Doctoral theses at NTNU, 2024:364

Davide Murari

Neural Networks, Differential Equations, and Structure Preservation

Doctoral thesis

NTNU
Norwegian University of Science and Technology
Thesis for the Degree of Philosophiae Doctor
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of Science and Technology

Davide Murari

Neural Networks, Differential Equations, and Structure Preservation

Thesis for the Degree of Philosophiae Doctor

Trondheim, September 2024

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

ISBN 978-82-326-8318-5 (printed ver.)
ISBN 978-82-326-8317-8 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

Doctoral theses at NTNU, 2024:364

Printed by NTNU Grafisk senter

Neural Networks, Differential Equations, and Structure Preservation

Davide Murari

September 5, 2024

Preface

This thesis is submitted in partial fulfilment of the requirements for the degree of philosophiae doctor (PhD) at the Norwegian University of Science and Technology (NTNU). This thesis is a collection of papers that I have contributed to throughout my four-year PhD at NTNU.

Acknowledgements

The Department of Mathematical Sciences has financially supported my PhD project. Throughout the PhD, I have also been funded by the Isaac Newton Institute in Cambridge, UK, a fellowship of the Simons Foundation, and the program "Pure Mathematics in Norway".

I am deeply grateful to my supervisor, Elena Celledoni, who has taught me a lot during these four years and with whom I have a great connection and I shared several fruitful research and more personal discussions. I also thank her for the complete trust she has demonstrated in me during these years, for allowing me to propose new research ideas and directions, and for introducing me to many interesting people. Brynjulf Owren, my co-supervisor, has also played a fundamental role during my PhD thanks to our numerous research meetings and his support in my academic development. For this reason, I am sincerely thankful to him.

In my four years in Trondheim, I have been lucky enough to meet many friends, especially Andrea Leone and Ergys Çokaj, with whom I have collaborated on some research projects, shared the office and the house for three years, and we also have had plenty of memorable experiences and funny moments together. I want to extend special recognition to James Jackaman. We collaborated on a research project, I assisted him while he was teaching a course, and he has assisted me in many situations. We have also attended numerous conferences together, often staying in peculiar hotels and catching inconvenient flights.

During my PhD, I have had the chance to spend four months in Cambridge,

UK. I want to thank Ferdia Sherry, who helped me in many ways while in Cambridge and with whom I collaborated on multiple research projects. The stays in the UK could not have been possible without Carola-Bibiane Schönlieb. She has always supported my research visits there, and we also had numerous interesting research discussions, especially when I attended the programme "Mathematics of Deep Learning" at the Isaac Newton Institute in 2021. During my stay in Cambridge in 2023, I also met Moshe Eliasof. I am very thankful to him for our discussions and collaboration and for sharing his experience with Machine Learning conferences.

I also want to thank my other collaborators Matthias J. Ehrhardt and Lisa Maria Kreusser (University of Bath), Marta Betcke (University College London), Martina Stavole, Sigrid Leyendecker, and Rodrigo T Sato Martín de Almagro (FAU, Erlangen). A special mention goes to Nicola Sansonetto (University of Verona), who was my supervisor during the Master's thesis and with whom we have stayed in touch during these years, discussing research, especially during his one-week stay at NTNU in 2023 and my trips back to Italy.

Lastly, but most importantly, I extend my heartfelt gratitude to my big family: my parents, il Save e la Susi, my siblings, la Ari, la Vale e il Teo, and my grandparents, la Norma, la Silvana e Adelino. They have always shown interest and appreciation in what I do and supported me during these years.

On a personal level, these four years have been a great experience, not only academically but also personally. I have had the privilege to interact with many people, be challenged by the multifaceted aspects of working in Academia, and travel to places I never thought I could ever visit. I have also been lucky to share my experiences with those who follow my YouTube channel and have consistently shown support and interest in what I do.

The PhD has taught me a lot, and I look forward to seeing what will happen afterwards.

