Deep neural networks and partial differential equations

Léon Migus

Sorbonne Université

École doctorale **École doctorale de sciences mathématiques de Paris centre (ED 386)**
Laboratoires *Institut des Systèmes Intelligents et de Robotique*, *Laboratoire Jacques-Louis Lions* et *INRIA*

Réseaux de neurones profonds et équations aux dérivées partielles

Par **Léon Migus**

Thèse de doctorat de **Mathématiques**

Dirigée par	Julien Salomon	Directeur	Directeur de Recherche à l'INRIA
	Patrick Gallinari	Co-directeur	Professeur des universités à Sorbonne Université

A été présentée et soutenue publiquement le 1er décembre 2023 devant un jury composé de:

Rapporteurs	Emmanuel Frénod	Professeur des universités à l'Université Bretagne Sud
	Gilles Louppe	Professeur à l'Université de Liège
Examinateurs	Olga Mula	Associate Professor at Eindhoven University of Technology
	Rodolphe Turpault	Professeur des Universités à l'Institut de Mathématiques de Bordeaux **Président du Jury**
	Romain Tavenard	Professeur des Universités à l'Université de Rennes 2
Invité	Emmanuel Audusse	Maître de conférences à l'Université Paris 13

Dédié à mes parents et à mon oncle

Deep neural networks and partial differential equations

Abstract

The study of physical systems, modeled by partial differential equations (PDEs), represents a cornerstone of scientific research. These equations, describing the relation between some function and its partial derivatives across variables, are vital for modeling diverse phenomena, for e.g. fluid dynamics and heat transfer, with applications in various domains such as climate science and astronomy. In the recent years, data has become readily available, explaining partly the rise of data-driven methods and more particularly Deep Learning (DL) methods. They excel in training complex models on a large amount of data, while being computationally effective at inference. However, for physical systems, even with apparently large amount of data, data is often scarce compared to the complexity of the problems, which is a challenge for DL methods. More importantly, the problems faced when applying DL methods to physical systems are very different from the usual DL problems, with physical problems potentially being be ill-posed, chaotic or very sensitive to initial conditions. In addition to these main challenges, practitioners in numerical analysis or the industry seek theoretical or experimental guarantees of convergence, which DL methods can lack of.

In this thesis, we tackle some of these challenges through three distinct approaches. In the first part of this work, we apply concepts from numerical analysis into DL frameworks, offering two perspectives: (i) the incorporation of multigrid numerical schemes into a Multi-Scale DL architecture, the Multipole Graph Neural Operator, demonstrating its efficacy in solving steady-state Darcy flow and 1D viscous unsteady Burgers' equations (ii) the adaptation of implicit numerical schemes into neural networks, ensuring forecasting stability for dynamical systems via some constraints on the weights of the neural network, leading to improved long-term forecasting results for two transport PDEs. In the second part of this work, we design a hybrid model to address the friction law design in Shallow-Water equations. This implies learning the friction law from observations through a numerical solver. The experiments focus on a vast analysis of the robustness and convergence for a stationary case and confirm the efficacy on the dynamic case. In the third part of this work, we explore continuous methods through two works based on Implicit Neural Representations (INRs): (i) INFINITY is a INR based method that can be applied to static PDE problems. It is tested on the RANS equations for surrogate modeling of airfoils. INFINITY can accurately infer physical fields throughout the volume and surface, leading to a correct prediction of the drag and lift coefficients, which are crucial for airfoil design. (ii) TimeFlow is a general framework using INRs to impute and forecast time series. By its continuous nature, TimeFlow can handle missing data, irregular sampling and unaligned observations from multiple sensors while having similar performances to state-of-the-art algorithms and being able to generalize to unseen samples and time windows.

Réseaux de neurones profonds et équations aux dérivées partielles

Résumé

L'étude des systèmes physiques, modélisés par des équations aux dérivées partielles, représente une pierre angulaire de la recherche scientifique. Ces équations, qui décrivent la relation entre une fonction et ses dérivées partielles à travers différentes variables, sont essentielles pour modéliser divers phénomènes avec des applications en climatologie ou astronomie par exemple. La récente accessibilité en masse des données a favorisé l'essor des méthodes basées sur les données et plus particulièrement des méthodes d'apprentissage profond (Deep Learning, DL), qui peuvent apprendre des modèles complexes en utilisant des grandes quantités de données. Cependant, pour les systèmes physiques, même avec une quantité apparemment importante de données, les données sont souvent rares par rapport à la complexité des problèmes, ce qui constitue un défi pour ces méthodes d'apprentissage profond. De plus, ces problèmes posent des défis particuliers aux algorithmes de DL, en pouvant être mal posés, chaotiques ou très sensibles aux conditions initiales. En plus de ces défis, des garanties théoriques ou expérimentales de convergence sont recherchées, ce que les méthodes DL peuvent ne pas avoir.

Dans cette thèse, nous nous attaquons à certains de ces défis par trois approches distinctes. Dans la première partie de ce travail, nous appliquons des concepts de l'analyse numérique au DL, en offrant deux perspectives : (i) l'incorporation de schémas numériques multi-grilles dans une architecture DL multi-échelle, démontrant son efficacité dans la résolution de la loi de Darcy et des équations de Burgers 1D visqueuses et non stationnaires. (ii) l'adaptation de schémas numériques implicites dans des réseaux de neurones profonds, garantissant la stabilité des prévisions pour les systèmes dynamiques par le biais de certaines contraintes sur les poids du réseau, conduisant à de meilleurs résultats de prévision à long terme pour deux équations de transport. Dans la deuxième partie de ce travail, nous concevons un modèle hybride pour aborder la conception de la loi de frottement dans les équations de Saint-Venant. Cela implique l'apprentissage de la loi de frottement à partir des observations à travers un schéma numérique. Les expériences consistent en une vaste analyse de la robustesse et de la convergence pour un cas stationnaire et confirment l'efficacité sur un cas dynamique. Dans la troisième partie de ce travail, nous explorons les méthodes continues à travers deux travaux basés sur les représentations neuronales implicites (Implicit Neural Representations, INR) : (i) INFINITY est une méthode fondée sur les INRs qui peut être appliquée aux problèmes statiques. Elle est testée sur les équations de RANS pour la modélisation de profils aérodynamiques, et conduit à une prédiction correcte des champs physiques et des coefficients de traînée et de portance. (ii) TimeFlow est un algorithme général qui utilise les INR pour imputer et prévoir des séries temporelles. De par sa nature continue, TimeFlow peut gérer les données manquantes, l'échantillonnage irrégulier et les observations non alignées provenant de capteurs multiples.

Remerciements

Arrivé au terme de ce doctorat, qui fut une aventure riche en émotions, je tiens à remercier tous ceux qui ont rendu cette aventure possible et qui m'ont aidé à la mener à terme.
Je tiens tout d'abord à remercier mes directeurs de thèse, Julien et Patrick. J'ai grandement apprécié votre aide tout au long de cette thèse, vous avez su m'inspirer, me soutenir et m'avez accordé une grande confiance. Je te suis très reconnaissant Patrick d'avoir accepté de me prendre en thèse avec toi et de m'avoir donné des conseils et directions toujours pertinents. Je te remercie grandement aussi Julien d'avoir assuré un service toujours régulier, de m'avoir guidé avec plein de propositions et d'avoir été à l'écoute de mes idées, j'ai beaucoup apprécié nos échanges.
Je remercie ensuite l'ensemble des membres du jury, qui ont accepté d'examiner et d'évaluer mon travail de recherche scientifique de ces trois dernières années. Je tiens tout d'abord à remercier Emmanuel Frénod et Gilles Louppe d'avoir accepté de rapporter ma thèse, c'est un travail conséquent et je vous en suis reconnaissant. Je tiens ensuite à remercier Olga Mula, Rodolphe Turpault et Romain Tavenard d'avoir accepté de participer au jury ainsi que d'évaluer ma thèse.
Je tiens à remercier les personnes avec qui j'ai eu la chance de collaborer ainsi que les membres du MLIA et du LJLL que j'ai pu côtoyer au quotidien. J'ai grandement apprécié l'atmosphère de travail et l'ambiance de collaboration à laquelle j'ai eu la chance de participer. Je tiens tout d'abord à remercier Yuan. Sans toi je n'aurais pas réussi à mener à bien cette thèse, tu m'as beaucoup guidé et aidé durant ces trois années, et j'ai grandement apprécié collaborer avec toi, merci pour tous tes conseils et échanges. Tu as été mon troisième encadrant en quelque sorte, je t'en suis très reconnaissant. Je tiens ensuite à remercier Etienne et Louis, c'était un vrai plaisir de collaborer avec vous, j'ai beaucoup appris et vraiment apprécié nos travaux, vous avez égayé ma troisième année. Je tiens également à remercier Emmanuel Audusse, j'apprécie beaucoup nos échanges, c'était très intéressant de travailler avec toi, tu as toujours eu plein de remarques pertinentes qui m'ont poussé dans des directions intéressantes, un vrai plaisir ! Merci aussi à Ahmed, avec Yuan vous avez formé un duo efficace et complémentaire et j'ai beaucoup appris de nos collaborations. Je remercie tous ceux avec qui j'ai eu la chance de partager mon quotidien durant ces trois années, sans être exhaustif, de Rémi et Alexandre au LJLL au début de ma

thèse, à Agnès, Jean-Yves, Matthieu et Marie durant notamment les périodes plus dures de la pandémie, et Lise, Paul, Pierre Thomas, Tanguy et Tristan pour les périodes plus récentes de cette aventure. Merci à Awatef, Christophe, Nadine et Sylvie ainsi que le secrétariat du LJLL qui ont permis que cette thèse se déroule dans de bonnes conditions.
Je tiens également à remercier le Sorbonne Center for Artificial Intelligence (SCAI) pour m'avoir accordé un financement et avoir rendu ce doctorat possible.
Cette thèse est l'aboutissement d'un long chemin académique. Je tiens à remercier l'ensemble du corps enseignant que j'ai eu la chance d'avoir durant ma scolarité. J'ai toujours été bien encadré et ait grandement apprécié apprendre durant les différentes phases de ma scolarité, jusqu'au baccalauréat, en classes préparatoires, en école d'ingénieur et à l'université en Angleterre. J'ai une pensée particulière pour Mme Mandalka et Mme Rousseau, qui m'ont donné le goût du travail et Benoit Valiron, Kevin Cohen et Marina Evangelou, qui m'ont fait découvrir et apprécier la recherche.
Je tiens à remercier tous mes amis hors du travail, qui m'ont aidé à me construire et m'ont soutenu durant ces trois années de thèse. Sans être exaustif, j'ai une pensée pour Alice, Arthur, François, Hugo, Jeya, Jules, Léo, Léa, Louis-Solenn, Marianne, Paul, Pierre et Roman. Une dédicace évidemment aussi au Palier de rêve, avec qui j'ai passé plein de bons moments en école d'ingénieur et après, et dont les échanges et expériences m'ont aidé à trouver ma voie.
Je tiens enfin et surtout à remercier ma famille et mes parents, à qui je dois tout. Mettre en mots tout ce que je dois à ma famille est impossible. Je pense fort à ma mère, elle m'a accompagné durant tous les moments importants de ma vie, a su me soutenir durant les moments durs et durant toute ma scolarité, j'ai une chance infinie. J'ai une pensée émue pour mon père, j'aurais aimé partagé ces moments avec toi. Je pense à mon oncle Alain, je te dois beaucoup, tu m'as donné le goût du travail et tu m'as accompagné durant toute ma scolarité. J'ai de souvenirs émus de tes leçons dominicales de mathématiques, merci aussi pour tout tes conseils en général et durant mon doctorat également. J'ai une pensé particulière pour Nadine, Mimi et Doulé, et Christian et Isabelle.

Symbols

Domain	Symbol	Name	Description
Neural network	θ	weights of a neural network	
	W, b	weights and bias of a linear layer	
	K	kernel	
	z	latent space of neural network	
	ν	a probablistic measure	
	Ω	domain	
	h_w	hyper-network	
	$\mathcal{B}$	sample batch	
	$\mathcal{L}$	loss	
	$\mathcal{N}(x)$	domain of neighbours of x	
	t	time	
Numerical scheme	$(\Delta t,\, \Delta x)$	time and space discretization step	
	$\lambda_{pf}(M)$	Perron–Frobenius eigenvalue	
Shallow water equation	u	water speed	
	h	water height	
	b	topology	
	g	gravitational constant	
	K_f	friction law	
	S_f	laws of friction family	$S_f = K_f(h, u)\|u\|$
	C_f	friction coefficient	
	$(\alpha,\, \beta)$	coefficients of the friction law	
	Q_e	input water flow	
	h_s	water height on the right of the domain	
RANS equation	$d(\boldsymbol{x})$	distance function	
	$\boldsymbol{n}(\boldsymbol{x}) = (n_x(\boldsymbol{x}), n_y(\boldsymbol{x}))$	normal vectors of the mesh nodes on the airfoil surface	
	$(V_x,\, V_y)$	inlet velocity values	
	(v_x, v_y)	velocities	
	p	pressure	
	ν_t	turbulent kinematic viscosity	
	$\partial\Omega_i$	boundary conditions	
	$\mathcal{S}_i$	surface mesh	
	$\mathcal{X}_i$	mesh	

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

Modeling and studying physical systems is one of the largest area of research in science, that has been developed over centuries. More precisely, physical systems can often be modeled by partial differential equations (PDEs). PDEs are equations containing a function and its partial derivative across several of its variables. They can model systems such as fluids with the Navier-Stokes equation or heat exchange with the Heat equation. They are ubiquitous, and crucial for many domains, such as climate science or astronomy. They are our object of research.

1.1.1 Increasing amount of data and data-driven methods

Over the last decades, with the increasing amount of sensors, data has become highly available. This can be seen across different fields, from baseball to economic forecasting (Lohr, 2012). However, data is not always of good quality and often needs to be processed. For the study of physical systems, data can come from two sources:

- Real-world data, such as satellite images of phenomena.
- Simulation data, which comes from numerically simulating some phenomenon.

Both sources have their challenges and limitations. Problems with real-world data can sometimes be ill-posed, not all the information that is required to solve the problem is available. This is illustrated with the NEMO engine simulations (Madec et al., 2017). Indeed, when trying to predict the sea-surface temperatures (SST) on some regions of the ocean, such as in De Bézenac et al. (2019) and shown in Figure 1.1, the influence of the outside regions makes it impossible to forecast when the horizon is not small. In addition to this issue, real-world data is often scarce compared to the complexity of the problem. Indeed, even huge datasets can be too

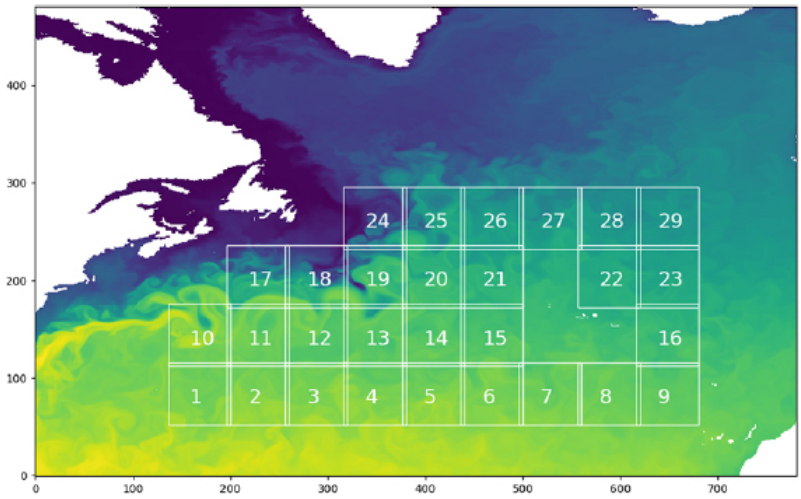


Figure 1.1: Sub regions of the SST dataset used in De Bézenac et al. (2019).

small. This can be seen on the HadSST4 dataset (Kennedy et al., 2019), which is composed of monthly grids from quality-controlled *in situ* SST measurements. Despite the size of the dataset, with monthly grids of 1296 points from 1850 to 2018, this dataset is particularly challenging because there are missing values and the resolution is coarse, hence features like western boundary currents and eddies are not resolved.

On the other hand, numerical simulations give good quality data, but they are still simulations, and often can not be used directly for real-world problems. This is illustrated for a greenhouse control system (Kim et al., 2017), where the physical simulations are not enough to accurately control the system, and need to be complemented by a data model. There is still an in-between, with high-fidelity simulations on complex problems, such as the AirfRANS dataset (Bonnet et al., 2022), which simulates an airfoil design optimization problem.

The global increase in amount of data has led to the development and success of data-driven methods. Among these methods, Deep Learning (DL) methods have been applied successfully to many problems, from diagnosis in healthcare to conversational agents with GPT-3. The idea behind DL is to learn highly non-linear mappings from lots of data, using architectures with many parameters (up to hundreds of billion). They can be applied to many different tasks, with mainly classification, regression and generation. One of their strengths is the small computation time at inference, using parallel computing with Graphics Processing Units (GPUs). In addition, they can handle unstructured data.

1.1.2 Model-based methods and their challenges

On the contrary to plain data-driven methods, model-based methods use prior knowledge to provide models, instead of the pure data knowledge. Modeling physi-

cal systems forms the foundation of physics, paving the way for some of humanity's greatest discoveries, ranging from our understanding of gravity to the development of general relativity. These models are defined by PDEs, as explained earlier. However, these PDEs are often intractable analytically, hence the development of numerical analysis, which aims to numerically solve these equations to predict physical systems. Numerical solvers are derived with theoretical properties in order to offer some guarantee on the accuracy of the proposed numerical solution. This will be further detailed in 2.1.2.
These schemes can be generic or tailored for specific problems, and they can quickly become very complex. This is illustrated for geotechnical problems, where experts design successful numerical solvers, but which need to be carefully crafted (Schweiger et al., 2019). Hence expert knowledge is required to use them in real-world applications, which is quite restrictive. Additionally, they are very costly computationally. Indeed, the whole scheme needs to be run again for a new simulation, the previous computations cannot be used again. For instance, to compute the trajectory of a tennis ball, a new costly simulation needs to be run for each shot, which can be prohibitive for real-world applications. Some solutions have been developed to mitigate this issue, but the core of the problem still is the incapacity to add existing data to these models.
Another issue with these models is that they often do not totally represent the true system. Indeed, the model is based on some assumptions to offer a general view and comprehension of the system. But these assumptions and simplifications also limit the accuracy of the model in various cases. For instance, the Helmholtz equation is a simplification of the wave equation by removing the time dependency, hence its applications are restricted, but its analysis is simpler.

1.1.3 Challenges in applying data-driven methods to model-based problems

A first and widely explored way of adding data to a physical model is data assimilation. It consists in correcting the solution given by a numerical solver with some observations. More precisely, the scheme outputs a first guess at some update time, and this guess is then updated to reduce the error between this first guess and some observations. This procedure is repeated until the final time is reached. It has been initially developed for weather prediction, and has been successfully applied in large-scale weather forecasting systems such as the European Centre for Medium-Range Weather Forecasts (ECMWF; Derber and Bouttier (1999)) or in temperature, dust, and ice retrievals for the Martian atmosphere with the Mars Climate Sounder (Montabone et al., 2014). However, they are restricted to some specific applications, since they are not part of a model, they can only correct it. Moreover, they are still computationally very costly, since simulations need to be

run for each new trajectory. To conclude on data assimilation, it is a very useful set of tools, that are used for real-world applications, but can only correct physical simulations and need to be used with complex numerical solvers, hence an expensive cost and an expert knowledge needed.
These limitations opened the way for the application of DL methods to these problems. Indeed, being very efficient at inference time and able to learn highly-non linear mappings from data, they are a natural fit. However, as explained earlier, data is often scarce, which is a challenge for these methods. More importantly, the complexity and challenges of the problems are different than other areas where DL has been applied, problems can be ill-posed, chaotic or very sensitive to initial conditions. In addition to these main challenges, practitioners in numerical analysis or the industry seek theoretical or experimental guarantees of convergence, which DL methods can lack of. This is in this exciting context that this thesis has been written, where recent research try to overcome these challenges in order to successfully apply DL methods to dynamical systems.

1.2 Contributions of the thesis

In this thesis, we aim at addressing some of the challenges presented previously through different angles:

(i) Incorporating numerical analysis ideas and theoretical guarantees into DL frameworks.

(ii) Designing a hybrid approach combining numerical solvers and DL to provide a model using both data and prior knowledge.

(iii) Designing continuous models to predict dynamical systems.

The first angle of incorporating numerical analysis ideas into DL frameworks is divided into two parts. The first work tackles incorporating different variations of multigrid numerical schemes into a multi-scale DL architecture, namely a Multipole Graph Neural Operator. The method shows interesting results on two families of PDEs, the steady-state of Darcy flow and the 1D viscous unsteady Burgers' equation. The second work tackles adapting implicit numerical schemes into a neural network in order to ensure a forecasting stability when predicting dynamical systems. This leads us to introduce hard constraints on the network. Our experimental results then validate our stability property, and show improved results at long-term forecasting for two transports PDEs. These two works, detailed in Chapter 3, led to the two following publications in International Conference on Learning Representations (ICLR) workshops (Migus et al., 2022, 2023).

Migus, L., Yin, Y., Mazari, J. A., and Gallinari, P. (2022, November). Multi-Scale Physical Representations for Approximating PDE Solutions with Graph Neural Operators. In Topological, Algebraic and Geometric Learning Workshops 2022 (pp. 332-340). PMLR.

Migus, L., Salomon, J., and Gallinari, P. (2023, May). Stability of implicit neural networks for long-term forecasting in dynamical systems. In ICLR 2023 Workshop on Physics for Machine Learning.

The second angle is designing a hybrid model that tackles the Shallow-Water equations. These equations contain a friction law but there is no consensus on the formulation of this law. This parametrized law is then often optimized without too much theoretical justifications. In this work, we aim at replacing this friction law with a data-driven model integrated in the equations and thus interacting with a numerical solver. This leads to the design of a hybrid model that can learn from observations to reproduce an effective friction law. This work, detailed in Chapter 4, is a collaboration with Emmanuel Audusse and will soon be submitted to a journal.

Migus, L., Salomon, J., Audusse E. and Gallinari, P. . Learning a friction law for the Shallow Water equations through observations. *Soon to be submitted to a computational physics oriented journal.*

The third and final angle of this thesis is the design of continuous methods and is divided into two works using a very similar method. The first work tackles an airfoil optimization problem modeled by the Reynolds-averaged Navier-Stokes equations. Based on a new family of DL methods called Implicit Neural Representations (INRs), this work consists in designing an efficient surrogate model which is then showed to accurately infer physical fields throughout the volume and surface. This leads to a correct prediction of the drag and lift coefficients while adhering to the equations, which is crucial for airfoil optimization. The second work tackles time series modeling. Based on INRs as well, this work tackles imputation and forecasting of time series. Using the continuous formulation of INRs and some additional properties, the method designed in this work can handle missing data, irregular sampling and unaligned observations from multiple sensors. Moreover, the experimental results show state-of-the-art performances while being able to generalize to unseen samples and time windows. These two works, detailed in Chapter 5, led to one publication in a International Conference on Machine Learning (ICML) workshop (Serrano et al., 2023b) and one article under review in ICLR 2024 (Naour et al., 2023).

Serrano, L., **Migus, L.**, Yin, Y., Mazari, J. A., and Gallinari, P. (2023). INFINITY: Neural Field Modeling for Reynolds-Averaged Navier-Stokes Equations. In ICML 2023 workshop on the synergy of scientific and machine learning modeling.

Naour, E. L., Serrano, L., **Migus, L.**, Yin, Y., Agoua, G., Baskiotis, N., Gallinari, P., and Guigue, V. (2023). Time Series Continuous Modeling for Imputation and Forecasting with Implicit Neural Representations. Under review at ICLR 2024.

1.3 Structure of the thesis

This thesis is organized as follows. In Chapter 2, the main notions of numerical analysis and DL are presented, before a dive into the DL methods for dynamical systems. In Chapter 3, the contributions on integrating notions of numerical analysis in DL methods are presented. In Chapter 4, the contribution on hybrid modeling for the Shallow-Water equations is detailed, while in Chapter 5, the contributions on INRs for the airfoil optimization problem and time series modeling are presented.

Chapter 2

Background and Related Work

In order to fully understand the content of this thesis, some background and discussion on recent related work is needed. This chapter first reviews the main notions of numerical analysis used in this thesis, before presenting the basics of DL. Finally, a related work of the links between DL and numerical analysis is detailed.

2.1 Numerical analysis

Numerical analysis is the mathematical field that study numerical algorithms to solve continuous problems. It includes designing efficient new algorithms, but also proving and guaranteeing some theoretical properties about these algorithms. In this thesis, a lot of inspiration is taken from this field. Before presenting the numerical schemes themselves, some basics on continuous problems need to be detailed.

2.1.1 Partial differential equations

As explained in Section 1.1.2, physical phenomena are often modeled by differential equations. They are usually continuous, which is the topic of this thesis. This section first presents ordinary differential equations (ODEs), before defining partial differential equations (PDEs) and hyperbolic PDEs.

ODE An ODE, as defined in Definition 2.1.1, is an equation involving derivatives of functions depending on a single variable, which is often the time. They can model many physical phenomena. As a common thread, Lotka–Volterra (LV) equations are used throughout this paragraph to illustrate the concept of an ODE. These equations can model the dynamics of a biological system of prey and predators, which is illustrated in Figure 2.1. The state of the system is the number of preys and predators at a given time t. The population of prey and predators starts from an initial population of these two, which is the initial state, and then it evolves in time accordingly to how many prey are available for predators and how many

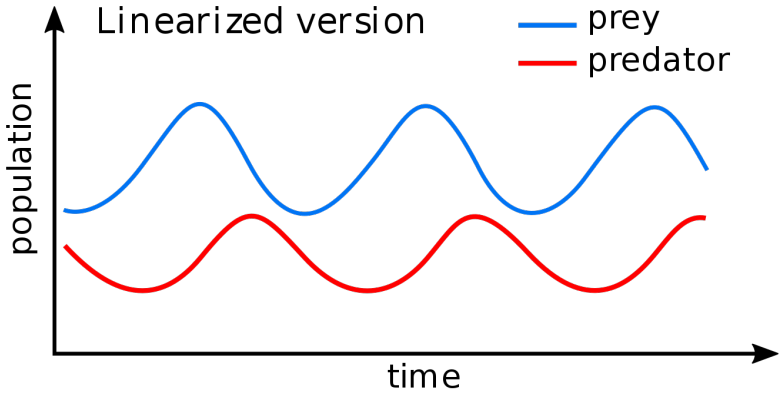


Figure 2.1: Example of a LV equations solution.

predators are eating preys. This is a dynamical system modeled by an ODE, which is defined in Example 2.1.1.

Definition 2.1.1 (First-order ordinary differential equation)**.** *Considering* $u : \mathbb{R} \to \mathbb{R}^p$ *and* $f : \mathbb{R} \to \mathbb{R}^p$*, where* f *is continuous on an open set* D *of* $\mathbb{R}^{p+1}$*, a first order ODE is defined by:*

$$\forall t \in I \subset \mathbb{R}, \frac{du}{dt} = f(t, u(t)), \tag{2.1}$$

where t *is the time variable and* u *the state variable.*

Remark 2.1.1 (Linear ODE)**.** *If* $\forall t \in I \subset \mathbb{R}, f(t, u(t)) = A(t)u(t) + b(t)$*, where* $A(t)$ *and* $b(t)$ *are continuous functions, Equation (2.1) is a linear ODE, and if, furthermore,* $b(t) = 0$*, it is a linear homogeneous ODE.*

Example 2.1.1 (Lotka-Voleterra ODE)**.** *The LV equations are defined by:*

$$\begin{aligned} \frac{du_1}{dt} &= au_1 - bu_1u_2, \\ \frac{du_2}{dt} &= cu_1u_2 - du_2, \end{aligned} \tag{2.2}$$

where u_1 *and* u_2 *are the population density of the prey and the predator respectively,* a *and* d *the maximum growth rate of the prey and the predator,* b *the effect of the predators on the preys growth rate and* c *the effect of the preys on the predators growth rate. This model boils down to a few biological hypothesis that are transcribed mathematically. For instance, the food supply of the predator population depends entirely on the size of the prey population, which is modeled by the variable* u_2 *depending only on* u_1 *and* u_2*, not some other preys. It is possible to add other hypothesis to the LV equations, which leads to the generalized LV system of equations, which can model more complex phenomena.*

Once an equation is defined, predicting the phenomena behind the equation is equivalent to finding a solution, which is defined in Definition 2.1.2. For instance, once the LV equations are defined, the goal is to predict the populations, as shown in Figure 2.1.

Definition 2.1.2 (Solution of an ODE)**.** *A solution of an ODE is defined by a pair* (J, u)*, with* $J \subset \mathbb{R}$ *and* u *a function differentiable on* J *with values in* $\mathbb{R}^p$*, such that Equation* (2.1) *is satisfied and* $\forall t \in J, (t, u(t)) \in D$*.*

Example 2.1.2 (Lotka-Voleterra solution)**.** *The solutions of LV equations are periodic but cannot be analytically expressed. However, around the equilibrium point* $u_1^{eq} = d/c$ *and* $u_2^{eq} = a/b$*, the equation can be linearized, and the solution around this point is then a simple harmonic motion. This is illustrated in Figure 2.1. This trajectory is coherent with the biological model; when the number of preys increases, the number of predators increases, then because predators eat the preys the number of preys decreases and in consequence so the number of predators. In this closed-system with these assumptions, this motion never stops.*

There can be an infinity of solution to an ODE. However, in most cases, the goal is more precise than finding a solution to an ODE, it is to find the solution to an ODE with additional constraints. For instance, for the Lotka-Volterra equations, the goal is to find how the populations will evolve in time given the population sizes at a initial time. Without the initial population sizes, the problem does not correspond to the underlying biological problem. This leads to the notion of initial value problem (IVP), defined in Definition 2.1.3.

Definition 2.1.3 (IVP)**.** *Considering* $(t, \eta) \in D$*, solving the IVP associated to Equation* (2.1) *and initial condition:*

$$u(t_0) = \eta \tag{2.3}$$

consists in finding a solution (J, u) *of Equation* (2.1)*, such that* $t_0 \in J$ *and* u *satisfies Equation* (2.3)*.*

In some cases, existence and/or uniqueness can be guaranteed, with the Peano existence theorem and the Cauchy-Lipschitz theorem. These theorems can be applied in most cases and can be found in Peano and Peano (1990); Legendre (2018).

Remark 2.1.2 (System of ODE)**.** *An ODE of order* k *is written as:*

$$\frac{d^k u}{dt^k}(t) = f(t, u(t), \frac{du}{dt}(t), ..., \frac{d^{k-1}u}{dt^{k-1}}(t)) \tag{2.4}$$

However, we only focused on first-order ODEs in this section. This is due to the fact that higher orders ODEs can be written down as first-order ODEs. Indeed, by defining $y := (u, \frac{du}{dt}, ..., \frac{d^{k-1}u}{dt^{k-1}})$*, Equation* (2.4) *can be written as* $\frac{dy}{dt}(t) = F(t, y(t))$*, with* $F := (y_2, y_3, ..., y_k, f(t, y_1, ..., y_k))$*.*

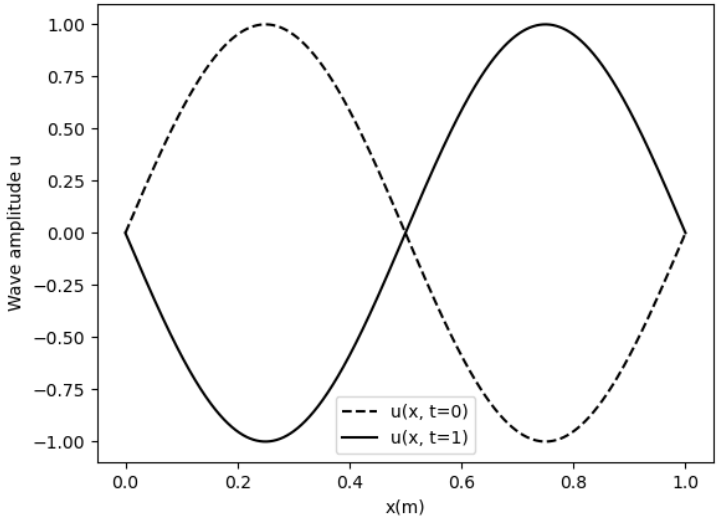


Figure 2.2: Example of a 1D wave equation solution with boundary conditions.

PDE PDEs are a more general family of equations, that include ODEs. They include spatial derivatives in addition to time derivatives, instead of only time derivatives for an ODE, and are defined in Definition 2.1.4. An example of a phenomenon described by a PDE is the propagation of waves in space. As can be seen in Figure 2.2, wave propagation seems pretty simple. The applications covered by such a type of equation are various, ranging from telecommunications, oceanography and seismology. Among them, music is an interesting example. Indeed, understanding how a string vibrate allows us to understand how different notes emerge from different pulsations of the wave. This gave rise to a consensus on the vibration which gives the A note, with a frequency of 440 Hz. However, putting this physical behavior into an equation is more complex, as presented in Example 2.1.3.

Definition 2.1.4 (Partial differential equations)**.** *Considering $x \in \mathbb{R}^d$ and $t > 0$, a PDE is defined by:*

$$\frac{\partial u}{\partial t}(t,x) + \sum_{j=1}^{d}(A_j(u)\frac{\partial u}{\partial x_j})(t,x) = 0, \tag{2.5}$$

where $u : \mathbb{R}^{d+1} \to \mathbb{R}^p$ is an unknown function and $A_j : \mathbb{R}^p \to \mathbb{R}^p$, $1 \leq j \leq d$, is a known smooth function.

Example 2.1.3 (Wave equation)**.** *The 1D wave equation is defined by:*

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, \tag{2.6}$$

where c is a fixed non-negative real coefficient.

As previously done for ODEs, problems of interest often involve some initial state. For PDEs, this state has a spatial component. This leads to Definition 2.1.5.

Definition 2.1.5 (IBVP). *Considering $t > 0$ and $u_0 : \mathbb{R}^d \to \mathbb{R}^p$, solving the initial boundary value problem (IBVP) associated with Equation (2.5) and the initial condition:*

$$\forall x \in \mathbb{R}^d, u(0, x) = u_0(x) \tag{2.7}$$

consists in finding a solution (J, u) of Equation (2.5), such that $u_0 \in J$ and u satisfies Equation (2.7).

In addition to this initial state, some spatial constraints can be added. In the wave propagation example, the wave is stopped at $x = 0$ and $x = 1$. In the case of a guitar for instance, the string stops at the nut and at the bridge. Without these spatial conditions, the wave would go on forever, which is not suitable to study acoustic. As seen in this example, these constraints are usually crucial to model the right phenomenon. They can be defined in multiple ways. Given the spatial domain Ω, let $\partial\Omega$ be the boundary. A boundary condition is then a condition on u imposed on the boundary $\partial\Omega$. A IVBP with a boundary condition is called a boundary value problem (BVP). There are two main types of boundary conditions which are often used, namely, the Dirichlet boundary condition and the Neumann boundary condition. The former consists in specifying the value of u along the boundary, i.e., for a given $f : \mathbb{R}^d \to \mathbb{R}^p$, $\forall x \in \partial\Omega, u(t, x) = f(x)$. On the other hand, Neumann boundary condition consists in specifying the value of the derivative of u along the boundary, i.e., for a given $f : \mathbb{R}^d \to \mathbb{R}^p$ and with n the unit normal to $\partial\Omega$, $\frac{\partial u(t,x)}{\partial n} = f(x)$.
This thesis tends to focus more precisely on hyperbolic PDEs, defined in Definition 2.1.6, and can be found in the first two chapters, with the Advection, Burgers' and Shallow Water equations. These are a subgroup of PDEs that arise mainly in fluid mechanics, when there is some transport phenomenon. They describe systems in which information spreads at finite velocity, and solutions can be discontinuous. A classic instance of an hyperbolic PDE is the wave propagation equation.

Definition 2.1.6 (Hyperbolic PDE). *Equation (2.5) is hyperbolic on an set $U \in \mathbb{R}^p$ if and only if the matrix $A(u, \alpha) := \sum_{j=1}^{d} \alpha_j A_j(u)$ only possesses real eigenvalues and is diagonalizable $\forall u \in U$ and $\alpha \in \mathbb{R}^d$. Furthermore, if, in addition to these conditions, the eigenvalues of $A(u, \alpha)$ are distinct, Equation (2.5) is strictly hyperbolic.*

In addition to finding the solution of a IBVP, characterizing this solution is a main topic of research. Among this characterizations, the stability of the solution is of great interest and is used in various related works applying numerical analysis concepts to neural networks as presented in Section 2.3.1.2. Stability is defined for an ODE in Definition 2.1.7. For a PDE, its definition depends on the type of equation considered. It follows the same principle as for the ODE, which is to bound the evolution of the solution by the initial value. Since this stability is harder to

define and frequently harder to prove, the Lyapunov stability is often considered. This stability is the stability near a point of equilibrium x_e, i.e. $u(x_e) = 0$. For an ODE, it is defined in Definition 2.1.8. For a PDE, it can defined as well, since the equation around a point of equilibrium can be linearized. More details can be found in Beck (2012). The idea of this stability is that, if the equation has a point of equilibrium, if the solution is close to it, then it must remain close to it. There are other definitions of this stability which enforces the solution to converge to this equilibrium (asymptotically stable) and control its convergence (exponentially stable).

Definition 2.1.7. *(Stability of the solution of a IVP) A solution of a IVP, as defined in Definition 2.1.3, is stable if and only if, there exists $M \in \mathbb{R}$ such that:*

$$\|u(t) - \tilde{u}(t)\| \leq M\|u(0) - \tilde{u}(0)\|,$$

with u and $\tilde{u}$ being two trajectories with different initial conditions.

Definition 2.1.8 (Lyapunov stability of a IVP)**.** *For a IVP, as defined in Definition 2.1.3, Lyapunov stability is defined by:*

$$\forall \epsilon, \exists \delta, \text{ if } \|x(0) - x_e\| \leq \delta, \text{ then, } \|x(t) - x_e\| \leq \epsilon.$$

2.1.2 Numerical schemes

In order to predict dynamical systems, deriving analytical solutions of PDEs is often not possible. To overcome this challenge, numerical analysis focuses on numerically solving PDEs with algorithms called schemes. The prediction with these schemes can be particularly difficult as illustrated in the Lorenz system of equations, presented in Example 2.1.4, where the solution can be chaotic and never periodic. This equation was originally derived to model atmospheric convection, and is a well-known example of an intractable chaotic system. As shown in Figure 2.3, which is obtained with a numerical scheme, they can compute good approximations of solutions in difficult cases that cannot be found analytically. However, by introducing numerical errors, which may be truncation or discretization errors, the solution obtained with a numerical schemes can be poor if small difference in initial conditions lead to big difference at long horizons and the discretization size is not chosen to be small enough. In this section we detail the main designs of schemes for ODEs then PDEs before diving into the desired theoretical properties of schemes.

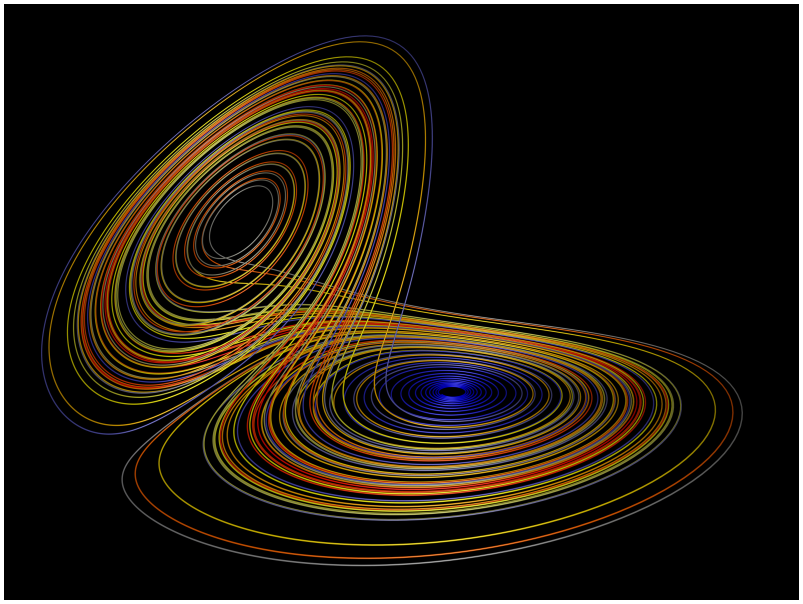


Figure 2.3: Example of a Lorenz system numerically obtained solution from a given angle.

Example 2.1.4 (Lorenz system)**.** *The Lorenz system of equations is defined by:*

$$\begin{aligned}\frac{\mathrm{d}u_1}{\mathrm{d}t} &= \sigma(u_2 - u_1),\\ \frac{\mathrm{d}u_2}{\mathrm{d}t} &= u_1(r - u_3) - u_2,\\ \frac{\mathrm{d}u_3}{\mathrm{d}t} &= u_1u_2 - u_3.\end{aligned}$$

With $u(0) = (-8, 8, r-1)$*,* $\sigma = 10$*,* $r = 28$ *and* $b = 8/3$*, the solution of the associated IVP is chaotic and never periodic.*

Numerical schemes for ODEs To compute an approximation of the solution of Equation (2.1), the interval $[t_0, T]$, where t_0 is the initial time and T the final time of the solution, must be sub-divided into smaller intervals. Each interval is denoted by $[t_n, t_{n+1}]$, with $n \in [0, N]$, N being the number of sub-divisions. The step for a given interval is defined by $h_n := t_{n+1} - t_n$. The aim is to offer a good approximation u_n of $u(t_n)$. The most simple and well-known method is Euler method, defined by $u_{n+1} = u_n + hf(t_n, u_n)$, with h being fixed. More generally, Runge-Kutta (RK) methods are a well-known class of numerical schemes including Euler scheme and

are used in Chapter 3 and Chapter 4. A RK scheme with s stages is defined by:

$$\begin{aligned} k_n^1 &= f(t_n, u_n) \\ k_n^2 &= f(t_n + c_2 h, u_n + h a_{2,1} k_n^1) \\ k_n^3 &= f(t_n + c_3 h, u_n + h(a_{3,1} k_n^1 + a_{3,2} k_n^2)) \\ &... \\ k_n^s &= f(t_n + c_s h, u_n + h(a_{s,1} k_n^1 + ... + a_{s,s-1} k_n^{s-1})) \\ u_{n+1} &= u_n + h(b_1 k_n^1 + ... + b_s k_n^s) \end{aligned} \tag{2.8}$$

with c_i, $a_{i,j}$ and b_j being fixed coefficients.
Among this class, another widely used scheme is RK4, defined by $c = (0, 1/2, 1/2, 1)$, $a_{0,} = (1/2, 0, 0)$, $a_{1,} = (1/2, 0)$, $a_{3,} = (1)$ and $b = (1/6, 2/6, 2/6, 1/6)$. An important property of numerical schemes is their order, it gives an indication on the convergence rate of the method and is properly defined in Definition 2.1.9.

Definition 2.1.9 (Order of a scheme for an ODE)**.** *Equation* (2.8) *is of order p if, for each IVP problem, $u_1 - u(t_0 + h) = \mathcal{O}(h^{p+1})$, for $h \to 0$.*
$u_1 - u(t_0 + h)$ is called the local error of the method.

It can then be proven that Euler method has an order of 1 and RK4 an order of 4. The greater the order, the quicker the convergence, which is very desirable. There exists also methods of order 5, with DOPRI5. However, the higher the order the higher the computational cost, so there is a trade-off between cost and convergence rate.
In order to solve real-world problems, methods with fixed step size are usually inefficient. Indeed, some local phenomena can occur, which require more points around those times. With a fixed step size, there is hard trade-off:

(i) The step is chosen to be very small. Then the scheme can capture these high-frequency phenomena but the computational cost is high and when these high-frequency phenomena do not occur, there is no need for such a fine resolution time grid.

(ii) The step is not chosen to be very small. The computational cost is not prohibitive but then the high-frequencies phenomena are not captured by the scheme, leading to poor results.

Then, a natural solution is to adapt the step size so that the error is the same everywhere. That is what almost all realistic schemes do. We do not detail these schemes nor the algorithm selection for the step sizes since it is out of the scope of this thesis. Some adaptive step schemes are used, especially in Chapter 4, but are not a main part of the different works presented here. For more details on adaptive schemes, see Hairer and Abdulle (2001).

In order to compute better approximations, another natural idea is to use more than one previous step to predict the next one. RK methods are one-step method, but multi-steps methods are also quite popular, with the main ones being Adams methods. In a general form, they are written as $u_{n+1} = u_n + h\sum_{j=0}^{k-1}\gamma_j\Delta^j f_n$, where $\Delta f_n = f_n - f_{n-1}$, $f_n := f(t_n)$ and $\gamma_j := \frac{1}{j!}\int_0^1 \prod_{i=0}^{j-1}(i+s)ds$. For $k = 1$, it corresponds to Euler method, and for $k = 2$, the derivation leads to $u_{n+1} = u_n + h(\frac{3}{2}f_n - \frac{1}{2}f_{n-1})$. As powerful as they are, these methods, and RK methods as well, can exhibit stability issues. A common notion used to study numerical schemes for ODEs is the A-stability, as defined in Definition 2.1.10. It consists in studying the stability of the scheme on the test equation, defined by:

$$\begin{aligned} \frac{du}{dt} &= \lambda u \\ u(0) &= 1, \end{aligned} \tag{2.9}$$

where $\lambda \in \mathbb{C}$.

Definition 2.1.10 (A-stability)**.** *A numerical scheme is said to be A-stable or absolutely stable if and only if its solution u_n to Equation (2.9) is bounded where the equation is bounded, i.e. if it respects $\{z \in \mathbb{C}; Re(z) \leq 0\} \subseteq S$, where $z := h\lambda$, $h \in \mathbb{R}$ and $S := \{z \in \mathbb{C};\ (u_n)_{n\geq 0}$ is bounded $\}$.*

The restriction $Re(z) \leq 0$ is due to Equation (2.9) not being bounded otherwise. A-stability is an interesting characterization of schemes, especially due to the fact that non-linear problems are usually solved by linearizing around a point of equilibrium. This stability is used in related works applying numerical analysis concepts to neural networks as presented in Section 2.3.1.2. However, this stability does not correspond to the stability of the true problem, because the scheme is used on a different equation than the test equation. Stability of schemes is defined later in Definition 2.1.16.

A study of previously described schemes leads to the conclusion that none of them is A-stable (Hairer and Abdulle, 2001). For instance, Euler explicit is not A-stable because its stability set S is a disk of center -1 and radius 1. This inspires the design of implicit schemes, which are A-stable. An implicit scheme is a scheme that implies to solve an equation $G(u_{n+1}, u_n, ..., u_{n-k}) = 0$ at each step. Similarly to explicit Adams methods, implicit Adams methods are constructed by approximating an integral. Among them, the most used implicit scheme is implicit Euler scheme, defined by $u_{n+1} = u_n + hf(t_{n+1}, u_{n+1})$. Solving $G(u_{n+1}, u_n, ..., u_{n-k}) = 0$ is usually not possible analytically and is done numerically, by writing it down as a minimization problem or a root-finding problem. Guaranteeing the A-stability property hence leads to a higher computational cost, the one of solving an equation at each time step.

In addition to A-stability, another common studied stability is the zero-stability, as

defined for one-step schemes in Definition 2.1.11. It consists in characterizing the accumulation of perturbations for the discretization size going to 0, when solving a IVP and is used in some related works presented in Section 2.3.1.2.

Definition 2.1.11 (Zero-stability)**.** *Given a one-step scheme* $u_{n+1} := u_n + h_n\phi_f(u_n)$*, where* ϕ_f *is a continuous function that defines the scheme, the perturbed scheme associated is* $u^{pert}_{n+1} := u^{pert}_n + h_n(\phi_f(u^{pert}_n) + \epsilon_n)$*, with* ϵ_n *the perturbations. The initialization of the IVPs associated to the schemes are* u_0 *and* u^{pert}_0*. The one-step scheme is said to be zero-stable if and only if there exists* C*, independent of the discretization, such that:*

$$\forall N \in \mathbb{N}^* \max_{0\leq n\leq N} |u_n - u^{pert}_n| \leq C(|u_0 - u^{pert}_0| + \max_{0\leq i\leq N-1} |\epsilon_i|)$$

Numerical schemes for PDEs In a more general form, numerical schemes are used to solve PDEs as well. Finite difference method schemes are used and studied in some form in every chapter of this thesis, and have been already presented for the ODE case, RK and Adams methods being finite difference methods.

Definition 2.1.12 (Numerical scheme)**.** *A numerical scheme is a numerical method approximating* u *by discretizing the domain of interest. One of the most commonly used families of schemes is the finite difference method which approximates* u *with* $u^{n+1}_j = H(U_{n+1}, U_n, ..., U_{n-k}) \approx u(t_{n+1}, x_j)$*, where* $U_n := (u^n_{j-k}, ..., u^n_{j+k})$ *and* H *is a continuous function.*

If there is no dependency of H on U_{n+1}, the scheme is then explicit. From Definition 2.1.12, the definition of numerical schemes for ODEs can be retrieved, by removing the dependency on space. Schemes for PDEs are harder to design because they need a spatial and a time discretization step. As will be seen later, some trade-offs need to be made between the two.
Numerical schemes are designed to converge on the problems they are applied to. In order to guarantee convergence, there are certain cases where two other properties are enough. These two properties are consistency and stability.

Consistency Consistency is a notion that depends on the local truncation error of the method, defined in Definition 2.1.13. This error is obtained by inserting the true solution of the considered equation in the scheme. This supposes that the solution of the equation is known, so this is intended as an analysis tool to study numerical schemes. Consistency then consists in the error of the scheme being asymptotically null when the true solution is inserted in the scheme, as defined in Definition 2.1.14. Δt corresponds to the fixed time step and Δx to the fixed space step.

Definition 2.1.13 (Local truncation error)**.** *The local truncation error* ϵ^{n+1}_j *of a method is defined by* $\epsilon^{n+1}_j := u(t_{n+1}, x_j) - u^{n+1}_j$*, where* u^{n+1}_j *is computed with* $U_n := (u(t_n, x_{j-k}), ..., u(t_n, x_{j+k}))$*.*

Definition 2.1.14 (Consistency). *A numerical scheme is consistent if and only if* $\lim_{\Delta t, \Delta x \to 0} \frac{\epsilon_j^{n+1}}{\Delta t} = 0$.

The order of a scheme, defined in Definition 2.1.9 for an ODE, is linked with the notion of consistency. Indeed, the order of a scheme is the convergence rate of the scheme. This link is clearly seen in Definition 2.1.15, which defines the order of a scheme for a PDE.

Definition 2.1.15 (Order of a scheme for a PDE). *A scheme is said to be of order* p *in time and* q *in space,* $p, q \in \mathbb{N}^*$, *if:*

$$\frac{\epsilon_j^{n+1}}{\Delta t} = O(\Delta t^p) + O(\Delta x^q), \text{ for } \Delta t, \Delta x \to 0 \tag{2.10}$$

Furthermore, if Δt *and* Δx *are linearly dependent, i.e.* $\exists k, \Delta t = k\Delta x$, *a scheme is said to be of order* p *if:*

$$\frac{\epsilon_j^{n+1}}{\Delta t} = O(\Delta t^p), \text{ for } \Delta t \to 0 \tag{2.11}$$

Stability In addition to consistency, stability is another property of great importance and is a foundational part of Chapter 3. A-stability has already been defined in Definition 2.1.10 for numerical schemes for ODEs. This is a special case of stability. In a general way, the stability of a scheme means that the error is not amplified during time iterations, as defined in Definition 2.1.16. This notion is close to the stability of a dynamical system, defined in Definition 2.1.7. The stability of a system is a property of the system to study, but the stability of the scheme is a condition for the scheme to better solve the system. Hence, when numerically designing numerical methods to solve PDEs, the stability of a scheme is of great practical interest.

Definition 2.1.16 (Stability). *A numerical scheme solution* $(u_n)_{n\in\mathbb{N}}$ *of dimension* M *is stable in norm* L^p *if there exists for a time* T, $C(T)$ *independent of the time discretization step* Δt *such that:*

$$\forall\ u^0 \in \mathbb{R}^M,\ n \geq 0;\ n\Delta t \leq T,\ \|u^n\|_p \leq C(T)\|u^0\|_p\ .$$

Convergence Stability of a scheme is ensuring that the scheme does not amplify errors by bounding its evolution by a value depending on the initial value. The ultimate goal in the design of a scheme is then to guarantee its convergence.

Definition 2.1.17 (Convergence). *Given the global error* $E_\Delta(t_n, x_j) := u(t_n, x_j) - u_j^n$ *defined over a grid discretized with a step* Δ, *a scheme is said to be convergent if and only if,* $\forall t_n \in \mathbb{R}, x_j \in \mathbb{R}^M, E_\Delta(t_n, x_j) \xrightarrow[\Delta \to 0]{} 0$ *and* $\forall u_0 \in \mathbb{R}^M, x_j \in \mathbb{R}^M, u(0, x_j) \xrightarrow[\Delta \to 0]{} u_0$.

Convergence is a property often impossible to prove. However, in some cases, stability and consistency can lead to convergence, as stated in Theorem 2.1.1.

