University of Amsterdam

Masters Thesis

An inverse problem approach for closure modelling

Author:
Viviane Desgrange

Supervisor:
Dr. Tristan van Leeuwen
Examiner:
Prof. dr. Daan T. Crommelin
Second assessor:
Syver D. Agdestein

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Computational Science

in the

Centrum Wiskunde & Informatica
Informatics Institute

December 2022

Declaration of Authorship

I, Viviane Desgrange, declare that this thesis, entitled 'An inverse problem approach for closure modelling' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at the University of Amsterdam.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

[illegible]

Date: 5 December 2022

UNIVERSITY OF AMSTERDAM

Abstract

Faculty of Science
Informatics Institute

Master of Science in Computational Science

An inverse problem approach for closure modelling

by Viviane DESGRANGE

Most of the real-world physical phenomena can be described by mathematical models governed by partial differential equations (PDE). These differential equations might be too complex to be solved analytically, therefore requiring advanced computational capabilities to be solved numerically. This defines the field of Scientific Computing. The main drawbacks lies in the fact that these Full-Order Models (FOM) require large range of parameters, solved on high-dimensional grid and using small time steps to performed accurate simulations able to capture the sharpest details, such that the computational resources involved are a bottleneck on large-scale application. Multiples methods of Model Order Reduction (MOR) were implemented to avoid these difficulties. Instead of solving the FOM, these are Reduced order models (ROM), able to capture the main dynamics while reducing the complexity of the system, which are numerically solved. These ROM usually trade accuracy for time complexity and require a closure term to capture the information loss which occurred when reducing the system. In the past decade, the progress in the field of machine learning brings to light new techniques either to learn these closure terms or to learn new type of ROM, such as the right-hand side of a PDE.

This thesis, beside reviewing the existing MOR methods, aims at assessing some of the machine learning techniques which rise in the recent years applied to simple linear and non-linear PDEs on coarse grid. It investigates how models should be adapted to the problem considered, how to parameterize them to accurately approximate the FOM, with an emphasis on the inverse problem aspect of the methods employed during training. It appears that pure neural network without prior knowledge of the model are able to learn the high-level dynamical aspects of the FOM with a similar if not higher accuracy than the baseline ROM models. The closure models which use a neural network as a closure term achieve even higher accuracy. Additionally, continuous models appear to slightly surpass discrete models. However, these results drastically drop when they must learn model dynamics such as discontinuities or high frequencies.

Eventually, this thesis investigated usability of ML models when applied to the Proper Orthogonal Decomposition (POD-ROM) methods which drastically reduce the system complexity to the number of modes composing the reduced basis. The method is generalised from a single advection shock example to a set of snapshots.

Hence, while this thesis only covers a small proportion of the ML models shows that the latest advancement brought by deep learning methods to reduced order modeling achieve hopeful results, not only for the learning of small closure term but also for the inference of complete reduced order models.

Acknowledgements

I would like to thank the following people for the help they bring in this project.

Tristan van Leeuwen, who originally suggested this project and agreed to supervise this Master thesis, for his advice and helpful criticism throughout this thesis project.

Syver Agdestein, second assessor of this thesis, whose numerous advice helped me deepen my understanding of the subject and overcome some of the challenge encountered.

The Computational Imaging and Scientific Computing groups of the Centrum Wiskunde & Informatica, whose members contributed to make a friendly and motivating work environment.

My family, for the encouragement and support they have given me since 2019 when I decided to apply to this Master.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

AD	**A**utomatic **D**ifferentation
ADAM	**A**daptive **M**oment **O**ptimization
BC	**B**oundary **Conditions**
BVP	**B**oundary **V**alue **P**roblem
CAE	**C**onvolutional **A**uto **E**ncoder
CNN	**C**onvolutional **N**eural **N**etwork
DFT	**D**iscrete **F**ourier **T**ransform
DNS	**D**irect **N**umerical **S**imulation
DO	**D**iscretize-then-**O**ptimize
FD	**F**inite **D**ifferences
FE	**F**inite **E**lements
FFT	**F**ast **F**ourier **T**ransform
FNN	**F**eedforward **N**eural **N**etwork
FOM	**F**ull-**O**rder-**M**odel
FRNN	**F**eedforward **N**eural **N**etwork
LSTM	**Long**-**S**hort **T**erm **M**emory
NIROM	**N**on-Intrusive **R**educe **O**rder **M**odeling
NO	**N**eural **O**perators
NODE	**N**eural **O**rdinary **D**ifferential **E**quations
OD	**O**ptimize-then-**D**iscretize
ODE	**O**rdinary **D**ifferential **E**quation
PCA	**P**rincipal **C**omponent **A**nalysis
PDE	**P**artial **D**ifferential **E**quation
PDF	**P**robability **D**ensity **F**unction
PINN	**P**hysic-**I**nformed **N**eural **N**etwork

POD	**P**roper **O**rthogonal **D**ecomposition
ResNet	**Res**idual **Net**work
RK4	Classical **4**th-order **R**unge-**K**utta method
ROM	**R**educe-**O**rder-**M**odel
ROR	**R**educe-**O**rder-**R**epresentation
SGD	**S**tochastic **G**radient **D**escent
SVD	**S**ingular **V**alue **D**ecomposition
Tsit5	**Tsit**ouras **5**/4 Runge-Kutta method

Nomenclature

$u_t \equiv \frac{\partial u}{\partial t}$	partial derivatives
$u_{xx} \equiv \frac{\partial^2 u}{\partial x^2}$	partial derivatives
w	weights
b	bias
θ	neural network parameters
Φ	basis
Φ_r	reduced basis
$\sigma(x)$	sigmoid function
$\tanh(x)$	hyperbolic tangent function
$\mathcal{L}$	loss function
$C(w, b)$	cost function
F	exact analytical forward operator
F_θ	approximated operator
f_{fom}	full order model operator
f_{rom}	reduced order model operator
$u(t, x)$	scalar field value at spatial point x and time t
$u(t)$	discrete vector field value at time t
ν	diffusion coefficient (viscosity)
Re	Reynolds number

Chapter 1

Introduction

1.1 Background

1.1.1 Closure model and model order reduction

This section will provide a brief introduction to the closure modeling term and its relation with Reduced Order Model (ROM) techniques. Most of the phenomena in physics, chemistry, biology, etc. Can be described by mathematical models governed by partial differential equations (PDEs)

$$\frac{\partial u}{\partial t}(t,x) = F\left(t, x, u, \frac{\partial u}{\partial x_i}, \frac{\partial^2 u}{\partial x_i \partial x_j}, ...\right) \tag{1.1}$$

The main drawback of these differential equations lie in the equations being usually too complex, involving a large range of parameters or a system of multiple PDEs, to be able to compute an exact solution. However, it is possible to find an approximate solution to the underlying PDE system by making usage of numerical models. Such models are named full-order model (FOM) and usually of high dimensions. The main drawback of these models lie in the equations being usually too complex, involving a large range of parameters or system of multiple PDEs, to be able to perform a computationally efficient direct numerical simulation (DNS) at a large or industrial scale (i.e. an adequate time and memory consumption). Consequently, reducing the computational complexity of these mathematical models through model order reduction techniques appears as a requirement.

The reduced order models (ROM) aim at approximating the solution $u(t,x)$ by $u(t)$ while maintaining the dynamics of the system. It can consist in reducing the number of ODEs in the system by retaining a limited number of bases or solving the system on a coarse grid. A reduction will trigger a loss of information, this means that the approximation $u(t)$ is composed of an unclosed and a closed term. The closed term corresponds to the dynamics of the system obtains from $u(t)$, while the unclosed term corresponds to the loss dynamics, discarded by the reduction method. This brief definition show how one of the research area of reduced order model encompasses closure models which have been a fundamental discipline studied in physical phenomena modeling; and, in the context of this thesis, the fluid dynamics phenomena.

As it will be further described in the subsequent sections; model order reduction methods were subject of research for several decades, and can be classified nowadays in different categories including proper orthogonal decomposition methods, reduced basis methods, balancing methods, simplified physics and the nonlinear manifold methods. Additionally, the last two decades saw the emergence of data-driven reduced order modeling techniques. While it won't be possible to cover the large range of methodologies which emerged from the decades of research, this thesis picked a few relevant methods to describe and used as a basis to our inverse problem approach of closure modeling.

1.1.2 Inverse problem approach and optimization problem

We defined the model order reduction methodology, its relation to closure modeling and a preview of some of the known approaches.
However, this thesis does not only focus on the studies and development of a closure model applied to a specific physic phenomenon but also on how to infer an approximated model F_θ of the PDE representing the phenomena studied, from partial measurements (snapshots) of the solutions of the PDEs, a problem which can be defined as a PDE-constrained optimization problem

$$\min_\theta \mathbb{E}_{u_0\sim\pi_0} \int_0^T \|u_\theta(t) - u(t)\|^2 \mathrm{d}t \quad \text{s.t.} \quad \frac{\partial u_\theta}{\partial t} = F_\theta(t, u_\theta),\ u(0,x) = u_\theta(0,x) = u_0(x) \tag{1.2}$$

where $u(t)$ is a known snapshot obtain numerically or through experimental measurements of the time-dependent solution defined on a regular domain $\Omega \in \mathbb{R}^n$ *s.t.* $u \ : \ \Omega \times [0,T] \to \mathbb{R}^n$. F_θ, parametrized by θ, is the operator used in the simplified model f_{rom} which try to approximate the full-order model f_{fom}. $u_\theta(t) = G(u_0)(t)$ is the approximated solution obtained by solving the differential equation $u_t = F_\theta(t,u)$ constrained by initial boundary values and condition u_0.

Questions that arise are how can we parameterize the operator F_θ?
A brief enumeration of the possible approaches to reconstruct the operator contains neural differential equations (Neural ODE, Neural DDE, etc.), neural operators, physics-informed neural networks (PINN), inference of non-linear dynamic properties [BPK16] or analytical reconstruction. A more detailed account of these approaches and the selected methodology will be explained in the literature overview. Most of the methods listed enter in the category of the data-driven approach which can be closely related to inverse problems such as studying the well-posedness of the PDE-constrained optimization problem.

1.2 Research questions

Overall, the main goal of this thesis consists in studying data-driven methods to parameterize the closure model operator F_θ with consideration to the PDE-constrained optimization problem 1.2. The inverse problem aspects is evaluated with regards to the numerical methods used by these data-driven approach.

The first objective aim at reviewing and bench-marking the recent ROM methods to improve our understanding of the topic.

The second objective consists in evaluating the efficiency of selected data-driven model reconstruction methods [LKA+20, Kid22, Cal20, GBC16], precisely the neural ordinary differential equations, to simple linear and non-linear equations in comparison to their full-order model. The linear case focusing on the ability to exactly retrieve the exact operator F, rather than strictly focusing on the accuracy of the prediction, even if these topics are closely related. In the non-linear case, this results in the ability to recover dynamics of non-linear term, which can corresponds to high-frequencies, or discontinuities. In both cases, an other objective aim at looking at the efficiency of interpolation and extrapolation of the obtained closure model, in other words, how accurate our operator works when applied to unknown spacial data or prediction on time data. Such tasks will cover experiments to evaluate the influence of regularization and choice of architecture on these errors to improve the accuracy of the prediction.

The last objective aim at evaluating these same methods when applied to closure models such as down-scaling and POD problems. These case do not focus on the ability to learn the dynamics of the exact model when projected into a reduced space. This latest case is inspired by the work from Gupta and Kochkov [GL21, KSA+21].

In brief, the thesis' objectives are resumed through these questions, each of them affiliated to a chapter:

1. *How accurate is the operator F_θ retrieved through data-driven reconstruction methods?*
    - Which machine learning model should be considered to model the operator? Discrete neural network, neural differential equation, physic-informed neural network, neural operator...
    - How should be design the neural network architecture?
    - Is there a form of neural network applicable to all problems?
    - Which training procedure should be adopted to parameterize the operator F_θ?
    - How accurate are the predictions of the operators encompassing PDE right-hand side?
    - And when used as a closure term as a complement to a baseline model?
2. *How efficient are data-driven reconstruction methods to infer the dynamics properties of non-linear PDE?*
    - Can a same architecture be used on every parameterization of a PDE (i.e. Viscous and Inviscid Burgers equation)?
    - Are they able to learn low-scale dynamics such as discontinuities (shock in Inviscid Burgers) or high frequencies (Korteweg–De Vries)?
3. *Can data-driven methods be used to improve or accelerate existing closure models?*
    - On a simple down-scaling problem?
    - On a POD projection-based problem?

1.3 Overview

This section provides a summary of the topics covered by each thesis chapter.

The chapter 2 is dedicated to the literature review of closure modeling techniques and the data-driven reconstruction method brought in by the latest works. This chapter also covers the necessary notions and background required to understand and have a constructive criticism of the deep learning methods which were selected and applied through the experiments of this thesis.
In the chapter 3, the first experiments are performed on a linear diffusion equation. They provide insight on the inverse problem aspect of the data-driven reconstruction methods and give the first answers to their efficiency for approximating the operator used for prediction tasks.
The chapter 4 introduce the non-linear Burgers equation on which the experiments from the subsequent chapters will be performed. More importantly, it covers the down-scaling and POD problems whose baseline models will be used as a point of comparison to answer the remaining research questions listed above.
The chapter 5 directly extends the previous methodology-dedicated chapter. It investigates the choice of architecture to parameterize the modeled operators, study the efficiency to retrieve dynamics from a non-linear model on a down-scaled problem context. From this knowledge, the chapter then tries to evaluate the capacity of the data-driven approach to learn the loss information when the FOM complexity is drastically reduced through a Galerkin projection onto a reduced space.
Eventually, the chapter 6 resumes the discussions of the previous chapters and advises on the different directions from which this thesis work could be extended.

Chapter 2

Literature review

2.1 Introduction

This chapter covers the notions required to understand the closure modeling problems aspects tackled through this thesis. Firstly, it performs a review of the multiple reduced order modeling methods of the past decades, prior to focusing on the recent data-driven modeling approach brought in by the scientific literature. As the data-driven modeling approach is the principal subject of this thesis, this chapter does not limit itself to the high-level description of the method, it also elaborates further on the numerical aspect of this approach; thus, it provides some explanation the family of neural networks studied, the automatic differentiation and sensitivity analysis methods used to calibrate the neural network architectures and the optimization algorithms considered. It also introduces some of the inverse problems encountered, a topic extended by the subsequent chapter. We hope that the readers of this thesis whose background might only cover a sample of the necessary mentions will, through this chapter, obtain a better understanding of the relation between reduced order models, closure models, differential equations, deep learning approximation methods and dynamical systems.

2.2 Closure model and reduced order model review

As briefly discussed in the introduction, the model order reduction methods have been the main subject of research topics for decades as a mean of capturing the structure and dynamical behavior of underlying flows while reducing the computational cost of the *full-order models* (FOM):

$$\frac{\partial u}{\partial t} = f_{\text{FOM}}(u), \ \ u \in \mathbb{R}^{\mathbb{N}_x}. \tag{2.1}$$

Despite the computational cost saving they bring, *reduced order models* (ROM) remain a class of low-fidelity models. It explains the large extent of works done to obtain ROMs which would fit mimic as accurately as possible the *direct numerical simulation* (DNS). Consequently, since their introduction, the range of ROM approaches expanded greatly, going from the *first principle Galerkin* methods, through *closure* modeling and *data-driven dynamical system* identification, to the recent *physics-informed data-driven* models. The latter approach, data-driven modeling, only arose in the past decades with the emergence of machine learning techniques, but already brings great changes to this field. In this section, we give an overview of the methods listed above and highlight some of the techniques we implemented or made usage of as part of the operator reconstruction problems tackled in this thesis.

2.2.1 Reduced order representation

Let's consider the application of model reduction techniques to problems such as parameterized PDEs, differential equations depending on a set of parameters. *Reduced order representations* (ROR) should be seen as the most simple ROMs. They usually take advantage of the visible underlying and coherent structures of the problem to catch the main features of the model. They can be represented as

$$u(t,x) = \sum_{k=1}^{r} a_k(t)\varphi_k(x), \tag{2.2}$$

where x corresponds to spatial dimension, t time, $a_k(t)$ are the time dependent coefficients, $\varphi_k(x)$ the orthonormal basis functions (obtained using Fouriers series, resolution of eigenvalue problems, etc.), m the total number of basis functions and $r \ll m$ the number of basis retained. The simplest example is the *proper orthogonal decomposition* (POD) [L.67], also known as principal component analysis (PCA) or singular value decomposition (SVD). It aims at decomposing a arbitrary vector field $u(t,x) \in \mathbb{R}^{\mathbb{N}_x}$ representing the mathematical model evolution into a set of POD modes. The first POD modes (ordered by eigenvalues) are known to capture most of the energy from the original variable, in other words, the main dynamics of the system. See section 4 for further details. Considering that the POD modes are usually obtained from a high-fidelity DNS (time snapshots of the solution for the POD), it must be highlighted that the purpose of the ROR is to be able to represent the dynamics of the PDE for different sets of parameters. For instance, in the case of a linear problem [QMN15]

$$f(\mu) = \mathcal{L}(\mu)u(\mu) \tag{2.3}$$

with the linear operator $\mathcal{L} : V \to V'$, $f : V \to \mathbb{R}$ a linear transformation of V, parametrized by $\mu \in \mathcal{P}$ which correspond to any possible entries: it can be a source term f, a diffusion coefficient ν or initial conditions u_0 in the convection–diffusion equation.

This thesis cannot honor the full extent of projection-based ROR approaches available nowadays [SESO+21]. The POD-Galerkin projection (POD-GP), the empirical interpolation method (EIM) [BMNP04] (a modification of POD which reduces the complexity of computing non-linear term of the PDE), the discrete empirical interpolation method (DEIM) [CS10] (another variant able to reduce the dimension of a set of ODEs), and spectral POD [SPO16] are only a handful of the existing methods. Therefore, this thesis will restrain itself to implement the classical POD-GP, with the purpose of evaluating the efficiency of recent data-driven methods to closure modeling rather than the existing ROR approaches.

Now, we should highlight the main issues with the existing RORs approaches. The number of modes truncated and the accuracy it gives are only usable in the context of an academic problem (most publications are limited to a dozen of modes [MML+20]). They can hardly be used on large-scale applications considering the accuracy desired. Additionally, we should highlight that the majority of publications focus on small rather than high Reynolds numbers (the ratio of inertial to viscous forces within a fluid), making it less susceptible to turbulence. This causes the necessity to work on models for different viscous forces.

2.2.2 Closure modeling

As previously described, the usage of ROMs can gives the possibility of computing solutions independent of the original FOM attributes. In the example of a convection-diffusion equation, this signifies independence of the dimension n of the FOM, the diffusion coefficient ν, the initial conditions, etc. However if we consider a projection-based ROM, we know that the wave models are usually described by a slow decaying *Kolmogorov n-width* (the rate of error arising from projection on the best possible bases) [GU19]. Consequently, it might be necessary to maintain a large number of modes in order to not lose too much information about the system, which circles back to the computational resource consumption problem of FOM. This example leads to the next main approach to ROM and one of the main aspects of the thesis: closure models. They consist of determining the influence of unresolved components on the ROM dynamics obtained from the resolved term. Let's consider a ROM

$$\frac{\partial v}{\partial t} = f_{\text{ROM}}(v) + \epsilon(v) \tag{2.4}$$

where $v(t)$ is the solution to the ROM which try to approximate the solution $u(t) \in \mathbb{R}^{\mathbb{N}_x}$ to the FOM, $f_{\text{ROM}}(v)$ described the ROM resolved dynamics and $\epsilon(v)$ is the *closure term* in charge of providing the influence of the ROM unresolved dynamics (i.e. the discarded modes). We highlight the idea that the closure term $\epsilon(v)$ applies to any type of model error; for instance it will model the influence of the sub-grid scale dynamics only described on a fine grid, for methods such as finite differences taking place on a coarse grid $f_{\text{coarse}}(v)$.

$$\frac{\partial v}{\partial t} = f_{\text{coarse}}(v) + \epsilon(v), \tag{2.5}$$

where $v(t) \in \mathbb{R}^{\mathbb{N}_d}$, $\mathbb{N}_d \ll \mathbb{N}_x$, $\epsilon(v) = f_{\text{downscale}}(v) - f_{\text{coarse}}(v)$. In another instance where the ROM uses a reduced number of POD basis ϕ_r, the closure term models the influence of the discarded basis functions ϕ_d:

$$\frac{\partial v}{\partial t} = f_{\text{POD}}(v) + \epsilon(v) = \phi_r^{\intercal} f_{\text{FOM}}(\phi_r v) + \epsilon(v), \tag{2.6}$$

where $v(t) = \phi_r^{\intercal} u(t)$, $\phi_r \in \mathcal{X}^r = \text{span}\{\phi^1 ... \phi^{N_r}\} \in \mathbb{R}^{\mathbb{N}_r \times \mathbb{N}_x}$ and $\mathbb{N}_r \ll \mathbb{N}_s$ is the number of basis functions.

