x^2},\dots)) \\ \implies \frac{\partial \bar{u}(t,x)}{\partial t} &= \mathcal{A}(F(t,x,u,\frac{\partial u}{\partial x},\frac{\partial^2 u}{\partial x^2},\dots)) \end{aligned} \tag{6.4}$$

the reduced variable $\bar{u}$'s dynamics becomes unclosed because the PDE's operator F and the reduced operator $\mathcal{A}$ do no commute. Given that the dynamics of $\bar{u}$ are approximated by the reduced model f_{ROM} we can define a closure model f_c that brings the gaps due to non commutativity by setting :

$$f_c(t,\bar{u}) := \mathcal{A}(F(t,x,u,\frac{\partial u}{\partial x},\frac{\partial^2 u}{\partial x^2},\dots)) - f_{ROM}(t,\bar{u}) \tag{6.5}$$

Incorporating this additional flexibility in modeling is crucial for enhancing the accuracy and long-term stability of ROMs [165]. For any given ROM denoted as f_{ROM}, the exact dynamics of the reduced state $\bar{u}(t,x)$ can be expressed as follows :

$$\frac{\partial \bar{u}(t,x)}{\partial t} = f_{ROM}(t,\bar{u}) + f_c(t,\bar{u}) \tag{6.6}$$

where f_c is the closure term model associated to the ROM model f_{ROM}, also known as *unresolved tendency*, or *model error* in different disciplines. Similarly, for the parameterized model F_θ of the full state u, a correction term f_c of the same form that estimates F_θ's error could enhance its prediction. Choosing an

1. Assuming the reduction operator commutes with temporal differentiation

appropriate f_c can be case dependent and is an active research area [3, 158], one straightforward way is to use Equation 6.6's modelling choice and to set the Markovian term f_c as a highly non-linear function parameterized by a neural network. In practice, solving Equation 6.6 numerically typically employs the method of lines, which involves discretizing the spatial dimensions to transform the PDE into an ODE [66]. This leads us to operate within the framework of Neural ODE [30]. This approach takes advantage of the continuous time setting of ODEs and neural networks expressive power.

6.3 . Extending Neural ODE with time delays : Neural DDE

6.3.1 . The issue of non-locality

The inherent nature of closure models necessitates the implicit representation of certain degrees of freedom, a choice driven by both practical considerations and the need for enhanced computational efficiency. This implicit methodology, while beneficial, can introduce non-local effects on the explicitly resolved, large-scale degrees of freedom, presenting a complex challenge in model development. A prime example of this complexity emerges in time-dependent scenarios where closure is employed due to limitations in spatial resolution. In such instances, constructing an accurate closure for a specific resolved point often demands the incorporation of spatio-temporal data from not only adjacent but potentially distant locations. This requirement underscores the intricate interplay between spatial and temporal scales in the system. Recognizing the multifaceted nature of this challenge, our approach leverages temporal information to address the non-local effects that arise as a consequence of the reduction operator $\mathcal{A}$. Importantly, the same principles that necessitate these temporal considerations in the closure model also apply to the correction terms in the parameterized model F_θ. The addition of non-Markovian terms to F_θ serves multiple crucial purposes. It enables the model to handle multiple time scales concurrently, a feature often present in complex systems.

Therefore, such an assessment motivates the use of incorporating delayed values of the reduced state $\bar{u}(t-\tau, x)$ (or the full state $u(t-\tau, x)$) where $\tau \in \mathbb{R}^+$ transforming the ODE formulation into a Delay Differential Equation (DDE) one [12, 131]. The emergence of neural network-based delay differential equations, referred to as **Neural DDEs**, is relatively recent [201, 120, 200]. This DDE approach retains the benefits of Neural ODEs while increasing expressive power through the additional information provided by the delayed arguments in the dynamics equation. With this new formulation, the dyna-

mics with the new closure term is defined by :

$$\frac{\partial \bar{u}(t,x)}{\partial t} = f_{ROM}(t,\bar{u}) + f_c(t,\bar{u}(t-\tau_1,x),\ldots,\bar{u}(t-\tau_k,x)) \tag{6.7}$$

and similarly, with the full state u we have :

$$\frac{\partial u(t,x)}{\partial t} = F_\theta(t,u) + f_c(t,u(t-\tau_1,x),\ldots,u(t-\tau_k,x)) \tag{6.8}$$

where f_c denotes the closure term related to the ROM model f_{ROM} or the correction term related to the imperfect model F_θ and $\tau_1,\ldots,\tau_k$ are constant positive delays.

In a similar vein, authors from [116] derive a specialized ODE model that can be seen as a non-Markovian correction structure, which they call **Complementary Deep Reduced Order Model** (CD-ROM) for ROMs, although this can be used for other parametric models like F_θ. They propose a continuous embedding of the past information with exponential decay for correction (or closure) term that acts as a memory term for u (or $\bar{u}$) :

$$y(t,x) := f_c(t,\bar{u}) = \int_{-\infty}^{t} e^{(t-\tau)\lambda}\bar{u}(\tau,x)d\tau \tag{6.9}$$

This new proposed equation can be solved by augmenting the original system (Equation 6.7 or 6.8) with the memory's $y(t)$ dynamics :

$$\frac{\partial y(t,x)}{\partial t} = \bar{u}(t,x) - \lambda y(t,x) \tag{6.10}$$

Additionally, an encoding map E_θ is used to lift the observations into a higher dimensional space to increase the dimension of the memory and the model's expressiveness :

$$\frac{\partial \bar{u}(t,x)}{\partial t} = f_{ROM}(t,\bar{u}) + \int_{-\infty}^{t} e^{(t-\tau)\lambda}E_\theta(\bar{u}(\tau,x))d\tau \tag{6.11}$$

Thus, the overall augmented ODE formulation of the CD ROM model becomes :

$$\begin{aligned} \frac{\partial \bar{u}(t,x)}{\partial t} &= f_{ROM}(t,\bar{u}) + R_\theta(y) \\ \frac{\partial y(t,x)}{\partial t} &= E_\theta(\bar{u}) - \lambda y(t,x) \end{aligned} \tag{6.12}$$

where R_θ is yet another neural network.

The CD and DDE correction/closure terms exhibit both similarities and distinct differences. The most evident distinction is that the CD-ROM framework

can be solved using standard ODE solvers, whereas this is not the case for DDEs. The CD-ROM's memory term $y(t, x)$ incorporates continuous exponential decay contributions, which comes at the expense of solving a larger system of ODEs compared to the lower dimensional DDE. In contrast, the DDE closure terms remain within the same space, but with discrete delays. Leveraging neural networks in conjunction with DDEs allows for the modeling of non-linear relationships with the delayed terms that have learnable delays, rather than forcing a specific structural form. This flexibility provides an advantage over the CD-ROM's more constrained exponential decay structure.

6.3.2 . Link to Mori–Zwanzig formalism

Through various approximations, the MZ formalism, defined in Section 5.2, has enabled the development of reduced order models and closures for a diverse range of systems [17, 170, 99, 97, 139, 138, 75]. In our case, we can clearly see that both MZ structure (Equation 3.7) and Equation 6.7 & 6.8 do possess similar structures. The parameterized model F_θ or the ROM model f_{ROM} can be associated with the Markovian term, while the closure/correction term represents the memory and noise components. Since the noise term develops in a space orthogonal to the observed variables and often has a minimal impact on the dynamics, the closure/correction terms can be identified with the memory term.

6.4 . Results

In this section, we qualitatively and quantitatively compare the CD, ODE and DDE correction/closure terms introduced in Section 6.2 & 6.3 on a variety of benchmarks hereafter. Such experimentation will show that DDE terms outperforms their CD and ODE counterparts on a wide margin.

6.4.1 . Closure modelling with the ROMs

KS System We choose to experiment with the Kuramoto–Sivashinsky (KS) System [91] whose 1D dynamics $u(t, x)$ is :

$$\frac{\partial u}{\partial t} + \frac{\partial^2 u}{\partial x^2} + \frac{\partial^4 u}{\partial x^4} + \frac{1}{2}\frac{\partial u^2}{\partial x} = 0.$$

The spatial domain is set to $\mathcal{D}_x = [0, L = 22]$, uniformly discretized with 128 points, inducing a chaotic regime. The dataset is divided into training, validation, and test sets consisting of 2048, 128, and 100 trajectories, respectively, all having uniformly random initial conditions. The training, validation and testing trajectories all span the time interval $t \in [0, 30]$.

Our initial experiment employs a POD Galerkin approach, introduced in Appendix A.9.1, as a ROM where we select 4, 8, and 10 POD modes to represent

low, medium, and high energy capture of the system, respectively accounting for 58%, 87% and 94% of the system's energy (i.e. variance) represented in Figure 6.1. In practice, the more modes selected in our POD Galerkin the smaller the correction required by the closure term f_c needs to be. We provide in Appendix A.9.2 details of the training procedure along with the neural network architecture used.

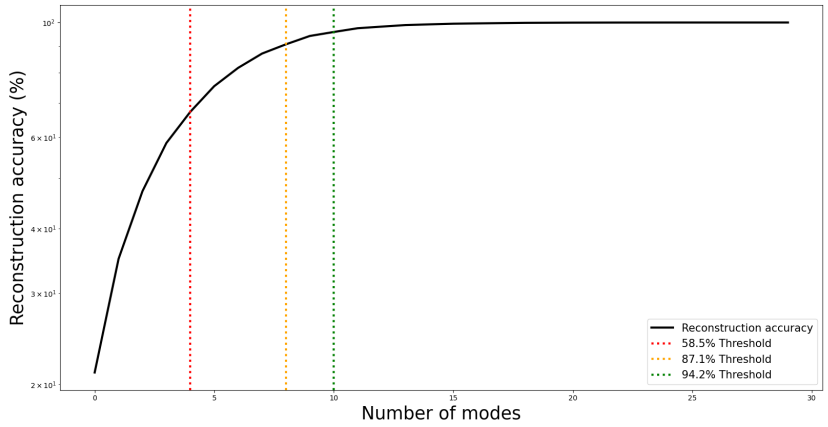

Figure 6.1 – KS's reconstruction accuracy

Figure **??** and Table 6.1 displays the test MSE loss over the three different POD Galerkin ROMs. When utilizing the same model architecture f_c, it is consistently demonstrated that the DDE closure term with learnable delays outperforms the CD and ODE closure terms across all data regimes (low, medium, and high). This performance difference becomes especially pronounced in low data regimes of the 4-mode POD Galerkin ROM, indicating that the Neural ODE closures are not well-suited for such scenarios. Figure 6.2 along with their respective representations in the original state space in Figure 6.3, showcase randomly sampled data points from the test dataset for the 4 modes POD Galerkin ROM. We decided to choose the 4 modes POD Galerkin since the discrepancy between models is the largest. As the number of modes increase, the discrepancy between ODE and DDE closures decrease quantitatively in Table 6.1. This trend is expected since the ROM captures most of the system's information. By studying these specific ROMs, we can identify where ODE closure terms fall short and how incorporating past states with the DDE closure term can address ODE's deficiencies in low data scenarios. Interestingly, the specialized closure model CD-ROM enhances results only in the low-data regime for the POD-Galerkin method. However, introducing an additional high-dimensional proxy variable $\mathbf{y}(t)$ does not improve the learning process of the CD closure term in high-data regimes.

Model	4 modes	8 modes	10 modes
CD-ROM	7.94 ± 0.53	0.43 ± 0.24	0.099 ± 0.019
ODE-ROM	13.61 ± 0.34	0.44 ± 0.044	0.084 ± 0.0028
DDE-ROM	$\mathbf{3.39 \pm 0.034}$	$\mathbf{0.18 \pm 0.07}$	$\mathbf{0.067 \pm 0.0078}$

Table 6.1 – Summary of model performance metrics, i.e. the test MSE loss. CD, ODE, and DDE closure terms are compared across the 4, 8, and 10 modes POD Galerkin ROM.

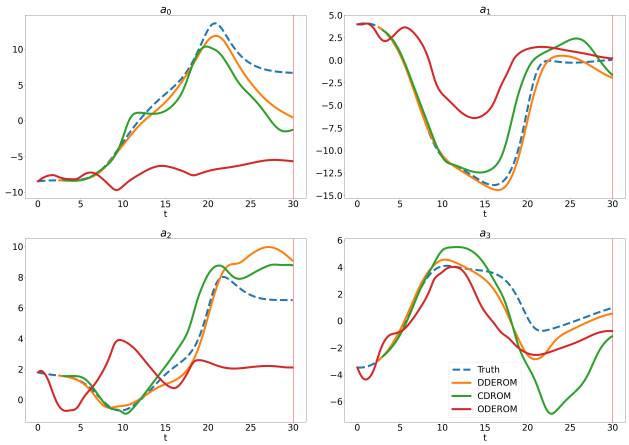

Figure 6.2 – Randomly sampled predictions of POD Galerkin ROM (4 modes) from KS test set

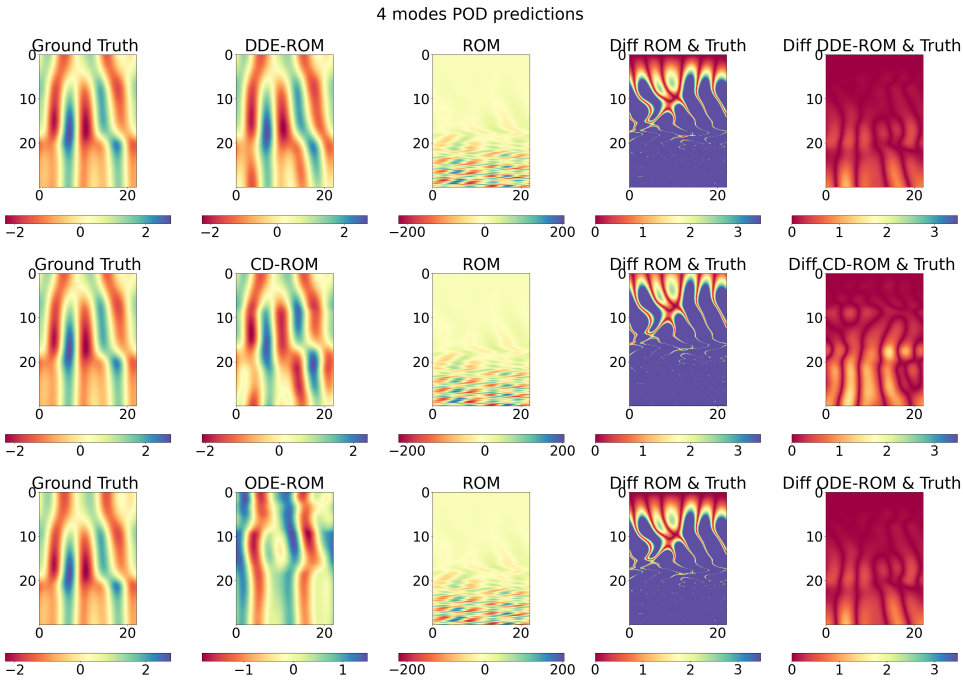

Figure 6.3 – Randomly sampled predictions of POD Galerkin ROM (4 modes) from KS test set reconstructed in original state space

Discussion on the number of delays in Neural DDEs Figures 6.4, 6.5 and 6.6 illustrate the combined training loss for 1, 2 and 3 delays used in the DDE closure term for each POD Galerkin model of the KS system. The red and blue curves represent the training and validation data, respectively. Notably, these curves are unaffected by the number of delays for this particular reduced-order model and dynamical system. Such a result was also found for our correction term experiments in section 6.4.2.

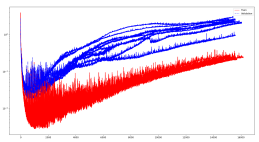

Figure 6.4 – 4 modes train loss for DDE-ROM with {1,2,3} delays

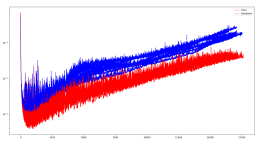

Figure 6.5 – 8 modes train loss for DDE-ROM with {1,2,3} delays

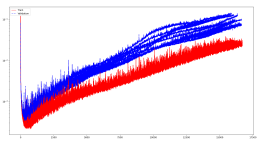

Figure 6.6 – 8 modes train loss for DDE-ROM with {1,2,3} delays

Discussion on Neural DDE's delay values The task of selecting optimal delays has led to extensive research focused on creating metrics for their

evaluation, as highlighted in studies such as [28, 144, 174]. Despite these efforts, a definitive method has yet to be established. For example, a widely used heuristic suggests setting delays as a multiple of one-quarter of the system's period. In our experiments, we circumvented this issue by employing a data-driven approach to learn the delays (i.e. via stochastic gradient descent) and did not observe alignment with these conventional heuristics. Instead, we observed that randomly initialized delays often remained relatively close to their initial values. Further exploration of the delay learning process and its relationship with traditional methods is intended for future research.

6.4.2 . Correction modelling on the KS System and Kolmogorov flow

In our experiment, as a parametric model F_θ we chose a linear model (i.e $F_\theta := A$) that is learned by regression beforehand. Although more complex modelling can be used, our purpose is to convey the utility and efficiency of non-Markovian correction terms compared to other methods.

KS System The third column in Figure 6.8 showcases the linear model's rollout performance on a random sample. Additionally, the fourth column displays its absolute error, which highlights the model's imperfections and indicates that a correction term is necessary to improve its predictive accuracy and reliability.

Table 6.7 showcases the test set MSE of the linear model with the DDE, ODE and CD correction terms. Compared to the POD Galerkin ROM, the DDE correction term of the studied data-driven approach surpasses the ODE one's by a margin. Figure 6.8 visually shows this difference where ODE display numerical artifacts as time grows. The ODE correction term yields non-physical prediction compared to both CD and DDE corrections.

Model	Test MSE
ODE	0.55 ± 0.001
CD	0.23 ± 0.064
DDE	$\mathbf{0.067 \pm 0.017}$

Figure 6.7 – Summary of model performance metrics, comparing the test MSE loss of the linear model with its ODE, CD and DDE correction terms

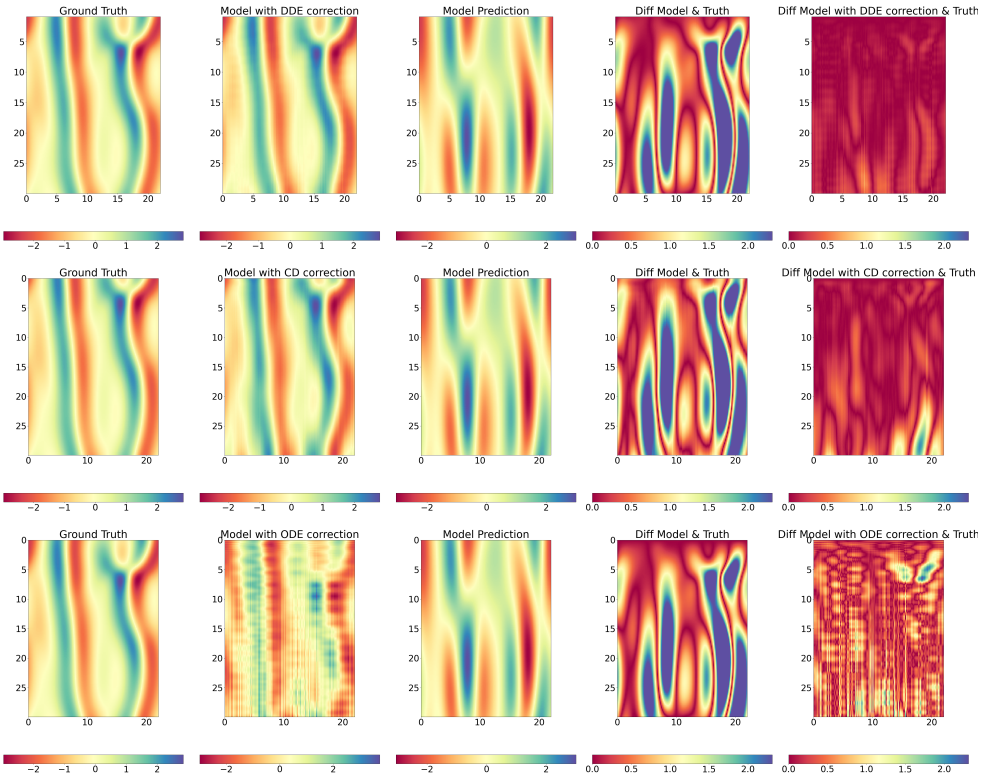

Figure 6.8 – Randomly sampled predictions of the linear model with its ODE, CD or DDE correction terms from the KS test set

Kolmogorov flow As another common fluid-dynamics benchmark, we apply each correction term models to the 2D Kolmogorov flow, a variant of the incompressible Navier-Stokes flow. The time-dependent PDE is defined as :

$$\frac{\partial u}{\partial t} + \nabla \cdot (u \otimes u) = \nu \nabla^2 u - \frac{1}{\rho} \nabla p + \mathbf{f}$$
$$\nabla \cdot u = 0$$

where $u : [0, T] \times \chi \to \mathbb{R}^2$ is the solution, χ the spatial domain, $\otimes$ the tensor product, ν the kinematic viscosity, ρ the fluid density, p the pressure field, and, finally, f the external forcing. Following previous work [171, 85], we set the forcing to $\mathbf{f} = \sin(4y)\hat{x} - 0.1u$, the density $\rho = 1$, and viscosity $\nu = 0.001$, which corresponds to a Reynolds number of 1000 and is integrated up to $T = 4.86$ ($\mathrm{d}t = 0.049$). The package `jax-cfd` is used to generate the 32×32 resolution data [84]. The dataset consists of 290 trajectories and the train, validation, test split opted is 80%, 10% and 10%.