Theorem 2.1.1 (Lax equivalence theorem)**.** *A necessary and sufficient condition for the convergence of an approximation to the solution of a well-posed linear problem problem by a consistent finite-difference method is that the method is stable. A proof can be found in Tekriwal et al. (2021).*

In order to avoid convergence problems, it is important to carefully select the space and the time discretization. This is illustrated by the Courant–Friedrichs–Lewy (CFL) condition. This is a necessary condition for the convergence of a scheme. It arises from a Von Neumann analysis, which is a stability study using Fourier series. The Fourier series is performed on the error of the scheme. Studying the norm of this error can naturally lead to a Fourier analysis to bound this error. The CFL condition constrains the time step and the space step. For instance, the CFL condition for a linear equation $\frac{\partial u}{\partial t}(t,x) = -a\frac{\partial u}{\partial x}(t,x)$, with $a \in \mathbb{R}$ and using a 3-step method, is $|a|\frac{\Delta t}{\Delta x} \leq 1$. In this thesis, the CFL condition is applied in a PDE solving scheme in Chapter 4.
PDEs can exhibit particular behaviors that impact the design of numerical schemes. Among these behaviors, one of the most encountered is the multi-scale property. This happens when the PDE exhibits phenomena that occurs at different scales. For instance, when modeling an airfoil with Reynolds-Averaged Navier-Stokes equations, some high-frequency turbulence occurs around the surface of the airfoil, contrasting with the low-frequency phenomenon of the airplane moving. This is particularly challenging to predict since the scheme must capture both phenomena, and can be very costly since a very fine mesh is needed to capture high-frequency phenomena. A family of methods designed for these problems is called multigrid methods. They use a hierarchy of grids in order to reduce the computational cost. Among these methods, three are particularly studied and are applied to neural network architectures in Chapter 3:

- *V-cycle*: It typically starts at the coarsest grid level and proceeds towards finer grids. After reaching the finest grid, the process reverses, and corrections are propagated back to the coarser grids.

- *F-cycle*: Similar to the V-cycle, it starts at the coarsest grid level and proceeds to finer grids. However, it differs in that it performs additional V-cycles at each grid level before proceeding to the next finer level. This increased level of correction can lead to more accurate solutions but requires more computational resources.

- *W-cycle*: It is an extension of the V-cycle. Like the V-cycle, it starts at the coarsest grid and works its way to the finest grid. At each level, it performs

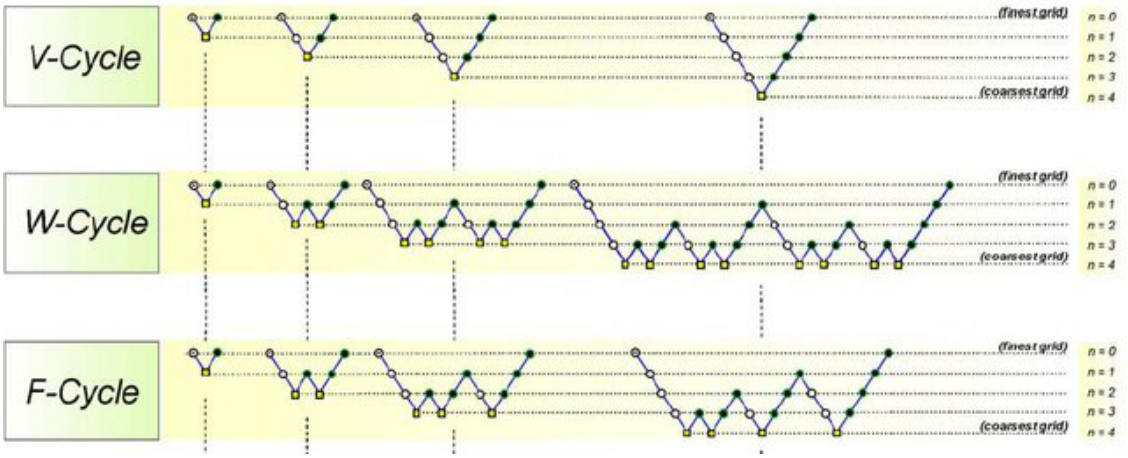


Figure 2.4: Example of an V-, F- and W-cycles numerical schemes for different levels. Taken from Cordazzo et al. (2007).

multiple V-cycles (typically two or more) before proceeding to the next finer level.

These schemes are illustrated in Figure 2.4. More details on this topic can be found in John (2013).

2.2 Deep Learning

Machine learning is the field of solving problems by designing algorithms that use data to learn highly non-linear mappings. Inside this field, DL is the study of algorithms with many layers, a layer being a block of parameters of the method. These algorithms are used to solve various types of problems, but in this thesis we focus on supervised learning, as defined in Definition 2.2.1.

Definition 2.2.1 (Supervised learning)**.** *Supervised learning consists in learning a parametrized mapping* $f : \mathcal{X} \times \Theta \to \mathcal{Y}$ *from a finite training set composed of input values* $X \in \mathcal{X}$ *and output values* $Y \in \mathcal{Y}$*, where* $\mathcal{X}$ *is the input space,* $\mathcal{Y}$ *is the output space and* Θ *is the parameter space. For simplicity of notation,* $f_\theta(x) := f(x, \theta)$*. What is called learning is the update of the weights* θ *according to an optimization problem using the training set. A classic optimization problem is to find* θ^* *such that* $\theta^* := \arg\min_{\theta \in \Theta} \|f_\theta(X) - Y\|^2$*. This learning contrasts with unsupervised learning, where no output values are available to train the model.*

The goal of DL algorithms is to generalize to unseen data, this is a fitting problem, not an interpolation problem. That is why the data is usually split into a training set X_{tr}, Y_{tr} and a test set X_{ts}, Y_{ts}. The training set is then also split into a training set and a validation set. The latter is used to verify at each step of the

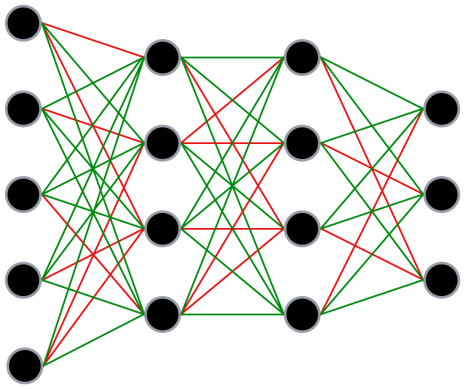


Figure 2.5: Example of a MLP with 4 layers.

training if the network is over-fitted. The test set is not used during the training and is crucial to see if the network has been properly trained, since the best θ on the training set is almost never the best one on the test set. The span of application of DL methods is huge, with one of the first breakthrough being the classification of images with residual networks (He et al., 2016). However, in this thesis, we focus on regression problems, which imply to predict one or several continuous variables Y from a set of input variables X. In this section, we first present two common DL architectures that are used in this thesis, before describing the training process of DL methods.

2.2.1 Common architectures

Many DL architectures have been developed and each one is designed for a specific problem. The design of DL architectures is one of the main areas of research in the community and is a challenging task. Different categories of architectures can be distinguished. Inside each of these categories, a lot of different choices must be made. These choices are often crucial for the algorithms to perform well. These are not thoroughly discussed here, as they depend on the problem at hand and hence are discussed within each work. In this section, we describe two main widely used architectures, multilayer perceptron (MLP) and convolutional neural network (CNN).

MLP A MLP, or feedforward articifial neural network, is a network consisting of several layers fully connected by weights and a non-linear activation function. An illustration of a MLP is shown in Figure 2.5. A MLP can be written as $f_\theta = \sigma_l \circ L_l \circ ... \circ \sigma_1 \circ L_1$, where σ_i is a local element-wise activation function and L_i is an affine transformation $x \mapsto W_i x + b_i$. An activation function is a non-linear function. This architecture is inspired by biological neural networks, where many neurons are

connected by synapses to each other. The information is then transmitted by firing some neurons. This leads to the construction of the affine transformation which links all the neurons by the weights W and the activation function which forces the neuron to be fired or not. This inspired the popular choice of the ReLU activation function, defined by $\text{ReLU}(x) := \max(0, x)$, which forces the network to let the information pass or not. This function is null for negative values, which led to the design of the Leaky ReLU function. This function is defined by $\text{LeakyReLU}(x) = \max(0, x) + \lambda \min(0, x)$, where λ is a hyper-parameter to tune. Another common choice for the activation function is the hyperbolic tangent tanh. The design of activation functions is an active field of research, even though ReLU and tanh functions remain very popular, in part due to their simplicity and their proven efficiency. This architecture is used in Chapter 3 and Chapter 5.

CNN CNN architectures were firstly designed for vision problems, for e.g. image classification. They include biases that favors their application to some problems. One well-known bias is the shift-equivariance, i.e., a shift of the input to a convolutional layer produces a shift in the output feature maps by the same amount. The global idea of a CNN is to apply convolutional filters to input images, which activates certain features from the images. Different variations and improvements were added to the original idea, but a vanilla 1D CNN, with an input X of size (C_{in}, L_s) and a kernel K of size K_k, can be defined by:

$$\begin{aligned} \text{CNN} &:= avgpool \circ \sigma \circ conv \circ ... \circ \sigma \circ conv \\ \forall j \in [|1, C_{out}|], conv(X, K)_j &:= (B)_j + \sum_{k=0}^{C_{in}-1} (K)_{j,k} \star X_k, \end{aligned} \tag{2.12}$$

where $\star$ is the cross-correlation, B is a bias vector of size C_{out}, with (C_{out}, L_s) being the size of the output of a convolutional layer and *avgpool* being an average over all dimensions in order to output a single value. Given the problem, this last layer can be changed and a lot of different options other than an average pooling are available. Similarly, a normalization is usually added in order to improve the training stability. There are different types of normalization as well, layer normalization, weight normalization and batch normalization being the main ones. In addition to this layer, a padding layer can be added. Indeed, depending on the size of the image and the kernel, this image may need to be augmented by zeros in order to apply the kernel to it.
In order to avoid training issues, an important development in DL algorithms, and especially CNNs, is the residual neural network (ResNet, He et al. (2016)). It consists in adding a skip connection to the regular layer, for e.g. for a CNN, $CNN_{res} := avgpool \circ \sigma \circ (conv + L) \circ ... \circ \sigma \circ (conv + L)$, with L a linear layer. Usually skip connections are implemented every few layers and not every layer. This architecture

is used in Chapter 4.

GNN A graph neural network is a DL architectures which takes as input a graph. This makes this architecture very interesting when tackling problems with complex geometries. An example of graph that can be used for representing meshes encountered in PDEs applications is an euclidean graph whose nodes are the points in a domain Ω and whose edges have a length equal to the euclidean distance between those points. An adjacency matrix is a matrix containing values of 1 when two nodes are neighbors and values of 0 of when two nodes are not neighbors. It can be constructed from the edges lengths, with for instance all the points being within a sphere of radius r and a center the point x considered. A GNN then consists in updating the graph with some functions. A common architecture is the Message Passing Neural Network. A passage from one layer to another consists in:

$$\mathbf{h}_u = \phi(\mathbf{x}_u, \bigoplus_{v \in N_u} \psi(\mathbf{x}_u, \mathbf{x}_v, \mathbf{e}_{u,v})),$$

where $\mathbf{x}_u$ is the node to update, $(\mathbf{x}_v)_{v \in N_u}$, are the neighboring nodes of u, $\mathbf{e}_{u,v}$ the edges from u to v, $\bigoplus$ a permutation invariant aggregation operator that can accept an arbitrary number of inputs, for e.g. element-wise sum, and ϕ and ψ are two functions, referred respectively as the update and message functions. The idea of this architecture is to update the values of the nodes using information from neighboring nodes. Many other architectures of GNN exist, for more details see Bronstein et al. (2021). An important properties of many GNN architectures is the permutation equivariance, i.e. for the node-wise function $\mathbf{F}$, the graph $\mathbf{X}$ and the adjacency matrix $\mathbf{A}$, $\mathbf{F}(\mathbf{PX}, \mathbf{PAP}^\top) = \mathbf{PF}(\mathbf{X}, \mathbf{A})$ for any permutation matrix $\mathbf{P}$. However, a drawback of these methods is their sensitivity to the neighbors, for instance they may not be efficient if trained on a dense neighborhood and tested on a sparse one. This strong dependence to the sampling can limit their applicability and generalization capacities in some problems, especially with complex PDE, where the sampling can vary a lot between examples with different geometries. This type of architecture is used in various related work which solve PDEs as presented in Section 2.3.2 and is a implemented as a baseline in Chapter 5.

2.2.2 Training

In order to update the weights of the neural network, gradient descent types algorithms are used. First a loss $\mathcal{L}$ must be chosen for the optimization problem. For regression problems, due to its smoothness, a mean squared error (MSE) loss is often used, $\text{MSE}(x, y) := \|x - y\|^2$. Then, back-propagation is used to compute the gradients of the weights. It consists in a backward pass of the loss in order to adjust the parameters. More details on this topic can be found in Rumelhart et al. (1986).

A gradient descent, $\frac{d\theta}{dt} = -l_r \nabla_\theta \mathcal{L}(f_\theta, X_{tr}, Y_{tr})$, with l_r the learning rate, can then be applied to train the model. When the dataset is large, which is almost always the case, stochastic gradient descent (SGD) is performed. It consists in dividing the data into batches and doing the updates on these batches and not the entire training set. More advanced methods are used in practice, with the most popular one being Adam (Kingma and Ba, 2014).
In addition to this main framework of training, many different techniques are used to improve the training of neural networks. Among these numerous techniques, we mainly use three in this thesis:

- *Scheduling*: It consists in adapting the learning rate to the training process. Usually, at the beginning of the training, it is beneficial to explore more the parameter space, which means having a high learning rate l_r. However, when the network improves during training, it becomes more interesting to reduce the variance of the SGD by having a lower l_r. Many different scheduling are used, the linear scheduling being very popular. It consists in updating l_r by γl_r every n_{steps} steps.

- *Weight decay*: It consists in changing the loss by adding $\lambda \|\theta\|^2$, with λ a hyper-parameter usually chosen to be very small. Indeed, the goal is to stabilize the training, not to change the optimization objective.

- *Initialization*: Choosing a good initialization of the parameters is a crucial area of research in DL. Many different methods have been developped. A widely used one is Xavier initialization (Glorot and Bengio, 2010), which for the uniform scheme and a linear layer, consists in drawing the weights W of the layer from $\mathcal{U}(-a, a)$, where $a := gain\sqrt{\frac{6}{f_{in}+f_{out}}}$, f_{in} and f_{out} being respectively the number of inputs and outputs and $gain$ a hyper-parameter, and the biases are set to 0.

2.3 Deep Learning and Numerical Analysis

The study of DL and numerical analysis is a vast and developing field of research. Many links have been found between the two domains. In this section, we will outline two main directions:

(i) Applying numerical analysis ideas to DL approaches.

(ii) Using DL methods to solve PDEs.

2.3.1 Numerical schemes for Deep Learning

Numerical analysis is a vast field with many different results. Applying ideas from this field to DL has been done in various ways. In this thesis, we focus on integrating

numerical schemes into DL methods, which became popular with the Neural ODE framework (Chen et al., 2018). However, even if not discussed here, many other ideas have been applied to DL, such as SDE-Net (Kong et al., 2020), which provides uncertainty quantification based on a stochastic differential equation, or Hamiltonian neural networks (Greydanus et al., 2019) which leverage Hamiltonian mechanics to add biases to solve physical systems and be reversible in time.

2.3.1.1 Neural ODE

The first links between numerical schemes and neural networks have been outlined with the ResNet architecture, presented in Section 2.2. Indeed going from layer to the next of a convolutional ResNet can be written as $x_{n+1} = conv(x_n, K_n) + W_n x_n + b_n$, which is an Euler explicit scheme formula solving $\frac{dx}{dt} = f(t, x(t))$ with $f(t, x(t)) := conv(x(t), K(t)) + b(t) + W(t)x(t) - x(t)$, with initial condition x_0 being the input of the neural network. The most popular work using this link is Neural ODE (Chen et al., 2018), which then provides numerous architectures based on the fact that other schemes can be used to solve the underlying PDE of a residual network. This leads to the formulation of the network $\text{NeuralODE}(z(t_0)) := \text{ODESolve}(z(t_0), f, t_0, t_1, \theta)$. This formulation entails a flexible architecture, especially by using adaptive steps schemes and higher order schemes such as RK4. However, this work has its limitations. Indeed, it was shown that this initial architecture could not learn simple functions and that the flow induced by the initial formulation was not simple. To overcome this challenge, the solution is to augment the space on which the ODE is solved (Dupont et al., 2019). Still with this improvement and with many follow-up works, the framework remains hard to use and not always very efficient. In the same spirit, Deep Equilibrium (DEQ) models are more effective (Bai et al., 2019). They imply to solve a fixed-point equation to obtain the network, i.e., $\text{DEQ}(x) = f_\theta(\text{DEQ}(x), x)$. This formulation is easier to use, is more memory efficient than Neural ODE, and has become quite popular.

2.3.1.2 Stability

Given the established links between numerical schemes and neural networks, many concepts can be applied to neural network architectures. Among these concepts, stability is of great interest and is a focus of this thesis and especially of Chapter 3. Stability is presented through three angles in this section; robustness, which consists in controlling the Lipschitz constant of the network, continuous stability, which consists in applying the stability of a differential equation to neural networks, and numerical schemes stability.

Robustness Robustness analysis is closely associated with the examination of a network's Lipschitz constant, primarily in the context of adversarial examples.

It was first shown that minor additive perturbations to input data could result in significant output perturbations at the final network layer, presenting challenges in managing adversarial scenarios (Szegedy et al., 2013). This observation bears similarities to stability definitions in schemes and differential equations, although the Lipschitz constant was chosen as the primary approach. Robustness analysis has then become wider than the study of Lipschitz constant, with a range of adversarial attacks to overcome, and its evaluation is a topic of research (Carlini and Wagner, 2017; Hendrycks and Dietterich, 2019). We focus only on the Lipschitz constant in this section, being the main link between robustness analysis and numerical analysis. Studying the Lipschitz constant L_k of each layer k leads to the following equations:

$$L := \prod_{k=1}^{K} L_k$$
$$\|f_\theta(x) - f_\theta(x+r)\| \leq L\|r\|,$$

with f_θ the whole network.
A high Lipschitz constant implies that the network's output can lead to significant perturbations, highlighting the need to control this constant for robustness. However, estimating the Lipschitz constant can be challenging, and various research efforts (Bartlett et al., 2017; Balan et al., 2018; Weng et al., 2018; Scaman and Virmaux, 2018; Zou et al., 2019; Latorre et al., 2020; Jordan and Dimakis, 2020; Kim et al., 2021) have aimed to provide upper bounds. Notably, Fazlyab et al. (2019); Drenkow et al. (2021) offer a comprehensive review of the literature.
Beyond estimation, controlling this bound is essential, as demonstrated in more recent work (Bungert et al., 2021), which integrates the constraint into training and addresses the global Lipschitz constant, departing from per-layer considerations. In the same spirit, Gouk et al. (2021) enforce the Lipschitz constraint with a constrained optimization problem, while Pauli et al. (2021) vary the training procedure and Leino et al. (2021) uses an efficient architecture. However, controlling the Lipschitz constant can harm the expressiveness of the network (Zhang et al., 2022a).
These concepts find application in analyzing the stability, particularly in terms of robustness, of recurrent networks, as evidenced in studies such as Miller and Hardt (2018); Bonassi et al. (2020). They apply these principles to diverse problems solved using recurrent networks, like language modeling and slot-filling, focusing on LSTM and GRU networks, delving into Input-to-State Stability and Incremental Input-to-State Stability properties, which ultimately impose constraints on network weights.
Robustness analysis has also been applied to ResNet architectures, with for e.g. invertible ResNets (Behrmann et al., 2019). This field, born from the challenges posed by adversarial examples, has significantly improved the robustness of neural networks. However, it's worth noting that the Lipschitz constant, although valuable,

offers a worst-case perspective on stability and doesn't exploit the interpretation of neural networks as dynamic systems, it is a functional approach.
Other successful approaches have tackled robustness without using the Lipschitz constant, mainly through different training strategies (Zheng et al., 2016; Raj et al., 2020). Not using explicit Lipschitz constants, Zhang et al. (2022b) bounds the forward and backward processes, which is close to the Lipschitz analysis as well as the stability definition. This work is motivated by stabilizing the network and compete with methods such as batch normalization.

Continuous stability Though robustness analysis has seen many applications in adversarial contexts, works exploiting the link between neural networks and dynamical systems have tried to provide more theoretically grounded robust architectures. This is in fact where the first links between the two notions were developed, before the Neural ODE framework.
The first concept applied to neural networks was continuous stability, i.e. the stability of the solution of a PDE as defined and described in Section 2.1.1. It is still what is most studied in the literature. For instance, foundational works by Haber and Ruthotto, such as Haber and Ruthotto (2017); Chang et al. (2018, 2019), establish the importance of continuous stability by employing a key condition based on the eigenvalues of the Jacobian matrix $J(t)$ of the differential equations solved within neural networks: $\max_{i=1,\dots,n} Re(\lambda_i(J(t))) \leq 0, \forall t \in [0, T]$. This condition is also applied in generative modeling scenarios (Kaltenbach and Koutsourelakis, 2021). Haber and Ruthotto continue to emphasize such stability concepts in Ruthotto and Haber (2019), employing directly the definition of stability of PDEs.
Beyond these foundational papers in deep learning and numerical analysis, numerous other studies delve into continuous stability. For instance, Rothauge et al. (2019) examines the continuous problem but employs linearization techniques for in-depth analysis, particularly concerning ResNets.
Meanwhile, Manek and Kolter (2020); Lawrence et al. (2021) introduce network models that leverage Lyapunov theory to ensure stability, thereby enabling the learning of stable dynamics. In a similar vein, Massaroli et al. (2020); Matusik et al. (2020) also focus on Lyapunov theory, emphasizing asymptotic stability as described in Section 2.1.1, with the taking a more mathematical perspective.
Other works, such as Ciccone et al. (2018); Güler et al. (2019), explore the concept of asymptotic stability around an equilibrium point, employing cascades of unrolled nonlinear blocks to study the underlying continuous equations of each block.
In the realm of discrete equations, Mamakoukas et al. (2020) investigates model stability, especially in control-oriented applications like physical robot control. Similarly, Tuor et al. (2020) studies discrete equations, resulting in weight constraints that differ from those obtained in continuous studies, offering optimization strategies during training.

Furthermore, Revay and Manchester (2020) introduces a contraction study, which consists in the model predicting two close trajectories when given two close trajectories. They examine the true model stability rather than stability around an equilibrium. The model then is stable in the sense that it contracts the input, and is also convex, which helps for the optimization under constraints to ensure this stability. This line of work is then closely related to numerical analysis but applied to applied to RNNs, using an input-output stability.
Lastly, in Zhang and Schaeffer (2020), a comprehensive study is conducted starting with the continuous equation analysis, before delving into discrete stability. While this work provides extensive insights into stability, it focuses primarily on ResNets and may not prioritize practical performance improvements.

Stability of schemes Applying stability of schemes to neural networks has also been a subject of exploration, with various approaches. The most common one is A-stability, which assesses stability on the test equation, as defined in Definition 2.1.10. This concept, although not always explicitly stated, is employed in Haber and Ruthotto (2017); Chang et al. (2018, 2019), as they refer to this stability in their analyses. A similar rationale for choosing implicit methods can be observed in Reshniak and Webster (2021), which aligns with the A-stability principle. In a similar vein, Li et al. (2020a) utilizes C-stability, which ensures that the input remains bounded by a constant C less than 1 along with a perturbation δ, focusing on defense against adversarial attacks.
In Chen et al. (2022), zero-stability, as defined in Definition 2.1.11, is applied to residual networks. It is shown that applying this stability allows for better generalization and robustness than applying A-stability for instance. Similarly, zero-stability is used to provide a stability of neural networks for forecasting dynamical systems with graph neural networks (Brandstetter et al., 2022).
In Kim et al. (2020), strong stability preserving high order Runge–Kutta schemes are applied to neural network architectures. These schemes, which are applied to PDEs, consist in preserving the stability of Euler schemes on some problems and designing higher order method from this scheme, hence ensuring stability and high order.
Stability of schemes has also been studied for the discovery of differential equations (Keller and Du, 2021; Du et al., 2022). This is quite a different problem, but the approach is a classic numerical analysis approach, proving consistency and stability before proving convergence of the proposed method.
However, similar to continuous stability, the most precise on applying numerical scheme stability to neural network is Zhang and Schaeffer (2020). It rigorously proves stability for ResNet type architectures, as expressed in Definition 2.1.16, establishing bounds on coefficients. Despite this rigorous analysis, the study remains somewhat confined to the architectures they develop, potentially limiting its broader

applications. Moreover, as mentioned earlier, it doesn't extensively explore the practical implementation of these bounds.
In addition to this work, some recent general work has been done on applying numerical schemes concept to DL (Alt et al., 2023). They present a lot of different contributions. Notably, they apply stability in norm to design a stable CNN architecture. They also translate variants of explicit schemes, implicit schemes and multi-grid scheme to neural network concepts.

2.3.2 Solving PDEs with neural networks

On the other hand of the works applying numerical analysis concepts to neural networks, a main area of research is predicting dynamical systems with neural networks. This field is diverse and has become very popular.

2.3.2.1 Incorporating prior knowledge in DL algorithms

The first foundational works apply neural network to PDEs using prior knowledge. This is still a blooming field, since solving PDEs is a very complex problem, and in order to be used in real-world applications, often need some combination with numerical analysis tools to be efficient.
The main idea of using prior knowledge is to incorporate a neural network inside a numerical scheme to take advantage of the properties of both fields, i.e. robustness of schemes and capacity to learn from data. In this line of work, the study of a neural network inside a warping scheme to predict sea surface temperatures has been a foundational work and helped popularize these ideas (De Bézenac et al., 2019). Following similar ideas, (Thuerey et al., 2021) has developed various methods around the use of differentiable numerical simulations for deep learning; error correction (Um et al., 2020), PDE control (Holl et al., 2020) or inverse problems (Holl et al., 2021). Similarly, graph neural networks (GNNs) were used in combination with a coarse numerical solver to increase the simulation speed (Belbute-Peres et al., 2020). We follow this global line of work in Chapter 4 by combining neural networks and numerical schemes to provide a friction law learned from data for the Shallow Water equations.
On the other hand, prior knowledge can be added during training to constrain the network to respect this knowledge. This led to two closely related works, DGM (Sirignano and Spiliopoulos, 2018) and PINNs (Raissi et al., 2019), which add boundary conditions and other constraints in the loss, thus changing the optimization problem. This line of work is very popular, due to the simplicity of its formulation and its broad range of applications, even though the training of these methods is very instable (Krishnapriyan et al., 2021).
In addition to these works, using neural network inside numerical analysis dimension reduction techniques such as Proper orthogonal decomposition (POD) or Dynamic

Mode Decomposition (DMD) is a popular trend, especially inside the fluid mechanics community (Kutz et al., 2016; Brunton et al., 2020).

2.3.2.2 Learning directly the PDE

While incorporating prior knowledge within DL frameworks is an interesting area of research, which allows for better accuracy, it still faces lots of challenges. For methods combining schemes and neural networks, the challenges posed by numerical simulations remain (computational cost, need of expert knowledge, etc...), while for methods incorporating knowledge in the optimization problems, it was shown that these methods are not very efficient on various problems (Krishnapriyan et al., 2021). Thus, designing methods that can directly solve PDEs is a thriving field. This section presents various methods that attempt this challenging task, focusing on methods that are used in this thesis.

2.3.2.3 GNNs

Among the various DL algorithms, GNNs are a natural fit for PDE applications and are implemented as a baseline in Chapter 5. Indeed, these methods take a graph as input, which is the same format as meshes for numerical simulations. This explains their popularity when it comes to replacing traditional numerical solver by neural network solvers. The first successful method using these algorithms is MeshGraphNet (Pfaff et al., 2021). It is trained to pass messages on a mesh graph and can adapt the mesh discretization during the forward pass. It is tested on various physical system dynamics, including aerodynamics, structural mechanics, and clot and is shown to be 1-2 orders of magnitude faster than the simulation it is trained on. However, as promising this work is, the experiments were not conducted on real-world dataset and the generalization capacities are restricted. Inspired by zero-stability and using GNNs as a core method, Brandstetter et al. (2022) designed an efficient method for predicting dynamical systems, with various techniques.

2.3.2.4 Operators: DeepONet, GNO, FNO

The methods presented so far learn a function whose domain is a real coordinate space. However, when solving PDEs, the operations at stake are operators, i.e. functions whose domains are function spaces. This is then natural to design methods that learn operators to solve PDEs. The first one is DeepONet (Lu et al., 2019). The arguments of using such a framework are based on the universal approximation theorem for operator (Chen and Chen, 1995), which is similar to the universal approximation theorem for neural networks (Leshno et al., 1993). DeepONet is then a DL algorithm that takes as input the values of the input function u at different fixed sensors $s_1, ..., s_m$, which are the same for each input function in this framework, and

the points x on which to evaluate the operator. DeepONet is divided into two parts, a network which takes the input function values as input and a network which takes the coordinate on which to evaluate the operator. The results of these networks is then multiplied to predict the output. This method is tested on different datasets, namely a simple 1D dynamics with 2 cases, a pendulum with an external force and a diffusion-reaction system with a source term. One drawback of DeepONet is the need to have a fixed set of tensors.
This led to the design of other operators, which are implemented in Chapter 3. Among them, Graph Kernel Network (GNO, Anandkumar et al. (2020)) and Multipole Graph Neural Operator (MGNO, Li et al. (2020c)) leverage GNNs to learn operators. The following iterative transformation is applied:

$$v_{t+1}(x) = \sigma\Big(W v_t(x) + b(x) + \int_\Omega K_\phi(x, x')v_t(x')\mathrm{d}\nu(x')\Big) \qquad \forall x \in \Omega, \tag{2.13}$$

where $\theta = \{W, b, \phi\}$ are parameters, σ a nonlinear Lipschitz activation function and K_ϕ is called a kernel. In GNO and MGNO, the kernel is obtained with a Monte Carlo estimation $\frac{1}{|\mathcal{N}(x)|}\sum_{x'\in\mathcal{N}(x)} K_\phi(x, x')v_t(x')$. In GNO, the kernel is then implemented as a message passing graph neural network, with $\mathcal{N}(x)$ the total number of neighbors of x. In MGNO, the kernel is implemented as a multi-scale kernel:

$$K \approx K_{1,1} + K_{1,2}K_{2,2}K_{2,1} + K_{1,2}K_{2,3}K_{3,3}K_{3,2}K_{2,1} + \cdots,$$

where $K_{l,l'}$ is the kernel from scale l to l', the finest scale is 1.
Following GNO and MGNO, a very popular operator leveraging the Fourier transform has been designed and is called Fourier Neural Operator (FNO, Li et al. (2020b)). It uses the same iterative transformation defined in Equation (2.13). The kernel is however implemented differently, instead of using a neural network to compute the kernel, the whole integral is computed using a transformation:

$$\int_\Omega K_\phi(x, x')v_t(x')\mathrm{d}\nu(x')\Big) = \mathcal{F}^{-1}(R_\phi \mathcal{F} v_t)(x) \qquad \forall x \in \Omega,$$

with $\mathcal{F}$ the Fourier transform, $\mathcal{F}^{-1}$ its inverse and R_ϕ a linear transform on the lower Fourier modes and a cut of the higher modes. In practice, when possible, the Fourier transform is computed with the Fast Fourier Transform (FFT). FNO has become very popular, leading to many follow-up works (Tran et al., 2021; Kovachki et al., 2021a; Li et al., 2022). A comparison between DeepONet led to the conclusion that both methods show similar results on simple settings, but DeepONet seems to perform better on more complex tasks (Lu et al., 2022). Also the properties of both methods are different, notably FNO does not need a fixed set of sensors, which makes FNO less sensible to the discretization.

2.3.2.5 INRs

In addition to neural operator, a new approach has been recently developed with continuous DL algorithms, which are the basis of Chapter 5. These are called Implicit Neural Representations (INRs) and have shown great results in various image related problems (Park et al., 2019; Mildenhall et al., 2021; Dupont et al., 2022a). Most popular methods for image related problems use the image as a discretized fixed grid, which is quite convenient if the images are each the same size and have the same channels (for instance RGB). This approach has proven to be highly efficient, however, it cannot handle different grids, hence limiting its applicability in various contexts, especially 3D shape reconstruction. Hence, a continuous approach, or INR, is natural solution to this problem. A INR is then a parameterized continuous function f_θ that maps coordinates of dimension m to their corresponding function values, a vector of dimension d, i.e., $f_\theta \colon \mathbb{R}^m \to \mathbb{R}^d$. The INR is usually implemented by a MLP with modified inputs and a specific activation function to efficiently represent high frequency details in the data, as empirically demonstrated for images in Sitzmann et al. (2020b); Tancik et al. (2020a).
Since these methods are continuous approaches, they are particularly suited for PDEs applications. Hence, following their increase in popularity in image related problems, they have been applied to various dynamical problems. MAgNet (Boussif et al., 2022) uses a INR to encode the geometry into a graph embedding and a GNN to forecast the dynamics. Having the INR for the encoding makes this approach more robust to the mesh, on the contrary to plain GNNs which are very sensitive to the discretization. This architecture showed interesting results on super-resolution tasks, i.e. generalizing on finer meshes than encountered during training. In a similar spirit, DINo (Yin et al., 2023) leverages INRs for PDE dynamics forecasting and investigate the generalization capacities of this approach compared to previous approaches. Their approach consists in encoding a frame with a network to produce a latent code which modifies the weights of the INR. This code is then propagated with a Neural ODE to produce a code correspond to the frame at a given time T, which is then decoded by modulating the INR to produce the final frame. This modulation is done to be able to encode different frames with the same INR. The training then consists in learning the modulation and the weights of the INR to reproduce an input image, before learning the parameters of the Neural ODE to propagate the latent code and predict the frame at time T. The generalization abilities of this architecture are very interesting for practitioners, namely the ability to extrapolate at arbitrary spatial and temporal locations, to learn from sparse irregular grids or manifolds and to generalize to new grids or resolution. In order to tackles as well static problems and offer an alternative in various problems to previous neural operators, CORAL (Serrano et al., 2023a) leverages an architecture using modulated INRs, with one representing the input function of an operator and the other the

output function of an operator. For forecasting tasks, the code is forecast with a neural network, that can be a Neural ODE. This flexible approach is not constrained by the mesh and can be applied to various problem domains, including PDE solving, spatio-temporal forecasting, and inverse problems like geometric design. INRs are a very interesting class of models with applications in PDEs prediction, which further confirms the possibility of solving directly PDEs with neural networks.

Chapter 3

Numerical schemes for Deep Learning

In this chapter, we study how to use numerical schemes to improve DL architectures. We aim at using the theories and ideas developed in numerical analysis, which have been developed for over a century. There are many potential applications for such concepts, the main idea being to add theoretical guarantees or intuitions to improve inductive biases of neural networks when solving dynamical systems.
First, we take inspiration from multi-grid methods to offer new options of architectures for Multipole Graph Neural Operators (MGNOs) (Li et al., 2020c). This leads to the comparison of three different models and a novel view on the MGNO architecture. We present competitive results on the two datasets used in the original paper. This work led to a publication at Proceedings of Topological, Algebraic, and Geometric Learning Workshops 2022.

Migus, L., Yin, Y., Mazari, J. A., and Gallinari, P. (2022, November). Multi-Scale Physical Representations for Approximating PDE Solutions with Graph Neural Operators. In Topological, Algebraic and Geometric Learning Workshops 2022 (pp. 332-340). PMLR.

Next, inspired by Chen et al. (2018); Zhang and Schaeffer (2020), which established connections between numerical schemes and residual neural networks, we design a neural network based on implicit numerical schemes. By incorporating theoretically grounded constraints, our architecture possesses an auto-regressive stability property. This property proves to be highly advantageous for forecasting dynamical systems, as it addresses the common issue of divergence over time that is often encountered in DL frameworks, as highlighted in our study. This work led to a publication at ICLR 2023 Workshop on Physics for Machine Learning.

Migus, L., Salomon, J., and Gallinari, P. (2023, May). Stability of implicit neural networks for long-term forecasting in dynamical systems. In ICLR 2023 Workshop on Physics for Machine Learning.

3.1 Multi-scale Physical Representations for Approximating PDE Solutions with Graph Neural Operators

Representing physical signals at different scales is a challenging problem in engineering. Several multi-scale modeling tools have been developed to describe physical systems governed by Partial Differential Equations (PDEs). These tools are at the crossroad of principled physical models and numerical schemes. Recently, data-driven models have been introduced to speed-up the approximation of PDE solutions compared to numerical solvers. They can be trained to represent the continuous PDE function with a discretized mesh. Among these recent data-driven methods, neural integral operators are a class that learn a mapping between function spaces. These functions are discretized on graphs (meshes) which are appropriate for modeling interactions in physical phenomena. In this work, we study three multi-resolution schemes with integral kernel operators that can be approximated with Message Passing Graph Neural Networks (MPGNNs). To validate our study, we make extensive MPGNNs experiments with well-chosen metrics considering steady and unsteady PDEs.

3.1.1 Introduction and motivation

Principled modeling of physical phenomena gives rigorous and interpretable mathematical models, e.g. Differential Equations (DEs). However, solving these equations analytically is impossible in most practical cases. To circumvent that, one could seek for an approximated solution through numerical analysis. When the phenomenon involves information and energy exchange in different ranges, methods with multi-scale modeling, e.g. multi-grid (multi-resolution) methods, are proposed for solving DEs. Said otherwise, multi-scale modeling consists in solving problems that have important features at different scales in time and space. An example of such problems is the Navier-Stokes equations, which can model an airfoil. In this instance, turbulence occurs around the airfoil, at high frequency, contrasting with the movement of the airfoil, which produces low-frequency phenomena. In order to solve these problems with adapted multi-scale schemes, interactions at different scales are modeled with a pyramidal discretization (Bergot and Duruflé, 2013; O'Malley et al., 2018). With the multi-scale modeling, the solvers converge more quickly than single-scale methods (Lie et al., 2017; Passieux et al., 2010).

In Deep Learning (DL), many methods have been proposed for approximating PDE solutions on a regular grid at a single scale (Um et al., 2020; Thuerey et al., 2020). However, in real world applications, the domain of PDEs is often discretized on meshes (represented by Euclidean graphs where vertices are points in an Euclidean domain and the edges with the distance between those points), where the nodes and the edges represent respectively the physical states and their interaction. In this case, we use Graph Neural Networks (GNNs) instead of more classical deep learning architectures such as MLPs, e.g. Pfaff et al. (2021); Xu et al. (2021).

For regular grids, some methods use U-net (Ronneberger et al., 2015), e.g. Wandel et al. (2021), to enable long-range interactions. Recently, Multipole Graph Neural Operator (MGNO, Li et al., 2020c) introduces a new graph-based method, which learns an operator mapping between two function spaces by a MPGNN with a multilevel graph, therefore offering a natural modeling of PDEs with a GNN as well as good performances with the multilevel graph. Li et al. (2020c) focuses on reducing computation cost of long-range correlation, inspired by fast multipole methods.

In this work, we explore new ways to extend the multi-scale modeling capacity with neural networks. We would like to shed light on different numerical schemes and understand the reason for the choice of multi-scale schemes from the lens of DL. We observe in practice that the multi-scale DL models share a similar structure with some of the numerical schemes . For example, both are composed of a straight-through downscaling and upscaling process, U-net shares a very similar structure with V-cycle schemes in multi-scale numerical analysis (Jaysaval et al., 2016). We then draw inspiration from discretized multi-resolution schemes (Jaysaval et al., 2016) such as W-cycle and F-cycle to propose new architectures.

We propose new multi-scale DL architectures, based on the original MGNO, for learning representation of multi-scale physical signals by approximating the functional spaces of PDEs. They are tested with steady and unsteady physical systems. This opens perspectives to rethink the architecture design for multi-scale problems and help practitioners using U-net to include these numerical schemes variants in their study. To the best of our knowledge, this is the first work to explore new architecture designs for multi-scale problems within the Machine Learning (ML) community.

We organize our paper as follows: in subsection 2, we describe formally Graph Neural Operators and the multi-resolution schemes. In subsection 3, we present our evaluation protocol and ablation study. Finally, in subsection 4, we conclude with some remarks and perspectives opened by our work.

3.1.2 Neural Operator and its graph instantiations

Neural Operator (Kovachki et al., 2021b) aims at learning a map from a function defined over a domain (typically Euclidean space) to another with parameterized models, especially neural networks (NNs). The objective is to learn a map $G_\theta : a \in \mathcal{F} \mapsto u \in \mathcal{F}'$, where $\mathcal{F}, \mathcal{F}'$ are two function spaces. At each position in the domain $x \in \Omega$, the following iterative transformation is applied:

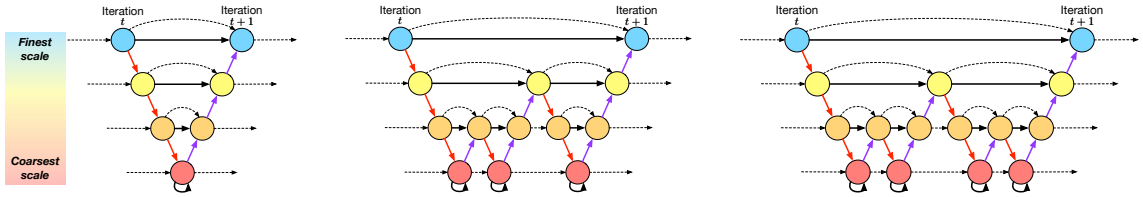
$$v_{t+1}(x) = \sigma\Big(W v_t(x) + b(x) + \int_\Omega K_\phi(x, x') v_t(x') \mathrm{d}\nu(x')\Big) \qquad \forall x \in \Omega$$

where $\theta = \{W, b, \phi\}$ are parameters. The parameters W, b are local affine transformation of $v_t(x)$ at each location x. The parameterized kernel K_ϕ integrates $v_t(x')$ over all $x' \in \Omega$, with $x' \sim \nu$ and ν is a (probabilistic) measure over Ω. The function σ is a nonlinear Lipschitz activation function. The iteration starts from $t = 0$ with the input function $a = v_0$, the solution is the function at final iteration T, i.e. $u = v_T$.

GNO and MGNO (Anandkumar et al., 2020). Given that integrating over the whole domain Ω is intractable, one possible simplification is to integrate only in a subdomain around x, i.e. $s(x) \subset \Omega$. By limiting this subdomain to some i.i.d. sampled neighbors $x' \in \mathcal{N}(x) \subset s(x)$, we obtain:

$$\int_\Omega K_\phi(x, x') v_t(x') \mathrm{d}\nu(x') \approx \int_{s(x)} K_\phi(x, x') v_t(x') \mathrm{d}\nu(x') \approx \frac{1}{|\mathcal{N}(x)|} \sum_{x' \in \mathcal{N}(x)} K_\phi(x, x') v_t(x')$$

which can be implemented with a message passing graph neural network, called GNO. The graph is constructed by connecting the point at position x to its neighbors x' such that $\forall x' \in \mathcal{N}(x), \|x - x'\| \leq r$. Here $|\mathcal{N}(x)|$ stands for the total number of the



(a) V-MGNO (b) F-MGNO (c) W-MGNO

Figure 3.1: Illustration of MGNOs with different schemes: (a) the original V-MGNO, (b) F-MGNO, and (c) W-MGNO. $\rightarrow$ (red) are downscale kernels, $\rightarrow$ (black) are in-scale kernels, $\rightarrow$ (purple) are upscale kernels. $\dashrightarrow$ are skip connections. See Figure 3.2 for an example of the iterative process.

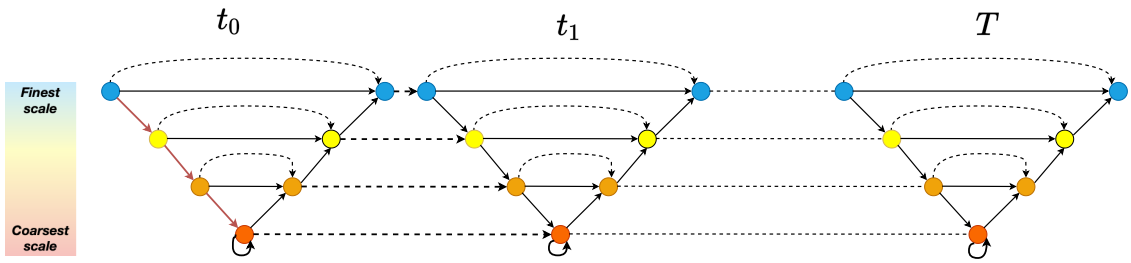


Figure 3.2: Example of an iterative process with V-cycle multi-resolution architecture. Similar process is applied for F-cycle and W-cycle.

neighbors of x. However, when modeling long-range interactions is necessary, a large $s(x)$ should be considered in GNO, which comes with an expensive computational cost. To overcome this problem, MGNO considers the following multi-scale kernel matrix decomposition of the graph kernel:

$$K \approx K_{1,1} + K_{1,2}K_{2,2}K_{2,1} + K_{1,2}K_{2,3}K_{3,3}K_{3,2}K_{2,1} + \cdots$$

where $K_{l,l'}$ is the kernel from scale l to l', the finest scale is 1. To implement this multi-scale kernel, several architectural designs are proposed in the following subsection.

Multi-scale kernel implementations. We present in this subsection, three architectures for multi-scale kernel implementation: V-MGNO, F-MGNO, and W-MGNO. The V-MGNO architecture is proposed in the original paper (Li et al., 2020c), whereas F-MGNO, and W-MGNO are inspired by multi-resolution methods (Jaysaval et al., 2016). The V-, F-, and W-MGNO are iterative processes and have in common three types of kernels as shown in Figure 3.1: downscale, intrascale, and upscale kernels. The input information is propagated in the multi-scale graph by a cascade of downscale contraction, intrascale transformation, and upscale expansion. The downscale kernels reduce the number of points of the graph on the contrary to upscale which increases it. Intrascale kernels keep the same dimension and propagate the information to kernels with the same size. The overall architectures as presented

in Figure 3.1 combines these kernels to allow propagation of information at different scales. The implementation then mainly consists of deciding the number of points in the graph at each scale.

Metrics. The goal here is to empirically study the performances of these multi-resolution architectures and their learning stability. We define the following characteristics to compare these multi-scale architectures:

- *Number of scales*: indicates the number of scales from the finest to the coarsest one. The choice of the finest and the coarsest scales depends on the discretization scheme, and also on the cutoff energy, as well as the computational budget.
- *Intra-cycle Kernel Sharing*: indicates whether the kernels are shared inside each iteration. This is because F- and W-MGNO may have several kernels for the same downsampling or upsampling action, e.g. between the coarsest two scales, W-MGNO (shown in Figure 3.1) need to several downscaling (red arrows) or upscaling (purple arrows).
- *Iteration Kernel Sharing*: indicates if the kernels are shared across all iterations. This allows to study the stability and the complexity of optimizing iterative processes w.r.t. a given multi-scale architecture.

3.1.3 Experiments

We evaluate the performance of our multi-resolution schemes F-MGNO and W-MGNO on two families of PDEs, namely 2D steady-state of *Darcy flow* and 1D viscous unsteady *Burgers' equation*.

Datasets. *Darcy flow.* We construct our first dataset with the following 2-D steady-state PDE

$$\begin{cases} -\nabla \cdot (a(x)\nabla u(x)) = f(x), & x \in (0,1)^2 \\ u(x) = 0, & x \in \partial(0,1)^2 \end{cases}$$

where a is a random piecewise constant function parameterizing the PDE, and f a function. In the experiments, we approximate the mapping $(a, \nabla a) \mapsto u$.

Burgers' equation. In the second dataset, we consider the following 1-D viscous unsteady PDE

$$\begin{cases} \partial_t u(x,t) + \partial_x(u^2(x,t)/2) = \nu \partial_{xx} u(x,t), & x \in (0,2\pi), t \in (0,1] \\ u(x,0) = u_0(x), & x \in (0,2\pi) \end{cases}$$

with periodic boundary conditions. In the experiments, we approximate the mapping from the initial condition u_0 to the solution at time $t = 1$, i.e. $u_0 \mapsto u(\cdot, 1)$.

For both datasets, 100 trajectories were generated for train and 100 others for validation and test.

Pipelines Graph nodes are uniformly sampled over the domain with different sample number at each scale. Graph edges in each scale are calculated to all points in the same scale within a given distance. For multi-scale models, edges between scales are also calculated likewise. Input node features are values of the input function $v_0(x)$ and their position x. Each model predicts $u(x)$. To avoid divergence in training, we use the orthogonal initialization across all models. A number of scales of 1 to 4 is tested. The different levels are taken to be respectively 1600, 400, 100 and 25 points for each scale.

Baseline methods. We compare our proposed F-MGNO and W-MGNO w.r.t. GNO and the original V-MGNO. To study to what extent the kernel construction from operator learning standpoint is helpful, proposed architectures are also compared with MLP and GCN (Kipf and Welling, 2017). Methods are categorized into single-scale (MLP, GCN, GNO) and multi-scale (V-, F-, W-MGNO). Single scales method do not offer any levels with different resolutions.

On learning stability. During training, we found out that the multi-scale architectures with Kaiming initialization (He et al., 2015) used in the original paper may lead to divergence in training. This is caused by extremely large loss and gradient at the beginning of the training. One possible explanation is that, as the input is transformed through many kernels, an improper initialization will amplify the norm of features along all the forward steps. We therefore chose orthogonal initialization to better control the norm of the output.

We did a broad hyperparameter search to tune the model. This showed that the initialization was crucial in MGNO architectures, with a main importance for the learning rate and even more for the initialization gain of the orthogonal initialization.

Results. We show our results in Table 3.1 for *Darcy flow* and *Burgers' equation*. For all variants of MGNO, we report the results of the best architectures for each model. More detailed results are shown in Table 3.2 and Table 3.3.

We found that multi-scales methods outperform single-scale ones in both training and test error. For both datasets, multi-scale methods achieve a decrease of 80-90% training error compared to the single-scale baselines. This suggests that a better modeling of long-range interaction improves the expressiveness of the neural network. The same improvement tendency was also observed at test time. All vari-

Table 3.1: Results of our best settings compared to baselines on *Darcy flow* and *Burgers' equation*. We calculate the means and standard deviations for each model based on 4 runs with different seeds. All multi-scale models results are achieved with 4 scales.

		Model	Intra-cycle Kernel Sharing	Iteration Kernel Sharing	Train Error ($\times 10^{-2}$)	Test Error ($\times 10^{-2}$)
Darcy flow	**Single-scale**	MLP	N/A	N/A	11.90±0.20	12.02±0.40
		GCN	N/A	N/A	11.87±0.15	11.88±0.19
		GNO	N/A	N/A	6.45±0.21	7.13±0.15
	Multi-scale	V-MGNO	N/A	✗	2.69±0.18	**5.68±0.30**
		F-MGNO (Ours)	✗	✗	2.76±0.15	5.80±0.32
		W-MGNO (Ours)	✗	✗	**2.23±0.11**	5.91±0.28
Burgers'	**Single-scale**	MLP	N/A	N/A	41.89±0.40	42.07±0.11
		GCN	N/A	N/A	27.88±1.46	31.00±1.22
		GNO	N/A	N/A	15.30±0.17	53.14±0.86
	Multi-scale	V-MGNO	N/A	✓	4.25±0.10	25.76±0.39
		F-MGNO (Ours)	✓	✓	**3.19±0.05**	25.20±0.20
		W-MGNO (Ours)	✗	✓	3.47±0.07	**24.91±0.37**

ants of MGNO reduce the test error by 10-50%. This improvement is less significant than at training time as it may be limited by number of training data samples.

Among the multi-scale models, for both PDEs, more complex F-/W-MGNO performs slightly better in training, i.e. 2.69 with V-MGNO down to 2.23 (-17%) with W-MGNO for Darcy flow, and 4.25 with V-MGNO down to 3.19 (-25%) with F-MGNO for Burgers'. However, we did not perceive a significant difference in test error.

Ablation studies. We conducted large-scale ablation studies with results in Tables 3.2 and 3.3. We analyze the influence of different metrics on impacts of architectural metrics on prediction errors:

- *The impact of number of scales*: We observe that the more the scales, the lower the training error. This shows an increasing tendency in model expressiveness w.r.t. scales.

- *The impact of kernel sharing*: We observe that sharing parameters may help improving training. However, no major differences in test are noticed when evaluating generalization to test samples.

Note that for some scales different variants of MGNO are the same. For 2-scale models, V-, F-, and W-MGNO are equivalent. For 3-scale models, F- and W-MGNO are equivalent.

Table 3.2: Ablation studies for Burgers' equation.