Having discussed the definition and a few instances of closure models, it is important to address the fact that multiple approaches exist to model the closure term (an aspect discussed in the subsequent section).

1. Structural closure modeling focuses on the analytical aspects;
2. Stochastic closure modeling try to avoid the deterministic aspects previous methods have;

3. Data-driven closure modeling takes advantage of data-sets obtained by solving the FOM to improve the ROM.

2.3 Data-driven modeling

DNS is computationally expensive and ROM drops accuracy in the resolution of the governing equations for large scale modeling problems. Implicit and explicit methods used by numerical solvers require fine-grid discretization and solving a high-dimensional equation to be able to provide accurate results, making it a challenge to simulate physical phenomena such as fluid flows on a large scale. However, the latest data-driven modeling approaches to ROM brought by the state-of-the-art methods in the machine learning field offer a new range of methods to tackle these problems. Such methods make use of data to learn how to predict the solutions to the modeling problems rather than to solve them directly, and have demonstrated great improvements in computational efficiency.

Used both for operator inference and closure modeling, some of the latest works focus on retrieving the unknown dynamics of the system [BPK16]. Others showed the efficiency of using deep learning to determine the closure term required to represent the sub-grid-scale effects on coarse grid models [PMK+20] or the influence of discarded modes [MML+20, MLB21, GL21] in fluid dynamics reduced models. Fully data-driven methods have been used to determine PDE solvers for similar applications [KSA+21]. Other domains come into the equation given that these methods were also considered with an emphasis on speed rather than accuracy (beyond the capacities of standard closure modeling techniques) as a way to perform fast simulations [KAT+19].

Logically, the drawback of such approaches is based on the methodology to obtain training data of high quality. Data of poor quality leads to bad predictions in addition to having to use expensive numerical solvers to generate then. The generalization problem also arises from the deep learning method used, as it is not clear whether the trained neural networks can extrapolate for other parameters. Additionally, a distinction must be done between regular neural networks, neural differential equations (NODE, NDDE, etc.) [Kid22], physic-informed neural networks (PINNs), neural operator, etc. As such, this section tries to provide to the reader the necessary notions to some of the data-driven methods used in this thesis. This thesis will put emphasis on the methods applied to the retrieval of unknown dynamics. A fast review of the techniques is provided here, postponing the details of the actual implemented procedures to the next section.

Operator inference Operator inference can be defined as the deduction of ROM operators from data. The initial definition referred to using machine learning techniques as a non-intrusive method to determine the linear and non-linear operators [PW16] [YGBK21] composing a ROM (for instance in the set of equations obtains from POD-GP 4.30) instead of having to explicitly compute them. Logically, this approach can be generalized to learn continuous direct models or closure terms; and subsequently on the choice of numerical methods. Convolutional and residual neural networks [KSA+21] were used to successfully model two-dimensional turbulent flows based on the Navier-Stokes equations. The experiments resulted in a solution as accurate as finite-difference and finite-volume based DNS with a $10\times$ finer resolution and a $40-80\times$ computational speed-up. Similarly, [MBO21] used a deep convolutional auto-encoder (CAE) to learn the ROR and a long-short term memory (LSTM) to learn the time evolution of an advection problem. Pure feedforward neural networks [HD20] and recurrent neural networks [WRH19] were also used in this intention and generated encouraging results. Neural networks were also used to tackle problems related to non-intrusive non-linear reduced order modeling (NIMOR), which usually requires an intrusive approach. The main idea consists of learning the dynamics of the system in a reduced space rather than from the high-dimensional FOM. Lee [LC20] uses a CAE to compute an approximation of the non-linear manifold reduced order model applied to advection problems such as the one dimensional Burgers equation. Kneifl [KGF21] applies a similar idea, using a neural networks to learn the reduced basis obtained from the POD-Galerkin projection.

Neural Operator Finally, one of the latest approaches is known as *neural operators* (NO). While the previous methods focus on learning mappings between finite-dimensional spaces, neural operators try to learn mappings between continuous spaces, which can be seen as infinite-dimensional operators. The main advantage of the neural operator is its resolution invariance. The neural operator need to be trained once and can be applied to any grid resolution, while the usual neural network (finite-dimensional operator) will only be able to determine the solution to a PDE on a finite dimensional space of same size as it was trained on [LKA+20].

Physics-informed neural network Scientific machine learning approach to PDEs requires the acquisition of trust-worthy data of the evaluated equations, but also the knowledge of its underlying physic. While the neural networks are used for the capacity to learn non-linearity of the evaluated models, the presence of the non-linear terms in these equations remains a challenge and they require deeper neural network architecture as the complexity of the PDEs increase. From there, raise the necessity of physics-informed neural networks (PINN) [RPK19] which directly model the solution u by integrating some knowledge of the PDE such as conservation laws (Mass-energy, momentum, etc.), terms of the PDE itself, initial conditions. Let's consider the solution u to the Bateman-Burgers equation:

$$u_t = uu_x - \nu u_{xx} \tag{2.7}$$

The PINN $f(t, x; \theta)$ is obtained by replacing the variable $u(t, x)$ with a neural network $\mathrm{NN}(t, x; \theta)$ and using the automatic differentiation (AD) 2.4.3 method to enforce the PDE form:

$$f(t, x; \theta) = \frac{\partial}{\partial t}\mathrm{NN}(t, x; \theta) - \mathrm{NN}(t, x; \theta)\frac{\partial}{\partial x}\mathrm{NN}(t, x; \theta) + \nu\frac{\partial^2}{\partial x^2}\mathrm{NN}(t, x; \theta) \tag{2.8}$$

When training the PINN, the optimisation problem is defined by the minimisation of a loss function composed the difference between $u(t, x)$ and $\mathrm{NN}(t, x; \theta)$, and the constrained of the PDE system.

Neural differential equations More recently, a new class of data-driven methods have emerged in the scientific computing field. Neural differential equations [CRBD18] [Kid22] (known in their simplest form as Neural ODEs) consist of using a differentiable equation solver as a central element of the computational graph used to train the neural network.

This section has attempted to provide a brief summary of the closure modeling with an emphasis on the data-driven modeling approaches to ROM. With regards to the latest approach, early experiments reviewed different form of neural network, but the work performed in this thesis eventually focused on the Neural ODEs.

2.4 Notions on Neural Ordinary Differential Equations

As a first approach, we consider the neural ordinary differential equations (NODE) to parameterize the closure model operator F_θ with consideration to the PDE-constrained optimization problem. A recent addition to the list of neural network architectures. Neural ODE can be defined as a differential equation using a neural network to parameterize the operator F_θ [CRBD18]:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = F_\theta(u), \quad u(0) = u_0, \tag{2.9}$$

where $F_\theta : \mathbb{R} \times \mathbb{R}^{d_1 \dots d_n} \to \mathbb{R}^{d_1 \dots d_n}$ is the approximated operator parameterized by θ and $u : [0, T] \to \mathbb{R}^{d_1 \dots d_n}$ the solution. This operator covers the case of the PDE being approximated by a pure neural network architecture, and the case where neural network is used as a closure term $\epsilon(u)$.

In the case where a NODE would be use to learn a closure term, it can be affiliated with a residual neural network (ResNet). In general, NODEs can be considered as a continuous counterpart to discrete models such as recurrent neural networks (RNN), long short-term memory (LSTM) or gated recurrent unit (GRU). The definition of the neural network architecture is carried out analysing the terms of the model. Common choices are feed-forward (FNN) or convolutional neural networks (CNN). One of the advantages of the latest instance is to approximate methods such as finite differences. Similarly, if we were interested in including a latent term, neural delay differential equations (NDDE) could be considered.

Learning a NODE in the initial value problem (IVP) case consists of using a numerical ODE solver as part of the learning process:

$$u(t_1) = u(t_0) + \int_{t_0}^{t_1} F_\theta(t, u(t))\,\mathrm{d}t \tag{2.10}$$

The initial value $u_0 = u(t_0)$ is provided together with the parameterized operator F_θ to the numerical solver. The algorithm will learn F_θ when minimizing an objective function $\mathcal{L}$. While the idea is simple, we will discuss the multiple issues encountered and solutions can be derived from it in the following section.

2.4.1 Well-posedness in the sense of Hadamard

In the following section we discuss the inverse problem aspect of the NODE through a study of its well-posedness. We modify the notation in Equation (2.9) and use the general form

$$f = K(u), \tag{2.11}$$

where $K : \mathbb{U} \to \mathbb{F}$ the forward operator mapping the function $u \in \mathbb{U}$ to the function $f \in \mathbb{V}$ is the exact model operator we are trying to determine through F_θ. As we want to check if the problem 2.9 fulfill the requirements of a well-posed problem in the sense of *Hadamard*:

1. Existence: there exists a solution u s.t. $K(u) = f$
2. Uniqueness: the solution u is unique
3. Stability: the solution's depends continuously on the data.

In the IVP context, existence and uniqueness of the problem 2.9 is insured by the *Picard-Lindelöf theorem*, as long as its right-hand side is *Lipschitz* continuous.

Definition 2.1 (Lipschitz continuous)**.** A function $f : \mathbb{U} \to \mathbb{V}$ is Lipschitz continuous if $\exists C \geq 0$ s.t. $\forall (u_1, u_2) \in \mathbb{U}^2,\ ||f(u_1) - f(u_2)|| \leq C||u_1 - u_2||$.

Theorem 2.2 (Picard-Lindelöf theorem)**.** *If a function $f : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}^d$ is continuous in t and Lipschitz continuous in u, with $u_0 \in \mathbb{R}^d$; then there exists a unique differentiable solution $u : \mathbb{R} \to \mathbb{R}^d$ to the differential equation $\frac{\mathrm{d}u}{\mathrm{d}t} = f(t, u),\ u(0) = u_0$.*

From this demonstration we have a proof of the existence and uniqueness to the equation 2.9. Therefore, we can make usage of NODEs as a data-driven operator approximation method for problem such as PDEs.

2.4.2 Sensitivity Analysis

Forward pass Finding analytical solutions is not usually possible for non-linear differential equations such as *Navier-Stokes* equations. The usage of numerical solvers is necessary to generate data-sets for our models to train the NODEs parameterized by θ. For non-stiff equations (e.g. advection-diffusion equations), the main subject of study of this thesis, Runge-Kutta methods provide a set of implicit and explicit methods to solve our ODE. For the generation of the training, validation and test data, we use the *Tsitouras 5/4 Runge-Kutta method* (Tsit5) [Tsi11] and the *classical 4th-order Runge-Kutta* (RK4). Now, in the context of a neural ODE with complex architecture, we should consider other methods due to the numerical instability when solving *stiff* equations. RK4 or Tsit5 should still work on short term predictions, but the accumulated error will affect extrapolated predictions. Traditionally, implicit methods such as the *Rosenbrock* family of single-step methods are used. However, they are computationally expensive, requiring short time steps to guarantee numerical stability and accuracy; from it two issues arise in the training of the neural network. First, most implicit methods compute the Jacobian of the PDEs system once per iteration (e.g. using Newton's method), a task which is computationally expensive for neural networks. Second, the high number of time steps will trigger an evaluation of the neural network as well. Overall, during forward propagation the numerical solvers can be used to train the NODE to solve the IVP, each intermediate operation of the solvers is known.

Backward pass The main difficulty lies into the backward pass, considering that we need to be able to back-propagate through the ODE in order to obtain the gradients with respect to the parameters θ, as we need to be able to compute $\frac{\partial \mathcal{L}}{\partial \theta}$ to minimise the cost function. For instance, back-propagating through all the intermediate operations known of the forward pass has a memory cost which should be considered, on top of introducing numerical errors.

Multiple approaches are available for the back-propagation step. *Discretize-Optimize* (DO) This straightforward approach is the main method used to train the neural network. As a result of the numerical solver performing continuous and differentiable operations, it is possible perform the back-propagation step through all the operations of the forward propagation. Discretize-optimize got this name by the idea that the optimization step (differentiation) is performed after the ODE is discretized in time (using an ODE solver). Such an approach is accurate and fast as it does not approximate the gradients, as the computational graph is fully known and *automatic differentiation* (AD) can be used. However, it requires a sub-optimal memory usage since it is necessary to store the intermediate operations of the numerical solver on top of introducing additional numerical errors. Additionally, we notice that training a continuous model like NODE using the discretize-optimize approach along with a the Forward Euler method, is equivalent to using the NODE as the difference between each snapshots of a differential equation solutions, that is

$$u(t+\Delta t) = u(t) + \Delta t\,(F_\theta(t, u(t))) \tag{2.12}$$

Optimize-Discretize (OD) is the next popular approach we consider in this work, also known as *continuous adjoint method.* Instead of back-propagating through the numerical solver, gradients are computed by solving an other ODE backward in time, also known as the *adjoint sensitivity method*[Pon87].

2.4.2.1 Backsolve adjoint method

The backsolve adjoint method is the original proposition made by [CRBD18], solving an ODE backward in time allow a memory complexity of $O(1)$.

Algorithm 1: Backsolve adjoint method for computing derivatives of an ODE Problem

```
Require: $\theta$, $t_0$, $t_1$, $u(t_1)$, $\frac{\partial \mathcal{L}}{\partial u(t_1)}$
1: function F$(u(t), a(t), t, \theta)$                                   ▷ ODE System to solve
2:     $\frac{\mathrm{d}u}{\mathrm{d}t} = F_\theta(t, u(t))$
3:     $\frac{\partial a_u}{\partial t}(t) = -a_u(t)^\intercal \frac{\partial F_\theta}{\partial u}(t, u(t))$
4:     $\frac{\partial a_\theta}{\partial \theta}(t) = -a_u(t)^\intercal \frac{\partial F_\theta}{\partial \theta}(t, u(t))$
5: end function
6: $s_0 = [u(t_1), \frac{\partial \mathcal{L}}{\partial u(t_1)}, 0]$                          ▷ Initial conditions
7: $u(t_0)$, $\frac{\partial \mathcal{L}}{\partial u(t_0)}$, $\frac{\partial \mathcal{L}}{\partial \theta}$ = SolveODE$(F, s_0, [t_1, t_0], \theta)$        ▷ Backward pass
8: return $\frac{\partial \mathcal{L}}{\partial \theta}$, $\frac{\partial \mathcal{L}}{\partial u(t_0)}$                                  ▷ Computed gradients
```

Consider the loss function $\mathcal{L}$. We try to minimize it with respect to the model parameters θ by computing $\frac{\partial \mathcal{L}(u(T))}{\partial \theta}$. Let's consider the solution to the ODE $u(t) : [t_0, t_1] \to \mathbb{R}^d$:

$$u(t_1) = u(t_0) + \int_{t_0}^{t_1} F_\theta(t, u(t))\,\mathrm{d}t, \tag{2.13}$$

$$\mathcal{L}(u(t_1)) = \mathit{ode\_solver}(F_\theta, u_0, t_0, t_1), \tag{2.14}$$

$$u(t_0) = u_0. \tag{2.15}$$

The adjoint method checks how the gradient of the loss function depends on the hidden states $u(t)$. The dynamics of the adjoint variable $a(t) = \frac{\partial \mathcal{L}}{\partial u(t)}(t)$ are studied through its gradient:

$$\frac{\partial a_u}{\partial t}(t) = -a_u(t)^\intercal \frac{\partial F_\theta}{\partial u}(t, u(t)), \quad a_u(t_1) = \frac{\partial \mathcal{L}}{\partial u(t_1)}, \tag{2.16}$$

$$\frac{\partial a_\theta}{\partial \theta}(t) = -a_u(t)^\intercal \frac{\partial F_\theta}{\partial \theta}(t, u(t)), \quad a_\theta(t_1) = 0. \tag{2.17}$$

By calling the numerical solver, $\frac{\partial \mathcal{L}}{\partial u_0}$ can be computed from $a(t_1)$, and the required values of $u(t)$ can easily be computed backward in time. Finally, $\frac{\partial \mathcal{L}(u(T))}{\partial \theta}$ can be computed through a last integral:

$$\frac{\partial \mathcal{L}}{\partial \theta} = -\int_{t_1}^{t_0} \frac{\partial a_\theta}{\partial \theta}(t)\, \mathrm{d}t. \tag{2.18}$$

Automatic differentiation can be used to compute $a_u^\intercal \frac{\partial F_\theta(u)}{\partial u}$ and $a_u^\intercal \frac{\partial F_\theta(u)}{\partial \theta}$, this increase the time complexity in favor of an improved memory complexity and numerical error in comparison to the DO approach.

Unfortunately, backward solutions to non-linear differential equations may not be stable. This is enhanced in the case of stiff equations. This can be explained by the ill-posedness of multiple ODEs when solved backward in time. We refer to the chapter 3.2, which analyse the ill-posedness of an inverse problem applied to the linear diffusion equation $u_t = \kappa u_{xx}$, to get a demonstration of the unstability and difficulties brings by back-propagation.

Some implementations such as the Julia library *DifferentialEquations.jl* [RN17] introduce checkpoints during back-propagation where the backward resolution of the ODE is reinitialized to avoid these issues. The memory complexity is barely increased $\mathcal{O}(1 + n_{\text{checkpoints}})$, however this trade-off leads to an accumulation of numerical errors in the gradient computation, requiring very small time steps for stiff equations.

2.4.2.2 Interpolating adjoint method

Since the work from Chen [CRBD18], several improved methods have been introduced. Using a similar idea, the interpolating adjoint method proposed by Rackauckas [RMM+20] slightly increases memory complexity in favor of stability. It consists of solving the ODE in reverse using interpolation (while the backsolve adjoint method does not). Snapshots (u_i, t_i) of the forward propagation are used for check-pointing during the backward propagation such that the ODE is solved for each interval $t \in [t_{i-1}, t_i]$. Given that, the backward propagation requires memory to hold the interpolation between the two checkpoints, and the two associated forward solutions of the ODE. The stability of this method make it preferable when training neural ODEs.

The OD methods introduced above will be used in the experiments of this work to learn closure models. Other approaches exist (e.g. reversible solvers) however in this thesis we will not focus on them.

Algorithm 2: Interpolating adjoint method for computing derivatives of an ODE Problem

```
Require: $\theta$, $t_0$, $t_1$, $u(t_1)$, $\frac{\partial \mathcal{L}}{\partial u(t_1)}$
1: function F($u(t)$, $a(t)$, $t$, $\theta$)                                ▷ ODE System to solve
2:     $\frac{\partial a_u}{\partial t}(t) = -a_u(t)^\intercal \frac{\partial F_\theta}{\partial u}(t, u(t))$
3:     $\frac{\partial a_\theta}{\partial \theta}(t) = -a_u(t)^\intercal \frac{\partial F_\theta}{\partial \theta}(t, u(t))$
4: end function
5: $s_0 = [u(t_1), \frac{\partial \mathcal{L}}{\partial u(t_1)}, 0]$                         ▷ Initial conditions
6: $u(t_0)$, $\frac{\partial \mathcal{L}}{\partial u(t_0)}$, $\frac{\partial \mathcal{L}}{\partial \theta}$ = SolveODE($F, s_0, [t_1, t_0], \theta$)
7: return $\frac{\partial \mathcal{L}}{\partial \theta}$, $\frac{\partial \mathcal{L}}{\partial u(t_0)}$                          ▷ Computed gradients
```

2.4.3 Automatic Differentiation

In the previous section we mentioned automatic differentiation (AD), which, as its named indicates, is a set of methods enabling us, from computing the numerical values of a function, to automatically compute numerical values of the derivatives of this function with the same accuracy as if we specified the derivative function itself by hand. Thus AD must not be confused with the computationally expensive finite differences method (which is also subject to round-off errors), nor with symbolic differentiation which aims at determining the analytical expression for the derivative of the function, a method known for its limitations to small equations. This explains the recent development and heavy usage of AD, considering the importance of derivatives in sensitivity analysis, inverse problems and simulation. AD is based on two principle ideas:

1. A complicated function can be described as a sequence of elementary operations and function applications (addition, multiplication, exp, cos, log, etc.) with known derivatives.
2. The derivatives of a composed function is equal to the composition of the derivative of the composing functions, also known as the *chain rule* (CR).