Davide Murari
Trondheim, September 5, 2024

Acknowledging AI tools

I acknowledge that, throughout, I have utilised AI tools for the following purposes: (i) Grammarly - to correct grammatical errors and rephrase sentences for clarity and readability. I attest that I have reviewed the feedback generated by Grammarly, and based on that, I have revised the writing using my own words and expressions. (ii) ChatGPT from OpenAI - to better organise plot components and make them easier to comprehend.

Contents

Introduction

1.1 Framework

Many mathematicians, physicists, and engineers are occupied with approximating functions given some observed data or a governing rule that the unknown function has to satisfy. For example, we can accurately describe many phenomena thanks to ordinary and partial differential equations (ODEs and PDEs), such as multi-body physical systems dynamics [61, 82], fluid dynamics [1, 4], or the spread of a viral infection [20, 43]. However, analytically solving them is generally impossible, so methods for approximating their solutions are needed. Further, modern sensors are constantly collecting measurements in almost any conceivable situation. For example, sensors in a smartwatch can keep track of relevant parameters like the heart rate or the number of steps a person walks. In these situations, it is of interest to make predictions based on the collected data, e.g. to infer if a person is getting sick based on the collected measurements. Additionally, if a person has watched movies in a particular genre, streaming companies are interested in recommending other movies the user might appreciate, i.e., to develop a function approximating the user's preferences. These are all different instances of the same problem, with their respective complexities and techniques developed throughout many years of research. We can approximate the solutions of differential equations using numerical methods like the Runge-Kutta methods for ODEs [37, 38, 48, 90], and the Finite Element Method for PDEs [5, 75, 86]. Techniques like those presented in [40, 54, 84] help find approximate predictions based on time series data. Finally, streaming companies can provide good user recommendations using, for example, low-rank matrix completion algorithms [71, 91].

In recent years, there has been a rise of attention towards neural networks as methods for function approximation. Generally speaking, a neural network is a highly flexible parametric map whose parameters are chosen so that the resulting map accurately solves a task of interest. A few instances of such a task are solving a differential equation, classifying a set of images, or predicting the next frame of a video. Especially when dealing with high-dimensional spaces [3, 34, 42], or noisy observations [2, 65], neural networks often lead to improved results compared to previously developed methods.

The research in this area began with Frank Rosenblatt who developed the perceptron, a non-linear parametric function attempting to replicate the functioning of biological neurons [78]. Even though there have been some exciting developments like Hopfield Neural Networks [45, 64], this research direction seemed less promising in the 1970s and 1980s, especially after the publication of the book "Perceptrons" [68]. This book considers a quite restricted version of Rosenblatt's perceptron, and the authors prove that these paramet-

ric models can not implement relatively simple logical functions like XOR. Of high importance in the development of the field was the paper by Rumelhart, Hinton, and Williams [79] where the authors published an experimental analysis of the Backpropagation algorithm, which is still used nowadays to find the parameters of neural networks. In the 1990s, the interest in the field grew quite considerably, both in terms of experimental successes and theoretical understanding, see, e.g. [17, 44, 58, 59, 73]. Neural networks found their real traction when computing resources, like graphics cards, improved their time and memory efficiency, leading to impressive results in computer vision, generative modelling, and several other contexts [30, 32, 42, 53, 55, 87, e.g.]. Experimental results are the drivers of this success. At the same time, we still have a relatively poor theoretical understanding of why these models work, when they fail, how to interpret them, and many more questions.

The recent decade has seen an upsurge in the interest of mathematicians in studying neural networks, and the field of the Mathematics of Deep Learning is taking shape, see [33]. Several researchers are analysing and designing networks using both new and well-known techniques from various fields of mathematics, like numerical analysis, dynamical systems, functional analysis, or differential geometry. On the other hand, numerous research groups have been adopting neural networks as tools to solve problems in science, like image denoising or approximating dynamical systems with observed data. This thesis is particularly relevant to the connections between dynamical systems, numerical analysis, and neural networks. There are several links between these subjects of study, and we focus on the following three research questions:

- Can numerical methods for ODEs and PDEs help design neural networks?
- Can neural networks be adopted to approximate the equations of motion of an unknown dynamical system?
- Can neural networks, possibly coupled with other methods from numerical analysis, provide accurate solutions to differential equations?