Method	Scales	Intra-cycle Sharing	Iteration Kernel Sharing	Train Error ($\times 10^{-2}$)	Test Error ($\times 10^{-2}$)	Time (s) /epoch	N. params (M)
MLP	1	N/A	N/A	41.89±0.40	42.07±0.11	2.8	0.017
GCN	1	N/A	N/A	27.88±1.46	31.00±1.22	1.9	0.025
GNO	1	N/A	N/A	15.30±0.17	53.14±0.86	27.1	1.10
V/F/W-MGNO	2	N/A	✗	7.21±0.61	25.63±0.53	73.1	10.91
V/F/W-MGNO	2	N/A	✓	6.35±0.17	25.67±0.46	66.5	2.74
V-MGNO	3	N/A	✗	4.76±0.24	27.22±1.19	77.5	14.13
V-MGNO	3	N/A	✓	4.22±0.14	26.65±0.49	62.8	3.54
F/W-MGNO	3	✗	✓	4.59±0.16	26.64±1.38	77.3	4.89
F/W-MGNO	3	✓	✓	4.02±0.15	25.29±0.86	85.8	3.54
V-MGNO	4	N/A	✗	4.67±0.16	26.19±0.26	70.1	15.75
V-MGNO	4	N/A	✓	4.25±0.10	25.76±0.39	74.8	3.95
F-MGNO	4	✗	✓	3.59±0.09	25.45±0.63	89.5	6.39
F-MGNO	4	✓	✓	3.19±0.05	25.20±0.20	77.5	3.95
W-MGNO	4	✗	✓	3.47±0.07	24.91±0.37	100.4	7.06
W-MGNO	4	✓	✓	3.10±0.03	25.61±0.31	77.7	3.95

Table 3.3: Ablation studies for *Darcy flow*.

Method	Scales	Intra-cycle Kernel Sharing	Iteration Kernel Sharing	Train Error ($\times 10^{-2}$)	Test Error ($\times 10^{-2}$)	Time /epoch (s)	N. params (M)
MLP	1	N/A	N/A	11.90±0.20	12.02±0.40	2.0	0.02
GCN	1	N/A	N/A	11.87±0.15	11.88±0.19	34.3	0.03
GNO	1	N/A	N/A	6.45±0.21	7.13±0.15	13.1	0.03
V/F/W-MGNO	2	N/A	✗	4.76±0.35	6.67±0.43	25.8	11.0
V/F/W-MGNO	2	N/A	✓	4.88±0.40	6.59±0.24	25.9	2.74
V-MGNO	3	N/A	✗	3.63±0.21	5.77±0.12	28.0	14.1
V-MGNO	3	N/A	✓	3.24±0.22	6.11±0.32	27.8	3.55
F/W-MGNO	3	✗	✗	3.03±0.22	5.94±0.38	33.0	19.5
F/W-MGNO	3	✓	✗	2.63±0.19	6.41±0.24	33.5	14.1
F/W-MGNO	3	✗	✓	2.47±0.15	5.76±0.25	34.5	4.90
F/W-MGNO	3	✓	✓	2.26±0.13	5.87±0.23	34.6	3.55
V-MGNO	4	N/A	✗	2.69±0.18	5.68±0.30	28.3	15.8
V-MGNO	4	N/A	✓	2.02±0.11	5.96±0.22	28.3	3.95
F-MGNO	4	✗	✗	2.76±0.15	5.80±0.32	39.2	25.5
F-MGNO	4	✓	✗	1.88±0.12	5.84±0.18	37.1	15.8
F-MGNO	4	✗	✓	1.78±0.05	6.14±0.32	36.5	6.39
F-MGNO	4	✓	✓	1.60±0.13	6.18±0.31	36.9	3.95
W-MGNO	4	✗	✗	2.23±0.11	5.91±0.28	40.9	28.2
W-MGNO	4	✓	✗	1.68±0.13	6.17±0.28	39.2	15.8
W-MGNO	4	✗	✓	1.85±0.08	6.07±0.33	39.1	7.07
W-MGNO	4	✓	✓	1.57±0.10	6.29±0.27	40.5	3.95

Discussion. To conclude, we confirm with our experiments that multi-scale methods are important for better modeling physical signals, by efficiently modeling long-range interactions in the spatial domain. Some improvements in training were observed with F- and W-MGNO, which may indicate an increase in model

expressiveness with F- and W-cycle schemes . However, compared to the V-MGNO implementation, the evidence is not strong enough to support this claim. We suggest further investigation with other graph-based approaches to better understand more complex cycles. In order to better generalize to unseen data, more constraints could be added to control train and test trade-off.

3.1.4 Conclusion

In this work, we empirically studied the challenging task of representing physical signals at different scales from DL perspective. To do so, we developed two novel multi-scale architectures inspired by discretized multi-resolution schemes (Jaysaval et al., 2016) and a neural integral operator, which is motivated by multipole theory (Li et al., 2020c). To the best of our knowledge, this is the first work that proposes to study and design multi-scale DL architectures from a numerical scheme standpoint. We proposed two MPGNNs to approximate this neural integral operator and we implemented it via F-cycle and W-cycle schemes. The latter are iterative processes and hence are challenging to optimize. We defined a set of metrics to evaluate the learning stability of these iterative processes and their performances. We validated our work on two types of PDEs discretized on graphs.

We argue that this work could open perspectives to study novel multi-scale neural architectures, beyond U-net, and V-F-W-cycle schemes, suitable for multi-scale or scarce data. One may consider a further study of discretized multi-resolution schemes including the properties of their architectures and optimization procedures. This could help to design more interpretable and efficient architectures. Moreover, we think that studying multi-scale neural architectures from a discretized multi-resolution scheme standpoint could help to get insights about the capabilities of multi-scale neural architectures to reproduce some properties of discretized multi-resolution schemes.

3.2 Stability of implicit neural networks for long-term forecasting in dynamical systems

Forecasting physical signals in long time range is a challenging task in Partial Differential Equations (PDEs) research. To circumvent limitations of traditional solvers, many different Deep Learning methods have been proposed. They are based on auto-regressive methods and exhibit stability issues. Drawing inspiration from the stability property of implicit numerical schemes, we introduce a stable auto-regressive implicit neural network. We develop a theory based on the stability definition of numerical schemes to ensure the stability in forecasting of this network. It leads us to introduce hard constraints on its weights and propagate the dynamics in a latent space. Our experimental results validate our stability property, and show improved results at long-term forecasting for two transports PDEs.

3.2.1 Introduction and motivation

Numerical simulations are one of the main tools to study systems described by PDEs, which are essential for many applications including, e.g., fluid dynamics and climate science. However, solving these systems and even more using them to predict long term phenomenon remains a complex challenge, mainly due to the accumulation of errors over time. In an attempt to overcome the limitations of traditional solvers and to exploit the available data, many different deep learning methods have been proposed. For the task of forecasting spatio-temporal dynamics, Ayed et al. (2019) used a standard residual network with convolutions and Sorteberg et al. (2019); Lino et al. (2020); Fotiadis et al. (2020) used Long short-term memory (LSTM) and Convolutional neural network (CNN) for the wave equation. In Wiewel et al. (2019); Kim et al. (2019a), good performances are obtained by unrolling the dynamics within the latent spaces of neural networks. More recently, Brandstetter et al. (2022) used graph neural networks with several tricks and showed improved results for forecasting PDEs solutions. Importantly, these methods all solve the PDE iteratively, meaning that they are auto-regressive, the output of the model is used as the input for the next time step. Another line of recent methods that have greatly improved the learning of PDE dynamics are Neural Operators (Li et al., 2020b). These methods can be used as operators or as auto-regressive methods to forecast. However, when used as operators, they do not generalize well beyond the training horizon. Crucially, these auto-regressive methods tend to accumulate errors over time with no possible control, and respond quite poorly in case of change in the distribution of the data. This leads to stability problems, especially over long periods of time beyond the training horizon.

In the numerical analysis community, the stability issue has been well studied and is usually dealt with implicit schemes. By definition, they imply to solve an

equation to go from a time step to the next one but they are generally more stable than explicit schemes. This can be seen on the test equation $\frac{\mathrm{d}y}{\mathrm{d}t} = \lambda y$, where Euler implicit schemes are always stable while Euler explicit schemes are not. Interpreting residual neural networks as numerical schemes, one can apply such schemes and gain theoretical insights on the properties of neural networks. This has already been done in various forms in Haber and Ruthotto (2017); Chen et al. (2018), but not applied to forecasting. Moreover, these works use either the stability of the underlying continuous equation or the stability of the numerical scheme on the test equation and its derivatives, which is not the stability of the numerical scheme on the studied equation. More details on this topic can be found in Section 2.1. Since a network is discrete, the latter is the most relevant. More precisely, We therefore use the stability in norm of schemes, as defined in Definition 3.2.1. In DL, this definition has been applied to image classification problems (Zhang and Schaeffer, 2020). To the best of our knowledge, this work is the first attempt to forecast PDEs with neural networks using stability as studied in the numerical analysis community.

Using implicit schemes in DL has already been done in different contexts. The earliest line of works tackles image problems, with Haber et al. (2019) designing semi-implicit ResNets and Li et al. (2020a); Shen et al. (2020); Reshniak and Webster (2021) designing different implicit ResNets. The second line of works focuses on dynamical problems. In this way, Nonnenmacher and Greenberg (2021) designed linear implicit layers, which learn and solve linear systems, and Horie and Mitsume (2022) used an implicit scheme as part of graph neural network solvers to improve forecasting generalization with different boundary conditions. While tackling different types of problems, none of these methods guarantees the forecast stability. For our analysis, we restrict ourselves to ResNet-type networks, i.e., networks with residual connections. Indeed, we use the connection between ResNet networks and numerical schemes to develop our architecture. We introduce hard constraints on the weights of the network and unroll the dynamics within the latent space of our network. Hence, by modifying the classic implicit ResNet architecture, our method can forecast dynamical system at long range without diverging. We apply these theoretical constraints in our architecture to two 1D transport problems: the *Advection equation* and *Burgers' equation*.

To better investigate networks stability, we perform our experiments under the following challenging setting: during the training phase, we exclusively provide the network with data at time $t = 0$ to predict the solution at a short time $t = \Delta t$. We subsequently evaluate its performance in forecasting over a much longer horizon at time $t = N \cdot \Delta t$, where $N \gg 1$. Note that our setting is harder that the conventional setting presented for e.g. in Brandstetter et al. (2022).Indeed, we only use the evolution of the solution between two time steps for training, instead of using the solution at many time steps.

3.2.2 Method

To guarantee structural forecasting stability, we take inspiration from implicit schemes. We focus our study on an implicit ResNet with a ReLU activation function. In our approach, an equation is solved at each layer, namely $z_{n+1} := z_n + R_n(z_{n+1})$ with z in $\mathbb{R}^M$ and n in $\mathbb{N}$ and $R_n(z) = \mathrm{ReLU}(W_n z + b_n)$ where W_n is a upper triangular matrix. The latter constraint is motivated below.

Implicit ResNet stability analysis

To study the stability of our proposed architecture, we first need to ensure that it is well-defined, by solving $z = z_n + R_n(z)$. In order to do so, we first define the Perron–Frobenius eigenvalue, before stating the root existence, which uses this eigenvalue. Let A be a non-negative square matrix, i.e. with non-negative entries.

Theorem 3.2.1 (Perron–Frobenius theorem)**.** *A admits a real eigenvalue that is larger than the modulus of any other eigenvalue.*

This non negative eigenvalue is called the Perron–Frobenius eigenvalue and is denoted $\lambda_{pf}(A)$.

Theorem 3.2.2 (Root existence)**.** *Given $n \in \mathbb{N}$ and that ReLU is non-expansive, if $\lambda_{pf}(|W_n|) < 1$, then x defined as $x = x_n + R_n(x)$ exists.*

The proof is available in theorem 2.2 of El Ghaoui et al. (2019). They show that the solution can be obtained using a fixed-point iteration. However they do not offer any analytical solution.
We can now study the stability of our proposed architecture. The recursion defining $(z_n)_{n\in\mathbb{N}}$ reads as an implicit Euler scheme with a step size of 1. As described in the introduction, an implicit scheme is usually more stable than an explicit one. We first recall the definition of the stability in norm L^p for schemes, as found in Section 10.4.2 of Legendre (2018). This property ensures that our architecture has an auto-regressive stability.

Definition 3.2.1 (Stability in norm)**.** *A numerical scheme solution $(z_n)_{n\in\mathbb{N}}$ of dimension M is stable in norm L^p if there exists for a time T, $C(T)$ independent of the time discretization step Δt such that:*

$$\forall\ z_0 \in \mathbb{R}^M,\ n \geq 0;\ n\Delta t \leq T,\ \|z_n\|_p \leq C(T)\|z_0\|_p\ .$$

This general definition of stability with respect to the norm ensures that a scheme does not amplify errors. This definition is equivalent to several others in the numerical analysis community.
Suppose that the spectrum of W_n is contained in $[-1, 0)$ for every integer n, we can assert that $(z_n)_{n\in\mathbb{N}}$ is well-defined, using theorem 3.2.2. The proof of the stability

of our Implicit ResNet network is then by induction on the dimension and is given below.

Theorem 3.2.3 (Stability theorem)**.** *If the elements of the diagonal of W_n are in $[-1, 0)$ for every integer n, then $(z_n)_{n\in\mathbb{N}}$ is stable in norm L^p.*

In order to prove Theorem 3.2.3, we first define the variables used, before proving a lemma about the expression of an auxiliary sequence $v_n^{(m)}$. We then proceed to the full proof.

Definitions and notation Let $(\alpha_n^{(m_1,m_2)})_{m_1,m_2\in[1:M]^2}$ be the strict upper values of W_n and $(b_n^{(m)})_{m\in[1:M]}$ the values of b_n. We suppose that $(\alpha_n^{(m_1,m_2)})_{n\in\mathbb{N}}$ and $(b_n^{(m)})_{n\in\mathbb{N}}$ are bounded. Let $Q := \max_{m_1\in[|0,M-1|],\, m_2\in[|m_1+1,m|]}(\max_{n\in\mathbb{N}}(|\alpha_n^{(m_1,m_2)}|))$ and $B := \max_{n\in\mathbb{N},\, m\in[|1,M|]}(b_n^{(m)})$. Let $(-\lambda_n^{(m)})_{m\in[1:M]}$ be the values of the diagonal of W_n, and $P := \min_{n\in\mathbb{N},\, m\in[\![1,M]\!]}(\lambda_n^{(m)}))$, with P being finite and positive by hypothesis.
Given an integer n and $z_n \in \mathbb{R}^M$, we denote by $z_n^{(m)}$ the n^{th} iteration of the m^{th} dimension of the sequence (z_n). For m in $[|1, M|]$, let $S_m := \max_{j\in[|1,m|],\, k\in\mathbb{N}}(z_k^{(j)})$ and $S_0 = 0$.

Definition 3.2.2. *We define, given $n \in \mathbb{N}$ and $m \in [|0, M-1|]$, $v_n^{(m)}$ by the recursion:*

$$v_{n+1}^{(m)} := \frac{v_n^{(m)} + \sum_{j=1}^{m-1} \alpha_n^{((m-1),j)} z_{n+1}^{(j)} + b_n^{(m)}}{1+\lambda_{n+1}^{(m)}}. \tag{3.1}$$

Lemma 3.2.1 (Explicit expression of $v_n^{(m)}$)**.** *Given integer n and m in $[|1, M|]$, an explicit expression of $v_n^{(m)}$ is given by:*

$$v_n^{(m)} = z_0^{(m)} \prod_{k=1}^{n} \frac{1}{1+\lambda_k^{(m)}} + \sum_{k=1}^{n} \Big(\prod_{l=k}^{n} \frac{1}{1+\lambda_l^{(m)}} \sum_{j=1}^{m-1} \alpha_{k-1}^{((m-1),j)} z_k^{(j)} \Big) + \sum_{k=1}^{n} \Big(\prod_{l=k}^{n} \frac{1}{1+\lambda_l^{(m)}} b_{k-1}^{(m)} \Big).$$

Proof. In order to obtain an explicit expression of $v_n^{(m)}$, we write out all the terms of $v_n^{(m)}$. Let i be an integer in $[0, n]$. We then multiply each term by $\prod_{k=2}^{i+1} \frac{1}{1+\lambda_k^{(m)}}$:

$$\prod_{k=2}^{i+1} \frac{1}{1+\lambda_k^{(m)}} \Big(v_{n+1-i}^{(m)} - \frac{1}{1+\lambda_{n+1-i}^{(m)}} z_{n-i}^{(m)} \Big) = \Big(\frac{\sum_{j=1}^{m-1} \alpha_{n-i}^{((m-1),j)} z_{n+1-i}^{(j)} + b_{n-i}^{(m)}}{1+\lambda_{n+1-i}^{(m)}} \Big) \prod_{k=2}^{i+1} \frac{1}{1+\lambda_k^{(m)}}. \tag{3.2}$$

We thus obtain a telescoping sum by summing Equation 3.2 for every i in $[0, n]$. □

Proof of theorem 3.2.3

Proof. Hereafter, we prove that, for m in $[|1, M|]$, $(z_n^{(m)})_{n\in\mathbb{N}}$ is bounded. The proof is by induction on m.

For the base case $m = 1$, let n be an integer. We will show that $(z_n^{(1)})_{n\in\mathbb{N}}$ is bounded.

It is easily seen that:

$$z_{n+1}^{(1)} = \begin{cases} z_n^{(1)} & \text{, if } -\lambda_{n+1}^{(1)} z_n^{(1)} + b_n^{(1)} \leq 0 \\ \frac{1}{1+\lambda_{n+1}^{(1)}}(z_n^{(1)} + b_n^{(1)}) & \text{, else.} \end{cases}$$

Let $u_n^{(1)} := z_0^{(1)}$ and $v_{n+1}^{(1)} := \frac{v_n^{[1)} + b_n^{(1)}}{(1+\lambda_{n+1}^{(1)})}$. We then have $\min(u_n^{(1)}, v_n^{(1)}) \leq z_n^{(1)} \leq \max(u_n^{(1)}, v_n^{(1)})$.
Using lemma 3.2.1, $v_n^{(1)}$ may be written as:

$$v_{n+1} = z_0^{(1)} \prod_{k=1}^{n+1} \frac{1}{(1+\lambda_k^{(1)})} + \sum_{k=1}^{n+1} (\prod_{l=k}^{n+1} \frac{1}{(1+\lambda_l^{(1)})} b_{k-1}^{(1)}). \tag{3.3}$$

We can then bound the second term of the right-hand side of Equation 3.3:

$$\begin{aligned} |\sum_{k=1}^{n+1} (\prod_{l=k}^{n+1} \frac{1}{(1+\lambda_l^{(1)})} b_{k-1}^{(1)})| &\leq B \sum_{k=1}^{n+1} (\frac{1}{(1+P)^{n+1-k}} \\ &\leq B \frac{(1+P)^{n+1} - (1+P)}{P(1+P)^{n+1}}. \end{aligned} \tag{3.4}$$

Combining Equation 3.3 and Equation 3.4, we can assert that $v_n^{(1)}$ is bounded.
Since $\min(z_0^{(1)}, v_n^{(1)}) \leq z_n^{(1)} \leq \max(z_0^{(1)}, v_n^{(1)})$, we can conclude that $(z_n^{(1)})_{n\in\mathbb{N}}$ is bounded.
Suppose $\forall\ j \in [|0, m|], (z_n^{(j)})_{n\in\mathbb{N}}$ bounded, we will now prove that $(z_n^{(m+1)})_{n\in\mathbb{N}}$ is bounded.
We first solve Equation 3.5 to find an expression of $z_{n+1}^{(m+1)}$.

$$Z = z_n^{(m+1)} + \max(0, -\lambda_{n+1}^{(m+1)} Z + \sum_{j=1}^{m} \alpha_n^{(m,j)} z_{n+1}^{(j)} + b_n^{(m+1)}). \tag{3.5}$$

Summarizing, we obtain:

$$z_{n+1}^{(m+1)} = \begin{cases} z_n^{(m+1)} & \text{, if } -\lambda_{n+1}^{(m+1)} z_n^{(m+1)} + \sum_{j=1}^{m} \alpha_n^{(m,j)} z_{n+1}^{(j)} + b_n^{(m+1)} \leq 0 \\ \frac{z_n^{(m+1)} + \sum_{j=1}^{m} \alpha_n^{(m,j)} z_{n+1}^{(j)} + b_n^{(m+1)}}{(1+\lambda_{n+1}^{(m+1)})} & \text{, else.} \end{cases}$$

Let $u_n^{(m+1)} := z_0^{(m+1)}$ and $v_{n+1}^{(m+1)} := \frac{v_n^{(m+1)} + \sum_{j=1}^{m} \alpha_n^{(m,j)} z_{n+1}^{(j)} + b_n^{(m+1)}}{(1+\lambda_{n+1}^{(m+1)})}$. We then have that

$$\min(u_n^{(m+1)}, v_n^{(m+1)}) \leq z_n^{(m+1)} \leq \max(u_n^{(m+1)}, v_n^{(m+1)}).$$

Using lemma 3.2.1, $v_n^{(m+1)}$ may be written as:

$$v_{n+1}^{(m+1)} = z_0^{(m+1)} \prod_{k=1}^{n+1} \frac{1}{1+\lambda_k^{(m+1)}} + \sum_{k=1}^{n+1} \left(\prod_{l=k}^{n+1} \frac{1}{1+\lambda_l^{(m+1)}} \sum_{j=1}^{m} \alpha_{k-1}^{(m,j)} z_k^{(j)} \right) + \sum_{k=1}^{n+1} \left(\prod_{l=k}^{n+1} \frac{1}{1+\lambda_l^{(m+1)}} b_{k-1}^{(m+1)} \right).$$

It is easily seen that the first and third terms of $v_n^{(m+1)}$ are bounded. We still wish to bound the second term of $v_n^{(m+1)}$. Using the induction hypothesis, S_m is finite. We can then bound the second term of $v_n^{(m+1)}$:

$$\begin{aligned}
\left| \sum_{k=1}^{n+1} \left(\prod_{l=k}^{n+1} \frac{1}{1+\lambda_l^{(m+1)}} \sum_{j=1}^{m} \alpha_{k-1}^{(m,j)} z_k^{(j)} \right) \right| &\leq \left| \sum_{k=1}^{n+1} \left(\frac{1}{(1+P)^{n+1-k}} \sum_{j=1}^{m} \alpha_{k-1}^{(m,j)} z_k^{(j)} \right) \right| \\
&\leq \left| \sum_{k=1}^{n+1} \frac{1}{(1+P)^{n+1-k}} mQS_m \right| \\
&\leq mQS_m \frac{1}{(1+P)^{n+1}} \sum_{k=1}^{n+1} \left((1+P)^k \right. \\
&\leq mQS_m \frac{(1+P)^{n+2} - (1+P)}{P(1+P)^{n+1}}. \qquad (3.6)
\end{aligned}$$

Since Equation 3.6 shows that the second term of $v_n^{(m+1)}$ is bounded, $v_n^{(m+1)}$ is bounded, hence we can conclude that $z_n^{(m+1)}$ is bounded.
Since both the base case and the induction step have been proved to be true, by mathematical induction for every m in $[|1, M|]$, $(z_n^{(m)})_{n\in\mathbb{N}}$ is bounded. Hence $z_n = (z_n^{(1)}, ..., z_n^{(M)})$ is bounded. □

Implementation

To validate practically our theoretical results, we choose a setting that highlights stability issues. We then test our implementation of an implicit ResNet. In order to respect the assumptions of theorem 3.2.3, we forecast the dynamics in the latent space, as detailed below.

Setting We first learn the trajectory at a given small time step Δt. We only give data at $t = 0$ to predict data at $t = \Delta t$ during the training phase. We then forecast in long-term, at $N \cdot \Delta t$ with $N \gg 1$. This very restricted setting allows us to see how the network will react in forecasting with changes in the distribution and error accumulation. Usually neural network forecasting methods use data from $t = 0$ to $T = L \cdot \Delta t$ for the training which allows to use different tricks to stabilize the network, such as predicting multiple time steps at the same time. However, in this work, we want to analyze how the network behaves without any trick that can

slow down divergence. Indeed, the tricks used in the other settings do not actually guarantee stability of the network. The training is performed with a mean-squared error (MSE) loss.

Implicit neural network architecture To implement a neural network from Theorem 3.2.3, we use the encode-process-decode architecture with residual blocks; $x_{\Delta t} = f_{dec} \circ f_{res}^K \circ ... \circ f_{res}^1 \circ f_{enc}(x_0)$, with $f_{res}^k(z) = f_{res}^{k-1}(z) + \text{ReLU}(W_{k-1} \cdot f_{res}^k(z) + b_{k-1})$, $f_{dec}(z) = W_{dec} \cdot z + b_{dec}$ and $f_{enc}(x) = W_{enc} \cdot x + b_{enc}$. The encoder and decoder are linear, and the encoder projects the data onto a smaller dimension M. The full architecture is illustrated in Figure 3.3. The specificity of our architecture is that the residual blocks are connected with an implicit Euler scheme iteration. To do so, we use a differentiable root-finding algorithm (Kasim and Vinko, 2020). More precisely, we use the first Broyden method (van de Rotten and Lunel, 2005) to solve the root-finding problem. It is a quasi-Newton method. This helped getting better results compared to other algorithms.

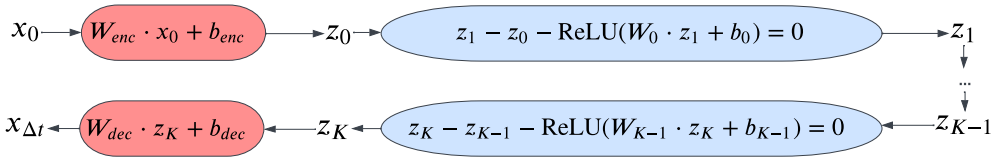


Figure 3.3: Implicit neural network architecture with K residual blocks.

Latent space forecasting As in Wiewel et al. (2019); Kim et al. (2019a), the forecast can be done within the latent space of the network; $x_{N \cdot \Delta t} = f_{dec} \circ f_{Kblocks} \circ ... \circ f_{Kblocks} \circ f_{enc}(x_0)$, with $f_{Kblocks} = f_{res}^K \circ ... \circ f_{res}^1$. To predict at time $N \cdot \Delta t$ from time $t = 0$, we apply N times the composition of the K residual blocks. It is illustrated in Figure 3.5, as Figure 3.4 illustrates the traditional approach. This propagation allows our network to respect the assumptions of theorem 3.2.3 and thus be stable.

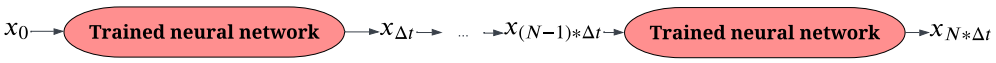


Figure 3.4: Traditional auto-regressive forecasting.

3.2.3 Experiments

We evaluate the performance of our method on the *Advection equation* and *Burgers' equation*.

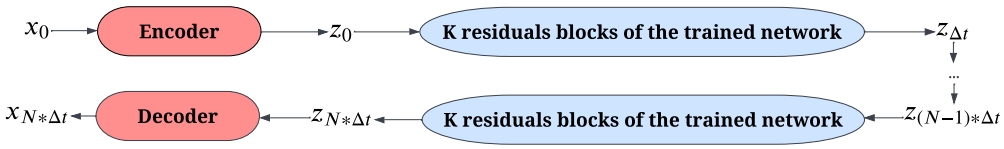


Figure 3.5: Latent-space auto-regressive forecasting.

Datasets *Advection equation.* We construct our first dataset with the following 1-D linear PDE,

$$\begin{cases} \partial_t u(x,t) = -\frac{1}{4}\partial_x u(x,t), & x \in (0, 2\pi),\ t \in \mathbb{R}^+ \\ u(x,0) = f_0(x), & x \in (0, 2\pi) \end{cases}$$

Burgers' equation. For the second dataset, we consider the following 1-D non-linear PDE,

$$\begin{cases} \partial_t u(x,t) + \partial_x(u^2(x,t)/2) = \partial_{xx} u(x,t), & x \in (0, 2\pi), t \in (0, 1] \\ u(x,0) = u_0(x), & x \in (0, 2\pi) \end{cases}$$

Both equations have periodic boundary conditions. We approximate the mapping from the initial condition f_0 to the solution at a given discretization time Δt, i.e. $u_0 \mapsto u(\cdot, \Delta t)$. We then forecast to longer time ranges. The time step Δt is set to 1 for the *Advection equation* with a grid of 100 points and 0.0005 for *Burgers' equation* with a grid of 128 points. Initial conditions in space correspond to a Gaussian where the amplitude and the mean are changed to produce different trajectories.

Baseline methods We compare our Implicit ResNet with respect to an Explicit ResNet with ReLU activation function and a Fourier Neural Operator (FNO) (Li et al., 2020b). We have also implemented two Explicit ResNet, with a tanh activation function and with batch normalization. We design the Explicit ResNet in the same way as our implicit version, with K layers of residual blocks that are linked by $z_{n+1} = z_n + R_n(z_n)$. Traditional methods forecast by using the output of the network at time t to predict the dynamics at time $t + \Delta t$. Consequently, to predict at time $N \cdot \Delta t$ from time $t = 0$, the baseline networks are iteratively applied N times, as illustrated in Figure 3.4.

Results Prediction errors are reported in Table 3.4 for the *Advection equation* and *Burgers' equation*. We also show the error according to the forecast time in Figure 3.6.

We find that the baseline methods diverge with the forecast horizon. They reach a MSE of more than 10^8 for the *Advection equation* and go to infinity for *Burgers'*

equation, respectively at time 400 and 0.15. Moreover, we see in Figure 3.6 that divergence in time is convex, so the increase in error is accelerating over time. We also find that our proposed Implicit ResNet presents better results by several orders of magnitude for both datasets. Moreover, we can see in Figure 3.6 that its curve in time is reaching a stable plateau, as expected from our theorem 3.2.3.

As for the training, traditional deep learning methods manage to learn very well the dynamics at $t = \Delta t$, with an MSE of two orders of magnitude better than our Implicit ResNet. However, the latter still manages to learn well the dynamics with a MSE of 10^{-2} for the *Advection equation* and 10^{-3} for *Burgers' equation*. This difference in training can mainly be explained by the longer training time of the Implicit ResNet, which made us take a smaller number of epochs for this network (1250 against 2500).

Table 3.4: Results of our approach compared to baselines on the *Advection equation* and *Burgers' equation*. We calculate the means and standard deviations of MSE for each model based on 5 runs with different seeds. The mid-range time is 40 for the *Advection equation* and 0.075 for *Burgers'* and the long range time is respectively 400 and 0.15. Recall that $\Delta t_{adv} = 1$ and $\Delta t_{bur} = 0.0005$. The train error is evaluated on trajectories at time Δt seen during training, and test error on trajectories at time Δt not seen during training. The forecast errors are computed on trajectories at time T given in the description for trajectories not seen during training.

	Model	Train Error ($\times 10^{-4}$)	Test Error ($\times 10^{-4}$)	Forecast error at mid-range $T_{adv} = 40 \cdot \Delta t_{adv}$ $T_{bur} = 150 \cdot \Delta t_{bur}$	Forecast error at long-range $T_{adv} = 400 \cdot \Delta t_{adv}$ $T_{bur} = 300 \cdot \Delta t_{bur}$
Advection	Explicit Res Net	**0.03 ± 0.01**	**0.09 ± 0.07**	0.25 ± 0.33	$4.7 \cdot 10^{31} \pm 1.0 \cdot 10^{32}$
	FNO	**0.04 ± 0.01**	**0.1 ± 0.08**	**0.03 ± 0.04**	$4.7 \cdot 10^{8} \pm 1.0 \cdot 10^{9}$
	Implicit ResNet (Ours)	14.0 ± 9.0	25.0 ± 27.0	27.4 ± 24.0	**27.5 ± 24.2**
Burgers'	Explicit Res Net	0.17 ± 0.03	0.90 ± 0.38	$2.77 \cdot 10^{19} \pm 6.2 \cdot 10^{19}$	$+\infty$
	FNO	**0.02 ± 0.002**	**0.03 ± 0.006**	$5.31 \cdot 10^{10} \pm 11.2 \cdot 10^{10}$	$+\infty$
	Implicit ResNet (Ours)	4.90 ± 0.64	7.91 ± 0.30	**0.67 ± 0.43**	**0.66 ± 0.44**

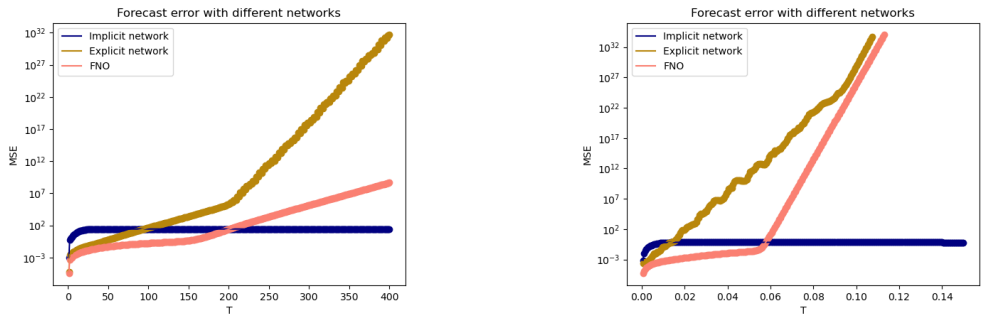


Figure 3.6: Forecast error for different neural network architectures for the *Advection equation* (left) and *Burger's equation* (right).

Discussion Figure 3.6 demonstrates the main benefits of our constrained implicit neural network. Our network is stable whereas the other methods diverge in time. This is of great interest, since our method does not suffer from error accumulation producing infinite errors. However, although being stable and far better than the baselines, it does not manage to forecast accurately the long-term dynamics. This is further confirmed by Table A.3, which shows high relative errors. Said otherwise, when stability is guaranteed, convergence is not. However, some kind of consistency is achieved, since the error at short-term times is low. We can also note that constraining the weights makes our network harder to train, but guarantees structural forecasting stability.

3.2.4 Conclusion

In this work, we studied the challenging task of long-term forecasting in dynamical systems. To do so, we developed a theoretical framework to analyze the stability of deep learning methods for forecasting dynamical systems. We then designed a constrained implicit neural network out of this framework. To the best of our knowledge, this is the first work that proposes to study deep learning architectures for forecasting dynamical systems from a numerical schema standpoint. We showed improved results with respect to deep learning baselines for two transport PDEs.

This work opens new perspectives to study neural networks forecasting stability from a numerical schema standpoint, thus offering more robust architectures. However, this analysis still needs improvements. Even though it ensures forecasting stability of our proposed network, it does not guarantee good convergence properties. We believe that developing this line of research could help overcome these challenges, and provide more robust architectures for forecasting dynamical systems in long time range.

Chapter 4

Deep Learning and differentiable solvers

This chapter is a study of the use of DL methods inside differentiable solvers applied to the Shallow Water equations. These equations contain a friction law but there is no consensus on the formulation of this law. This parametrized law is then often optimized without too much theoretical justifications (Delestre, 2010). In this work, we aim to replace this friction law with a data-driven model integrated in the equations and thus interacting with a numerical solver. This leads to the design of a hybrid model that can learn from observations to reproduce an effective friction law. This work is a collaboration with Emmanuel Audusse and will soon be submitted to a journal.

Migus, L., Salomon, J., Audusse E. and Gallinari, P. . Learning a friction law for the Shallow Water equations through observations. *Soon to be submitted to a computational physics oriented journal.*

4.1 Introduction

The study of PDEs is crucial to better model physical phenomena. They can be applied to lots of different domains, such as fluid dynamics or climate science. Their analytical formula can however contain terms that need to be adapted to a context. This challenge is usually tackled by using optimization techniques to provide parameter estimation. In some case, these parameters are not thoroughly motivated by physics, hence the estimation is purely data-driven. In addition, the solutions of PDEs are almost always obtained using numerical schema, which leads to numerical errors. These two problems lead to poor generalization to experimental data and new frameworks. Using a more efficient data-driven method is then a sensible line of research. Given the recent improvements of data-driven methods in the Deep Learning (DL) community, we present in this paper a solution to these problems using DL.
In this way, we focus on a modeling problem related to the Shallow Water equations. These equations represent various physical systems, from a dam breaking to water moving in a river, by modeling the water height and speed. One main challenge of these equations is the design of the friction law. A few general forms have been accepted in the community, but none of them gives raise to a consensus (Delestre, 2010). This is mainly due to the fact that in this model, the speed is supposed constant with respect to the height. However, the friction is localized in the bottom of shallow waters. Then, the modeling of the friction law is inherently non-physical. This is a similar problem as turbulence modeling in the Navier-Stokes equation, where several different models are used and discussed (Sagaut et al., 2013). So the Shallow Water equations are a natural application to use recent data-driven methods, by guessing a friction law purely from observations without parametric prior.
In order to design a purely data driven law, without supposing prior knowledge of the underlying friction law, a function can only be inferred through observations. Indeed, the friction law depends on the water height and speed, allowing for its reconstruction by observing these two variables. This requires learning through a numerical scheme, which offers an estimation of the observations based on the friction law. Consequently, the learned law needs to be adjusted accordingly through the scheme. This framework allows to tackle different settings and can be easily applied to experimental data, by correcting discretization errors in the model, given a reasonably performing differentiable scheme. In what follows, we show the different generalization properties of our method on a ODE case of the Shallow Water equations, before focusing on the more complex dynamic PDE case.
This article is organized as follows. In Section 4.2, we present the related work, with a paragraph on the Shallow Water equations friction laws and another one on hybrid differentiable physics for deep learning. It notably describes (Yin et al., 2021), which is our main reference for this method. In Section 4.3, we define more pre-

cisely the Shallow Water equations and the cases that we will study. In particular, we derive the stationary ODE case. In Section 4.4, we describe the neural network architecture we used as well as the different numerical schemes and the training procedure. In Section 4.5, we show the training of our method on the ODE case and compare it by directly training on the friction law. This gives a perspective on what can be achieved with our method, considering that the latter case does not correspond to experimental setting. This highlights the improvements obtained when learning directly a given friction law, along with strong overall results for our approach. In Section 4.6, we study the robustness of our approach with respect to noise, to friction laws, to the considered numerical scheme and its discretization and to the initial guess. In Section 4.7, we show that our method is still efficient on a complex dynamical setting.

4.2 Related work

Friction law choices for the Shallow Water equations The Shallow Water equations are described in Section 4.3.1 and defined by Equation (4.1). Various empirical friction laws exist in the literature, mostly derived from the experimental analysis of flows in pipes and open channels (Flamant, 1891). Among all these laws, two main families of classical laws are often considered in hydrology, namely, the laws of Manning and Strickler and the laws of Darcy-Weisbach and Chézy. The laws of the Manning and Strickler family can be written as $S_f(h,u) := \frac{C_f|u|u}{h^{4/3}}$, where C_f is the coefficient of friction. It can be written in several ways. By $C_f := n^2$, Manning's law of friction (Hervouet, 2003; Smith et al., 2007) is obtained, where n is the Manning coefficient. There are tables that reference values for n as a function of soil (see among others the site of the software FishXing version 3[1]). Most of these tables have been developed from the work done in French (1985). With $C_f := \frac{1}{K^2}$, Strickler's law of friction (Viollet et al., 2003; Hervouet, 2003) is obtained, where K is the Strickler coefficient. The Darcy-Weisbach and Chézy family laws can be written as $S_f(h,u) := \frac{C_f|u|u}{h}$. Here again C_f can be written in several ways. With $C_f := \frac{1}{C^2}$, the law of Chézy (Hervouet, 2003; Smith et al., 2007) is obtained, where C is the Chézy coefficient. With $C_f := \frac{f}{8g}$, the Darcy-Weisbach law (Viollet et al., 2003; Chanson, 2006; Smith et al., 2007) is obtained, where f is the Darcy-Weisbach coefficient. In some hydrological models, these coefficients are replaced by non constant terms. In Fiedler and Ramirez (2000); Dunkerley (2002); Raff and Ramírez (2005), we find a Darcy-Weisbach friction law with the coefficient $f = \frac{K_0}{R_e} = \frac{\nu K_0}{|Q_e|}$, where ν is the kinematic viscosity, K_0 depends on the nature of the soil, and $R_e := \frac{|Q_e|}{\nu}$ is the Reynolds number (which is interpreted as the ratio between inertial and viscous forces). In this case a linear friction law in u is found; $S_f = \frac{\nu K_0 u}{8gh^2}$. In Lawrence (1997); Smith et al. (2007), friction coefficients depend on

the size of the soil micro-rugosities.
Manning's and Darcy-Weisbach's laws with constant friction coefficients are widely used in hydrology. All these friction laws can be written in the following general form $S_f = K_f(h,u)|u|$. In the rest of this work, we will try to model K_f. This section is highly inspired by Delestre (2010).
As we can see, there is no consensus for the modeling of the friction law in the Shallow Water equations. That is what motivated our use of deep neural networks to model the friction law from the data-driven paradigm.
In addition to the research on the friction law formulation, many computational tools have been proposed to evaluate Manning's coefficient n, which is of crucial importance for river flood prediction for instance. This can be particularly challenging since, in real-world experiments, n appear to be space dependent, so that significant computational efforts are required to get an accurate estimation. This problem is often tackled in terms of an optimization problem in the literature, see Agresta et al. (2021); Ayvaz (2013); Askar and Al-Jumaily (2008); Birgin and Martínez (2022); Ding et al. (2004); Ding and Wang (2005); Marcus et al. (1992). These works try to provide or apply efficient optimization algorithms to this problem. A challenge of using data-driven techniques is that the optimization must be linked with the simulation. This line of research explored the use of optimization algorithms to provide an accurate estimation of n in real-world applications, resulting in a parametric friction law with a space dependent parameter.

Hybrid differentiable physics for deep learning Several lines of work in the deep learning community explore combining numerical schemes and neural network. In this way, Chen et al. (2018) paved the way by explicitly showing links between common deep neural architectures and numerical solvers. However, this work and the related papers focused on improving neural network architectures and training for a variety of tasks. More specifically applied to physics problems, (Thuerey et al., 2021) has developed a lot of methods around the use of differentiable physics simulations for deep learning; error correction (Um et al., 2020), PDE control (Holl et al., 2020) or inverse problems (Holl et al., 2021). (Belbute-Peres et al., 2020) used similar ideas of a neural network combined with a differentiable simulator. They trained a graph neural network to predict on a finer mesh from a coarse resolution given by the simulator, which improves the computational time. In this paper, we follow the approach proposed by APHYNITY (Yin et al., 2021). In this framework, data-driven models offset incomplete physical dynamics. The basic idea is to learn the error between a simple physics model and reality. The learning problem is formulated such that the physical model explains as much as possible the data, while the data-driven component only describes information that cannot be captured by the physical model. This paradigm is different from the one considered in this paper. Indeed, instead of minimizing a data-driven component, we aim here at completing

a Shallow Water model by a learned friction law which is a plain part of the model. However, the paradigm of APHYNITY is closely related to ours in its formulation. Indeed, given a model $\frac{dX_t}{d_t} = F(X_t)$, the APHYNITY approach decomposes F into $F_a + F_p \approx F$, where F_p encodes the incomplete physical knowledge and F_a is the data-driven augmentation term complementing F_p. F_a is a parametric function with only a few parameters, whereas F_p has a free form and many parameters to optimize. Their optimization problem is then formulated as $\min_{F_p \in \mathcal{F}, F_a \in \mathcal{F}} \|F_a\|$ subject to $\forall X \in \mathcal{D}, \forall t, F(X_t) = (F_p + F_a)(X_t)$. This is a joint constrained optimization, which is quite complex to perform. In practice, the condition on F is ensured by using a solver, hence this condition depends on the solver and the discretization. Our problem on the other hand is formulated similarly, but for a PDE, namely

$$gh\partial_x b - \partial_t(hu) - \partial_x(hu^2 + \frac{gh^2}{2}) = uhK_f(h, |u|)$$
$$\partial_t h + \partial_x(hu) = 0.$$

In this case, $X = (h, hu)$, $Fp = (-\partial x(hu), -\partial x(hu^2 + gh^2/2) - gh\partial xb)$ and $Fa = (0, Khu)$. On the contrary to APHYNITY, the optimization problem is formulated as $\forall X \in \mathcal{D}, \forall t \min_{F_a \in \mathcal{F}} \|(F_p + F_a)(X_t) - F(X_t)\|$. In practice, similarly to APHYNITY, the optimization is performed on a sequence obtained with a numerical solver, thus being sensible to the scheme and the discretization. Our problem is an unconstrained problem with only one variable to optimize, since F_p is fixed in our case. Indeed, all terms in F_p are known, for instance the gravity g. An extension of APHYNITY could be implemented for modified Shallow Water equations where there is a parameter in front of the pressure term to take into account that the vertical profile of speed is not constant. However, this is not assumed in this work.

4.3 Problem

This work focuses on the Shallow Water equations. These equations can represent water moving in a river, a swimming pool or a dam breaking for instance. We will study a few different scenarios.

4.3.1 Shallow Water equations

Introduced in Saint-Venant et al. (1871), the Shallow Water equations can be written as:

$$\begin{aligned} \partial_t h + \partial_x(hu) &= 0 \\ \partial_t(hu) + \partial_x(hu^2 + \frac{gh^2}{2}) &= -gh\partial_x b - uhK_f(h, |u|) \end{aligned} \tag{4.1}$$

where u is the water speed, h the water height, b the topography, g the gravitational constant and K_f the friction law. An example of friction that we will take for our experiments is:

$$K_f(h,u) = \frac{C_f * |u|^\alpha}{h^\beta}. \tag{4.2}$$

As detailed in Section 4.2, the choice of the friction K_f is usually motivated by numerical experiments. Depending on the problems, some values of C_f, α and β are fixed. Since this choice is not entirely motivated by physics, we would like to have a pure data-driven choice of K_f. Hence, we design a neural network K_{NN} to help in better approximating solutions of the Shallow Water equations. The resulting system reads:

$$\partial_t h + \partial_x(hu) = 0 \tag{4.3}$$

$$\partial_t(hu) + \partial_x(hu^2 + \frac{gh^2}{2}) = -gh\partial_x b - uhK_{NN}(h,u). \tag{4.4}$$

In this work, unless stated, we will use the Manning and Strickler family of laws, i.e. $\beta = \frac{4}{3}$ and $\alpha = 1$. We will also set C_f to 0.2, which is equivalent to a Manning coefficient n of 0.04. The positivity of the output of the neural network can be easily guaranteed by changing $K_{NN}(h,u)$ into $|K_{NN}(h,u)|$.

4.3.2 Case of study

In order to learn the friction, we consider different settings. Each setting is chosen to ensure that our framework is flexible, from a more simple stationary setting to a harder dynamic setting.

4.3.2.1 River

Description As a first setting, we choose a river setting controlled by an upstream water flow and a downstream water height. Such experiments aim at studying stationary configurations, which are obtained after some transient states. Real data can then be obtained by extracting quantitative data from flows pictures. It follows that to get close to this setting, we derive a new set of equations.
This stationary regime is described by an ODE. This regime is derived from Equation (4.1) by setting the derivative in time to 0. As can be seen in Equation (4.5), the flow is then constant and equal to the upstream flow Q_e. This is ensured only if the regime does not change. For a flat topography, the resulting system reads:

$$\begin{aligned} \frac{d}{dx}(\frac{Q_e^2}{h} + \frac{gh^2}{2}) &= -Q_e K_f(h, Q_e) \\ h(x_{final}) &= h_s, \end{aligned} \tag{4.5}$$

So that:

$$\frac{d}{dx}h(x)(-\frac{Q_e^2}{h^2}+gh) = -Q_eK_f(h,Q_e) \tag{4.6}$$

$$\frac{d}{dx}h(x) = -Q_eK_f(h,Q_e)\frac{h^2}{gh^3-Q_e^2}. \tag{4.7}$$

We consider a stable subsonic fluid. This property holds if $Q_e \leq \sqrt{gh}h$, otherwise the denominator is null at some point, which induces two zones, a supersonic one and a subsonic one. The passage from one zone to another implies different phenomena than the phenomenon considered with the ODE. Since $\frac{d}{dx}h(x) \leq 0$, h increases from right to left, so that $Q_e \leq \sqrt{gh_e}h_e$ is a sufficient condition. The term $Q_e = \sqrt{gh_s}h_s$ is called the Froude line. Also, since $Q_e = uh$, we have:

$$K_f(h,Q_e) = \frac{C_fQ_e}{h^{7/3}}. \tag{4.8}$$

Representative data sampling for the river case of study In order to have representative data, we sample the data in a particular manner. For each chosen Q_e, which is randomly sampled between 0.5 and 2, we first choose the downstream water height $h_s := h_f+\delta$, where h_f corresponds to the Froude limit, where $Q_e-\sqrt{gh_s}h_s = 0$ and $\delta = 0.25$. At the end of the corresponding simulation, for each Q_e, we then choose the new downstream water height h_s to be equal to the upstream water height h_e of the previous simulation. We repeat this operation only once, i.e. $N_{repet} := 2$. This algorithm is described in Algorithm 1. This allows us to have representative data for each couple of Q_e and h_s. We also choose a logarithmic scale for x to ensure a better distribution along the h_s axis. A density plot of the data is represented in Figure 4.1a. A comparison with taking random values of Q_e and h_s can be seen in Figure 4.1. This confirms that the sampling is more representative with the described algorithm. Figure 4.2 shows the true friction induced by this sampling for this case of study.
The setting we consider is as follows:

River setting

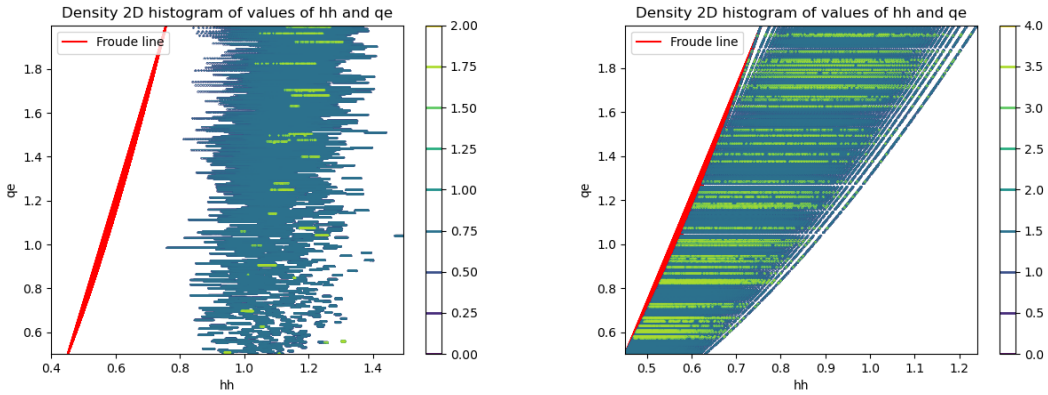
1. Fluvial boundary conditions with varying input flow between 0.5 and 2.

2. Fluvial boundary conditions with varying output height chosen accordingly to Q_e.

3. Flat topography.

Algorithm 1: Representative data sampling

for $i = 1, N_{traj}$ **do**
  $Qe \sim U(Qe_-, Qe_+)$;
  $hs = [\frac{Qe^{2/3}}{\sqrt{g}} + \delta]$;
  for $j = 2, N_{repet}$ **do**
    $h = Solver(x, x_0 = hs, Qe, K_f)$;
    $h_s = [h_s, h[-1]]$;
  end
end



(a) Histogram of the values of Q_e and h for a random data creation.

(b) Histogram of the values of Q_e and h for a more representative data creation.

Figure 4.1: Discrete histogram of values for random creation and more representative creation.

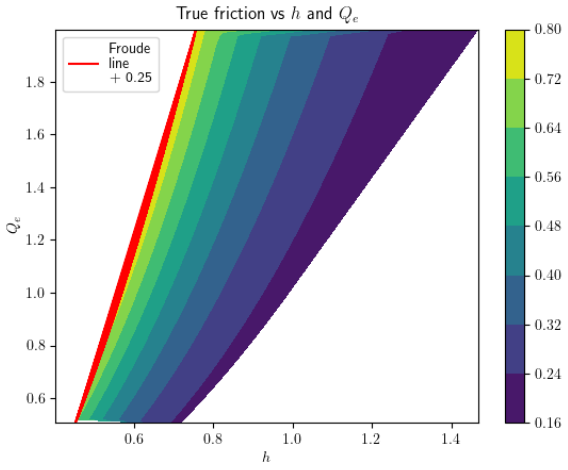


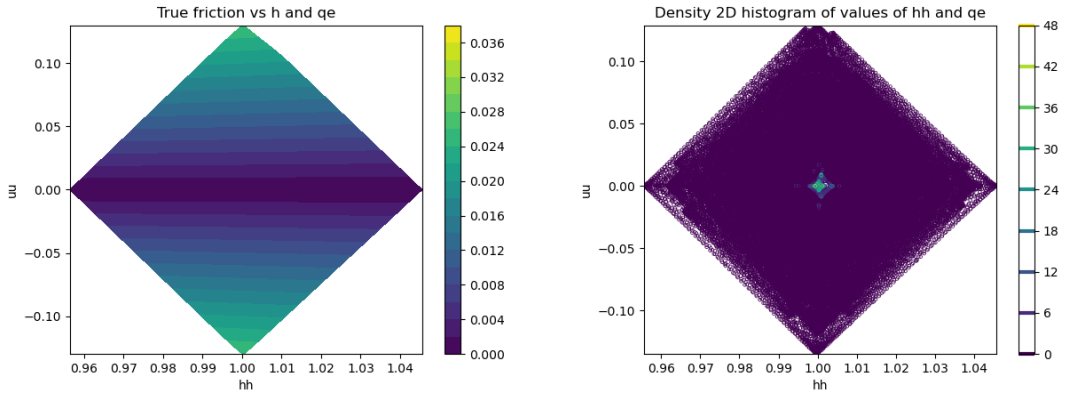
Figure 4.2: True friction for the stationary case.