Forward accumulation In existing libraries, we encounter two types of AD: forward and reverse accumulation (differentiation), depending on the direction in which the chain rule is be applied [BBBCD00]. Consider the equation $y = f(u)$ and its Jacobian $J = \frac{\partial y}{\partial u}$. Forward accumulation (FA) consists of specifying the independent variable on which differentiation is performed and compute the derivative of each sub-expression recursively, where the sub-expression are represented by the independent variable v augmented by $\dot{v} = \frac{\partial v}{\partial x}$. Following the chain rule from inside to outside, it will define $\dot{v_i} = \sum_{j \prec i} \frac{\partial v}{\partial v_j} \dot{v_j}$.

Reverse accumulation Consider the equation $y = f(u)$ and its Jacobian $J = \frac{\partial y}{\partial u}$. Reverse accumulation (RA) consists of specifying the dependent variable on which the differentiation is performed, the derivative being determined with respect to the outer function, rather than the inner function (as in FA). In this case, the variable v is considered through its adjoint $\bar{v} = \frac{\partial y}{\partial v}$. Following the chain rule from outside to inside, we define $\dot{v_j} = \sum_{j \succ i} \bar{v_i} \frac{\partial v}{\partial v_j}$.

The choice of automatic differentiation method depends on the size of the problem. Usually, RA is preferred to FA on functions $f : \mathbb{R}^n \to \mathbb{R}^m$ where $m < n$ and vice versa. In the specific case of deep learning, RA is preferred given that the training involves a gradient based minimization of a scalar error value, a task for which RA is more efficient than FA. We will not be implementing AD directly, however we use it through libraries in our experiments when performing NODE training (thus the necessity of this highlight). In Julia, there is a wide range of AD packages. Among them we use *Zygote.jl* which implements reverse accumulation using source-to-source code transformation for Vector-Jacobian products (VJP).

2.4.4 Note on network architecture for Neural differential equations

Existing literature shows that complex architectures are not necessary to obtain encouraging results with NODEs. A neural network trained to model the right-hand side of a PDE equation to be solved will not require as many parameters than a neural network trained to directly predict the snapshot $u(t)$ from u_0. Gupta [GL21] and Malik [MML$^+$20] used feed-forward neural networks (FNN) (see Appendix B) in their attempt to learn a neural network closure term for the POD-GP ROM applied to a case using Bateman-Burgers equation. The network was composed of 1-5 fully connected hidden layers with continuous activation functions (hyperbolic tangent, identity). The root mean square error based objective function decreased on both the training and validation data sets. We can also consider the architectures previously used in pure neural network cases [KSA$^+$21, PMK$^+$20, MBO21]. The general idea consists of taking the structure of the modeled PDEs into consideration when choosing the architecture, e.g. by using convolutional layers (see Appendix B) to mimic spatial derivatives of different orders, introducing additional channels to emulate non-linear terms, inter alia.

2.4.5 Objective functions

Regardless of the designed architecture used, the neural network training will consist of minimising the objective function $\mathcal{C}(\theta)$, such that $\theta^* = \arg\min \mathcal{C}(\theta)$ is the optimal model parameters for solving the initial value problem (IVP) with $\mathcal{C}(\theta) = \sum_i \mathcal{L}(u_i;\theta)$ the aggregation of all data batches. If we consider the cases of NODEs (2.9), there are two approaches for the initial choice of loss function $\mathcal{L}$, that is:

A-priori approach Also known as *derivative fitting*, compare the closure model F_θ directly to the time derivative of the ODE reference solution $\frac{\mathrm{d}u_{\text{ref}}}{\mathrm{d}t}$, rather than to the solution u_{ref} itself:

$$\mathcal{L}(u,t;\theta) = \left\| F_\theta(t,u(t)) - \frac{\mathrm{d}u}{\mathrm{d}t}(t) \right\|_2^2 . \tag{2.19}$$

Derivative fitting is accommodating in the fact that, given parameters θ, evaluating the cost function and its gradient via the closure model for pre-computed reference snapshots $u_{\text{ref}}(t)$ is fast, as compared to computing the entire resulting trajectories $u_\theta(t)$. Additionally, computing the gradient of the loss function $\mathcal{L}$ with respect to the parameters θ, $\frac{\partial \mathcal{L}}{\partial \theta}$, a required task for back propagation, can be obtained using the chain rule.

However, in the context of this thesis, reference data is obtained by solving known PDEs. It is therefore an elementary task to compute the reference time derivatives up to a specific order of accuracy (using finite differences, automatic or symbolic differentiation, etc.). This task might not be possible on reference data obtained from physical measurements, nor on stiff equations. In practice (e.g. section 4, it has also been observed than the a-priori approach leads to predictions diverging from the reference solution. If an operator F_θ approximates the time derivative correctly (a-priori correctness), the resulting solution trajectory might still not be correct (a-posteriori correctness).

A-posteriori approach Also known as *trajectory fitting*, this approach consists of using an ODE solver to obtain the predicted ODE solution trajectory u_θ for a given parameter vector θ and compare it directly to the reference solution trajectory u_{ref}. The main drawbacks of this approach reside in the computational resources involved in solving the ODE defined by F_θ and the difficulty of computing $\frac{\partial \mathcal{L}}{\partial \theta}$ when performing back-propagation through the ODE solver. This gradient can be computed using the optimise-discretise (OD) methods described in the section 2.4.2 such as backsolve or interpolating adjoint methods.

$$\mathcal{L}(u, t; \theta) = \|u_\theta(t) - u(t)\|_2^2 \tag{2.20}$$

It should be highlighted that several questions arise as we wonder which approach would suits the best for accurate model reconstructions in IVP resolution. Depending on the problem modelled, it might be preferable to use the derivative over trajectory fitting approach. With regards to the drawback of these methods, a two-steps combination of these methods could be considered using *early stopping* to avoid over-fitting in the derivative fitting case. Different variant of the loss function $\mathcal{L}$ (other than $\mathcal{L}_2$-norm) could be considered.

Chapter 3

An inverse problem approach through a linear diffusion equation

3.1 Introduction

This chapter offers an overview of the inverse problems encountered through this thesis. It focuses on the one dimensional linear diffusion equation (aka. Heat equation) [Fou09], an example simple enough to covers the problem of stability and ill-posedness which affect some of the methods used. These will help us gain insight on the data-driven operator approximation method applied in the subsequent chapters to the non-linear differential equations, such as the Bateman-Burgers [Bur48] and Korteweg–De Vries [DJG95] equations.

3.2 Diffusion equation

We start our evaluation of the data-driven operator approximation methods by considering the problem of stability with the one dimensional diffusion equation (aka. the Heat equation), a simple linear problem convenient to set up the protocol which would be use to reconstruct operator on non-linear partial differential equations. Let u be a time- and space-dependant solution to the Heat equation on the domain $\Omega \subset \mathbb{R}$, the equation is defined as:

$$u_t = \kappa u_{xx}, \tag{3.1}$$

subject to periodic Dirichlet boundary conditions with initial conditions:

$$u(0,x) = u_0(x),\ x \in \Omega = (0,1), \tag{3.2}$$

$$u(t,0) = u(t,1),\ t \in (0,T). \tag{3.3}$$

The equation 3.1 is known to have solutions satisfying its initial value conditions in the form

$$u(t,x) = e^{-k^2 t} e^{ikx}, \;\; k \in \mathbb{N} \tag{3.4}$$

$$u(t,x) = e^{-\pi^2 k^2 t} \sin(\pi k x) \tag{3.5}$$

In the latest case, the operator $\mathcal{K} = -\frac{\partial^2}{\partial x^2}$ has eigenfunctions

$$a_k(x) = \sqrt{\frac{2}{L}} \sin\left(\frac{\pi k x}{L}\right) \tag{3.6}$$

with the eigenvalues

$$\lambda_k = \frac{\pi^2 k^2}{L^2} \tag{3.7}$$

which form an orthogonal basis in $\mathcal{L}_2(\Omega)$. The solution u can be written in the form of

$$u(t,x) = \sum_k c_k e^{-\kappa \lambda_k t} a_k(x), \tag{3.8}$$

where $(c_k)_{k=1}^{\infty}$ represent the Fourier coefficients of the initial value. They are given by

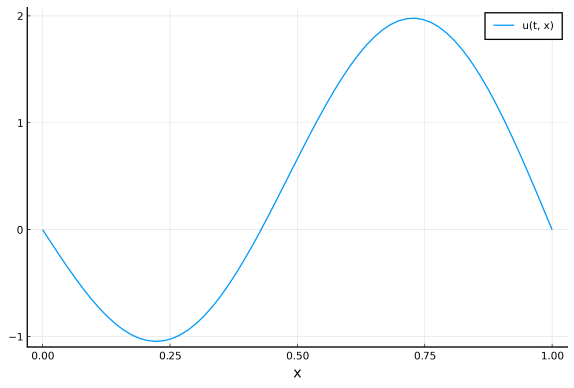
$$c_k = \int_\Omega u_0(x) a_k(x) \,\mathrm{d}x, \quad k \in \mathbb{N}. \tag{3.9}$$

As $k \in \mathbb{N}$ and $\lambda_k > 0$, the solution decays when t increases since $e^{-\kappa\lambda_k t} \to 0$. Therefore, for these conditions the zero solution $\|u(t,x)\| \to 0$ is stable. It should be noted that in this forward case, we define stability in the fact that the *perturbations* will decay to zero. This analysis stand as well for fixed boundary conditions of the $u(t,0) = u(t,1) = 0, \; t \in (0,T)$ at the condition that the values tends to 0 for $x = 0, 1$.

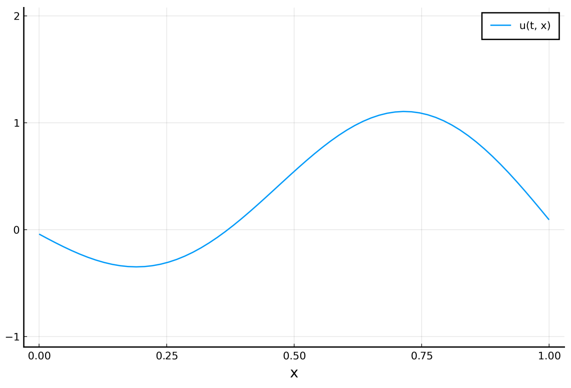
Let $\mathcal{S} : u_0 \mapsto u_T$ denote the forward solver operator. Now, it should be highlighted that the backward solving of the diffusion equation $\mathcal{S}u_0(x) = u(T,x)$ is an ill-posed inverse problem. As irregularities in the solution $u(t,x)$ decay when solving forward in time, they get amplified when solving backwards in time. Finding the initial conditions $u_0(x)$ from $u(T,x)$ becomes exponentially unstable, as observed in the Figure 3.1.

The ill-posedness of the elementary linear diffusion equation is a great instance of the difficulties encountered when training a neural ordinary differential equation during the backward propagation phase. As seen in the sensitivity analysis section 2.4.2, using a method such as back-solve adjoint which requires the resolution of an ODE backward in time, while working with complex architecture to represent the operator K will result in instabilities; triggering the necessity of the introduction of check-pointing, as well as method such as interpolating adjoint which get rid of the ODE backward resolution.

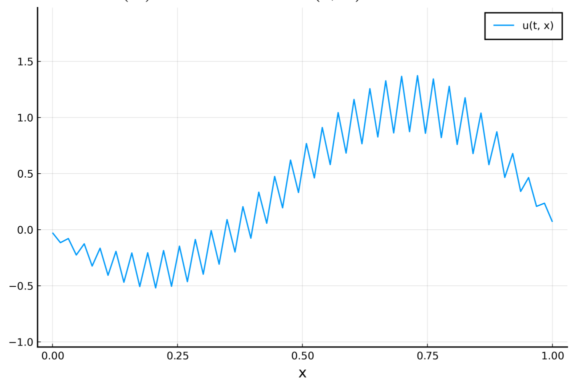
In the results we use Equation (3.8) to generate data-sets. We then learn K_θ, a discretizion of the diffusion operator $\mathcal{K} = \frac{\partial^2}{\partial x^2}$. Note that $\mathcal{K}$ can be discretized as a matrix K using a second-order



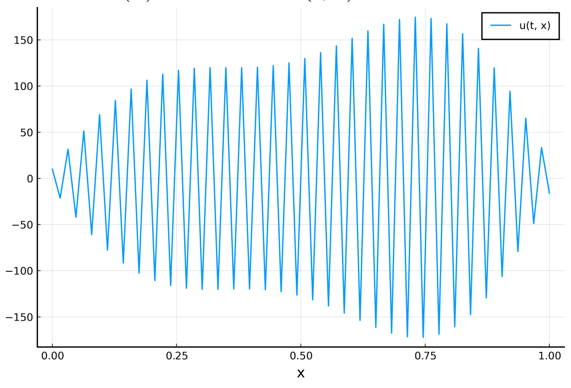
(A) Initial state $u(t, x)$ at $t = 0$



(B) Final state $u(t, x)$ at $t = 2$



(C) Backward solution $u(t, x)$ at $t = 1.94$



(D) Backward solution $u(t, x)$ at $t = 1.90$

FIGURE 3.1: Heat equation with diffusion constant $\kappa = 0.01$, $x \in \Omega = (0, 1)$. The top figures represent forward resolution of the equation using the Tsitouras 5 method, a 4^{th} order 5-stage Runge-Kutta method for $t \in [0, 2]$. The bottom figures represent the backward resolution of the equation from $t \in [2, 0]$. Ill-posedness can be observed as the problem of retrieving the initial state $u_0(x)$ from the final state $u(t = 2, x)$ is exponentially unstable.

accurate central finite difference method:

$$K = \frac{1}{\Delta x^2} \begin{pmatrix} 0 & \dots & \dots & \dots & 0 \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ 0 & \dots & \dots & \dots & 0 \end{pmatrix}. \tag{3.10}$$

This approach acts as an efficient warm-up to check if data-driven methods are able to learn an approximation to K.

3.3 Data-driven approximation method applied to linear equation

Before considering a non-linear case, we are interested into evaluating the efficiency of the operator reconstructed while not spending computational resources on training neural network on high-dimension cases. We set-up a set of high-dimensional numerical solutions to the linear Heat equation generated from different initial conditions. This data-set, see an example in Figure 3.2, is based on the wave-like analytical solution described in the previous section 3.2.

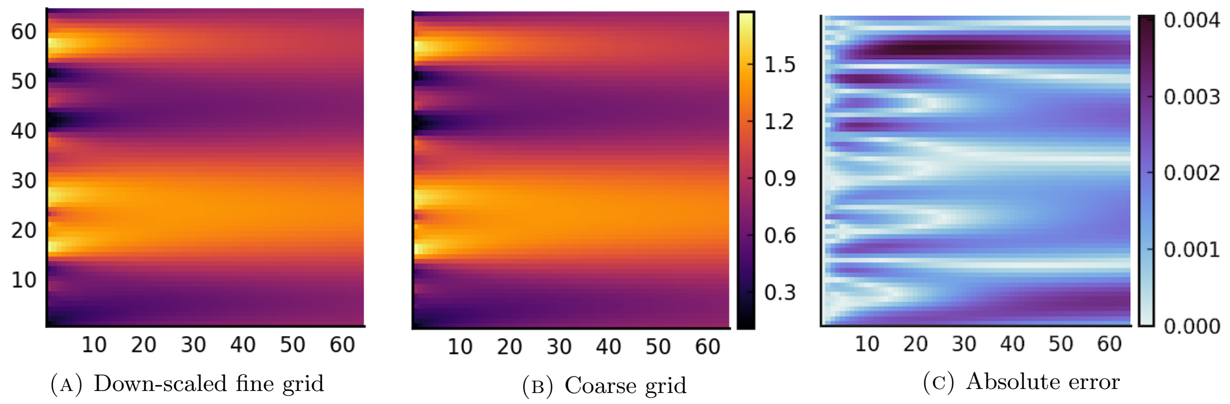


FIGURE 3.2: Heat equation with diffusion constant $\kappa = 0.01$, $x \in \Omega = (0,1)$, $t \in [0,1]$. The left figures represents a fine grid resolution of the equation using the Runge-Kutta 5^{th} order method downscaled to a 64-by-64 grid. The middle figure represents a coarse grid solution to the equation. The right figure shows the absolute error between the downscaled and coarse grid solutions.

The high dimensional solutions to the diffusion equation are used to create the down-scaled solutions. Down-scaling is performed by averaging the results on the x-axis from the fine-grid over a batch of cells. We highlight the fact that this example focuses on *Markovian* models where the time evolution by definition only depends on the current state. Therefore the reduction of the trajectory over time is done by a sampling at even intervals rather than averaging over a batch of trajectories; the latest choice would generate *non-Markovian* models. Such a choice would require the usage of a delay differential equation (DDE) rather than an ODE.

It can be observed that even for the elementary diffusion equation, a coarse grid cannot capture the finest details of the FOM. It doesn't reflect, like other simplest ROMs, the most detailed elements of the simulation, this is especially visible in the non-linear problem case.

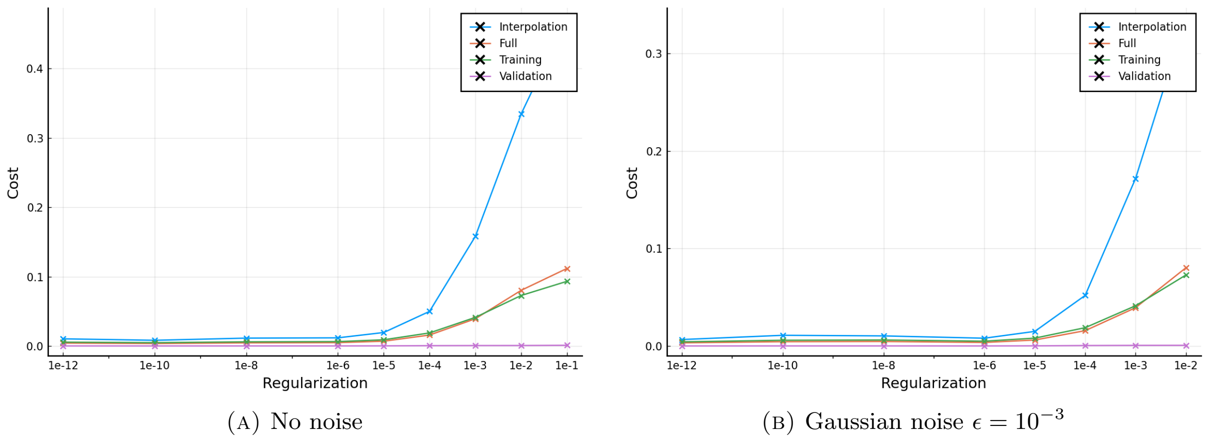
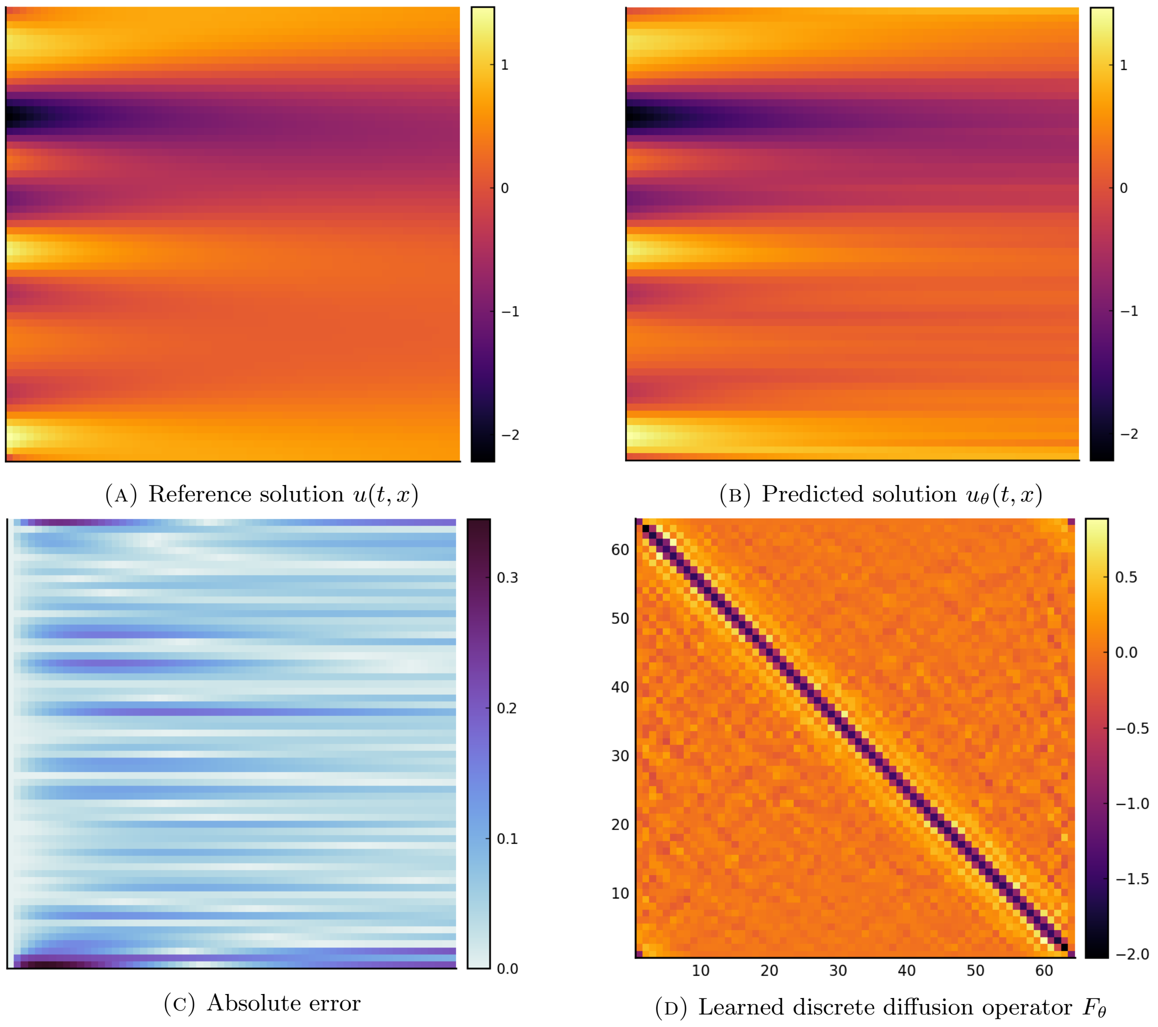


FIGURE 3.3: Objective function error after training a neural ODE using intrusive method with regards to the $\mathcal{L}_2$ regularization. The figures contain different levels of Gaussian noise added to the training data. The interpolation curve introduces evaluation on a new dataset. While full, training and validation curves are based on the dataset used for training.