Outline of this thesis. This thesis is divided into two main parts.

We first consider using dynamical systems and numerical methods to design neural networks with some desired level of interpretability or some in-built property of interest, which we call *structured neural networks*. Paper 1 [16] introduces our framework for designing structured neural networks. The experimental part of this paper focuses on enforcing $1-$Lipschitz regularity on a neural network. Paper 2 [26] examines how to reduce the sensitivity to input

perturbations for Graph Neural Networks. Paper 3 [14] considers the problem of approximating space-time discretisations of PDEs and presents a strategy to build norm-preserving neural networks.

The second part of this thesis studies neural networks as tools for solving scientific problems traditionally solved using classical numerical methods. Paper 4 [12] does not involve neural networks but provides the theoretical background on Lie group numerical methods that we then use in Paper 5 [15] to approximate the Hamiltonian of constrained mechanical systems. Paper 6 [10] focuses on the problem of approximating the solution operator sending the boundary conditions of Euler's elastica to the corresponding solution. Paper 7 presents a provably convergent parallel-in-time numerical solver for ODEs, where part of the algorithm is based on neural networks.

The remaining pages of this introductory chapter present the methods and techniques that we use in the subsequent chapters. This exposition does not aim to be exhaustive, but we provide a generous list of references to further the understanding. We start by introducing the basics of neural networks, focusing on several architectures of interest for this thesis. We then present some geometric numerical methods for ODEs since we use them to build neural networks with a specific structure. This background material is followed by a section dedicated to combining these tools to construct neural networks with a desired structure and another where we show how to use neural networks to approximate differential equations and their solutions in a principled manner. The introduction concludes with a summary of the seven papers in this thesis.

The introduction chapter includes a few numerical examples with associated code available at the corresponding GitHub repository[1].

1.2 Basics of neural networks

This section aims to mathematically define what is a neural network. Further, we focus on the neural network architectures, i.e., parametrisation strategies, that we utilise in the included papers.

Neural networks (NNs) are a class of machine learning methods. They can be defined as parametric maps, that we denote with $\mathcal{N}_\theta$, depending on a set of parameters θ belonging to some space Θ. The space Θ can be a linear space or a non-linear manifold. Throughout, we will always refer to $\mathcal{N}_\theta$ as a map between two linear spaces, i.e., $\mathcal{N}_\theta : \mathbb{R}^d \to \mathbb{R}^c$ for some $c, d \in \mathbb{N}$. Indeed, even when $\mathcal{N}_\theta$ acts on points belonging to a non-linear submanifold $\mathcal{M}$ of $\mathbb{R}^d$, we always

[1] https://github.com/davidemurari/examplesIntroduction

embed $\mathcal{M}$ in $\mathbb{R}^d$ and express its points in terms of their ambient space coordinates. NNs are generally expressed as the composition $\mathcal{N}_\theta = F_{\theta_L} \circ \cdots \circ F_{\theta_1}$, $\theta = (\theta_1, ..., \theta_L)$, of L parametric maps $F_{\theta_i} : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$, $i = 1, ..., L$, where $d_0 = d$ and $d_L = c$. L is the number of layers of the network, and it is usual to have the layers F_{θ_i} to be similarly parametrised. More explicitly, these layers tend to consist of linear maps after which a scalar function $\sigma : \mathbb{R} \to \mathbb{R}$, called the *activation function*, is applied to each of the input entries. Common examples of these activation functions are $\mathrm{ReLU}(x) = \max\{0, x\}$, $\mathrm{LeakyReLU}(x) = \max\{ax, x\}$ for $a \in (0, 1)$, the sigmoid function $\sigma(x) = 1/(1 + \exp(-x))$, and the hyperbolic tangent $\sigma(x) = \tanh(x)$. Motivated by the resemblance of these (artificial) neural networks to biological ones, we call *neurons* the components of the vectors obtained while processing the input vector with the network layers. Furthermore, we call *neural network architecture* the parametrisation strategy adopted to design the L layers. Choosing the right architecture for a particular problem is crucial, as it defines the search space in which the approximate solution to the problem will be found. When the number of layers L is larger than two, we refer to $\mathcal{N}_\theta$ as a *deep neural network*. *Deep Learning* is the area of machine learning focused on deep networks.