4.3.2.2 Closed swimming pool

In this second setting, a closed swimming pool with sinusoidal initial conditions is considered. The property of interest in this case is the unsteadiness. This setting is close to a damped pendulum equation if the water height does not vary much. By increasing the oscillations of h, the range of values of h is higher, then the friction law is more complex, due to its exponential dependency on the water height. This leads to the design of three different settings: small variations one, medium variations and large variations. In the small variations setting, the friction is almost linear with h, hence being close to a damped pendulum equation. On the contrary, the medium variations setting showcases a greater influence of the water height, making it a typical use case. The large variations settings is intended to be more challenging, with great variations of the water height. The density plots and friction laws from these setting are shown for 5 trajectories in Figures 4.3, 4.4 and 4.5b. The three settings are detailed below:

Small variations setting

1. Periodic boundary conditions
2. Flat topography
3. Sinusoidal initial conditions, with little variation in the conditioning (variance of 0.001 for the mean and 0.05 in the amplitude).



(a) True friction. (b) Histogram of the values of Q_e and h.

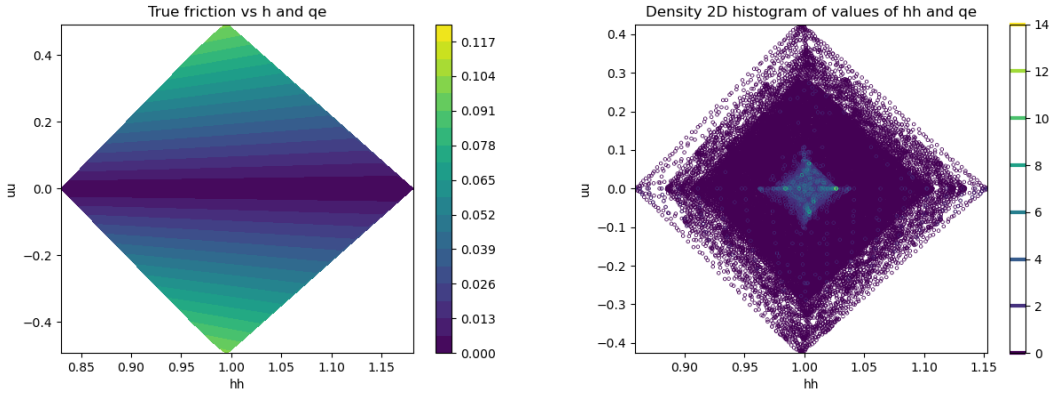
Figure 4.3: Small variations PDE setting true friction and density.

Medium variations setting

1. Periodic boundary conditions.

2. Flat topography.

3. Sinusoidal initial conditions, with medium variation in the conditioning (variance of 0.01 for the mean and 0.2 in the amplitude).



(a) True friction. (b) Histogram of the values of Q_e and h.

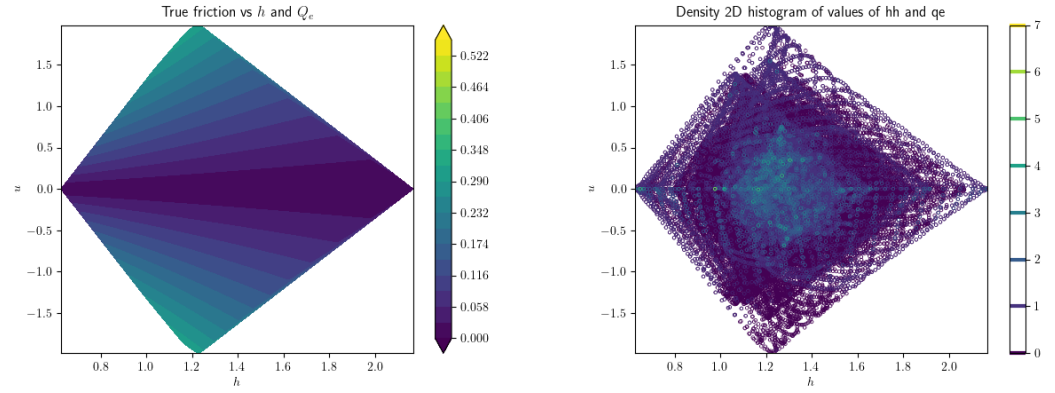
Figure 4.4: Medium variations PDE setting true friction and density.

Large variations setting

1. Periodic boundary conditions.

2. Flat topography.

3. Sinusoidal initial conditions, with medium variation in the conditioning (variance of 0.4 for the mean and 0.8 in the amplitude).



(a) True friction. (b) Histogram of the values of Q_e and h.

Figure 4.5: large variations PDE setting true friction and density.

4.4 Method

This section describes the method used in this article. It first describe the neural network architecture, before delving into the different numerical schemes used and finally illustrating the training process.

4.4.1 Neural Network

The neural network used in this work is a well-known deep learning architecture, namely a convolutional neural network (CNN) LeCun et al. (1989). More precisely, this network is a ResNet He et al. (2016) type architecture and consists of successive blocks of CNN with a residual connection. We consider an input I is of size (C_{in}, L_s), an output of size (C_{out}, L_s) and a kernel K of size K_k. The kernel is applied channel per channel. C_{in} is the number of channels of our input and C_{out} of the output. In our case, we have $N_C := C_{in} = C_{out}$, except for the first layer, where $C_{in} = 1$. The term L_s is the length of the sequence. In order for the output sequence to have the same length as the input, the kernel needs to receive a different input. Padding is used to solve this problem. This technique consists in adding p zeros on the left and on the right of the sequence. The integer p is chosen accordingly to K_k. We denote by pa the corresponding operation, then $I^{pa} = pa(I)$. A single convolution between K and I is defined by $\forall j \in [|1, C_{out}|], conv(I, K)_j := (B)_j + \sum_{k=0}^{C_{in}-1} (K)_{j,k} \star I_k^{pa}$, where $\star$ is the cross-correlation and B is a bias vector of size C_{out}. The kernel K is learned during the backward pass, and is different for each convolution in the architecture. We also define weight normalization as $wnorm(v) := g\frac{v}{\|v\|}$, with $g \in \mathbb{R}^{L_s}$ being learned. The norm is computed independently per output channel. We then have a residual block:

$$ResBlock(I) := \sigma \circ [(wnorm \circ conv)^3(I) + wnorm(WI + B)] \tag{4.9}$$

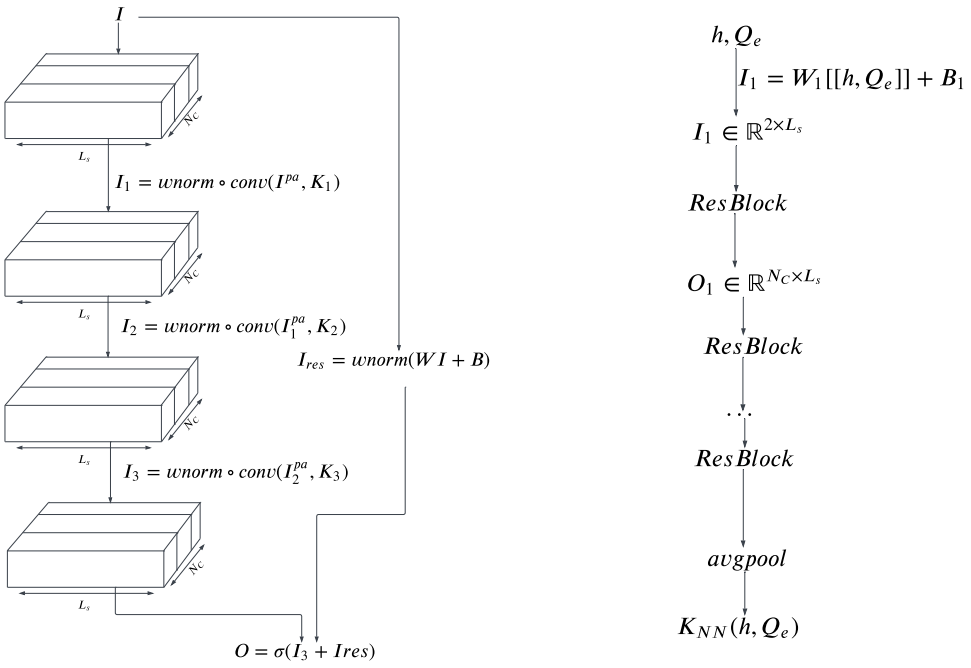
with σ an activation function and W a linear layer of size (C_{out}, C_{in}) and B a vector of size (C_{out}, L_s). The last layer $avgpool$ is an average over the channel and the sequence length dimensions. Moreover, the first input is transformed to be of size $(1, L_s)$ with a linear layer W_1 and a bias B_1, of size $(L_s, 1)$ and $(1, L_s)$. The whole network is then:

$$Network(I) = avgpool \circ ResBlock^{Depth}(W_1 I + B_1). \tag{4.10}$$

Figure 4.6 illustrates the architecture presented here. A few parameters can be tuned in this model:

- The sequence length L_s, which is 2 for the input, since we have h and Q_e. L_s is set to 10 for the other convolutions.

- The number of channels N_C, which is set to 64 for the ODE case.
- The kernel size K_k, which is set to 3. The padding p is set accordingly to 1 to obtain a stable L_s through the depth of the network.
- The depth of the network $Depth$.
- The activation function σ. A Leaky Rectified Linear Unit (LeakyReLU) is chosen and can be written as $\sigma(x) = \max(0, x) + \epsilon \min(0, x)$, with $\epsilon = 10^{-2}$.



(a) Residual block illustration.

(b) Whole network illustration.

Figure 4.6: CNN network architecture illustration.

4.4.2 Numerical schemes

In order to back-propagate the gradients through the neural network, the numerical scheme needs to be differentiable. For the river setting, the problem is encoded as an ODE, all the classical numerical schemes can be used, from an explicit Euler scheme to a Runge-Kutta of order 5 of Dormand-Prince-Shampine scheme. We will study the influence of the choice of the solver for the ODE case in section 4.6.1.3, with Chen (2018) for the implementation. The swimming pool example is more complex and encoded by a PDE. In this case, we use a specific numerical scheme, which is differentiable in the domain we use it. The scheme is a Rusanov scheme (Bouchut, 2004), and is not differentiable only when the water height is null, which

is not interesting to study and does not occur in our experiments. Unless specified, we use grids of size 100.

4.4.3 Training

As explained above, the loss is back-propagated through the differentiable solver and then through the network. Training and inference are illustrated in Figure 4.7. In the PDE case, the combination of time and space discretization increases significantly the computational cost. We therefore constraint the number of back-propagation in time to remain under a given limit.
Training a neural network is learning a mapping from the data without over-fitting it, so that the network can generalize to new cases. In order to achieve this, the training process is inherently stochastic with the use of Adam optimization algorithm, which is an extension to stochastic gradient descent. Tuning the learning rate l_r can reduce this influence, but obtaining a null error on the training set is not the goal. There is a trade-off between exploring the parameter space and reducing the variance of the results.
The training is performed with a linear scheduler, which improves the performances. The idea behind a scheduler is to reduce the learning rate l_r as the training is improving. A linear scheduler decreases this rate linearly by γ a given number of steps, which is chosen to be 8 for the ODE case. For the ODE case, we set the number of epochs to 1200 and the batch size is 64. For the PDE case, we cannot parallelize the examples, since the scheme has not a fixed temporal step. As a consequence, the batch size is 1. This can also cause alignment issue when comparing to ground truth. The ground truth reference solution must be created with the same temporal steps as the network solution. Another problem occurring for spatio-temporal schemes during training is the propagation through different time steps. In order to solve this challenge, the whole scheme is run again at time step and the loss is back-propagated, hence there is no problem of alignment and the network is updated regularly. Indeed, if the propagation is done without restarting the scheme, then the network is not consistent due to the error accumulation of the network. In our case, this is limited since the network improves at each re-run of the scheme.
For the initialization of the network, we use a uniform Xavier initialization Glorot and Bengio (2010), with a gain set to 4. For the ODE case of study, we performed a broad grid search as can be seen in Table 4.1.
In DL, the data is usually split into two sets, the training set and the test set. The training set is used to update the weights of the neural network. It is usually split into a training set and a validation set. The latter is used to verify at each step of the training if the network is over-fitting. The test set is distinct from the train set and not used during the training, only to test if the network can generalize to

Table 4.1: Hyper-parameter grid search table for the river case of study.

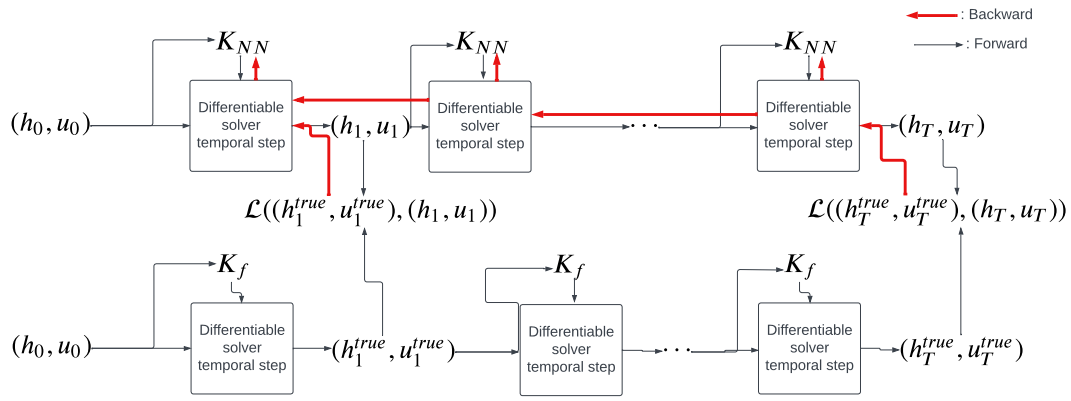
Network N_C	Network *Depth*	Optimizer γ	Optimizer lr
32	1	0.2	0.0005
64	3	0.4	0.005
128		0.8	0.05

trajectories not seen during training. Different losses can be used for the training phase. We will use the mean-squared error (MSE), which is defined by:

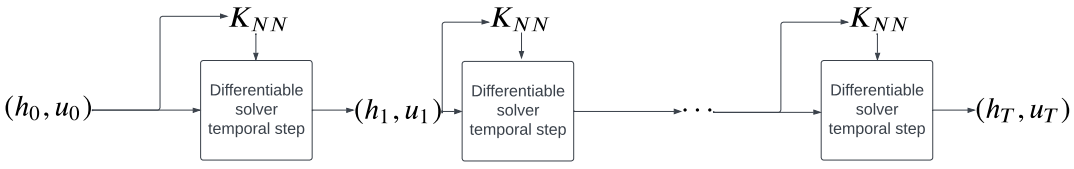
$$\text{MSE}(x, y) = \|x - y\|_2.$$

For testing, we will usually use the mean absolute error (MAE), defined as:

$$\text{MAE}(x, y) = \|x - y\|_1. \tag{4.11}$$



(a) Training through a temporal differentiable solver.



(b) Inference through a temporal differentiable solver.

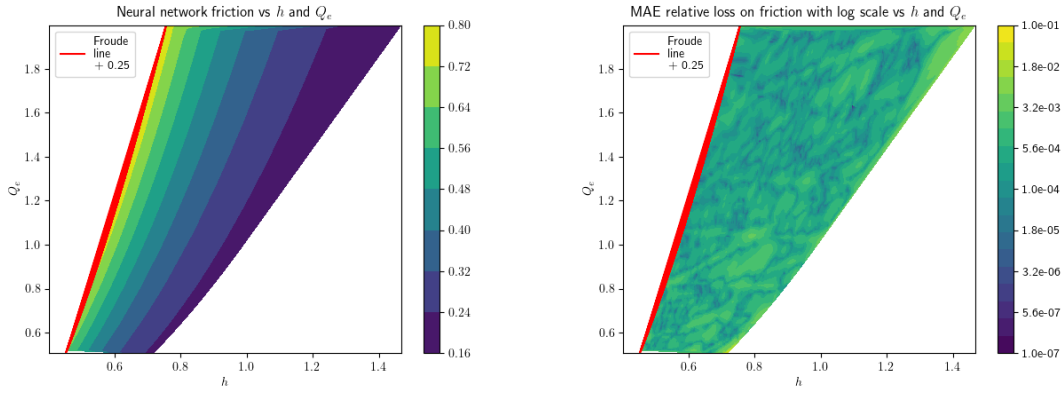
Figure 4.7: Training and inference through a temporal differentiable solver. The ODE case corresponds to stopping at the first time step.

4.5 Learning the friction through a scheme vs learning the friction directly

In order to learn the friction law, we suppose that it is not known, so that Equation (4.1) is not algebraically closed. However, to make a precise analysis of our method, we consider in this work synthetic data, i.e., data generated by numerical simulation where the friction law is fixed. In this framework, we can compare learning directly the friction and learning it through a scheme, i.e. through observations. In the direct learning, the training simply consists in learning a known function, and the in the non-direct learning, it is more complicated since the results depend on the choice of the numerical solver and the discretization and their impact on the precision of the results. This comparison sets a reference with the direct learning and thus allows for the study of the loss induced by learning through a scheme. Moreover, learning directly helps developing ideas to improve the non-direct learning. However, as explained earlier, this baseline does not solve the problem at hand, since it assumes a known friction law. This section will be restricted to the study of the river case.

4.5.1 Learning directly

In this first part, the friction is learned directly with a neural network, with the same points that the network trained with observations will get. The friction law is learned very accurately, as can be seen in Figures 4.8 and 4.11. The network obtains a MAE, metric defined in Equation (4.11), around 1e−4 and a visually coherent friction law. Moreover, as can be seen in Figure 4.8b, the relative error is less than 2% everywhere and less not on the boundaries.



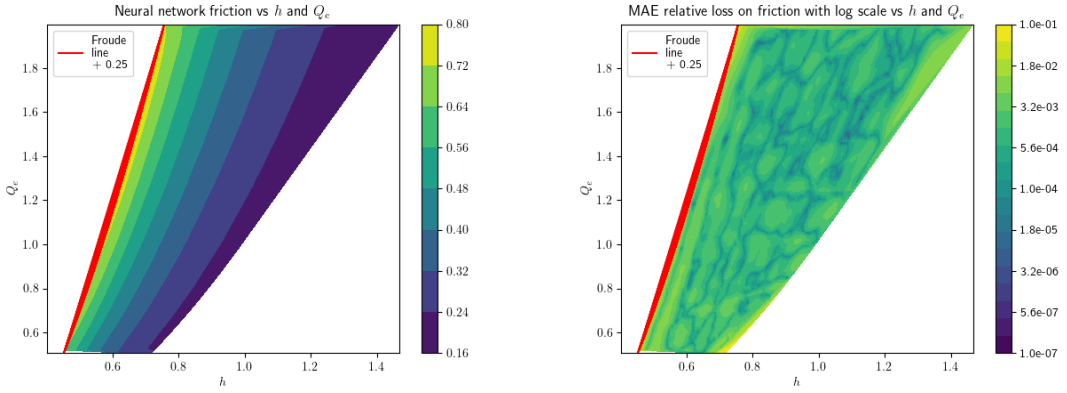
(a) Learned friction.

(b) Relative MAE on friction.

Figure 4.8: Friction laws of the model and the neural network by learning directly.

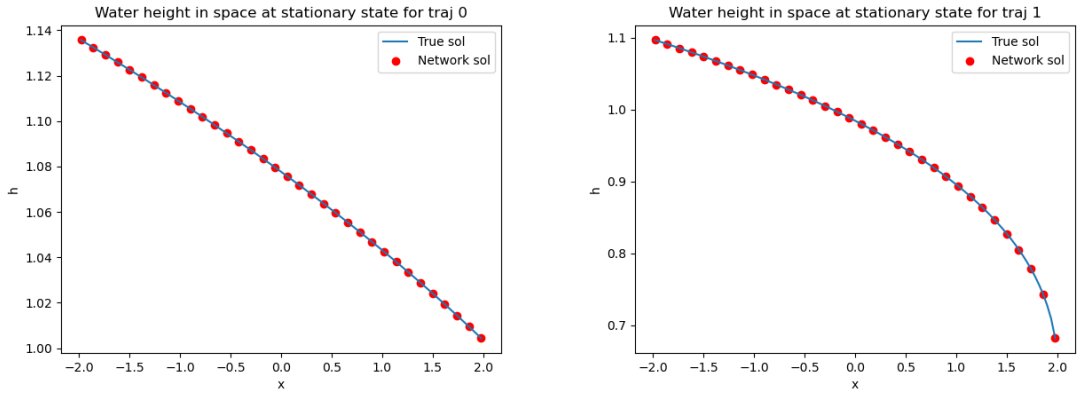
4.5.2 Learning through a scheme

In this more challenging setting, computed with a RK4 scheme with 100 points, learning through a scheme yields also very good performances. The friction law is learned quite accurately, as can be seen in Figures 4.9 and 4.11. The network obtains a MAE error around 5e−4 and a visually coherent friction law. Moreover, as can be seen in Figure 4.9b, the relative error is less than 10% everywhere and less not on the boundaries. Since the network has been given only the observations, the information on which it should fairly be tested is the reconstruction of trajectories of the water height. This can be seen on Figure 4.10.



(a) Learned friction. (b) Relative MAE on friction.

Figure 4.9: Friction laws of the model and the neural network by learning through a scheme.



(a) Trajectory 1. (b) Trajectory 2.

Figure 4.10: Two trajectories of water height states predicted by the solver with the neural network trained through it. They correspond to two different initial conditions. The network solution is plotted in red and the true solution in blue.

4.5.3 Discussion

As can be seen in Figure 4.11 and in Table 4.2, learning directly the friction law leads to better results. More precisely, for the test set, i.e., new trajectories, as presented in Table 4.2, the MAE error on the friction is more than two times lower for the direct learning, and the MAE on the height is four times lower for the direct learning. This is to be expected, since by learning through a numerical scheme, the neural network needs to interact with the scheme. However, the results obtained by learning through the schemes are very good and the friction law proposed by the network is coherent qualitatively. Additionally, as can be seen in Figure 4.11, the training of non direct learning leads to better results for the water height on the training set. This is to be expected since the optimization target is the water height, as is the friction for the direct learning. However, for the test set, the direct learning leads to better errors on both, since guessing the right friction is crucial for accurately predicting water height in general settings. Hence, the non-direct learning leads to better results on the training set, i.e. trajectories seen during training, but worse results on the test, the network is over-fitting. Before diving deeper into the robustness of this type of learning, we have found that learning through observations on the water height and flow allows us to reconstruct accurately the friction law.

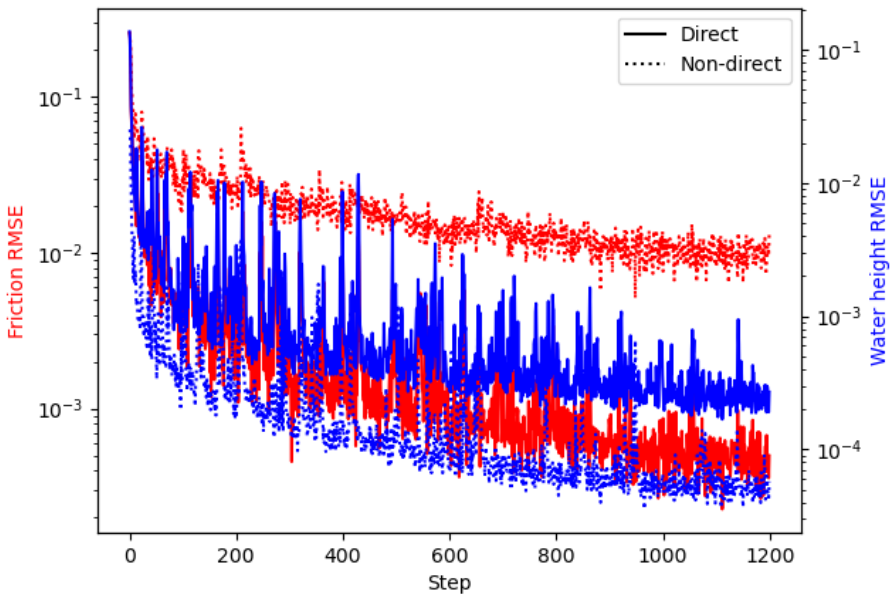


Figure 4.11: Training friction and water height losses for direct and non direct learning.

Table 4.2: Results of our approach against learning directly for different settings at test time.

Direct learning	Friction MAE ($\times 10^{-4}$)	Friction RMSE ($\times 10^{-3}$)	Height MAE ($\times 10^{-4}$)	Height RMSE ($\times 10^{-2}$)
✓	1.71	0.30	2.22	0.40
✗	5.45	1.55	7.95	1.44

4.6 Analysis of the ODE setting

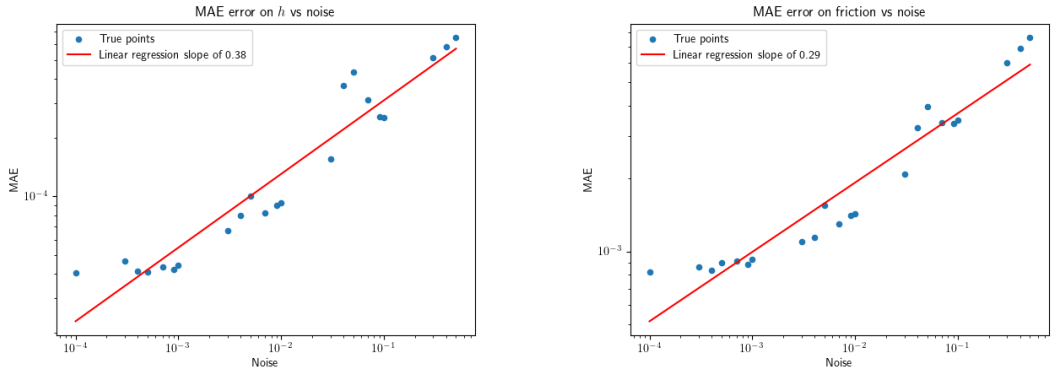
In Section 4.5, it was shown that the network could be effectively trained to learn an accurate friction law through the observations. We then want to test how robust this training is, and what are the influences of the different choices made. A first part is dedicated to the robustness study. It uses the same global setting as Section 4.5 and varies different elements. The first test is the robustness of the training to noise in the measurements of the observations. Then, the influence of learning different friction laws is analyzed, which is crucial to see if the network can be used in various settings. Furthermore, different numerical schemes are tested, in order to understand how the properties of different schemes affect the training. Finally, different initial guesses of the friction laws are used to see if the network can gain from some incomplete domain knowledge. This first part is a complete study of the robustness of the proposed approach, in the case of the river study. The second part is a convergence analysis. We suppose that a high fidelity true solution is available. Then, different discretization grid sizes and schemes are tested. This a realistic setting of the river case.

4.6.1 Robustness

This section is a preliminary study to test the robustness of our approach, which includes noise injection, variation of laws to retrieve, variation of schemes and tests of different initial guesses.

4.6.1.1 Learning with noise in the observations

When measurements of physical values are done, there is always a uncertainty on the values being recorded physically. It is important that models to be deployed are somewhat robust to this uncertainty. This can be modeled by noise. In this section, a Gaussian noise with a given standard deviation σ is added to the measurements of the water height. The flow Q_e is assumed not to be perturbed by noise. In practice, we train a neural network with noisy data with a varying noise level, and we then test it on a test set with no noise.

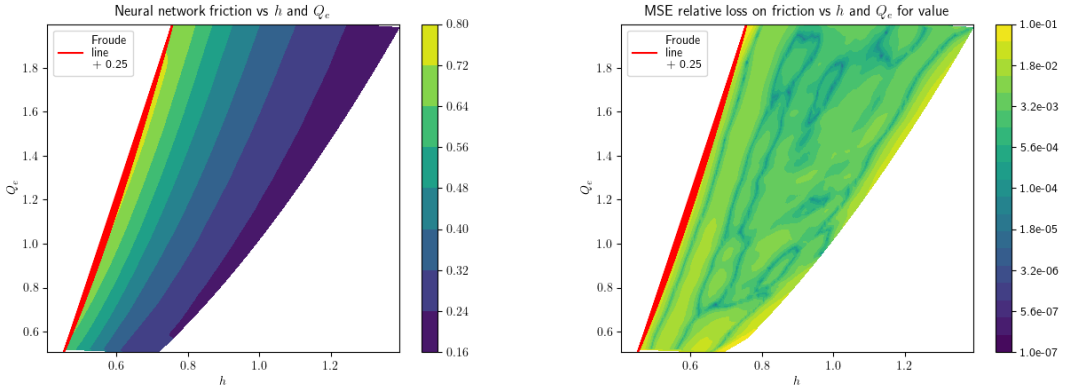


(a) Water height MAE vs noise.

(b) Friction MAE vs noise.

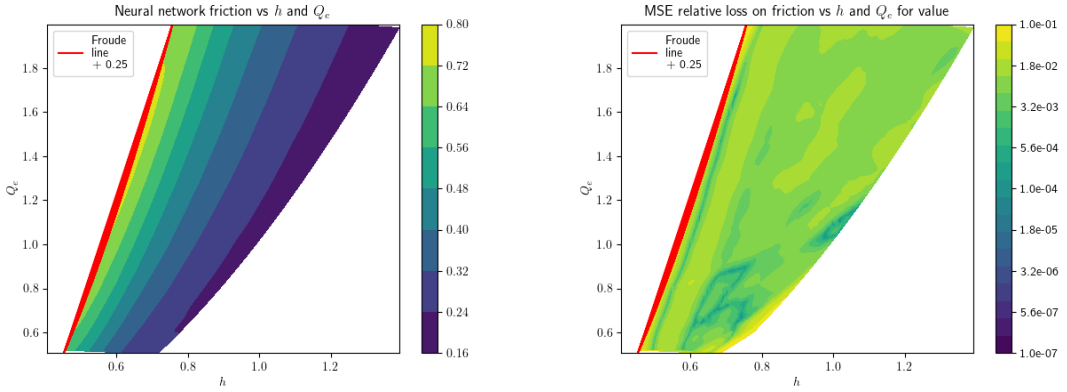
Figure 4.12: Noise study. The results are obtained on a test set with no noise, i.e. the law is learned with noisy data and the results presented here are the test on data with no noise.

In Figure 4.12, it is shown that the model error increases with the noise. The learning with noise produces less accurate friction laws when the data is more noisy. This is expected, since when the noise increases, it gets harder for the model to differentiate noise from the original data. However, this increase in the error is still reasonable, up to a certain level of noise, where the noise is unrealistically high anyway (more than 50% of the value). It is interesting to note that the increase in error is exponential when the noise increases. The MAE on the water height is exponential with a factor of 0.38 and the MAE on the friction is exponential with a factor of 0.29, as illustrated by the linear regression on the log-log plot in Figure 4.12. In Figures 4.13 and 4.14, it is shown that the model still maintain good qualitative results for the friction law with two levels of noise. Indeed, the relative errors are below 5% almost everywhere, and the worst errors around the top left and bottom right edges is still less than 10%. The relative error in the friction learned with more noise in Figure 4.14a is brighter than the one with less noise in Figure 4.13a, which indicates a higher error when the data is more noisy, which confirms the quantitative analysis from Figure 4.12.



(a) Learned friction. (b) Relative MSE on friction.

Figure 4.13: Friction laws of the model and the neural network with noise of 0.0125.

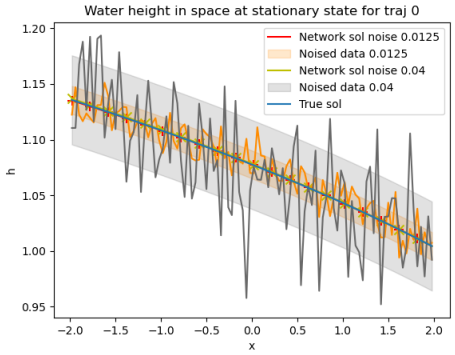


(a) Learned friction. (b) Relative MSE on friction.

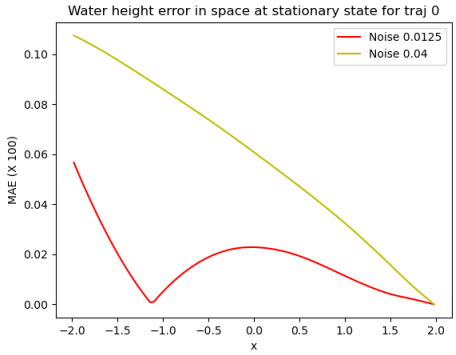
Figure 4.14: Friction laws of the model and the neural network with noise of 0.04.

In Figure 4.15, two trajectories associated with these levels are represented. The left plots show the different trajectories and showcase the complexity of the task, with noisy trajectories that are nothing close to the original noiseless ones, with many peaks. This highlights that the observations produced by the combination of the neural network and the numerical scheme are very good, since they are very close to the true one. The error map on the right side of Figure 4.15 also shows that with a higher noise, errors are more propagated through the trajectory leading to higher errors at the end of the trajectory (which corresponds to $x = -2.0$, going from $x = 2$). With a lower noise, the error accumulation is less pronounced and not always monotone as in Trajectory 1. The case of trajectory 1 might be explained by the fact that around $x = 0$ and $x = -1$ the values of h encountered have been seen during training, leading to a lower error. The loss in accuracy for the water height when the noise increases can then be highly attributed to error accumulation due to

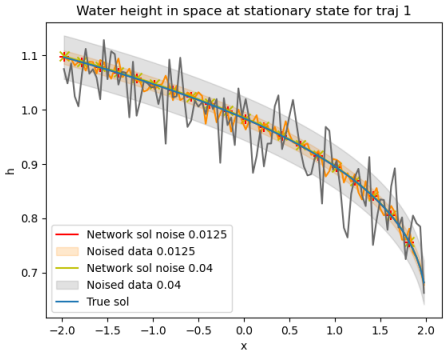
a less precise learned friction law. In addition to these results, Table B.1 shows the precise results about the increase of error with the noise, by presenting the MAE of the friction and the relative height MAE for a few different levels of noise.



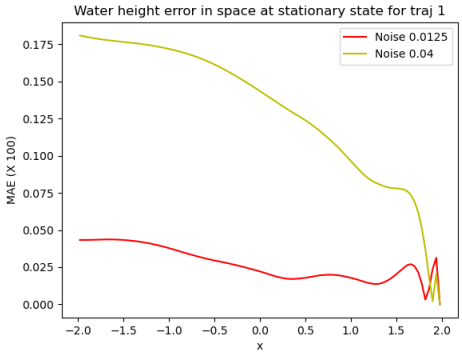
(a) Trajectory 1.



(b) Trajectory 1 error.



(c) Trajectory 2.



(d) Trajectory 2 error.

Figure 4.15: Examples of trajectories with noise of 0.0125 and 0.04. On the left side, the trajectory is plotted in blue and two example trajectories associated to the two levels of noise are in grey and orange, with the band around being the noise level. In green and red are the network trajectory solution trained with these noise. On the right, the errors between the true trajectory and the trajectories learned with noise are shown.

4.6.1.2 Testing different friction laws

As can be seen in Table 4.3, lots of choices for C_f are possible depending on the considered cases. In the overall work presented here, the value of C_f is fixed to 0.2. However, in order to test if the proposed method can be used in various cases, it is important to study if different models can be learned. Hence, in order to test the robustness of our approach, models with different C_f and β are learned. The coefficients C_f and β are presented in Section 4.3.

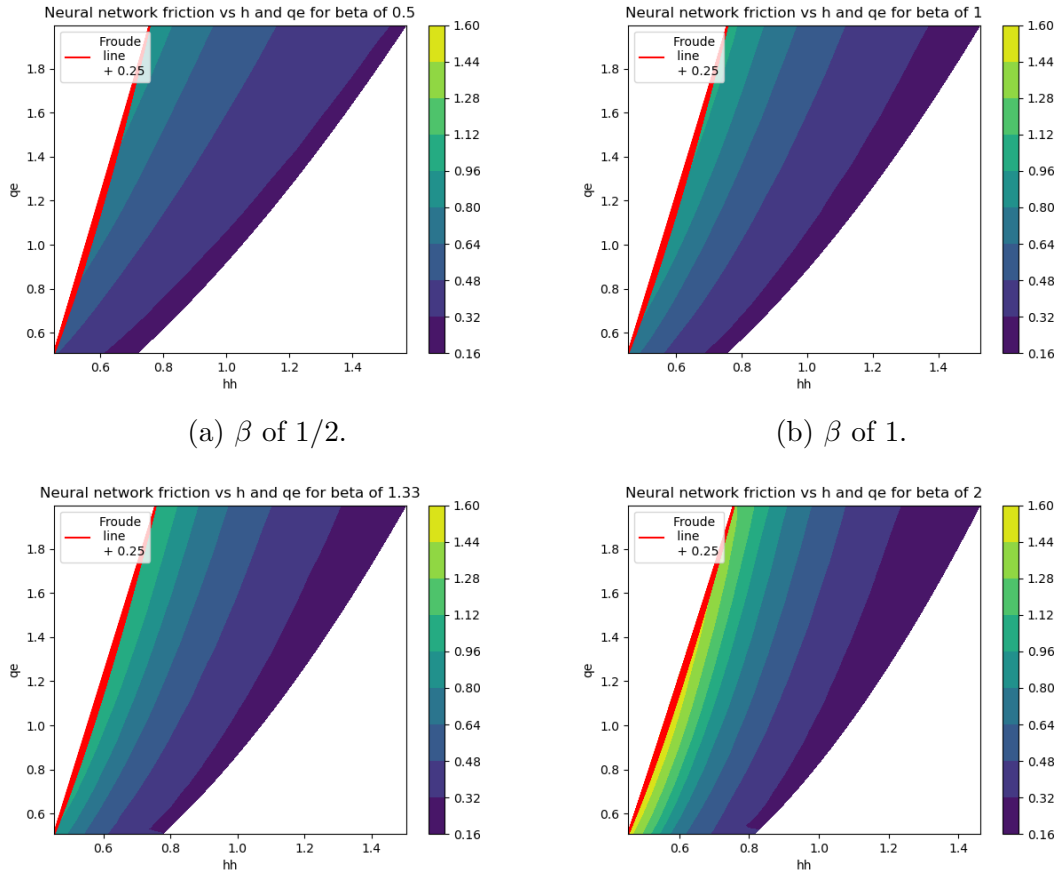
Table 4.3: Manning roughness coefficients for different land use classes. Table taken from Van der Sande et al. (2003). As defined in Section 4.2, the Manning coefficient n is the square root of the friction coefficient C_f, i.e., $C_f := n^2$.

Land use classes	**Manning roughness coefficient**	C_f
Residential building	0.200	0.447
Private/public garden	0.100	0.317
Grass in built-up area	0.259	0.509
Pavement/other urban area	0.050	0.224
Waterside	0.050	0.224
Sand deposit area	0.120	0.346
Road	0.013	0.114
Railroad	0.033	0.182
Industrial company/agency	0.200	0.447
Pasture	0.259	0.509
Winter wheat	0.127	0.356
Nursery	0.200	0.447
Fallow	0.120	0.346
Natural vegetation	0.100	0.317
Deciduous forest land	0.200	0.447
Mixed forest land	0.200	0.447
Water	0.030	0.173

Table 4.4: Height and friction errors for different friction laws. We have chosen different β and C_f. Friction MAE and relative MAE are expressed with factor $\times 10^{-3}$ and height with factor $\times 10^{-4}$.

β	**MAE**		**Relative MAE**		**MAE**		**Relative MAE**	
	Friction	**Height**	**Friction**	**Height**	**Friction**	**Height**	**Friction**	**Height**
	$C_f = 0.1$				$C_f = 0.2$			
1/2	0.20	0.22	1.10	0.29	0.29	0.16	0.91	0.21
2/3	0.27	0.32	1.43	0.42	0.34	0.18	0.96	0.24
1	0.34	0.24	1.56	0.34	0.45	0.21	1.14	0.28
4/3	0.41	0.26	1.73	0.40	0.63	0.32	1.40	0.42
2	0.46	0.23	1.56	0.32	1.29	0.46	2.48	0.62
	$C_f = 0.3$				$C_f = 0.5$			
1/2	0.84	0.43	1.81	0.54	1.34	0.58	2.04	0.63
2/3	1.07	0.57	2.23	0.70	1.57	0.65	2.32	0.73
1	1.56	0.74	2.94	0.92	1.94	0.78	2.66	0.85
4/3	2.10	0.85	3.78	1.08	2.56	0.88	3.28	0.99
2	1.80	0.51	2.81	0.64	3.32	1.16	3.72	1.23

In Table 4.4, the errors for the different friction laws are shown. When β increases, the training gets harder. This is due to the fact that β influences the law exponentially, hence when β increases the amplitude of the law varies a lot more, and it is harder to capture. The influence of C_f on the law is linear, but we can see that it changes the results a lot. When C_f increases, the amplitude increases, and it is harder to capture the law. A difference of more than an order of magnitude between the best and the worst cases for the MAE can be observed in Table 4.4. However, for all the different laws, the relative error on the friction is less than 0.5 %, namely 0.378%, and the difference between the best and worst case in relative MAE is about four times. The water height error follows the same trends as the friction. However, the difference between the best and worst cases is less important for the MAE, around 5 times, but similar for the relative MAE, four times. We can quantitatively conclude that our approach handles different use cases with great accuracy.



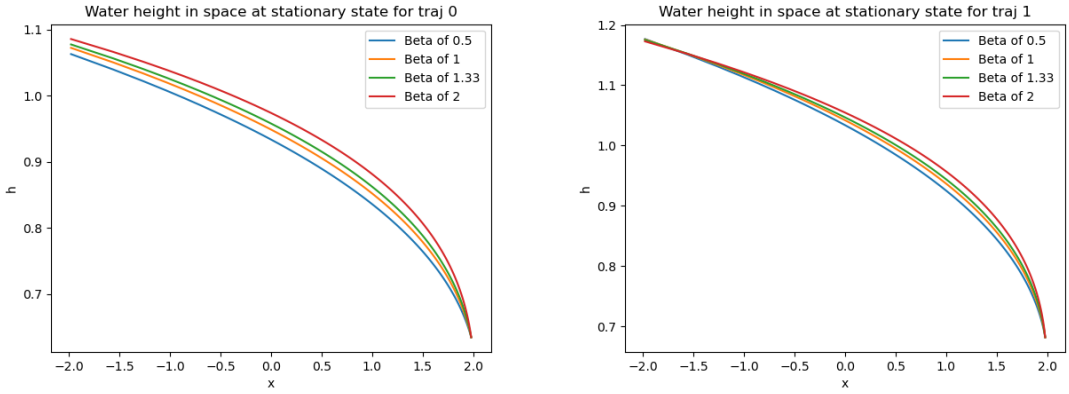
(a) β of 1/2.

(b) β of 1.

(c) β of 4/3.

(d) β of 2.

Figure 4.16: Friction laws guessed by the neural network for different friction law with changing β for $C_f = 0.3$.

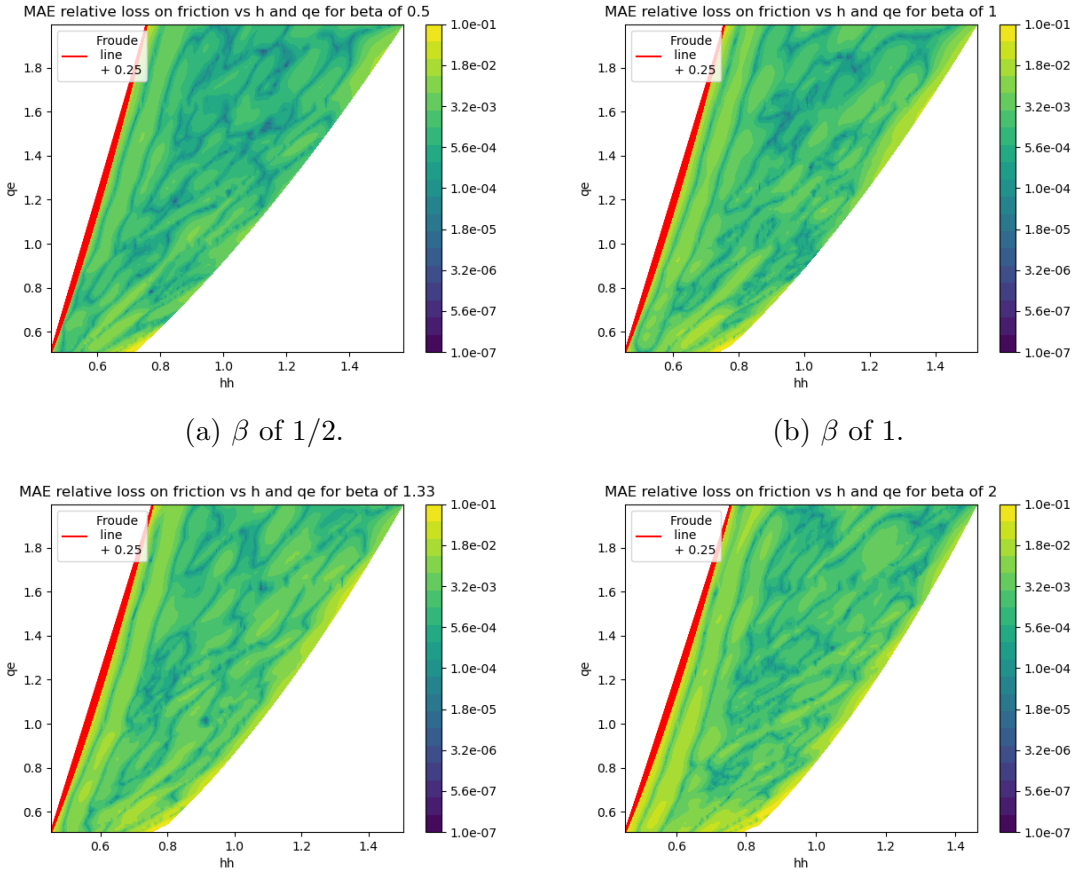


(a) Trajectory 1.

(b) Trajectory 2.

Figure 4.17: Examples of true trajectories with different β for $C_f = 0.3$.

Qualitatively, Figure 4.16 shows different friction laws. The values taken by the water height change between the laws, a higher β reduces the range of values. Moreover, the friction decreases with a higher β for higher water heights. In Figure 4.17, different trajectories with these laws are shown. This helps representing the physical phenomena. As expected, when β increases, the rise of the height is firstly slowed, due to more friction added, before being accelerated for higher values of the height, when friction is reduced for $h \geq 1$. This can be seen in the second trajectory of Figure 4.17 when around $x = -1.75$ the trajectories cross. Moreover, Figure 4.18 confirms the quantitative results, with a higher error when β is higher, as is seen the color code being brighter for higher β. In addition, the models struggles to capture the edges, when values are either low or high, with a bright zone, especially for high values of h and low value of Q_e. The neural network has globally a constant error throughout the domain, and this does not change for different friction laws. Overall, the qualitative analysis also validates the robustness of the proposed approach, as well as as showing that laws with a higher range of values and more variations are harder to capture.



(a) β of 1/2.

(b) β of 1.

(c) β of 4/3.

(d) β of 2.

Figure 4.18: Relative friction absolute error for different friction laws with changing β for $C_f = 0.3$.

4.6.1.3 Learning with different schemes

In order to perform simulations required by our approach, a scheme needs to be chosen. It is then crucial to understand the influence of this scheme on the results. Since synthetic data is used in this work, the scheme with which the data is created can also be changed. So we test a comparison changing the true scheme, i.e. the scheme which created the data, and the network scheme, i.e. the scheme with which the neural network learns. A point of interest of these experiments is to study if the neural network can correct the errors due to the scheme.
The different schemes used in this study are

- Euler. A first-order fixed-step method.
- Midpoint. A second-order fixed-step method.

- Bosh3. A third-order method with adaptive-step. More precisely, the Bogacki–Shampine method.
- RK4. A fourth-order scheme with fixed-step. More precisely, a fourth-order Runge-Kutta with 3/8 rule.
- Dopri5. A fifth-order method with adaptive-step. More precisely, a Dormand–Prince method of fifth-order.

In Table 4.5 and Table 4.6, it is shown that using a first order scheme does not lead to excellent results if the data was created with more complex schemes and vice-versa. Indeed, the errors on the friction using an Euler scheme with another scheme are almost an order of magnitude higher compared to other schemes. The other schemes have similar results between each other. However, the midpoint method, which is a second-order method, is slightly less efficient with the other methods than the other higher order methods when predicting the friction. Another interesting aspect of the results on the friction is that Table 4.5 is symmetric, using a given numerical scheme as the true scheme or the network scheme is equivalent. Moreover, the diagonal presents the lower values, since the settings are similar and the network does not have to compensate for a different scheme when learning the friction law
However, the results on the water height are slightly different. Indeed, using an Euler scheme with another scheme leads to results with performances twice lower, and using a Midpoint scheme is similar to the other schemes. These results, presented in Table 4.6, could have been expected to be all constant since the optimization is performed with the water height as a target. The experiments show that learning that the errors induced by an Euler scheme need to be compensated highly in the friction law learned, hence a high error in friction, and cannot be entirely compensated when predicting the water height.
We can conclude from this part, that using a first-order method makes it harder for the network to learn properly data generated from higher-order schemes. However, the network has learned to adapt to other high-order schemes and update the friction law to correct for the errors induced by using different schemes, hence leading to a less physical law but closest to the data and more applicable in practical cases.

Table 4.5: Friction error for different schemes. Results in MAE (10^{-3}). True solution scheme by column and network scheme by column

Network scheme \ **True scheme**	**Euler**	**Midpoint**	**Bosh3**	**RK4**	**Dopri5**
Euler	0.62	3.89	3.86	3.98	3.84
Midpoint	4.04	0.63	0.80	0.79	0.74
Bosh3	3.99	0.78	0.64	0.61	0.57
RK4	3.84	0.79	0.66	0.61	0.61
Dopri5	3.73	0.80	0.67	0.61	0.62

Table 4.6: Water height error for different schemes. Results in MAE (10^{-4}). True solution scheme by column and network scheme by column

Network scheme \ True scheme	Euler	Midpoint	Bosh3	RK4	Dopri5
Euler	0.31	0.56	0.83	0.80	0.80
Midpoint	0.95	0.27	0.42	0.41	0.35
Bosh3	0.50	0.32	0.32	0.30	0.43
Rk4	0.68	0.34	0.40	0.33	0.28
Dopri5	0.51	0.40	0.30	0.26	0.30

4.6.1.4 Is is better to learn with an initial guess of the friction?

As detailed in Section 4.2, there is no consensus on the friction law. However, common laws still emerge and good priors can be known on some cases. In such a context, it is natural to try to incorporate these priors in our approach. In order to do so, the neural network is added to a guessed friction law. The sum of the two then aims to reproduce the true friction law. This section is then closer to the APHINITY framework(Yin et al., 2021), the goal being to complete a physical model with strong priors with a data-driven method. In order to test the influence of the prior, the general form of the friction presented in Equation (4.2) is chosen and C_f and β are varied for the initial friction guess. The network learns a correction between the guessed law and the true one. The true friction law is computed with $C_f = 0.2$, $\beta = \frac{4}{3}$ and $\alpha = 1$, as stated in Section 4.3.
The different MAE and relative MAE results for different initial guesses for the friction and the water height are presented in Table 4.7. In order to better interpret if a prior is good, Table 4.8 details the L^1, L^2 and L^∞ norms between the true law and the initial guesses. The results are not always as expected. First, the case with no prior is sometimes better than guesses with better priors, as is illustrated for the guess with $C_f = 0.3$ and $\beta = 4/3$, where the guess is twice better in L^1 norm than no guess, but the MAE on the friction and height is around three times higher. Second, one may expect that the final error is highly correlated to the distance between laws. Such a result is observed in some cases, e.g., for $C_f = 0.1$. When β increases, the laws get closer to the true one, and so are the combination of the neural network and the prior. However, this correlation is not always true, as is the case for $C_f = 0.3$. The priors are worse when β increases, but the overall guess is not always worse. This can be seen for the guess with $C_f = 0.2$ as well, where the guess with $\beta = 2/3$ is better than with $\beta = 1$, but the errors on the friction and height are slightly worse In addition to these findings, guessing with the right laws leads to a very low reconstruction error, as expected. A null error could have been expected, however, as explained in Section 4.4.3, the network is trained in a stochastic manner and through a numerical scheme, thus numerical residuals have

an influence on the results.

Table 4.7: Height and friction errors for different guessed friction laws. We have chosen different β and C_f for the guess. Friction MAE and relative MAE are expressed with factor $\times 10^{-3}$ and height with factor $\times 10^{-4}$. The true friction coefficients are 0.2 for C_f and 4/3 for β, they are underlined in the table.

β	MAE		Relative MAE		MAE		Relative MAE	
	Friction	**Height**	**Friction**	**Height**	**Friction**	**Height**	**Friction**	**Height**
	$C_f = 0.1$				$\underline{C_f = 0.2}$			
1/2	0.47	0.19	1.11	0.27	0.57	0.27	1.35	0.37
2/3	0.41	0.16	0.95	0.21	0.43	0.20	1.16	0.25
1	0.38	0.21	0.93	0.27	0.46	0.24	1.16	0.31
$\underline{4/3}$	0.29	0.12	0.70	0.16	0.01	0.01	0.03	0.02
	$C_f = 0.3$				$C_f = 0$			
1/2	0.60	0.35	1.60	0.40	0.63	0.32	1.40	0.42
2/3	0.44	0.27	1.14	0.33				
1	0.77	0.60	2.22	0.73				
4/3	1.51	0.97	4.9	1.23				

Table 4.8: Difference in norm between different friction laws. L^2 corresponds to the L^2 norm difference between the initial guess and the true law, and L^∞ corresponds to the infinity norm difference between the two. It is expressed with a factor $\times 10^{-2}$ for the L^2 norm and with $\times 10^{-1}$ for the L^∞ and L^1 norms. The true reference friction law parameters are highlight by being underlined.