Figure 6.9 showcases the linear model's predictions over time in the second row. These predictions clearly highlight the necessity of incorporating closure terms to achieve improved accuracy.

Model	
ODE-ROM	1.44 ± 0.025
CD ROM	1.23 ± 0.035
DDE-ROM	$\mathbf{1.01 \pm 0.048}$

Table 6.2 – Test MSE loss on the Kolmogorov flow for CD, ODE and DDE corection terms with the linear model.

Table 6.2 summarizes the performance of each correction term on the Kolmogorov flow, once again highlighting the superior performance of the DDE term over its ODE counterparts. Figure 6.9 provides a visual representation of the predictions at various time stamps, with each row depicting the Ground Truth trajectory, the linear model, the linear model with DDE correction term, the linear model with ODE correction term, and the linear model with CD correction term. Additionally, Figure 6.10 includes the absolute error corresponding to each model. Notably, at approximately $t \approx 3.9$, only the DDE correction term is able to reconstruct most of the flows features, while the ODE correction terms quickly diverge to an undesired state. Surprisingly, the CD correction term only marginally improves upon its ODE counterpart.

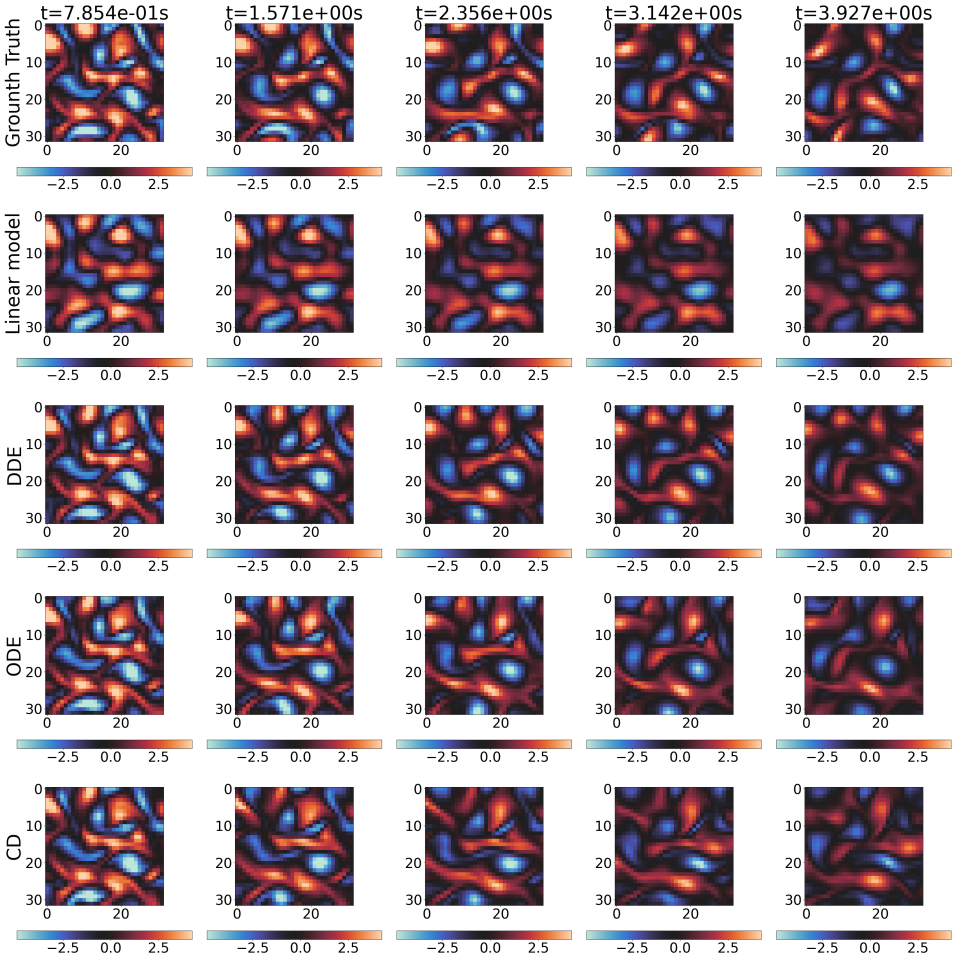

Figure 6.9 – Random test sample from the Kolmogorov flow. CD, ODE and DDE correction terms are compared with the linear model.

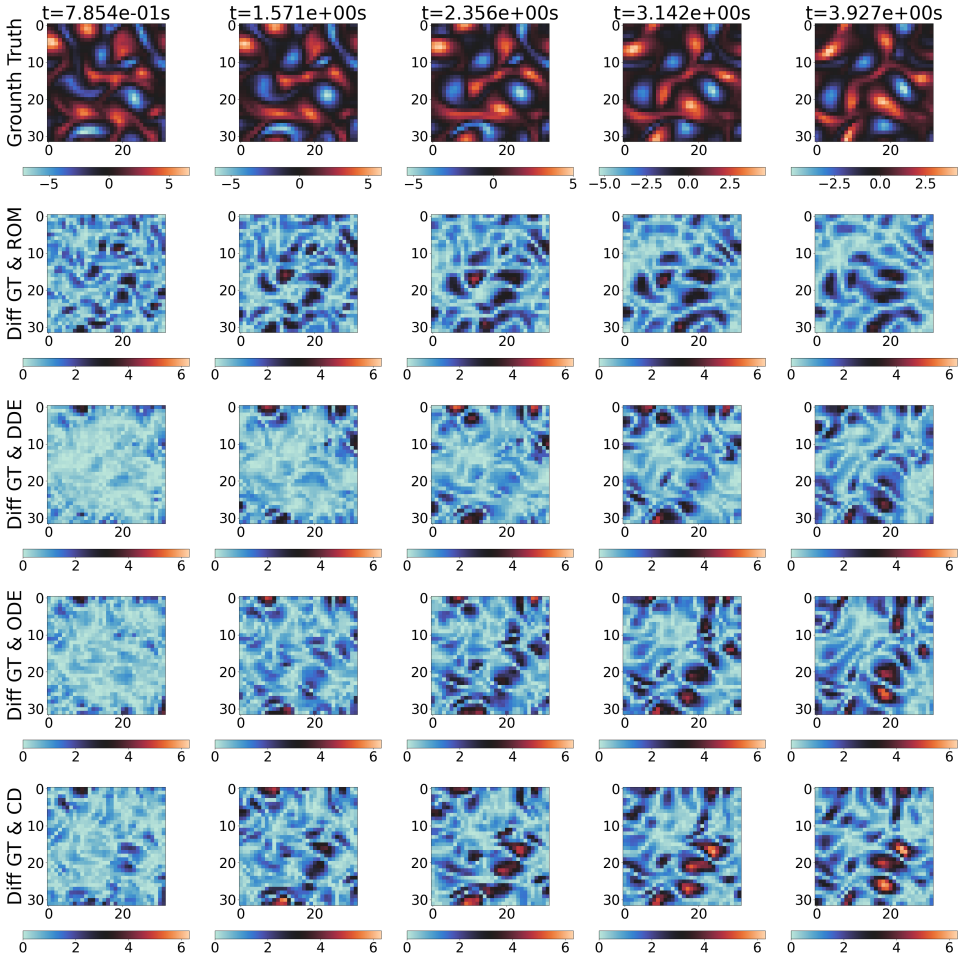

Figure 6.10 – Random test sample from the Kolmogorov flow. The absolute error of CD, ODE and DDE correction terms with the linear model are compared.

6.5 . Conclusion

In this work, we have shown that incorporating non-Markovian correction and closure terms can significantly enhance the modeling of complex dynamical systems. By going beyond traditional ODE-based approaches, our methodology leverages the rich theoretical foundations of the Mori–Zwanzig formalism to capture memory effects and unresolved degrees of freedom. The results on benchmark systems like the Kuramoto–Sivashinsky equation and the Kolmogorov flow demonstrate the superior performance of our approach compared to standard ODE methods. The ability to jointly learn the delay and the correction/closure terms provides greater modeling flexibility and accuracy, unlocking new possibilities for data-driven discovery of governing equa-

tions. In future work, a deeper analysis of delay learning is intended to try to join the non-Markovian literature with the one from time delay embedding.

7 - Conclusion

Each chapter possesses its own conclusion section. This "conclusion" chapter offers some final **personal** thoughts on the future of data-driven dynamical systems modeling and also future works

Final thoughts

With the surge in popularity of AI, the application of these techniques has spread across various scientific domains, and the field of Dynamical Systems is no exception. Several methods are available to fit dynamical systems, with some of the most prominent being Neural DEs, Neural Operators, and, I believe, Neural Fields in the near future. We aim to offer some general insights and reflections on these methodologies.

Neural Differential Equations (DEs) Since the inception of Neural Ordinary Differential Equations (ODE), the field of Neural DEs has rapidly evolved, with the introduction of new methods like Neural Controlled Differential Equations, Neural Stochastic Differential Equations, and Neural Delay Differential Equations (DDEs), as well as their applications in diverse areas such as science and finance. The field of Neural DEs possesses the advantage of leveraging the well-developed numerical differential equation solvers that have been established over the years, though their widespread adoption has been somewhat limited. This delay in widespread adoption could be attributed to the more involved theoretical foundations compared to regular deep learning methods, as well as the relatively higher computational costs associated with training. However, with the advent of more powerful GPU hardware, this perspective on Neural DEs within the community may shift.

Neural Operators have gained significant popularity in recent years and have received more research attention compared to Neural DEs. Neural Operators typically aim to learn a mapping from the current state $u(t)$ to the future state $u(t + \mathrm{d}t)$, where u represents the state and $\mathrm{d}t$ denotes the time step. By applying this mapping recursively, the system can be simulated. Compared to Neural DEs, which require a certain number of intermediate computations (depending on the solver used) to obtain $u(t+\mathrm{d}t)$ from $u(t)$, Neural Operators rely on a single forward pass, allowing for larger models to fit into available RAM. However, Neural Operators generally require small-time steps $\mathrm{d}t$ to be accurate, which can lead to excessive computation time. Additionally, NOs are designed to be invariant to spatial discretization, a feature that Neural DEs

do not inherently possess. Lastly, to obtain states between t and $t + \mathrm{d}t$, NOs must employ interpolation schemes, which can lead to inaccurate estimates. As with any method, there is a trade-off between the various approaches.

Neural Fields is a very recent method that have been used for scientific machine learning. PINNs can be seen as a special case of Neural Fields with a physics-aware loss function. Authors from [63] showed promising results with Transformer architectures. Given that Transformers are the state-of-the-art, this Neural Field point of view coupled with Transformers and large amounts of data could sprout a new promising branch in scientific machine learning.

Dynamics and Fourier Modes The current methods of NOs and most likely Neural Fields and Neural ODEs as well, are limited in their ability to capture only the dominant Fourier modes of the underlying dynamics. This represents a limitation of the existing approaches. To address this, the authors have proposed a new method called PDE-Refiner [102] that can capture higher-order Fourier modes by modeling the solution estimate $u(t + \mathrm{d}t)$ as a refinement process that can be optimized as a denoising objective. This novel methodology presents a very promising research direction for improving the expressiveness of these data-driven dynamical system models.

The prevailing approach in the scientific community in Deep Learning for modeling partially observed systems primarily involves the use of latent variables (e.g., Latent ODEs, LSTMs, GRUs). While this approach is valid, we contend that incorporating transparency with delayed states as non-Markovian features can be advantageous, as we try to have endeavored to demonstrate throughout this thesis. In fact, a significant majority of systems can be modeled using Integro-Differential Equations (IDEs), suggesting that the historical states (or a continuous contribution of past states) should be sufficient for accurate modeling. Nevertheless, it is important to acknowledge that gradient descent methods may (and will) not consistently converge to this optimal representation. This limitation suggests the potential benefits of a hybrid approach, combining both latent variables and delayed states. We propose this as a promising direction for future research, which could potentially offer a more robust and comprehensive modeling framework.

The intersection of Dynamical Systems and Deep Learning remains in its formative stages, and I firmly believe that continued cross-pollination between these fields will catalyze significant advancements in both domains. A promising avenue for future research lies in expanding the capabilities of Neural DDEs. This could involve a thorough investigation of how various delay structures impact model performance, as well as the development and exploration

of specialized auxiliary loss functions tailored for delay components. Such endeavors have the potential to not only enhance our understanding of these hybrid models but also to improve their practical efficacy in capturing complex dynamical behaviors.

Foundational Physics Models comparable to LLMs in the field of Physics, represent the next frontier where we anticipate notable advancements. Recently, IBM and NASA announced new foundational models designed for Weather and Climate Applications [163]. Much like the LLM landscape, the overwhelming majority of these models are based on Transformers. Will Neural DEs be part of this emerging trend? It's a complex question, and my answer is both yes and no. Yes, because these foundational models, behind the scenes, utilize an Euler scheme with a small $\mathrm{d}t$ to simulate dynamics, which allows them to be expressed using Neural DEs. However, the answer is also no, since their training objectives differ from those of Neural DEs; foundational models employ one-time step prediction training losses and leverage rollouts to obtain long-term estimates. In contrast, Neural DEs predict segments of trajectories based on initial conditions and regress on these estimated trajectory segments.

Future Works

Model Predictive Control (MPC) is widely regarded as one of the most effective modern control strategies. Integrating MPC with Neural DDEs for controlling partially observed systems has yet to be done and could offer a powerful framework for controlling complex dynamical systems. Such a combination is expected since DDEs and control theory intersect.

Learning ODE vs DDEs One question that this thesis arises is : When should we prefer DDEs over a neural ODE in a latent space, and vice versa? This question is still open, unanswered and could be a promising research direction. One could think of using an encoder-decoder structure to compress the state and then learn ODE/DDE in the resulting encoded latent space.

Thank you

It remains to thank the reader for their attention. We hope that this thesis has provided a comprehensive overview of the current state of the art in data-driven dynamical systems continuous time modeling, as well as a glimpse into the future of this exciting field. We also hope that neural-network DDEs have gained new supporters and that readers feel encouraged to delve deeper into

this captivating area of research.

7.1 . Synthese en francais

Les systemes dynamiques sont omnipresents dans les sciences et plus particulierement dans l'ingenierie. Ces dernieres sont decrits par des equations differentielles partielles (EDP) qui peuvent être complexes et coûteuses à résoudre. Par conséquent, modéliser ces systèmes est essentiel pour de nombreuses applications, telles que le contrôle ou la prévision à long terme. En général, nous observons ces systèmes de manière incomplète, car nous ne connaissons pas précisément le phénomène sous-jacent ou ne pouvons pas mesurer toutes les variables d'état. La simulation de ces systèmes partiellement observés constitue un défi majeur en raison de l'absence d'informations complètes.

Dans cette thèse, nous explorons comment utiliser les données pour modéliser efficacement des systèmes partiellement observés. En nous appuyant sur l'étude de la théorie des systèmes partiellement observables et le formalisme de Mori–Zwanzig, nous proposons une formulation générale à l'aide d'équations différentielles à retards constants (EDR). Nous suggérons de combiner l'apprentissage profond avec les EDR constants pour développer une nouvelle approche permettant de modéliser ces systèmes dynamiques partiellement observés. Avant de s'attarder sur ce sujet, nous avons etudies les systemes dynamiques qui suivent des EDR avec differents types de delais. Parmi les delais explores, nous avions les delais constants, les delais dependant du temps et les delais dependant de l'etat. De nombreux modèles ont été comparés à ce nouveau type de modèle, et il a été conclu que les modèles à retard étaient les plus performants. En revisitant les équations différentielles à retard (EDR) constantes appliquées aux systèmes dynamiques, nous avons établi la méthode de l'adjoint spécifique à ces modèles, tout en soulignant que le retard pouvait être paramétré. Intégrer le retard comme paramètre dans le processus d'apprentissage a permis d'obtenir un modèle plus flexible et performant comparé à son homologue avec un retard constant. Cette intégration a été motivée par la théorie de l'embédologie, qui a pris son essort apres le théorème de Takens. En effet, ce domaine etudie entre autres et explorer des criteres pour essayer de caracteriser la qualite du retard lors de l'apprentissage d'un systeme oartiellement observees. Des recherches préliminaires ont été réalisées pour comparer et observer les similitudes entre des techniques populaires de ce domaine et notre approche exclusivement basée sur les données. Il apparaît qu'aucune des valeurs de retard de nos expériences ne corrélait avec ces critères. Une voie de recherche prometteuse consisterait à explorer cela plus en détail et à intégrer des fonctions de coûts qui pourraient être en adéquation avec ces critères.

Ensuite, une étude sur les modèles réduits et imparfaits, ainsi que leur

couplage avec des termes de correction ou de fermeture, a été menée sur des systèmes dynamiques. Nous avons comparé des termes purement markoviens, non markoviens grâce aux EDR, et hybrides, tels que les modèles récemment introduits CD-ROM [116]. Les exemples étudiés sont classiques dans la littérature, tels que le système de Kuramoto-Shivanksi et l'écoulement de Kolmogorov. Les principaux points à retenir de cette étude sont que, dans le cas des modèles réduits contenant peu d'informations sur le système, les fermetures à EDR surpassent largement les autres méthodes. Lorsque les modèles réduits sont de bien meilleure qualité, les fermetures EDR demeurent les meilleures options, bien que les autres méthodes réduisent considérablement l'écart. En ce qui concerne les modèles imparfaits, les modèles de correction avec EDR se sont révélés les plus performants, mais avec des marges moins significatives.
Enfin, un dernier aspect de la thèse consiste à appliquer des modèles simulant des EDR avec le contrôle prédictif basé sur le modèle (*model predictive control*). Ce chapitre est actuellement en cours de rédaction et les résultats seront exposés dans la version finale de la thèse.