Once a model $\mathcal{N}_\theta$ is chosen, i.e., an architecture is fixed, one needs to find a good set of parameters θ that allow for $\mathcal{N}_\theta$ to solve sufficiently accurately the task of interest. The selection of the parameters is generally the result of the approximate solution of an optimisation problem where a cost function, more commonly called *loss function*, is minimised. This phase is called *neural network training*.

The loss function can combine multiple terms that might or might not depend on data. If the loss depends on data, we refer to the learning task as *supervised learning*, otherwise as *semi-supervised* or *unsupervised learning*. In the simplest case of aiming to get an approximate function $\mathcal{N}_{\theta^*}$ that fits as best as possible a set of input-output pairs $\left\{\left(\mathbf{x}_n, \mathbf{y}_n = f\left(\mathbf{x}_n\right)\right)\right\}_{n=1}^{N}$ related to an unknown function $f : \mathbb{R}^d \to \mathbb{R}^c$, one of the most common choices for the loss function is called *mean squared error* and is expressed as

$$\mathcal{L}\left(\theta\right) = \frac{1}{N} \sum_{n=1}^{N} \left\| \mathcal{N}_\theta\left(\mathbf{x}_n\right) - \mathbf{y}_n \right\|_2^2 \to \min. \tag{1.2.1}$$

The loss function in (1.2.1) or slight modifications are adopted in most of the papers in this thesis. Another loss function we use is the *cross-entropy* loss, which is commonly chosen for classification tasks as those considered in Chap-

ters 2 and 3. The cross-entropy loss consists of

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{j=1}^{c} y_{n,j}\log\left(p_{n,j}\right),\ p_{n,j} = \frac{\exp\left(\mathcal{N}_\theta(\mathbf{x}_n)_j\right)}{\sum_{k=1}^{c}\exp\left(\mathcal{N}_\theta(\mathbf{x}_n)_k\right)}, \tag{1.2.2}$$

where c is the number of classes partitioning the dataset, and $\mathbf{y}_n$ is a vector of C entries $y_{n,j}$ where only the one corresponding to the correct class of $\mathbf{x}_n$ takes value 1, while the others are all zeros. Most of the tasks considered in the papers are supervised learning tasks. One exception is made in Chapter 8, where we design neural networks aiming to solve a differential equation only given the equation itself. We comment on the loss function adopted for this problem in Section 1.5.3.

We dedicate a considerable part of this thesis to designing neural networks that have a desired structure. We elaborate on what we mean by "structure" later in the introduction. But first, let us present the architectures we work with. We start with *Fully Connected Neural Networks* (FCNNs) that are fundamental in Chapters 6, 7, and 8. The introduction to FCNNs also covers the basics of *Extreme Learning Machines* (ELMs), which are FCNNs with some weights set to random values rather than being optimised. ELMs are used in Chapter 8. In Chapters 2 and 4, we consider *Convolutional Neural Networks* (CNNs). CNNs have weights that inherently encode translation equivariance, i.e., symmetry. We then move to *Graph Neural Networks* (GNNs), considered in Chapter 3, where the techniques adopted to design CNNs for grid-structured datasets are extended to graph-structured datasets.

While the architectures introduced above differ in how the weights are restricted, neural networks can also be categorised according to how the input is utilised by the layers F_{θ_i}. There are two significant classes for this thesis: *Feedforward Neural Networks* and *Residual Neural Networks* (ResNets). There is no particular restriction on how the parameters get involved in feedforward neural networks. On the other hand, ResNets are based on layers of the form

$$F_{\theta_i}(\mathbf{x}) = \mathbf{x} + \mathcal{F}_{\theta_i}(\mathbf{x}), \tag{1.2.3}$$

where $\mathcal{F}_{\theta_i}$ is a parametric function preserving the dimension of the input $\mathbf{x}$. In other words, ResNets rely on layers where the parameters come into play only in the residual term $\mathcal{F}_{\theta_i}(\mathbf{x}) = F_{\theta_i}(\mathbf{x}) - \mathbf{x}$. Due to this restriction on the updates, one layer of a ResNet has coinciding input and output dimensions. This way of processing the input has been introduced in [42] to overcome the drawbacks of feedforward neural networks, for which deeper networks generally tend to lead to worse performance, contrary to what one would expect.