β	L^1	L^2	L^∞	L^1	L^2	L^∞
	$C_f = 0.1$			$\underline{C_f = 0.2}$		
1/2	1.66	3.43	5.12	0.50	0.51	3.11
2/3	1.63	3.27	4.89	0.41	0.35	2.64
1	1.58	2.93	4.37	0.21	0.10	1.49
$\underline{4/3}$	1.51	2.57	3.85	0	0	0
	$C_f = 0.3$			$C_f = 0$		
1/2	1.11	1.52	1.99	3.02	10.3	7.69
2/3	1.15	1.53	2.02			
1	1.31	1.80	2.81			
4/3	1.51	2.57	3.85			

4.6.2 Convergence

Schemes solve problems on a specific grid. The size of the grid is of great importance, since it determines the extent of high-frequency information accessible to the network and the number of points encountered during training. In this section, we study the impact of the grid size. In order to do so, we explore three different scenarios. The first scenario involves varying the number of observed trajectories for a

given discretization. The second scenario consists in altering the spatial discretization while ensuring that the network receives the same points for each discretization, eliminating the influence of data point quantity. The third scenario combines the previous settings and makes varying only the discretization, in the sense that both discretization and number of points considered in the training change. These cases allow us to fully understand the different choices and influences.
The Figure 4.20b shows the convergence of the RK4 scheme for the ODE. It consists in plotting the difference in norm L^1 between the values discretized on a given grid and a grid twice as big. This helps us deciding which discretization to choose to have a solution that should be almost independent of the choice of the scheme. Indeed, when increasing the discretization grid, numerical diffusion should be reduced. Hence, we choose a grid of 2^{16} to test our network against. The solution is sub-sampled to match the different discretization sizes. This procedure should avoid any bias of the choice of scheme to create the synthetic high-fidelity solution. Note that the curve presented in Figure 4.20b follows a line of slope -4, which is expected since RK4 is a scheme of order 4 and thus is supposed to convergence asymptotically as h^4 until the computer limit. The convergence of the Euler and Midpoint schemes is also presented in the Appendix in Figure B.2 and both schemes follow a convergence of their respective order 1 and 2.
In Table 4.9, it is shown that increasing the number of trajectories considered during the training helps the network better learn. However from 16 trajectories, the improvement is marginal. This suggests that the network does not need a lot of trajectories to have good performances. It is also important to note that with 16 trajectories of 128 points, the network still observes a lot of points.
The errors when tested on different schemes and with different discretization sizes are presented in Table 4.10 and Table 4.11. In Table 4.10, the influence of the number of points seen during training is removed. However, as observed in Table 4.9, this influence is quickly marginal, hence both tables end up having similar results. The main finding, as illustrated in Figure 4.20a, is that the error is reduced by an order of 2 when the grid size is twice larger, thus showing a convergence of order 1 no matter what the scheme is. By increasing the discretization size, the scheme produces results closer to the true solution, and thus the network has to adapt less the friction law to correct the diffusion. This is illustrated in Figure 4.19, where diffusion can be observed, and is reduced as the discretization size increases. Additionally, Euler scheme is slightly less efficient than the other ones when the discretization increases, especially for predicting the water height. For instance, for a discretization size of 512, the error on the water height is more than twice the error of the other schemes, but the friction error is just slightly higher than for the other schemes. This can explained by the order of Euler scheme, which need larger grids to obtain similar performances than higher order schemes. Midpoint and RK4 schemes have very similar results.

In addition to the neural network approach, we tested the regular numerical analysis approach for the third scenario. C_f, β and α are tuned according to the data using a gradient descent type algorithm. The training curves for a grid of size 512 and RK4 scheme can be found in Figure B.3. The results are shown in Table 4.12. They show the same patterns as Table 4.11, where the error is reduced by an order of 2 when the grid is size is increased by 2 and Euler scheme performs less efficiently than the other two. This is not expected since the convergence rate could have followed the order of the schemes. One possible explanation is that the tolerance is not enough to be in an asymptotic regime and hence the convergence can be of a different order of the scheme. Moreover, our neural network approach outperforms the traditional numerical analysis, especially for small grids. When the discretization size increases, diffusion is reduced and the predicting a law with the same analytical form as the underlying true law becomes better than not putting any prior on the learned parameters. However, even for a grid of size 512, the neural network obtains better results than the traditional approach. This was not possible due to the computational cost, but by increasing even more the grid size, the traditional approach would probably finish by outperforming our DL approach. We can hence conclude from Table 4.12 that, since only three parameters can be adjusted, the fixed form traditional approach struggles to correct the various sources of errors that comes from the use of a different schemes, the diffusion induced by the discretization and the number of training samples.
This section highlights the convergence of our method on a realistic case with a high-fidelity solution, but also its weaknesses when there are not enough data points or the discretization size is too small. It manages to adapt the friction law to correct for diffusion, but not to a very efficient extent, even if it still better than the traditional numerical analysis approach.

Table 4.9: Height and friction MAE for different numbers of trajectories seen during training of three schemes. The number of points seen during training changes for each case and the discretization is set to 128 points. Friction MAE is expressed with factor $\times 10^{-3}$ and height with factor $\times 10^{-4}$.

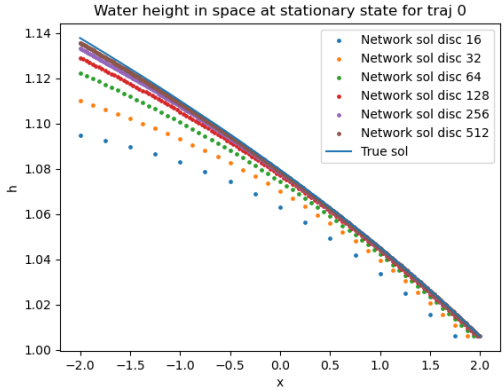
Scheme \Traj	**2**		**4**		**8**	
	Friction	**Height**	**Friction**	**Height**	**Friction**	**Height**
Euler	105.18	172.50	42.06	54.74	25.68	22.78
Midpoint	106.12	175.42	42.16	58.11	25.23	23.48
RK4	106.04	175.20	42.24	58.34	25.91	24.41
	16		**32**		**64**	
Euler	13.58	8.63	13.17	6.64	12.57	5.53
Midpoint	13.03	9.20	10.40	6.07	10.16	5.27
RK4	13.11	9.24	10.47	5.89	10.19	5.19
	128		**256**		**512**	
Euler	12.55	5.20	12.21	4.40	11.65	1.68
Midpoint	9.67	4.64	9.59	3.91	9.37	2.23
RK4	9.78	4.61	9.60	3.77	9.41	2.10

Table 4.10: Height and friction MAE for different discretization sizes of three schemes. The number of points seen during training is the same for each case and the discretization change. Friction MAE is expressed with factor $\times 10^{-3}$ and height with factor $\times 10^{-4}$. 500 trajectories are used for training. The test can be done on the grid size seen during training, which is 16, or the whole grid, which would thus test the extrapolation capabilities of the network. The height 16 column corresponds to testing with 16 points and the others correspond to testing on the whole grid.

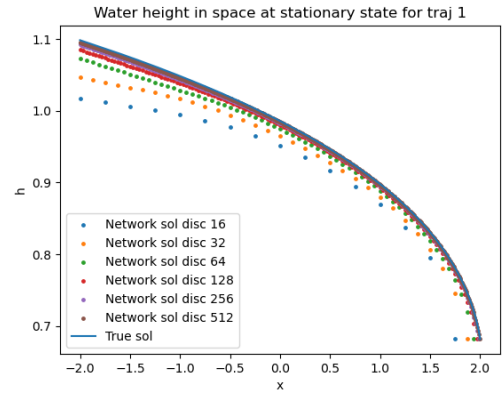
Scheme \Disc	**16**			**32**		
	Friction	**Height**	**Height 16**	**Friction**	**Height**	**Height 16**
Euler	82.63	37.32	37.32	41.07	19.76	21.09
Midpoint	58.24	7.83	7.83	30.10	7.08	7.42
RK4	60.61	1.07	1.07	30.65	3.92	3.98
	64			**128**		
Euler	20.43	10.86	11.22	9.95	5.76	5.62
Midpoint	14.76	3.79	3.91	7.95	3.23	3.23
RK4	15.15	3.37	3.35	7.44	2.55	2.54
	256			**512**		
Euler	4.98	2.93	2.55	2.64	1.63	1.55
Midpoint	3.78	1.26	1.21	2.20	0.73	0.77
RK4	3.79	1.22	1.16	2.16	0.72	0.77

Table 4.11: Height and friction MAE for different discretization of three schemes. The number of points seen during training and the discretization change. Friction MAE is expressed with factor $\times 10^{-3}$ and height with factor $\times 10^{-4}$. 500 trajectories are used for training.

Scheme \Disc	**16**		**32**		**64**	
	Friction	**Height**	**Friction**	**Height**	**Friction**	**Height**
Euler	82.63	37.32	43.93	19.21	23.01	10.56
Midpoint	58.24	7.83	32.45	3.58	17.49	2.88
RK4	60.61	1.07	33.33	2.24	18.16	3.25
	128		**256**		**512**	
Euler	11.57	5.56	5.92	3.10	3.07	1.68
Midpoint	9.01	1.81	4.60	1.52	2.53	0.61
RK4	8.92	1.49	4.78	0.97	2.50	0.63

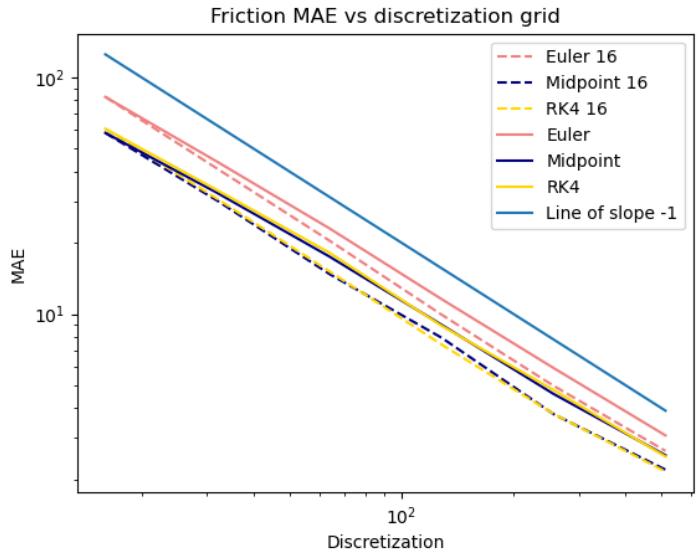


(a) Trajectory 1.

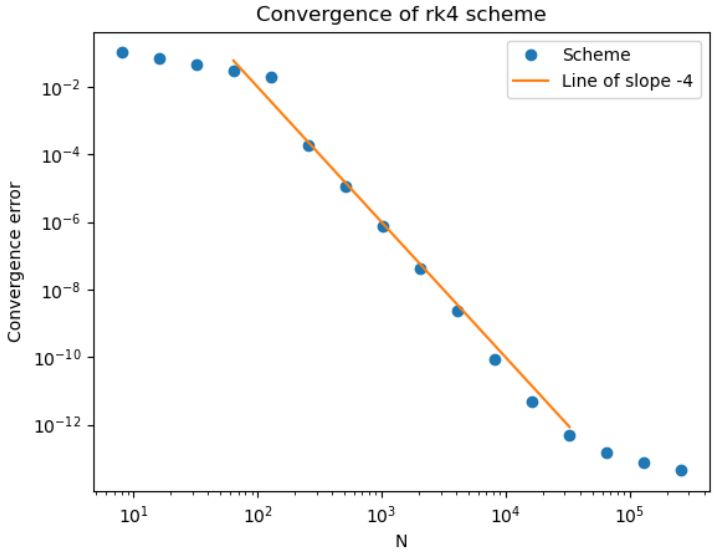


(b) Trajectory 2.

Figure 4.19: Examples of trajectories with different discretization sizes with network trained with these sizes.



(a) Friction MAE against the discretization grid sizes for the last two scenarios, i.e. learning with 16 points per trajectory and learning with all the points in the trajectory.



(b) Convergence of the RK4 scheme (without any neural network) for the river case.

Figure 4.20: Study of the impact of discretization on the training of the neural network and study of the convergence of the RK4 scheme.

Table 4.12: Height and friction MAE for different discretization of three schemes with 3 parameters estimation. The parameter estimation is done using Adam, no neural network has been used for this table. The number of points seen during training and the discretization change. Friction MAE is expressed with factor $\times 10^{-3}$ and height with factor $\times 10^{-4}$. 500 trajectories are used for training.

Scheme \Disc	**16**		**32**		**64**	
	Friction	**Height**	**Friction**	**Height**	**Friction**	**Height**
Euler	97.03	68.77	50.27	32.98	25.61	15.87
Midpoint	71.61	49.33	38.72	25.51	20.51	13.24
RK4	76.71	94.48	39.53	25.42	20.70	12.89
	128		**256**		**512**	
Euler	12.99	7.86	6.39	3.80	3.21	1.90
Midpoint	10.29	6.51	5.25	3.32	2.55	1.59
RK4	10.36	6.45	5.25	3.29	2.55	1.58

4.7 PDE experiments

The previous sections showed experimental results for the ODE stationary case. In this section, we investigate the more complex time-dependent PDE case, as detailed in Section 4.3. This is a more challenging study, since there are both a spatial and a time component to deal with. The training is then harder than the stationary case, and is explained in Section 4.4.3 and illustrated in Figure 4.7. This study focuses on three settings, a small variations, a medium variations and a large variations setting. They highlight different cases of the friction law. This section details the experimental results for each setting.

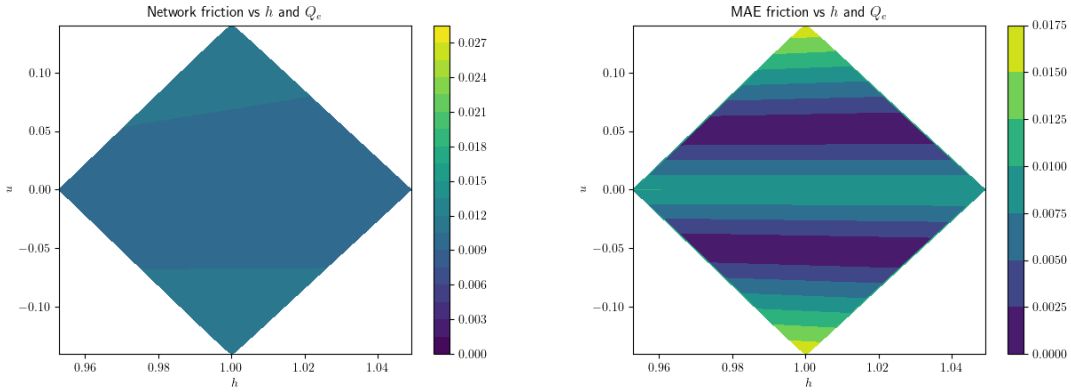
4.7.1 Small variations setting

As can be seen in Figure 4.21, the friction law learned has a small MAE error over domain, but is not very convincing. Since the variations of the law are small and the influence of the height almost negligible, the neural network gives an almost constant law. However, as can be seen in Figure 4.22 and Figure 4.23, the combination of the network and the scheme manages to reconstruct accurately the trajectories for the water height and flow. The quantitative results presented in Table 4.13 further confirms the good results, with an overall small error for the friction and for the water height and flow. An additional explanation for the friction loss is that the distribution of the values taken during training, presented in Figure 4.3b, is highly concentrated in the center of the figure, hence the network manages to accurately predict the friction in this domain. It still struggles to have the right value in the center precisely, where it should be null (for a null water flow). This can be explained by two factors:

(i) When the friction is null, it has no influence on the water height and flow, hence it is not important to predict it accurately to have the same error tolerance on h and u as for other values of the friction.

(ii) The domain of the friction where the friction is null is an extrema of the friction function, which is always harder to predict.

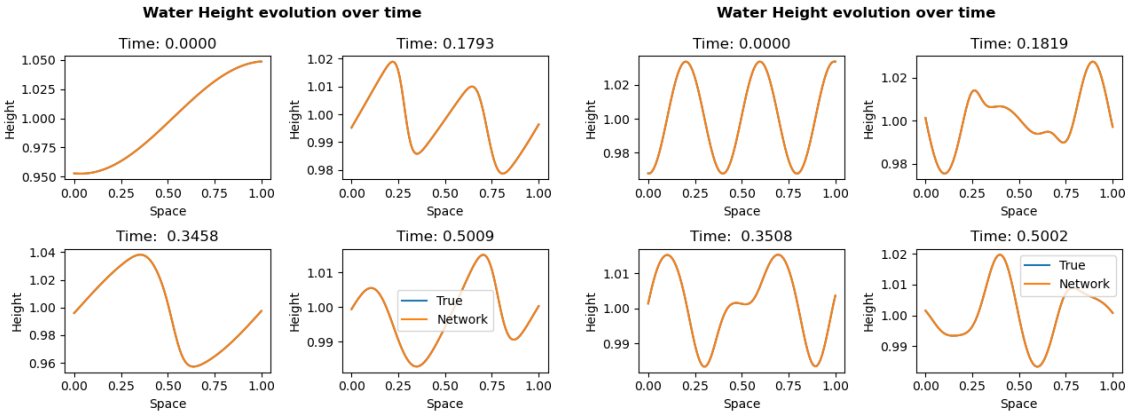
A further study of the loss with the time is shown in Figure 4.24. This illustrates several phenomena. The first one is that the error over time of the water height and flow increases, which is expected since the scheme accumulates error over time. The second one is that the friction error is high at the beginning of the trajectory. This is also expected since the values taken at the beginning are the values least seen during the training. The last one is the big peak around 0.3s. This peak can be explained by the fact that the water flow changes direction around 0.3s, which leads to a null friction, hence worst results on the friction and the water height and better results on the flow (which is null). This is illustrated as well in Figure 4.22 and Figure 4.23, where the water flow changes convexity and sign from 0.2s to 0.5s, with the values around 0.3s getting closer to 0. Similarly, the water height peaks change, the highest peak becomes the lower one and vice-versa.



(a) Neural network friction.

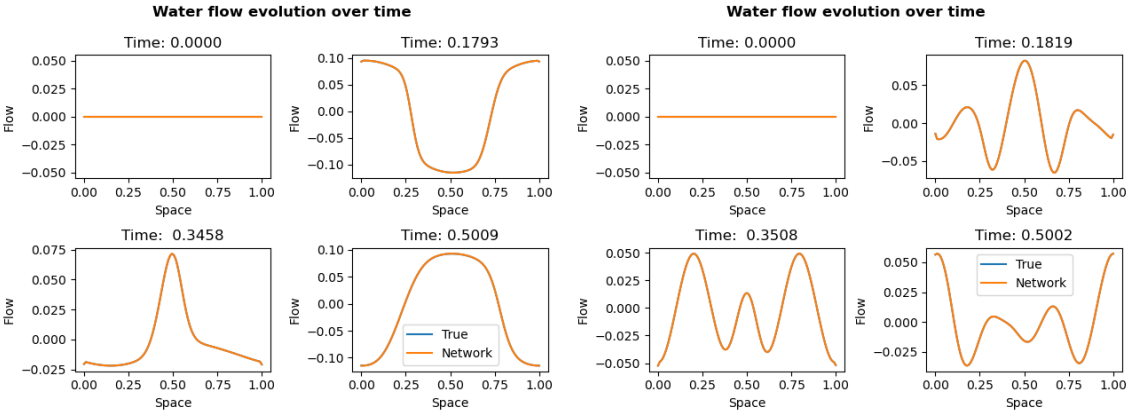
(b) MAE on the friction.

Figure 4.21: Friction law and error of the neural network for the small variations PDE setting.



(a) Trajectory 1 (b) Trajectory 2

Figure 4.22: Snapshots of the water height with time for the small variations setting.



(a) Trajectory 1 (b) Trajectory 2

Figure 4.23: Snapshots of the water flow with time for the small variations setting.

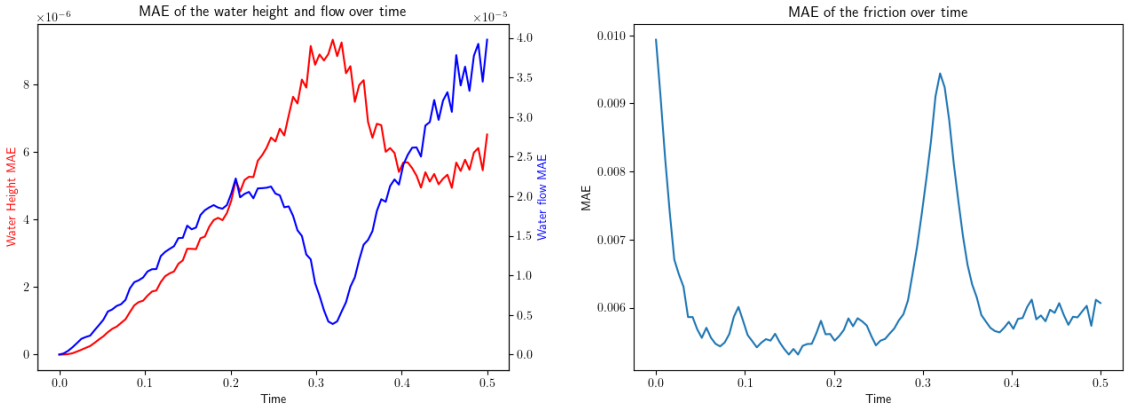
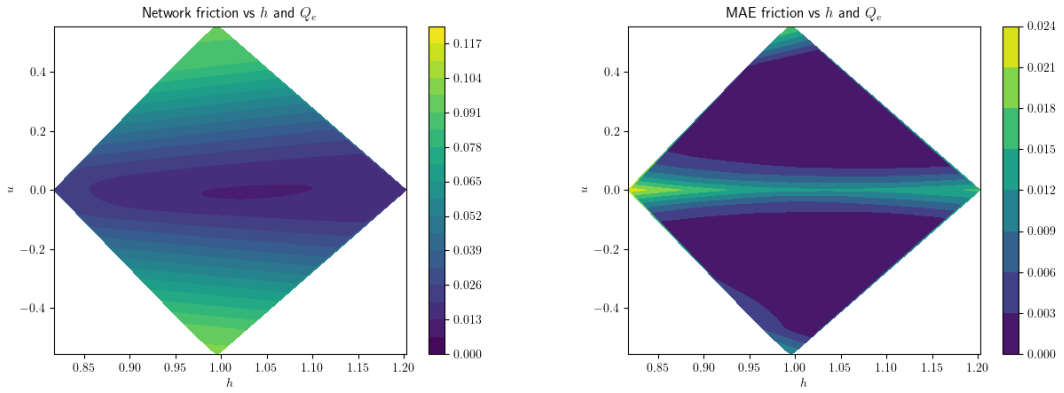


Figure 4.24: Water height, flow and friction MAE over time averaged for 25 trajectories.

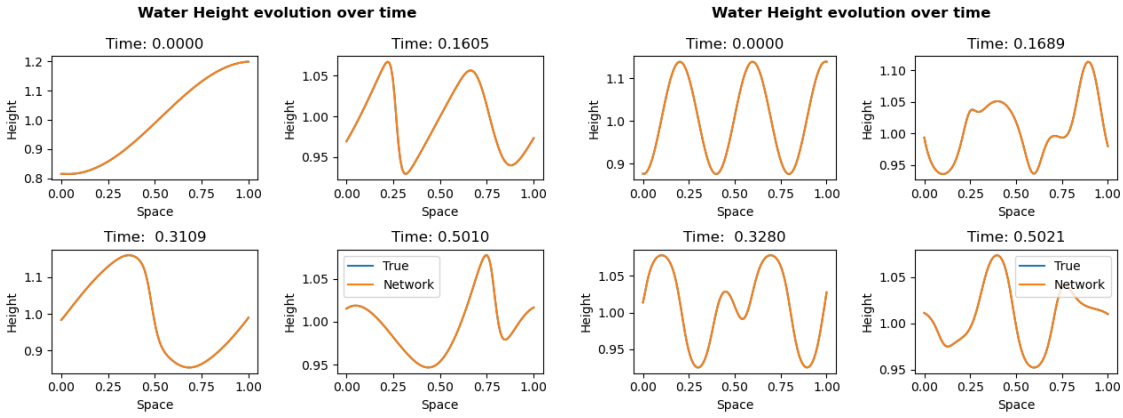
4.7.2 Medium variations setting

The results of the medium variation setting are slightly different than the ones of the small variations setting. Indeed, since the variations are bigger, the friction law has a higher range of values and the water height a bigger influence, where previsouly it was almost null. As explained in Section 4.3, this makes the medium variation case a more practical one. The network then gives an interesting friction law, reflecting this higher range. This can be seen in Figure 4.25, which also presents the error plot over the domain. It illustrates the same phenomena as the one considered in Section 4.7.1. For values around a null water flow, where the friction should be null, the network struggles to give a null value of the friction. However, as shown qualitatively in Figure 4.26 and Figure 4.27 and further confirmed quantitatively in Table 4.13, the network and the scheme manage to reconstruct accurately the different trajectories. Figure 4.28 illustrates the same phenomena as Figure 4.24. The only difference is that the peak is smaller, due to the higher friction.



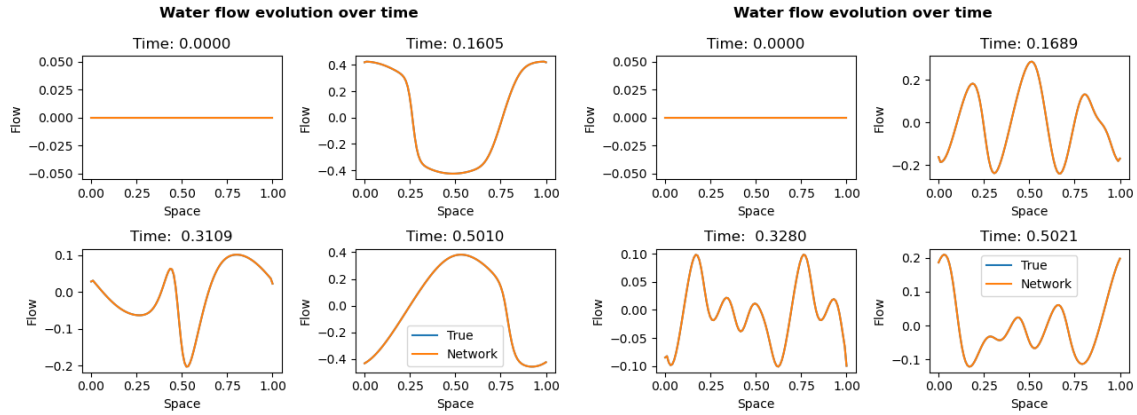
(a) Neural network friction. (b) MAE on the friction.

Figure 4.25: Friction law and error of the neural network for the medium variations PDE setting.



(a) Trajectory 1 (b) Trajectory 2

Figure 4.26: Snapshots of the water height with time for the medium variations setting.



(a) Trajectory 1 (b) Trajectory 2

Figure 4.27: Snapshots of the water flow with time for the medium variations setting.

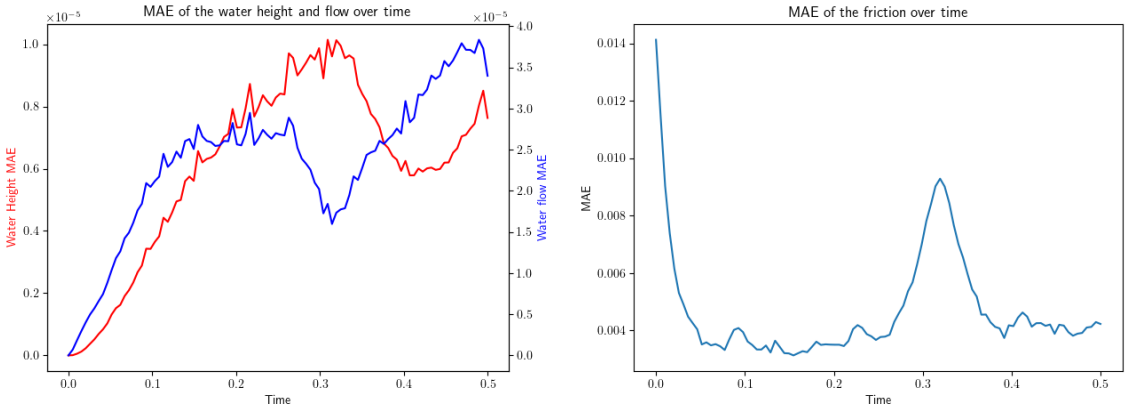
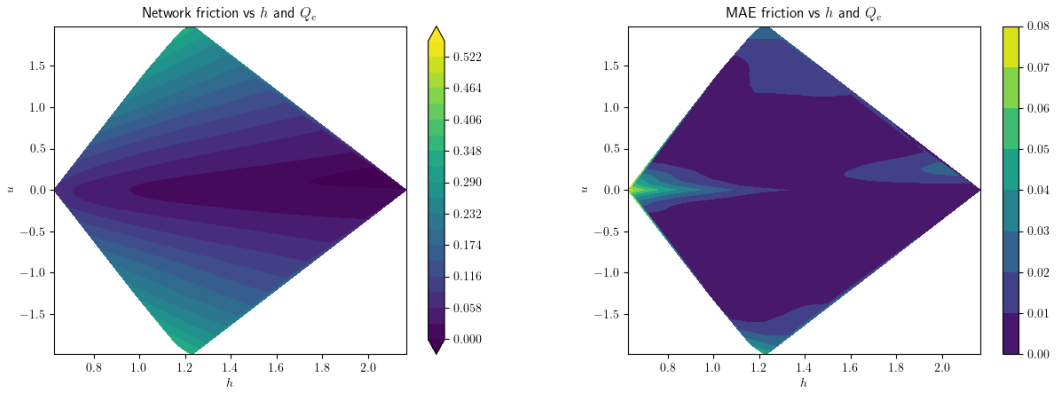


Figure 4.28: Water height, flow and friction MAE over time averaged for 25 trajectories.

4.7.3 Large variations setting

In this setting, we test the limit of our method by considering a very challenging case. In Table 4.13, it is shown that the network managed to learn similarly well the friction law, but since the variations are higher, a similar error on the friction with the other settings leads to a higher error in height and flow prediction. When data is sparse and the values should be null, the network struggles, which is illustrated in Figure 4.29, where the line $u = 0$ should lead to a null value of the friction. Instead of this, values for $h > 1$ are almost null, because they add up to most of the values seen, but values for $h < 1$, and especially low values of h lead to the highest error on the plot. This emphasis the need to be cautious for the friction law given by the network around the initial conditions.



(a) Neural network friction.

(b) MAE on the friction.

Figure 4.29: Friction law and error of the neural network for the large variations PDE setting.

4.7.4 Discussion

As seen in the previous sections and in Table 4.13, the neural network provides friction laws with small errors, but which are not always qualitatively satisfying. However, the results on the water flow and height are both qualitatively and quantitatively interesting. Note that only 5 trajectories, with the same initial conditions for each setting, are used to produce this table, hence the variance on the results can be large. Interestingly, the error on the friction for the small variations setting is higher than for the other higher variations setting. This can be explained by the fact that to have similar or better results on the water height and flow in this setting, the error on the friction does not need to be smaller. For medium and large variations settings, the relative error on the friction is similar, but the error on the water height and flow is one order of magnitude higher. This is due to small errors on the friction leading to high errors on the water height and flow.
To conclude on this section, the results show that our approach can be used for hard PDE time-dependent problems, and manages to give a friction law that accurately reconstructs various trajectories, which is the main goal of this project. Further studies are presented in B.2 to give a more detailed view on the results.

Table 4.13: Results of our approach against learning directly for different settings at test time. Since the friction can be null, relative error is computed as $\|K_{NN} - K_f\|^2/\|K_f\|^2$.

Setting	**Relative friction MAE**	**Friction MAE** ($\times 10^{-2}$)	**Height MAE** ($\times 10^{-5}$)	**Flow MAE** ($\times 10^{-4}$)
Small variations	0.88	0.57	0.48	0.16
Medium variations	0.16	0.40	0.56	0.18
Large variations	0.07	0.43	4.28	1.38

Chapter 5

INR architectures for dynamical systems

In this chapter, we study how to apply implicit neural representation (INR) methods to dynamical systems. These novel algorithms presented in Section 2.3 are a natural fit for these systems given their continuous nature.
First, we design INFINITY, a INR based method that can be applied to static PDE problems. It is tested on the challenging RANS equations for surrogate modeling using the realistic AirfRANS dataset (Bonnet et al., 2022), which models airfoils. We show that INFINITY can accurately infer physical fields throughout the volume and surface, leading to a correct prediction of the drag and lift coefficients, which are crucial for airfoil design. This work led to a publication at the ICML 2023 workshop on the synergy of scientific and machine learning modeling.

Serrano, L., **Migus, L.**, Yin, Y., Mazari, J. A., and Gallinari, P. (2023). INFINITY: Neural Field Modeling for Reynolds-Averaged Navier-Stokes Equations. In ICML 2023 workshop on the synergy of scientific and machine learning modeling.

Next, we design TimeFlow, a general framework using INRs to impute and forecast time series. By its continuous nature, TimeFlow can handle missing data, irregular sampling and unaligned observations from multiple sensors while having similar performances to state-of-the-art algorithms and being able to generalize to unseen samples and time windows. The experiments have been conducted on three extensive time series datasets, namely the Electricity, Solar and Traffic datasets. This work led to a submission at ICLR 2024, where it is currently under review.

Naour, E. L., Serrano, L., **Migus, L.**, Yin, Y., Agoua, G., Baskiotis, N., Gallinari, P., and Guigue, V. (2023). Time Series Continuous Modeling for Imputation and Forecasting with Implicit Neural Representations. Under review at ICLR 2024.

5.1 Neural Field Modeling for Reynolds-Averaged Navier-Stokes Equations

For numerical design, the development of efficient and accurate surrogate models is paramount. They allow us to approximate complex physical phenomena, thereby reducing the computational burden of direct numerical simulations. We propose INFINITY, a deep learning model that utilizes implicit neural representations (INRs) to address this challenge. Our framework encodes geometric information and physical fields into compact representations and learns a mapping between them to infer the physical fields. We use an airfoil design optimization problem as an example task and we evaluate our approach on the challenging AirfRANS dataset, which closely resembles real-world industrial use-cases. The experimental results demonstrate that our framework achieves state-of-the-art performance by accurately inferring physical fields throughout the volume and surface. Additionally we demonstrate its applicability in contexts such as design exploration and shape optimization: our model can correctly predict drag and lift coefficients while adhering to the equations.

5.1.1 Introduction and motivation

Numerical simulations are essential for analyzing systems governed by partial differential equations (PDEs) in fields like fluid dynamics and climate science. These

simulations involve discretizing the domain and solving the equations using methods such as finite differences, finite elements, or finite volumes (Reddy, 2019; Grossmann et al., 2007; Eymard et al., 2000). Since direct numerical simulation (DNS) can be computationally expensive or intractable, it is crucial to develop computationally efficient yet accurate surrogate models to accelerate the design process. Surrogate modeling for industrial applications, however, poses several challenges. The meshes used in these applications are extensive, consisting of hundreds of thousands of cells, and they also exhibit unstructured data and involve multi-scale phenomena. A typical example is the design of airfoils which will be our application focus, although the ideas can be easily implemented for other design tasks. In this domain, a new costly simulation must be run for each mesh during the optimization process, leading to time-consuming processes. Additionally, the design process focuses on finding the optimal shape for an airfoil that minimizes the force required for flight. Experts typically maximize the lift-over-drag ratio by solving equations across the entire mesh, with particular emphasis on the surface where various multi-scale phenomena occur.

Recently, deep learning methods have emerged as promising approaches for constructing surrogate models. However, the progress in this field was initially hindered by the lack of evaluation datasets representative of real-world data. The machine learning community has begun to address this issue by developing benchmarks. In this work, we utilize the a recent AirfRANS dataset Bonnet et al. (2022), which aims to replicate real-world industrial scenarios. This comprehensive benchmark provides an evaluation framework to assess the capabilities of deep learning (DL) in modeling the two-dimensional incompressible steady-state Reynolds-Averaged Navier-Stokes (RANS) equations for airfoils. Additionally, this 2D dataset encompasses a wide range of airfoil shapes derived from NASA's early works (Cummings et al., 2015), various turbulence effects characterized by Reynolds numbers and different angles of attack.

The Navier-Stokes equations are widely used in fluid dynamics, and as a result, numerous neural network surrogates have been proposed for their modeling in different contexts. Initial attempts all relied on grid-based approaches such as convolutional Neural Networks (CNNs) (Um et al., 2020; Thuerey et al., 2020; Mohan et al., 2020; Wandel et al., 2020; Obiols-Sales et al., 2020; Gupta et al., 2021; Wang et al., 2020). CNNs face challenges when dealing with the irregular meshes used in computational fluid dynamics (CFD). Graph Neural Networks (GNNs) have shown promise Pfaff et al. (2021) but they have limitations in terms of receptive field size and information propagation across distant nodes, especially for large meshes. Additionally, GNNs struggle when the mesh is too dense and cannot fit into the memory of GPUs, necessitating sub-sampling. This limitation restricts their application in contexts where large meshes with multi-scale phenomena are prevalent. Furthermore, the evaluation of the models has primarily focused on traditional machine

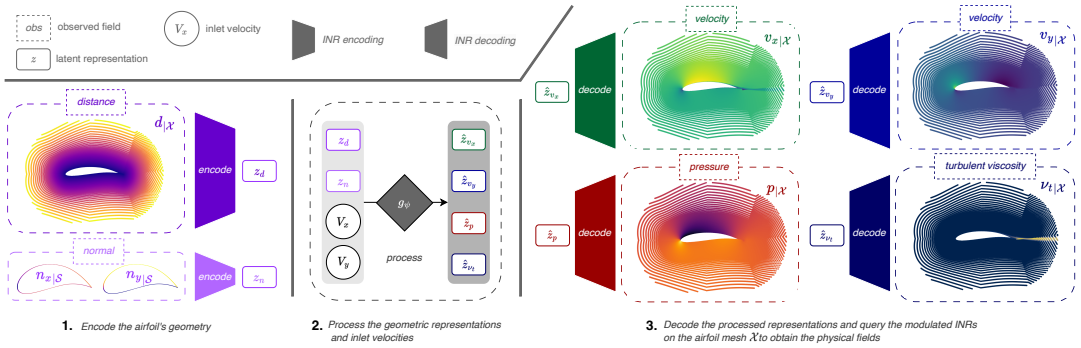


Figure 5.1: The inference of INFINITY proceeds in three steps. 1. We encode the distance function d and the normal components n_x, n_y into the latent representations z_d and z_n. 2. We process these codes along with the inlet velocities V_x, V_y to obtain the predicted output codes $\hat{z}_{v_x}, \hat{z}_{v_y}, \hat{z}_p, \hat{z}_{\nu_t}$ corresponding respectively to velocity, pressure and viscosity. 3. The processed codes are decoded with the modulated INRs, which can be queried directly at any mesh position $\boldsymbol{x} \in \mathcal{X}$.

learning scores, such as global error over the entire domain (a.k.a. *volume*), rather than more design-oriented scores, including local error in the surface area surrounding the airfoil (a.k.a. *surface*) and errors in the aerodynamic forces of interest, such as drag and lift.

Leveraging recent advances in implicit neural representations (INRs) (Sitzmann et al., 2020a; Mildenhall et al., 2021), which have shown successful applications in physics problems (Yin et al., 2023), we introduce INFINITY, a model that utilizes coordinate-based networks to encode geometric information and physical fields into concise representations. INFINITY establishes a mapping between variables representing the problem's geometry and the corresponding physical fields, within this representation space. It possesses several unique features: (i) it is robust to varying mesh sampling, allowing for adaptability to different geometries, (ii) it effectively captures multi-scale phenomena, resulting in state-of-the-art scores for both volume and surface evaluations, (iii) as a continuous surrogate model, it can be used to accelerate the evaluation of different meshes during the design process, leading to significant speed-up. Importantly, we verified that INFINITY's field predictions accurately produce the correct lift and drag forces clearly outperforming all the baselines.

5.1.2 Method

5.1.2.1 Problem setting

We aim at proposing a surrogate model for airfoil design optimization in scenarios where the amount of available training data is limited ($n_{tr} \leq 1000$). Each airfoil is associated with a domain Ω_i, which is linked to a specific geometry. Consequently,

different meshes $\mathcal{X}_i$ are generated within each domain. The characterization of an airfoil involves defining boundary conditions on $\partial\Omega_i$ corresponding to the airfoil surface, which are discretized into a surface mesh $\mathcal{S}_i$.

The geometric inputs for our model include the following information:

- *Node positions* $\boldsymbol{x}$ represent the coordinates of each node within the airfoil's domain.
- *Distance function* $d(\boldsymbol{x})$ provides the distance from each node to the surface of the airfoil.
- *Normal vectors of the mesh nodes on the airfoil surface* $\boldsymbol{n}(\boldsymbol{x}) = (n_x(\boldsymbol{x}), n_y(\boldsymbol{x}))$ specify the direction perpendicular to the airfoil surface at each node.

In addition to the geometric inputs, we also have access to the inlet velocity values V_x and V_y, denoting the horizontal and vertical components of the velocity, respectively. It is worth noting that, on average, a mesh consists of approximately 200,000 nodes, providing a detailed representation of the airfoil's geometry.

The primary objective of the design optimization process is to maximize the lift-over-drag coefficient ratio, which serves as the key performance metric. To achieve this, we place significant emphasis on evaluating the relative errors in both the drag and lift coefficients, as well as assessing the Spearman correlation between predicted and actual values.

Rather than directly predicting the drag and lift values, our approach focuses on inferring various fluid fields associated with the airfoil's geometry. This includes calculating the velocities (v_x, v_y), pressure p, and turbulent kinematic viscosity ν_t on the mesh nodes, following the experimental protocol proposed in Bonnet et al. (2022). Therefore the inputs of our surrogate model are $(V_x, V_y, d|_{\mathcal{X}_i}, n_x|_{\mathcal{S}_i}, n_y|_{\mathcal{S}_i})_{i=1}^{n_{tr}}$, and the outputs are $(v_x|_{\mathcal{X}_i}, v_y|_{\mathcal{X}_i}, p|_{\mathcal{X}_i}, \nu_t|_{\mathcal{X}_i})_{i=1}^{n_{tr}}$. The output physical fields provide valuable insights into the underlying behavior of the fluid and its interaction with the airfoil's geometry. The drag and lift coefficients are calculated based on the predictions of the trained model while respecting the form of the RANS equations. This approach enables us to obtain a comprehensive understanding of the underlying fluid behavior and its relationship with the airfoil's geometry, thereby ultimately enhancing the accuracy of drag and lift estimation.

5.1.2.2 Model

We present INFINITY: Implicit Neural Fields for INterpretIng geomeTry and inferring phYsics.

Modulated INR In our model, we will treat each geometric input (d or n) or physical output function (v, p, or ν) separately and each will be modeled by an INR.

Let us then consider a generic function u, which will represent either an input geometric field or an output physical field defined over a domain Ω or at its boundary $\partial\Omega$. Let us denote u_i the function corresponding to a specific airfoil example. u_i will be represented by an INR $f_{\theta_u,\phi_{u_i}}$ with two sets of parameters: parameters θ_u shared by all the u_i, and modulation parameters ϕ_{u_i} specific to each individual function u_i. In our airfoil example, ϕ_{u_i} enables the INR to handle different geometries. Overall, this decomposition allows the modulated INR to capture both shared characteristics among the example's functions u_i and the unique properties of each one. INFINITY leverages latent representations inferred from the modulation spaces of the INRs. These latent representations, denoted as z_{u_i}, are compact codes that encode information from the INRs' parameters. They serve as inputs to a hypernetwork h_u, with weights w_u, which computes the modulation parameters $\phi_{u_i} = h_u(z_{u_i})$.

In this work we use Fourier Features (Tancik et al., 2020b) as an INR backbone and apply shift modulation (Perez et al., 2018): $f_{\theta,\phi_{u_i}}(\boldsymbol{x}) = \boldsymbol{W}_L\big(\chi_{L-1} \circ \chi_{L-2} \circ \cdots \circ \chi_0(\boldsymbol{x})\big) + \boldsymbol{b}_L$, with $\chi_j(\eta_j) = \sigma\big(\boldsymbol{W}_j\eta_j + \boldsymbol{b}_j + (\boldsymbol{\phi_{u_i}})_j\big)$. We note $\eta_0 = \boldsymbol{x}$ and $(\eta_j)_{j\geq 1}$ the hidden activations throughout the network. Hence, the parameters $\theta = (\boldsymbol{W}_j, \boldsymbol{b}_j)_{j=0}^{L}$ are shared between all examples and the modulation $\phi_{u_i} = ((\boldsymbol{\phi_{u_i}})_j)_{j=0}^{L-1}$ is specific to a single example. We compute the modulation parameters $\phi_{u_i} = ((\boldsymbol{\phi_{u_i}})_j)_{j=0}^{L-1}$ from z with a linear hypernetwork.

With the learned shared parameters (θ_u, w_u), the modulated INR enables two processes: decoding and encoding (see Figure 5.1). Decoding refers to mapping a given code z_{u_i} to the corresponding INR function $f_{\theta_u,\phi_{u_i}}$, where $\phi_{u_i} = h_u(z_{u_i})$, while encoding involves generating a code z_{u_i} given a function u_i, providing a compact representation of the function within the modulation space of the INR.

To obtain the compact code z_{u_i} for reconstructing the original field u_i using the INR, an inverse problem is solved through a procedure called *auto-decoding*. The objective is to compress the necessary information into z_{u_i} such that the reconstructed value $\tilde{u}_i(\boldsymbol{x}) = f_{\theta_u,\phi_{u_i}}(\boldsymbol{x})$ approximates the original value $u_i(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathcal{X}_i$. The approximate solution to this inverse problem is computed iteratively through a gradient descent optimization process:

$$\begin{aligned} &z_{u_i}^{(0)} = 0, \\ &z_{u_i}^{(k+1)} = z_{u_i}^{(k)} - \alpha \nabla_{z_{u_i}^{(k)}} \mathcal{L}_{\mu_i}(f_{\theta_u,\phi_{u_i}^{(k)}}, u_i), \\ &\text{with } \phi_{u_i}^{(k)} = h_u(z_{u_i}^{(k)}) \text{ for } 0 \leq k \leq K-1. \end{aligned} \tag{5.1}$$

where α is the inner loop learning rate, K the number of inner steps, and $\mathcal{L}_{\mu_i}(u_i, \tilde{u}_i) = \mathbb{E}_{\boldsymbol{x}\sim\mu_i}[(u_i(\boldsymbol{x}) - \tilde{u}_i(\boldsymbol{x}))^2]$ where μ_i is a measure defined through the observation grid $\mathcal{X}_i$ $\mu_i(\cdot) = \sum_{x\in\mathcal{X}_i} \delta_x(\cdot)$, with $\delta_x(\cdot)$ the Dirac measure.

As indicated before, we treat each input and output function independently: there are two input functions denoted as (d, n) and four output functions denoted as (v_x, v_y, p, ν_t). Each $u_i \in \{d, n, v_x, v_y, p, \nu_t\}$ is represented by a modulated INR

$f_{\theta_u,\phi_{u_i}}$, where u_i stands for a field specific to an airfoil example. INFINITY then learns a mapping between the latent representations of the geometric input fields and the latent representations of the physics output fields.

Inference As illustrated in Figure 5.1, INFINITY follows a three-step procedure: *encode*, *process*, and *decode*.

- *Encode:* Given the geometric input functions d_i, n_i and the corresponding INR learned parameters, respectively θ_d, w_d and θ_n, w_n, functions d_i, n_i are encoded into the latent codes z_{d_i}, z_{n_i} according to Equation (5.1). Since we can query the INRs anywhere within the domain, we can hence freely encode functions without mesh constraints. This lets us freely encode inputs with different geometries.

- *Process:* Once we obtain z_{d_i} and z_{n_i}, we can infer the latent output codes $\left(\hat{z}_{v_{xi}}, \hat{z}_{v_{y_i}}, \hat{z}_{p_i}, \hat{z}_{\nu_{ti}}\right) = g_\psi\Big(\left(z_{d_i}, z_{n_i}, V_{xi}, V_{y_i}\right)\Big)$. We consider here that g_ψ is implemented through an MLP with parameters ψ.

- *Decode:* We decode each processed output code $\left(\hat{z}_{v_{xi}}, \hat{z}_{v_{y_i}}, \hat{z}_{p_i}, \hat{z}_{\nu_{ti}}\right)$ with their associated hypernetwork and modulated INR. We make use of the INRs to freely query a physical field at any point within its spatial domain. These components generate the final output functions by mapping the latent codes back to the output space.

		INFINITY	GraphSAGE	MLP	Graph U-Net	PointNet
Volume	v_x	**0.06 ± 0.01**	0.83 ± 0.01	0.95 ± 0.06	1.52 ± 0.34	3.50 ± 1.04
	v_y	**0.06 ± 0.01**	0.99 ± 0.05	0.98 ± 0.17	2.03 ± 0.39	3.64 ± 1.26
	p	**0.25 ± 0.01**	0.66 ± 0.05	0.74 ± 0.13	0.66 ± 0.08	1.15 ± 0.23
	ν_t	**1.32 ± 0.08**	1.60 ± 0.21	1.90 ± 0.10	1.46 ± 0.14	2.92 ± 0.48
Surface	$p_{\|\mathcal{S}}$	**0.07 ± 0.01**	0.66 ± 0.10	1.13 ± 0.14	0.39 ± 0.07	0.93 ± 0.26
Relative error	C_D	**0.366 ± 0.023**	4.050 ± 0.704	4.289 ± 0.679	10.385 ± 1.895	14.637 ± 3.668
	C_L	**0.081 ± 0.007**	0.517 ± 0.162	0.767 ± 0.108	0.489 ± 0.105	0.742 ± 0.186
Spearman correlation	ρ_D	**0.578 ± 0.050**	-0.303 ± 0.124	-0.117 ± 0.256	-0.138 ± 0.258	-0.022 ± 0.097
	ρ_L	**0.997 ± 0.001**	0.965 ± 0.011	0.913 ± 0.018	0.967 ± 0.019	0.938 ± 0.023
Inference time (μs)		98 ± 70	20.9 ± 2.3	13.3 ± 0.2	357.8 ± 36.9	33.9 ± 3.5

Table 5.1: Test results on AirfRANS. Mean squared error (MSE) on normalized fields expressed with factor ($\times 10^{-2}$) for the volume and ($\times 10^{-1}$) for the surface. Relative errors C_D, C_L on the drag and lift and Spearman correlations ρ_D, ρ_L on the drag and lift. The results from the baselines are taken from Bonnet et al. (2022).

5.1.2.3 Training

We implement a two-step training procedure that first learns the modulated INR parameters θ_u and ϕ_{u_i} for all input and output functions, before training the map g_ψ. During the training of the INRs we force the *auto-decoding* process to take only a few gradient steps to encode the geometric functions or physical fields. This enhances the INR capability to encode new geometrical inputs in a few steps at test time, and also reduces the space size of the target output codes. This regularization prevents the different INRs to memorize the training sets into the individual codes. In order to obtain a network that is capable of quickly encoding new geometrical and physical inputs, we employ a second-order meta-learning training algorithm based on CAVIA (Zintgraf et al., 2019b). Compared to a first-order scheme such as Reptile (Nichol et al., 2018a), the outer loop back-propagates the gradient through the K inner steps, consuming more memory. Indeed, we need to compute gradients of gradients but this yields higher reconstruction results with the modulated INR. We experimentally found that using 3 inner-steps for training, or testing, was sufficient to obtain very low reconstruction errors for the geometric or physical fields. Using more inner-steps would result in a higher computation cost with only a marginal gain in reconstruction capacity. We outline the training pipeline of a modulated INR in Algorithm 2.

Algorithm 2: Modulated INR training

while *no convergence* **do**
  Sample a batch $\mathcal{B}$ of data $(u_i)_{i\in\mathcal{B}}$;
  Set codes to zero $z_{u_i} \leftarrow 0$ for i in $\mathcal{B}$;
  for $i \in \mathcal{B}$ *and step* $\in \{1, ..., K_u\}$ **do**
    $\phi_{u_i} = h_u(z_{u_i})$;
    $z_{u_i} \leftarrow z_{u_i} - \alpha_a \nabla_{z_{u_i}} \mathcal{L}_{\mathcal{X}_i}(f_{\theta_u,\phi_{u_i}}, u_i)$;
  end
  for i *in* $\mathcal{B}$*:* **do**
    $\phi_{u_i} = h_u(z_{u_i})$;
  end
  $\theta_u \leftarrow \theta_u - \eta \frac{1}{|\mathcal{B}|} \sum_{i\in\mathcal{B}} \nabla_{\theta_u} \mathcal{L}_{\mathcal{X}_i}(f_{\theta_u,\phi_{u_i}}, u_i)$;
  $w_u \leftarrow w_u - \eta \frac{1}{|\mathcal{B}|} \sum_{i\in\mathcal{B}} \nabla_{w_u} \mathcal{L}_{\mathcal{X}_i}(f_{\theta_u,\phi_{u_i}}, u_i)$;
end

Once the different INRs have been fitted, we encode the functions into the input codes z_{d_i}, z_{n_i} and target codes $z_{v_{xi}}, z_{v_{y_i}}, z_{p_i}, z_{\nu_{ti}}$. The training of g_ψ is performed in the small dimensional z-code space, and is supervised through the MSE loss with the target codes.