A - Appendix

A.1 . Backpropagation

Let $f_1, \ldots, f_l$ be some collection of functions whose derivatives we know how to compute and f be the composition of these functions, i.e. $f : x \mapsto f_l \circ f_{l-1} \circ \cdots \circ f_1(x)$. Via chain rule we can compute the derivative of the output with respect to the input as follows :

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\mathrm{d}f_l}{\mathrm{d}f_{l-1}} \frac{\mathrm{d}f_{l-1}}{\mathrm{d}f_{l-2}} \cdots \frac{\mathrm{d}f_1}{\mathrm{d}x} \tag{A.1}$$

Autodifferentation (AD) packages do the computation of Equation A.1 for us by successively computing the known derivatives of the functions f_i. Two main approaches are used to evaluate Equation A.1 that differ in the order in which the Jacobian $\frac{\mathrm{d}f_i}{\mathrm{d}f_{i-1}}$ are multiplied.

Forward-mode Forward-mode AD, known as forward sensitivity in optimization, computes $\frac{\mathrm{d}f}{\mathrm{d}x}$ by evaluating the terms from right to left, i.e. by recursively computing :

$$\frac{\mathrm{d}f_i}{\mathrm{d}x} = \frac{\mathrm{d}f_i}{\mathrm{d}f_{i-1}} \frac{\mathrm{d}f_{i-1}}{\mathrm{d}x}, \quad i \in \{2, \ldots, l\} \tag{A.2}$$

Forward-mode chooses an input variable and calculates the sensitivity of every intermediate variable with respect to that input variable.

Reverse-mode Reverse-mode AD, also known as reverse sensitivity in optimization or as backpropagation in machine learning, computes $\frac{\mathrm{d}y}{\mathrm{d}x}$ by evaluating the terms from left to right, i.e. by recursively computing :

$$\frac{\mathrm{d}f_l}{\mathrm{d}f_{i-1}} = \frac{\mathrm{d}f_l}{\mathrm{d}f_i} \frac{\mathrm{d}f_i}{\mathrm{d}f_{i-1}}, \quad i \in \{l-1, \ldots, 1\} \tag{A.3}$$

where we stipulate that $f_0 = x$. Reverse-mode chooses to calculate the sensitivity of the output with respect to each of the intermediate variables. Reverse-mode AD needs a forward pass to compute the intermediate variables and a backward pass to compute the sensitivities. These intermediate values need to be stored during the forward pass to be used during the backward pass (in `torch` this is seen with the argument `torch.requires_grad`).

Efficiency Depending on the function f at hand, one approach can be more efficient than the other. In machine learning where we are used to minimizing a scalar-valued loss function, the reverse-mode is much more efficient. Suppose that x is a vector (i.e. network's parameters), the intermediate layers $f_{i<l}$ are vectors (intermediate layers are vectors) and f_l is a scalar-valued function (i.e. the loss function). Evaluating Equation A.1 is a matrix-matrix multiplication and Equation A.1 is a matrix-vector multiplication. The latter is much more efficient than the former and this why backpropagation is the method of choice in machine learning to optimize a model.

Remark A.1.1. *Here we provide a quick example of forward-mode and reverse-mode in practice. Let a function $f : \mathbb{R}^{n_i} \to \mathbb{R}^{n_o}$ defined as follows that we wish to compute is derivative with respect to its input x :*

$$y = f(x) = k \circ l \circ m(x) \tag{A.4}$$

where $x \in \mathbb{R}^{n_i}$, $m : \mathbb{R}^{n_i} \to \mathbb{R}^{n_m}$, $l : \mathbb{R}^{n_m} \to \mathbb{R}^{n_l}$, $k : \mathbb{R}^{n_l} \to \mathbb{R}^{n_o}$ and $\circ$ is the composition operator. By using intermediate variables $a = m(x)$, $b = l(a)$ and $c = k(b)$, we can rewrite f and its derivative as follows :

$$a = m(x), \quad b = l(a), \quad y = k(b) \tag{A.5}$$

Via chain rule :

$$\underbrace{\frac{\mathrm{d}y}{\mathrm{d}x}}_{n_o \times n_i} = \underbrace{\frac{\partial k(b)}{\partial b}}_{n_o \times n_k} \underbrace{\frac{\partial l(a)}{\partial a}}_{n_k \times n_l} \underbrace{\frac{\mathrm{d}m(x)}{\mathrm{d}x}}_{n_l \times n_i} \tag{A.6}$$

where the under braces represent the size of each Jacobian matrices. Forward-mode AD, will compute :

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{\partial k(b)}{\partial b} \left(\frac{\partial l(a)}{\partial a} \frac{\mathrm{d}m(x)}{\mathrm{d}x} \right). \tag{A.7}$$

which involves $n_k \cdot n_i \cdot (n_l + n_o)$ multiplications. In reverse-mode AD will compute :

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \left(\frac{\partial k(b)}{\partial b} \frac{\partial l(a)}{\partial a} \right) \frac{\mathrm{d}m(x)}{\mathrm{d}x}. \tag{A.8}$$

which involves $n_o \cdot n_l \cdot (n_k + n_i)$ multiplications. Depending on the values of n_i, n_l, n_k and n_o, one approach can be more efficient than the other.

A.2 . Proof of theorem 2.3.2 ODE Adjoint

We have the following optimization problem :

$$\begin{aligned} &\arg\min_{\theta} \;\; L(x(T)), \\ \textit{s.t.} \;\; &L(x(T)) = \int_0^T l(x(t))\mathrm{d}t, \\ &\frac{\mathrm{d}x(t)}{\mathrm{d}t} - f_\theta(t, x(t)) = 0, \\ &x(0) = x_0 \end{aligned} \tag{A.9}$$

Generally, these kinds of problems are solved with a gradient descent method, which requires evaluation of the following gradient :

$$\frac{dL}{d\theta} = \frac{\partial L(x(T))}{\partial x}\frac{\partial x(T)}{\partial \theta} \tag{A.10}$$

We evaluate the sensitivity of the following Lagrangian :

$$J = L(x(T)) + \int_0^T \lambda(t)\left(\frac{\mathrm{d}x(t)}{\mathrm{d}t} - f_\theta(t, x(t))\right)\mathrm{d}t \tag{A.11}$$

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = f_\theta(x, t) \implies \frac{dJ}{d\theta} = \frac{dL}{d\theta} \tag{A.12}$$

The vector of Lagrangian multiplier λ is a function of time. Distributing the product in integrating the first terms by parts leads to an expression where the sensitivity $\frac{\partial x}{\partial \theta}$ can be isolated.

$$J = L(x(T)) + [\lambda(t)x(t)]_0^T - \int_0^T \left(\frac{\mathrm{d}\lambda(t)}{\mathrm{d}t}x(t) + \lambda(t)f(x, t, \theta)\right)\mathrm{d}t \tag{A.13}$$

$$\begin{aligned} \implies \frac{dJ}{d\theta} = &\left(\frac{\partial L(x(T))}{\partial x} + \lambda(T)\right)\frac{\partial x(T)}{\partial \theta} - \underbrace{\lambda(0)\frac{\partial x(0)}{\partial \theta}}_{=0} \\ &- \int_0^T \left(\frac{\mathrm{d}\lambda(t)}{\mathrm{d}t} + \lambda(t)\frac{\partial f(t, x(t))}{\partial x}\right)\frac{\partial x(t)}{\partial \theta} + \lambda(t)\frac{\partial f(t, x(t))}{\partial \theta}\mathrm{d}t \end{aligned} \tag{A.14}$$

$$\begin{aligned} \implies \frac{dJ}{d\theta} = &-\int_0^T \left(\frac{\mathrm{d}\lambda(t)}{\mathrm{d}t} + \lambda(t)\frac{\partial f(t, x(t))}{\partial x}\right)\frac{\partial x(t)}{\partial \theta}\mathrm{d}t + \left(\frac{\partial L(x(T))}{\partial x} + \lambda(T)\right)\frac{\partial x(T)}{\partial \theta} \\ &- \int_0^T \lambda(t)\frac{\partial f(t, x(t))}{\partial \theta}\mathrm{d}t \end{aligned} \tag{A.15}$$

In order to avoid the computation of $\frac{\partial x}{\partial \theta}$ we derive the adjoint dynamics and its initial condition which is :

$$\frac{d\lambda(t)}{dt} = -\lambda(t)\frac{\partial f(t, x(t))}{\partial x}, \quad \lambda(T) = -\frac{\partial L(x(T))}{\partial x} \tag{A.16}$$

Going back to the gradient of the cost function w.r.t. to the network's parameters θ (Equation A.15), we obtain :

$$\frac{dL}{d\theta} = -\int_0^T \lambda(t)\frac{\partial f(t, x(t))}{\partial \theta}\mathrm{d}t \tag{A.17}$$

A.3 . Proof of theorem 2.5.1 DDE Adjoint

Proof is inspired from Calver and Enright [25] and put into ML context.

We want to solve the optimization problem where τ may appear in our parameter vector θ. For convenience, we use the following notation $y(t) = x(t-\tau)$ and $\alpha(t) = t - \tau$.

$$\begin{aligned} & \arg\min_{\theta} \; L(x(T)), \\ \text{s.t. } & L(x(T)) = \int_0^T l(x(t))\mathrm{d}t, \\ & \frac{\mathrm{d}x(t)}{\mathrm{d}t} - f_\theta(t, x(t), x(t-\tau)) = 0, \\ & x(t \leq 0) = \psi_\theta(t). \end{aligned} \tag{A.18}$$

We consider the following Lagrangian :

$$\begin{aligned} J &= L(x(T)) + \int_0^T \lambda(t)\Big(\frac{\mathrm{d}x(t)}{\mathrm{d}t} - f_\theta(t, x(t), x(t-\tau))\Big)\mathrm{d}t. \\ \implies \frac{\mathrm{d}J}{\mathrm{d}\theta} &= \frac{\mathrm{d}L}{\mathrm{d}\theta} \end{aligned} \tag{A.19}$$

Integration by parts yields :

$$J = \int_0^T l(x(t))\mathrm{d}t + \Big[\lambda(t)x(t)\Big]_0^T - \int_0^T \Big(\frac{\mathrm{d}\lambda(t)}{\mathrm{d}t}x(t) + \lambda(t) f_\theta(t, x(t), x(t-\tau))\Big)\mathrm{d}t. \tag{A.20}$$

Taking the derivative w.r.t. θ :

$$\begin{aligned} \frac{\mathrm{d}J}{\mathrm{d}\theta} = & \int_0^T \frac{\partial l(x(t))}{\partial x}\frac{\partial x(t)}{\partial \theta}\mathrm{d}t + \lambda(T)\frac{\partial x(T)}{\partial \theta} - \lambda(0)\frac{\partial x(0)}{\partial \theta} \\ & + \int_0^T -\frac{\mathrm{d}\lambda(t)}{\mathrm{d}t}\frac{\partial x(t)}{\partial \theta}\mathrm{d}t + \int_0^T -\lambda(t)\frac{\partial f_\theta(t, x(t), x(t-\tau))}{\partial \theta}\mathrm{d}t \\ & + \int_0^T -\lambda(t)\frac{\partial f_\theta(t, x(t), x(t-\tau))}{\partial x}\frac{\partial x(t)}{\partial \theta}\mathrm{d}t \\ & + \int_0^T -\lambda(t)\frac{\partial f_\theta(t, x(t), x(t-\tau))}{\partial y}[\frac{\partial x(t-\tau)}{\partial \theta} + x'(t-\tau)\frac{\partial \alpha(t)}{\partial \theta}]\mathrm{d}t. \end{aligned} \tag{A.21}$$

Rearranging integrals

We rework the first part of the last term to close the equation on $x(t)$:

$$\int_0^T -\lambda(t)\frac{\partial f_\theta(t, x(t), x(t-\tau))}{\partial y}\frac{\partial x(t-\tau)}{\partial \theta}\mathrm{d}t = \int_{-\tau}^{T-\tau} -\lambda(t+\tau)\frac{\partial f_\theta(t+\tau, x(t+\tau), x(t))}{\partial y}\frac{\partial x(t)}{\partial \theta}\mathrm{d}t \tag{A.22}$$

Choosing the multipliers so that $\lambda(t \geq T) = 0$, we get :

$$\int_0^T -\lambda(t)\frac{\partial f_\theta(t, x(t), x(t-\tau))}{\partial y}\frac{\partial x(t-\tau)}{\partial \theta}\mathrm{d}t = \int_{-\tau}^T -\lambda(t+\tau)\frac{\partial f_\theta(t+\tau, x(t+\tau), x(t))}{\partial y}\frac{\partial x(t)}{\partial \theta}\mathrm{d}t \tag{A.23}$$

By splitting the integral in two parts, we get :

$$\begin{aligned}\int_0^T -\lambda(t)\frac{\partial f_\theta(t, x(t), x(t-\tau))}{\partial y}\frac{\partial x(t-\tau)}{\partial \theta}\mathrm{d}t = &\int_{-\tau}^0 -\lambda(t+\tau)\frac{\partial f_\theta(t+\tau, x(t+\tau), x(t))}{\partial y}\frac{\partial \psi_\theta(t)}{\partial \theta}\mathrm{d}t \\ &+ \int_0^T -\lambda(t+\tau)\frac{\partial f_\theta(t+\tau, x(t+\tau), x(t))}{\partial y}\frac{\partial x(t)}{\partial \theta}\mathrm{d}t\end{aligned} \tag{A.24}$$

Adjoint Equation

Finally, injecting this result, we rearrange the terms in Eq.A.21 :

$$\begin{aligned}\frac{\mathrm{d}J}{\mathrm{d}\theta} = &-\int_0^T \Big(\frac{\mathrm{d}\lambda(t)}{\mathrm{d}t} - \frac{\partial l(x(t))}{\partial x} + \lambda(t)\frac{\partial f_\theta(t, x(t), x(t-\tau))}{\partial x} + \lambda(t+\tau)\frac{\partial f_\theta(t, x(t+\tau), x(t))}{\partial y}\Big)\frac{\partial x(t)}{\partial \theta}\mathrm{d}t \\ &- \int_0^T \lambda(t)\left(\frac{\partial f_\theta(t, x(t), x(t-\tau))}{\partial \theta} + \frac{\partial f_\theta(t, x(t), x(t-\tau))}{\partial y}x'(t-\tau)\frac{\partial \alpha(t)}{\partial \theta}\right)\mathrm{d}t \\ &- \int_{-\tau}^0 \lambda(t+\tau)\frac{\partial f_\theta(t+\tau, x(t+\tau), x(t))}{\partial y}\frac{\partial \psi_\theta(t)}{\partial \theta}\mathrm{d}t \\ &+\cancelto{0}{\lambda(T)\frac{\partial x(T)}{\partial \theta}} - \lambda(0)\frac{\partial x(0)}{\partial \theta}\end{aligned} \tag{A.25}$$

The last term vanishes because of the chosen adjoint final condition $\lambda(t \geq T) = 0$, thus we get the following adjoint dynamics, to be integrated backwards in time :

$$\begin{aligned}&\frac{\mathrm{d}\lambda(t)}{\mathrm{d}t} = \frac{\partial l(x(t))}{\partial x} - \lambda(t)\frac{\partial f_\theta(t, x(t), x(t-\tau))}{\partial x} - \lambda(t+\tau)\frac{\partial f_\theta(t+\tau, x(t+\tau), x(t))}{\partial y}, \\ &\lambda(t \geq T) = 0.\end{aligned} \tag{A.26}$$

Hence, the gradient's loss w.r.t to the parameters is :

$$\begin{aligned}\frac{\mathrm{d}L}{\mathrm{d}\theta} = &-\int_0^T \lambda(t)\left(\frac{\partial f_\theta(t, x(t), x(t-\tau))}{\partial \theta} + \frac{\partial f_\theta(t, x(t), x(t-\tau))}{\partial y}x'(t-\tau)\frac{\partial \alpha(t)}{\partial \theta}\right)\mathrm{d}t \\ &- \int_{-\tau}^0 \lambda(t+\tau)\frac{\partial f_\theta(t+\tau, x(t+\tau), x(t))}{\partial y}\frac{\partial \psi_\theta(t)}{\partial \theta}\mathrm{d}t - \lambda(0)\frac{\partial x(0)}{\partial \theta}\end{aligned} \tag{A.27}$$

We will set ourselves in the case where the history function doesn't depend on θ (it will given as input to the neural network), the gradient simplifies even further.

$$\frac{\mathrm{d}L}{\mathrm{d}\theta} = -\int_0^T \lambda(t)\left(\frac{\partial f_\theta(t, x(t), x(t-\tau))}{\partial \theta} + \frac{\partial f_\theta(t, x(t), x(t-\tau))}{\partial y} x'(t-\tau)\frac{\partial \alpha(t)}{\partial \theta}\right)\mathrm{d}t \tag{A.28}$$

Usually, the term τ will be parameterized by a constant, thus $\frac{\partial \alpha(t)}{\partial \theta} = \frac{\partial t-\tau}{\partial \tau} = -1$, given us :

$$\frac{\mathrm{d}L}{\mathrm{d}\theta} = -\int_0^T \lambda(t)\left(\frac{\partial f_\theta(t, x(t), x(t-\tau))}{\partial \theta} - \frac{\partial f_\theta(t, x(t), x(t-\tau))}{\partial y} x'(t-\tau)\right)\mathrm{d}t \tag{A.29}$$

Notes on the derivative of the loss

Practically, the loss L is evaluated from a finite number N of points in time :

$$L(x(T)) = \int_0^T l(x(t))\mathrm{d}t \tag{A.30}$$

$$= \int_0^T \Big[\sum_{i=1}^N \bar{l}(x(t_i))\delta(t-t_i)\Big]\mathrm{d}t. \tag{A.31}$$

With $\bar{l}$ a function computing the objective for each sampled point. This yields the following gradient :

$$\frac{\partial l(x(t))}{\partial x} = \sum_{i=1}^N \frac{\partial \bar{l}(x(t_i))}{\partial x}\delta(t-t_i). \tag{A.32}$$

This term is then always null, except for $t = t_i$, this is why the adjoint dynamics in Eq.(A.25) are integrated from one sampling point t_i to the previous t_{i-1}, where the adjoint state is incremented as follows :

$$\lambda(t_i^-) = \lambda(t_i^+) - \frac{\partial \bar{l}(x(t_i))}{\partial x}. \tag{A.33}$$

which corresponds to integrating the Dirac in Eq.(A.32) in reverse time for an infinitesimal time.