ResNets are the core of most of the included papers, given their close connection with discretisations of initial value problems. Indeed, one can see the map in (1.2.3) as one step of the explicit Euler method with step size equal to 1 applied to the non-autonomous differential equation

$$\begin{cases} \dot{\mathbf{y}}(t) = \mathcal{F}\left(\mathbf{y}(t), \theta(t)\right) \\ \mathbf{y}(t_i) = \mathbf{x} \end{cases},$$

where $\mathcal{F}_{\theta_i}(\cdot) = \mathcal{F}\left(\cdot, \theta(t_i)\right)$, and $\dot{\mathbf{y}}(t) := \frac{d}{dt}\mathbf{y}(t)$. This connection between differential equations and neural network architectures allows us to borrow from the fields of dynamical systems and numerical analysis to model neural networks behaving as desired.

1.2.1 Fully connected neural networks

Fully connected neural networks (FCNNs) are the broadest family of neural networks since they have no constraints on their weights. Indeed, their weights are generic dense matrices and vectors so that all the entries of an input vector can interact at every network layer, hence the name fully connected. For this reason, FCNNs can realise all of the other architectures we mention. The absence of weight restrictions does not necessarily make FCNNs better than more constrained architectures. Indeed, properly structuring the weights is often necessary for optimal performance. A simple parametrisation strategy is the following

$$\mathcal{F}_{\theta_i}(\mathbf{x}) = \sigma\left(\mathbf{A}_i\mathbf{x} + \mathbf{b}_i\right), \ \theta_i = \left(\mathbf{A}_i, \mathbf{b}_i\right),$$

for a suitably shaped weight matrix $\mathbf{A}_i$ and bias vector $\mathbf{b}_i$. If there is no further assumption on the structure of these parameters, one calls the resulting architecture fully connected.

In Chapter 8, we consider FCNNs of the form

$$\mathcal{N}_\theta\left(t; \mathbf{x}\right) = \mathbf{x} + \mathbf{B}\sigma\left(\mathbf{A}t + \mathbf{b}\right) - \mathbf{B}\sigma\left(\mathbf{b}\right), \ t \in [0, +\infty), \ \mathbf{x} \in \mathbb{R}^d,$$

where $\theta = \left(\mathbf{A}, \mathbf{B}, \mathbf{b}\right)$ are the weights of the network, but $\mathbf{A}$ and $\mathbf{b}$ are randomly initialised and not optimised. This design strategy, leading to so-called Extreme Learning Machines (ELMs), allows a cheaper training procedure since only the weight matrix $\mathbf{B}$ needs to be determined. We expand on this construction in Chapter 8, showing the effectiveness of these seemingly simple models in the context of finding ODE solutions.

1.2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are among the most historically successful neural network architectures. CNNs have led to incredibly accurate results in image classification and segmentation [35, 42, 55, 67, e.g.]. They originate from studies on the visual cortex of cats, for which specific visual field areas excite particular neurons [47]. This biological insight led to the consideration of highly structured weights in contrast to the dense weight matrices of FCNNs. CNNs result from composing non-linear activation functions with linear maps realised by discrete convolutional operations, whose filters can be trained. These are the only two types of functions we compose to obtain the CNNs in this thesis. On the other hand, one could include additional operations like pooling and dropout layers, possibly improving the network performance while making it harder to structure and mathematically interpret.