In order to assess the effectiveness of NODEs to approximate a differential equation operator, we generate a data-set of 256 reference solutions $u_{ref}(t, x)$ with $t, x \in (0, 2) \times \Omega$ using 64 snapshots over time. The objective function uses the trajectory approach, as it was confirmed through experiment that the derivative approach results in divergence of the resolution. The $\mathcal{L}_2$ regularization weight is fixed at 10^{-8} according to the results in the Figure 3.3, a value which will be confirmed again in the non-linear PDE experiments. As observed in Figure 3.4d, as F_θ is close to the true operator F displayed in equation 3.10.



(A) Reference solution $u(t, x)$

(B) Predicted solution $u_\theta(t, x)$

(C) Absolute error

(D) Learned discrete diffusion operator F_θ

FIGURE 3.4: An instance of prediction returned by the approximated operator F_θ from the right-hand side of the diffusion equation obtained with training of a NODE. The example is an interpolation application as the initial condition used are not part of the original training and validation data-set. $x \in \Omega = (0, 1)$, $t \in [0, 1]$, $\kappa = 10^{-2}$.

3.4 Discussion

In this chapter, we performed a first evaluation of the usability of ML models to infer the dynamics of differential equations.

First, the linear diffusion equation gave us a concrete example of the difficulties brought in by the

optimise-then-discretize approach when training NODE, given that method such as back-solve adjoint which consists in solving a differential equation backward in time, to obtain the loss derivative with regards to the model parameters $\frac{\partial \mathcal{L}}{\partial \theta}$, is inevitably an ill-posed inverse problem.
Next, given that the linear diffusion is the simplest model possible, it allowed us to analyse together its capacity to predict an accurate solution to the PDE but also how the model would be parameterized. Using a simple layer, we observed that NODE would reconstitute an operator F_θ close to the reference operator F. We hypothesize than NODE used to learn full right-hand side of an equation, can and might aim at recovering the sub-lying linear and non-linear term of the learned equations, rather than converging to a radically different model. This suggests that the architecture of the NODE should be selected such that it can specifically handled the term of the equation, for instance, using convolutional layers to approximate a derivative which would usually be discretized using finite difference method.

Chapter 4

Evaluation of data-driven methods applied to non-linearity

4.1 Introduction

The main task of this section is to apply the data-driven operator approximation methods reviewed previously in the chapters 2 and 3 to approximate the model operators for a non-linear partial differential equation. While the objectives of this part consist in the evaluation of the accuracy of these approximated operators to interpolate solution from unknown initial conditions; this chapter also answers interrogations on the choice of neural network architecture to handle non-linear terms in the PDEs, and recover discarded information from a low fidelity model. As it will be seen through experiments, while neural networks are known to be universal approximators of continuous functions [HSW89], it remains a challenge to obtain a suitable approximation using acceptable computational resources.

This chapter helps deepen our understanding of differentiable ODE solvers in the context of neural ODE training. To do so, the subsequent sections introduce the non-linear Bateman-Burgers equation and its different variants (viscous and inviscid), and highlight some of the emerging phenomena such as discontinuities (shock waves) in the inviscid Burgers equations. They cover the known solution and the numerical methods implemented to compute the solution to the high fidelity model which will serve as a base for the training data-set used by the data-driven operator approximation methods. Given that, the chapter also aims at evaluating the usability on closure model for accurate predictions by using a neural ODE as an approximation to the right-hand side of the equation or as a closure term associated to the well-known low fidelity model (POD-GP) of the same equation, a case where low fidelity might generate more difficulties. It will briefly discuss the analytical closure term conventionally used and its interest in the context of this thesis.

4.2 Bateman-Burgers equation

As an introduction to a non-linear case, the subsequent step consists in shifting the methodology developed from the linear diffusion equation to the Bateman-Burgers equation. The Burgers equation represents the velocity $u(t, x)$ at each instant t and location x of a viscous fluid (with viscosity ν) flowing through an ideal pipe. This equation is of interests for the stability problems resulting from its different form, given that its non-linearity has been used for modeling wide range of phenomena such as shock waves, wave propagation, etc. where there exists balancing viscous and convective forces.

Simplification of Navier-Stokes Consider the incompressible Navier-Stokes equations

$$\rho\left(\frac{\partial u}{\partial t} + u \cdot \nabla u\right) = -\nabla p + \nu \Delta u + G \tag{4.1}$$

with density ρ, fluid viscosity constant ν, pressure p, external force G and u the flow velocity. If we assume that there is no external force G and the pressure term p is negligible, then Equation (4.1) leads to the *non-conservative form* of second-order *viscid Burgers equation*:

$$\frac{\partial u}{\partial t} = -u \cdot \nabla u + \nu \Delta u. \tag{4.2}$$

Here, $u(t, x)$ is a time- and space-dependant flow velocity solution on the periodic domain $\Omega \subset \mathbb{R}$, $\nu = \frac{\mu}{\rho}$ is the kinematic viscosity (aka. diffusion coefficient), $-\frac{1}{2}\frac{\partial u^2}{\partial x}$ is the the advection term, and $\nu \frac{\partial u^2}{\partial x^2}$ is the diffusion term.

From equation (4.2), we can derive the Burgers equation in its *conservative form*, a form usually preferred for numerical integration of non-linear problems for which the solution might develop discontinuities (aka. shock waves):

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2} - \frac{1}{2}\frac{\partial u^2}{\partial x}. \tag{4.3}$$

Eventually, the non-conservative form leads us to consider the first-order *inviscid Burgers equation* ($\nu \to 0$), from which such shock waves can emerge:

$$\frac{\partial u}{\partial t} = -u \frac{\partial u}{\partial x}. \tag{4.4}$$

For simplicity, we always set the forcing term $G = 0$, likewise the spatial periodic boundary conditions $u(t, 0) = u(t, L)$.

4.2.1 Fourier spectral method for viscous Burgers equation

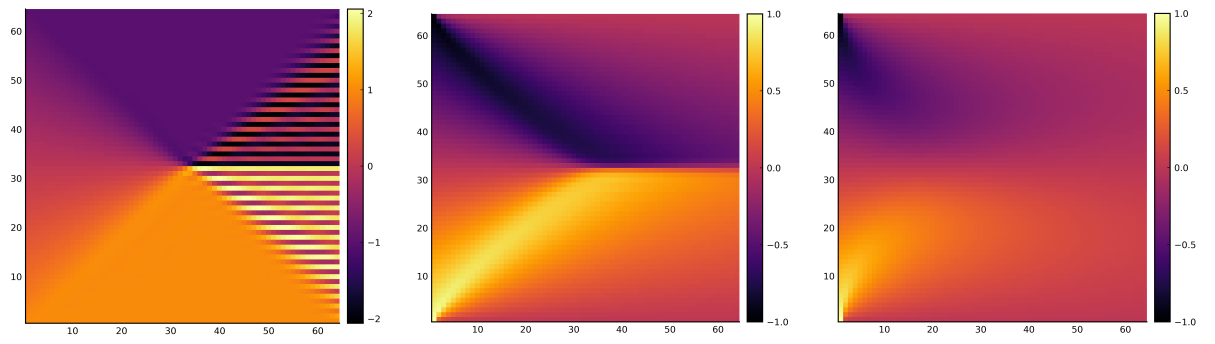
In order to compare low and high dimensional FOM of the system to ROM, the pseudo-spectral method [BDH+86] (a function-space method) is used to solve the problem (4.2). This method gives a smooth solution to the IVP and boundary value problem (BVP) on a periodic domain. We approximate the solution $u(t, x)$ of the equation (4.2) as a truncated sum of basis functions (usually sin and cos waves) using the fast Fourier transformation (FFT):

$$u(t, x) \approx \hat{u}_n(t, x) = \sum_n c_n(t) e^{i k_n x} \text{ with } k_n = \frac{2\pi f_n}{L}, \tag{4.5}$$

where c_n are the discrete Fourier coefficients and $(f_n)_n$ represent the discrete Fourier transform (DFT) sample frequencies. The DFT and IDFT enable us to easily compute the derivatives of the approximated solution $u_n(t,x)$ (see Equation (4.5)).

$$\hat{u}_x = ik\hat{u}_n \tag{4.6}$$
$$\hat{u}_{xx} = -k^2\hat{u}_n \tag{4.7}$$

Substituting 4.5 into the PDE lets us generate a system of ODEs which can be easily solved using the Tsit5 method (see the viscous solution in Figure 4.1). The approximated solutions computed by the Fourier spectral method are used to generate the viscous Burgers data-set used to train the NODEs.



(A) Conservative inviscid Burgers equation ($\nu \to 0$) (B) Non-conservative viscous burger equation ($\nu = 0.01$) (C) Non-conservative viscous burgers equation ($\nu = 0.1$)

FIGURE 4.1: Evolution of the solution to the Burgers equation in non-conservative viscous and conservative inviscid cases for $x \in [0,1]$, $t \in [0,2]$ with initial conditions $u(0,x) = -x$. The formation of a shock can be observed in the inviscid case (left) at $t = 1$.

4.2.2 Godunov conservative schemes for inviscid Burgers equation

Not all methodologies can be applied to solve the different forms (conservative or non-conservative, inviscid or viscous) of the Burgers equation, and it should be notice than the above implementation of the pseudo-spectral method does not handle the *inviscid* ($\nu \to 0$) form of the equation. To numerically solve it, we change our approach and consider the *Godunov*'s conservative scheme.

Godunov's scheme Suggested by Godunov [GB59], this method is based on finite volume methods [L+02] and typically used to solve hyperbolic PDE with discontinuities, such as inviscid Burgers equation. In it's simplest form it is a first-order accurate method in both time and space.

Let's consider the hyperbolic problem

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0 \tag{4.8}$$

Let's also define U_i^n, a numerical solution at time t_n, which approximate $u(t_n, x)$ on a grid cell i.

$$U_i^n \approx \frac{1}{\Delta x} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} u(t_n, x) dx \tag{4.9}$$

With $t_n = n\Delta t$, $x_{i+k} = x_{\min} + (i+k)\Delta x$. To get ride of the approximation, we define $\tilde{u}(t_n, x)$ a step function which will depends on the grid discretization.

$$U_i^{n+1} = \frac{1}{\Delta x} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \tilde{u}(t_{n+1}, x) dx \tag{4.10}$$

The above formulation means we are using U_i^n to define $u(t_n, x)$ on a discretized grid space $x_{i-\frac{1}{2}} < x < x_{i+\frac{1}{2}}$.

We need to integrate the problem over $[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$ and obtain the cell average, however it is not necessary to perform the explicit integration in (4.10) as $\tilde{u}(t_{n+1}, x)$ would be too complex. Instead, the step function $\tilde{u}(t, x_{i+k})$ is constant in time, so we can use it. Let's consider the *integral form* of *conservation law* [L+02]:

$$\frac{\partial}{\partial t} \int_{x_a}^{x_b} u(t, x) dx = F(x_a) - F(x_b) \tag{4.11}$$

It can be rewritten as

$$\frac{\partial}{\partial t} \int_{x_a}^{x_b} u(t, x) dx = f(u(t, x_a)) - f(u(t, x_b)) \tag{4.12}$$

Then, from the previous definition of U_i^{n+1} in equation (4.10), we can define the derivative

$$\frac{\partial U_i}{\partial t} = \frac{1}{\Delta x} \Big(f(u(t, x_{i-\frac{1}{2}})) - f(u(t, x_{i+\frac{1}{2}})) \Big) \tag{4.13}$$

And infer the equation

$$U_i^{t+1} = U_i^t - \frac{1}{\Delta x} \Big(f(u(t, x_{i+\frac{1}{2}})) - f(u(t, x_{i-\frac{1}{2}})) \Big) \tag{4.14}$$

We can then define the numerical flux functions $F(X, Y)$ based on the integral over time, easier to handle, such that:

$$F(U_{i-1}^n, U_i^n) = \frac{1}{\Delta t} \int_{t_n}^{t_{n+1}} f(u(t, x_{i-\frac{1}{2}})) dt \tag{4.15}$$

$$F(U_i^n, U_{i+1}^n) = \frac{1}{\Delta t} \int_{t_n}^{t_{n+1}} f(u(t, x_{i+\frac{1}{2}})) dt \tag{4.16}$$

Replacing the integral in the previous equation (4.14) leads to the Godunov's formulation:

$$U_i^{n+1} = U_i^n - \frac{\Delta t}{\Delta x} \left(F(U_i^n, U_{i+1}^n) - F(U_{i-1}^n, U_i^n) \right) \tag{4.17}$$

While we now have the Godunov's formulation used to solve the hyperbolic problem (4.8), we still needs to compute the numerical flux (4.16) composing it. The next step then requires to compute at least an approximated solution to a *Riemann* problem.

Riemann problem The Riemann problem is used in finite-volume methods and consists of hyperbolic equations with specific initial data. Usually it has the form

$$f(x,0) = \begin{cases} f_a & x \leq 0, \\ f_b & x > 0. \end{cases} \tag{4.18}$$

Solving the problem between two states U^n_{i-1} and U^n_i let us determine the numerical flux $F^n_{i-\frac{1}{2}}$ required in Godunov's method. We consider here a method to approximate Riemann solution has it is computationaly expensive to solve it exactly.

4.3 An application to closure modeling

The previous chapter gave us insights on usage of NODEs and stability of the network training while the previous section introduces the Burgers equation and the schemes used to compute a solution using a FOM (i.e. the Fourier pseudo-spectral method or Godunov method depending on the PDE parametrization). The FOM model is visualized in Figure 4.2.

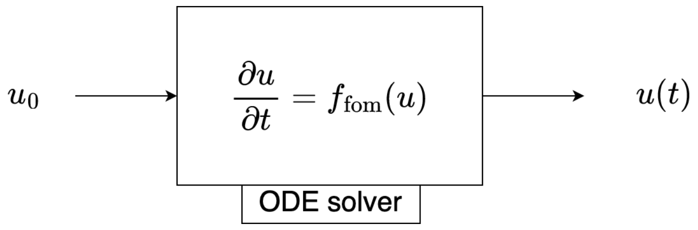


FIGURE 4.2: Architecture of a full-order model (model 1). From $u(t)$, we use an ODE solver to compute $u(t+\Delta t)$. This model is used for every reference data either for down-scaled problem or true POD coefficient.

We now evaluate the efficiency of the methods used to approximate the operators when applied to closure modeling. As part of this thesis work, we define two different approaches with the *down-scaling* and *POD-Galerkin projection* types of problems.

4.3.1 Down-scaling problem

The down-scaling problem was already introduced with the diffusion equation. From a high-dimensional fine grid solution $u(t)$, a reference down-scaled solution $v(t) = Du(t)$ is used as a point of comparison to evaluate the data-driven method approach. From a coarse grid field v, we can also reconstruct a fine grid field using an upscaling operator U.

$$v(t) = Du(t), \quad D \text{ down-scaling}, \quad v(t) \in \mathcal{P} \tag{4.19}$$

$$u_{\text{rom}}(t) = Uv(t), \quad U \text{ up-scaling}, \quad u_{\text{rom}}(t) \in \mathbb{R}^{\mathbb{N}_x} \tag{4.20}$$

The models encompassing the equation's right-hand side are summarized in Table 4.1. They consist of a pure neural ODE (i.e. model 3), or a closure model composed of the baseline low-dimensional coarse grid solution (i.e. the baseline model 2), and a closure term (i.e. model 4). It is interesting to consider this problem to retrieve the dynamics of a non-linear equation on a lower scale, which might include discontinuities (i.e. inviscid equation) or high-frequencies (i.e. Korteweg–De Vries equation).

N	**Model**	**Formulation**	**Definition**
1	Full-order model	$\frac{\mathrm{d}v}{\mathrm{d}t} = Df_{1024}(u)$	Downscaled fine grid discretization
2	Baseline model	$\frac{\mathrm{d}v}{\mathrm{d}t} = f_{64}(v)$	Coarse grid discretization
3	Pure NODE	$\frac{\mathrm{d}v}{\mathrm{d}t} = \mathrm{NN}(v;\theta)$	Neural ODE encompassing model
4	Neural closure model	$\frac{\mathrm{d}v}{\mathrm{d}t} = g(v) + \mathrm{NN}(v;\theta)$	Coarse grid + Neural ODE for unresolved terms

Table 4.1: Different ODE models for the coarse-grid field $v(t) = Du(t)$. Contrary to POD problem, $v(t)$ is not a projection of $u(t)$ onto global modes, but represents a coarser discretization of the same continuous field.

However, this approach cannot be considered as an appropriate ROM given that the computational complexity isn't thoroughly reduced. An appropriate reference would be $u(t)$ instead of the downscaled $v(t)$, a problem which can be related to the super-resolution topic. This can be done by considering a ROM able to reduce the ODE system to solve.

4.3.2 Proper orthogonal decomposition problem

Next, we introduce the well-known POD problem type [OT13] [APS+21], from which we investigate ROM able to drastically reduce the dimension of the ODE system to solve. The full order model solution u is projected onto modes stored as columns in the matrix Φ. The coefficents are denoted by $a = \Phi^\intercal u$, and the ROM prediction is reconstructed from the coefficients: $u_{\mathrm{rom}}(t) = \Phi a(t)$. The different models for the evolution of the coefficients a are presented in Table 4.2.

N	**Model**	**Formulation**	**Definition**
1	Full-order model	$\frac{\mathrm{d}a}{\mathrm{d}t} = \Phi^\intercal f_{\mathrm{fom}}(u(t))$	Projected reference model
2	Baseline model	$\frac{\mathrm{d}a}{\mathrm{d}t} = g(a) = \Phi^\intercal f_{\mathrm{fom}}(\bar{u} + \Phi a)$	POD + Galerkin projection
3	Pure NODE	$\frac{\mathrm{d}a}{\mathrm{d}t} = \mathrm{NN}(a;\theta)$	Neural ODE encompassing model.
4	Neural closure model	$\frac{\mathrm{d}a}{\mathrm{d}t} = g(a) + \mathrm{NN}(a;\theta)$	ROM + Neural ODE for unresolved terms

Table 4.2: Different ODE models for the POD coefficients $a(t) = \Phi^\intercal u(t)$.

Proper orthogonal decomposition

The original idea behind proper orthogonal decomposition (POD), which can be related to the principal component analysis (PCA), focus on determining a space $\mathcal{X}^t = \mathrm{span}\{\theta^1 ... \theta^{N_t}\}$, known as

POD modes, which approximate the snapshot of a random vector field $u(t)$ solution to the model studied, catching the main features of it with regards to the norm $\mathcal{L}^2$ [Wei19].