A.4 . Proof of theorem 2.7.2 IDE adjoint

Proof is adapted from [196]. We wish to solve the optimization problem :

$$\begin{aligned} &\min_{\theta} \;\; L(x(T)), \\ \textit{s.t.} \;\; &L(x(T)) = \int_0^T l(x(t))\mathrm{d}t, \\ &\frac{\mathrm{d}x(t)}{\mathrm{d}t} - f_\theta(t, x(t)) - \int_{\alpha(t)}^{\beta(t)} K_\theta(t,s)F_\theta(x(s))ds = 0 \\ &x(0) = x_0. \end{aligned} \tag{A.34}$$

We consider the following Lagrangian :

$$J = L(x(T)) + \int_0^T \lambda(t)\left(\frac{\mathrm{d}x(t)}{\mathrm{d}t} - f_\theta(t, x(t)) - \int_{\alpha(t)}^{\beta(t)} K_\theta(t,s)F_\theta(x(s))ds\right)\mathrm{d}t. \tag{A.35}$$

Integration by parts yields :

$$\begin{aligned} J = &\int_0^T l(x(t))\mathrm{d}t + \Big[\lambda(t)x(t)\Big]_0^T \\ &- \int_0^T \frac{\mathrm{d}\lambda(t)}{\mathrm{d}t}x(t) + \lambda(t)\left(f_\theta(t, x(t)) + \int_{\alpha(t)}^{\beta(t)} K_\theta(t,s)F_\theta(x(s))ds\right)\mathrm{d}t. \end{aligned} \tag{A.36}$$

Taking the derivative w.r.t. θ and highlighting the terms that contain $\frac{\partial x}{\partial \theta}$ in blue, we get :

$$\begin{aligned} \frac{\mathrm{d}J}{\mathrm{d}\theta} = &\int_0^T \frac{\partial l(x(t))}{\partial x}\frac{\partial x(t)}{\partial \theta}\mathrm{d}t + \lambda(T)\frac{\partial x(T)}{\partial \theta} - \lambda(0)\cancelto{0}{\frac{\partial x_0}{\partial \theta}} \\ &+ \int_0^T -\frac{\mathrm{d}\lambda(t)}{\mathrm{d}t}\frac{\partial x(t)}{\partial \theta}\mathrm{d}t + \int_0^T -\lambda(t)\frac{\partial f_\theta(t, x(t))}{\partial x}\frac{\partial x(t)}{\partial \theta}\mathrm{d}t + \int_0^T -\lambda(t)\frac{\partial f_\theta(t, x(t))}{\partial \theta}\mathrm{d}t \\ &+ \int_0^T \int_{\alpha(t)}^{\beta(t)} -\lambda(t)\left(\frac{\partial K_\theta(t,s)}{\partial \theta}F_\theta(x(s)) + K_\theta(t,s)[\frac{\partial F_\theta(x(s))}{\partial x}\frac{\partial x(s)}{\partial \theta} + \frac{\partial F_\theta(x(s))}{\partial \theta}]\right)ds\mathrm{d}t. \end{aligned} \tag{A.37}$$

Let's rearrange Equation A.37 to avoid computation of $\frac{\partial x}{\partial \theta}$ and we chose the adjoint initial condition $\lambda(T) = 0$:

$$\begin{aligned}\frac{\mathrm{d}J}{\mathrm{d}\theta} = &\int_0^T \left(\frac{\partial g(x(t))}{\partial x} - \frac{\mathrm{d}\lambda(t)}{\mathrm{d}t} - \lambda(t) \frac{\partial f_\theta(t, x(t))}{\partial x} \right) \frac{\partial x(t)}{\partial \theta} \mathrm{d}t \\ &+ \int_0^T \int_{\alpha(t)}^{\beta(t)} -\lambda(t) K_\theta(t,s) \frac{\partial F_\theta(x(s))}{\partial x} \frac{\partial x(s)}{\partial \theta} ds \mathrm{d}t. \\ &+ \int_0^T -\lambda(t) \frac{\partial f_\theta(t, x(t))}{\partial \theta} \mathrm{d}t \\ &+ \int_0^T \int_{\alpha(t)}^{\beta(t)} -\lambda(t) \left[\frac{\partial K_\theta(t,s)}{\partial \theta} F_\theta(x(s)) + K_\theta(t,s) \frac{\partial F_\theta(x(s))}{\partial \theta} \right] ds \mathrm{d}t.\end{aligned} \tag{A.38}$$

We can isolate $\frac{\partial x(t)}{\partial \theta}$ of the second integral term by using the trick $\frac{\partial x(s)}{\partial \theta} = \frac{\partial x(s)}{\partial x(t)} \frac{\partial x(t)}{\partial \theta}$ and getting the formulae of the adjoint dynamics :

$$\begin{aligned}&\frac{\mathrm{d}\lambda(t)}{\mathrm{d}t} = \frac{\partial l(x(t))}{\partial x} - \lambda(t) \frac{\partial f_\theta(t, x(t))}{\partial x} - \lambda(t) \int_{\alpha(t)}^{\beta(t)} K_\theta(t,s) \frac{\partial F_\theta(x(s))}{\partial x} \frac{\partial x(s)}{\partial x(t)} ds \\ &\lambda(T) = 0.\end{aligned} \tag{A.39}$$

The term $\frac{\partial x(s)}{\partial x(t)}$ is null except when $s = t$ where it is equal to 1. Therefore, the final adjoint dynamics is :

$$\begin{aligned}&\frac{\mathrm{d}\lambda(t)}{\mathrm{d}t} = \frac{\partial g(x(t))}{\partial x} - \lambda(t) \frac{\partial f_\theta(t, x(t))}{\partial x} - \lambda(t) K_\theta(t,t) \frac{\partial F_\theta(x(t))}{\partial x} \\ &\lambda(T) = 0.\end{aligned} \tag{A.40}$$

Going back to the gradient of the cost function w.r.t to the network's parameters θ (Equation A.40), we obtain :

$$\begin{aligned}\frac{\mathrm{d}J}{\mathrm{d}\theta} = &+ \int_0^T -\lambda(t) \frac{\partial f_\theta(t, x(t))}{\partial \theta} \mathrm{d}t \\ &+ \int_0^T \int_{\alpha(t)}^{\beta(t)} -\lambda(t) \left[\frac{\partial K_\theta(t,s)}{\partial \theta} F_\theta(x(s)) + + K_\theta(t,s) \frac{\partial F_\theta(x(s))}{\partial \theta} \right] ds \mathrm{d}t.\end{aligned} \tag{A.41}$$

A.5 . `torchdde` Package

The `torchdde` package is a Python library that offers a flexible and user-friendly API for solving Delay Differential Equations (DDEs) with constant delays in PyTorch. Developed during the thesis, it addresses the gap in DDE support within the machine learning community. The package is designed for seamless integration into existing PyTorch workflows and features numerically robust DDE solvers. Additionally, it implements the adjoint method for training DDEs with constant delays, enabling joint optimization of the vector field and delays with this method. In this section, we outline the key features of the `torchdde` package, and present some experimental checks to validate the codebase. You can check out the repository in GitHub [1] !

A.5.1 . Key Features

`torchdde` is a library designed for training neural networks with Constant Lag (i.e., constant delays) Delay Differential Equations (DDEs) in PyTorch. It offers several key features :

- This package includes various solvers for ODEs and DDEs, such as Euler, Runge–Kutta 2 (RK2), RK4, Dormand–Prince (Dormand–Prince 5(4)), and Implicit Euler.
- These solvers are implemented with PyTorch operations and are therefore compatible with the autograd functionality, enabling smooth integration with neural networks. The adjoint method is also included.
- Users can easily train a DDE model with learnable delays.
- The library features a straightforward API that facilitates easy incorporation into existing PyTorch workflows. Specifically, `torchdde` provides a single entry point function, `integrate`, for integrating DDEs.
- Additionally, the package can be utilized for training ODEs, despite the availability of other libraries like `torchdiffeq` or `diffrax`.

A.5.2 . Code verification

We conducted several experiments to validate our `torchdde` codebase, presented below.

First experiment

In order to validate the learning of delays with `torchdde`, we first perform a sanity check on a simple toy problem. We consider a simple DDE with a single delay, defined by the following equation :

1. https://github.com/thibmonsel/torchdde

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = x(t)(1 - x(t - \tau))$$
$$x(t < 0) = x_0$$

The initial condition is set to $x_0 = [2.0, 3.0, 4.0]$, with a delay of $\tau = 1$, and the system is integrated using the Dormand–Prince 5(4) solver over the interval $t \in [0, 20]$. We define a model with only the parameter τ and provide the ground-truth vector field (i.e $x(t)(1 - x(t - \tau))$), and train it using the Dormand–Prince 5(4) solver. Figure A.1 illustrates the model's loss landscape, highlighting the effect of τ on the loss (using MSE loss, though other types could work as well). The convexity of the loss landscape makes it an ideal candidate to demonstrate the model's capability to learn the delay using both the adjoint method (optimize-then-discrete) and traditional backpropagation (discrete-then-optimize). Figure A.2 presents the loss curve during training with both methods. The model successfully converges to the actual delay value $\tau = 1$, and both training methods yield identical loss curves when initialized with the same learning rate, optimizer, and delay. This validates the correct implementation of the adjoint method and the `torchdde` package.

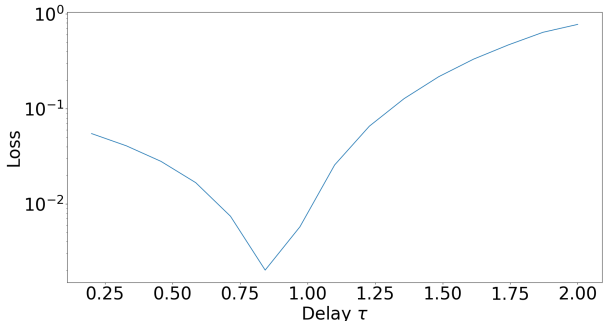

Figure A.1 – Influence of the delay τ on the loss

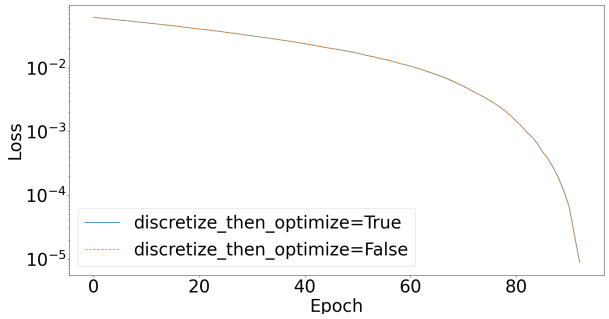

Figure A.2 – Training loss curve for the adjoint method and traditional backpropagation

Second experiment

In order to validate the learning of several delays with `torchdde`, we perform another test on a simple toy problem. We consider a simple DDE with two delays, defined by the following equation :

$$\begin{aligned} \frac{\mathrm{d}x(t)}{\mathrm{d}t} &= -x(t-\tau_1) - x(t-\tau_2) \\ x(t<0) &= x_0 \end{aligned}$$

The initial condition is set to $x_0 = [3.0]$, the delay $\tau_1 = 0.5, \tau_2 = 0.25$ and integrated using the Dormand–Prince 5(4) solver for $t \in [0, 3]$. We define a model with only the parameters $\{\tau_1, \tau_2\}$ and provide the ground-truth vector field, and train it using the Dormand–Prince 5(4) solver. Figure A.3 (along with its 3D projection in Figure **??**) illustrates the model's loss landscape, highlighting the effect of $\{\tau_1, \tau_2\}$ on the loss (using MSE loss, though other types could work as well). The convexity of the loss landscape makes it an ideal candidate to demonstrate the model's capability to learn the delay using both the adjoint method (optimize-then-discrete) and traditional backpropagation (discrete-then-optimize). Figure A.4 presents the loss curve during training with both methods. The model successfully converges to the actual delay values, and both training methods yield similar loss curves when initialized with the same learning rate, optimizer, and delay. Once again, this validates the correct implementation of the adjoint method and the `torchdde` package. Regarding the discrepancy in the loss in Figure A.4 between the two methods, it is due to the fact that the adjoint method is numerically less precise than the traditional backpropagation method. This is a known issue with the adjoint method, and the discrepancy is expected to some degree.

Remark A.5.1. *The task of fitting the various delay parameters results in a convex optimization landscape that is exceptionally flat. This can pose difficulties for the*

application of stochastic gradient descent (SGD) optimization. Situations where both the vector field and the delay parameters require learning are not frequently encountered in practice

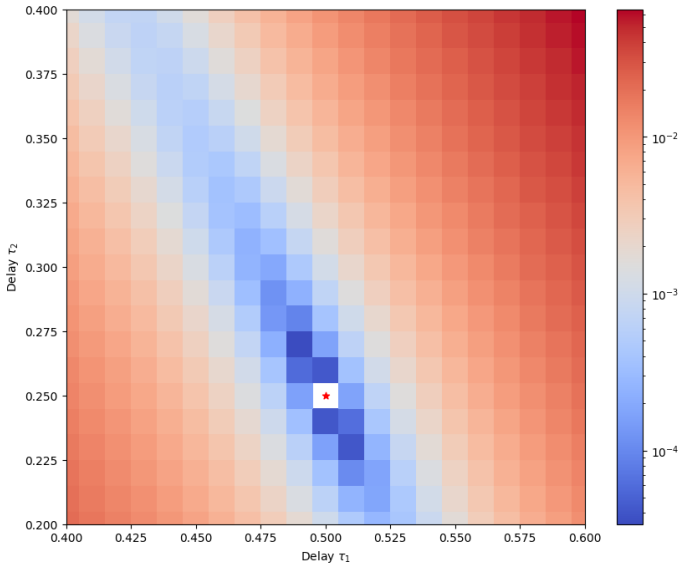

Figure A.3 – Influence of the delay τ_1, τ_2 on the loss for the two delays system where $\star$ indicates the optimal delay values

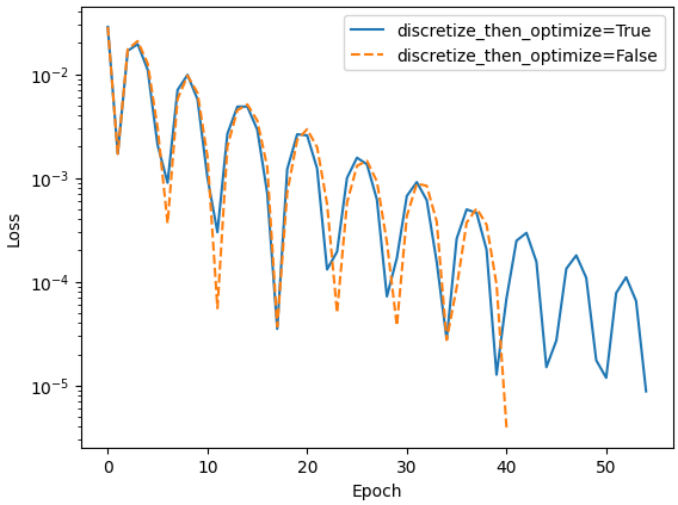

Figure A.4 – Training loss curve for the adjoint method and traditional backpropagation for the two delays system

Third experiment

Another aspect to validate is the joint training of delays and the vector field. While we are not addressing a simple toy problem in this paragraph, Section 5.4 thoroughly demonstrates this capability.

Time and memory benchmarks

We provide time and memory benchmarks on some of `torchdde`'s solvers. In order to compare both training methods of optimize-then-discretize (i.e. the adjoint method) and discretize-then-optimize (i.e. regular backpropagation), we present the Brusselator's experiment (seen in Section 5.4) time duration and memory usage for various solvers during training (with a batch size of 1024) in Tables A.1 and A.2, respectively. The results are as expected : the adjoint method is slower (by a small factor) and consumes less memory than the regular backpropagation. These results are consistent with NODE's examination of the adjoint method and conventional backpropagation trade offs [29].

	Adjoint	Backpropagation
RK4	4.8 ± 0.23	1.89 ± 0.09
RK2	$2.4 \pm .005$	0.90 ± 0.005
Euler	1.5 ± 0.01	0.47 ± 0.003

Table A.1 – Clock Time (s) per batch

	Adjoint	Backpropagation
RK4	2.2 ± 18	2.87 ± 4
RK2	2.15 ± 20	2.48 ± 3
Euler	2.09 ± 15	2.264 ± 9

Table A.2 – GPU consumption (Gb $\pm$ Mb) per batch

Figure A.5 & A.6 compares respectively time and memory consumption for a forward pass of a Neural DDE with varying number of delays, each having approximately 28k parameters. We use a setup that includes an RK4 solver with a time step of $\mathrm{d}t = 0.1$, a batch size of 128, Neural DDE with delays of [1, 3, 5, 10, 20], a state dimension of $g(t)$ equal to 100, an upper integration bound varying from $t = 0.2$ to $t = 5.0$ seconds, and where the notation "#i" in the figure indicates the number of delays used in the Neural DDE.

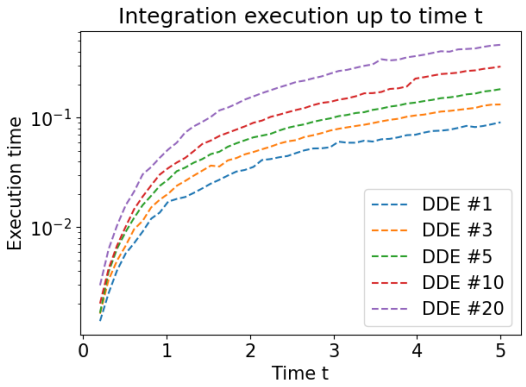

Figure A.5 – Time duration of forward pass averaged over 5 runs

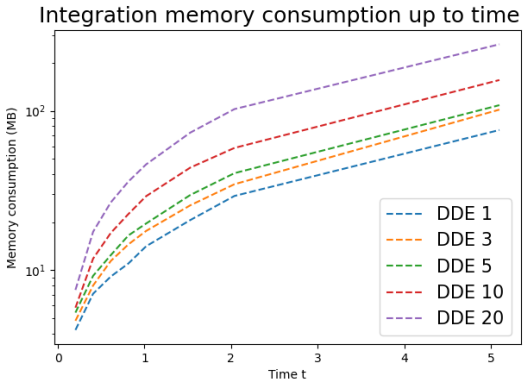

Figure A.6 – Memory consumption of forward pass averaged over 5 runs

Figures A.5 and A.6 demonstrate that the time duration increases linearly in relation to both the integration time t and the number of delays, and that the memory allocation similarly scales linearly with the integration time t and the number of delays.

Discretize then optimize or optimize then discretize for DDEs?

he decision between using the continuous adjoint method and traditional backpropagation involves a trade-off between time efficiency and memory usage. The findings in Chapter 5 were achieved through the optimize-then-discretize method. In contrast, the results presented in Chapter 6, which were obtained later in the PhD, employed the discretize-then-optimize approach.

Ultimately, the discretize-then-optimize method (i.e., regular backpropagation) leads to faster training and improved practical performance, an observation also found for Neural ODE in Patrick's Kidger thesis [80].

A.6 . Overview in DDE integration

A.6.1 . Introduction

A general purpose DDE is defined as follows :

$$\begin{aligned} &\frac{dx(t)}{\mathrm{d}t} = f_\theta(t, x(t), x(t-\tau_1), \ldots, x(t-\tau_k)) \\ &\tau_i = \tau_i(t, x(t)), \quad \forall i \in [\![1, k]\!] \\ &x(t \leq 0) = \psi(t), \end{aligned} \tag{A.42}$$

where θ represent the parameters, $f_\theta : \mathbb{R} \times \mathbb{R}^{d_1 \times \cdots \times d_k} \times \cdots \times \mathbb{R}^{d_1 \times \cdots \times d_k} \to \mathbb{R}^{d_1 \times \cdots \times d_k}$ is an arbitrary neural network, $\psi : \mathbb{R} \to \mathbb{R}^{d_1 \times \cdots \times d_k}$ the DDE's history function, $\tau_i : \mathbb{R} \times \mathbb{R}^{d_1 \times \cdots \times d_k} \to \mathbb{R}^+$ the delay functions and $x : [0, T] \to \mathbb{R}^{d_1 \times \cdots \times d_k}$ the equation's solution. The history function ψ is analogous to the ODE's initial condition.

To provide a gentle introduction, we examine a DDE featuring a single constant delay $\tau \in \mathbb{R}^+$ and a constant history function to be integrated over the time interval $[0, T]$. The DDE takes the form :

$$\begin{aligned} &\frac{dx(t)}{\mathrm{d}t} = f(t, x(t), x(t-\tau)) \\ &x(t \leq 0) = x_0 \end{aligned} \tag{A.43}$$

In this specific scenario, at $t = 0$, $\psi'(t = 0^-) \neq x'(t = 0^+)$. This discrepancy commonly occurs, leading to a lack of smooth linkage between the solution x and the history function $\psi(t)$ at $t = 0$, guaranteeing only C^0 continuity. This phenomenon, known as a derivative jump (or discontinuity or breaking points), can propagate from the initial time $t = 0$ throughout the time interval $[0, T]$, as we will see later on. Consequently, vigilance is necessary regarding these breaking points when integrating DDE.

On the time interval $t \in]0; \tau[$ there are no discontinuities and the DDE becomes

$$\begin{aligned} &\frac{dx(t)}{\mathrm{d}t} = f(t, x(t), x_0) \\ &x(0) = x_0 \end{aligned} \tag{A.44}$$

This formulation problem reshapes the DDE problem into an ODE one that we know how to solve with ease. Let us introduce the interpolated function $\psi_1(t)$ to be the solution of the DDE of Equation A.44 on the interval $]0; \tau[$.