For a more extensive treatment of CNNs, we refer to [31, Chapter 9]. We instead introduce them by working out a 3×3 example of a discrete convolution. Let us consider a matrix $\mathbf{U} \in \mathbb{R}^{3\times 3}$ and a 3×3 convolutional filter $\mathbf{K} \in \mathbb{R}^{3\times 3}$. The matrix $\mathbf{U}$ can be seen as the evaluation of a signal $u : \mathbb{R}^2 \to \mathbb{R}$ onto a 3×3 grid. Depending on the desired output size obtained convolving $\mathbf{K}$ with $\mathbf{U}$, i.e., computing $\mathbf{K} * \mathbf{U}$, one might have to pad the input $\mathbf{U}$ to explicitly show how the signal u extends outside the grid. For this example, supposing we are interested in preserving the input size, i.e., in "same" convolutions, we add two rows and two columns. In the case of a periodic signal, we get $\mathbf{U}_P$ defined as

$$\mathbf{U}_P = \begin{bmatrix} u_{33} & u_{31} & u_{32} & u_{33} & u_{31} \\ u_{13} & u_{11} & u_{12} & u_{13} & u_{11} \\ u_{23} & u_{21} & u_{22} & u_{23} & u_{21} \\ u_{33} & u_{31} & u_{32} & u_{33} & u_{31} \\ u_{13} & u_{11} & u_{12} & u_{13} & u_{11} \end{bmatrix}.$$

In general, regardless of the size of the input $\mathbf{U}$, for 3×3 "same" convolutions one has to add two rows and two columns around the matrix $\mathbf{U}$. The convolution operation defined by a 3×3 filter is a linear map obtained by placing the kernel $\mathbf{K}$ on all the contiguous 3×3 submatrices of the padded input $\mathbf{U}_P$. When the filter is placed over a submatrix, the Frobenius inner product of the two matrices is computed, and its value will be one entry of the result. We show this procedure in the following example:

$$\mathbf{U}_P = \begin{bmatrix} u_{33} & u_{31} & u_{32} & u_{33} & u_{31} \\ u_{13} & u_{11} & u_{12} & u_{13} & u_{11} \\ u_{23} & u_{21} & u_{22} & u_{23} & u_{21} \\ u_{33} & u_{31} & u_{32} & u_{33} & u_{31} \\ u_{13} & u_{11} & u_{12} & u_{13} & u_{11} \end{bmatrix} \implies r_{11} = \operatorname{trace}\left(\begin{bmatrix} u_{33} & u_{31} & u_{32} \\ u_{13} & u_{11} & u_{12} \\ u_{23} & u_{21} & u_{22} \end{bmatrix}^\top \mathbf{K} \right)$$

where r_{11} is the first entry of the output of the convolution operation $\mathbf{R}$. The locality of this convolution operation closely relates it to the finite difference method. We exploit this connection in Chapter 4 to analyse the accuracy of CNNs in approximating PDE solutions.

Since convolutional layers are linear, they can be expressed as matrix-vector products. Indeed, using the vectorisation operator $\mathrm{vec}: \mathbb{R}^{c\times d} \to \mathbb{R}^{c\cdot d}$, one can find a matrix $\hat{\mathbf{K}}$ such that $\mathrm{vec}(\mathbf{K} * \mathbf{U}) = \hat{\mathbf{K}}\mathrm{vec}(\mathbf{U})$ for every $\mathbf{U} \in \mathbb{R}^{c\times d}$. Such a matrix can be shown to be a block Toeplitz matrix [81, Lemma 1]. The convolution operation can be extended to higher-order tensors, for example, for the red-green-blue channels of a coloured image. One can define it as in the two-dimensional case but repeating the procedure for each input channel.

1.2.3 Graph Neural Networks

The last architecture we consider in this thesis is Graph Neural Networks (GNNs). *Graph signal processing* [72, 80] is a field of mathematics focusing on analysing and processing information structured as a graph. In the last few years, graph signal processing has seen a rise in interest, and often the proposed methodologies are based on GNNs, i.e., neural networks adapted to the structure typical of graphs [6, 25, 29, 88, 92, e.g.].

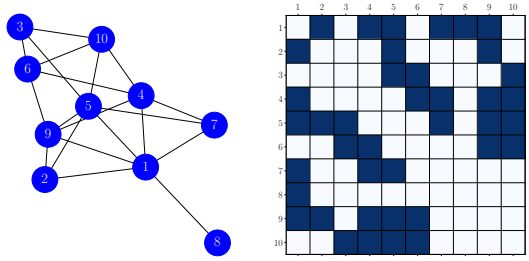


Figure 1.1: Example of a graph on the left and its adjacency matrix $\mathbf{A}$ on the right, where filled squares correspond to ones in the matrix.