5.1.3 Experiments

Baselines We use the same baselines as Bonnet et al. (2022); GraphSAGE (Hamilton et al., 2017), a PointNet (Charles et al., 2017), a Graph U-Net (Gao and Ji, 2019) and a MLP. Those baselines have been initially chosen as they process in different ways the inputs. The results are given for the setup "full data regime" of AirfRANS, using 800 samples for training and 200 for testing.

Results In Table 5.1, the INFINITY model demonstrates superior inference capabilities on the volume and surface compared to the baselines. Indeed, It achieves significantly lower error values on the volume velocity and pressure fields, while exhibiting an order-of-magnitude lower MSE on the surface pressure. This substantial gain in prediction power translates to order of magnitude lower relative errors on the drag and lift forces, accompanied by high positive Spearman correlations. These results indicate a strong alignment between INFINITY's predictions and the true drag and lift forces. Consequently, INFINITY emerges as the only model capable of predicting accurately physical fields on the volume and surface while maintaining coherent and accurate drag and lift estimations. On the downside, the INFINITY model has a longer inference time compared to GraphSAGE and PointNet. However, this increased inference time is still within an acceptable range, considering its superior performance and that a numerical solver needs approximately 20 minutes to complete a simulation. Furthermore, it is counterbalanced by the ability to query the full mesh directly, in stark contrast to graph-based methods that necessitate sub-sampling to process the inputs.

5.1.4 Conclusion

We introduce INFINITY, a model that utilizes coordinate-based networks to encode geometric information and physical fields into compact representations. INFINITY establishes a mapping between geometry and physical fields within a reduced representation space. We validated our model on AirfRANS, a challenging dataset for the Reynolds-Averaged Navier-Stokes equation, where it significantly outperforms previous baselines across all relevant performance metrics. At post-processing stage, the predicted fields yield accurate lift and drag forces. This validates INFINITY's potential as a surrogate design model, where it could be plugged in any design optimization or exploration loop.

5.2 Time Series Continuous Modeling for Imputation and Forecasting with Implicit Neural Representations

We introduce a novel modeling approach for time series imputation and forecasting, tailored to address the challenges often encountered in real-world data, such as irregular samples, missing data, or unaligned measurements from multiple sensors. Our method relies on a continuous-time-dependent model of the series' evolution dynamics. It leverages adaptations of conditional, implicit neural representations for sequential data. A modulation mechanism, driven by a meta-learning algorithm, allows adaptation to unseen samples and extrapolation beyond observed time-windows for long-term predictions. The model provides a highly flexible and unified framework for imputation and forecasting tasks across a wide range of challenging scenarios. It achieves state-of-the-art performance on classical benchmarks and outperforms alternative time-continuous models.

5.2.1 Introduction

Time series analysis and modeling are ubiquitous in a wide range of fields, including industry, medicine, and climate science. The variety, heterogeneity and increasing number of deployed sensors, raise new challenges when dealing with real-world problems for which current methods often fail. For example, data are frequently irregularly sampled, contain missing values, or are unaligned when collected from distributed sensors (Schulz and Stattegger, 1997; Clark and Bjørnstad, 2004). Recent advancements in deep learning have significantly improved state-of-the-art performance in both data imputation (Cao et al., 2018; Du et al., 2023) and forecasting tasks (Zeng et al., 2022; Nie et al., 2022). Many state-of-the-art models, such as transformers, have been primarily designed for dense and regular grids (Wu et al., 2021; Nie et al., 2022; Du et al., 2023). They struggle to handle irregular data and often suffer from significant performance degradation (Chen et al., 2001; Kim et al., 2019b).

Our objective is to explore alternatives to SOTA transformers able to handle, in a unified framework, imputation and forecasting tasks for irregularly, arbitrarily sampled, and unaligned time series sources. Time-dependent continuous models (Rasmussen and Williams, 2006; Garnelo et al., 2018; Rubanova et al., 2019) offer such an alternative. However, until now, their performance has lagged significantly behind that of models designed for regular discrete grids. A few years ago, implicit neural representations (INRs) emerged as a powerful tool for representing images as continuous functions of spatial coordinates (Sitzmann et al., 2020b; Tancik et al., 2020a) with recent new applications such as image generation (Dupont et al., 2022b)

or even modeling dynamical systems (Yin et al., 2023).

In this work, we leverage the potential of conditional INR models within a meta-learning approach to introduce TimeFlow: a unified framework designed for modeling continuous time series and addressing imputation and forecasting tasks with irregular and unaligned observations. Our key contributions are the following:

- We propose a novel framework that excels in modeling time series as continuous functions of time, accepting arbitrary time step inputs, thus enabling the handling of irregular and unaligned time series for both imputation and forecasting tasks. This is one of the very first attempts to adapt INRs that enables efficient handling of both imputation and forecasting tasks within a unified framework. The methodology which leverages the synergy between the model components, evidenced in the context of this application, is a pioneering contribution to the field.

- We conducted an extensive comparison with state-of-the-art continuous and discrete models. It demonstrates that our approach outperforms continuous and discrete SOTA deep learning approaches for imputation. As for long-term forecasting, it outperforms existing continuous models both on regular and irregular samples. It is on par with SOTA discrete models on regularly sampled time series while allowing for a much greater flexibility for irregular samplings, allowing to cope with situations where discrete models fail. Furthermore, we prove that our method effortlessly handles previously unseen time series and new time windows, making it well-suited for real-world applications.

5.2.2 Related work

Discrete methods for time series imputation and forecasting. Recently, Deep Learning (DL) methods have been widely used for both time series imputation and forecasting. For imputation, BRITS (Cao et al., 2018) uses a bidirectional recurrent neural network (RNN). Alternative frameworks were later explored, e.g., GAN-based (Luo et al., 2018, 2019; Liu et al., 2019), VAE-based (Fortuin et al., 2020), diffusion-based (Tashiro et al., 2021), matrix factorization-based (TIDER, Liu et al., 2023) and transformer-based (SAITS, Du et al., 2023) approaches. These methods cannot handle irregular time series. In situations involving multiple sensors, such as those placed at different locations, incorporating new sensors necessitates retraining the entire model, thereby limiting their usability. For forecasting, most recent DL SOTA models are based on transformers. Initial approaches apply plain transformers directly to the series, each token being a series element (Zhou et al., 2021; Liu et al., 2022; Wu et al., 2021; Zhou et al., 2022). These transformers may underperform linear models as shown in (Zeng et al., 2022). PatchTST (Nie et al.,

2022) significantly improved transformers SOTA performance by considering subseries as tokens of the series. However, all these models cannot handle properly irregularly sampled look-back windows.

Continuous methods for time series. Gaussian Processes (Rasmussen and Williams, 2006) have been a popular family of methods for modeling time series as continuous functions. They require choosing an appropriate kernel (Corani et al., 2021) and may suffer limitations in large dimensions settings. Neural Processes (NPs) (Garnelo et al., 2018; Kim et al., 2019b) parameterize Gaussian processes through an encoder-decoder architecture leading to more computationally efficient implementations. NPs have been used to model simple signals for imputation and forecasting tasks, but struggle with more complex signals. Bilos et al. (2023) parameterizes a Gaussian Process through a diffusion model, but the model has difficulty adapting to a large number of timestamps. Other approaches such as Brouwer et al. (2019) and Rubanova et al. (2019) model time series continuously with latent ordinary differential equations. mTAN (Shukla and Marlin, 2021) a transformer model uses an attention mechanism to impute irregular time series. While these approaches have shown significant progress in continuous modeling for time series, they are less efficient than the aforementioned discrete models for regular time series and lack extrapolation capability when dealing with complex dynamics.

Implicit neural representations. The recent development of implicit neural representations (INRs) has led to impressive results in computer vision (Sitzmann et al., 2020b; Tancik et al., 2020a; Fathony et al., 2021; Mildenhall et al., 2021). INRs can represent data as a continuous function, which can be queried at any coordinate. While they have been applied in other fields such as physics (Yin et al., 2023) and meteorology (Huang and Hoefler, 2023), there has been limited research on INRs for time series analysis. Prior works (Fons et al., 2022; Jeong and Shin, 2022) focused on time series generation for data augmentation and on time series encoding for reconstruction but are limited by their fixed grid input requirement. DeepTime (Woo et al., 2022) is the closest work to our contribution. DeepTime learns a set of basis INR functions from a training set of multiple time series and combines them using a Ridge regressor. This regressor allows it to adapt to new time series. It has been designed for forecasting only. The original version cannot handle imputation properly and was adapted to do so for our comparisons. In our experiments, we will demonstrate that TimeFlow significantly outperforms DeepTime in imputation and also in forecasting tasks when dealing with missing values in the look-back window. TimeFlow also shows a slight advantage over DeepTime in forecasting regularly sampled series.

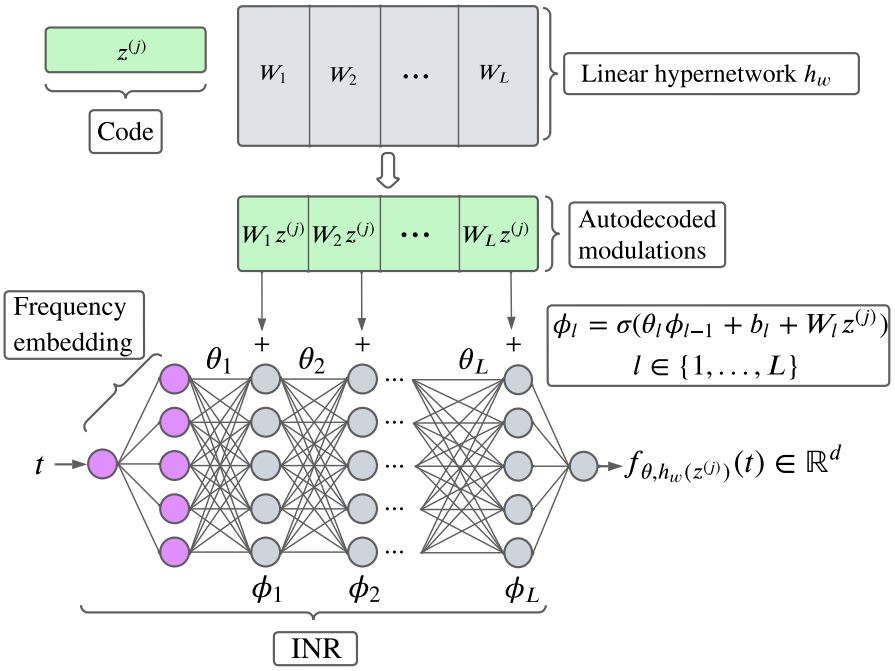


Figure 5.2: Overview of TimeFlow architecture. Forward pass to approximate the time series $x^{(j)}$. σ stands for the ReLU activation function.

5.2.3 The TimeFlow framework

In this section, we present TimeFlow, a modulated-INR architecture that can be used to reconstruct, impute and forecast time series.

5.2.3.1 Problem setting

We aim to develop a unified framework for time series imputation and forecasting that reduces dependency on a fixed sampling scheme for time series. We introduce the following notations for both tasks. During training, in the imputation setting, we have access to time series in an observation set denoted as $\mathcal{T}_{in}$, which is a subset of the complete time series observation set $\mathcal{T}$. In the forecasting setting, we observe time series within a limited past time grid, referred to as the 'look-back window' and denoted as $\mathcal{T}_{in}$ (a subset of $\mathcal{T}$), as well as a future grid, the 'horizon', denoted as $\mathcal{T}_{out}$ (also a subset of $\mathcal{T}$). At test time, in both cases, and given an observed subset $\mathcal{T}^*_{in}$ included in a possibly new temporal window $\mathcal{T}^*$, our objective is to infer the time series values within $\mathcal{T}^*$.

5.2.3.2 Key components

Our framework is articulated around three key components. (i) **INR-based time–continuous functions**: a time series x is represented by a time-continuous function

$f: t \in \mathbb{R}_+ \to f(t) \in \mathbb{R}^d$ that can be queried at any time t. For that, we employ implicit neural representations (INRs), which are neural networks capable of learning a parameterized continuous function f_θ from discrete data by minimizing the reconstruction loss between observed data and network's outputs. (ii) **Conditional INRs with modulations**: An INR can represent only one function, whether it's an image or a time series. To effectively represent a collection of time series $(x^{(j)})_j$ using INRs, we improve their encoding by incorporating per-sample modulations, which we denote as $\psi^{(j)}$. These modulations condition the parameters θ of the INRs. We use the notation $f_{\theta,\psi^{(j)}}$ to refer to the conditioned INR with the modulations $\psi^{(j)}$. (iii) **Optimization-based encoding**: the conditioning modulation parameters $\psi^{(j)}$ are calculated as a function of codes $z^{(j)}$ that represent the individual sample series. We acquire these codes $z^{(j)}$ through a meta-learning optimization process using an auto-decoding strategy. Notably, auto-decoding has been found to be more efficient for this purpose than set encoders (Kim et al., 2019b). In the following sections, we will elaborate on each component of our method. Given that the choices made for each component and the methodology developed to enhance their synergy are essential aspects, we provide a discussion of the various choices involved in section 5.2.3.4.

INR-based time-continuous functions. We implement our INR with Fourier features and a feed-forward network (FFN) with ReLU activations, i.e. for a time coordinate $t \in \mathcal{T}$, the output of the INR f_θ is given by $f_\theta(t) = \text{FFN}(\gamma(t))$. The Fourier Features $\gamma(\cdot)$ are a frequency embedding of the time coordinates used to capture high-frequencies (Tancik et al., 2020a; Mildenhall et al., 2021). In our case, we chose $\gamma(t) := (\sin(\pi t), \cos(\pi t), \cdots, \sin(2^{N-1}\pi t), \cos(2^{N-1}\pi t))$, with N the number of fixed frequencies. For an INR with L layers, the output is computed as follows: (i) we get the frequency embedding $\phi_0 = \gamma(t)$, (ii) we update the hidden states according to $\phi_l = \text{ReLU}(\theta_l \phi_{l-1} + b_l)$ for $l = 1, \ldots, L$, (iii) we project onto the output space $f_\theta(t) = \theta_{L+1}\phi_L + b_{L+1}$.

Conditional INRs with modulations. As indicated, sample conditioning of the INR is performed through modulations of its parameters. In order to adapt rapidly the model to new samples, the conditioning should rely only on a small number of the INR parameters. This is achieved by modifying only the biases of the INR through the introduction of an additional bias term $\psi_l^{(j)}$ for each layer l, also known as *shift modulation*. To further limit the versatility of the conditioning, we generate the instance modulations $\psi^{(j)}$ from compact codes $z^{(j)}$ through a linear hypernetwork h with parameters w, i.e., $\psi^{(j)} = h_w(z^{(j)})$. Consequently, the approximation of a time series $x^{(j)}$, denoted globally as $f_{\theta,h_w(z^{(j)})}$, will depend on shared parameters θ and w that are common among all the INRs involved in modeling the series family and on the code $z^{(j)}$ specific to series $x^{(j)}$. The output of the

l-th layer of the modulated INR is given by $\phi_l = \text{ReLU}(\theta_l \phi_{l-1} + b_l + \psi_l^{(j)})$, where $\psi_l^{(j)} = W_l z^{(j)}$, and $w := (W_l)_{l=1}^L$ are the parameters of the hypernetwork h_w. This design enables gathering information across samples into the common parameters of the INR and hypernetwork, while the codes contain only specific information about their respective time-series samples. The architecture is illustrated in fig. 5.2.

Algorithm 3: TimeFlow Training

while *no convergence* **do**
  Sample batch $\mathcal{B}$ of data $(x^{(j)})_{j\in\mathcal{B}}$;
  Set codes to zero $z^{(j)} \leftarrow 0, \forall j \in \mathcal{B}$;
  `// inner loop for encoding:`
  for $j \in \mathcal{B}$ *and step* $\in \{1, ..., K\}$ **do**
    $z^{(j)} \leftarrow z^{(j)} - \alpha \nabla_{z^{(j)}} \mathcal{L}_{\mathcal{T}_{in}^{(j)}}(f_{\theta, h_w(z^{(j)})}, x^{(j)})$;
  end
  `// outer loop step:`
  $[\theta, w] \leftarrow$
   $[\theta, w] - \eta \nabla_{[\theta,w]} \frac{1}{|\mathcal{B}|} \sum_{j\in\mathcal{B}} [\mathcal{L}_{\mathcal{T}_{in}^{(j)}}(f_{\theta, h_w(z^{(j)})}, x^{(j)}) + \lambda \mathcal{L}_{\mathcal{T}_{out}^{(j)}}(f_{\theta, h_w(z^{(j)})}, x^{(j)})]$;
end

Optimization-based encoding. We condition the INR using the data from $\mathcal{T}_{in}$, and learn the shared INR and hypernetwork parameters θ and w using $\mathcal{T}_{in}$ for both imputation and forecasting, and $\mathcal{T}_{out}$ for forecasting only. We achieve the conditioning on $\mathcal{T}_{in}$ by optimizing the codes $z^{(j)}$ through gradient descent. The joint optimization of the codes and common parameters is challenging. In TimeFlow, it is achieved through a meta-learning approach, adapted from Dupont et al. (2022b) and Zintgraf et al. (2019a). The objective is to learn shared parameters so that the code $z^{(j)}$ can be adapted in just a few gradient steps for a new series $x^{(j)}$. For training, we perform parameter optimization at two levels: the inner-loop and the outer-loop. The inner-loop adapts the code $z^{(j)}$ to condition the network on the set $\mathcal{T}_{in}^{(j)}$, while the outer-loop updates the common parameters using $\mathcal{T}_{in}^{(j)}$ and also $\mathcal{T}_{out}^{(j)}$ for forecasting. We present our training optimization in Algorithm 3. At each training epoch and for each batch of data $\mathcal{B}$ composed of time series $x^{(j)}$ sampled from the training set, we first update individually the codes $z^{(j)}$ in the inner loop, before updating the common parameters in the outer loop using a loss over the whole batch. We introduce a parameter λ to weight the importance of the loss over $\mathcal{T}_{out}$ w.r.t. the loss over $\mathcal{T}_{in}$ for the outer-loop. In practice, when $\mathcal{T}_{out}$ exists, i.e. for forecasting, we set $\lambda = 1$ and $\lambda = 0$ otherwise. We use an MSE loss over the observations grid $\mathcal{L}_{\mathcal{T}}(x_t, \tilde{x_t}) := \mathbb{E}_{t\sim\mathcal{T}}[(x_t - \tilde{x_t})^2]$. We denote α and η the learning rates of the inner- and outer-loop. Using $K = 3$ steps for training and testing is sufficient for our experiments thanks to the use of second-order meta-learning as explained in section 5.2.3.4.

Algorithm 4: TimeFlow Inference with trained θ, w

For the j-th series $(x^{(j)})$, set code to zero $z^{*(j)} \leftarrow 0$;
for *step* $\in \{1, ..., K\}$ **do**
  $z^{*(j)} \leftarrow z^{*(j)} - \alpha \nabla_{z^{*(j)}} \mathcal{L}_{\mathcal{T}_{in}^{*(j)}}(f_{\theta, h_w(z^{*(j)})}, x_t)$;
end
Query $f_{\theta, h_w(z^{*(j)})}(t)$ for any $t \in \mathcal{T}^{*(j)}$;

5.2.3.3 TimeFlow inference

During the inference process, we aim to infer the time series value for each timestamp in the dense grid $\mathcal{T}^{*(j)}$ based on the partial observation grid $\mathcal{T}_{in}^{*(j)} \subset \mathcal{T}^{*(j)}$. We can encounter two scenarios: (i) One where we observe the same time window as during training ($\mathcal{T}^{*(j)} = \mathcal{T}^{(j)}$) as in the imputation setting in section 5.2.4.1. (ii) One, where we are dealing with a newly observed time window ($\mathcal{T}^{*(j)} \neq \mathcal{T}^{(j)}$), as in the forecasting setting in section 5.2.4.2. At inference, the parameters θ and w are kept fixed to their final training values. We optimize the individual parameters $z^{*(j)}$ based on the newly observed grid $\mathcal{T}_{in}^{*(j)}$ using the K inner-steps of the meta-learning algorithm as described in algorithm 4. We are then in position to query $f_{\theta, h_w(z^{*(j)})}(t)$ for any given timestamp $t \in \mathcal{T}^{*(j)}$.

5.2.3.4 Discussion on implementation choices

As indicated before, adapting the components and enhancing their synergy for the tasks of imputation and forecasting is not trivial and require careful choices. We conducted several ablation studies to provide a comprehensive examination of key implementation choices of our framework. Our findings indicate that: • An FFN with Fourier Features outperformed other popular INRs for the tasks considered in this study (Appendix C.1.1.2). • TimeFlow with a set encoder for learning the compact conditioning codes z in place of the auto-decoding strategy used here, proved much less effective on complex datasets (Appendix C.1.1.5, Table C.5). • Replacing the 2nd-order optimization for a 1st-order one, such as REPTILE, led to unstable training (Appendix C.1.1.5, Table C.4). • Complexifying the modulation by introducing scaling parameters in addition to shift parameters did not provide performance gains (Appendix C.1.1.6). • Using 3 inner steps for training and inference struck a favorable balance between reconstruction capabilities and computational efficiency (Appendix C.1.1.4). • A latent code z dimension of 128 was optimal for our tasks (Appendix C.1.1.3).

5.2.4 Experiments

We conducted a comprehensive evaluation of our TimeFlow framework across three different tasks, comparing its performance to state-of-the-art continuous and discrete baseline methods. In Section 5.2.4.1, we assess TimeFlow's capabilities to impute sparsely observed time series under various sampling rates. Section 5.2.4.2 focuses on long-term forecasting, where we evaluate TimeFlow over standard long-term forecasting horizons. In Section 5.2.4.3, we tackle a challenging task forecasting with incomplete look-back windows, thus combining the challenges of imputation and forecasting. This demonstrates TimeFlow's versatility and performance.

Datasets. Our approach is well-suited for handling a large number of homogeneous phenomena measured over time. We tested our framework on three extensive multivariate datasets where a single phenomenon is measured at multiple locations over time. They are commonly used in time series imputation and long-term forecasting literature. The *Electricity* dataset comprises hourly electricity load curves of 321 customers in Portugal, spanning the years 2012 to 2014. The *Traffic* dataset is composed of hourly road occupancy rates from 862 locations in San Francisco during 2015 and 2016. Lastly, the *Solar* dataset contains measurements of solar power production from 137 photovoltaic plants in Alabama, recorded at 10-minute intervals in 2006. Additionally, we have created an hourly version, *SolarH*, for the sake of consistency in the forecasting section. These datasets exhibit diversity in various characteristics: • They exhibit diverse temporal frequencies, including daily and weekly seasonality observed in the *Traffic* and *Electricity* datasets, while the *Solar* dataset possesses only daily frequency. • There is individual variability across data samples and more pronounced trends in the *Electricity* dataset compared to the *Traffic* and *Solar* datasets.

5.2.4.1 Imputation

We consider the classical imputation setting where n time series are partially observed over a given time window. Using our approach, we can predict for each time series the value at any timestamp t in that time window based on partial observations.

Setting. For a time series $x^{(j)}$, we denote the set of observed points as $\mathcal{T}_{in}^{(j)}$ and the ground truth set of points as $\mathcal{T}^{(j)}$. The observed time grids may be irregularly spaced and may differ across the different time series ($\mathcal{T}_{in}^{(j_1)} \neq \mathcal{T}_{in}^{(j_2)}, \forall j_1 \neq j_2$). The model is trained for each $x^{(j)}$ following algorithm 3. Then, we aim to infer for any unobserved $t \in \mathcal{T}^{(j)}$ the missing value $x_t^{(j)}$ conditioned on $\mathcal{T}_{in}^{(j)}$ according to algorithm 4. For this imputation task, the TimeFlow training and inference procedures are detailed in section 5.2.3 and illustrated in fig. 5.3. For comparison

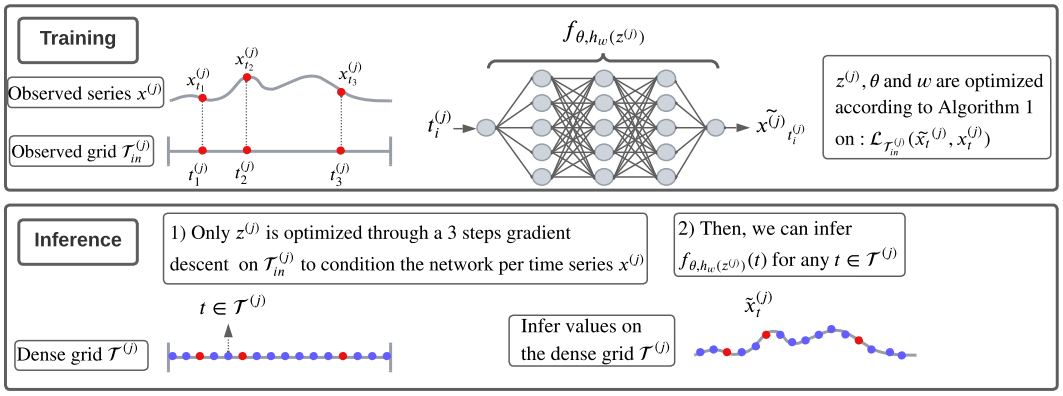


Figure 5.3: Training and inference procedures of TimeFlow for imputation. (i) During training, for each time series $x^{(j)}$, our observations (red dots ●) are restricted to the sparsely sampled grid, denoted as $\mathcal{T}_{in}^{(j)}$. (ii) During inference, our objective is to infer the values over the dense grids $\mathcal{T}^{(j)}$, on the unobserved data points (such as the blue dots ● on the figure).

with the SOTA imputation baselines, we assume that the ground truth time grid is the same for each sample. The subsampling rate τ is define as the rate of observed values.

Baselines. We compare TimeFlow with various baselines, including discrete imputation methods, such as CSDI (Tashiro et al., 2021), SAITS (Du et al., 2023), BRITS (Cao et al., 2018), and TIDER (Liu et al., 2023), and continuous ones, such as Neural Process (NP, Garnelo et al., 2018), mTAN (Shukla and Marlin, 2021), and DeepTime with slight adjustments (Woo et al., 2022) (details cf. appendix C.1.3.3). For each dataset, we divide the series into five independent periods (each time window consists of 2000 timestamps for *Electricity* and *Traffic*, and 10,000 timestamps for *Solar*), perform imputation on each time window and average the performance to obtain robust results. We evaluate the quality of the models for different subsampling rates, ranging from the easiest $\tau = 0.5$ to the most difficult $\tau = 0.05$.

Results. We show in table 5.2 that TimeFlow outperforms both discrete and continuous models across almost all τs for the given datasets. The relative improvements of TimeFlow over baselines are significant, ranging from 15% to 50%. Especially for the lowest sampling rate $\tau = 0.05$, TimeFlow outperforms all discrete baselines, demonstrating the advantages of continuous modeling. Additionally, it achieves lower imputation errors compared to continuous models in all but one cases. Qualitatively, we see on example series in fig. 5.4 that our model shows significant imputation capabilities, with on a subsampling rate at $\tau = 0.1$ on the *Electricity* dataset. It captures well different frequencies and amplitudes in a chal-

Table 5.2: Mean MAE imputation results on the missing grid only. Each time series is divided into 5 time windows onto which imputation is performed, and the performances are averaged over the 5 windows. In the table, τ stands for the subsampling rate, i.e. the proportion of observed points considered for each time window. Bold results are best, underlined results are second best. TimeFlow improvement represents the overall percentage improvement achieved by TimeFlow compared to the specific method being considered.

		Continuous methods				Discrete methods			
	τ	TimeFlow	DeepTime	mTAN	Neural Process	CSDI	SAITS	BRITS	TIDER
Electricity	0.05	**0.324 ± 0.013**	0.379 ± 0.037	0.575 ± 0.039	0.357 ± 0.015	0.462 ± 0.021	0.384 ± 0.019	0.329 ± 0.015	0.427 ± 0.010
	0.10	**0.250 ± 0.010**	0.333 ± 0.034	0.412 ± 0.047	0.417 ± 0.057	0.398 ± 0.072	0.308 ± 0.011	0.287 ± 0.015	0.399 ± 0.009
	0.20	**0.225 ± 0.008**	0.244 ± 0.013	0.342 ± 0.014	0.320 ± 0.017	0.341 ± 0.068	0.261 ± 0.008	0.245 ± 0.011	0.391 ± 0.010
	0.30	**0.212 ± 0.007**	0.240 ± 0.014	0.335 ± 0.015	0.300 ± 0.022	0.277 ± 0.059	0.236 ± 0.008	0.221 ± 0.008	0.384 ± 0.009
	0.50	0.194 ± 0.007	0.227 ± 0.012	0.340 ± 0.022	0.297 ± 0.016	**0.168 ± 0.003**	0.209 ± 0.008	0.193 ± 0.008	0.386 ± 0.009
Solar	0.05	**0.095 ± 0.015**	0.190 ± 0.020	0.241 ± 0.102	0.115 ± 0.015	0.374 ± 0.033	0.142 ± 0.016	0.165 ± 0.014	0.291 ± 0.009
	0.10	**0.083 ± 0.015**	0.159 ± 0.013	0.251 ± 0.081	0.114 ± 0.014	0.375 ± 0.038	0.124 ± 0.018	0.132 ± 0.015	0.276 ± 0.010
	0.20	**0.072 ± 0.015**	0.149 ± 0.020	0.314 ± 0.035	0.109 ± 0.016	0.217 ± 0.023	0.108 ± 0.014	0.109 ± 0.012	0.270 ± 0.010
	0.30	**0.061 ± 0.012**	0.135 ± 0.014	0.338 ± 0.05	0.108 ± 0.016	0.156 ± 0.002	0.100 ± 0.015	0.098 ± 0.012	0.266 ± 0.010
	0.50	**0.054 ± 0.013**	0.098 ± 0.013	0.315 ± 0.080	0.107 ± 0.015	0.079 ± 0.011	0.094 ± 0.013	0.088 ± 0.013	0.262 ± 0.009
Traffic	0.05	0.283 ± 0.016	**0.246 ± 0.010**	0.406 ± 0.074	0.318 ± 0.014	0.337 ± 0.045	0.293 ± 0.007	0.261 ± 0.010	0.363 ± 0.007
	0.10	**0.211 ± 0.012**	0.214 ± 0.007	0.319 ± 0.025	0.288 ± 0.018	0.288 ± 0.017	0.237 ± 0.006	0.245 ± 0.009	0.362 ± 0.006
	0.20	**0.168 ± 0.006**	0.216 ± 0.006	0.270 ± 0.012	0.271 ± 0.011	0.269 ± 0.017	0.197 ± 0.005	0.224 ± 0.008	0.361 ± 0.006
	0.30	**0.151 ± 0.007**	0.172 ± 0.008	0.251 ± 0.006	0.259 ± 0.012	0.240 ± 0.037	0.180 ± 0.006	0.197 ± 0.007	0.355 ± 0.006
	0.50	**0.139 ± 0.007**	0.171 ± 0.005	0.278 ± 0.040	0.240 ± 0.021	0.144 ± 0.022	0.160 ± 0.008	0.161 ± 0.060	0.354 ± 0.007
TimeFlow improvement		/	20.5 %	49.1 %	30.5 %	38.9 %	16.9 %	14.7 %	50.9 %

lenging case (sample 35), although it underestimates the amplitude of some peaks. In a more challenging scenario (sample 25), where the series exhibit additional trend changes and frequency variations within the data, TimeFlow correctly imputes most timestamps, outperforming BRITS, which is the best-performing method for the *Electricity* dataset.

Imputation on previously unseen time series. In more practical scenarios, such as cases involving the installation of new sensors, we often encounter new time series originating from the same underlying phenomenon. In such instances, it becomes crucial to make inferences for these previously unseen time series. Thanks to efficient adaptation in latent space, our model can easily be applied to these new time series (as shown in appendix C.1.3.2, table C.9), contrasting with SOTA methods like SAITS and BRITS, which require full model retraining on the whole set of time series.

5.2.4.2 Forecasting

In this section, we are interested in the conventional long-term forecasting scenario. It consists in predicting the phenomenon in a specific future period, the horizon, based on the history of a limited past period, the look-back window. The forecaster is trained on a set of n observed time series for a given time window (train period)

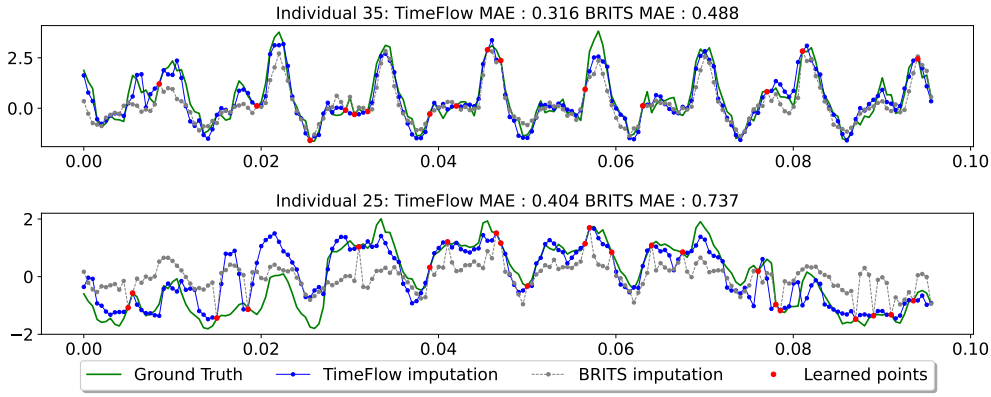


Figure 5.4: *Electricity dataset.* TimeFlow imputation (blue line) and BRITS imputation (gray line) with 10% of known point (red points) on the eight first days of samples 35 (top) and 25 (bottom).

and tested on new time windows.

Setting For a given time series $x^{(j)}$, $\mathcal{T}_{in}^{(j)}$ denotes the look-back window and $\mathcal{T}_{out}^{(j)}$ the horizon of H points. During training, at each epoch, we train $f_{\theta,h_w(z^{(j)})}$ following algorithm 3 with randomly drawn pairs of look-back window and horizon $(\mathcal{T}_{in}^{(j)} \cup \mathcal{T}_{out}^{(j)})_{j\in\mathcal{B}}$ within the observed train period. Then, for a new time window $\mathcal{T}^{*(j)}$, given a look-back window $\mathcal{T}_{in}^{*(j)}$ we forecast future values any $t \in \mathcal{T}^{*(j)}$, the horizon interval, following algorithm 4. We illustrate the training and inference of TimeFlow for the forecasting task in fig. 5.5.

Baselines. To evaluate the quality of our model in long-term forecasting, we compare it to the discrete baselines PatchTST (Nie et al., 2022), DLinear (Zeng et al., 2022), AutoFormer (Wu et al., 2021), and Informer (Zhou et al., 2021). We also include continuous baselines DeepTime and Neural Process (NP). In table 5.3, we present the forecasting results for standard horizons in long-term forecasting: $H \in \{96, 192, 336, 720\}$. The look-back window length is fixed to 512.

Results. The results in table 5.3 show that our approach ranks in the top two across all datasets and horizons and is the overall best continuous method. TimeFlow's performance is comparable to the current SOTA model PatchTST, with only 2% relative difference. Moreover, TimeFlow shows consistent results across the three datasets, whereas the other best discrete and continuous baselines, i.e. PatchTST and DeepTime, performance drops for some datasets. We also note that, despite the great performance of the SOTA PatchTST, other transformer-based baselines (discrete methods in table 5.3) perform poorly. We provide a detailed insight on these results in appendix C.1.4.1. Overall, although this evaluation setting favors discrete methods because the time series are observed at evenly distributed

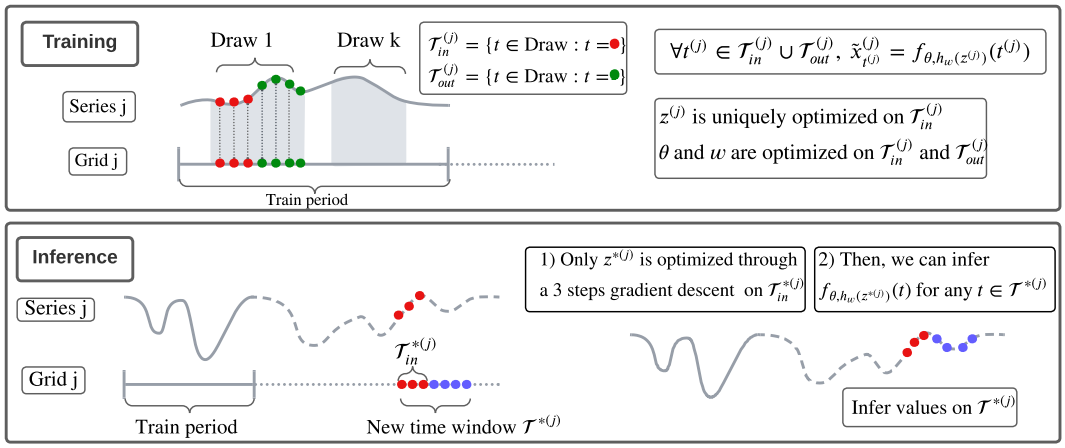


Figure 5.5: Training and inference procedure of TimeFlow for forecasting. (i) During training (top-figure), for each time series $x^{(j)}$, we observe some look-back window/horizon drawing pairs in the trained period. TimeFlow is trained with algorithm 3 to predict all observed timestamps in this drawing pairs while being conditioned by the observed look-back window. (ii) Once TimeFlow is optimized, the objective during inference (bottom-figure) is to infer the horizon over new time windows (blue dots •) while being conditioned by the newly observed look-back window (red dots •).

time steps, TimeFlow consistently performs as well as PatchTST and outperforms all the other methods, whether discrete or continuous. It is the first time that a continuous model has achieved the same level of performance as discrete methods within their specific setting.

Forecasting on previously unseen time series. TimeFlow considers that the series observed at different locations are independent, similar to PatchTST, NP, and DeepTime. This allows it to generalize to previously unseen time series from the same phenomenon. Note that this is not the case for most discrete methods. We show in appendix C.1.4.4, table C.14 that TimeFlow is able to generalize to previously unseen time series with no significant performance drop.

5.2.4.3 Challenging task: Forecast while imputing incomplete look-back windows

In real-world scenarios, it is common to encounter missing or irregularly sampled series when making predictions on new time windows (Cinar et al., 2018; Tang et al., 2020). Continuous methods can handle these cases, as they are designed to accommodate irregular sampling within the look-back window. In this section, we formulate a task to simulate these real-world scenarios. It's worth noting that this task is often encountered in practice but is rarely considered in the DL literature.

Table 5.3: Mean MAE forecast results averaged over different time windows. Each time, the model is trained on one time window and tested on the others (there are 2 windows for *SolarH* and 5 for *Electricity* and *Traffic*). H stands for the horizon. Bold results are best, and underlined results are second best. TimeFlow improvement represents the overall percentage improvement achieved by TimeFlow compared to the specific method being considered.

		Continuous methods			Discrete methods			
	H	TimeFlow	DeepTime	Neural Process	Patch-TST	DLinear	AutoFormer	Informer
Electricity	96	0.228 ± 0.028	0.244 ± 0.026	0.392 ± 0.045	**0.221 ± 0.023**	0.241 ± 0.030	0.546 ± 0.277	0.603 ± 0.255
	192	0.238 ± 0.020	0.252 ± 0.019	0.401 ± 0.046	**0.229 ± 0.020**	0.252 ± 0.025	0.500 ± 0.190	0.690 ± 0.291
	336	0.270 ± 0.031	0.284 ± 0.034	0.434 ± 0.076	**0.251 ± 0.027**	0.288 ± 0.038	0.523 ± 0.188	0.736 ± 0.271
	720	0.316 ± 0.055	0.359 ± 0.051	0.607 ± 0.150	**0.297 ± 0.039**	0.365 ± 0.059	0.631 ± 0.237	0.746 ± 0.265
SolarH	96	**0.190 ± 0.013**	**0.190 ± 0.020**	0.221 ± 0.048	0.262 ± 0.070	0.208 ± 0.014	0.245 ± 0.045	0.248 ± 0.022
	192	**0.202 ± 0.020**	0.204 ± 0.028	0.244 ± 0.048	0.253 ± 0.051	0.217 ± 0.022	0.333 ± 0.107	0.270 ± 0.031
	336	0.209 ± 0.017	**0.199 ± 0.026**	0.240 ± 0.006	0.259 ± 0.071	0.217 ± 0.026	0.334 ± 0.079	0.328 ± 0.048
	720	**0.218 ± 0.041**	0.229 ± 0.024	0.403 ± 0.147	0.267 ± 0.064	0.249 ± 0.034	0.351 ± 0.055	0.337 ± 0.037
Traffic	96	0.217 ± 0.032	0.228 ± 0.032	0.283 ± 0.027	**0.203 ± 0.037**	0.228 ± 0.033	0.319 ± 0.059	0.372 ± 0.078
	192	0.212 ± 0.028	0.220 ± 0.022	0.292 ± 0.024	**0.197 ± 0.030**	0.221 ± 0.023	0.368 ± 0.057	0.511 ± 0.247
	336	0.238 ± 0.034	0.245 ± 0.038	0.305 ± 0.039	**0.222 ± 0.039**	0.250 ± 0.040	0.434 ± 0.061	0.561 ± 0.263
	720	0.279 ± 0.050	0.290 ± 0.052	0.339 ± 0.038	**0.269 ± 0.057**	0.300 ± 0.057	0.462 ± 0.062	0.638 ± 0.067
TimeFlow improvement		/	4.30 %	32.2 %	-2.14 %	14.31 %	47.57 %	54.83 %

Setting and baselines. This scenario is similar to the forecast setting in section 5.2.4.2 and illustrated in fig. 5.5. The difference is that during inference, the look-back window is subsampled at a rate τ smaller than the one used for the training phase. This simulates a situation with missing observations in the look back window. Consequently, two distinct tasks emerge during the inference phase: imputing missing points within the sparsely observed look-back window, and forecasting over the horizon with this degraded context. In table 5.4, we compare to the two other continuous baselines, DeepTime and NP on *Electricity* and *Traffic* for different τs and horizons.

Results. In table 5.4, the results show that TimeFlow consistently outperforms other methods in imputation and forecasting for every scenarios. When comparing with the complete look-back windows observations scenario from table 5.3, one observes that at a 0.5 sampling rate, TimeFlow presents only a slight reduction in performance, whereas other baseline methods experience more significant drops. For instance, when we compare forecast results between a complete window and a $\tau = 0.5$ subsampled window for *Electricity* with a forecasting horizon of $H = 96$, TimeFlow's error increases by a mere 4.6% (from 0.228 to 0.239). In contrast, DeepTime's error grows by over 10% (from 0.244 to 0.270), and NP experiences a rise of around 25% (from 0.392 to 0.486). For lower sampling rates, TimeFlow still delivers correct predictions. Qualitatively, we see on the series example in fig. 5.6 that despite observing only 10% of the look-back window, the model can correctly infer both the complete look-back window and the horizon. Both quantitative and qual-

Table 5.4: MAE results for forecasting with missing values in the look-back window. τ stands for the percentage of observed values in the look-back window. Best results are in bold. TimeFlow improvement represents the overall percentage improvement (for each task) achieved by TimeFlow compared to the specific method being considered.

			TimeFlow		DeepTime		Neural Process	
	H	τ	Imputation error	Forecast error	Imputation error	Forecast error	Imputation error	Forecast error
Electricity	96	0.5	**0.151 ± 0.003**	**0.239 ± 0.013**	0.209 ± 0.004	0.270 ± 0.019	0.460 ± 0.048	0.486 ± 0.078
		0.2	**0.208 ± 0.006**	**0.260 ± 0.015**	0.249 ± 0.006	0.296 ± 0.023	0.644 ± 0.079	0.650 ± 0.095
		0.1	**0.272 ± 0.006**	**0.295 ± 0.016**	0.284 ± 0.007	0.324 ± 0.026	0.740 ± 0.083	0.737 ± 0.106
	192	0.5	**0.149 ± 0.004**	**0.235 ± 0.011**	0.204 ± 0.004	0.265 ± 0.018	0.461 ± 0.045	0.498 ± 0.070
		0.2	**0.209 ± 0.006**	**0.257 ± 0.013**	0.244 ± 0.007	0.290 ± 0.023	0.601 ± 0.075	0.626 ± 0.101
		0.1	**0.274 ± 0.010**	**0.289 ± 0.016**	0.282 ± 0.007	0.315 ± 0.025	0.461 ± 0.045	0.724 ± 0.090
Traffic	96	0.5	**0.180 ± 0.016**	**0.219 ± 0.026**	0.272 ± 0.028	0.243 ± 0.030	0.436 ± 0.025	0.444 ± 0.047
		0.2	**0.239 ± 0.019**	**0.243 ± 0.027**	0.335 ± 0.026	0.293 ± 0.027	0.596 ± 0.049	0.597 ± 0.075
		0.1	**0.312 ± 0.020**	**0.290 ± 0.027**	0.385 ± 0.025	0.344 ± 0.027	0.734 ± 0.102	0.731 ± 0.132
	192	0.5	**0.176 ± 0.014**	**0.217 ± 0.017**	0.241 ± 0.027	0.234 ± 0.021	0.477 ± 0.042	0.476 ± 0.043
		0.2	**0.233 ± 0.017**	**0.236 ± 0.021**	0.286 ± 0.027	0.276 ± 0.020	0.685 ± 0.109	0.678 ± 0.108
		0.1	**0.304 ± 0.019**	**0.277 ± 0.021**	0.331 ± 0.025	0.324 ± 0.021	0.888 ± 0.178	0.877 ± 0.174
TimeFlow improvement			/	/	22.7 %	12.0 %	62.3 %	59.4 %

itative results show the robustness and efficiency of TimeFlow on this particularly challenging setting.

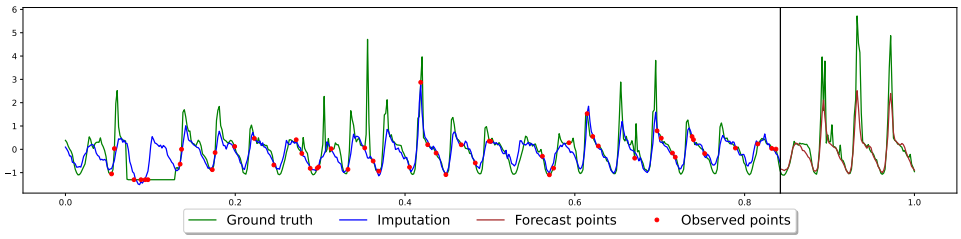


Figure 5.6: *Traffic dataset, sample 95.* In this figure, TimeFlow simultaneously imputes and forecasts at horizon 96 with a 10% partially observed look-back window of length 512.

5.2.4.4 Conclusion and Discussion

We have introduced a unified framework for continuous time series modeling leveraging conditional INR and meta-learning. Our experiments have demonstrated superior performance compared to other continuous methods, and better or comparable results to SOTA discrete methods. One of the standout features of our framework is its inherent continuity and the ability to modulate the INR parameters. This unique flexibility lets TimeFlow effectively tackle a wide array of challenges, including forecasting in the presence of missing values, accommodating irregular time steps, and extending the trained model's applicability to previously unseen time

series and new time windows. Our empirical results have shown TimeFlow's effectiveness in handling homogeneous multivariate time series. As a logical next step, extending TimeFlow's capabilities to address heterogeneous multivariate phenomena represents a promising direction for future research.

Chapter 6

Conclusion

6.1 Conclusion

In this thesis, we have addressed several challenges at the intersection of numerical analysis and deep learning. This research was conducted through three angles addressing various challenges, from the design of a friction law in the Shallow Water equations to the forecasting of time series.
The first angle focused on incorporating numerical analysis concepts into deep learning frameworks, showing promising results. By incorporating multigrid numerical schemes into a multi-scale deep learning architecture, namely the Multipole Graph Neural Operator, this work demonstrated its efficacy in solving static PDE problems. Notably, it successfully tackled steady-state Darcy flow and the 1D viscous unsteady Burgers' equation, highlighting the potential of numerical integration within deep learning for solving partial differential equations. Furthermore, the integration of implicit numerical schemes into neural networks, with added constraints, led to a stable neural network in forecasting. This work showed improved long-term forecasting in two transport PDEs. This innovative approach represents a significant step towards incorporating the strengths of numerical analysis into deep learning algorithms.
The second angle of this thesis centered on designing a hybrid model to address the friction law design in Shallow-Water equations, which is a current challenge among the numerical analysis community. This hybrid model consists of a data-driven friction law integrated into a numerical scheme, thus learning the friction from the observations, namely the water height and flow. This framework is robust to various cases, as shown in the experiments. Indeed, by learning through a solver, when confronted with more realistic data, the DL component manages to compensate for diffusion or discretization errors. Moreover, this approach was shown to be effective on the complex PDE dynamic case as well. This approach illustrates the benefits of combining the strengths of DL by learning from data and numerical analysis by modeling accurately physical systems.

The third and final angle revolved around the development of continuous modeling, with two distinct applications demonstrating the versatility of this approach. The first application addressed airfoil optimization, a critical concern in aerospace engineering. Leveraging Implicit Neural Representations (INRs), we designed an efficient surrogate model for the prediction of physical fields throughout airfoil volumes and surfaces. The ability to predict drag and lift coefficients while adhering to the governing equations offers great potential for optimizing aerodynamic designs. In the second application, the focus shifted to time series modeling, with INRs being employed to handle missing data, irregular sampling, and unaligned observations from multiple sensors. The method demonstrated state-of-the-art performance and generalization abilities, showcasing its adaptability to a wide range of time-series data problems.
In summary, this thesis contributed to the field of numerical analsys and deep learning, showing links in various directions between the two domains. It highlighted the possibilities and advantages of applying numerical analysis concepts into deep learning frameworks, hybrid modeling and continuous modeling. This research has provided solutions to diverse challenges across various domains, from solving dynamic PDEs to optimizing aerodynamic designs and handling time series data. These findings not only contribute to the advancement of scientific knowledge but also have practical implications for industries ranging from aerospace to environmental science. The promising results from this work opens the way for future works and further explorations, some of them being highlighted in the upcoming Section 6.2.

6.2 Perspectives

Following the works presented in this thesis, many directions can be taken to improve upon this work. This sections presents a few directions.

Improving convergence performances in forecasting dynamical systems. As shown in Section 3.2, we designed a method to ensure stability when forecasting dynamical systems. However, as the experiments illustrate, the method does not manage to converge to the solution, hence this solution is not yet very useful. We believe, given the good performances of FNO, that an implicit scheme inspired FNO, with constraints on its weights, could lead to better results. Some experiments have been conducted in this direction but due to a lack of time were not finished. These experiments led to better results but the theoretical results of stability in forecasting could not yet be guaranteed. Designing such a structural stable network in forecasting with good performances would be of great interest of the community in addition to the existing literature.

Hybrid modeling for the Shallow Water equations. In addition to the vast amount of experiments performed for the friction design in the Shallow Water equations, other extensions are possible. A main extension is to learn a friction law dependent on space. Indeed, in many practical applications, the friction depends on the space, for instance in the case of a river, the bottom of a river can change and thus change the friction. In order to learn such a friction law with a neural network, space must be added as an input. However, with the special nature of this input relative to the function, it must be treated differently than the water height and flow. First experiments were made using a network that modifies this input before concatenating it to the height and flow. This is an exciting direction for this research and could enhance its applicability in practical cases.
Another direction, which is more complex, is to study regime changes for the ODE case. Indeed, in the current experiments, the regime (subsonic or supersonic) is the same across the domain. However, the regime can be different across the domain, for instance subsonic on the left side and supersonic on the right side. This implies adding constraints on the ODE in order to be able to study this phenomenon. This change of regime leads to discontinuity for the ODE case and hence must be very challenging for the network to handle. A solution to circumvent this issue is to study a regime change in the PDE case at long-term, hence being close to the ODE case, which is the stationary state of the PDE. With this approach, it would possible to study this complex regime change, given that the PDE discretization must be precise in order to capture this phenomenon. Studying this is of great theoretical and practical interest and very challenging.
Changing the problem setting is also an interesting direction. It could be assumed that a prior on the law is learned and is incorporated directly into the neural network. This would implemented by adding guessed C_f, α and β as input of the neural network. Then, in order to improve the training, a continuation method/curriculum learning approach could be implemented. This could consist in training for parameters that correspond to easier laws and then increasing the parameters to tackle harder laws, as has been done in Krishnapriyan et al. (2021).

Continuous modeling for surrogate modeling Continuous modeling for surrogate modeling is very promising as shown with INFINITY in a challenging setting. In order to extend this work, the first direction would be to apply this approach on more datasets to test its robustness. A second direction that is in progress is to add a diffusion model as an encoder. This would allow to produce a probability distribution on the results in addition to the already existing continuity property and would be of great interest for practitioners. A third direction is to add priors into the model. Indeed, different settings of the AirfRANS dataset are designed to test the generalization properties of the approaches with respect to the angle of attack, the Reynolds number and scarcity of data. DL approaches and INFINITY would

benefit greatly from prior information such as constraining the network to respect the boundary conditions. Many implementations can follow this intuition, but we have not found yet any that would keep the efficiency of INFINITY.