On the time interval $t \in]\tau; 2\tau[$ there are no discontinuities and the DDE becomes :

$$\begin{aligned} \frac{dx(t)}{\mathrm{d}t} &= f(t, x(t), \psi_1(t-\tau)) \\ x(\tau) &= \psi_1(0) \end{aligned}$$

Once again we have an ODE on this interval. Iteratively, we can solve the DDE on successive intervals and this will yield a piecewise continuous solution because of the initial point discontinuity.

One might have noticed that during an integration step, we need the interpolated function of $x(t-\tau)$. This means that DDE methods are based on the *continuous extensions* of numerical ODE schemes (i.e. we need to have an interpolated solution available at all times during integration).

Remark A.6.1. *We observed that DDEs with constant delays exhibit discontinuity points at multiples of τ. However, this behavior differs for DDEs with time- or state-dependent delays, as we will explore later. Monitoring these breaking points is essential for the accurate integration of DDEs.*

A.6.2 . Discontinuity tracking

In the general case, discontinuity that arise from the delay terms are not known a priori unless we are dealing with constant delays [164]. During each integration step of our DDE, one must check for discontinuities by checking the roots of the following functions g_{is} defined in Equation A.45 [206]. Let us stipulate that we integrate from t_n to t_{n+1} and the previous detected discontinuities are $\{\lambda_{-m}, \ldots, \lambda_0, \ldots, \lambda_{r-1}\}$ where the first $m+1$ jumps $\{\lambda_{-m}, \ldots, \lambda_0\}$ are given by the history function and the initial point and the rest were found during previous integration steps. Let

$$\forall (i,s) \in [\![0, r]\!] \times [\![-m, r-1]\!], \quad g_{is}(t) = t - \tau_i - \lambda_s \tag{A.45}$$

The new discontinuity λ_r is defined as

$$\lambda_r = \min\{\lambda > \lambda_{r-1}, \lambda \text{ is a root of odd multiplicity of } g_{is}(t)\}$$

If λ_r is null then the integration step is valid otherwise you redo one from t_n to λ_r. A detailed algorithm procedure is given in [206].

The first to do such an iterative process to find the discontinuities λ_r and modify the integration step bounds is Paul [140]. An alternative approach relies on step size control was proposed by Oberle and Pesch [132] and Neves

[127]. These methods, give up on tracking the discontinuities, which are instead assumed to be automatically included by estimating the error of the integration step. A rejected step will result in a detection of a discontinuity jump and this is the default implementation done in Julia *DelayDiffEq* package [188].

For an ODE method of order p, we usually ask the solution to be at least $\mathcal{C}^{p+1}$ continuous. Therefore, to have a successful integration, it is crucial to include in the mesh of points all the discontinuity of $\frac{\mathrm{d}^k y}{\mathrm{d}t^k}$ at least for $k \leq p+1$. Consequently, discontinuity tracking needs to abide by these rules.

A.6.3 . Unconstrained time stepping

Regardless of the method chosen to integrate a DDE; relying on the error estimate of the step size method or tracking the breaking points, being able to take arbitrarily large steps that are suggested by the numerical solver is a nice to have. This means that sometimes the formulation of our problem becomes implicit because our approximation solution x applied to all delays terms in our integration step is simply not yet known. This makes the overall method implicit even if the discrete method we are using is explicit. We call this occurrence overlapping, Zivari-Piran and Enright [206] shows that the issue at hand is well-defined and solvable for time and state dependent delays.

Let us briefly describe the algorithmic procedure when we are dealing with overlapping (ie $t_{n+1} - t_n > \tau$). Given equation A.43 and an integration step from t_n to t_{n+1}. $x(t-\tau)$ is at the very best partially known and need to be extrapolated to get a good approximation of $x(t_{n+1})$. The following actions are taken :

- Choose an initial guess for the interpolant Π_n of $x(t-\tau)$ in $[t_n; t_{n+1}]$.
- Compute the solution $x(t_{n+1})$ using the interpolant Π_n and by stepping the solver
- Update the interpolant Π_n using the computed solution
- End if the interpolant has converged

The initial guess is usually the extrapolation of the interpolant of the previous step and the end criterion of convergence can vary across cases.

A.6.4 . Pseudocode for DDE solver

Following the detailed explanation of the challenges posed by DDE, we present the pseudocode of the DDE solver implemented by Zivari-Piran and Enright [206] that is detailed in Algorithm 4, where the general outline of one integration step of a DDE is shown; the DDE solver is illustrated in Algorithm 3. For sake of simplicity, we suppose for the pseudocode a single time delay DDE since the general case does not differ from it.

Algorithm 3 Pseudocode for DDE solver

1: **Input :**
Vector field $f(t, x, x(t-\tau))$
Integration bound t_0, t_F
History function $\psi(t)$
Set of history function's discontinuities $\Lambda = \{\lambda_{-m}, \ldots, \lambda_0\}$.
2: Choose an initial $\mathrm{d}t$
3: Declare $t_n = t_0, t_{n+1} = t_0 + \mathrm{d}t$
4: Declare interpolated estimated solution $\hat{x}(t) = \psi(t)$ for $t < t_0$
5: **repeat**
6: Algorithm 1
7: **until** $t_{n+1} = t_F$

Algorithm 4 Pseudocode for one DDE numerical integration step

Input :
Vector field $f(t, x, x(t-\tau))$
Integration bound t_n, t_{n+1}
Interpolated estimated solution $\hat{x}(t)$ in $[t_0; t_n]$
Set of detected discontinuities $\Lambda = \{\lambda_{-m}, \ldots, \lambda_0, \ldots, \lambda_{r-1}\}$.
if $t_{n+1} - t_n > \min(\Lambda)$ **then**
  Declare the interpolant $\Pi_n = \hat{x}$ of $x(t-\tau)$ in $[t_n; t_{n+1}]$
  while the interpolant Π_n has not converged **do**
    Define $f_{\mathrm{ODE}}(t, x) = f(t, x, \Pi_n(t))$
    Step the solver $x(t_{n+1}) = ODESolve(f_{\mathrm{ODE}}, t_n, t_{n+1}, \hat{x}(t_n))$
    Update Π_n using the computed solution $x(t_{n+1})$.
  end while
else
  Define $f_{\mathrm{ODE}}(t, x) = f(t, x, \hat{x}(t-\tau))$
  Step the solver $x(t_{n+1}) = ODESolve(f_{\mathrm{ODE}}, t_n, t_{n+1}, \hat{x}(t_n))$
end if
Determine next time step t_{next} from solver
if step is accepted **then**
  Return updated $\hat{x}(t)$, next integration bounds t_{n+1}, t_{next} and Λ.
else
  Check for discontinuities in $[t_n; t_{n+1}]$ i.e
  $\lambda_r = \min\{\lambda > \lambda_{r-1} : \lambda$ is a root of odd multiplicity of $g_i(t, x(t)), i \leq r-1\}$
  where $g_i(t, x(t)) = t - \tau(t, x(t)) - \lambda_i$
  if a discontinuity is found, λ_{r+1} **then**
    Return same $\hat{x}(t)$, next integration bounds t_n, λ_{r+1} and $\Lambda \cup \{\lambda_r\}$.
  else
    Return same $\hat{x}(t)$, next integration bounds t_n, t_{next} and Λ.
  end if
end if
Output :
Interpolated estimated solution
Next integration bounds
Updated set of discontinuities

A.7 . Appendix for Time and State Dependent DDE

A.7.1 . Memory and time complexity of Neural SDDDE

Neural SDDDEs rely on an ODE solver, thus function evaluations are associated with the same computational cost as NODEs. However, Neural SDDDE has some extra constraints making the method more computationally involved. Hereafter, we compare the complexity of these two schemes; we define S as the number of stages in the Runge–Kutta (RK) scheme used for the time-integration, N the total number of integration steps, and d the state's dimension.

Memory complexity DDE integration necessitates keeping a record of all previous states in memory due to the presence of delayed terms. This is because at any given time t, the DDE solver must be able to accurately determine $x(t-\tau)$ through interpolation of the stored state history. This extra amount of extra memory needed depends on the solver used. For example, the additional memory required when using a RK solver for one trajectory is $O(SNd)$. The model's memory footprint is not affected by the number of delays.

Time complexity In comparison to NODE, the solution estimate $\widehat{x}$ needs to be evaluated for each delayed state argument, i.e., $x(t-\tau_i)$. The cost of evaluating the interpolant is small compared to the cost of computing its coefficients. Similarly, the time complexity is conditioned by the solver used. For example, for a RK scheme, the coefficient computation scales linearly with the number of stages. Hence, Neural SDDDE adds a time cost $O(SD\,d)$ compared to NODE for each vector field function evaluation.

A.7.2 . Training information

Table A.3 sums up the MLP architecture of each IVP model (i.e., NODE, ANODE and Neural SDDDE) for each dynamical system. ANODE has an arbitrary augmented state of dimension 10 except for the PDE that has 100. Neural Laplace's architecture is the default one taken from the official implementation for all systems. The learning rate and the number of epochs are the same for all models. The optimizer used is `AdaBelief` [204]. Table A.4 gives the number of parameters for each model. In all of our experiments, we used the Dormand–Prince 5(4) [41] solver across all of our models.

	Width	Depth	Activation	Epochs	lr
Time Dependent DDE	64	3	relu	2000	.001
State Dependent DDE	64	3	relu	1000	.001
Diffusion PDE DDE	128	3	relu	500	.0001

Table A.3 – Model and training hyperparameters

	NODE	ANODE	Neural DDE	Neural Laplace
Time Dependent DDE	8513	9815	8578	17194
State Dependent DDE	8513	9815	8642	17194
Diffusion PDE DDE	58852	84552	71653	17194

Table A.4 – Number of parameters for each DDE system

A.7.3 . Data generation parameters

We expose in Table A.5 the parameters used for each dataset generation. The start integration time is always $T_0 = 0$. T_F refers to the end time integration. `NUM_STEPS` equally spaced points are sampled in $[T_0, T_F]$. The specific delays `DELAYS` and the constant history function $\psi(t)$ function domain are given. Each training dataset comprises 256 data points and the test set of 32 data points. We used our own DDE solver to generate the data (Dormand–Prince 5(4) solver [41] was used.). We then double-checked and compared its validity with Julia's DDE solver. $\mathcal{U}$ refers to the uniform distribution. For example, Time Dependent DDE's constant history function value is uniformly sampled between 0.1 and 2.0.

	T_F	num_steps	delays	$\psi(t)$
Time Dependent DDE	20.0	200	$2\sin(t)$	$\mathcal{U}(0.1, 2.0)$
State Dependent DDE	10.0	150	$0.5\cos(x(t))$	$\mathcal{U}(0.1, 1.0)$
Diffusion PDE DDE	4.0	100	1.0	$\mathcal{U}(0.1, 4.0)$

Table A.5 – Dataset generation information

A.7.4 . History step function experiment hyperparameters

In table A.6 we give the parameters used for each experiment. Extrapolated $\psi(t)$ indicates the possible value of the constant history function. τ_{max} and c_0, c_1 are described in Section 4.3. For the Diffusion Delay PDE, as stated in Section 4.3.2, the other history step function is omitted.

	Extrapolated $\psi(t)$	τ_{max}	c_0	c_1
Time Dependent DDE	$\mathcal{U}(2.0, 3.0)$	3.0	0.1	3.0
State Dependent DDE	$\mathcal{U}(-1.0, 0.1)$	$1/2$	-1.0	1.0

Table A.6 – System specific values for each testing experiment

A.8 . Appendix for Neural DDEs with Learnable Delays for Partially Observed Dynamical Systems

A.8.1 . Cancelling out the noise term $F(x,t)$

One possibility is by applying the projector $\mathcal{P}$ we get rid of the fluctuation/noise term.

$$\frac{\partial}{\partial t}\mathcal{P}e^{t\mathcal{L}} = \mathcal{P}e^{t\mathcal{L}}\mathcal{P}\mathcal{L} + \int_0^t \mathcal{P}e^{s\mathcal{L}}\mathcal{P}\mathcal{L}e^{(t-s)\mathcal{Q}\mathcal{L}}\mathcal{Q}\mathcal{L}ds \tag{A.46}$$

since the $e^{t\mathcal{Q}}\mathcal{Q}\mathcal{L}$ and $\mathcal{P}$ live in orthogonal subspaces. Instead of learning your observable g you consider $\mathcal{P}g$.

The other option depends on information that is unavailable as it is orthogonal to the observed subspace. However, this term vanishes if the history of the observed subspace is known, and the orthogonal dynamics are dissipative [116].

A.8.2 . t-model derivation

In the case of the slowly decaying memory (i.e., *t-model*), we have the approximation :

$$e^{t\mathcal{Q}\mathcal{L}} \approx e^{t\mathcal{L}}. \tag{A.47}$$

By rewriting the memory term of the Mori–Zwanzig equation (Eq. **??**)

$$\int_0^t e^{(t-s)\mathcal{L}}\mathcal{P}\mathcal{L}e^{s\mathcal{Q}\mathcal{L}}\mathcal{Q}\mathcal{L}ds = \int_0^t \mathcal{L}e^{(t-s)\mathcal{L}}e^{s\mathcal{Q}\mathcal{L}}\mathcal{Q}\mathcal{L}ds - \int_0^t e^{(t-s)\mathcal{L}}e^{s\mathcal{Q}\mathcal{L}}\mathcal{Q}\mathcal{L}\mathcal{Q}\mathcal{L}ds.$$

where we used the commutation of $\mathcal{L}$ and $\mathcal{Q}\mathcal{L}$ with $e^{t\mathcal{L}}$ and $e^{s\mathcal{Q}\mathcal{L}}$, respectively. By using Equation A.47, which eliminates the s dependence of both integrands we get :

$$\int_0^t e^{(t-s)\mathcal{L}}\mathcal{P}\mathcal{L}e^{s\mathcal{Q}\mathcal{L}}\mathcal{Q}\mathcal{L}ds \approx te^{t\mathcal{L}}\mathcal{P}\mathcal{L}\mathcal{Q}\mathcal{L}.$$

Only time dependence remains of the memory integral. We refer to [203] for a more detailed discussion on the *t-model*.

A.8.3 . Learning the delays

As a reminder, Table 5.2 provides the number of delays used in each experiment.

For each experiment, we present randomly selected models and display the evolution of delays during training in Figure A.7. Empirically, across all our experiments, we observe that the delays converge to specific values. Notably,

if the system is periodic, these values do not match the system's period as observed in the Brusselator experiment. This makes sense, as incorporating such a delay would not provide any additional information to the NDDE model. For the KS system experiments, we see that the delays that are initialized close to each tend to spread out during the training phase in order to maximize the system's information diversity.

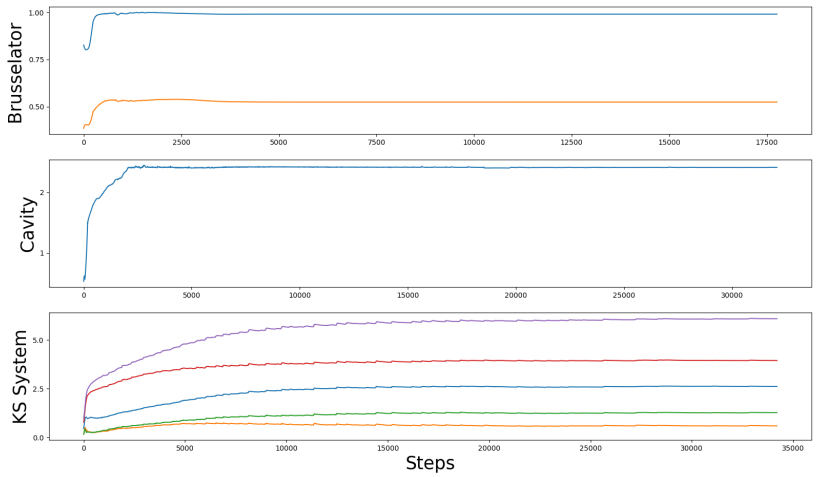

Figure A.7 – Delay's evolution during training for each experiment mentioned on each subplot's y-axis

A.8.4 . Neural IDE and Neural DDE Benchmark

Firstly, let us compare both Neural IDE and Neural DDE analytically where any function f_θ denotes a parameterized network :

$$\frac{dg}{\mathrm{d}t} = M_\theta(g(t)) - \int_0^t K_\theta(g(t-s), s)ds \tag{A.48}$$

$$\frac{dg}{\mathrm{d}t} = f_{\theta_1}(g(t)) + f_{\theta_2}(t, g(t), g(t-\tau_1), \ldots, g(t-\tau_n)) \tag{A.49}$$

The second term on the right-hand side of Equation A.48 is much more computationally involved than that of the second term on the right-hand side of Equation A.49. Indeed, Equation A.49 only needs 2 function evaluations to evaluate it's right-hand side(RHS). On the other hand, the number of function evaluation required to integrate Equation A.48 will scale as t grows in order to get a correct evaluation of the integral term. The original Neural IDE used a Monte Carlo integration with a fixed number of samples to approximate the integral term.

Computation Time

Figure A.8 compares the computation time of a forward pass between a Neural IDE and Neural DDE of the same size (roughly 500 parameters). We use a setup that includes an RK4 solver with a time step of $\mathrm{d}t = 0.1$, a batch size of 128, and a Neural DDE configured with 5 delays. Different state $g(t)$ dimensions are tested : $[5, 10, 50, 100]$. The upper integration bound is varied from $t = 0.2$ to $t = 2.0$ seconds.

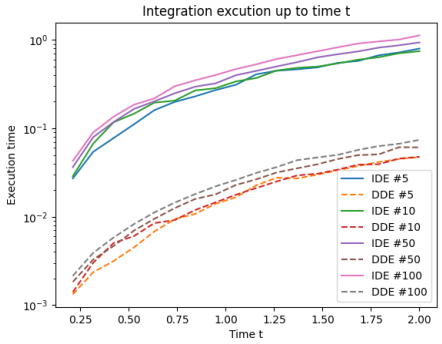

Figure A.8 – Time duration of forward pass averaged over 5 runs

This benchmark clearly shows how expensive Neural IDE is. NDDEs integration are at least an order of magnitude faster. (Please note that in Figure A.8, the notation "#i" refers to the number of features that our state $g(x, t)$ has).

Memory Consumption

We also performed memory profiling between the two methods and compared their memory needs for one forward pass. We use a setup that includes an RK4 solver with a time step of $\mathrm{d}t = 0.1$, a batch size of 128, and a Neural DDE configured with 5 delays. The dimension of the different state $g(t)$ is 100, and the upper integration bound is varied from $t = 0.2$ to $t = 2.0$ seconds.

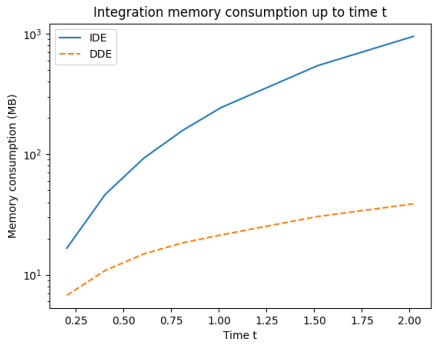

Figure A.9 – Memory consumption of forward pass averaged over 5 runs

For extremely small tasks, the memory requirements of IDEs are excessive, as depicted in Figure A.9. Despite this, we attempted to implement the Neural IDE method in our experiments. Unfortunately, during training, we encountered memory issues, even with a simple problem like the Brusselator. The reason behind this is that the integral component needs to be recalculated at each integration step, causing scalability problems when the integration duration is extensive.

A.8.5 . Proof of Proposition 5.2.1

Let us start by stating Takens' theorem as expressed by [130, 173] :

Theorem A.8.1. *Takens' embedding theorem Let M be compact. There is an open dense subset $\mathcal{D}$ of $Diff(M) \times C^k(M, \mathbb{R})$ with the property that the Takens map*

$$h : M \to \mathbb{R}^{2m+1}$$

given by $h(x) = (g(x), g(\phi(x)), g(\phi \circ \phi(x)), ...g(\phi^{2m}(x)))$ is an embedding of C^k manifolds, when $(\phi, g) \in \mathcal{D}$.