The first step consists of generating a snapshot matrix in the discrete domain, composed of a set of snapshots of the normalised solution $\hat{u}(t,x)$ over time ($\bar{u}$ are the time-averaged spatial values).

$$S = [\tilde{u}^1|\tilde{u}^2|...|\tilde{u}^{N_t}] \in \mathbb{R}^{\mathbb{N}_x \times \mathbb{N}_t} \tag{4.21}$$

$$\tilde{u}^i = u^i - \bar{u} \tag{4.22}$$

$$\bar{u} = \frac{1}{N_t}\sum_{i=1}^{\mathbb{N}_t} u^i \tag{4.23}$$

The basis are obtained in the next step using the singular value decomposition (SVD)

$$U\Sigma V^\intercal = S \tag{4.24}$$

where $U \in \mathbb{R}^{\mathbb{N}_x \times \mathbb{N}_x}$ and $V \in \mathbb{R}^{\mathbb{N}_t \times \mathbb{N}_t}$ are an orthogonal matrices, and $\Sigma \in \mathbb{R}^{\mathbb{N}_x \times \mathbb{N}_t}$ a rectangular diagonal matrix containing the singular values. Eventually we are able to obtain the POD ROM $\hat{U} = \Phi_r\ \Phi_r^\intercal U$. The POD approach is presented in Algorithm 3.

Algorithm 3: SVD based Proper orthogonal decomposition

```
Require: M, k
1: $\bar{M} \leftarrow \mathrm{mean}(M)$                ▷ Temporal mean
2: $S \leftarrow M - \bar{M}$                        ▷ Normalize
3: $U\Sigma V^\intercal \leftarrow SVD(S)$
4: $\Phi_r \leftarrow U[:, 1:k]$                     ▷ Reduce basis
5: $\hat{M} \leftarrow \Phi_r\ \Phi_r^\intercal M + \bar{M}$   ▷ Reduce model
6: return $\Phi_r$, $\hat{M}$
```

We note that an eigenvalue decomposition approach was also considered (see Appendix B), both approaches are intimately related. Some of the papers which served as a base of the work of this chapter, used the different approaches to determine reduced basis. While the SVD is suitable for non-square matrices and remain in a domain of real values as long as the matrix decomposed $U \in \mathbb{R}^{n \times m}$; the eigenvalue decomposition is only usable with square matrices, moreover, unsymmetrical real matrices might lead to complex eigenvalues and eigenvectors. Because of the latest point, the complexity to solve eigenproblem make it unsuitable for large asymmetric matrices. Given that data-set used are composed of asymmetric large matrices in the real domain, we settled for the SVD approach.

The orthogonality property of the POD basis Φ at hand allow us to perform a Galerkin projection onto a reduced space $\mathcal{P}$. Usually, the energy contribution criterion is used to choose the optimal number of POD bases retained in $\mathcal{P}$ such that it maintains a pre-defined energy percentage ϵ:

$$\frac{\sum_{i=1}^{\mathbb{N}_r} \lambda_i}{\sum_{i=1}^{\mathbb{N}_x} \lambda_i} \geqslant \epsilon, \tag{4.25}$$

where $\mathbb{N}_r$ is the number of retained basis modes among the total ($\mathbb{N}_x$) amount of modes, $(\lambda_i)_{i=1}^{\mathbb{N}_x}$ are the eigenvalues of $S^\intercal S$, and $\epsilon \approx 0.999$.

Galerkin Projection

The POD basis provides us with the underlying dynamics of the model. Let's consider a non-linear equation under a generic ODE formulation. The full order model operator f can be decomposed into a linear part L and non-linear part N:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = f(u) = L(u) + N(u). \tag{4.26}$$

We can approximate the FOM solution $u(t)$ in a low-dimensional subspace as a composition of the reduced basis using the Galerkin method:

$$a(t) = \Phi^{\mathsf{T}}(u(t) - \bar{u}), \tag{4.27}$$

resulting in the approximation

$$u(t) \approx \bar{u} + \Phi a(t), \tag{4.28}$$

where $\Phi \in \mathbb{R}^{N_x \times N_r}$ is the reduced basis, $a(t) \in \mathbb{R}^{N_r}$ are the time dependent coefficients, and $\bar{u} \in \mathbb{R}^{N_x}$ is a reference field (e.g. a time average).

Now, we use the orthogonal property of the POD basis $\Phi^{\mathsf{T}}\Phi = I$, so that we can express the equation (4.26) in a reduced basis. The idea consists in using the approximation (4.27) as a replacement in the equation (4.26). Considering $\bar{u}$ and Φ are independent from t, we simplify the equation

$$\begin{aligned}
\frac{\mathrm{d}a}{\mathrm{d}t} &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\Phi^{\mathsf{T}}(u(t) - \bar{u})\right) \\
&= \Phi^{T}\frac{\mathrm{d}u}{\mathrm{d}t} \\
&= \Phi^{T} f(u) \\
&\approx \Phi^{T} f(\bar{u} + \Phi a) \\
&= \Phi^{\mathsf{T}} L(\bar{u}) + \Phi^{\mathsf{T}} L(\Phi a) + \Phi^{\mathsf{T}} N(\bar{u} + \Phi a).
\end{aligned} \tag{4.29}$$

By grouping together the constant ($\mathcal{B}$), linear ($\mathcal{L}$), and quadratic ($\mathcal{N}$) parts (with respect to a), we obtain the equations

$$\frac{\mathrm{d}a}{\mathrm{d}t} = \mathcal{B} + \mathcal{L}a + a^{\mathsf{T}}\mathcal{N}a. \tag{4.30}$$

Using the three operators $\mathcal{B}$, $\mathcal{L}$, and $\mathcal{N}$ adapted to the equation studied, the equation can be solved by using Runge–Kutta or equivalent methods.

We remember the decomposition (4.27) where $a(t) = \Phi^{\mathsf{T}}(u(t) - \bar{u})$ and the system (4.29). This is the baseline model $\frac{\mathrm{d}a}{\mathrm{d}t} = \Phi^{\mathsf{T}} f(\bar{u} + \Phi a)$ (i.e. model 2 in Table 4.2) used as a point of comparison. We highlight that the above model can be implemented directly in the experiment, or slightly improved by computing the operators offline. Indeed, consider the Burgers equation

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2} - \frac{1}{2}\frac{\partial u^2}{\partial x}, \tag{4.31}$$

It can be discretized using the finite difference operators D_x and D_{xx} representing the continuous operators $\frac{\partial}{\partial x}$ and $\frac{\partial^2}{\partial x^2}$ respectively:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = f(u) = \underbrace{\nu D_{xx} u}_{L(u)} - \underbrace{\frac{1}{2} D_x u^2}_{N(u)}. \tag{4.32}$$

Using the approximation (4.28) for $u(t)$, we retrieve a system of ODEs for the modal coefficients a(t) as in (4.30):

$$\frac{\mathrm{d}a_k}{\mathrm{d}t} = \mathcal{B}_k + \sum_i^{N_r} \mathcal{L}_k^i a_i + \sum_i^{N_r} \sum_j^{N_r} \mathcal{N}_k^{ij} a_i a_j. \tag{4.33}$$

The constant, linear, and quadratic parts may be precomputed as follows:

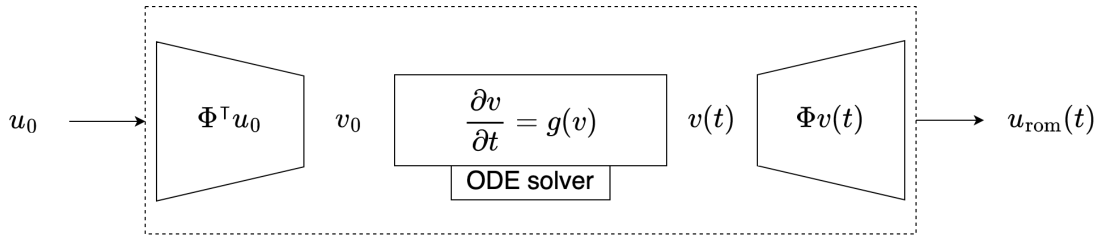
$$\mathcal{B}_k = \varphi_k^\intercal \left(\nu D_{xx} \bar{u} - \frac{1}{2} D_x \bar{u}^2 \right), \tag{4.34}$$

$$\mathcal{L}_k^i = \varphi_k^\intercal \left(\nu D_{xx} \varphi_i - D_x(\bar{u}\varphi_i) \right), \tag{4.35}$$

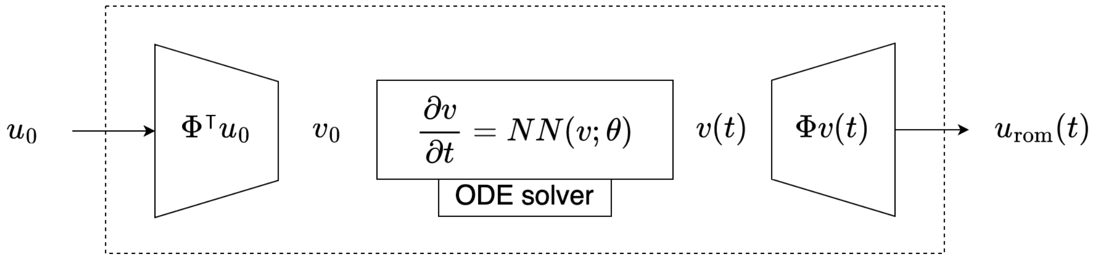
$$\mathcal{N}_k^{ij} = \frac{1}{2} \varphi_k^\intercal D_x(\varphi_i \varphi_j), \tag{4.36}$$

where φ_k denotes the k-th column of Φ and $1 \leq k \leq N_r$. The results are the same but the computation time is substantially improved.

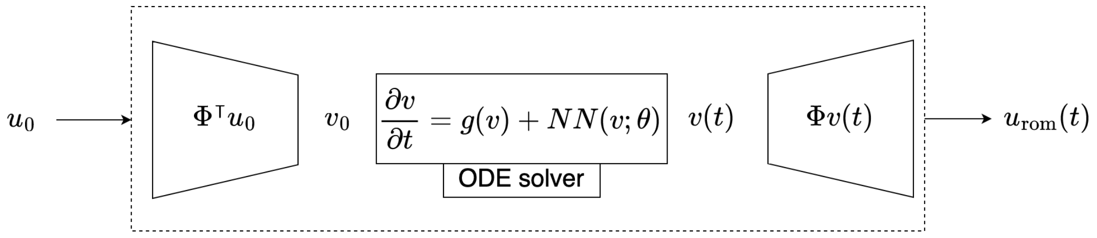
Again, a neural ODE encompassing model (i.e. model 3 in the Table 4.2), and a closure model composed of the baseline Galerkin projection model and a closure term (i.e. model 4) are evaluated. Figure 4.3 helps to visualize how the POD approach is of interest for its capacity to reduced dimensions of the system, and how NODEs could be helpful. This latest part is motivated by the work of Maulik [MML+20] and [GL21] who both used this approach for NODE and memory-based model such as LSTM and NDDEs. However, the experiments were restricted to a single condition and focus on the prediction over time. In our experiments, we rather try to apply this approach to large set of snapshots $u(t)$.



(A) Baseline model (model 2)



(B) Pure neural network model (model 3)



(C) Neural closure model (model 4)

FIGURE 4.3: Architecture of evaluated reduced order models used in the POD-projection problem. Notice that these schema do not subtract the temporal mean $\bar{u}$, this case corresponds to a context where the temporal mean would be $\bar{u} = 0$, a possibility if the POD basis are computed on a large trajectories data-set. Or if the POD-basis are build wihtout subtracted mean, therefore encompassing its influence. To corresponds to the Galerkin projection methodology listed above, the schema should replace $\Phi^\intercal u_0$ by $\Phi^\intercal(u_0 - \bar{u})$, in a same way $\Phi v_(t)$ would become $\Phi v_(t) + \bar{u}$

Chapter 5

Numerical experiments and results

5.1 Introduction

We performed a range of numerical experiments on simulated data with the main objective of studying the efficiency of data-driven reconstruction methods when applied to non-linear differential equations. Some of the methodologies and tools were already introduced in the Chapter 3. They are extended in this chapter to answer most of the research questions evoked in our introduction.

To do so, multiple questions arise including the choice of an objective function, the selection of an optimisation algorithm, the sensitivity to initialisation and to conditioning, and fundamentally on the choice of architecture to learn non-linear term of the PDE. This first section provides to us the tools to tackle the problem of how to parameterize the operator F_θ depending of the problem considered. As it will be discussed, the answer is not universal and the architecture of the problem depends on the ROM selected (pure encompassing neural network or closure model), as well as the problem type it tries to tackle such as down-scaling (to a coarser grid on the same space) or POD projection (onto a different space of global coefficients).

5.2 Packages

During the past few years, a wide range of packages and libraries were developed in the field of scientific machine learning. This thesis take advantage of the open source modules available in the `JULIA` language `SciML` ecosystem for the following numerical experiments, allowing us to not re-implement most of the algorithms evoked.

DifferentialEquations.jl An ecosystem for solving differential equations. Offers the implementation of (non-)stiff ODE solving algorithm such as Runge-Kutta methods [RN17].

SciMLSensitivity.jl Sensitivity Analysis and Adjoints. Adjoint method for back-propagation in neural differential equations [MDI+21].

Flux.jl Abstraction to define Machine Learning model [ISF+18]

DiffEqFlux.jl Adaptation of `Flux.jl` package to neural differential equations and universal differential euations [RIM+19].

Zygote.jl Next-generation source-to-source automatic differentiation (AD) system [Inn18].

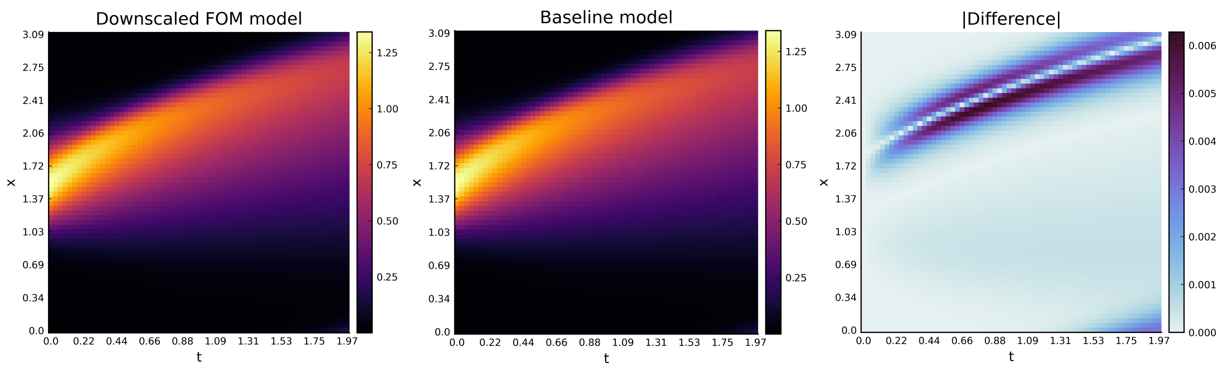
Optimization.jl A wrapping package for a large scope of local and global optimization algorithms (i.e. ADAM).

5.3 Training procedure for data-driven approximation method

5.3.1 Data and metrics

We select two sets of initial conditions all with periodic boundary conditions.
The first set uses a Gaussian stochastic process to generate similar waves in order to evaluate data-driven methods to prediction on a single case while excluding the complex interaction of multiple waves.

$$u_0(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{5.1}$$



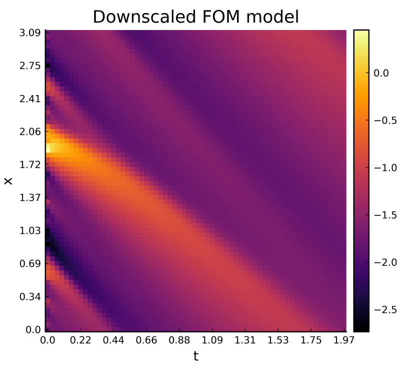
(A) Down-scaled fine grid (64×64) (B) Coarse grid (64×64) (C) Absolute Difference (64×64)

FIGURE 5.1: Evolution of a solution of inviscid Burgers equation using Gaussian-based initial conditions. With $x \in [0, \pi]$, $t \in [0, 6]$, $\nu \to 0$.

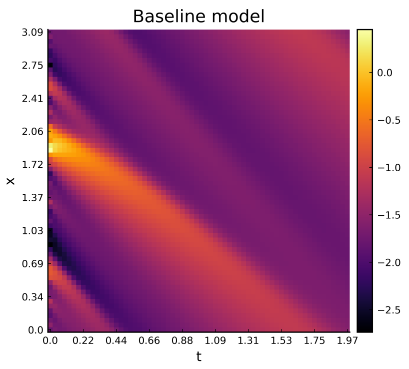
The second set uses a Fourier series to generate initial conditions and random variation in the phase and amplitude decay. These conditions create snapshots rich in information, an advantage to train the neural network. It also enables simulation of random shock discontinuities in the inviscid Burgers equation, one of the phenomena we aim at recovering in our operator reconstruction [HDK14]. The complexity of the interacting shock wave will enable us to experiment on the capacities of the approximated operator F_θ obtained to interpolate for each form of the equations.

$$u_0(x;\omega) = \sum_{k=1}^{K} \left(a_k(\omega) \sin\left(\frac{2\pi k x}{L}\right) + b_k(\omega) \cos\left(\frac{2\pi k x}{L}\right) \right) \tag{5.2}$$

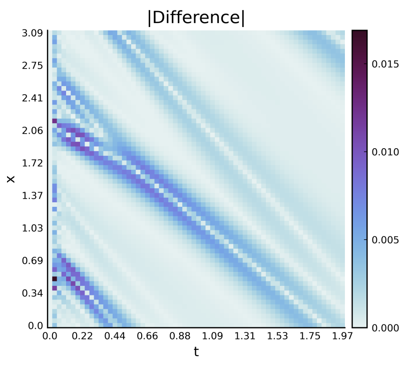
where $a_k(\omega) = -sin(\phi * \omega)$, $b_k = cos(\phi * \omega)$ follow a one-point probability density function (PDF) follow a Gaussian distribution with $\omega = \{\mu = 0,\ \sigma^2 = 1\}$. A large K generates rugged initial conditions resulting in shock discontinuities early over time while a low K will give smoother conditions and discontinuities later over time. The final time t is eventually extended for extrapolation (prediction) tasks.



(A) Down-scaled fine grid (64×64)

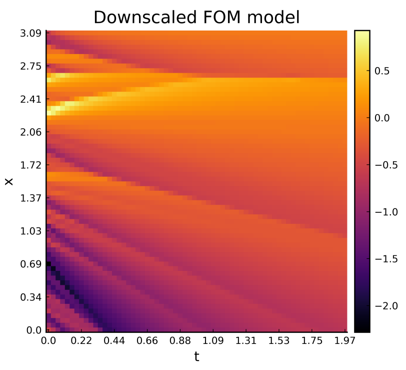


(B) Coarse grid (64×64)

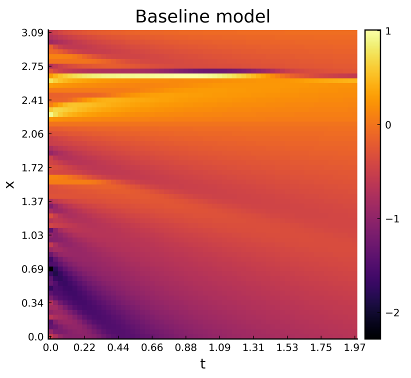


(C) Absolute difference (64×64)

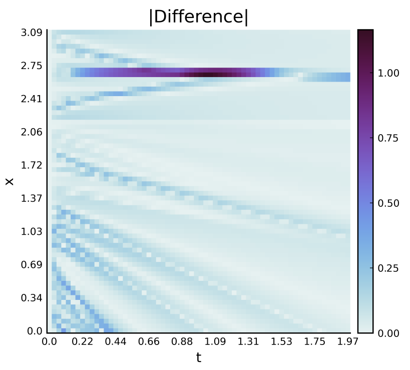
FIGURE 5.2: Evolution of one solution to the viscous Burgers equation using Fourier initial conditions. With $x \in [0, \pi]$, $t \in [0, 2]$, $m = 40$, $\nu = 0.04$.



(A) Down-scaled grid (64×64)



(B) Coarse grid (64×64)



(C) Absolute difference (64×64)

FIGURE 5.3: Evolution of one solution to the inviscid Burgers equation using Fourier initial conditions. Discontinuities (shock waves) are observable. However, low order models smooth the underlying dynamics with heavy loss of information. With $x \in [0, \pi]$, $t \in [0, 2]$, $m = 40$, $\nu \to 0$.

In all cases, we start with a fixed diffusion parameter ν to evaluate accuracy of our model to reconstruction and prediction tasks in a simplified context. Eventually, we consider the extension of Burgers equation parameters to a range of values as it is of interest to determine if data-driven method are sufficient to infer parameters of the dynamical system studied.