Continuous modeling for time series TimeFlow has shown promising abilities to impute and forecast time series where a single phenomenon is measured at multiple locations over time. It would be interesting to extend this research to time series with multiple phenomena measured at multiple locations over time. A few experiments have been performed to test our framework in these settings, but the results were not really convincing, nor the data. We have tested this extension by having a latent code that modulates several INRs, each fitting a phenomenon. Other ideas can be extended in this direction, an interesting one would be to design a network that can really take advantage of the potential links between the several time series phenomena occurring at the same time. Another direction in this work is to study other tasks, in addition to imputation and forecasting. Some preliminary studies have been conducted on classification problems. An idea that we did not manage to successfully implement was to use the latent codes for classification. These codes could be very powerful since they are a compact representation of the time series. However, as such, they were not constrained enough to produce interesting performances for downstream tasks. We have tried to add some biases in the loss to constrain this space, for e.g. codes that represent time series that are close should be close, but none of them proved efficient. This direction is interesting and promising, the goal being to find a good way to constrain the latent space of the modulated INR.

Bibliography

A. Agresta, M. Baioletti, C. Biscarini, F. Caraffini, A. Milani, and V. Santucci. Using optimisation meta-heuristics for the roughness estimation problem in river flow analysis. *Applied Sciences*, 11(22):10575, 2021.

T. Alt, K. Schrader, M. Augustin, P. Peter, and J. Weickert. Connections between numerical algorithms for pdes and neural networks. *Journal of Mathematical Imaging and Vision*, 65(1):185–208, 2023.

A. Anandkumar, K. Azizzadenesheli, K. Bhattacharya, N. Kovachki, Z. Li, B. Liu, and A. Stuart. Neural operator: Graph kernel network for partial differential equations. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020. URL https://openreview.net/forum?id=fg2ZFmXFO3.

M. K. Askar and K. Al-Jumaily. A nonlinear optimization model for estimating manning's roughness coefficient. In *Twelfth International Water Technology Conference*, pages 1299–1306. Citeseer, 2008.

I. Ayed, E. de Bézenac, A. Pajot, J. Brajard, and P. Gallinari. Learning dynamical systems from partial observations. *arXiv preprint arXiv:1902.11136*, 2019.

M. T. Ayvaz. A linked simulation–optimization model for simultaneously estimating the manning's surface roughness values and their parameter structures in shallow water flows. *Journal of hydrology*, 500:183–199, 2013.

S. Bai, J. Z. Kolter, and V. Koltun. Deep equilibrium models. *Advances in Neural Information Processing Systems*, 32, 2019.

R. Balan, M. Singh, and D. Zou. Lipschitz properties for deep convolutional networks. *Contemporary Mathematics*, 706:129–151, 2018.

P. Bartlett, D. J. Foster, and M. Telgarsky. Spectrally-normalized margin bounds for neural networks. *arXiv preprint arXiv:1706.08498*, 2017.

M. Beck. A brief introduction to stability theory for linear pdes. In *SIAM conference on Nonlinear Waves and Coherent Structures*, 2012.

J. Behrmann, W. Grathwohl, R. T. Chen, D. Duvenaud, and J.-H. Jacobsen. Invertible residual networks. In *International Conference on Machine Learning*, pages 573–582. PMLR, 2019.

F. d. A. Belbute-Peres, T. D. Economon, and J. Z. Kolter. Combining Differentiable PDE Solvers and Graph Neural Networks for Fluid Flow Prediction. In *International Conference on Machine Learning (ICML)*, 2020.

M. Bergot and M. Duruflé. Higher-Order Discontinuous Galerkin Method for Pyramidal Elements using Orthogonal Bases. *Numerical Methods for Partial Differential Equations*, 29(1):144–169, Jan. 2013. doi: 10.1002/num.21703. URL https://hal.archives-ouvertes.fr/hal-00547319.

M. Bilos, K. Rasul, A. Schneider, Y. Nevmyvaka, and S. Günnemann. Modeling temporal data as continuous functions with stochastic process diffusion. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *International Conference on Machine Learning, ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 2452–2470. PMLR, 2023.

E. G. Birgin and J. M. Martínez. Accelerated derivative-free nonlinear least-squares applied to the estimation of manning coefficients. *Computational Optimization and Applications*, 81(3):689–715, 2022.

F. Bonassi, M. Farina, and R. Scattolini. On the stability properties of gated recurrent units neural networks. *arXiv preprint arXiv:2011.06806*, 2020.

F. Bonnet, J. A. Mazari, P. Cinnella, and P. Gallinari. Airfrans : High fidelity computational fluid dynamics dataset for approximating reynolds-averaged navier – stokes solutions. In *Neurips 2022*, 2022.

F. Bouchut. *Nonlinear stability of finite Volume Methods for hyperbolic conservation laws: And Well-Balanced schemes for sources*. Springer Science & Business Media, 2004.

O. Boussif, Y. Bengio, L. Benabbou, and D. Assouline. Magnet: Mesh agnostic neural pde solver. *Advances in Neural Information Processing Systems*, 35:31972–31985, 2022.

J. Brandstetter, D. E. Worrall, and M. Welling. Message passing neural PDE solvers. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=vSix3HPYKSU.

M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

E. D. Brouwer, J. Simm, A. Arany, and Y. Moreau. Gru-ode-bayes: continuous modeling of sporadically-observed time series. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 7379–7390, 2019.

S. L. Brunton, B. R. Noack, and P. Koumoutsakos. Machine learning for fluid mechanics. *Annual review of fluid mechanics*, 52:477–508, 2020.

L. Bungert, R. Raab, T. Roith, L. Schwinn, and D. Tenbrinck. Clip: Cheap lipschitz training of neural networks. *arXiv preprint arXiv:2103.12531*, 2021.

W. Cao, D. Wang, J. Li, H. Zhou, Y. Li, and L. Li. Brits: bidirectional recurrent imputation for time series. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 6776–6786, 2018.

N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. Ieee, 2017.

B. Chang, L. Meng, E. Haber, L. Ruthotto, D. Begert, and E. Holtham. Reversible architectures for arbitrarily deep residual neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

B. Chang, M. Chen, E. Haber, and E. H. Chi. Antisymmetricrnn: A dynamical system view on recurrent neural networks. *arXiv preprint arXiv:1902.09689*, 2019.

H. Chanson. Solutions analytiques de l'onde de rupture de barrage sur plan horizontal et incliné. *La Houille Blanche*, 92(3):76–86, 2006.

R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017. doi: 10.1109/CVPR.2017.16.

H. Chen, S. Grant-Muller, L. Mussone, and F. Montgomery. A study of hybrid neural network approaches and the effects of missing data on traffic forecasting. *Neural Computing & Applications*, 10:277–286, 2001.

L. Chen, L. Jin, and M. Shang. Zero stability well predicts performance of convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 6268–6277, 2022.

R. T. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018.

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

T. Chen and H. Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE transactions on neural networks*, 6(4):911–917, 1995.

M. Ciccone, M. Gallieri, J. Masci, C. Osendorfer, and F. Gomez. Nais-net: Stable deep networks from non-autonomous differential equations. *arXiv preprint arXiv:1804.07209*, 2018.

Y. G. Cinar, H. Mirisaee, P. Goswami, É. Gaussier, and A. Aït-Bachir. Period-aware content attention rnns for time series forecasting with missing values. *Neurocomputing*, 312:177–186, 2018.

J. S. Clark and O. N. Bjørnstad. Population time series: process variability, observation errors, missing values, lags, and hidden states. *Ecology*, 85(11):3140–3150, 2004.

G. Corani, A. Benavoli, and M. Zaffalon. Time series forecasting with gaussian processes needs priors. In *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part IV 21*, pages 103–117. Springer, 2021.

J. Cordazzo, L. Karpinski, S. Keller, A. F. Silva, and C. R. Maliska. Additive correction multigrid applied to petroleum reservoir simulation, 2007.

R. M. Cummings, W. H. Mason, S. A. Morton, and D. R. McDaniel. *Applied computational aerodynamics: A modern engineering approach*, volume 53. Cambridge University Press, 2015.

E. De Bézenac, A. Pajot, and P. Gallinari. Deep learning for physical processes: Incorporating prior scientific knowledge. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124009, 2019.

O. Delestre. *Simulation du ruissellement d'eau de pluie sur des surfaces agricoles.* PhD thesis, Université d'Orléans; Université d'Orléans, 2010.

J. Derber and F. Bouttier. A reformulation of the background error covariance in the ecmwf global data assimilation system. *Tellus A: Dynamic Meteorology and Oceanography*, 51(2):195–221, 1999.

Y. Ding and S. S. Wang. Identification of manning's roughness coefficients in channel network using adjoint analysis. *International Journal of Computational Fluid Dynamics*, 19(1):3–13, 2005.

Y. Ding, Y. Jia, and S. S. Wang. Identification of manning's roughness coefficients in shallow water flows. *Journal of Hydraulic Engineering*, 130(6):501–510, 2004.

N. Drenkow, N. Sani, I. Shpitser, and M. Unberath. A systematic review of robustness in deep learning for computer vision: Mind the gap? *arXiv preprint arXiv:2112.00639*, 2021.

Q. Du, Y. Gu, H. Yang, and C. Zhou. The discovery of dynamics via linear multistep methods and deep learning: error estimation. *SIAM Journal on Numerical Analysis*, 60(4):2014–2045, 2022.

W. Du, D. Côté, and Y. Liu. Saits: Self-attention-based imputation for time series. *Expert Systems with Applications*, 219:119619, 2023.

D. Dunkerley. Surface tension and friction coefficients in shallow, laminar overland flows through organic litter. *Earth Surface Processes and Landforms: The Journal of the British Geomorphological Research Group*, 27(1):45–58, 2002.

E. Dupont, A. Doucet, and Y. W. Teh. Augmented neural odes. *arXiv preprint arXiv:1904.01681*, 2019.

E. Dupont, H. Kim, S. Eslami, D. Rezende, and D. Rosenbaum. From data to functa: Your data point is a function and you can treat it like one. *arXiv preprint arXiv:2201.12204*, 2022a.

E. Dupont, H. Kim, S. M. A. Eslami, D. J. Rezende, and D. Rosenbaum. From data to functa: Your data point is a function and you can treat it like one. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 5694–5725. PMLR, 2022b.

L. El Ghaoui, F. Gu, B. Travacca, A. Askari, and A. Y. Tsai. Implicit deep learning. *arXiv preprint arXiv:1908.06315*, 2, 2019.

R. Eymard, T. Gallouët, and R. Herbin. Finite volume methods. *Handbook of numerical analysis*, 7:713–1018, 2000.

R. Fathony, A. K. Sahu, D. Willmott, and J. Z. Kolter. Multiplicative filter networks. In *International Conference on Learning Representations*, 2021.

M. Fazlyab, A. Robey, H. Hassani, M. Morari, and G. J. Pappas. Efficient and accurate estimation of lipschitz constants for deep neural networks. *arXiv preprint arXiv:1906.04893*, 2019.

F. R. Fiedler and J. A. Ramirez. A numerical method for simulating discontinuous shallow flow over an infiltrating surface. *International journal for numerical methods in fluids*, 32(2):219–239, 2000.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

A. Flamant. *Mécanique appliquée: hydraulique*. Baudry, 1891.

E. Fons, A. Sztrajman, Y. El-Laham, A. Iosifidis, and S. Vyetrenko. Hypertime: Implicit neural representation for time series. *CoRR*, abs/2208.05836, 2022.

V. Fortuin, D. Baranchuk, G. Rätsch, and S. Mandt. Gp-vae: Deep probabilistic time series imputation. In *International conference on artificial intelligence and statistics*, pages 1651–1661. PMLR, 2020.

S. Fotiadis, E. Pignatelli, M. L. Valencia, C. Cantwell, A. Storkey, and A. A. Bharath. Comparing recurrent and convolutional neural networks for predicting wave propagation. *arXiv preprint arXiv:2002.08981*, 2020.

R. H. French. *Open-channel hydraulics*. McGraw-Hill New York, 1985.

H. Gao and S. Ji. Graph u-nets. In *international conference on machine learning*, pages 2083–2092. PMLR, 2019.

M. Garnelo, D. Rosenbaum, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. J. Rezende, and S. M. A. Eslami. Conditional neural processes. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, volume 80, pages 1690–1699. PMLR, 2018.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

H. Gouk, E. Frank, B. Pfahringer, and M. J. Cree. Regularisation of neural networks by enforcing lipschitz continuity. *Machine Learning*, 110:393–416, 2021.

S. Greydanus, M. Dzamba, and J. Yosinski. Hamiltonian neural networks. *Advances in neural information processing systems*, 32, 2019.

C. Grossmann, H.-G. Roos, and M. Stynes. *Numerical treatment of partial differential equations*, volume 154. Springer, 2007.

B. Güler, A. Laignelet, and P. Parpas. Towards robust and stable deep learning algorithms for forward backward stochastic differential equations. *arXiv preprint arXiv:1910.11623*, 2019.

G. Gupta, X. Xiao, and P. Bogdan. Multiwavelet-based operator learning for differential equations. *Advances in neural information processing systems*, 34:24048–24062, 2021.

E. Haber and L. Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, 2017.

E. Haber, K. Lensink, E. Treister, and L. Ruthotto. Imexnet a forward stable deep neural network. In *International Conference on Machine Learning*, pages 2525–2534. PMLR, 2019.

E. Hairer and A. Abdulle. *Introduction à l'analyse numérique*. Université de Genève-Section de mathématiques, 2001.

W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. URL http://arxiv.org/abs/1502.01852.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

D. Hendrycks and T. Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.

J.-M. Hervouet. *Hydrodynamique des écoulements à surface libre: Modélisation numérique avec la méthode des éléments finis*. Presses de l'école nationale des Ponts et Chaussées, 2003.

P. Holl, V. Koltun, and N. Thuerey. Learning to control pdes with differentiable physics. *CoRR*, abs/2001.07457, 2020. URL https://arxiv.org/abs/2001.07457.

P. Holl, V. Koltun, and N. Thuerey. Physical gradients for deep learning. *CoRR*, abs/2109.15048, 2021. URL https://arxiv.org/abs/2109.15048.

M. Horie and N. Mitsume. Physics-embedded neural networks: E (n)-equivariant graph neural pde solvers. *arXiv preprint arXiv:2205.11912*, 2022.

L. Huang and T. Hoefler. Compressing multidimensional weather and climate data into neural networks. In *International Conference on Learning Representations, ICLR*, 2023.

P. Jaysaval, D. V. Shantsev, S. de la Kethulle de Ryhove, and T. Bratteland. Fully anisotropic 3-D EM modelling on a Lebedev grid with a multigrid pre-conditioner. *Geophysical Journal International*, 207(3):1554–1572, 09 2016. ISSN 0956-540X. doi: 10.1093/gji/ggw352. URL https://doi.org/10.1093/gji/ggw352.

K. Jeong and Y. Shin. Time-series anomaly detection with implicit neural representation. *CoRR*, abs/2201.11950, 2022.

V. John. Multigrid methods. Technical report, Technical report, 2013.

M. Jordan and A. G. Dimakis. Exactly computing the local lipschitz constant of relu networks. *Advances in Neural Information Processing Systems*, 33:7344–7353, 2020.

S. Kaltenbach and P.-S. Koutsourelakis. Physics-aware, probabilistic model order reduction with guaranteed stability. *arXiv preprint arXiv:2101.05834*, 2021.

M. F. Kasim and S. M. Vinko. xi-torch: differentiable scientific computing library. *arXiv preprint arXiv:2010.01921*, 2020.

R. T. Keller and Q. Du. Discovery of dynamics using linear multistep methods. *SIAM Journal on Numerical Analysis*, 59(1):429–455, 2021.

J. J. Kennedy, N. Rayner, C. Atkinson, and R. Killick. An ensemble data set of sea surface temperature change from 1850: The met office hadley centre hadsst. 4.0. 0.0 data set. *Journal of Geophysical Research: Atmospheres*, 124(14):7719–7763, 2019.

B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, and B. Solenthaler. Deep fluids: A generative network for parameterized fluid simulations. In *Computer Graphics Forum*, volume 38, pages 59–70. Wiley Online Library, 2019a.

B. Kim, B. Chudomelka, J. Park, J. Kang, Y. Hong, and H. J. Kim. Robust neural networks inspired by strong stability preserving runge-kutta methods. In *European Conference on Computer Vision*, pages 416–432. Springer, 2020.

B. S. Kim, B. G. Kang, S. H. Choi, and T. G. Kim. Data modeling versus simulation modeling in the big data era: case study of a greenhouse control system. *Simulation*, 93(7):579–594, 2017.

H. Kim, G. Papamakarios, and A. Mnih. The lipschitz constant of self-attention. In *International Conference on Machine Learning*, pages 5562–5571. PMLR, 2021.

T. Kim, W. Ko, and J. Kim. Analysis and impact evaluation of missing data imputation in day-ahead pv generation forecasting. *Applied Sciences*, 9(1):204, 2019b.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

L. Kong, J. Sun, and C. Zhang. Sde-net: Equipping deep neural networks with uncertainty estimates. *arXiv preprint arXiv:2008.10546*, 2020.

N. Kovachki, S. Lanthaler, and S. Mishra. On universal approximation and error bounds for fourier neural operators. *The Journal of Machine Learning Research*, 22(1):13237–13312, 2021a.

N. B. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. M. Stuart, and A. Anandkumar. Neural operator: Learning maps between function spaces. *CoRR*, abs/2108.08481, 2021b. URL https://arxiv.org/abs/2108.08481.

A. S. Krishnapriyan, A. Gholami, S. Zhe, R. M. Kirby, and M. W. Mahoney. Characterizing possible failure modes in physics-informed neural networks. *arXiv preprint arXiv:2109.01050*, 2021.

J. N. Kutz, S. L. Brunton, B. W. Brunton, and J. L. Proctor. *Dynamic mode decomposition: data-driven modeling of complex systems*. SIAM, 2016.

F. Latorre, P. Rolland, and V. Cevher. Lipschitz constant estimation of neural networks via sparse polynomial optimization. *arXiv preprint arXiv:2004.08688*, 2020.

D. Lawrence. Macroscale surface roughness and frictional resistance in overland flow. *Earth Surface Processes and Landforms: The Journal of the British Geomorphological Group*, 22(4):365–382, 1997.

N. P. Lawrence, P. D. Loewen, M. G. Forbes, J. U. Backström, and R. B. Gopaluni. Almost surely stable deep dynamics. *arXiv preprint arXiv:2103.14722*, 2021.

Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989.

G. Legendre. Introduction à l'analyse numérique et au calcul scientifique. *Dauphine université de Paris*, 2018.

K. Leino, Z. Wang, and M. Fredrikson. Globally-robust neural networks. In *International Conference on Machine Learning*, pages 6212–6222. PMLR, 2021.

M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

M. Li, L. He, and Z. Lin. Implicit euler skip connections: Enhancing adversarial robustness via numerical stability. In *International Conference on Machine Learning*, pages 5874–5883. PMLR, 2020a.

Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020b.

Z. Li, D. Z. Huang, B. Liu, and A. Anandkumar. Fourier neural operator with learned deformations for pdes on general geometries. *arXiv preprint arXiv:2207.05209*, 2022.

Z.-Y. Li, N. B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Multipole graph neural operator for parametric partial differential equations. *ArXiv*, abs/2006.09535, 2020c.

K.-A. Lie, O. Møyner, and J. R. Natvig. Use of Multiple Multiscale Operators To Accelerate Simulation of Complex Geomodels. *SPE Journal*, 22(06):1929–1945, 08 2017. ISSN 1086-055X. doi: 10.2118/182701-PA. URL https://doi.org/10.2118/182701-PA.

M. Lino, C. Cantwell, S. Fotiadis, E. Pignatelli, and A. Bharath. Simulating surface wave dynamics with convolutional networks. *arXiv preprint arXiv:2012.00718*, 2020.

S. Liu, H. Yu, C. Liao, J. Li, W. Lin, A. X. Liu, and S. Dustdar. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, 2022.

S. Liu, X. Li, G. Cong, Y. Chen, and Y. Jiang. Multivariate time-series imputation with disentangled temporal representations. In *The Eleventh International Conference on Learning Representations, ICLR*, 2023.

Y. Liu, R. Yu, S. Zheng, E. Zhan, and Y. Yue. Naomi: Non-autoregressive multiresolution sequence imputation. *Advances in neural information processing systems*, 32, 2019.

S. Lohr. The age of big data. *New York Times*, 11(2012), 2012.

L. Lu, P. Jin, and G. E. Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

L. Lu, X. Meng, S. Cai, Z. Mao, S. Goswami, Z. Zhang, and G. E. Karniadakis. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, 2022.

Y. Luo, X. Cai, Y. Zhang, J. Xu, et al. Multivariate time series imputation with generative adversarial networks. *Advances in neural information processing systems*, 31, 2018.

Y. Luo, Y. Zhang, X. Cai, and X. Yuan. E2gan: End-to-end generative adversarial network for multivariate time series imputation. In *Proceedings of the 28th international joint conference on artificial intelligence*, pages 3094–3100. AAAI Press, 2019.

G. Madec, R. Bourdallé-Badie, P.-A. Bouttier, C. Bricaud, D. Bruciaferri, D. Calvert, J. Chanut, E. Clementi, A. Coward, D. Delrosso, et al. Nemo ocean engine. *Notes du Pôle de modélisation de l'Institut Pierre-Simon Laplace (IPSL): (27). ISSN 1288-1619*, 2017.

G. Mamakoukas, O. Xherija, and T. Murphey. Memory-efficient learning of stable linear dynamical systems for prediction and control. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

G. Manek and J. Z. Kolter. Learning stable deep dynamics models. *arXiv preprint arXiv:2001.06116*, 2020.

W. A. Marcus, K. Roberts, L. Harvey, and G. Tackman. An evaluation of methods for estimating manning's n in small mountain streams. *Mountain Research and Development*, pages 227–239, 1992.

S. Massaroli, M. Poli, M. Bin, J. Park, A. Yamashita, and H. Asama. Stable neural flows. *arXiv preprint arXiv:2003.08063*, 2020.

R. Matusik, A. Nowakowski, S. Plaskacz, and A. Rogowski. Finite-time stability for differential inclusions with applications to neural networks. *SIAM Journal on Control and Optimization*, 58(5):2854–2870, 2020.

L. Migus, Y. Yin, J. A. Mazari, and P. Gallinari. Multi-scale physical representations for approximating pde solutions with graph neural operators. *arXiv preprint arXiv:2206.14687*, 2022.

L. Migus, J. Salomon, and P. Gallinari. Stability of implicit neural networks for long-term forecasting in dynamical systems. *arXiv preprint arXiv:2305.17155*, 2023.

B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

J. Miller and M. Hardt. Stable recurrent models. *arXiv preprint arXiv:1805.10369*, 2018.

A. T. Mohan, N. Lubbers, D. Livescu, and M. Chertkov. Embedding hard physical constraints in convolutional neural networks for 3d turbulence. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020. URL https://openreview.net/forum?id=IaXBtMNFaa.

L. Montabone, K. Marsh, S. Lewis, P. Read, M. Smith, J. Holmes, A. Spiga, D. Lowe, and A. Pamment. The mars analysis correction data assimilation (macda) dataset v1. 0. *Geoscience Data Journal*, 1(2):129–139, 2014.

E. L. Naour, L. Serrano, L. Migus, Y. Yin, G. Agoua, N. Baskiotis, V. Guigue, et al. Time series continuous modeling for imputation and forecasting with implicit neural representations. *arXiv preprint arXiv:2306.05880*, 2023.

A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms, 2018a.

A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018b.

Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. *CoRR*, abs/2211.14730, 2022.

M. Nonnenmacher and D. S. Greenberg. Learning implicit pde integration with linear implicit layers. In *The Symbiosis of Deep Learning and Differential Equations*, 2021.

O. Obiols-Sales, A. Vishnu, N. Malaya, and A. Chandramowliswharan. Cfdnet: A deep learning-based accelerator for fluid simulations. In *Proceedings of the 34th ACM international conference on supercomputing*, pages 1–12, 2020.

B. O'Malley, J. Kópházi, M. Eaton, V. Badalassi, P. Warner, and A. Copestake. Pyramid finite elements for discontinuous and continuous discretizations of the neutron diffusion equation with applications to reactor physics. *Progress in Nuclear Energy*, 105:175–184, 2018. ISSN 0149-1970. doi: https://doi.org/10.

1016/j.pnucene.2017.12.006. URL https://www.sciencedirect.com/science/article/pii/S0149197017303062.

J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019.

J.-C. Passieux, P. Ladevèze, and D. Néron. A scalable time-space multiscale domain decomposition method: adaptive time scales separation. *Computational Mechanics*, 46(4):621–633, 2010. doi: 10.1007/s00466-010-0504-2. URL https://hal.archives-ouvertes.fr/hal-00485747.

P. Pauli, A. Koch, J. Berberich, P. Kohler, and F. Allgöwer. Training robust neural networks using lipschitz bounds. *IEEE Control Systems Letters*, 6:121–126, 2021.

G. Peano and G. Peano. *Démonstration de l'intégrabilité des équations différentielles ordinaires*. Springer, 1990.

E. Perez, F. Strub, H. D. Vries, V. Dumoulin, and A. Courville. Film: Visual reasoning with a general conditioning layer. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 2018. ISSN 2159-5399. doi: 10.1609/aaai.v32i1.11671.

T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021.

D. A. Raff and J. A. Ramírez. A physical, mechanistic and fully coupled hillslope hydrology model. *International journal for numerical methods in fluids*, 49(11): 1193–1212, 2005.

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

A. Raj, Y. Bresler, and B. Li. Improving robustness of deep-learning-based image reconstruction. In *International Conference on Machine Learning*, pages 7932–7942. PMLR, 2020.

C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, 2006.

J. N. Reddy. *Introduction to the finite element method*. McGraw-Hill Education, 2019.

V. Reshniak and C. G. Webster. Robust learning with implicit residual networks. *Machine Learning and Knowledge Extraction*, 3(1):34–55, 2021.

M. Revay and I. Manchester. Contracting implicit recurrent neural networks: Stable models with improved trainability. In *Learning for Dynamics and Control*, pages 393–403. PMLR, 2020.

O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

K. Rothauge, Z. Yao, Z. Hu, and M. W. Mahoney. Residual networks as nonlinear systems: Stability analysis using linearization. *arXiv preprint arXiv:1905.13386*, 2019.

Y. Rubanova, R. T. Q. Chen, and D. Duvenaud. Latent odes for irregularly-sampled time series. *CoRR*, abs/1907.03907, 2019.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

L. Ruthotto and E. Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, pages 1–13, 2019.

P. Sagaut, M. Terracol, and S. Deck. *Multiscale and multiresolution approaches in turbulence-LES, DES and Hybrid RANS/LES Methods: Applications and Guidelines*. World Scientific, 2013.

A. d. Saint-Venant et al. Theorie du mouvement non permanent des eaux, avec application aux crues des rivieres et a l'introduction de marees dans leurs lits. *Comptes rendus des seances de l'Academie des Sciences*, 36:174–154, 1871.

K. Scaman and A. Virmaux. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *arXiv preprint arXiv:1805.10965*, 2018.

M. Schulz and K. Stattegger. Spectrum: Spectral analysis of unevenly spaced paleoclimatic time series. *Computers & Geosciences*, 23(9):929–945, 1997.

H. Schweiger, C. Fabris, G. Ausweger, and L. Hauser. Examples of successful numerical modelling of complex geotechnical problems. *Innovative Infrastructure Solutions*, 4:1–10, 2019.

L. Serrano, L. L. Boudec, A. K. Koupaï, T. X. Wang, Y. Yin, J.-N. Vittaut, and P. Gallinari. Operator learning with neural fields: Tackling pdes on general geometries. *arXiv preprint arXiv:2306.07266*, 2023a.

L. Serrano, L. Migus, Y. Yin, J. A. Mazari, and P. Gallinari. Infinity: Neural field modeling for reynolds-averaged navier-stokes equations. *arXiv preprint arXiv:2307.13538*, 2023b.

J. Shen, Z. Li, L. Yu, G.-S. Xia, and W. Yang. Implicit euler ode networks for single-image dehazing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 218–219, 2020.

S. N. Shukla and B. M. Marlin. Multi-time attention networks for irregularly sampled time series. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

J. Sirignano and K. Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020a.

V. Sitzmann, J. N. P. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020b.

M. W. Smith, N. J. Cox, and L. J. Bracken. Applying flow resistance equations to overland flows. *Progress in Physical Geography*, 31(4):363–387, 2007.

W. E. Sorteberg, S. Garasto, C. C. Cantwell, and A. A. Bharath. Approximating the solution of surface wave propagation using deep neural networks. In *INNS Big Data and Deep Learning conference*, pages 246–256. Springer, 2019.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

M. Tancik, P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33:7537–7547, 2020a.

M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 2020-December, 2020b. ISSN 10495258.

X. Tang, H. Yao, Y. Sun, C. C. Aggarwal, P. Mitra, and S. Wang. Joint modeling of local and global temporal dynamics for multivariate time series forecasting with missing values. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI*, pages 5956–5963. AAAI Press, 2020.

Y. Tashiro, J. Song, Y. Song, and S. Ermon. Csdi: Conditional score-based diffusion models for probabilistic time series imputation. *Advances in Neural Information Processing Systems*, 34:24804–24816, 2021.

M. Tekriwal, K. Duraisamy, and J.-B. Jeannin. A formal proof of the lax equivalence theorem for finite difference schemes. In *NASA Formal Methods Symposium*, pages 322–339. Springer, 2021.

N. Thuerey, K. Weißenow, L. Prantl, and X. Hu. Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows. *AIAA Journal*, 58(1):25–36, 2020.

N. Thuerey, P. Holl, M. Müller, P. Schnell, F. Trost, and K. Um. Physics-based deep learning. *CoRR*, abs/2109.05237, 2021. URL https://arxiv.org/abs/2109.05237.

A. Tran, A. Mathews, L. Xie, and C. S. Ong. Factorized fourier neural operators. *arXiv preprint arXiv:2111.13802*, 2021.

A. Tuor, J. Drgona, and D. Vrabie. Constrained neural ordinary differential equations with stability guarantees. *arXiv preprint arXiv:2004.10883*, 2020.

K. Um, R. Brand, Y. R. Fei, P. Holl, and N. Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6111–6122. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/43e4e6a6f341e00671e123714de019a8-Paper.pdf.

B. van de Rotten and S. V. Lunel. A limited memory broyden method to solve high-dimensional systems of nonlinear equations. In *EQUADIFF 2003*, pages 196–201. World Scientific, 2005.

C. Van der Sande, S. De Jong, and A. De Roo. A segmentation and classification approach of ikonos-2 imagery for land cover mapping to assist flood risk and flood damage assessment. *International Journal of applied earth observation and geoinformation*, 4(3):217–229, 2003.

P.-L. Viollet, J.-P. Chabard, and P. Esposito. *Mécanique des fluides appliquée: écoulements incompressibles dans les circuits, canaux et rivières, autour des structures et dans l'environnement*. Presses des Ponts, 2003.

N. Wandel, M. Weinmann, and R. Klein. Learning incompressible fluid dynamics from scratch–towards fast, differentiable fluid models that generalize. *arXiv preprint arXiv:2006.08762*, 2020.

N. Wandel, M. Weinmann, and R. Klein. Learning incompressible fluid dynamics from scratch - towards fast, differentiable fluid models that generalize. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=KUDUoRsEphu.

R. Wang, K. Kashinath, M. Mustafa, A. Albert, and R. Yu. Towards physics-informed deep learning for turbulent flow prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1457–1466, 2020.

L. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, L. Daniel, D. Boning, and I. Dhillon. Towards fast computation of certified robustness for relu networks. In *International Conference on Machine Learning*, pages 5276–5285. PMLR, 2018.

S. Wiewel, M. Becher, and N. Thuerey. Latent space physics: Towards learning the temporal evolution of fluid flow. In *Computer graphics forum*, volume 38, pages 71–82. Wiley Online Library, 2019.

G. Woo, C. Liu, D. Sahoo, A. Kumar, and S. C. H. Hoi. Deeptime: Deep time-index meta-learning for non-stationary time-series forecasting. *CoRR*, abs/2207.06046, 2022.

H. Wu, J. Xu, J. Wang, and M. Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 22419–22430, 2021.

J. Xu, A. Pradhan, and K. Duraisamy. Conditionally parameterized, discretization-aware neural networks for mesh-based modeling of physical systems. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=0yMGEUQKd2D.

Y. Yin, V. Le Guen, J. Dona, E. de Bezenac, I. Ayed, N. Thome, and P. Gallinari. Augmenting physical models with deep networks for complex dynamics forecast-

ing. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124012, 2021.

Y. Yin, M. Kirchmeyer, J.-Y. Franceschi, A. Rakotomamonjy, and P. Gallinari. Continuous pde dynamics forecasting with implicit neural representations. In *International Conference on Learning Representations, ICLR*, 2023.

A. Zeng, M. Chen, L. Zhang, and Q. Xu. Are transformers effective for time series forecasting? *CoRR*, abs/2205.13504, 2022.

B. Zhang, D. Jiang, D. He, and L. Wang. Rethinking lipschitz neural networks and certified robustness: A boolean function perspective. *Advances in Neural Information Processing Systems*, 35:19398–19413, 2022a.

H. Zhang, D. Yu, M. Yi, W. Chen, and T.-Y. Liu. Stabilize deep resnet with a sharp scaling factor τ. *Machine Learning*, 111(9):3359–3392, 2022b.

L. Zhang and H. Schaeffer. Forward stability of resnet and its variants. *Journal of Mathematical Imaging and Vision*, 62(3):328–351, 2020.

S. Zheng, Y. Song, T. Leung, and I. Goodfellow. Improving the robustness of deep neural networks via stability training. In *Proceedings of the ieee conference on computer vision and pattern recognition*, pages 4480–4488, 2016.

H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, pages 11106–11115, 2021.

T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 27268–27286. PMLR, 2022.

L. Zintgraf, K. Shiarli, V. Kurin, K. Hofmann, and S. Whiteson. Fast context adaptation via meta-learning. In *International Conference on Machine Learning*, pages 7693–7702. PMLR, 2019a.

L. Zintgraf, K. Shiarli, V. Kurin, K. Hofmann, and S. Whiteson. Fast context adaptation via meta-learning. In *International Conference on Machine Learning*, pages 7693–7702. PMLR, 2019b.

D. Zou, R. Balan, and M. Singh. On lipschitz bounds of general convolutional neural networks. *IEEE Transactions on Information Theory*, 66(3):1738–1759, 2019.

Appendix A

Appendix of Chapter 3

A.1 Stability of implicit neural networks for long-term forecasting in dynamical systems

A.1.1 Details on the implementation

Our implicit neural network is using Rectified linear unit (ReLU) activation functions, as can be seen in Figure 3.3.

Definition A.1.1 (ReLU)**.** *A rectified linear unit (ReLU) function is defined componentwise to a vector by* $\forall\ x \in \mathbb{R}, ReLU(x) = \max(0, x)$.

It is one of the most common activation functions used in Deep Learning.

In order to constrain our network, we use upper triangular weights W_n. At each training epoch, we constrain the diagonal values to be between -1 and 0 after gradient descent. We choose a minimal value of 0.01, to ensure that the theorem hypothesis are respected. For values below -1, we set them to -1 and for values above 0.01, we set them to 0.01.

A.1.2 Details on experiments

Baseline Methods

In addition to an Explicit ResNet with ReLU activation function and a FNO, we have two variants for the explicit ResNet method.

- an explicit ResNet tanh, with $R_n(x) = \tanh(W_n x + b_n)$
- an explicit ResNet BN, with a ReLU activation function, and batch normalization at each hidden layer, to control the norm inside the network.

On training

The training details for each architecture on both equations are presented in the appendix in Table A.1. For the *Advection equation*, 350 examples were generated for train, 150 other ones for validation/test and 50 for forecasting tests. For *Burgers' equation*, 120 examples were generated for train, 30 other ones for validation/test and 50 for forecasting tests. Our experiments led to a few main training remarks.

Initialization The networks are not really sensitive to the initialization. The only pitfall is to initialize with high values. Then the network doesn't manage to converge as well as it could have. We initialize all networks with Xavier initialization with a gain of 1.

Learning rate scheduling We used learning rate scheduling. It improves performance by a factor of 100. It is crucial to use it for our problems, and to choose carefully its parameter. We found that a linear scheduling with carefully chosen decay and step size works well. A special attention needs to be placed on the initial learning rate as well.

FNO architecture For the FNO network, we chose 12 modes a width of 32 for the *Advection equation* and 16 modes and a width of 64 for *Burgers' equation*, as was done in the original article.

Training parameters

The training remarks detailed previously in section A.1.2 led to the choices showed in Table A.1.

Table A.1: Hyper-parameter choice for each architecture on the *Advection equation* and *Burgers' equation*.

	Model	Xavier gain	Initial learning rate	Decay	Step size	Epochs
Advection	Explicit Res Net	1	0.05	0.95	10	2500
	Explicit Res Net BN	1	0.05	0.98	10	2500
	Explicit Res Tanh	1	0.05	0.95	10	2500
	FNO	1	0.005	0.98	10	2500
	Implicit ResNet (Ours)	1	0.01	0.9	10	1250
Burgers'	Explicit Res Net	1	0.05	0.95	10	2500
	Explicit Res Net BN	1	0.05	0.98	10	2500
	Explicit Res Tanh	1	0.05	0.95	10	2500
	FNO	1	0.005	0.96	10	2500
	Implicit ResNet (Ours)	1	0.01	0.98	10	1250

Ablation study

In order to better investigate this task, we conducted experiments with additional architectures. The results are shown in Table A.2.

Table A.2: Ablation study for the *Advection equation* and *Burgers' equation*.

	Model	Train Error ($\times 10^{-4}$)	Test Error ($\times 10^{-4}$)	Forecast error at mid-range $T_{adv} = 40 \cdot \Delta t_{adv}$ $T_{bur} = 150 \cdot \Delta t_{bur}$	Forecast error at long-range $T_{adv} = 400 \cdot \Delta t_{adv}$ $T_{bur} = 300 \cdot \Delta t_{bur}$
Advection	Explicit Res Net	**0.03 ± 0.01**	**0.09 ± 0.07**	0.25 ± 0.33	$4.7 \cdot 10^{31} \pm 1.0 \cdot 10^{32}$
	Explicit Res Net BN	1.01 ± 0.32	317 ± 20	$1.2 \cdot 10^{24} \pm 2.8 \cdot 10^{24}$	$+\infty$
	Explicit Res Tanh	0.98 ± 0.1	14.0 ± 4.0	31.7 ± 70.5	$+\infty$
	FNO	**0.04 ± 0.01**	**0.1 ± 0.08**	**0.03 ± 0.04**	$4.7 \cdot 10^{8} \pm 1.0 \cdot 10^{9}$
	Implicit ResNet (Ours)	14.0 ± 9.0	25.0 ± 27.0	27.4 ± 24	**27.5 ± 24.2**
Burgers'	Explicit Res Net	0.17 ± 0.03	0.90 ± 0.38	$2.77 \cdot 10^{19} \pm 6.2 \cdot 10^{19}$	$+\infty$
	Explicit Res Net BN	0.51 ± 0.13	51.63 ± 34.89	$+\infty$	$+\infty$
	Explicit Res Tanh	0.84 ± 0.22	44.67 ± 11.58	$+\infty$	$+\infty$
	FNO	**0.02 ± 0.002**	**0.03 ± 0.006**	$5.31 \cdot 10^{10} \pm 11.2 \cdot 10^{10}$	$+\infty$
	Implicit ResNet (Ours)	4.90 ± 0.64	7.91 ± 0.30	**0.67 ± 0.43**	**0.66 ± 0.44**

Table A.3: Results of our approach compared to baselines on the *Advection equation* and *Burgers' equation*. We calculate the means and standard deviations of relative error for each model based on 5 runs with different seeds. The mid-range time is 40 for the *Advection equation* and 0.075 for *Burgers'* and the long range time is respectively 400 and 0.15. Recall that $\Delta t_{adv} = 1$ and $\Delta t_{bur} = 0.0005$. All relative errors are in percentages.

	Model	Test relative Error	Relative error at mid-range $T_{adv} = 40 \cdot \Delta t_{adv}$ $T_{bur} = 150 \cdot \Delta t_{bur}$	Relative error at long-range $T_{adv} = 400 \cdot \Delta t_{adv}$ $T_{bur} = 300 \cdot \Delta t_{bur}$
Advection	Explicit Res Net	**0.5 ± 0.009**	106.5 ± 66.0	$+\infty$
	FNO	1.1 ± 0.1	**34.5 ± 19.0**	$7.2 \cdot 10^{5} \pm 1.6 \cdot 10^{6}$
	Implicit ResNet (Ours)	10.7 ± 4.2	1026.8 ± 346.7	**1037.1 ± 347.9**
Burgers'	Explicit Res Net	2.9 ± 0.3	$6.9 \cdot 10^{10} \pm 1.5 \cdot 10^{11}$	$+\infty$
	FNO	**0.6 ± 0.004**	$3.6 \cdot 10^{6} \pm 8.0 \cdot 10^{6}$	$+\infty$
	Implicit ResNet (Ours)	10.7 ± 0.3	**277.7 ± 102.6**	**340.0 ± 129.9**

Appendix B

Appendix of Chapter 2

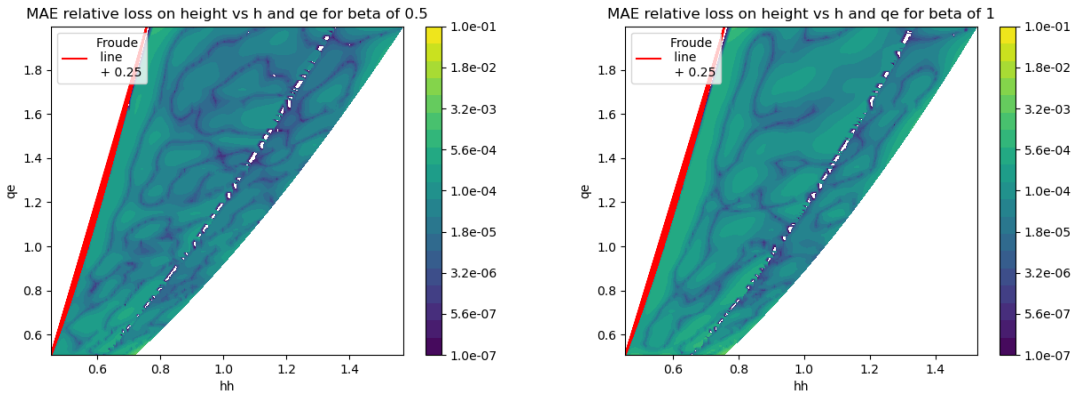
B.1 ODE analysis

B.1.1 Noise study

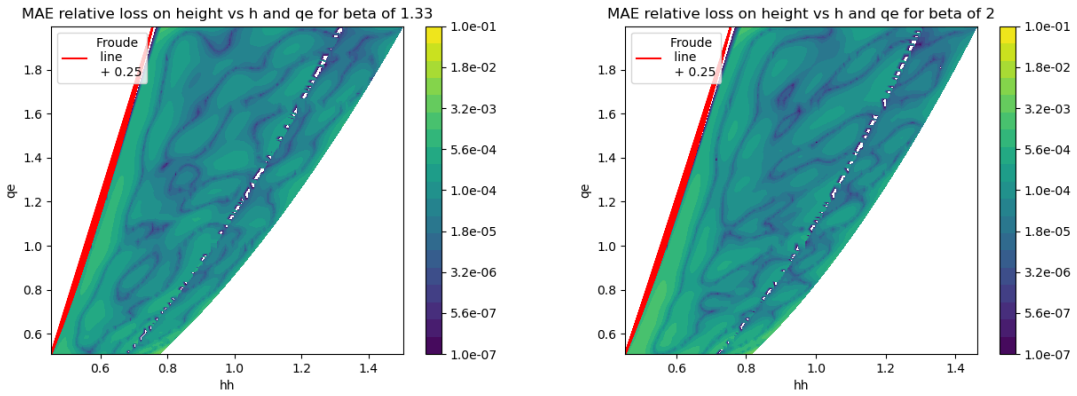
Table B.1: Friction errors for different noise standard deviations σ.

Noise σ	**Relative Friction MAE** ($\times 10^{-3}$)	**Relative Height MAE** ($\times 10^{-4}$)
0	0.61	0.28
0.0125	2.30	1.48
0.025	2.66	1.48
0.05	4.90	5.63
0.1	4.10	2.78
0.2	6.16	5.57
0.4	7.36	6.18

B.1.2 Different friction laws study



(a) β of 1/2. (b) β 1.

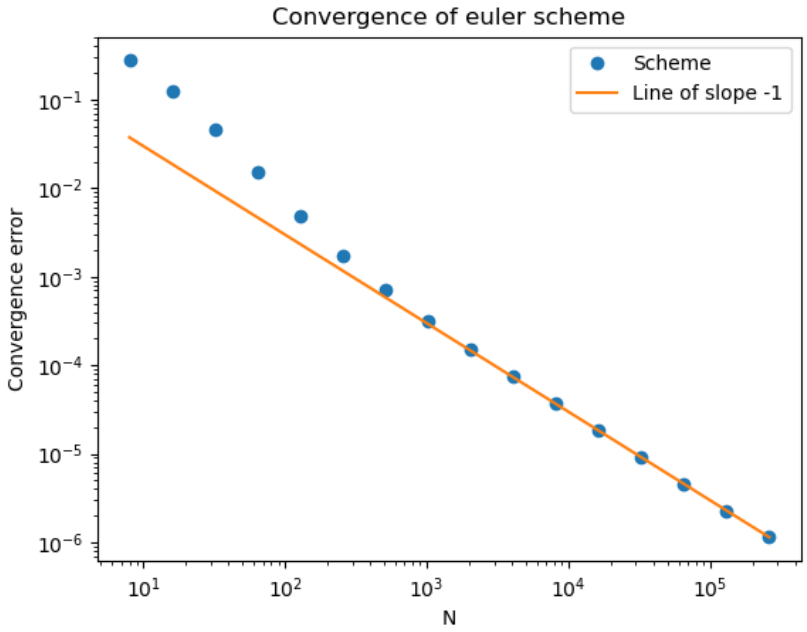


(c) β 4/3. (d) β 2.

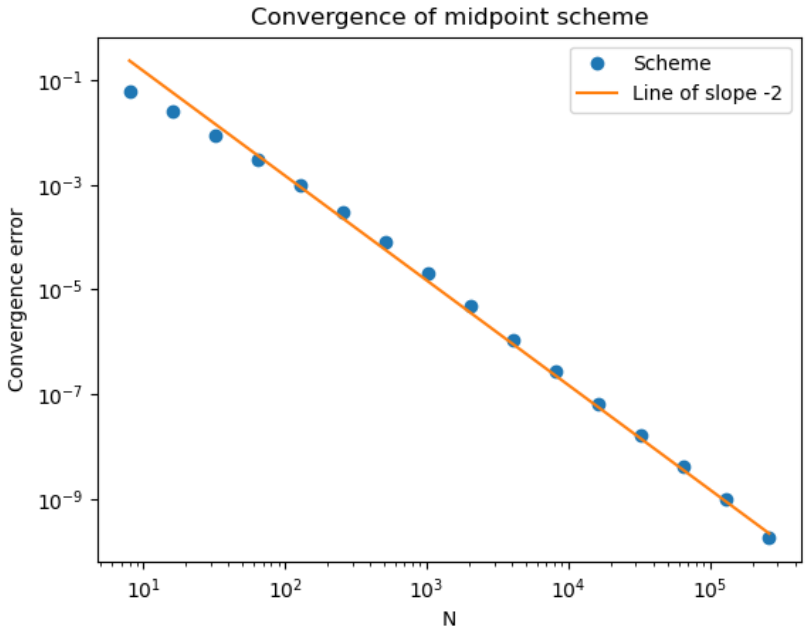
Figure B.1: Relative height absolute error for different friction laws with changing β for a coefficient of 0.3.

B.1.3 Convergence

As can be seen in Figure 4.20b andFigure B.2, the convergence of numerical schemes on the ODE case are coherent with the order of the schemes, a convergence of order 1 for Euler, of order 2 for midpoint and of order 4 for RK4.



(a) Convergence of Euler scheme (without any neural network) for the river case.



(b) Convergence of Midpoint scheme (without any neural network) for the river case.

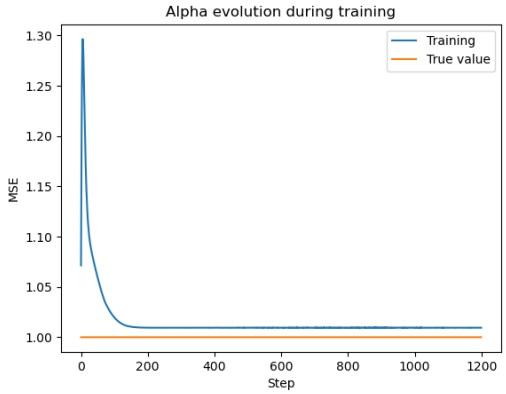
Figure B.2: Convergence of Euler and Midpoint schemes with the size of the grid.

The study of the traditional friction law learning, initialized with the underlying true parameters, leads to the results in Table B.2 and with a random initialization

in Table 4.12. A further study is available in Figure B.3 and Figure B.4, where the evolution of the parameters during the optimization is shown. With the true initial guesses, it is interesting to see the departure from the true values to adapt the friction to the numerical diffusion of the scheme. This illustrates that in cases where there is diffusion, as are more practical applications, the traditional approach does not have enough parameters to adapt the friction law.

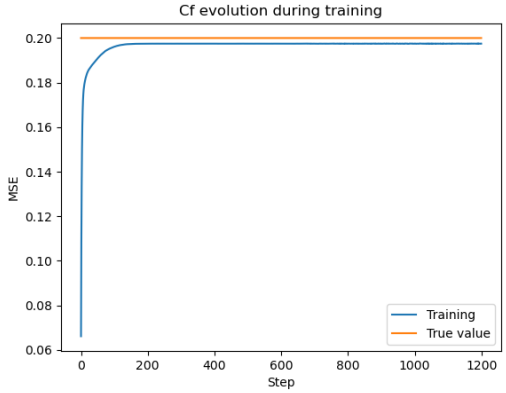
Table B.2: Height and friction MAE for different discretization of three schemes with 3 parameters estimation using true values of the parameters for the initialization. The parameter estimation is done using Adam, no neural network has been used for this table. The number of points seen during training and the discretization change. Friction MAE is expressed with factor $\times 10^{-3}$ and height with factor $\times 10^{-4}$. 500 trajectories are used for training.

Scheme \Disc	**16**		**32**		**64**	
	Friction	**Height**	**Friction**	**Height**	**Friction**	**Height**
Euler	96.99	68.79	50.18	32.97	25.43	15.72
Midpoint	68.48	52.19	38.64	25.59	20.06	12.82
RK4			39.64	25.65	20.29	12.50
	128		**256**		**512**	
Euler	12.74	7.65	6.37	3.78	3.18	1.88
Midpoint	10.17	6.40	5.11	3.18	2.56	1.59
RK4	10.22	6.31	5.12	3.17	2.56	1.59

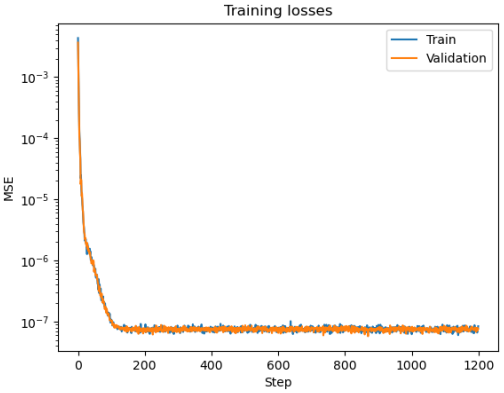


(a) α evolution during training.

(b) β evolution during training.

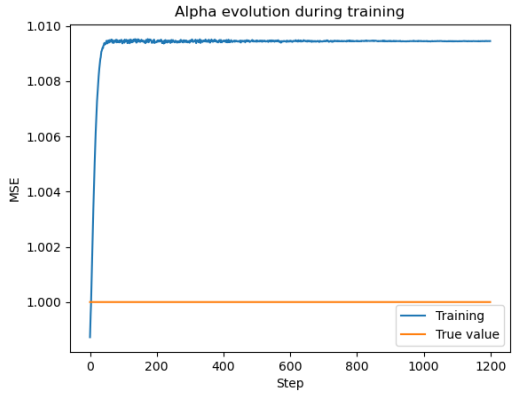


(c) C_f evolution during training.

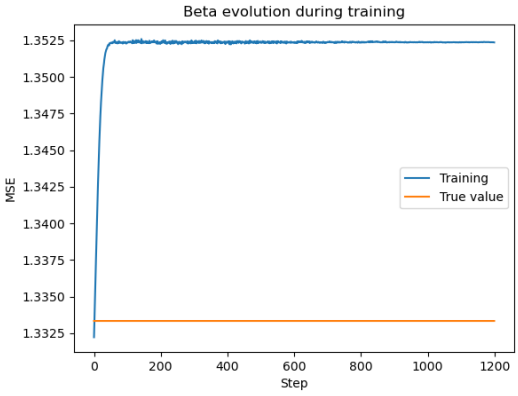


(d) Loss evolution during training.