Here, ϕ stands for the operator that advances the dynamical system by a time step τ, i.e. that sends $x(t)$ to $x(t+\tau)$, and g is the observable operator, that sends a full state $x(t)$ to actual observables $g(x(t)) =: g(t)$. Variants of this Theorem, e.g. [159], include the consideration of any set of different delays τ_i instead of uniformly spaces ones. The representation $h(x(t)) = (g(x(t)), g(x(t-\tau)), g(x(t-2\tau)), ...g(x(t-2m\tau))$ then becomes $h(x(t)) = (g(x(t)), g(x(t-\tau_1)), g(x(t-\tau_2)), ...g(x(t-\tau_{2m}))$. In the proof of Takens' theorem, m is the intrinsic dimension of the dynamical system, i.e. the one of the manifold M.

Now, given the full state x that follows the dynamics :

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = G(x), \quad x(0) = x_0 \tag{A.50}$$

we use the chain rule on the observable g :

$$\frac{\mathrm{d}g}{\mathrm{d}t} = g'(x(t))G(x) \tag{A.51}$$

By applying the inverse of the delay coordinate map h^{-1} from theorem A.8.1, which is invertible from its image as it is an embedding, we show that g's dynamics possesses a DDE structure.

$$\frac{\mathrm{d}g}{\mathrm{d}t} = (g' \times G) \circ h^{-1}(g(t), g(t-\tau_1), \ldots, g(t-\tau_n)) \tag{A.52}$$

The proof is completed by choosing $f = (g' \times G) \circ h^{-1} - M$ where M is obtained by the Mori–Zwanzig formalism (Equation **??**).

Remark A.8.2. *A theoretical development is dedicated to Takens' theorem in Section 3.3.*

A.8.6 . The importance of relevant delays

Let us consider a dynamical system evolving on a compact smooth manifold $\mathcal{S} \subset \mathbb{R}^d$, assumed to be an attractor. Let us consider a C^2 observable function $g : \mathcal{S} \to \mathbb{R}$.

Takens' theorem [173] rigorously discusses conditions under which a delay vector of a scalar-valued observable $(g(x(t)), g(x(t-\tau)), \ldots, g(x(t-p\tau))), p \in \mathbb{N}$ defines an embedding, a smooth diffeomorphism onto its image. It guarantees a topological equivalence between the original dynamical system and the one constructed from the memory of the observable. The dynamics of the system can then be reformulated on the set $(g(x(t)), g(x(t-\tau)), \ldots, g(x(t-p\tau)))$.

The Takens' theorem, later extended by [159], establishes a sufficient condition but does not provide information about the time delay τ. From a mathematical viewpoint, the delay could be arbitrary, besides some point wise values excluded by the theorem. In practice however, its value is instrumental in a successful embedding. If too small, entries of the delay vector data are too similar; if too large, the entries tend to be completely uncorrelated and cannot be numerically linked to a consistent dynamical system.

We here illustrate the impact of suitable delays in the relevance of the information available to inform the future evolution of the observable. We consider a simple 2-delay dynamical system described by :

$$\begin{aligned} g(t+\Delta t) &= \cos(g(t-\tau_1))\ \sin(g(t-\tau_2)) - \alpha\,\mathrm{sinc}(3\,g(t-\tau_1)) + \alpha\ \cos(g(t-\tau_2)), \\ g(t<0) &= \psi(t), \end{aligned}$$

with $\alpha = 0.2$, $\tau_1 = p_1^\star \, \Delta t$, $\tau_2 = p_2^\star \, \Delta t$, $p_1^\star = 125$ and $p_2^\star = 200$.

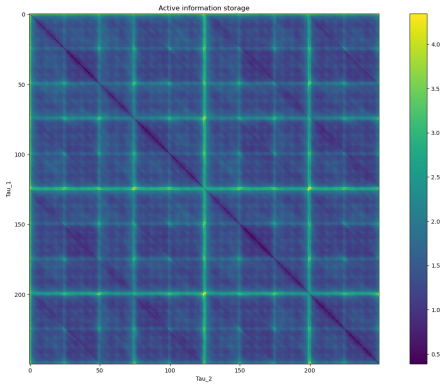

Figure A.10 – $\{\tau_1 = p_1 \, \Delta t, \tau_2 = p_2 \, \Delta t\}$-map of Delayed Mutual Information, $I\left(\left(g(t-\tau_1), g(t-\tau_2)\right), g(t)\right)$. The maximum is exhibited at $(125, 200)$ and $(200, 125)$, in accordance with $p_1^\star = 125$, $p_2^\star = 200$.

The relevance of the delays $\{\tau_1, \tau_2\}$ for informing $g(t + \Delta t)$ is assessed in terms of the mutual information $I\left(\left(g(t-\tau_1), g(t-\tau_2)\right), g(t)\right)$ and shown in Fig. A.10 as a 2-D map in terms of p_1 and p_2. The map is symmetric, consistently with the symmetry of the mutual information, $I\left(\left(g(t-\tau_1), g(t-\tau_2)\right), g(t)\right) = I\left(\left(g(t-\tau_2), g(t-\tau_1)\right), g(t)\right)$.

It can be seen that the amount of information shared between the current observation and a delay vector of the observable widely varies with the delays. The ability of the present Neural DDE method to learn the delays, in addition to the model f_θ, is thus key to its performance and wide applicability.

A.8.7 . Additional experiments

We present the Shallow Water equation dataset, available in the PDEBench-mark suite [172]. We put ourselves in the highly restrictive partially-observable setting by randomly sampling 4 points on the spatial grid and fit its dynamics. Finally, Table A.7 presents a summary of the test loss from these experiments along with the number of parameters for each model in Table A.8. Figure A.11 illustrates the performance of each model on the Shallow Water test set. Excluding NODE, all models yield satisfactory outcomes. However, when evaluating based on Mean Squared Error (MSE), NDDEs yielded better performance than the presented baselines.

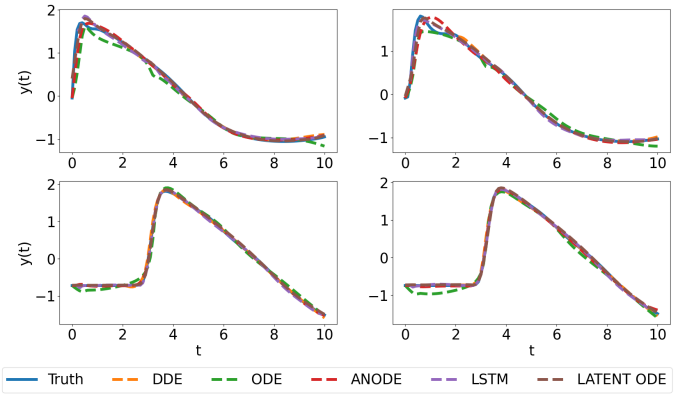

Figure A.11 – Random test sampled of Shallow Water dataset

	Shallow Water
LSTM	0.0031 ± 0.001
NODE	0.0460 ± 0.035
ANODE	0.0040 ± 0.001
Latent ODE	0.0058 ± 0.001
NDDE	$\mathbf{0.0010 \pm 0.0001}$

Table A.7 – Shallow Water test loss experiments averaged over 5 runs

	LSTM	NODE	ANODE	Latent ODE	NDDE
# Parameters	2512	2404	2534	5853	2662

Table A.8 – Number of parameters for Shallow Water experiment

A.8.8 . Training hyperparameters

To train our models we progressively feed them longer trajectory chunks if the patience hyperparameter is exceeded; this is done until the desired trajectory length is attained. Table A.9 displays the patience hyperparameter and how much trajectory length was given initially. Table A.10 refers to the number of parameters of each model. The loss function used across all experiments is the MSE loss, and we employ the Adam optimizer with a weight decay of 10^{-7}. Table A.15 provides the initial and final learning rates (lr_i, lr_f) for each experiment, which are associated with the scheduler. The scheduler is a StepLR

scheduler with a gamma factor ($\gamma = \exp\left\{\frac{\log \frac{lr_f}{lr_i}}{N}\right\}$, where N is the trajectory's length). The scheduler adjusts the learning rate as the trajectory length increases, allowing training to start with the initial learning rate lr_i and gradually decrease to the final learning rate lr_f. TableA.12 shows the width and depth of the MLPs for NODE, ANODE, and NDDE across all experiments. Additionally, we provide the hidden size and number of layers for the LSTM model in Table A.14. Finally, Table A.13 summarizes the Latent ODE hyperparameters, where the vector field f_θ (defined in the introduction) is an MLP with the width and depth specified in the second and third columns, the latent size of z_0 in the last column, and the RNN's hidden size in the fourth column. If some models has fewer parameters compared to others it is that we found that they provided better results with less. ANODE's augmented state dimension matches that of the number of delays used by NDDE displayed in Table 5.2. Due to the inherent different nature of each model, they provide output of different length and do not necessarily start at the same initial time t_0 : Table A.11 provides a MSE comparison along the common trajectory predicted by all models. Compared to Table 5.1, the results barely differ.

	KS	Cavity	Brusselator
Length Start	15%	50%	25%
Patience	40	50	20

Table A.9 – How long is the trajectory chunks given at first and the patience used for each experiment

	LSTM	NODE	ANODE	Latent ODE	NDDE
Brusselator	1764	3265	3395	3666	3331
KS	18130	9029	11609	8118	19343
Cavity	2234	2209	2274	3642	2242

Table A.10 – Number of parameters for each experiment

	Brusselator	KS	Cavity
LSTM	$\mathbf{0.0051 \pm 0.0031}$	0.77 ± 0.061	0.75 ± 0.51
NODE	0.75 ± 0.0014	0.71 ± 0.10	0.96 ± 0.0001
ANODE	$\mathbf{0.0050 \pm 0.0050}$	0.53 ± 0.052	0.65 ± 0.021
Latent ODE	0.014 ± 0.0076	0.43 ± 0.07	0.25 ± 0.14
NDDE	0.011 ± 0.0076	$\mathbf{0.30 \pm 0.032}$	$\mathbf{0.13 \pm 0.0081}$

Table A.11 – Test loss experiments averaged over 5 runs over common trajectory predictions

	NODE/ANODE/NDDE	
	Width	Depth
Brusselator	32	4
KS	64	3
Cavity	32	3
Shallow Water	32	3

Table A.12 – MLP width and depth for each experiment

Experiment	Width Size	Depth	Hidden Size	Latent Size
Brusselator	16	3	16	16
KS	32	3	16	16
Cavity	16	3	8	8
Shallow Water	32	3	8	8

Table A.13 – Configuration parameters for each experiment

Experiment	Hidden Size	Number of Layers
Brusselator	5	10
KS	25	5
Shallow Water	6	10
Cavity	7	7

Table A.14 – Hidden size and number of layers for each experiment for LSTM model

Experiment	lr_i	lr_f
Brusselator	0.001	0.0001
Cavity	0.005	0.00005
KS	0.01	0.0001
Shallow Water	0.001	0.00001

Table A.15 – Initial and final learning rates for each experiment

A.9 . Appendix for Non-Markovian closure or correction modelling for dynamical systems

A.9.1 . POD Galerkin

The typical approach of POD Galerkin ROMs is to project the high-fidelity dynamics onto a low-dimensional space using time-invariant spatial features [71, 96, 129, 16]. Through the use of Proper Orthogonal Decomposition (POD), also known as Principal Component Analysis (PCA), a high-dimensional dataset can be compressed into a significantly smaller one, awhile preserving most of its information. Practically, obtaining the POD involves decomposing a snapshot matrix $U \in \mathbb{R}^{m \times n}$ where m is the number of data points and n the system's state dimension (for fluid mechanics problems this would be its number of degrees of freedom). Such a decomposition is done with Singular Value Decompostion (SVD) :

$$U = V \Sigma W^T$$

where $V \in \mathbb{R}^{n \times m}$, $W \in \mathbb{R}^{m \times n}$ are the orthogonal matrices representing the left and right singular vectors and $\Sigma \in \mathbb{R}^{m \times m}$ is a diagonal matrix containing the singular values σ_i. V's columns represent the spatial modes of the POD and define the POD basis. This basis allows to do dimensionality reduction because, for any $r < n$, the subspace spanned by the first r columns, $V_r = \{v_1 | v_2 | \dots | v_r\}$ optimally approximates our snapshot matrix U in the sense that it minimizes the following reconstruction error E_r :

$$E_r = \|U - V_r V_r^T U\|_2, \quad \forall r \in [1, \dots, m] \tag{A.53}$$

Such an error can be related to the sum of the discarded Σ's singular values : $E_r = \sum_{i=r+1}^{n} \sigma_i^2$. The information (ie variance) captured by the first r POD modes can be quantified with :

$$R(r) = \frac{\sum_{i=r+1}^{n} \sigma_i^2}{\sum_{i=1}^{n} \sigma_i^2} \tag{A.54}$$

Given the operator $P = V_r V_r^T$ that projects onto the invariant subspace spanned by V_r and Q its orthogonal complement, the full state vector $\mathbf{u}(\mathbf{x}, t)$ can be decomposed as :

$$\mathbf{u}(\mathbf{x}, t) = P\mathbf{u}(\mathbf{x}, t) + Q\mathbf{u}(\mathbf{x}, t) \tag{A.55}$$

For dimensionality reduction, the key idea is to select a r such that enough variance has been accounted for in order to approximate the full system state $\mathbf{u}(t)$ as :

$$\mathbf{u}(\mathbf{x}, t) \approx V_r \bar{\mathbf{u}}(t) \tag{A.56}$$

where $\bar{\mathbf{u}}(t) = V_r^T \mathbf{u}(\mathbf{x}, t)$ (i.e. the reduced operator is $\mathcal{A}(x) = V_r^T x$) contains the temporal coefficients associated to V_r's spatial modes. Dealing with $\mathbf{a}(t)$ instead of the full system's state $\mathbf{u}(\mathbf{x}, t)$ with the requirement that $r << n$, sets us in the desired model reduction framework. Transitioning within this reduced framework, the Galerkin method is aptly applied to derive the dynamics of the reduced state $\bar{\mathbf{u}}(t)$. By projecting the full order model onto the POD basis we get :

$$\begin{aligned} \frac{\mathrm{d}\bar{\mathbf{u}}}{\mathrm{d}t} &\approx V_r^T f_{ROM}(t, V_r \mathbf{a}(t)) \\ \bar{\mathbf{u}}(0) &= V_r^T \mathbf{u}^0(\mathbf{x}) \end{aligned} \tag{A.57}$$

where f_{ROM} comprises the ROM operators of the studied system. With such a formulation the problem dimensionality has been reduced. Hereafter, two options are possible in order to solve the problems' dynamics, one more of interest to us. First, for each vector field evaluation f_{ROM} you must compute the reconstructed solution $\mathbf{u}(\mathbf{x}, t) \approx V_r^T \bar{\mathbf{u}}(t)$ which can be prohibitively expensive. In contrast, one can compute beforehand the reduced model's operators and this is done in our case to leverage its computational gain. Such a proposition is detailed and used with the Navier Stokes equation [116].

A.9.2 . Hyperparameter and training information

To train our models, we progressively provide them with longer trajectory chunks whenever the patience hyperparameter is exceeded, continuing this process until the desired trajectory length is achieved. Table A.16 presents the patience hyperparameter and the initial trajectory lengths for the KS system and Kolmogorov flow experiments. We utilized the Adam optimizer with a weight decay of 10^{-5} and a 'StepLR' scheduler with a gamma factor of $\gamma = 0.98$. All trajectories were integrated using an RK4 solver. Table A.16 also lists the initial learning rates (l_r) for all experiments.

	length start	patience	l_r
4 modes (KS)	16%	50	0.001
8 modes (KS)	16%	50	0.01
10 modes (KS)	16%	50	0.0005
Linear model (KS)	7%	30	0.001
Linear model (Kolmogorov)	50%	10	0.005

Table A.16 – Initial trajectory chunk lengths, patience values, and learning rates used for the KS system and Kolmogorov flow experiments

For the POD Galerkin ROM experiments, the ODE and DDE closure models were implemented as MLPs with a width of 128, depth of 4, and `SiLU` activation, resulting in approximately 53k parameters. We opted to use a single delay for the POD Galerkin experiments, as in subsection 6.4.1 empirically shows that utilizing 1, 2, or 3 delays does not enhance performance. For the linear model in the KS experiment, we designed the correction network as an MLP with a width of 84, depth of 4, and `SiLU` activation, to match the size of the POD Galerkin network. Since CD-ROM, ODE and DDE closure/correction terms differ by their formulation we tried to compare these models as fairly as possible by matching the number of parameters used, and using the same type of model architecture for E_θ and R_θ (please refer to paragraph 6.3.1).

For the 2D Kolmogorov flow, we chose a U-net architecture [150] for ODE and DDE correction terms, as these models typically perform well in computer vision tasks. The U-net architecture is further more motivated because it is often associated to leverage multiscale features whom are necessary to make accurate predictions [199]. The ODE and DDE correction term consists of roughly 7M parameters. The DDE correction has also 1 delay since increasing the number of delays didn't improve performance. As in the KS experiments, CD-ROM, ODE and DDE closure/correction terms differ by their formulation we tried to compare these models as fairly as possible by matching the number of parameters used, and using the same type of model architecture for E_θ and R_θ.

Bibliographie

[1] Henry DI Abarbanel, TA Carroll, LM Pecora, JJ Sidorowich, and L Sh Tsimring. Predicting physical variables in time-delay embedding. *Physical Review E*, 49(3) :1840, 1994.

[2] Yves Achdou, Olivier Bokanowski, and Tony Lelièvre. Partial differential equations in finance. *The Encyclopedia of Financial Models*, 2, 2012.

[3] Shady E. Ahmed, Suraj Pawar, Omer San, Adil Rasheed, Traian Iliescu, and Bernd R. Noack. On closures for reduced order models—a spectrum of first-principle to machine-learned avenues. *Physics of Fluids*, 33(9), September 2021. ISSN 1089-7666. doi : $10.1063/5.0061577$. URL http://dx.doi.org/10.1063/5.0061577.

[4] Julien Arino, Lin Wang, and Gail SK Wolkowicz. An alternative formulation for a delayed logistic equation. *Journal of theoretical biology*, 241(1) : 109–119, 2006.

[5] O. Arino, M.L. Hbid, and E.A. Dads. *Delay Differential Equations and Applications : Proceedings of the NATO Advanced Study Institute held in Marrakech, Morocco, 9-21 September 2002*. Nato Science Series II :. Springer Netherlands, 2009. ISBN 9789048104079.

[6] O. Arino, M.L. Hbid, and E.A. Dads. *Delay Differential Equations and Applications : Proceedings of the NATO Advanced Study Institute held in Marrakech, Morocco, 9-21 September 2002*. Nato Science Series II :. Springer Netherlands, 2009. ISBN 9789048104079.

[7] Saba Arshad and Gon-Woo Kim. Role of deep learning in loop closure detection for visual and lidar slam : A survey. *Sensors*, 21(4) :1243, 2021.

[8] H.T. Banks, J.E. Banks, Riccardo Bommarco, A.N. Laubmeier, N.J. Myers, Maj Rundlöf, and Kristen Tillman. Modeling bumble bee population dynamics with delay differential equations. *Ecological Modelling*, 351 :14–23, 2017. ISSN 0304-3800. doi : $\mathrm{https://doi.org/10.1016/j.ecolmodel.2017.02.011}$. URL https://www.sciencedirect.com/science/article/pii/S0304380017301606.

[9] Andrea Beck, David Flad, and Claus-Dieter Munz. Deep neural networks for data-driven les closure models. *Journal of Computational Physics*, 398 :108910, 2019.