Objective function The main objective of the neural network consists in building an accurate approximated operator F_θ for interpolation (reconstruction inside the training interval) and extrapolation (prediction outside of training interval). In this context of supervised training, the neural network is expected to match the known training data and predict accurate results with the unknown validation and test data-sets. We recall the two approaches for the initial choice of objective function (based on either the $\mathcal{L}^2$ or MSE error functions) which must be considered the a-priori approach (derivative fitting) or the a-posteriori approach (trajectory regression).

As we aim at analysing this accuracy, there are two approaches for the initial choice of objective function (based on either the $\mathcal{L}^2$ or MSE error functions) which must be considered: the intrusive approach (trajectory regression) as well as the non-intrusive approach (model regression).

A-priori approach $\mathcal{L}(\theta) = \left\| F_\theta(t,u) - \frac{\mathrm{d}u(t)}{\mathrm{d}t} \right\|_2^2$

A-posteriori approach $\mathcal{L}(\theta) = \|u_\theta(t) - u(t)\|_2^2$

Such that $u_\theta(t)$ and $u(t)$ are respectively the predicted and true solutions, F_θ the approximated model, therefore we look for F_{θ^*}, where $\theta^* = \arg\min C(\theta)$ is the optimal model resolving the IVP. The a-priori approach evaluate the operator itself through a comparison with the derivative $\frac{\mathrm{d}u}{\mathrm{d}t}$, while the a-posteriori approach use the solution obtained when solving the ODE which corresponds to the PDE-constrained optimization problem we described in our introduction. We eventually settled for the a-posteriori approach. Early experiments on the hyper-parameters tuning resulted in the absence of convergence using the a-priori approach. Indeed, if an operator $K = F_\theta$ approximates the derivative correctly, it does not imply it will obtains a correct prediction.

Optimization algorithm In order to minimize the solution to the PDE-constrained optimization problem, it was first decided to make usage of the well-known adaptive moment estimation (ADAM) optimization algorithm [KB14]. Indeed, the measurements of the true solution to the equation, which inevitably involve some noises, despite the choice of an appropriate numerical resolution scheme, make a stochastic optimization method such as ADAM an acceptable choice. Considering our noisy cost function C_θ, the method determine learning rates for different parameters from estimation of the first and second moments of its gradients, while adapting the step size based on the computed gradient, until convergence. It is a combination of the properties of the previous ADAGrad[DHS11] and RMSProp algorithms.

Regularization As mentioned before, over-fitting of data is one of the common issues likely to happen in neural network training. In the current context, while including stochasticity the initial conditions still use a similar pattern. It is, therefore, a requirement to consider regularization. In a first time, with the choice of a feed-forward neural network architecture, we consider the simple $\mathcal{L}^2$ regularization (*weight-decay*).

5.3.2 Model investigation: architectures for neural ordinary differential equation

The choice of neural network architecture is subject to scrutiny given that it needs to be able to handle the non-linearity term of the Burgers equation, as seen in the section 4.2. While multiple architectures such as convolutional neural network or auto-encoder [LC20] are considered; the universal approximation theorem [HSW89] tells us that feed-forward network architectures composed of a single hidden layer (and enough neurons) are sufficient to approximate any measurable function $f : \mathcal{R}^r \to \mathcal{R}$. Therefore, we investigate in a first time a standard multi-layer feed-forward network architecture as a direct approximation model F_θ, before considering more appropriate architecture. We highlight the fact that such architecture will be used to model the closure term ϵ_θ in the POD-based ROM.

Activation function We fix a linear activation function for the output layer has the regression problem we are solving requires unbounded output. For the hidden layers we settle for the *hyperbolic tangent* $t(z) = \tanh(z)$: its differential $h'(z) = 1 - \tanh^2(z)$ make it useful for back-propagation, its range $\tanh(z) \in [-1, 1]$ is of interest as it suffer less from vanishing gradient problem than the *logistic sigmoid* function $\sigma(z) \in [0, 1]$; in addition to be continuously differentiable, a requirement for the back-propagation through ODEs, therefore excluding functions like $\text{ReLU}(z)$.

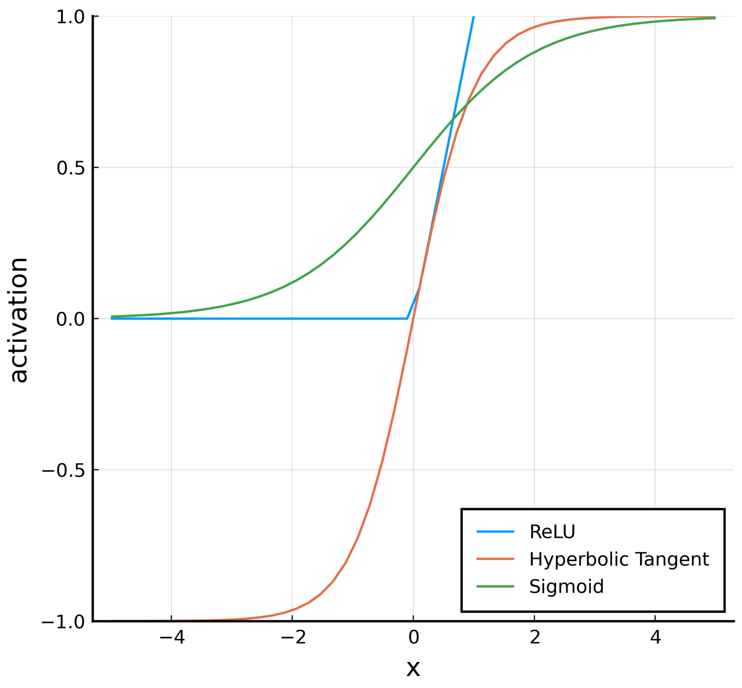
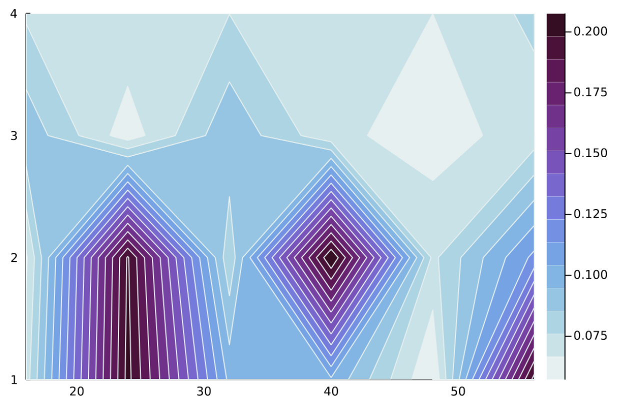


FIGURE 5.4: Visualization of activation functions

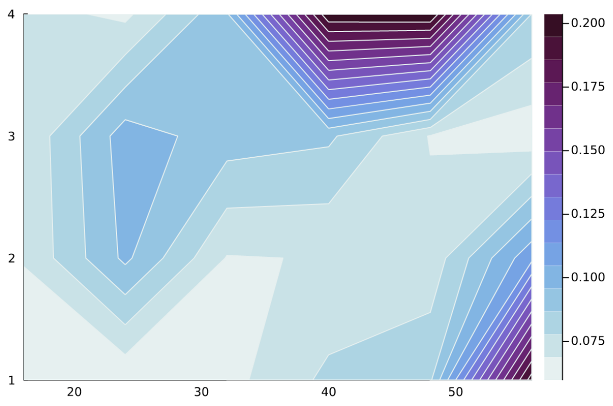
Initialization Initializing weights close to 0 would improve training as disturbing a nearly constant values is easier than dynamic random values. However, weights too close to zero will make variance of the input decreases through each hidden layers, leading to again a vanishing gradient problem. The *Xavier* initialization, which assume that variance of the objective function to be constant through each hidden layers was developed with this problem in mind. While it might be less efficient for neural ODE, we consider this initialization at least for our hyper-parameter tuning as a way to avoid the above problem.

Multi-layer feed-forward neural network

To begin, we consider a ROM where pure NODE approximates the right-hand side of the equation (i.e. model 3 in Table 4.1). We investigate the efficiency of the multi-layer feed-forward neural network to approximate operator F_θ. The first interrogation concerns the number of hidden layers and neurons which will compose the feed-forward network. We perform a search in the hyper-parameter space for different number of hidden layers and neurons, then evaluate the final intrusive objective function results. The neural network is trained on on a small data-set of 196 initial conditions with the diffusion parameter $\nu = 0.04$. It uses an initial learning rate $lr = 0.003$, a fixed batch size $\beta = 32$, regularization is also applied with a fixed weight decay $\lambda = 10^{-7}$ and addition of Gaussian noises to the input data ($\sigma = 0., 0.05$). We use the mean square error (MSE) as an objective function.



(A) No Gaussian noise



(B) Gaussian noise ($\rho = 0.05$)

FIGURE 5.5: Validation objective functions (MSE) resulting from training feed-forward network for different number of hidden layers and neurons. Gaussian noise are added to the prediction $\hat{u}$ considered to avoid over-fitting with ρ being the standard-deviation.

As seen in the Figure 5.5, regularization provides more accurate results on the validation data-set. The selection of the deep feed-forward neural network architecture as an approximation of F_θ is a compromise between the number of hidden layers and neurons. Increasing the number of hidden layers enables the neural network to produce increasingly more abstract patterns f_L by mapping the input f_0 through a series of non-linear activation functions σ_i and parametrization functions α_i. Withal, deeper neural networks are also more difficult to train: after propagating through several hidden layers the *vanishing gradient problem* might arise, the loss surfaces have many local minima giving similar performance on validation set making it more difficult to find global minimum or good local minima [CHM+14].

Hyper-parameter	values
Weight decay λ	$[10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]$
Noise	[.35, .3, .25, .2, .15, .1, .05, .01]
Batch size	[8, 16, 32]
Learning rate	[0.01, 0.003, 0.001]

TABLE 5.1: Hyper-parameter tuning of the feed-forward neural network. The error metric stabilised at $\mathcal{L}(\theta) = 0.01$ after 100 epochs.

Nevertheless, with regards to the results obtains it can be observed that the MSE remains high. To investigate further, we fixed a feed-forward network consisting of 3 layers, 32 neurons and evaluated on a wider range of parameters: regularization (weight decay and noise) parameters, learning rate, batch size, initial condition set size and number of epochs. The objective function results obtained, and interpolation on test set did not improve beyond $\mathcal{L}(\theta) = 0.01$, confirming the importance of handling the non-linearity of the Burgers equations.

Convolutional neural network

With regards to the terms composing the Burgers equation 5.3 in its viscous and inviscid form, a choice of a CNN rather than a fully-connected FNN architecture appears more appropriate.

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2} - \frac{1}{2}\frac{\partial u^2}{\partial x} = \mathcal{L}(u) + \mathcal{N}(u) \tag{5.3}$$

1. The kernel from the convolutional layers, assuming the choice of a sufficient size, should be able to learn efficiently the first and second-order derivatives respectively associated to the non-linear and linear term of the equation5.3, as it will focus on the local neighboring values of u_i^n on the discretise spatial domain.
2. The main difficulty resides in learning the non-linear term. As seen previously, it would requires to work on an architecture able to learn such term (for instance using a cross correlation hidden layer). Instead, a simple option consists into providing to the neural network u^2, as the equation 5.3 contains a quadratic term. Therefore, instead of receiving $u \in \mathbf{R}^{N_x}$, the neural network will have a layer generating a 2-dimensional matrix $U \in \mathbf{R}^{N_x \times 2} : [u, u^2]$.

In a similar manner to FNN, we perform a search in the parameters spaces to tune the CNN architecture, see Table 5.2. Despite the short amount of epochs $e = 100$, the search enable us to discard less appropriate architectures. While the optimal parameters correspond to the extreme

Hyper-parameter	Values	Optimal
Channels	[2, 2, 1], [2, 4, 4, 2, 1], [2, 4, 8, 8, 4, 2, 1]	[2, 4, 8, 8, 4, 2, 1]
Kernels	[3, 5, 9]	9
Batch size	[8, 16, 32]	8
Learning rate	[0.01, 0.003, 0.001]	0.1

TABLE 5.2: Description of the convolutional neural network

parameters, which suggest it might be worthy to extend the parameters' range, we eventually settle on the architectures described in the table 5.3 to perform the subsequent long-term training of our Neural ODE, given that the computational resources remain limited. Indeed, the small-scale of the grid discretization does not make GPU more efficient than CPU. The number of time step being the limiting factor.

Layer	Type	Channels	Activation	Parameters
1	Periodic extension	/	/	/
2	$u \mapsto [u, u^2]$	(1, 2)	/	/
3	Convolutional	(2, 4)	Hyperbolic Tan	76
4	Convolutional	(4, 8)	Hyperbolic Tan	296
5	Convolutional	(8, 8)	Hyperbolic Tan	584
6	Convolutional	(8, 4)	Hyperbolic Tan	292
7	Convolutional	(4, 2)	Hyperbolic Tan	74
8	Convolutional	(2, 1)	Hyperbolic Tan	19
9	Linear - Reduce	/	/	/

TABLE 5.3: Convolutional neural network architecture used to learn non-linear Burgers equation using Neural ODE with parameters $\theta \in \mathbb{R}^{1341}$. The kernel provides an efficient method to learn first and second-order derivative, while the quadratic term is directly provided to remove the difficulty of learning the non-linear term.

5.4 Approximation of Burgers equation through down-scaled based ROM

Before evaluating the models described in Table 4.2, we are interesting into the approximation of the right-hand side through pure neural ODE. This analysis similar to the one performed in the chapter 3 but applied to a non-linear equation. It also extends directly the previous section whose model investigation focused mostly on pure neural differential equations. We consider the reduced model based on down-scaling of a FOM.

5.4.1 Data and metrics

Parameters	Values
Solutions	256 (192 training & 64 validation)
Epochs	500
Batch size	16
Optimizer	Adam - $lr = 0.001$, $\beta = (0.9, 0.999)$
Regularization	Weight decay $\lambda = 10^{-7}$
ODE solver	Tsitouras 5/4 Runge-Kutta scheme (Tsit5)
Method	Interpolating Adjoint + Automatic Differentiation

TABLE 5.4: Neural ODE training parameters

First, the training and the evaluation of the model F_θ (e.g model 1 in Table 4.2) are performed on a set of solutions $u \in \mathbb{R}^{64\times 64}$ to the *viscous* Burgers equation based on the Fourier series initial conditions (see Equation 5.2). Similar training and validation data-sets were generated for the *inviscid* Burgers equation.
The set is composed of $\mathbb{N}_s = 256$ solutions $u \in \mathbb{R}^{\mathbb{N}_x \times \mathbb{N}_t}$, such that the FOM is discretized on $\mathbb{N}_x = 64$ grid points. The time step Δt use an adaptive algorithm with an absolute and relative tolerance fixed at $1e-6$ (i.e. locally correct to 6 digits), but only $\mathbb{N}_t = 64$ time snapshots are conserved for experiments. $\frac{3}{4}$ of the solutions are used for training and $\frac{1}{4}$ for validation. As stated earlier on, the neural network model F_θ receive the initial condition $u_0 = u(0,x) \in \mathbb{R}^{\mathbb{N}_x}$ for each solution as an input to reconstruct $u_\theta(t,x) \in \mathbb{R}^{\mathbb{N}_x \times \mathbb{N}_t}$, the prediction of the solution to Burgers equation.

Like in the architecture investigation, we use the a-posteriori approach to perform ridge loss optimization of the model F_{θ^*} where

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, C(\theta) = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}_{\text{mse}}(\theta) + \lambda \left\|\theta\right\|_2^2 \tag{5.4}$$

$$\mathcal{L}_{\text{mse}}(\theta) = \frac{1}{\mathbb{N}_x \mathbb{N}_t} \sum_{i=1}^{\mathbb{N}_x} \sum_{j=1}^{\mathbb{N}_t} \left(u_{\theta i}^j - u_i^j\right)^2 \tag{5.5}$$

Given that the Burgers equation is considered a non-stiff equation, the Tsitouras 5/4 Runge-Kutta scheme (Tsit5) is used by the Neural ODE during training, a method known for its relative stability.

5.4.2 Results: Viscous Burgers equation

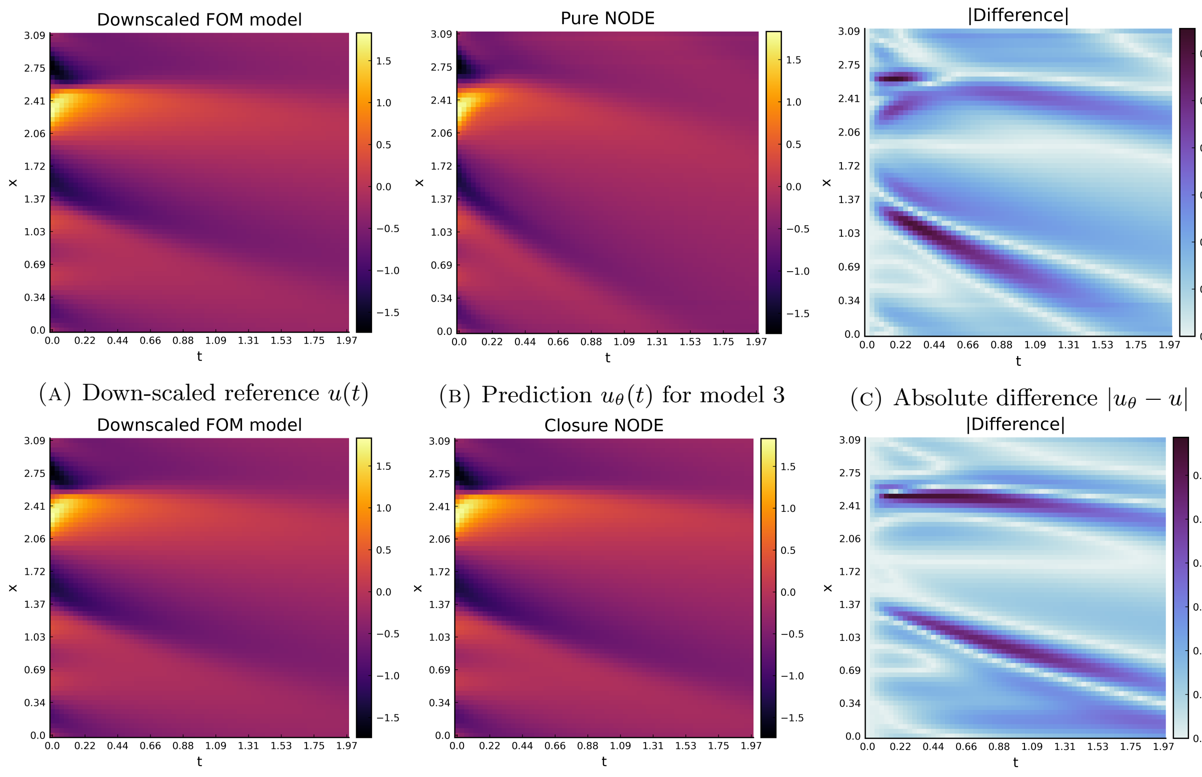
The model is evaluated on a test data-sets composed of 32 solutions not used during training. For each of these reference solutions, the initial condition is feed to the NODE which generates a prediction for the same reference time steps.

The Figure 5.6 provides an instance of prediction obtained from the model which generated the best results.

The results obtained after evaluating the different models and approaches on the test-dataset are visible in the table 5.5. Each one of these models were train at least 3 times. We then selected the model which generated the optimal results on the test data-set. It can been observed than the difference between results obtained from the DO and OD approach remains quite close, even if the Optimise-then-Discretize provided slightly better results.