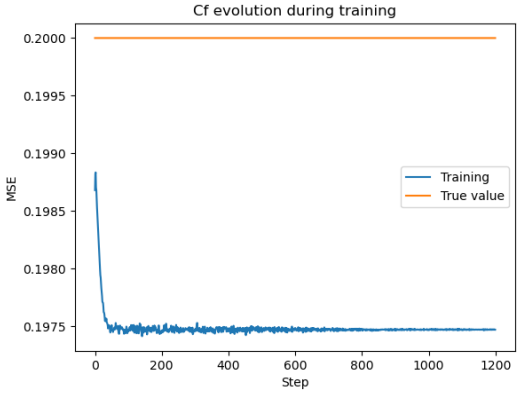
Figure B.3: Parameter evolution and training loss during the training of the numerical analysis approach for a grid of size 512 and RK4 scheme using high-fidelity solutions as a training set.



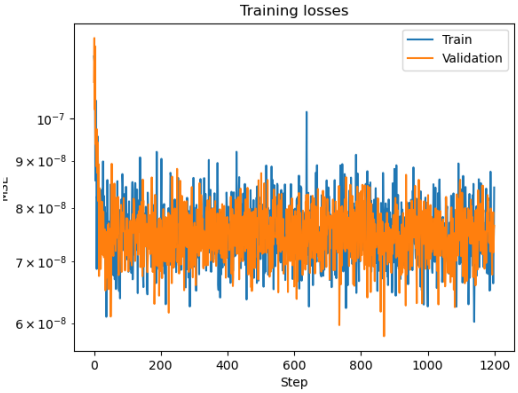
(a) α evolution during training.



(b) β evolution during training.



(c) C_f evolution during training.



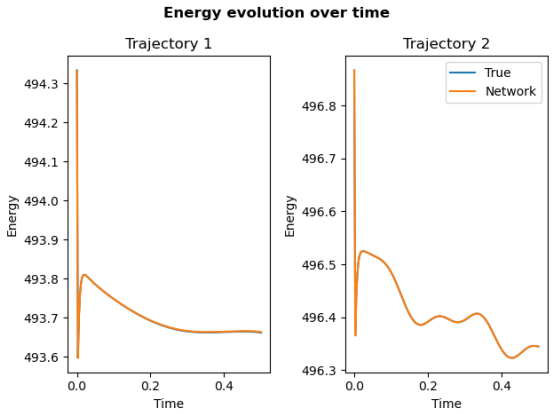
(d) Loss evolution during training.

Figure B.4: Parameter evolution and training loss during the training of the numerical analysis approach for a grid of size 512 and RK4 scheme using high-fidelity solutions as a training set and true values of the parameters for the initialization.

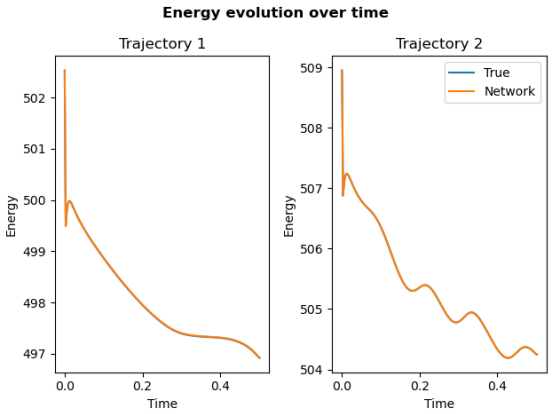
B.2 Swimming pool analysis

As can be seen in Figure B.5a and Figure B.5b, the energy is overall decreasing with time, which is to be expected and is a hard constraint of the Shallow Water equations. However, it is not monotone, which can be explained by the grid being not fine enough to guarantee monotony.

A view of the friction law for the PDE settings at $h = 1$ shows that the network struggles to capture the true value of the friction around $u = 0$, but capture it very accurately with higher water flows. This can be explained by the fact that when the water flow u is close to zero, the influence of the friction on the observations is negligible, thus the network does not need to learn it very accurately to reproduce the observations, namely the water height h and the water flow u.

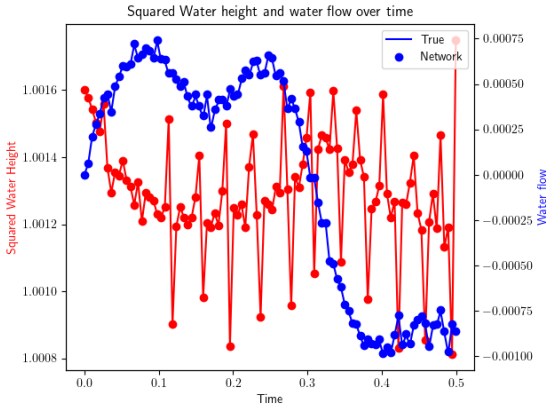


(a) Small variations energy for 2 trajectories

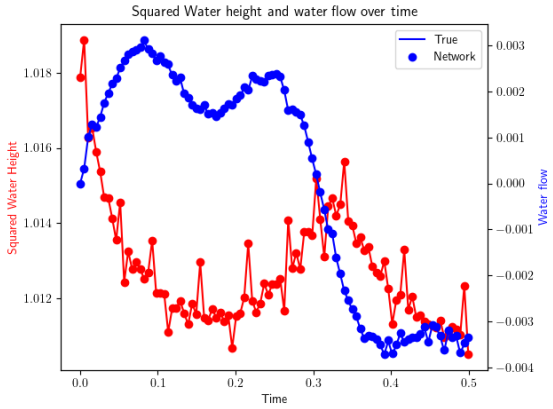


(b) Medium variations energy for 2 trajectories

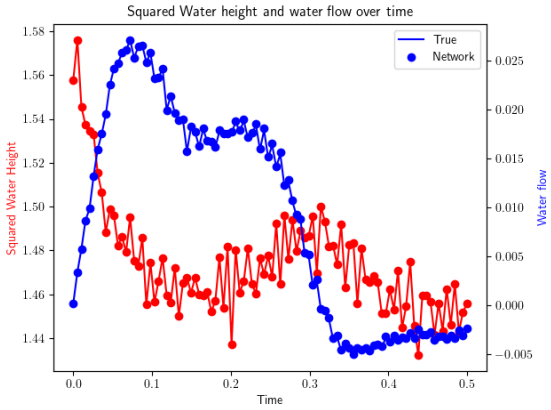
Figure B.5: Energy plots for two trajectories for the small and medium variations setting.



(a) Small variations squared water height and water flow for several trajectories for the small variations setting



(b) Medium variations squared water height and water flow for several trajectories for the medium variations setting



(c) Large variations squared water height and water flow for several trajectories for the medium variations setting

Figure B.6: Squared water height and water flow plots for several trajectories for the small and medium variations setting.

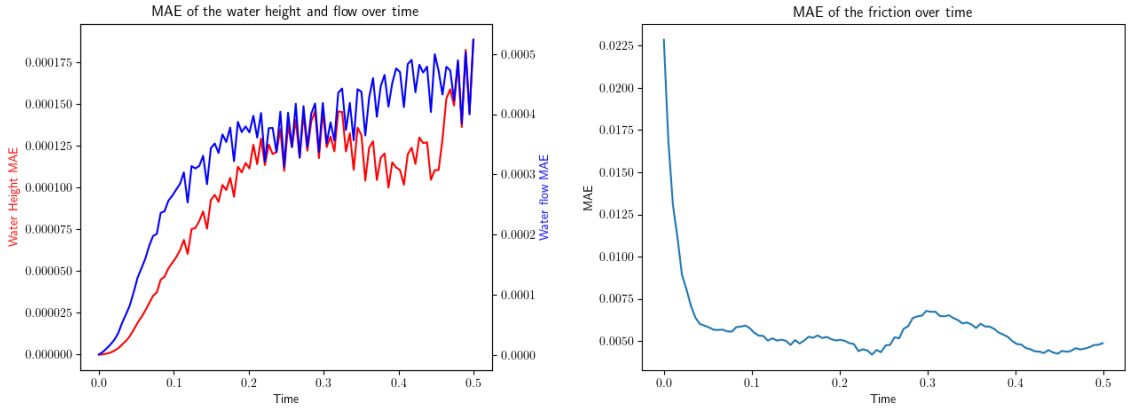
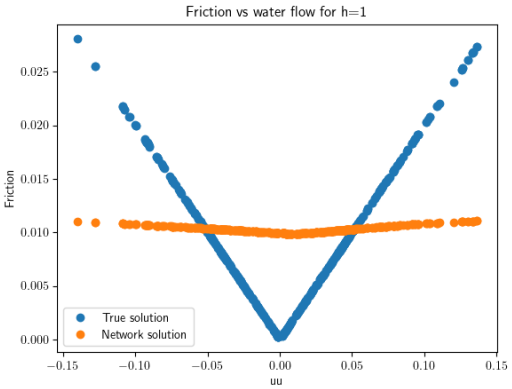
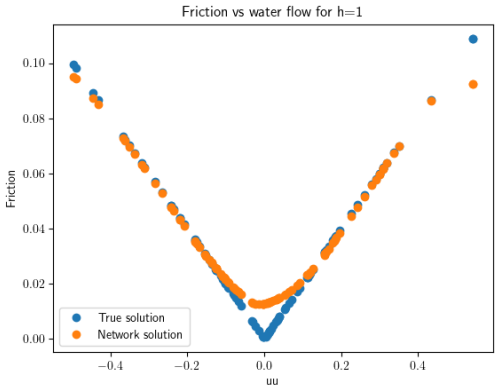


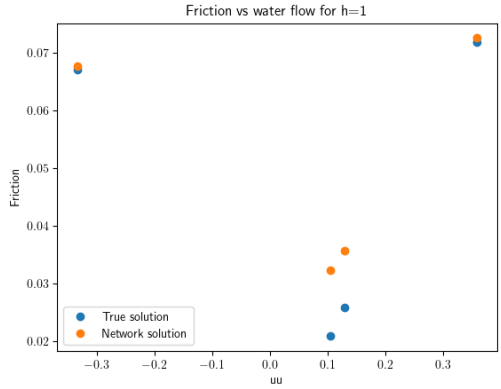
Figure B.7: Water height, flow and friction MAE over time averaged for 25 trajectories for the large variations PDE setting.



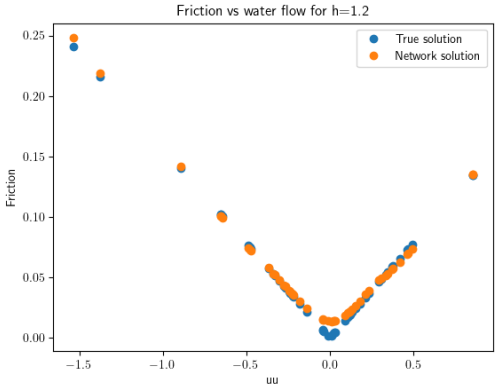
(a) View of the friction for $h = 1$ for the small variations setting



(b) View of the friction for $h = 1$ for the medium variations setting



(c) View of the friction for $h = 1$ for the large variations setting



(d) View of the friction for $h = 1.2$ for the large variations setting

Figure B.8: View of the friction for $h = 1$ for the three PDE setting.

Appendix C

Appendix of Chapter 5

C.1 Time Series Continuous Modeling for Imputation and Forecasting with Implicit Neural Representations

Reproductiblity statement

Our work is entirely reproducible, and all the references to the information in order to reproduce it are in this section.

Code. The code for all our experiments is available at this link.

Data. A subset of the processed data is available with the code in this link. The dataset description, processing and normalization are presented in appendix C.1.2.

Model. The model and the training details are presented in section 5.2.3 and the hyperparameter selection is available in appendix C.1.1.1.

GPU. We used NVIDIA TITAN RTX 24Go single GPU to conduct all the experiments for our method, which is coded in PyTorch (Python 3.9.2).

C.1.1 Architecture details and ablation studies

C.1.1.1 Architecture details

For all imputation and forecasting experiments we choose the following hyperparameters :

- z dimension: 128
- Number of layers: 5
- Hidden layers dimension: 256
- $\gamma(t) \in \mathbb{R}^{2\times 64}$
- z code learning rate (α in algorithm 3): 10^{-2}
- Hypernetwork and INR learning rate: 5×10^{-4}
- Number of steps in inner loop: $K = 3$
- Number of epochs: 4×10^4
- Batch size: 64

It is worth noting that the hyperparameters mentioned above remain consistent across all experiments conducted in the paper. We chose to maintain a fixed set of hyperparameters for our model, while other imputation and forecasting approaches commonly fine-tune hyperparameters based on a validation dataset. The obtained results exhibit high robustness across various settings, suggesting that the selected hyperparameters are already effective in achieving reliable outcomes.

C.1.1.2 Fourier features vs SIREN on imputation task

Baseline The SIREN network differs from the Fourier features network because it does not explicitly incorporate frequencies as input. Instead, it is a multi-layer perceptron network that utilizes sine activation functions. An adjustable parameter, denoted ω_0, is multiplied with the input matrices of the preceding layers to capture a broader range of frequencies. For this comparison, we adopt the same hyperparameters described in appendix C.1.1.1, selecting $\omega_0 = 30$ to align with Sitzmann et al. (2020b). Furthermore, we set the learning rate of both the hypernetwork and the INR to 5×10^{-5} to enhance training stability. In Table C.1, we compare the imputation results obtained by the Fourier features network and the SIREN network, specifically focusing on the first time window from the *Electricity*, *Traffic* and *Solar* datasets.

Table C.1: MAE imputation errors on the first time window of each dataset. Best results are bold.

	τ	TimeFlow	TimeFlow w SIREN
Electricity	0.05	**0.323**	0.466
	0.10	**0.252**	0.350
	0.20	**0.224**	0.242
	0.30	**0.211**	0.222
	0.50	**0.194**	0.209
Solar	0.05	**0.105**	0.114
	0.10	**0.083**	0.094
	0.20	**0.065**	0.079
	0.30	**0.061**	0.072
	0.50	**0.056**	0.066
Traffic	0.05	**0.292**	0.333
	0.10	**0.220**	0.252
	0.20	**0.168**	0.191
	0.30	**0.152**	0.163
	0.50	**0.141**	0.154

Results According to the results presented in table C.1, the Fourier features network outperforms the SIREN network in the imputation task on these datasets. Notably, the performance gap between the two network architectures are more pronounced at low sampling rates. This disparity can be attributed to the SIREN network's difficulty in accurately capturing high frequencies when the time series is sparsely observed. We hypothesize that the MLP with ReLU activations correctly learns the different frequencies of time series with multi-temporal patterns by switching on or off the Fourier embedding frequencies.

C.1.1.3 Influence of the latent code dimension

The dimension of the latent code z is a crucial parameter in our architecture. If it is too small, it underfits the timeseries. Consequently, this adversely affects the performance of both the imputation and forecasting tasks. On the other hand, if the dimension of z is too large, it can lead to overfitting, hindering the model's ability to generalize to new data points.

Baselines To investigate the impact of z dimensionality on the performance of TimeFlow, we conducted experiments on the *Electricity* dataset, specifically focusing on the imputation task. We varied the sizes of z within $\{32, 64, 128, 256\}$. The other hyperparameters are set as presented in appendix C.1.1.1. The obtained results for each z dimension are summarized in table C.2.

Results The results presented in table C.2 highlight the importance of the z-dimension, as it significantly impacts the results. We found that a dimension of 128

Table C.2: MAE imputation errors on the first time window of *Electricity* dataset. Best results are bold.

	τ	$\dim(z) = 32$	$\dim(z) = 64$	$\dim(z) = 128$	$\dim(z) = 256$
Electricity	0.05	0.370	0.364	**0.323**	0.354
	0.10	0.302	0.301	**0.252**	0.283
	0.20	0.269	0.265	**0.224**	0.247
	0.30	0.242	0.245	**0.211**	0.238
	0.50	0.224	0.240	**0.194**	0.217

was a suitable compromise for all our experiments.

C.1.1.4 Influence of the number of gradient steps

As can be seen in table C.3, using three gradient steps at inference yield an inference of less than 0.2 seconds. The latter can still be reduced by doing only one step at the cost of an increase in the forecasting error. As observed in table C.3, increasing the number of gradient steps above 3 steps during inference does not improve forecasting performance.

Table C.3: Inference time (in seconds) and MAE on the forecasting task on the *Electricity* dataset for a horizon of length 720, a look-back window of length 512, and a varying number of adaptation gradient steps. The statistics are computed over 10 runs using an NVIDIA TITAN RTX GPU.

Gradient descent steps	1	3	10	50	500	5000
Inference time (s)	0.109 ± 0.003	0.176 ± 0.009	0.427 ± 0.031	3.547 ± 0.135	17.722 ± 0.536	189.487 ± 8.060
MAE	0.351 ± 0.038	0.303 ± 0.041	0.300 ± 0.040	0.299 ± 0.039	0.302 ± 0.038	0.308 ± 0.037

C.1.1.5 TimeFlow variants with other meta-learning techniques

Baselines Before converging to the current architecture and optimization of TimeFlow, we explored different options to condition the INR with the observations. The first one was inspired by the neural process architecture, which uses a set encoder to transform a set of observations $(t_i, x_{t_i})_{i \in \mathcal{I}}$ into a latent code z by applying a pooling layer after a feed forward network. We observed that this encoder in combination with the modulated fourier features network was able to achieve relatively good results on the forecasting task but suffered of underfitting on more complex datasets such as *Electricity*.

This led us to consider auto-decoding methods instead, i.e. encoder-less architectures for conditioning the weights of the coordinate-based network. We trained TimeFlow with the REPTILE algorithm (Nichol et al., 2018b), which is a first-order meta-learning technique that adapts the code in a few steps of gradient descent. In

contrast with a second-order method, we observed that REPTILE was less costly to train but struggled to escape sub optimal minima, which led to unstable training and underfitting.

From an implementation point of view, the only difference between second order and first order, is that in the latter the code is detached from the computation graph before taking the outer-loop parameter update. When the code is not detached, it remains a function of the common parameters $z = z(\theta, w)$, which means that the computation graph for the outer-loop also includes the inner-loop updates to the codes. Therefore the outer-loop gradient update involves a gradient through a gradient and requires an additional backward pass through the INR to compute the Hessian. Please refer to Finn et al. (2017) for more technical details.

Table C.4: Comparison of second-order and first-order (REPTILE) meta learning for TimeFlow on the imputation task. Mean MAE results on the missing grid over five different time windows. τ stands for the subsampling rate. Bold results are best.

	τ	TimeFlow	TimeFlow w REPTILE
Electricity	0.05	**0.324 ± 0.013**	0.363 ± 0.062
	0.10	**0.250 ± 0.010**	0.343 ± 0.036
	0.20	**0.225 ± 0.008**	0.312 ± 0.043
	0.30	**0.212 ± 0.007**	0.308 ± 0.035
	0.50	**0.194 ± 0.007**	0.305 ± 0.046
Solar	0.05	**0.095 ± 0.015**	0.125 ± 0.025
	0.10	**0.083 ± 0.015**	0.123 ± 0.032
	0.20	**0.072 ± 0.015**	0.108 ± 0.021
	0.30	**0.061 ± 0.012**	0.105 ± 0.027
	0.50	**0.054 ± 0.013**	0.102 ± 0.021
Traffic	0.05	**0.283 ± 0.016**	0.304 ± 0.026
	0.10	**0.211 ± 0.012**	0.264 ± 0.009
	0.20	**0.168 ± 0.006**	0.242 ± 0.019
	0.30	**0.151 ± 0.007**	0.218 ± 0.020
	0.50	**0.139 ± 0.007**	0.216 ± 0.017

Results In Table C.4, we show the performance of first-order TimeFlow on the imputation task. In low sampling regimes the difference with TimeFlow is less perceptive, but its performance plateaus when the number of points increases. This is not surprising. Indeed, as though the task is actually simpler when τ increases, the optimization is made more difficult with the increased number of observations. We provide the performance of TimeFlow with a set encoder on the Forecasting task in Table C.5. We observed that this version failed to generalize well for complex datasets.

Table C.5: Comparison of optimization-based and set-encoder-based meta learning for TimeFlow on the forecasting task. Mean MAE forecast results over different time windows. H stands for the horizon. Bold results are best.

	H	TimeFlow	TimeFlow w set encoder
Electricity	96	**0.228 ± 0.026**	0.362 ± 0.032
	192	**0.238 ± 0.020**	0.360 ± 0.028
	336	**0.270 ± 0.031**	0.382 ± 0.038
	720	**0.316 ± 0.055**	0.431 ± 0.059
SolarH	96	**0.190 ± 0.013**	0.251 ± 0.071
	192	**0.202 ± 0.020**	0.239 ± 0.058
	336	**0.209 ± 0.017**	0.235 ± 0.040
	720	**0.218 ± 0.048**	0.231 ± 0.032
Traffic	96	**0.217 ± 0.036**	0.276 ± 0.031
	192	**0.212 ± 0.028**	0.281 ± 0.034
	336	**0.238 ± 0.034**	0.297 ± 0.042
	720	**0.279 ± 0.050**	0.333 ± 0.048

C.1.1.6 Influence of the modulation

In TimeFlow, we apply shift modulations to the parameters of the INR, i.e. for each layer l we only modify the biases of the network with an extra bias term $\phi_l^{(j)}$. We generate these bias terms with a linear hypernetwork that maps the code $z^{(j)}$ to the modulations. The output of the l-th layer of the modulated INR is thus given by $\phi_{l+1} = \text{ReLU}(\theta_l \phi_{l-1} + b_l + \psi_l^{(j)})$, where $\psi_l^{(j)} = W_l z^{(j)}$ and $(W_l)_{l=1}^L$ are parameters of the hypernetwork. However, another common modulation is the combination of the scale and shift modulation, which leads to the output of the l-th layer of the modulated INR being given by $\phi_{l+1} = \text{ReLU}((S_l z^{(j)}) \circ (\theta_l \phi_{l-1} + b_l) + \psi_l^{(j)})$, where $\psi_l^{(j)} = W_l z^{(j)}$, and $(W_l)_{l=1}^L$ and $(S_l)_{l=1}^L$ are parameters of the hypernetwork and $\circ$ is the Hadamard product.

In table C.6, we conduct additional experiments on the *Electricity* dataset in the forecasting setting with different time horizons. In these experiments, we compare two scenarios: one where the INR is modulated only by a shift factor and the other where the INR is modulated by both a shift and a scale factor. We kept the architecture and hyperparameters consistent with those described in Appendix C.1.1.1. The experiments shown in table C.6 indicate that the INR is longer to train with shift and scale modulations due to the increased number of parameters involved. Furthermore, we observe that the shift and scale modulated INR performed similarly or even worse than the INR with only shift modulation. These two drawbacks, namely an increased computational time and similar or worse performances, motivate modulating the INR only by a shift factor.

Table C.6: Ablation on modulations for the forecasting task on *Electricity* dataset for different horizons. Models are trained on a given time window and tested on four new time windows. Models are trained on a single NVIDIA TITAN RTX GPU.

	96		192		336		720	
	MAE	Training time	MAE	Training time	MAE	Training time	MAE	Training time
Shift	0.233 ± 0.014	2h30	0.245 ± 0.016	2h31	0.264 ± 0.020	2h33	0.303 ± 0.041	2h46
Shift and scale	0.257 ± 0.019	3h29	0.263 ± 0.014	3h32	0.268 ± 0.025	3h45	0.308 ± 0.037	4h14

C.1.1.7 Discussion on other hyperparameters

While the dimension of z is indeed a crucial hyperparameter, it is important to note that other hyperparameters also play a significant role in the performance of the INR. For example, the number of layers in the FFN directly affects the ability of the model to fit the time series. In our experiments, we have observed that using five or more layers yields good performance, and including additional layers can lead to slight improvements in the generalization settings.

Similarly, the number of frequencies used in the frequency embedding is another important hyperparameter. Using too few frequencies can limit the network's ability to capture patterns, while using too many frequencies can hinder its ability to generalize accurately.

The choice of learning rate is critical for achieving stable convergence during training. Therefore, in practice, we use a low learning rate combined with a cosine annealing scheduler to ensure stable and effective training.

C.1.2 Datasets and normalization

For the complete datasets, *Electricity* dataset is available here, *Traffic* dataset here and *Solar* data set here.

Datasets information table C.7 provides a concise overview of the main information about the datasets used for forecasting and imputation tasks.

Table C.7: Summary of datasets information

Dataset name	Number of samples	Number of time steps	Sampling frequency	Location	Years
Electricity	321	26 304	hourly	Portugal	2012 – 2014
Traffic	862	17 544	hourly	San Francisco bay	2015 – 2016
Solar	137	52 560	10 minutes	Alabama	2006
SolarH	137	8 760	hourly	Alabama	2006

z-normalization To preprocess each dataset, we apply the widely used z-normalization technique per-sample j on the entire series: $x^{(j)}_{norm} = \frac{x^{(j)} - \text{mean}(x^{(j)})}{\text{std}(x^{(j)})}$.

C.1.3 Imputation experiments

C.1.3.1 Models complexity

We can see in table C.8 that our method has fewer parameters than SOTA imputation methods, 10 times less than BRITS and 20 times less than SAITS. It is mainly due to their modelisation of interaction between samples. SAITS, which is based on transformers has the highest number of parameters when mTAN has the lowest number of parameters.

Table C.8: Number of parameters for each DL methods on the imputation task on the *Electricity* dataset.

	TimeFlow	DeepTime	NeuralProcess	mTAN	SAITS	BRITS	TIDER
Number of parameters	602k	1315k	248k	113k	11 137k	6 220k	1 034k

C.1.3.2 Imputation for previously unseen time series

Setting In this section we analyze in details the imputations results for previously unseen time series described in section 5.2.4.1. Specifically, TimeFlow is trained on a given set of time series within a defined time window and then used for inference on new time series. We train TimeFlow on 50 % of the samples and consider the remaining 50 % as the new time series.

We compare in table C.9 observed grid fit scores and missing grid inference scores for time series known at training and time series unknown at training.

Results The results presented in table C.9 indicate that the inference MAE for missing grids shows consistency between known and new samples, regardless of the data or sampling rate. However, it is worth noting that there is a slight drop in performance compared to the results in table table 5.2. This decrease is because in table C.9, the shared architecture is trained on only half the samples, affecting its overall performance.

C.1.3.3 Details on DeepTime adaptation for imputation

As DeepTime was proposed to address the forecasting task with a deeptime-index model, the authors did not tackle the task of imputation and left it out for future work. Given the success of this method and the motivation of our work, we wanted

Table C.9: TimeFlow MAE imputation errors results for imputation previsouly unseen time series.

		Known time series		New time series	
	τ	Fit	Inference	Fit	Inference
Electricity	0.05	0.060 ± 0.010	0.402 ± 0.021	0.142 ± 0.083	0.413 ± 0.026
	0.10	0.046 ± 0.006	0.302 ± 0.010	0.144 ± 0.098	0.309 ± 0.016
	0.20	0.067 ± 0.015	0.285 ± 0.014	0.154 ± 0.089	0.291 ± 0.022
	0.30	0.093 ± 0.022	0.266 ± 0.010	0.163 ± 0.073	0.271 ± 0.017
	0.50	0.108 ± 0.012	0.236 ± 0.010	0.167 ± 0.061	0.245 ± 0.017
Solar	0.05	0.014 ± 0.002	0.104 ± 0.015	0.050 ± 0.037	0.109 ± 0.016
	0.10	0.017 ± 0.002	0.092 ± 0.015	0.052 ± 0.036	0.099 ± 0.017
	0.20	0.028 ± 0.008	0.078 ± 0.014	0.058 ± 0.031	0.089 ± 0.017
	0.30	0.038 ± 0.009	0.072 ± 0.013	0.063 ± 0.028	0.084 ± 0.018
	0.50	0.045 ± 0.011	0.066 ± 0.013	0.067 ± 0.025	0.080 ± 0.019
Traffic	0.05	0.044 ± 0.003	0.291 ± 0.013	094 ± 0.051	0.291 ± 0.012
	0.10	0.033 ± 0.001	0.209 ± 0.010	0.093 ± 0.060	0.216 ± 0.012
	0.20	0.037 ± 0.006	0.175 ± 0.008	0.095 ± 0.058	0.186 ± 0.013
	0.30	0.048 ± 0.005	0.164 ± 0.006	0.098 ± 0.051	0.175 ± 0.013
	0.50	0.068 ± 0.004	0.159 ± 0.007	0.110 ± 0.042	0.169 ± 0.012

to explore its capabilities to impute time series with several subsampling rates. Following our current framework, we first tried to train the model in a self-supervised way, i.e. trying to reconstruct observations $x^{(j)} \in \mathcal{T}^{(j)}$ after the INR has been conditioned with the Ridge Regressor on the same set of observations, but discovered failure cases for $\tau \leq 0.20$. To be faithful to the original supervised training of DeepTime, we therefore randomly mask out 50% of the observations that we use as context for the Ridge Regressor and try to infer the other 50% (the targets) to train the INR.

We provide a qualitative comparison of the model's performance with these two different training procedures in Figure C.1. We can notice that the model that results from the self-supervised training perfectly fits the observations but completely misses the important patterns of the series. On the other hand, when DeepTime is trained to infer target values based on observations, it is able to capture the general trends. We think that in the small subsampling regime ($\tau \leq 0.20$), the Ridge Regressor easily fits very well all the observations which hinders the training of the INR's basis.

C.1.4 Forecasting experiments

C.1.4.1 Distinction between adjacent time windows and new time windows during inference

In section 5.2.4.2, we presented the forecasting results for periods outside the training period. These periods can be classified into two types: adjacent to or disjoint from the training period. fig. C.2 illustrates these distinct test periods for the *Electricity*

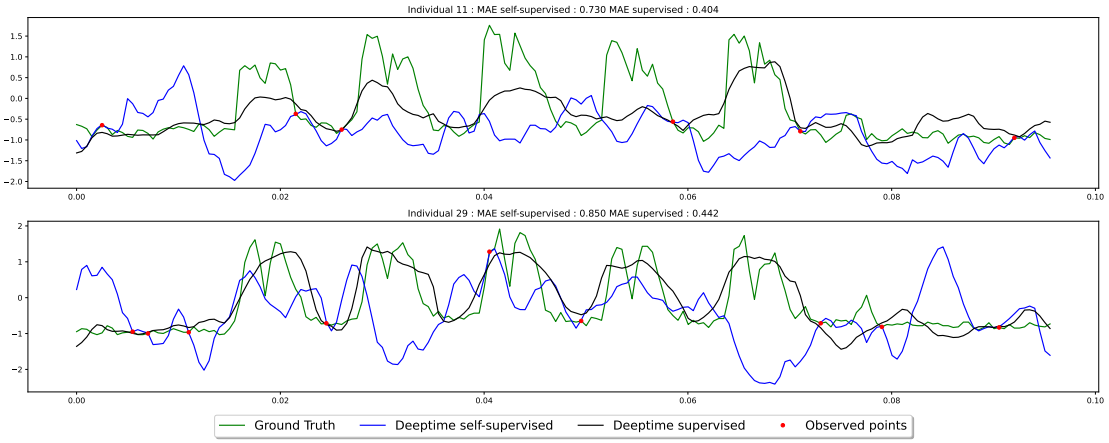


Figure C.1: *Electricity dataset*. Self supervised DeepTime imputation (blue line) and supervised DeepTime imputation (black line) with 5% of known point (red points) on the eight first days of samples 11 (top) and 29 (bottom).

dataset. The same principle applies to the *Traffic* and *SolarH* datasets, with one notable difference: the number of test periods is smaller in these datasets compared to *Electricity* dataset due to the fewer time steps available.

In table 5.3, we presented the results indistinctly for the two types of test periods: adjacent to and disjoint from the training window. Here, we aim at differentiating the results for these two types of window and emphasize their significant impact on Informer and AutoFormer results. Specifically, table C.10 showcases the results for the test periods adjacent to the training window. In contrast, table C.11 displays the results for the test periods disjointed from the training window

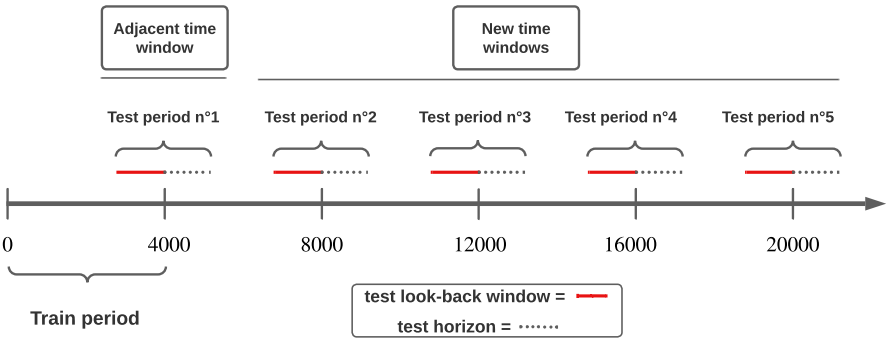


Figure C.2: Distinction between adjacent time windows and new time windows during inference for the *Electricity* dataset

Results TimeFlow, PatchTST, DLinear and DeepTime maintain consistent forecasting results whether tested on the period adjacent to the training period or on a disjoint period. However, AutoFormer and Informer show a significant drop in

performance when tested on new disjoint periods.

Table C.10: Mean MAE forecast results for adjacent time windows. H stands for the horizon. Bold results are best, underline results are second best.

		Continuous methods			Discrete methods			
	H	TimeFlow	DeepTime	Neural Process	Patch-TST	DLinear	AutoFormer	Informer
Electricity	96	0.218 ± 0.017	0.240 ± 0.027	0.392 ± 0.045	**0.214 ± 0.020**	0.236 ± 0.035	0.310 ± 0.031	0.293 ± 0.0184
	192	0.238 ± 0.012	0.251 ± 0.023	0.401 ± 0.046	**0.225 ± 0.017**	0.248 ± 0.032	0.322 ± 0.046	0.336 ± 0.032
	336	0.265 ± 0.036	0.290 ± 0.034	0.434 ± 0.075	**0.242 ± 0.024**	0.284 ± 0.043	0.330 ± 0.019	0.405 ± 0.044
	720	0.318 ± 0.073	0.356 ± 0.060	0.605 ± 0.149	**0.291 ± 0.040**	0.370 ± 0.086	0.456 ± 0.052	0.489 ± 0.072
SolarH	96	**0.172 ± 0.017**	0.197 ± 0.002	0.221 ± 0.048	0.232 ± 0.008	0.204 ± 0.002	0.261 ± 0.053	0.273 ± 0.023
	192	**0.198 ± 0.010**	0.202 ± 0.014	0.244 ± 0.048	0.231 ± 0.027	0.211 ± 0.012	0.312 ± 0.085	0.256 ± 0.026
	336	0.207 ± 0.019	**0.200 ± 0.012**	0.241 ± 0.005	0.254 ± 0.048	0.212 ± 0.019	0.341 ± 0.107	0.287 ± 0.006
	720	**0.215 ± 0.016**	0.240 ± 0.011	0.403 ± 0.147	0.271 ± 0.036	0.246 ± 0.015	0.368 ± 0.006	0.341 ± 0.049
Traffic	96	0.216 ± 0.033	0.229 ± 0.032	0.283 ± 0.028	**0.201 ± 0.031**	0.225 ± 0.034	0.299 ± 0.080	0.324 ± 0.113
	192	0.208 ± 0.021	0.220 ± 0.020	0.292 ± 0.023	**0.195 ± 0.024**	0.215 ± 0.022	0.320 ± 0.036	0.321 ± 0.052
	336	0.237 ± 0.040	0.247 ± 0.033	0.305 ± 0.039	**0.220 ± 0.036**	0.244 ± 0.035	0.450 ± 0.127	0.394 ± 0.066
	720	**0.266 ± 0.048**	0.290 ± 0.045	0.339 ± 0.037	0.268 ± 0.050	0.290 ± 0.047	0.630 ± 0.043	0.441 ± 0.055

Table C.11: Mean MAE forecast results for new time windows. H stands for the horizon. Bold results are best, underline results are second best.

		Continuous methods			Discrete methods			
	H	TimeFlow	DeepTime	Neural Process	Patch-TST	DLinear	AutoFormer	Informer
Electricity	96	0.230 ± 0.012	0.245 ± 0.026	0.392 ± 0.045	**0.222 ± 0.023**	0.240 ± 0.025	0.606 ± 0.281	0.605 ± 0.227
	192	0.246 ± 0.025	0.252 ± 0.018	0.401 ± 0.046	**0.231 ± 0.020**	0.257 ± 0.027	0.545 ± 0.186	0.776 ± 0.257
	336	0.271 ± 0.029	0.285 ± 0.034	0.434 ± 0.076	**0.253 ± 0.027**	0.298 ± 0.051	0.571 ± 0.181	0.823 ± 0.241
	720	0.316 ± 0.051	0.359 ± 0.048	0.607 ± 0.15	**0.299 ± 0.038**	0.373 ± 0.075	0.674 ± 0.245	0.811 ± 0.257
SolarH	96	0.208 ± 0.005	**0.206 ± 0.026**	0.221 ± 0.048	0.293 ± 0.089	0.212 ± 0.019	0.228 ± 0.027	0.234 ± 0.011
	192	**0.206 ± 0.012**	0.207 ± 0.037	0.244 ± 0.048	0.274 ± 0.060	0.223 ± 0.029	0.356 ± 0.122	0.280 ± 0.033
	336	0.211 ± 0.005	0.199 ± 0.035	0.240 ± 0.006	0.264 ± 0.088	0.223 ± 0.032	0.327 ± 0.029	0.366 ± 0.039
	720	**0.222 ± 0.020**	0.217 ± 0.028	0.403 ± 0.147	0.262 ± 0.083	0.251 ± 0.047	0.335 ± 0.075	0.333 ± 0.012
Traffic	96	0.218 ± 0.042	0.229 ± 0.032	0.283, 0.0275	**0.204 ± 0.039**	0.229 ± 0.032	0.326 ± 0.049	0.388 ± 0.055
	192	0.213 ± 0.028	0.220 ± 0.023	0.292, 0.0236	**0.198 ± 0.031**	0.223 ± 0.023	0.575 ± 0.254	0.381 ± 0.049
	336	0.239 ± 0.035	0.244 ± 0.040	0.305, 0.0392	**0.223 ± 0.040**	0.252 ± 0.042	0.598 ± 0.286	0.448 ± 0.055
	720	0.280 ± 0.047	0.290 ± 0.055	0.339, 0.0375	**0.270 ± 0.059**	0.304 ± 0.061	0.641 ± 0.072	0.468 ± 0.064

C.1.4.2 Plots comparison: TimeFlow vs PatchTST

table 5.3 demonstrates the similar forecasting performance of TimeFlow and PatchTST across all horizons. To visually represent their predictions, the figures below showcase the forecasted outcomes of these methods for two samples (24 and 38) and two horizons (96 and 192) on the *Electricity*, *SolarH*, and *Traffic* datasets.

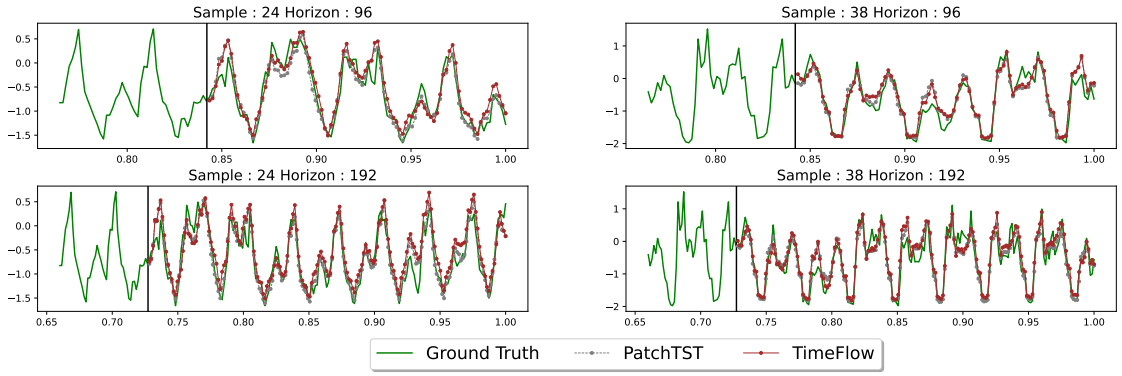


Figure C.3: Qualitative comparisons of TimeFlow vs PatchTST on the *Electricity* dataset for new time windows

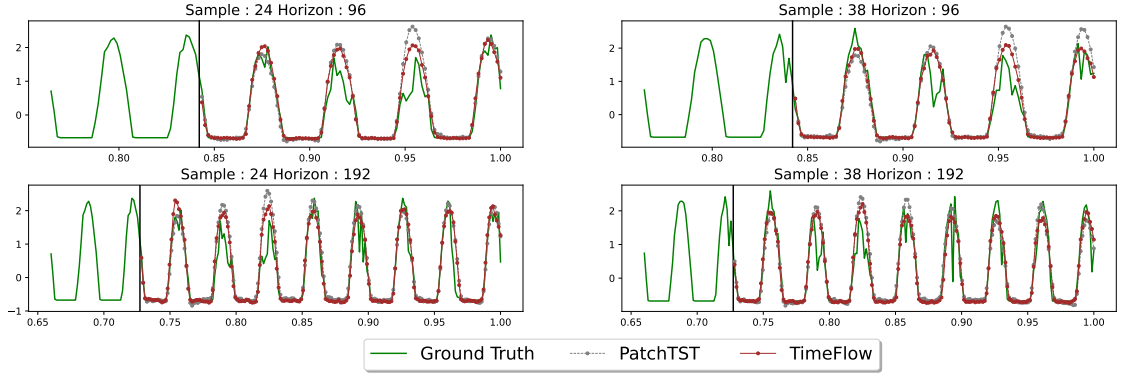


Figure C.4: Qualitative comparisons of TimeFlow vs PatchTST on the *SolarH* dataset for new time windows.

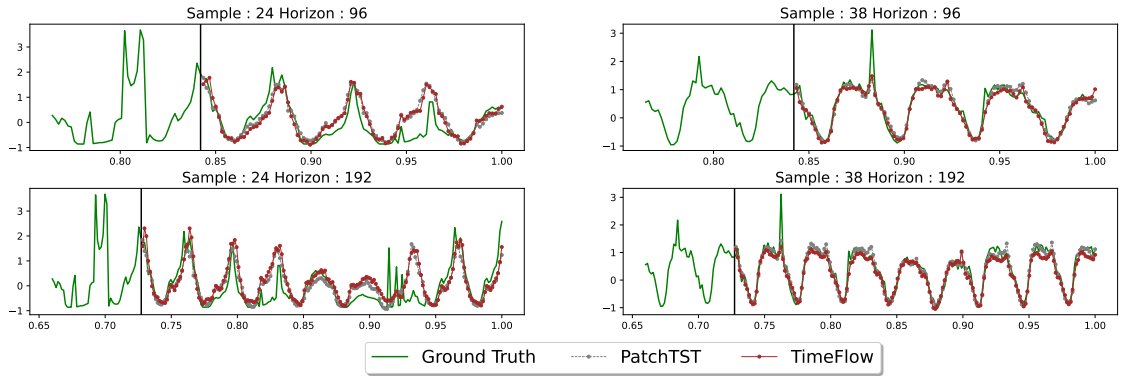


Figure C.5: Qualitative comparisons of TimeFlow vs PatchTST on the *Traffic* dataset for new time windows.

Results The visual analysis of the figures above reveals that the predictions of TimeFlow and PatchTST are remarkably similar. For instance, when examining sample 24 and horizon 192 of the *Traffic* dataset, both forecasters exhibit similar error patterns. The only noticeable distinction emerges in the *SolarH* dataset, where PatchTST tends to overestimate certain peaks.

C.1.4.3 Models complexity

In this section, we present the parameter counts and the inference time for the main forecasting baselines. Except for TimeFlow and DeepTime, the number of parameters varies with the number of samples, the look-back window, and the horizon. Thus, we report the number of parameters for two specific configurations, including a fixed dataset, a fixed look-back window, and a fixed horizon. In table C.12, we see that for PatchTST and DLinear, the larger the horizon, the more the number of parameters increases. In table C.13, it is shown that all methods' computational time increases with the horizon, which is expected. Moreover, TimeFlow is slower than the baselines that use forward computations only. Still, on the Electricity dataset, for example, the method can infer for 321 samples a horizon of 720 values with a look-back window of 512 timestamps in less than 0.2s, which does not look prohibitive for many real-world usages. This is mainly due to the small number of gradient steps at inference.

Table C.12: The number of parameters for main baselines on the forecasting task on the *Electricity* dataset for horizons 96 and 720. The look-back window size is 512.

	TimeFlow	DeepTime	Neural Process	Patch-TST	DLinear
96	602k	1 315k	480k	1 194k	98k
720	602k	1 315k	480k	6 306k	739k

Table C.13: Inference time (in seconds) for the forecasting task on the *Electricity* dataset with horizons 96 and 720 and a look-back window of length 512. The statistics are computed over 10 runs using an NVIDIA TITAN RTX GPU.

	TimeFlow	Patch-TST	DLinear	DeepTime	AutoFormer	Informer
96	0.147 ± 0.007	0.016 ± 0.002	0.007 ± 0.003	0.006 ± 0.002	0.027 ± 0.001	0.0191 ± 0.002
720	0.176 ± 0.009	0.020 ± 0.001	0.009 ± 0.001	0.010 ± 0.002	0.034± 0.001	0.0251 ± 0.002

C.1.4.4 Forecasting for previsouly unseen time series

Setting and baseline. As mentioned in section 5.2.4.2, most forecasters explicitly model the dependencies between samples, which limits their ability to generalize to new time series without retraining the entire model. However, TimeFlow, PatchTST, and DeepTime have the advantage of being reusable for new samples. In table C.14, we present the results of TimeFlow and PatchTST for new periods, considering both known samples and new samples. We train TimeFlow and PatchTST on 50 % of the samples and consider the remaining 50 % as the new time series.

Table C.14: MAE results over horizon for the forecasting task in the context of generalization to new time series.

	H	Known time series		New time series	
		TimeFlow MAE error	PatchTST MAE error	TimeFlow MAE error	PatchTST MAE error
Electricity	96	0.228 ± 0.023	0.211 ± 0.007	0.241 ± 0.023	0.224 ± 0.020
	192	0.244 ± 0.022	0.225 ± 0.014	0.254 ± 0.024	0.238 ± 0.024
	336	0.269 ± 0.036	0.267 ± 0.019	0.277 ± 0.033	0.285 ±0.005
	720	0.331 ± 0.058	0.310 ± 0.026	0.333 ± 0.059	0.331 ± 0.045
Traffic	96	0.226 ± 0.035	0.208 ± 0.036	0.222 ± 0.031	0.203 ± 0.037
	192	0.217 ± 0.028	0.202 ± 0.029	0.215 ± 0.026	0.199 ± 0.030
	336	0.242 ± 0.036	0.228± 0.041	0.240 ± 0.031	0.224± 0.036
	720	0.283 ± 0.053	0.275 ± 0.059	0.283 ± 0.049	0.272± 0.055
SolarH	96	0.237 ± 0.077	0.256 ± 0.055	0.236 ± 0.081	0.256 ± 0.062
	192	0.238 ± 0.051	0.251 ± 0.239	0.239 ± 0.058	0.250 ± 0.050
	336	0.220 ± 0.027	0.255 ± 0.663	0.220 ± 0.034	0.255 ± 0.066
	720	0.240 ± 0.039	0.267 ± 0.062	0.240 ± 0.042	0.267± 0.063

Results table C.14 demonstrates the good adaptability of both methods to new samples, as the difference in MAE between known and new samples is marginal.

C.1.4.5 Influence of the look-back window for forecasting

In fig. C.6, it is shown that both excessively short and overly long look-back windows can harm TimeFlow forecasting performance. More precisely, the performances increases with the look-back window size up to a certain size, where the performances then drop slowly.

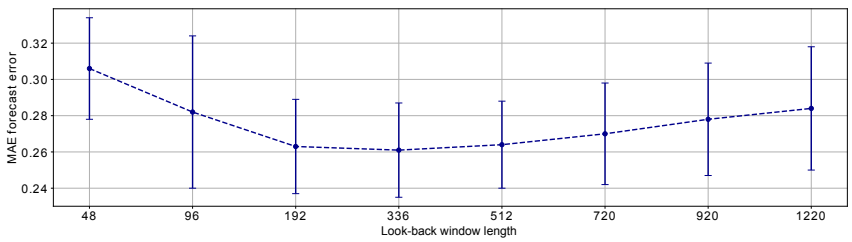


Figure C.6: MAE forecast error per look-back windows length for the *Electricity* dataset (horizon window length is 336). The model is trained on a given time window and tested on four new time windows.

C.1.4.6 Influence of the horizon length for forecasting

In fig. C.7, it is shown that the performances decrease with the length of the horizon. This is to be expected, since the longer the horizon, the harder the task.

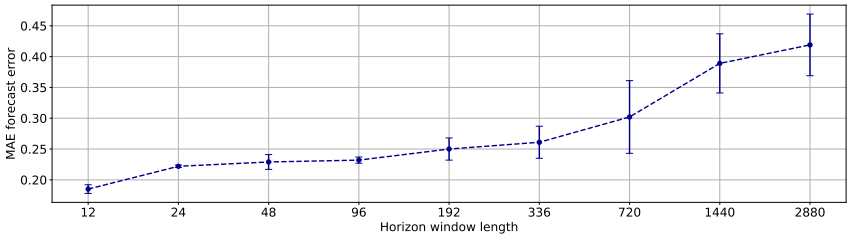


Figure C.7: MAE forecast error per horizons length for the *Electricity* dataset (look-back window length is 512). The model is trained on a given time window and tested on four new time windows.

Glossary

C | D | F | G | I | M | O | P | R

C

CFD Computational Fluid Dynamics. 95

CNN Convolutional Neural Network. 20, 21, 28, 43, 95

D

DL Deep Learning. 2, 4–7, 19–24, 28, 29, 31, 35, 36, 42, 44

F

FNO Fourier Neural Operator. 29, 30, 50, 51

G

GNN Graph Neural Network. 22, 31

GPU Graphic Processing Unit. 2, 95

I

INR Implicit Neural Representation. 5, 6, 31, 32, 94, 96–100, 102–108, 118

M

MAE Mean Absolute Error. xvi–xix, 66–71, 73–75, 78–80, 83–87, 89–92, 111, 114, 115, 142, 145, 150, 154–160, 162, 166, 167

MGNO Mutlipole Graph Neural Operator. xv, 30, 35–37, 39, 40

MLP Multi-layer Perceptron. xv, 20, 31, 35, 39–41, 99, 101

MSE Mean Squared Error. 22, 49–51, 66, 72, 99–101

O

ODE Ordinary Differential Equations. 7–17, 24, 54, 55, 58, 59, 64–66, 81, 86

P

PDE Partial Differential equation. xv, xvi, xviii, 1, 3, 4, 54, 56, 57, 61, 62, 64, 65, 86, 87, 90, 92

R

RMSE Root Mean Squared Error. 70

Deep neural networks and partial differential equations

Abstract

The study of physical systems, modeled by partial differential equations (PDEs), represents a cornerstone of scientific research. These equations, describing the relation between some function and its partial derivatives across variables, are vital for modeling diverse phenomena, for e.g. fluid dynamics and heat transfer, with applications in various domains such as climate science and astronomy. In the recent years, data has become readily available, explaining partly the rise of data-driven methods and more particularly Deep Learning (DL) methods. They excel in training complex models on a large amount of data, while being computationally effective at inference. However, for physical systems, even with apparently large amount of data, data is often scarce compared to the complexity of the problems, which is a challenge for DL methods. More importantly, the problems faced when applying DL methods to physical systems are very different from the usual DL problems, with physical problems potentially being be ill-posed, chaotic or very sensitive to initial conditions. In addition to these main challenges, practitioners in numerical analysis or the industry seek theoretical or experimental guarantees of convergence, which DL methods can lack of.

In this thesis, we tackle some of these challenges through three distinct approaches. In the first part of this work, we apply concepts from numerical analysis into DL frameworks, offering two perspectives: (i) the incorporation of multigrid numerical schemes into a Multi-Scale DL architecture, the Multipole Graph Neural Operator, demonstrating its efficacy in solving steady-state Darcy flow and 1D viscous unsteady Burgers' equations (ii) the adaptation of implicit numerical schemes into neural networks, ensuring forecasting stability for dynamical systems via some constraints on the weights of the neural network, leading to improved long-term forecasting results for two transport PDEs. In the second part of this work, we design a hybrid model to address the friction law design in Shallow-Water equations. This implies learning the friction law from observations through a numerical solver. The experiments focus on a vast analysis of the robustness and convergence for a stationary case and confirm the efficacy on the dynamic case. In the third part of this work, we explore continuous methods through two works based on Implicit Neural Representations (INRs): (i) INFINITY is a INR based method that can be applied to static PDE problems. It is tested on the RANS equations for surrogate modeling of airfoils. INFINITY can accurately infer physical fields throughout the volume and surface, leading to a correct prediction of the drag and lift coefficients, which are crucial for airfoil design. (ii) TimeFlow is a general framework using INRs to impute and forecast time series. By its continuous nature, TimeFlow can handle missing data, irregular sampling and unaligned observations from multiple sensors while having similar performances to state-of-the-art algorithms and being able to generalize to unseen samples and time windows.