[10] B. Begiashvili, N. Groun, J. Garicano-Mena, S. Le Clainche, and E. Valero. Data-driven modal decomposition methods as feature detection techniques for flow problems : A critical assessment. *Physics of Fluids*, 35(4), April 2023. ISSN 1089-7666. doi : 10.1063/5.0142102. URL http://dx.doi.org/10.1063/5.0142102.

[11] Jens Behrmann, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks, 2018.

[12] Bellen and Zennaro. *Numerical Methods for Delay Differential Equations*. Oxford University Press, Oxford, 2003.

[13] Richard Ernest Bellman and Kenneth L. Cooke. *Differential-Difference Equations*. RAND Corporation, Santa Monica, CA, 1963.

[14] Belousov. *A periodic reaction and its mechanism,in Collection of short papers on radiation medicine for 1958*. , Med. Publ. Moscow, 1959.

[15] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 97816055851 61. doi : 10.1145/1553374.1553380. URL https://doi.org/10.1145/1553374.1553380.

[16] M. Bergmann, C.-H. Bruneau, and A. Iollo. Enablers for robust pod models. *Journal of Computational Physics*, 228(2) :516–538, 2009. ISSN 0021-9991. doi : https://doi.org/10.1016/j.jcp.2008.09.024. URL https://www.sciencedirect.com/science/article/pii/S002199910800510X.

[17] David Bernstein. Optimal prediction of burgers's equation. *Multiscale Modeling & Simulation*, 6(1) :27–52, 2007.

[18] Lorenz T Biegler, Omar Ghattas, Matthias Heinkenschloss, and Bart van Bloemen Waanders. Large-scale pde-constrained optimization : an introduction. In *Large-scale PDE-constrained optimization*, pages 3–13. Springer, 2003.

[19] Felix Bloch. The principle of nuclear induction. *Science*, 118(3068) :425–430, 1953.

[20] Hermann Brunner. *Volterra integral equations : an introduction to theory and applications*, volume 30. Cambridge University Press, 2017.

[21] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15) : 3932–3937, March 2016. ISSN 1091-6490. doi : 10.1073/pnas.1517384113. URL http://dx.doi.org/10.1073/pnas.1517384113.

[22] Steven L. Brunton, Marko Budišić, Eurika Kaiser, and J. Nathan Kutz. Modern koopman theory for dynamical systems, 2021. URL https://arxiv.org/abs/2102.12086.

[23] John Charles Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2016.

[24] Jared L. Callaham, Steven L. Brunton, and Jean-Christophe Loiseau. On the role of nonlinear correlations in reduced-order modelling. *Journal of Fluid Mechanics*, 938, March 2022. ISSN 1469-7645. doi : 10.1017/jfm.2021.994. URL http://dx.doi.org/10.1017/jfm.2021.994.

[25] Jonathan Calver and W.H. Enright. Numerical methods for computing sensitivities for odes and ddes. *Numerical Algorithms*, 74, 04 2017. doi : 10.1007/s11075-016-0188-6.

[26] Francesco Camastra and Antonino Staiano. Intrinsic dimension estimation : Advances and open problems. *Information Sciences*, 328 :26–41, 2016.

[27] L. A. V. Carvalho and Kenneth L. Cooke. A nonlinear equation with piecewise continuous argument. *Differential and Integral Equations*, 1(3) : 359 – 367, 1988. doi : 10.57262/die/1371669564. URL https://doi.org/10.57262/die/1371669564.

[28] Martin Casdagli, Stephen Eubank, J Doyne Farmer, and John Gibson. State space reconstruction in the presence of noise. *Physica D : Nonlinear Phenomena*, 51(1-3) :52–98, 1991.

[29] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations, 2019. URL https://arxiv.org/abs/1806.07366.

[30] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations, 2019. URL https://arxiv.org/abs/1806.07366.

[31] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning

phrase representations using rnn encoder-decoder for statistical machine translation, 2014. URL https://arxiv.org/abs/1406.1078.

[32] Alexandre J. Chorin and Panagiotis Stinis. Problem reduction, renormalization, and memory, 2005.

[33] Alexandre J Chorin, Ole H Hald, and Raz Kupferman. Optimal prediction with memory. *Physica D : Nonlinear Phenomena*, 166(3-4) :239–257, 2002.

[34] Kenneth L Cooke and Joseph Wiener. A survey of differential equations with piecewise continuous arguments. In *Delay Differential Equations and Dynamical Systems : Proceedings of a Conference in honor of Kenneth Cooke held in Claremont, California, Jan. 13–16, 1990*, pages 1–15. Springer, 2006.

[35] Ian Cooper, Argha Mondal, and Chris G Antonopoulos. A sir model assumption for the spread of covid-19 in different communities. *Chaos, Solitons & Fractals*, 139 :110057, 2020.

[36] E Ait Dads, B Es-sebbar, and L Lhachimi. Almost periodicity in time-dependent and state-dependent delay differential equations. *Mediterranean Journal of Mathematics*, 19(6) :259, 2022.

[37] Marco David and Florian Méhats. Symplectic learning for hamiltonian neural networks. *Journal of Computational Physics*, 494 :112495, December 2023. ISSN 0021-9991. doi : 10.1016/j.jcp.2023.112495. URL http://dx.doi.org/10.1016/j.jcp.2023.112495.

[38] Shaan A. Desai, Marios Mattheakis, David Sondak, Pavlos Protopapas, and Stephen J. Roberts. Port-hamiltonian neural networks for learning explicit time-dependent dynamical systems. *Physical Review E*, 104(3), September 2021. ISSN 2470-0053. doi : 10.1103/physreve.104.034312. URL http://dx.doi.org/10.1103/PhysRevE.104.034312.

[39] Ethan R. Deyle and George Sugihara. Generalized theorems for nonlinear state space reconstruction. *PLOS ONE*, 6(3) :1–8, 03 2011. doi : 10.1371/journal.pone.0018295. URL https://doi.org/10.1371/journal.pone.0018295.

[40] Felix Dietrich, Thomas N Thiem, and Ioannis G Kevrekidis. On the koopman operator of algorithms. *SIAM Journal on Applied Dynamical Systems*, 19(2) :860–885, 2020.

[41] John R Dormand and Peter J Prince. A family of embedded Runge-Kutta formulae. *Journal of computational and applied mathematics*, 6(1) :19–26, 1980.

[42] Rodney D. Driver. Existence and stability of solutions of a delay-differential system. *Archive for Rational Mechanics and Analysis*, 10(1) : 401–426, Jan 1962. ISSN 1432-0673.

[43] Thai Duong and Nikolay Atanasov. Hamiltonian-based Neural ODE Networks on the SE(3) Manifold For Dynamics Learning and Control. In *Proceedings of Robotics : Science and Systems*, Virtual, July 2021. doi : 10.15607/RSS.2021.XVII.086.

[44] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes, 2019.

[45] Irving R Epstein. Differential delay equations in chemical kinetics : Some simple linear model systems. *The Journal of Chemical Physics*, 92(3) :1702–1712, 1990.

[46] Swinda K. J. Falkena, Courtney Quinn, Jan Sieber, and Henk A. Dijkstra. A delay equation model for the atlantic multidecadal oscillation. *Proceedings of the Royal Society A : Mathematical, Physical and Engineering Sciences*, 477(2246) :20200659, February 2021. ISSN 1471-2946. doi : 10.1098/rspa.2020.0659. URL http://dx.doi.org/10.1098/rspa.2020.0659.

[47] J. Doyne Farmer and John J. Sidorowich. Predicting chaotic time series. *Phys. Rev. Lett.*, 59 :845–848, Aug 1987. doi : 10.1103/PhysRevLett.59.845. URL https://link.aps.org/doi/10.1103/PhysRevLett.59.845.

[48] Andrew M. Fraser and Harry L. Swinney. Independent coordinates for strange attractors from mutual information. *Phys. Rev. A*, 33 :1134–1140, Feb 1986. doi : 10.1103/PhysRevA.33.1134. URL https://link.aps.org/doi/10.1103/PhysRevA.33.1134.

[49] Kai Fukami, Takaaki Murata, Kai Zhang, and Koji Fukagata. Sparse identification of nonlinear dynamics with low-dimensionalized flow representations. *Journal of Fluid Mechanics*, 926, September 2021. ISSN 1469-7645. doi : 10.1017/jfm.2021.697. URL http://dx.doi.org/10.1017/jfm.2021.697.

[50] Roshini Samanthi Gallage. *Approximation of continuously distributed delay differential equations*. Southern Illinois University at Carbondale, 2017.

[51] Han Gao, Jian-Xun Wang, and Matthew J. Zahr. Non-intrusive model reduction of large-scale, nonlinear dynamical systems using deep learning. *Physica D : Nonlinear Phenomena*, 412 :132614, November 2020. ISSN

0167-2789. doi : 10.1016/j.physd.2020.132614. URL http://dx.doi.org/10.1016/j.physd.2020.132614.

[52] Michael Ghil, Ilya Zaliapin, and Sylvester Thompson. A delay differential model of ENSO variability : parametric instability and the distribution of extremes. *Nonlinear Processes in Geophysics*, 15(3) :417–433, 2008.

[53] M Goldman and L Shen. Spin-spin relaxation in la f 3. *Physical Review*, 144(1) :321, 1966.

[54] Pablo Gómez, Håvard Hem Toftevaag, and Gabriele Meoni. torchquad : Numerical integration in arbitrary dimensions with pytorch. *J. Open Source Softw.*, 6 :3439, 2021. URL https://api.semanticscholar.org/CorpusID:237409134.

[55] Ayoub Gouasmi, Eric J Parish, and Karthik Duraisamy. A priori estimation of memory effects in reduced-order models of nonlinear systems using the mori–zwanzig formalism. *Proceedings of the Royal Society A : Mathematical, Physical and Engineering Sciences*, 473(2205) :20170385, 2017.

[56] Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord : Free-form continuous dynamics for scalable reversible generative models, 2018.

[57] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. *Advances in neural information processing systems*, 32, 2019.

[58] Andreas Griewank and Andrea Walther. Algorithm 799 : revolve : an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Transactions on Mathematical Software (TOMS)*, 26(1) :19–45, 2000.

[59] Sheng Guo, Weilin Huang, Haozhi Zhang, Chenfan Zhuang, Dengke Dong, Matthew R. Scott, and Dinglong Huang. Curriculumnet : Weakly supervised learning from large-scale web images, 2018. URL https://arxiv.org/abs/1808.01097.

[60] Abhinav Gupta and Pierre FJ Lermusiaux. Neural closure models for dynamical systems. *Proceedings of the Royal Society A*, 477(2252) :20201004, 2021.

[61] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1) :014004, December 2017. ISSN 1361-6420. doi : 10.1088/1361-6420/aa9a90. URL http://dx.doi.org/10.1088/1361-6420/aa9a90.

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

[63] Jan Hagnberger, Marimuthu Kalimuthu, Daniel Musekamp, and Mathias Niepert. Vectorized conditional neural fields : A framework for solving time-dependent parametric partial differential equations, 2024. URL https://arxiv.org/abs/2406.03919.

[64] Jack K Hale. A stability theorem for functional-differential equations. *Proceedings of the National Academy of Sciences*, 50(5) :942–946, 1963.

[65] LD Hall. Nuclear magnetic resonance. *Advances in carbohydrate chemistry*, 19 :51–93, 1964.

[66] Samir Hamdi, William E Schiesser, and Graham W Griffiths. Method of lines. *Scholarpedia*, 2(7) :2859, 2007.

[67] Zhongkai Hao, Jiachen Yao, Chang Su, Hang Su, Ziao Wang, Fanzhi Lu, Zeyu Xia, Yichi Zhang, Songming Liu, Lu Lu, et al. Pinnacle : A comprehensive benchmark of physics-informed neural networks for solving pdes. *arXiv preprint arXiv :2306.08827*, 2023.

[68] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

[69] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8) :1735–1780, 1997.

[70] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8) :1735–1780, 1997.

[71] Philip Holmes. *Turbulence, coherent structures, dynamical systems and symmetry*. Cambridge university press, 2012.

[72] Samuel I Holt, Zhaozhi Qian, and Mihaela van der Schaar. Neural Laplace : Learning diverse classes of differential equations in the Laplace domain. In *International Conference on Machine Learning*, pages 8811–8832. PMLR, 2022.

[73] Denis J Evans and Gary P Morriss. *Statistical mechanics of nonequilbrium liquids*. ANU Press, 2007.

[74] MI Jordan. Serial order : a parallel distributed processing approach. technical report, june 1985-march 1986. Technical report, California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science, 1986.

[75] Gerhard Jung, Martin Hanke, and Friederike Schmid. Iterative reconstruction of memory kernels. *Journal of chemical theory and computation*, 13(6) :2481–2488, 2017.

[76] Mason Kamb, Eurika Kaiser, Steven L Brunton, and J Nathan Kutz. Time-delay observables for koopman : Theory and applications. *SIAM Journal on Applied Dynamical Systems*, 19(2) :886–917, 2020.

[77] Artur Karimov, Erivelton G Nepomuceno, Aleksandra Tutueva, and Denis Butusov. Algebraic method for the reconstruction of partially observed nonlinear systems using differential and integral embedding. *Mathematics*, 8(2) :300, 2020.

[78] Andrew Keane, Bernd Krauskopf, and Henk A Dijkstra. The effect of state dependence in a delay differential equation model for the El Niño southern oscillation. *Philosophical Transactions of the Royal Society A*, 377 (2153) :20180121, 2019.

[79] Jacob Kelly, Jesse Bettencourt, Matthew J Johnson, and David K Duvenaud. Learning differential equations that are easy to solve. *Advances in Neural Information Processing Systems*, 33 :4370–4380, 2020.

[80] Patrick Kidger. On neural differential equations, 2022. URL https://arxiv.org/abs/2202.02435.

[81] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series, 2020. URL https://arxiv.org/abs/2005.08926.

[82] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. In *Advances in Neural Information Processing Systems*, volume 33, pages 6696–6707, 2020.

[83] H_S Kim, R Eykholt, and JD Salas. Nonlinear dynamics, delay times, and embedding windows. *Physica D : Nonlinear Phenomena*, 127(1-2) :48–60, 1999.

[84] Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21), 2021. ISSN 0027-8424. doi : 10.1073/pnas.2101784118. URL https://www.pnas.org/content/118/21/e2101784118.

[85] Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21), May 2021. ISSN 1091-6490. doi : $10.1073/\mathrm{pnas}.2101784118$. URL http://dx.doi.org/10.1073/pnas.2101784118.

[86] B. O. Koopman. Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences*, 17(5) :315–318, 1931. doi : $10.1073/\mathrm{pnas}.17.5.315$. URL https://www.pnas.org/doi/abs/10.1073/pnas.17.5.315.

[87] Bernard O Koopman. Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences*, 17(5) :315–318, 1931.

[88] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator : Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89) :1–97, 2023.

[89] K Hauke Kraemer, Maximilian Gelbrecht, Induja Pavithran, RI Sujith, and Norbert Marwan. Optimal state space reconstruction via monte carlo decision tree search. *Nonlinear Dynamics*, 108(2) :1525–1545, 2022.

[90] Kai-Hauke Krämer, George Datseris, Jürgen Kurths, Istvan Z Kiss, Jorge L Ocampo-Espindola, and Norbert Marwan. A unified and automated approach to attractor reconstruction. *New Journal of Physics*, 23(3) :033017, 2021.

[91] Nikolai A Kudryashov. Exact solutions of the generalized kuramoto–sivashinsky equation. *Physics Letters A*, 147(5-6) :287–291, 1990.

[92] Marius Kurz and Andrea Beck. A machine learning framework for les closure terms. *arXiv preprint arXiv :2010.03030*, 2020.

[93] Marius Kurz and Andrea Beck. A machine learning framework for les closure terms. *ETNA - Electronic Transactions on Numerical Analysis*, 56 : 117–137, 2022. ISSN 1068-9613.

[94] J. Nathan Kutz, Xing Fu, and Steven L. Brunton. Multi-resolution dynamic mode decomposition, 2015.

[95] Vangipuram Lakshmikantham. *Theory of integro-differential equations*, volume 1. CRC press, 1995.

[96] Toni Lassila, Andrea Manzoni, Alfio Quarteroni, and Gianluigi Rozza. Model order reduction in fluid dynamics : challenges and perspectives. *Reduced Order Methods for modeling and computational reduction*, pages 235–273, 2014.

[97] Huan Lei, Nathan A Baker, and Xiantao Li. Data-driven parameterization of the generalized langevin equation. *Proceedings of the National Academy of Sciences*, 113(50) :14183–14188, 2016.

[98] Qianxiao Li, Felix Dietrich, Erik M. Bollt, and Ioannis G. Kevrekidis. Extended dynamic mode decomposition with dictionary learning : A data-driven adaptive spectral decomposition of the koopman operator. *Chaos : An Interdisciplinary Journal of Nonlinear Science*, 27(10), October 2017. ISSN 1089-7682. doi : $10.1063/1.4993854$. URL http://dx.doi.org/10.1063/1.4993854.

[99] Zhen Li, Xin Bian, Xiantao Li, and George Em Karniadakis. Incorporation of memory effects in coarse-grained modeling via the mori–zwanzig formalism. *The Journal of chemical physics*, 143(24), 2015.

[100] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2021. URL https://arxiv.org/abs/2010.08895.

[101] Yen Ting Lin, Yifeng Tian, Marian Anghel, and Daniel Livescu. Data-driven learning for the mori–zwanzig formalism : a generalization of the koopman learning framework, 2021. URL https://arxiv.org/abs/2101.05873.

[102] Phillip Lippe, Bastiaan S. Veeling, Paris Perdikaris, Richard E. Turner, and Johannes Brandstetter. Pde-refiner : Achieving accurate long rollouts with neural pde solvers, 2023.

[103] Phillip Lippe, Bastiaan S. Veeling, Paris Perdikaris, Richard E. Turner, and Johannes Brandstetter. Pde-refiner : Achieving accurate long rollouts with neural pde solvers, 2023. URL https://arxiv.org/abs/2308.05732.

[104] Jean-Christophe Loiseau, Bernd R. Noack, and Steven L. Brunton. Sparse reduced-order modelling : sensor-based dynamics to full-state estimation. *Journal of Fluid Mechanics*, 844 :459–490, April 2018. ISSN 1469-7645. doi : $10.1017/\mathrm{jfm}.2018.147$. URL http://dx.doi.org/10.1017/jfm.2018.147.

[105] Edward N Lorenz. Deterministic nonperiodic flow. *Journal of atmospheric sciences*, 20(2) :130–141, 1963.

[106] Fei Lu, Kevin K Lin, and Alexandre J Chorin. Data-based stochastic model reduction for the kuramoto–sivashinsky equation. *Physica D : Nonlinear Phenomena*, 340 :46–57, 2017.

[107] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3) :218–229, March 2021. ISSN 2522-5839. doi : $10.1038/s42256\text{-}021\text{-}00302\text{-}5$. URL http://dx.doi.org/10.1038/s42256-021-00302-5.

[108] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde : A deep learning library for solving differential equations. *SIAM review*, 63(1) :208–228, 2021.

[109] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks : Bridging deep architectures and numerical differential equations, 2020. URL https://arxiv.org/abs/1710.10121.

[110] John L Lumley. Stochastic tools in turbulence. volume 12. applied mathematics and mechanics. *Technical Report No. AD0710311 82 : RISO-R-1653*, 1970.

[111] Romit Maulik, Arvind Mohan, Bethany Lusch, Sandeep Madireddy, Prasanna Balaprakash, and Daniel Livescu. Time-series learning of latent-space dynamics for reduced-order model closure. *Physica D : Nonlinear Phenomena*, 405 :132368, April 2020. ISSN 0167-2789. doi : $10.1016/j.physd.2020.132368$. URL http://dx.doi.org/10.1016/j.physd.2020.132368.