5.4.3 Results: Inviscid Burgers equation

The previous experiments emphasised the efficiency of models obtained through NODEs when applied to viscous Burgers equation 4.3. This choice was made due to the absence of discontinuities, shocks, with the presence of a linear term. A point of interest lies in evaluating the efficiency of the method to interpolate in the case of inviscid Burgers equation ($\nu \to 0$). This time, the previously selected models, see Tables 5.3 and 5.4, were trained and evaluated on the inviscid Burgers equation



(A) Down-scaled reference $u(t)$ (B) Prediction $u_\theta(t)$ for model 3 (C) Absolute difference $|u_\theta - u|$

(D) True solution $u = F(u_0)$ (E) Prediction $u_\theta(t)$ for model 4 (F) Absolute difference $|u_\theta - u|$

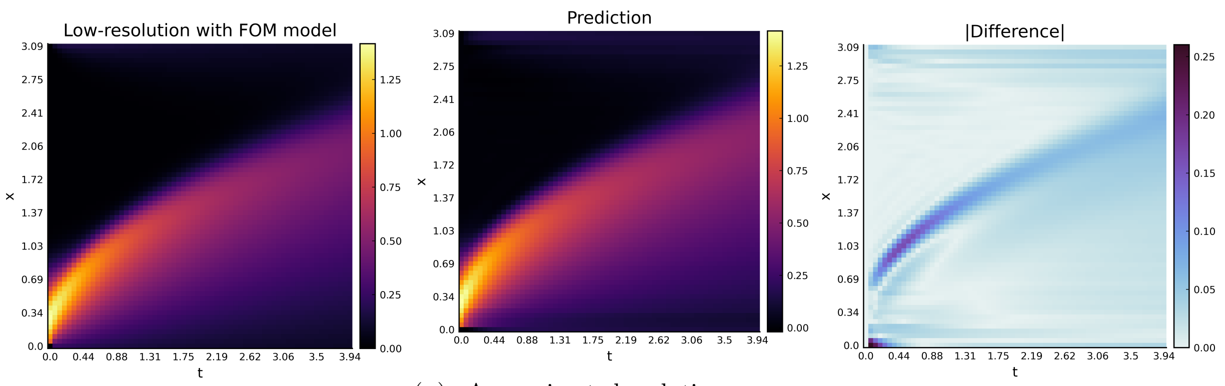
FIGURE 5.6: An instance of a prediction to Viscous Burgers equation interpolated from an initial condition vector u_0 from the test data-set (not validation data-set). $x \in [0, \pi]$, $t \in [0, 2]$, $\nu = 0.04$, $\mu = 10$ for the pure NODE and the neural closure models. In this specific case, the mean square error between the pure NODE model and reference solution is $\mathcal{L}_\theta \approx 0.03$. For the neural closure model $\mathcal{L}_\theta \approx 0.002$.

N	Model	Mean square error (MSE)	
		Optimise-Discretize	Discretize-Optimise
2	Baseline (coarse)	0.064	0.0734
3	Pure NODE	0.0076	0.0092
4	Neural closure model	0.0022	0.0031

TABLE 5.5: Mean square error for the different evaluated model on the Burgers equation on a test data-set of 32 solutions. Model were trained using the Optimise-then-Discretize approach (continuous model with Tsitouras5 method),and the Discretise-then-Optimise approach (continuous model acting as discrete model with Forward Euler)

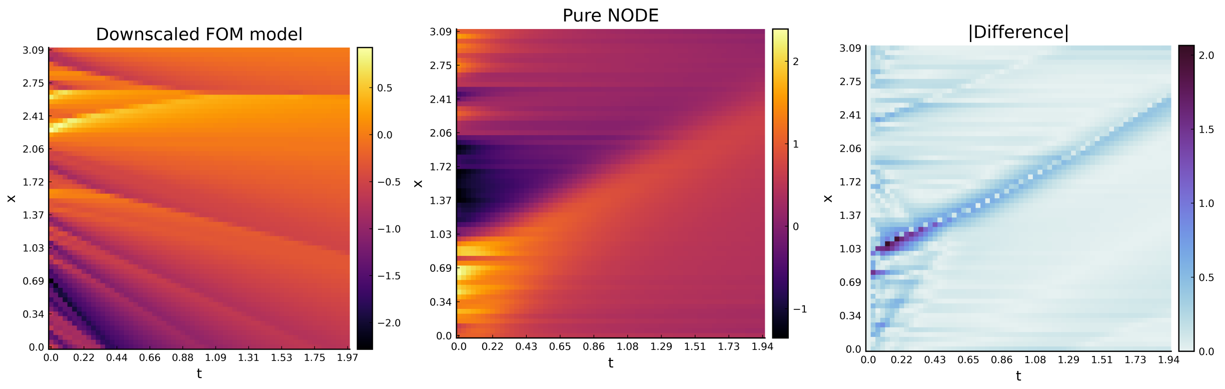
referenced solution obtained from the initial conditions u_0 5.2 with the Godunov schema described in the methodology chapter.

Unlike viscous Burgers equation, the results obtained suggest that while the NODE succeed to captures the main dynamics of the ODE, the discontinuities of the ODE requires a different approach to be covered properly. The Figure 5.9 shows how discontinuities get smoothed over time.



(A) True solution $u = F(u_0)$ (B) Approximated solution $u_\theta = F_\theta(u_0)$ (C) Absolute difference $|u_\theta - u|$

Figure 5.7: The pure NODE model also predict hopeful results on test data-set using different initial conditions and a longer time series(extrapolation). Gaussian-based initial condition with single wave with $x \in [0, \pi]$, $t \in [0, 4]$, $\nu = 0.04$. The mean square error is $\mathcal{L} \approx 9 \times 10^{-4}$.



(A) True solution $u = F(u_0)$ (B) Approximated solution $u_\theta = F_\theta(u_0)$ (C) Absolute difference $|u_\theta - u|$

Figure 5.8: An instance of a prediction to Inviscid Burgers equation interpolated from an initial condition vector u_0 from the test data-set (not validation data-set). $x \in [0, \pi]$, $t \in [0, 2]$, $\nu = 0$, $\mu = 100$ for the pure NODE.

5.4.4 Discussion

Firstly, 500 epochs might be considered as a short training time. However, we discard this concern as we highlight the fact that the number of epochs has been fix based on hyper-parameter tuning performed while training these neural networks (from 100 epochs to 500 epochs). The choice of trajectory fitting rather than derivative fitting associated to a large number of trajectories in the training data-set, resulted in a slower but more optimal training of the models as well. Based on the above experiments, we are able to give insight to some of our research questions.

1. First of all, it appears that the choice of the neural network architecture is of primary importance to parametrize the operator F_θ. The non-linear terms play a role of importance to infer an approximated operator whose accuracy would make of it an alternative to the coarse-grid baseline model. In the viscous Burgers equation instance, it was observed during the early investigation than the absence of the non-linear term u^2 leads to a mean square error $L_{\text{mse}}(\theta) \approx [0.05, 0.1]$ on the validation data-set using a pure NODE. In comparison, this

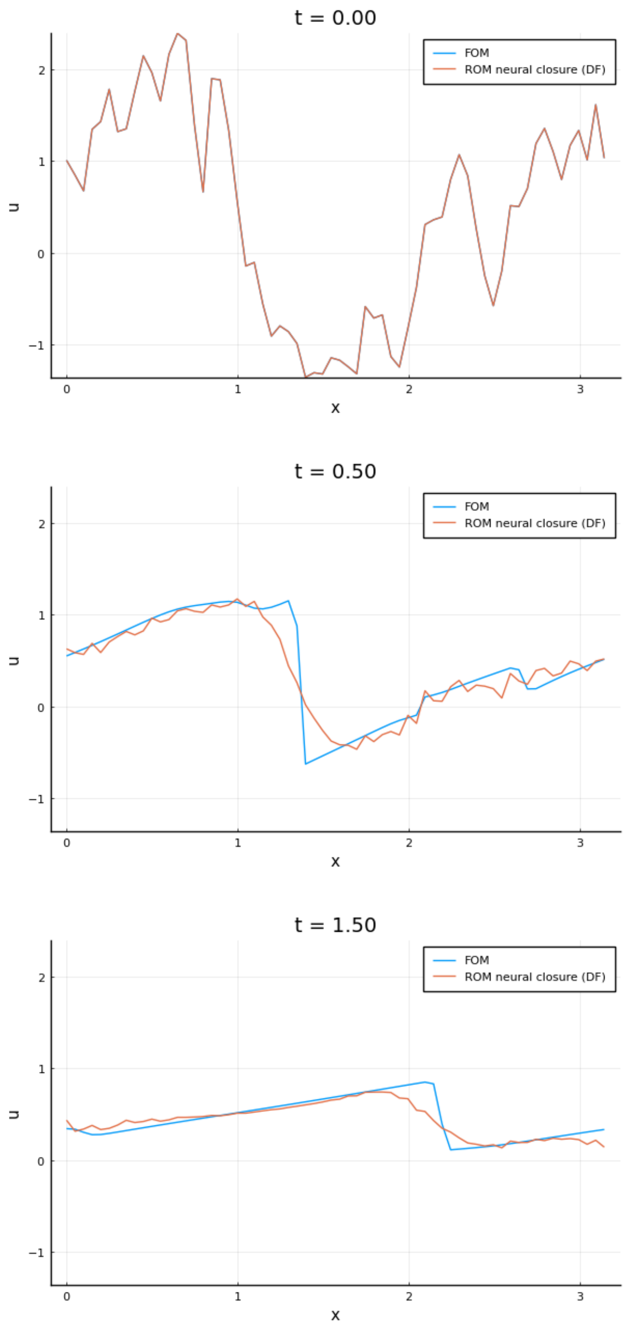


FIGURE 5.9: Prediction of a solution to Inviscid Burgers equation at time $t = [0, 0.5, 1.5]$. The prediction failed to handle the shock of the ODE. $x \in [0, \pi]$, $t \in [0, 2]$, $\nu = 0$, $\mu = 100$ for the pure NODE.

same model would give a MSE of $L_{\text{mse}}(\theta) \approx 0.001$ both for O-D and D-O approaches when including the non-linear term.

2. Next, not every dynamics property of a non-linear PDE can be inferred, at least on a coarse model. While we have observed than the viscous Burgers equation could successfully be modeled both with a encompassing neural network or a closure model, this approach failed on the inviscid case. The inviscid Burgers equation (and viscous where the diffusion $\nu \to 0$, also known as high Reynolds number), is known to be notoriously difficult to solve. In the

experiments of this thesis, not only both pure NODE and closure models failed to handle the shocks appearing over time, but the modeling of smooth sections was also impacted. This latest point suggests that the quadratic term of the differential equations brings some stability during the training of the neural network, in a similar way to the addition of Gaussian noise for regularization. We formulate the hypothesis that such instances can only be handled on a fine-grid. However, this defeat the purpose of our experiments which aimed at evaluating efficiency of NODEs on low-dimensional simulations.

3. Finally, we highlight the fact that these models were also trained on the Korteweg–De Vries equation, a mathematical model of waves on a shallow water surface. The main point of considering this equation lies in the fact that the Burgers equation remains a simple case given that its behavior consists in traveling wave and the diffusion parameter make its long-term behavior is easily predictable. This was confirmed by the Figure 5.7. On the contrary, the Korteweg–De Vries equation does not suffer from dissipation and could be used for long-term prediction. Additionally, its linear (Third-order derivative u_{xxx}) and non-linear components (u^2) were close to the viscous Burgers equation and suggested the same pure NODE model 3 would make accurate prediction when trained on it. It resulted than the training failed to converge to a local nor global minimum. This suggests that, just like for the inviscid Burgers equation, a consideration of the terms of the differential equations is not enough to get a proper architecture, dynamics (in the case of Korteweg–De Vries it would be high-frequencies) need to be handled individually.

5.5 Approximation of Burgers equation through POD based ROM

5.5.1 Data and metrics

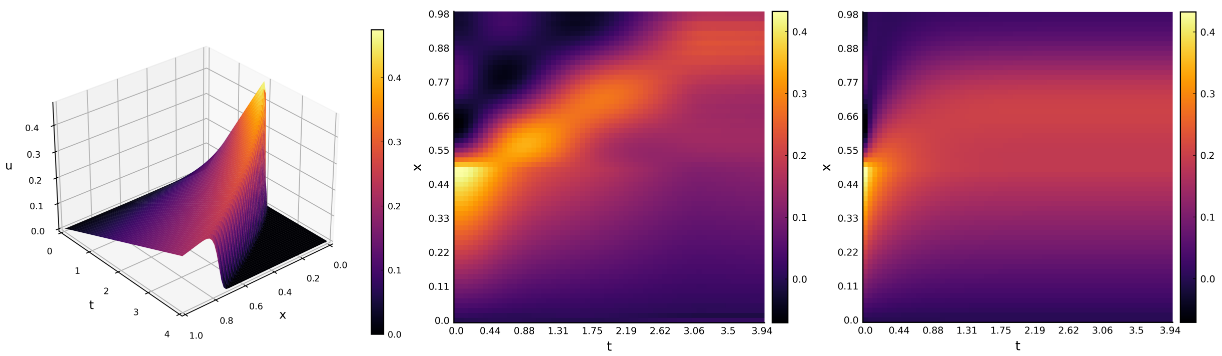
First of all, we reproduce some of the experiments from Maulik. To do so, we complement the data-sets introduced in the section 5.3.1 with a analytical solution to the Burgers equation based on a Gaussian function

$$u(t,x) = \frac{\frac{x}{t+1}}{1 + \sqrt{\frac{t+1}{\exp\left(\frac{1}{8\nu}\right)}} \exp\left(\frac{1}{\nu}\frac{x^2}{4t+4}\right)} \tag{5.6}$$

such that training will be using the initial and boundary value conditions

$$u_0(x) = \frac{x}{1 + \sqrt{\frac{1}{\exp\left(\frac{1}{8\nu}\right)}} \exp\left(\frac{1}{\nu}\frac{x^2}{4}\right)}, \quad x \in \Omega = [0,1], \quad u(0,t) = u(1,t) = 0 \tag{5.7}$$

This simple pattern allows us to experiment in reconstructing the closure term in the case of ROM such as POD-Galerkin projection to capture the full evolution of the solution.



(A) Downscaled surface (64 × 64)

(B) POD (64 × 64)

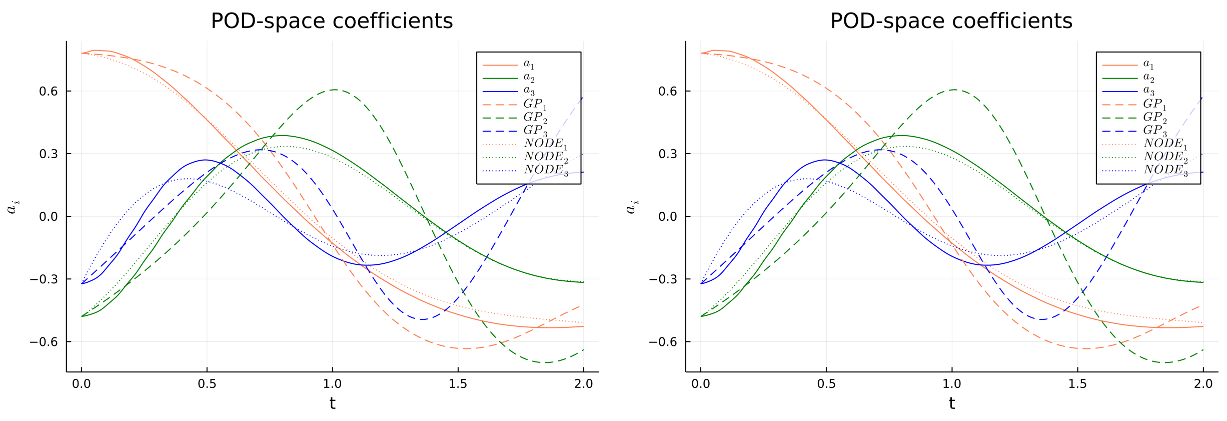
(C) POD GP (64 × 64)

FIGURE 5.10: Evolution of a solution to inviscid Burgers equation using analytical initial conditions. A discontinuity (shock wave) can be observed. With $x \in [0., 1.]$, $t \in [0., 4.]$, $\nu \to 0$

To quantify the difference between a reconstruction and the ground truth, the training use an a-posteriori approach with a mean-squared error based loss function $\mathcal{L}_{\text{mse}}(\theta)$ with Thikonov regularization, while the validation complement it with along with the relative difference error (RDE)

$$\mathcal{L}_{\text{rde}}(\theta) = \frac{a_\theta - a_{\text{ref}}}{a_{\text{ref}}} \tag{5.8}$$

5.5.2 Results: Application to advecting shock case



(A) $a_i(t) = \Phi_r^\intercal \hat{u}$ (B) $a_i(t) = \Phi_r^\intercal u$, Φ_r computed with temporal mean

Figure 5.11: Comparison of the true coefficients a_i, the coefficients obtained from Galerkin projection (GP_i) and determined with a pure NODE ($NODE_i$), of the reduced POD basis associated to the analytical solution to Burgers equation

Firstly, we reproduced the results obtained from Maulik and Gupta[MML+20, GL21] publications to insure the feasibility of our experiments. We use the analytical solution previously introduced as initial condition and compute the POD basis (we refer to the methodology chapter 4 for further details) along with the coefficients of the reduced basis $a_i(t)$ obtained by subtracting the temporal mean to the snapshots.

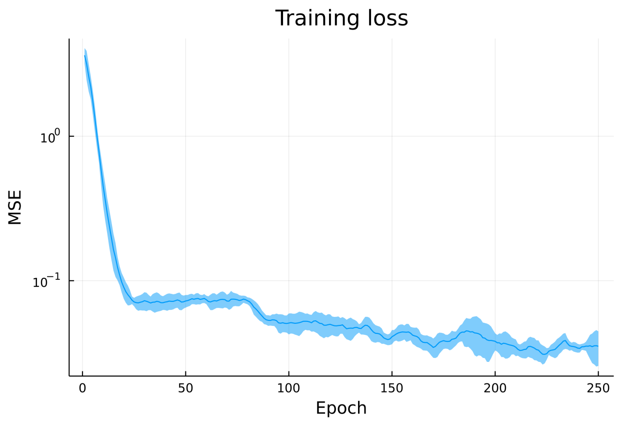


Figure 5.12: Convergence of the NODE training over 250 epochs (mean and standard deviation over 5 training) using a-posteriori approach with mean square error metric

As seen in the figures 5.11 and 5.14, NODE can be efficiently used to determined model the true coefficients from the reduced POD basis, more efficiently that through the Galerkin projection. As evoked in the methodology chapter, the purpose of this idea come with the fact that $u(t,x) \approx \bar{u}(x) + \Phi_r^\intercal(x)a(t)$ can be approximated through POD basis and these coefficients. In the case where the reduced basis Φ_r are obtained without subtracting the temporal mean, such as $u(t,x) \approx \Phi_r^\intercal(x)a(t)$, it makes it possible to determine a ROM $u_{\text{rom}}(t,x)$ from $a(0) = \Phi(u(0,x))$ with a computational complexity drastically reduced (by the number of modes discarded).

However, we highlight multiple shortcomings in the results from the previous section. While Maulik [MML+20] investigated the reconstruction, Gupta [GL21] main goals aimed at learning time-series

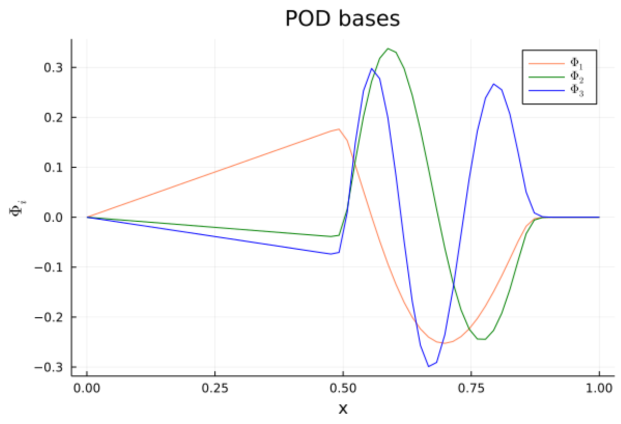
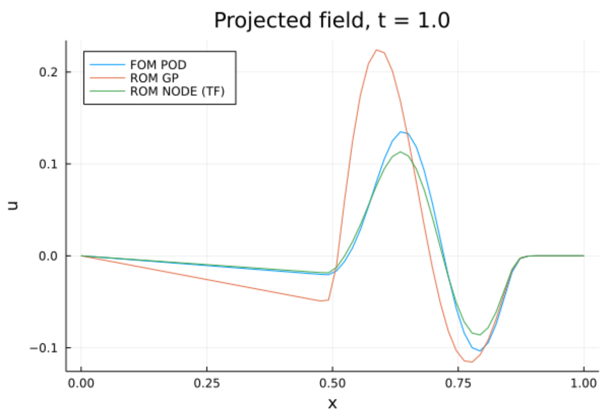
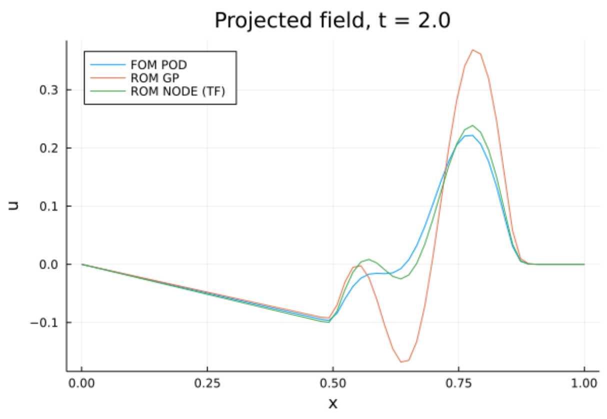


FIGURE 5.13: Three first POD basis vectors (columns of Φ)

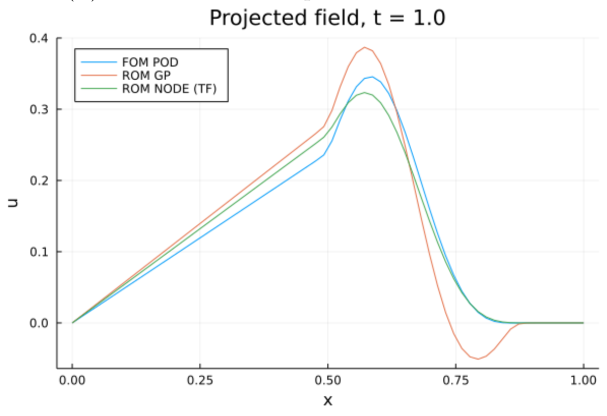
for long-range prediction. The experiments were performed on the single advecting shock such that the reduced POD-basis clearly over-fit this case (See the Figure 5.13) and a generalisation to a larger data-set was not covered. With this information in mind, having studied the methodology behind POD-ROM and Galerkin projection, and finally, having analysed the capacity of NODEs to learn the underlying dynamics of a model from large data-set in the previous sections; we make the hypothesis that it is possible to extend this work to handle a larger range of values. This would be considered as our next step to extend the research of this thesis.