[112] Alexandre Mauroy and Igor Mezić. Global stability analysis using the eigenfunctions of the koopman operator. *IEEE Transactions on Automatic Control*, 61(11) :3356–3369, 2016.

[113] Mark J McGuinness. The fractal dimension of the lorenz attractor. *Physics Letters A*, 99(1) :5–9, 1983.

[114] Hugo Melchers, B Koren, V Menkovski, dt Crommelin, and B Sanderse. *Machine learning for closure models*. PhD thesis, Master's thesis, Eindhoven University of Technology, 2022.

[115] Hugo Melchers, Daan Crommelin, Barry Koren, Vlado Menkovski, and Benjamin Sanderse. Comparison of neural closure models for discretised pdes. *Computers & Mathematics with Applications*, 143 :94–107, 2023.

[116] Emmanuel Menier, Michele Alessandro Bucci, Mouadh Yagoubi, Lionel Mathelin, and Marc Schoenauer. Cd-rom : Complemented deep - reduced order model. *Computer Methods in Applied Mechanics and Engineering*, 410 :115985, May 2023. ISSN 0045-7825. doi : 10.1016/j.cma.2023.115985. URL http://dx.doi.org/10.1016/j.cma.2023.115985.

[117] Igor Mezić. Spectral properties of dynamical systems, model reduction and decompositions. *Nonlinear Dynamics*, 41 :309–325, 2005.

[118] Igor Mezic. Spectrum of the koopman operator, spectral expansions in functional spaces, and state space geometry, 2019. URL https://arxiv.org/abs/1702.07597.

[119] Nicholas Minorsky. Self-excited oscillations in dynamical systems possessing retarded actions. *Journal of Applied Mechanics*, 1942.

[120] Thibault Monsel, Onofrio Semeraro, Lionel Mathelin, and Guillaume Charpiat. Neural state-dependent delay differential equations, 2023.

[121] Thibault Monsel, Emmanuel Menier, Lionel Mathelin, Onofrio Semeraro, and Guillaume Charpiat. Neural DDEs with Learnable Delays for Partially Observed Dynamical Systems. working paper or preprint, October 2024. URL https://hal.science/hal-04715748.

[122] Thibault Monsel, Onofrio Semeraro, Lionel Mathelin, and Guillaume Charpiat. Time and State Dependent Neural Delay Differential Equations. Proceedings of Machine Learning Research 255, ML-DE Workshop at ECAI 2024, 2024. URL https://hal.science/hal-04125875.

[123] Nicolas Morales, Liang Gu, and Yuqing Gao. Adding noise to improve noise robustness in speech recognition. In *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, volume 2, pages 930–933, 08 2007. doi : 10.21437/Interspeech.2007-335.

[124] Hazime Mori. A Continued-Fraction Representation of the Time-Correlation Functions. *Progress of Theoretical Physics*, 34(3) :399–416, 09 1965. ISSN 0033-068X. doi : 10.1143/PTP.34.399. URL https://doi.org/10.1143/PTP.34.399.

[125] Hazime Mori. Transport, Collective Motion, and Brownian Motion*). *Progress of Theoretical Physics*, 33(3) :423–455, 03 1965. ISSN 0033-068X. doi : 10.1143/PTP.33.423. URL https://doi.org/10.1143/PTP.33.423.

[126] Anatolii Dmitrievich Myshkis. General theory of differential equations with retarded arguments. *Uspekhi Matematicheskikh Nauk*, 4(5) :99–141, 1949.

[127] Kenneth W. Neves. Automatic integration of functional differential equations : An approach. *ACM Trans. Math. Softw.*, 1(4) :357–368, dec 1975. ISSN 0098-3500. doi : 10.1145/355656.355661. URL https://doi.org/10.1145/355656.355661.

[128] Chetan Nichkawde. Optimal state-space reconstruction using derivatives on projected manifold. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, 87(2) :022905, 2013.

[129] Bernd R. Noack and Helmut Eckelmann. A low-dimensional Galerkin method for the three-dimensional flow around a circular cylinder. *Physics of Fluids*, 6(1) :124–143, 01 1994. ISSN 1070-6631. doi : 10.1063/1.868433. URL https://doi.org/10.1063/1.868433.

[130] Lyle Noakes. The takens embedding theorem. *International Journal of Bifurcation and Chaos*, 1(04) :867–872, 1991.

[131] M.L. Hbid O. Arino and E. Ait Dads. *Delay Differential Equations and Applications*. NATO Science Series, Amsterdam, 2016.

[132] H. J. Oberle and H. J. Pesch. Numerical treatment of delay differential equations by hermite interpolation. *Numerische Mathematik*, 37(2) :235–255, Jun 1981. ISSN 0945-3245. doi : 10.1007/BF01398255. URL https://doi.org/10.1007/BF01398255.

[133] Maria Oprea, Mark Walth, Robert Stephany, Gabriella Torres Nothaft, Arnaldo Rodriguez-Gonzalez, and William Clark. Learning the delay using neural delay differential equations, 2023.

[134] N. H. Packard, J. P. Crutchfield, J. D. Farmer, and R. S. Shaw. Geometry from a time series. *Phys. Rev. Lett.*, 45 :712–716, Sep 1980. doi : 10.1103/PhysRevLett.45.712. URL https://link.aps.org/doi/10.1103/PhysRevLett.45.712.

[135] Norman H Packard, James P Crutchfield, J Doyne Farmer, and Robert S Shaw. Geometry from a time series. *Physical review letters*, 45(9) :712, 1980.

[136] CC Pain, MD Piggott, AJH Goddard, F Fang, GJ Gorman, DP Marshall, MD Eaton, PW Power, and CRE De Oliveira. Three-dimensional unstructured mesh ocean modelling. *Ocean Modelling*, 10(1-2) :5–33, 2005.

[137] Eric J. Parish and Karthik Duraisamy. Non-markovian closure models for large eddy simulations using the mori–zwanzig formalism. *Physical Review Fluids*, 2(1), January 2017. ISSN 2469-990X. doi : 10.1103/physrevfluids.2.014604. URL http://dx.doi.org/10.1103/PhysRevFluids.2.014604.

[138] Eric J Parish and Karthik Duraisamy. A dynamic subgrid scale model for large eddy simulations based on the mori–zwanzig formalism. *Journal of Computational Physics*, 349 :154–175, 2017.

[139] Eric J Parish and Karthik Duraisamy. Non-markovian closure models for large eddy simulations using the mori–zwanzig formalism. *Physical Review Fluids*, 2(1) :014604, 2017.

[140] Baker & Paul. Computing stability regions - runge–kutta methods for delay differential equations. *IMA Journal of Numerical Analysis*, 14 :347–362, 04 1993. doi : 10.5642/codee.201209.01.10.

[141] Louis M Pecora, Linda Moniz, Jonathan Nichols, and Thomas L Carroll. A unified approach to attractor reconstruction. *Chaos : An Interdisciplinary Journal of Nonlinear Science*, 17(1), 2007.

[142] Hans Pinckaers and Geert Litjens. Neural ordinary differential equations for semantic segmentation of individual colon glands, 2019.

[143] Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. Routledge, 2018.

[144] Alexei Potapov. Distortions of reconstruction for chaotic attractors. *Physica D : Nonlinear Phenomena*, 101(3-4) :207–226, 1997.

[145] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i) : Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv :1711.10561*, 2017.

[146] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part ii) : Data-driven discovery of nonlinear partial differential equations. *arXiv preprint arXiv :1711.10566*, 2017.

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

[148] Ramiro Rico-Martinez, K Krischer, IG Kevrekidis, MC Kube, and JL Hudson. Discrete-vs. continuous-time nonlinear signal processing of cu electrodissolution data. *Chemical Engineering Communications*, 118(1) : 25–48, 1992.

[149] Christian P Robert, George Casella, Christian P Robert, and George Casella. Monte carlo integration. *Monte Carlo statistical methods*, pages 71–138, 1999.

[150] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net : Convolutional networks for biomedical image segmentation, 2015.

[151] JE Rossiter. Wind-tunnel experiments on the flow over rectangular cavities at subsonic and transonic speeds. *Aeronautical Research Council and Reports and Memoranda*, 1964.

[152] Marc R Roussel. The use of delay differential equations in chemical kinetics. *The Journal of Physical Chemistry*, 100(20) :8323–8330, 1996.

[153] Yulia Rubanova, Ricky T. Q. Chen, and David Duvenaud. Latent odes for irregularly-sampled time series, 2019.

[154] Samuel H. Rudy, Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Data-driven discovery of partial differential equations, 2016.

[155] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning internal representations by error propagation, 1985.

[156] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2014.

[157] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations, 2018. URL https://arxiv.org/abs/1804.04272.

[158] Benjamin Sanderse, Panos Stinis, Romit Maulik, and Shady E. Ahmed. Scientific machine learning for closure models in multiscale problems : a review, 2024. URL https://arxiv.org/abs/2403.02913.

[159] Tim Sauer, James A Yorke, and Martin Casdagli. Embedology. *Journal of statistical Physics*, 65 :579–616, 1991.

[160] Andreas Schlaginhaufen, Philippe Wenk, Andreas Krause, and Florian Dörfler. Learning stable deep dynamics models for partially observed or delayed dynamical systems, 2021.

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

[162] Peter J. Schmid. Dynamic mode decomposition and its variants. *Annual Review of Fluid Mechanics*, 54(Volume 54, 2022) : 225–254, 2022. ISSN 1545-4479. doi : https://doi.org/10.1146/annurev-fluid-030121-015835. URL https://www.annualreviews.org/content/journals/10.1146/annurev-fluid-030121-015835.

[163] Johannes Schmude, Sujit Roy, Will Trojak, Johannes Jakubik, Daniel Salles Civitarese, Shraddha Singh, Julian Kuehnert, Kumar Ankur, Aman Gupta, Christopher E Phillips, Romeo Kienzler, Daniela Szwarcman, Vishal Gaur, Rajat Shinde, Rohit Lal, Arlindo Da Silva, Jorge Luis Guevara Diaz, Anne Jones, Simon Pfreundschuh, Amy Lin, Aditi Sheshadri, Udaysankar Nair, Valentine Anantharaj, Hendrik Hamann, Campbell Watson, Manil Maskey, Tsengdar J Lee, Juan Bernabe Moreno, and Rahul Ramachandran. Prithvi wxc : Foundation model for weather and climate, 2024. URL https://arxiv.org/abs/2409.13598.

[164] Lawrence Shampine and Skip Thompson. Delay-differential equations with constant lags. *CODEE Journal*, 9 :1–5, 01 2012. doi : 10.5642/codee.201209.01.10.

[165] William Snyder, Changhong Mou, Honghu Liu, Omer San, Raffaella De Vita, and Traian Iliescu. Reduced order model closures : A brief tutorial, 2022. URL https://arxiv.org/abs/2202.14017.

[166] Robert Stephany. Dde-find : Learning delay differential equations from noisy, limited data, 2024. URL https://arxiv.org/abs/2405.02661.

[167] Robert Stephany, Maria Antonia Oprea, Gabriella Torres Nothaft, Mark Walth, Arnaldo Rodriguez-Gonzalez, and William A Clark. Learning the delay in delay differential equations. In *ICLR 2024 Workshop on AI4DifferentialEquations In Science*, 2024.

[168] Panagiotis Stinis. Stochastic optimal prediction for the kuramoto–sivashinsky equation, 2003.

[169] Panagiotis Stinis. Higher order mori–zwanzig models for the euler equations, 2006.

[170] Panos Stinis. Numerical computation of solutions of the critical nonlinear schrödinger equation after the singularity. *Multiscale Modeling & Simulation*, 10(1) :48–60, 2012.

[171] Zhiqing Sun, Yiming Yang, and Shinjae Yoo. A neural pde solver with temporal stencil modeling, 2023.

[172] Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Dan MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. Pdebench : An extensive benchmark for scientific machine learning, 2023.

[173] Floris Takens. Detecting strange attractors in turbulence. In David Rand and Lai-Sang Young, editors, *Dynamical Systems and Turbulence, Warwick 1980*, pages 366–381, Berlin, Heidelberg, 1981. Springer Berlin Heidelberg. ISBN 978-3-540-38945-3.

[174] Eugene Tan, Shannon Algar, Débora Corrêa, Michael Small, Thomas Stemler, and David Walker. Selecting embedding delays : An overview of embedding techniques and a new method using persistent homology. *Chaos : An Interdisciplinary Journal of Nonlinear Science*, 33(3), 2023.

[175] Eugene Tan, Shannon Algar, Débora Corrêa, Michael Small, Thomas Stemler, and David Walker. Selecting embedding delays : An overview of embedding techniques and a new method using persistent homology. *Chaos : An Interdisciplinary Journal of Nonlinear Science*, 33 (3), March 2023. ISSN 1089-7682. doi : $10.1063/5.0137223$. URL http://dx.doi.org/10.1063/5.0137223.

[176] Eugene Tan, Shannon Algar, Débora Corrêa, Michael Small, Thomas Stemler, and David Walker. Selecting embedding delays : An overview of embedding techniques and a new method using persistent homology. *Chaos : An Interdisciplinary Journal of Nonlinear Science*, 33 (3) :032101, 03 2023. ISSN 1054-1500. doi : $10.1063/5.0137223$. URL https://doi.org/10.1063/5.0137223.

[177] Eugene Tan, Shannon Algar, Débora Corrêa, Michael Small, Thomas Stemler, and David Walker. Selecting embedding delays : An overview of embedding techniques and a new method using persistent homology. *Chaos : An Interdisciplinary Journal of Nonlinear Science*, 33 (3), March 2023. ISSN 1089-7682. doi : $10.1063/5.0137223$. URL http://dx.doi.org/10.1063/5.0137223.

[178] Xinliang Tian. On the intrinsic three-dimensionality of the flow normal to a circular disk, 2019. URL https://arxiv.org/abs/1906.10578.

[179] Yifeng Tian, Yen Ting Lin, Marian Anghel, and Daniel Livescu. Data-driven learning of Mori–Zwanzig operators for isotropic turbulence. *Physics of Fluids*, 33(12) :125118, December 2021. ISSN 1070-6631, 1089-7666. doi : $10.1063/5.0070548$. URL https://aip.scitation.org/doi/10.1063/5.0070548.

[180] Francesco Giacomo Tricomi. *Integral equations*, volume 5. Courier corporation, 1985.

[181] F Tuerke, François Lusseyran, Denisse Sciamarella, Luc Pastur, and G Artana. Nonlinear delayed feedback model for incompressible open cavity flow. *Physical Review Fluids*, 5(2) :024401, 2020.

[182] Belinda Tzen and Maxim Raginsky. Neural stochastic differential equations : Deep latent gaussian models in the diffusion limit. *arXiv preprint arXiv :1905.09883*, 2019.

[183] L. C. Uzal, G. L. Grinblat, and P. F. Verdes. Optimal reconstruction of dynamical systems : A noise amplification approach. *Physical Review E*, 84(1), July 2011. ISSN 1550-2376. doi : $10.1103/\mathrm{physreve.84.016223}$. URL http://dx.doi.org/10.1103/PhysRevE.84.016223.

[184] Harsha Vaddireddy, Adil Rasheed, Anne E. Staples, and Omer San. Feature engineering and symbolic regression methods for detecting hidden physics from sparse sensor observation data. *Physics of Fluids*, 32(1), January 2020. ISSN 1089-7666. doi : $10.1063/1.5136351$. URL http://dx.doi.org/10.1063/1.5136351.

[185] Daniele Venturi and Xiantao Li. The mori-zwanzig formulation of deep learning, 2023. URL https://arxiv.org/abs/2209.05544.

[186] Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks : analysis and efficient estimation. *Advances in Neural Information Processing Systems*, 31, 2018.

[187] Andrei G Vladimirov, Dmitry Turaev, and Gregory Kozyreff. Delay differential equations for mode-locked semiconductor lasers. *Optics letters*, 29(11) :1221–1223, 2004.

[188] David Widmann and Chris Rackauckas. Delaydiffeq : Generating delay differential equation solvers via recursive embedding of ordinary differential equation solvers, 2022. URL https://arxiv.org/abs/2208.12879.

[189] Daniel S Wilks and Robert L Wilby. The weather generation game : a review of stochastic weather models. *Progress in physical geography*, 23 (3) :329–357, 1999.

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

June 2015. ISSN 1432-1467. doi : 10.1007/s00332-015-9258-5. URL http://dx.doi.org/10.1007/s00332-015-9258-5.

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

[192] Fan Wu, Sanghyun Hong, Donsub Rim, Noseong Park, and Kookjin Lee. Mining causality from continuous-time dynamics models : An application to tsunami forecasting, 2022.

[193] Niklas Wulkow. *Modelling Observations of Dynamical Systems with Memory*. Dissertation, Freie Universitat Berlin, 2022. URL http://dx.doi.org/10.17169/refubium-35182.

[194] Zhonghui You, Jinmian Ye, Kunming Li, Zenglin Xu, and Ping Wang. Adversarial noise layer : Regularize neural network by adding noise. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 909–913, 2019.

[195] Emanuele Zappala. Spectral methods for neural integral equations, 2024. URL https://arxiv.org/abs/2312.05654.

[196] Emanuele Zappala, Antonio Henrique de Oliveira Fonseca, Andrew Henry Moberly, Michael James Higley, Chadi Abdallah, Jessica Cardin, and David van Dijk. Neural integro-differential equations, 2022. URL https://arxiv.org/abs/2206.14282.

[197] Emanuele Zappala, Antonio Henrique de Oliveira Fonseca, Josue Ortega Caro, and David van Dijk. Neural integral equations, 2023. URL https://arxiv.org/abs/2209.15190.

[198] A M Zhabotinskii. [PERIODIC COURSE OF THE OXIDATION OF MALONIC ACID IN a SOLUTION (STUDIES ON THE KINETICS OF BEOLUSOV'S REACTION)]. *Biofizika*, 9 :306–311, 1964.

[199] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. Unet++ : Redesigning skip connections to exploit multiscale features in image segmentation. *IEEE transactions on medical imaging*, 39(6) :1856–1867, 2019.

[200] Qunxi Zhu, Yao Guo, and Wei Lin. Neural delay differential equations, 2021.

[201] Qunxi Zhu, Yifei Shen, Dongsheng Li, and Wei Lin. Neural piecewise-constant delay differential equations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 9242–9250, 2022.

[202] Qunxi Zhu, Yao Guo, and Wei Lin. Neural delay differential equations : System reconstruction and image classification, 2023.

[203] Yuanran Zhu. *Mori–Zwanzig Equation : Theory and Applications*. University of California, Santa Cruz, 2019.

[204] Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James S. Duncan. Adabelief optimizer : Adapting stepsizes by the belief in observed gradients, 2020.

[205] Hossein Zivari-Piran and Wayne H Enright. Accurate first-order sensitivity analysis for delay differential equations : Part ii : The adjoint approach. *preprint, Department of Computer Science, University of Toronto*, 2009.

[206] Hossein Zivari-Piran and Wayne H Enright. An efficient unified approach for the numerical solution of delay differential equations. *Numerical Algorithms*, 53(2) :397–417, 2010.

[207] Robert Zwanzig. Approximate eigenfunctions of the liouville operator in classical many-body systems. *Phys. Rev.*, 144 :170–177, Apr 1966. doi : $10.1103/\mathrm{PhysRev.144.170}$. URL https://link.aps.org/doi/10.1103/PhysRev.144.170.

[208] Robert Zwanzig. Nonlinear generalized langevin equations. *Journal of Statistical Physics*, 9(3) :215–220, 1973.

[209] Robert Zwanzig, K. S. J. Nordholm, and W. C. Mitchell. Memory effects in irreversible thermodynamics : Corrected derivation of transport equations. *Phys. Rev. A*, 5 :2680–2682, Jun 1972. doi : $10.1103/\mathrm{PhysRevA.5.2680}$. URL https://link.aps.org/doi/10.1103/PhysRevA.5.2680.