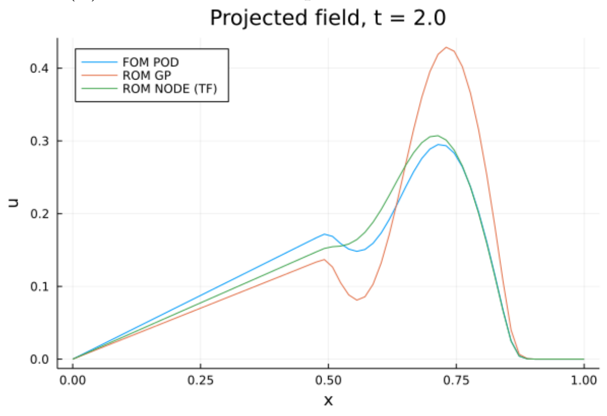
(A) At $t = 1$ with temporal mean subtracted



(B) At $t = 2$ with temporal mean subtracted



(C) At $t = 1$ with temporal mean included in reduced basis



(D) At $t = 2$ with temporal mean included in reduced basis

FIGURE 5.14: Field reconstruction ability for NODE model on the advecting shock reference solution to Burgers equation. Comparison with and without temporal mean subtracted (included in POD-reduced basis)

Chapter 6

Conclusion and future work

6.1 Conclusion

In this thesis we assessed neural ordinary differential equation, a class of neural networks emerging in the field of scientific computing, as an instance of data-driven reconstruction method used for reduced order modeling.

In the chapter 3, the analysis of the simple example such as the Diffusion equation gave us insights on the capacity of these neural networks to perform regression tasks accurately while reconstituting the underlying physics of the full-order model. Indeed, using a single layer, enough to model the diffusion equation, we observed that NODE would model an operator F_θ close to the true operator F. From this knowledge, we inferred that the choice of a neural ordinary differential equation architecture would benefit of evaluating the terms composing the differential equation.

This assumption was confirmed in the chapter 5 where a large range of neural network architectures were evaluated on the non-linear Burgers equation. It was determined that neural networks such as CNN were appropriate architectures to fully encompass the right-hand side of the evaluated differential equations, given that the convolutional layer's kernel local influence would help learn derivatives (similarly to the finite difference method). On the other side, feed-forward neural networks appear as a most accurate architecture to learn closure terms that model the effect of the discarded ROM modes, that is to say model error or unresolved tendency. Next, using down-scaled based reduced-order model, we have seen that the introduction of prior knowledge such as the time continuity using optimise-then-discretize approach, the default behavior when using NODE; or the mimicking of discrete models using a Forward Euler ODE solving method with a discretize-then-optimise approach, would in every case offers results superior to the baseline coarse models. In all logic, these results were further improved when using a closure model based on combination of the baseline model with a neural network as a closure term. However, while being efficiency to learn high-level dynamics, we also observed that these data-driven reconstruction methods were not applicable to all problems nor able to learn all system dynamics such as high-frequencies or discontinuities. For the Burgers equation, it was found that the second-order derivative term would bring stability to the model in a similar way of additive noise for regularization, while its absence would result in a complete inability of the model to approximate shocks.

6.2 Future research

This thesis can be further extended in multiple directions. First of all, we come back to the last assessments of this thesis. As seen with the POD-based reduced order modeling, NODE appears as a hopeful solution to learn discarded modes. Experiments performed using a single advecting shock initial condition proved that NODE models could be used to approximate the true time-dependent coefficients $a(t)$ and therefore to compute an approximation $u(t) \approx \bar{u} + \Phi a(t)$. The latest goal of this thesis consisted into extending this experiment to a large data-set, to observe if from a POD basis computed on a training data-set, it is possible to infer predict approximated solution from unknown initial condition using a system whose complexity would be reduced by the number of bases discarded. Next, we remind that all the experiments performed through this thesis on data-driven reconstruction method were based on the usage of neural differential equations, however other models could have been considered. More especially, we could consider the physics-informed neural networks which, in complete contrast to NODE whose main objective consists in approximating an operator F_θ, aim to predict a full trajectory rather than a single time step. Eventually, we realise that the rising quantity of machine learning techniques and neural network models provides a large range of possibilities to extends the field of reduced order model techniques either to at least reduce the computational resources involved or increase the accuracy of these models.

Appendix A

Eigenvalue decomposition based proper orthogonal decomposition

Another approach to proper orthogonal decomposition (POD) makes usage of the eigenvalue decomposition. While the SVD is suitable for non-square matrices and remain in a domain of real values as long as the matrix decomposed $U \in \mathbb{R}^{n \times m}$; the eigenproblem method is only usable with square matrices. Asymmetric real matrices might lead to complex eigenvalues and eigenvectors. In the POD context, this make this approach less optimal for large data-set.

The first step consists in generating a snapshot matrix in the discrete domain, composed of a set of snapshots of the normalised solution $\hat{u}(t, x)$ over time ($\bar{u}$ are the averaged spacial values over time).

$$S = [\tilde{u}^1 | \tilde{u}^2 | ... | \tilde{u}_t^N] \in \mathbb{R}^{\mathbb{N}_x \times \mathbb{N}_t} \tag{A.1}$$

$$\tilde{u}^i = u^i - \bar{u} \tag{A.2}$$

$$\bar{u} = \frac{1}{N_t} \sum_{i=1}^{\mathbb{N}_t} u^i \tag{A.3}$$

Next step consists in resolving an eigenvalue problem, in what is called the *method of snapshots*, using a correlation matrix $C = S^\intercal S$

$$CW = \Lambda W \tag{A.4}$$

$$\Lambda = \lambda_1, ..., \lambda_{\mathbb{N}_s} \in \mathbb{R}^{\mathbb{N}_t \times \mathbb{N}_t} \tag{A.5}$$

$$W \in \mathbb{R}^{N_t \times \mathbb{N}_t} \tag{A.6}$$

With Λ eigenvalues diagonal matrix and W eigenvector matrix. The POD basis matrix can simply be obtained through the equation $\theta = SW \in \mathbb{R}^{\mathbb{N}_x \times \mathbb{N}_t}$, which spans the space $\mathcal{X}^t$. However, as we are looking for a reduced basis, we will select the first $\mathbb{N}_r$ columns of the POD basis matrix θ such that $\mathbb{N}_r < \mathbb{N}_s$; the reduce basis $\Phi \in \mathbb{R}^{\mathbb{N}_x \times \mathbb{N}_r}$ obtained spans the new space $\mathcal{X}_r$:

$$\mathcal{X}^r = \text{span}\{\Phi^1 ... \Phi^{\mathbb{N}_r}\} \tag{A.7}$$

Eventually we are able to obtain the POD ROM $\hat{S}$ from the reduce basis coefficients A (catching the main features with regards to time)

$$A = \Phi^{\intercal} S \in \mathbb{R}^{\mathbb{N}_r \times \mathbb{N}_t} \tag{A.8}$$

$$\hat{S} = [\hat{u}^1 | ... | \hat{u}_t^{\mathbb{N}}] \approx \Phi A \in^{\mathbb{N}_x \times \mathbb{N}_t} \tag{A.9}$$

Algorithm 4: Eigenproblem based Proper orthogonal decomposition

```
Require: U, $k$
1: $S \leftarrow U - \text{mean}(U)$    ▷ Subtract temporal mean
2: $C \leftarrow S^{\intercal} S$    ▷ Covariance matrix
3: $\lambda$, $W \leftarrow \text{eigen}(C)$    ▷ Solve eigenvalue problem $CW = \Lambda W$
4: $i \leftarrow \text{sort}(|\lambda|)$    ▷ Sort by descending order
5: $\lambda, W \leftarrow \lambda[i], W[:, i]$
6: $D \leftarrow \text{Diagonal}(1/\sqrt{|\lambda|})$
7: $\Phi \leftarrow SWD$    ▷ POD basis
8: $\hat{U} \leftarrow \Phi[1:k]\ \Phi^{\intercal}[1:k]$    ▷ Reduce model
9: return $\Phi$, $\hat{U}$
```

Appendix B

Neural network architectures

B.1 Feedforward neural network

Feedforward neural network (FNN) are the most common architecture encountered in deep learning. They consists into a network of neurons F_θ, a series of affine transformation followed by an element-wise non-linear activation function.

$$z_i(x) = W_i x + b_i \tag{B.1}$$

$$F_\theta(x) = \sigma_L \circ f_L \circ \cdot \circ \sigma_1 \circ f_1(x) \tag{B.2}$$

With $W_i \in \mathbb{R}^{H_i \times H_{i-1}}$ the weight matrix, $b_i \in \mathbb{R}^{H_i}$ the bias vector, σ_i the activation function, H_i number of neurons in layer i, L the total number of layers and $\theta = W_1, b_1, \cdot, W_L, b_L$ the neural network set of parameters. Choice of activation function σ depends on the application for which the FNN will be used; hyperbolic tangent or sigmoid would be suitable for case like neuralODE which requires continuity, ReLU and Leaky ReLU would be suitable for application which might face sparse gradient, inter alia.

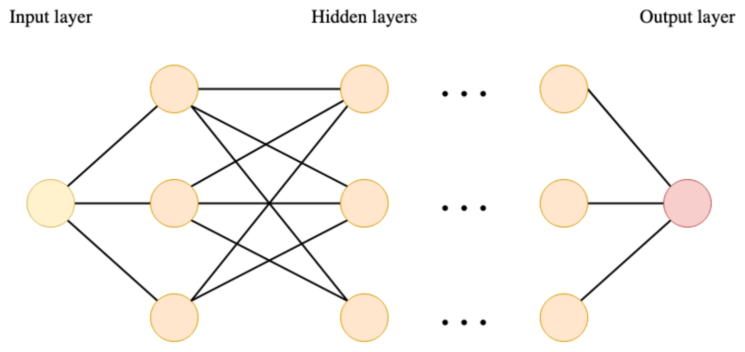


Figure B.1: Example of fully connected feedforward neural network

B.2 Convolutional neural network

Convolutional neural networks (CNN) are renowned for their capacity to consider local rather than global properties of the data. Such architecture are of interest in scientific computing application, where the kernels can mi-mick the relationship between discretized points (i.e. Central finite difference). The convolutional layer can be defined as a feature map such that each entry of its input is mapped to a local sample of the previous layer inputs, through a sliding convolutional kernel and activation function. Let's consider a 2-dimensional case with a feature map $\mathcal{H}$, the convolution operator can be represented as:

$$\mathcal{H}^h_{i,j,n} = \sigma^h \left(\sum_{(k,l)\in\mathcal{N}} \sum_{m\in C} \mathcal{K}_{k,l,m,n} H^{h-1}_{i+k,j+l,m} + \mathcal{B}^h_{i,j,n} \right) \tag{B.3}$$

Where $\mathcal{H}^h_{i,j,n}$ is the feature map at current layer h for element at row i, column j and channel n. $\mathcal{K}_{k,l,m,n}$ is the bank kernel with k, l the rows and columns offsets for each $2D$ kernel, m the channel input from the preceding layer and n the kernel associated to the current layer channel n. $H^{h-1}_{i+k,j+l,m}$ is the feature map of the previous layer $h-1$, where kernel offsets k, l are added to the input i, j. $\mathcal{B}^h_{i,j,n}$ and σ^h are respectively the bias term and activation function associated to this current layer.

Bibliography

[APS$^+$21] Shady E. Ahmed, Suraj Pawar, Omer San, Adil Rasheed, Traian Iliescu, and Bernd R. Noack. On closures for reduced order models—a spectrum of first-principle to machine-learned avenues. *Physics of Fluids*, 33(9):091301, sep 2021.

[BBBCD00] Michael Bartholomew-Biggs, Steven Brown, Bruce Christianson, and Laurence Dixon. Automatic differentiation of algorithms. *Journal of Computational and Applied Mathematics*, 124(1):171–190, 2000. Numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations.

[BDH$^+$86] C Basdevant, M Deville, P Haldenwang, J.M Lacroix, J Ouazzani, R Peyret, P Orlandi, and A.T Patera. Spectral and finite difference solutions of the burgers equation. *Computers & Fluids*, 14(1):23–41, 1986.

[BMNP04] Maxime Barrault, Yvon Maday, Ngoc Cuong Nguyen, and Anthony T. Patera. An 'empirical interpolation' method: application to efficient reduced-basis discretization of partial differential equations. *Comptes Rendus Mathematique*, 339(9):667–672, 2004.

[BPK16] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[Bur48] Johannes Martinus Burgers. A mathematical model illustrating the theory of turbulence. *Advances in Applied Mechanics*, 1:171–199, 1948.

[Cal20] Ovidiu Calin. *Deep Learning Architectures*. Springer Cham, 2020.

[CHM$^+$14] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. 2014.

[CRBD18] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. 2018.

[CS10] Saifon Chaturantabut and Danny C. Sorensen. Nonlinear model reduction via discrete empirical interpolation. *SIAM Journal on Scientific Computing*, 32(5):2737–2764, 2010.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. 12(null):2121–2159, july 2011.

[DJG95] Korteweg D. J. and de Vries G. On the change of form of long waves advancing in a rectangular canal, and on a new type of long stationary waves. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 39(240):422–443, 1895.

[Fou09] Jean Baptiste Joseph Fourier. *The Analytical Theory of Heat*. Cambridge Library Collection - Mathematics. Cambridge University Press, 2009.

[GB59] Sergei K. Godunov and I. Bohachevsky. Finite difference method for numerical computation of discontinuous solutions of the equations of fluid dynamics. *Matematičeskij sbornik*, 47(89)(3):271–306, 1959.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[GL21] A Gupta and P. F. J. Lermusiaux. Neural closure models for dynamical systems. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 477(2252), 08 2021.

[GU19] Constantin Greif and Karsten Urban. Decay of the kolmogorov n-width for wave problems, 2019.

[HD20] R. A. Heinonen and P. H. Diamond. Turbulence model reduction by deep learning. *Phys. Rev. E*, 101:061201, Jun 2020.

[HDK14] Cho Heyrim, Venturi Daniele, and George E. Karniadakis. Statistical analysis and simulation of random shocks in stochastic burgers equation. 2014.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[Inn18] Michael Innes. Don't unroll adjoint: Differentiating ssa-form programs. *CoRR*, abs/1810.07951, 2018.

[ISF$^+$18] Michael Innes, Elliot Saba, Keno Fischer, Dhairya Gandhi, Marco Concetto Rudilosso, Neethu Mariya Joy, Tejan Karmali, Avik Pal, and Viral Shah. Fashionable modelling with flux. *CoRR*, abs/1811.01457, 2018.

[KAT$^+$19] Byungsoo Kim, Vinicius C. Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. Deep fluids: A generative network for parameterized fluid simulations. *Computer Graphics Forum*, 38(2):59–70, may 2019.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[KGF21] Jonas Kneifl, Dennis Grunert, and Joerg Fehr. A nonintrusive nonlinear model reduction method for structural dynamical problems based on machine learning. *International Journal for Numerical Methods in Engineering*, 122(17):4774–4786, 2021.

[Kid22] Patrick Kidger. On neural differential equations, 2022.

[KSA^{+}21] Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner, and Stephan Hoyer. Machine learning accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

[L.67] Lumley J. L. The structure of inhomogeneous turbulent flows. *Atmospheric Turbulence and Radio Wave Propagation*, 1967.

[L^{+}02] Randall J LeVeque et al. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge university press, 2002.

[LC20] Kookjin Lee and Kevin T. Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *Journal of Computational Physics*, 404:108973, 2020.

[LKA^{+}20] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2020.

[MBO21] Nikolaj T. Mücke, Sander M. Bohté, and Cornelis W. Oosterlee. Reduced order modeling for parameterized time-dependent pdes using spatially and memory aware deep learning. *Journal of Computational Science*, 53:101408, 2021.

[MDI^{+}21] Yingbo Ma, Vaibhav Dixit, Michael J Innes, Xingjian Guo, and Chris Rackauckas. A comparison of automatic differentiation and continuous sensitivity analysis for derivatives of differential equation solutions. In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–9, 2021.

[MLB21] Romit Maulik, Bethany Lusch, and Prasanna Balaprakash. Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders. *Physics of Fluids*, 33(3):037106, 2021.

[MML^{+}20] Romit Maulik, Arvind Mohan, Bethany Lusch, Sandeep Madireddy, Prasanna Balaprakash, and Daniel Livescu. Time-series learning of latent-space dynamics for reduced-order model closure. *Physica D: Nonlinear Phenomena*, 405, 2020.

[OT13] San Omer and Iliescu Traian. Proper orthogonal decomposition closure models for fluid flows: burgers equation. 2013.

[PMK^{+}20] Jaideep Pathak, Mustafa Mustafa, Karthik Kashinath, Emmanuel Motheau, Thorsten Kurth, and Marcus Day. Using machine learning to augment coarse-grid computational fluid dynamics simulations. 2020.

[Pon87] Lev Semenovich Pontryagin. *Mathematical theory of optimal processes.* CRC press, 1987.

[PW16] Benjamin Peherstorfer and Karen Willcox. Data-driven operator inference for nonintrusive projection-based model reduction. *Computer Methods in Applied Mechanics and Engineering*, 306:196–215, July 2016.

[QMN15] Alfio Quarteroni, Andrea Manzoni, and Federico Negri. *Reduced basis methods for partial differential equations: an introduction*, volume 92. Springer, 2015.

[RIM$^+$19] Chris Rackauckas, Mike Innes, Yingbo Ma, Jesse Bettencourt, Lyndon White, and Vaibhav Dixit. Diffeqflux.jl-a julia library for neural differential equations. *arXiv preprint arXiv:1902.02376*, 2019.

[RMM$^+$20] Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, and Ali Ramadhan. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.

[RN17] Christopher Rackauckas and Qing Nie. Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia. *Journal of Open Research Software*, 5(1), 2017.

[RPK19] 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.

[SESO$^+$21] Ahmed Shady E., Pawar Suraj, San Omer, Rasheed Adil, Iliescu Traian, and Noack Bernd R. On closures for reduced order models – a spectrum of first-principle to machine-learned avenues. 08 2021.

[SPO16] Moritz Sieber, Christian Paschereit, and Kilian Oberleithner. Spectral proper orthogonal decomposition. *Journal of Fluid Mechanics*, 792:798–828, 03 2016.

[Tsi11] Ch. Tsitouras. Runge–kutta pairs of order 5(4) satisfying only the first column simplifying assumption. *Computers & Mathematics with Applications*, 62(2):770–775, 2011.

[Wei19] Julien Weiss. A tutorial on the proper orthogonal decomposition. pages AIAA 2019–3333. American Institute of Aeronautics and Astronautics, 2019.

[WRH19] Qian Wang, Nicolò Ripamonti, and Jan Hesthaven. Recurrent neural network closure of parametric pod-galerkin reduced-order models based on the mori-zwanzig formalism. 08 2019.

[YGBK21] Süleyman Yıldız, Pawan Goyal, Peter Benner, and Bulent Karasozen. Learning reduced-order dynamics for parametrized shallow water equations from data. *International Journal for Numerical Methods in Fluids*, 93, 05 2021.