A *graph* $\mathcal{G}$ is represented by a pair $(\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{\mathbf{x}_1, ..., \mathbf{x}_d\}$ is a set collecting the d *nodes* of the graph, and $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ is the collection of ordered pairs of nodes which are linked together, i.e., the collection of the *graph edges*. The graph is said to be *undirected* if for every pair $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{E}$ also $(\mathbf{x}_j, \mathbf{x}_i) \in \mathcal{E}$, as the one in Figure 1.1. It is *directed* otherwise. One can also associate a weight to each edge, leading to a *weighted graph*. A convenient way to represent the edges of a graph is with the *adjacency matrix* $\mathbf{A} \in \mathbb{R}^{d\times d}$, which is a binary matrix whose entry in row i and column j is one if $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{E}$, and zero

otherwise. The matrix $\mathbf{A}$ is symmetric for undirected graphs, see Figure 1.1.

Signals on graphs are represented by *feature vectors* associated with the graph nodes. We denote with $\mathbf{f}_i \in \mathbb{R}^c$ the feature vector associated to the node $\mathbf{x}_i$, and collect all the features in the *feature matrix* $\mathbf{F} \in \mathbb{R}^{d \times c}$ having $\mathbf{f}_i^\top$ as $i-$th row. A fundamental tool to analyse signals over a graph is the *gradient operator*, a linear operator closely resembling finite differences over a grid since its components represent the variations of a discrete signal over the graph. In Chapter 3, we use the gradient and the transpose gradient operators, defined as

$$G(\mathbf{A}) : \mathbb{R}^{d \times c} \to \mathbb{R}^{d \times d \times c}, \ \left(G(\mathbf{A})\,\mathbf{F}\right)_{ijk} = \mathbf{A}_{ij}\left(\mathbf{F}_{ik} - \mathbf{F}_{jk}\right),$$

$$G(\mathbf{A})^\top : \mathbb{R}^{d \times d \times c} \to \mathbb{R}^{d \times c}, \ \left(G(\mathbf{A})^\top \mathbf{O}\right)_{ik} = \sum_{j=1}^{d}\left(\mathbf{A}_{ij}\mathbf{O}_{ijk} - \mathbf{A}_{ji}\mathbf{O}_{jik}\right).$$

A GNN is a parametric map $\mathcal{N}_\theta : \mathbb{R}^{d \times c} \times \mathbb{R}^{d \times d} \to \mathbb{R}^o$, where $\mathcal{N}_\theta\left(\mathbf{F}, \mathbf{A}\right)$ is the network prediction, whose output dimension o depends on the task of interest. For example, in Chapter 3, we consider the task of node classification in graphs. In this case, o coincides with the number of classes we want to use to partition the nodes. This task consists of working with a single graph $\mathcal{G}$, for which the labels of part of its nodes are available, and then trying to provide an accurate prediction for the class of the remaining nodes. To make such a prediction, we train a neural network so it minimises the mismatch between the predicted and the true labels. We now provide a high-level introduction to the building blocks of GNNs, and more details on the specific solution we propose can be found in Chapter 3.

Several of the GNN layers locally aggregate the node features based on the graph connectivity. The simplest way to do so is via the adjacency matrix $\mathbf{A}$ with a linear map of the form $\left(\mathbf{F}, \mathbf{A}\right) \mapsto \mathbf{AFW}$, where $\mathbf{W} \in \mathbb{R}^{c \times c}$ is a trainable weight which linearly combines the columns of $\mathbf{F}$. A slightly more general variation could be $\left(\mathbf{F}, \mathbf{A}\right) \mapsto \mathbf{A}\sigma(\mathbf{FW})$ or alternatives of this construction suitably aggregating the features by using linear and non-linear maps.

A key property of GNNs is that they respect the dataset's structure. For example, when applied to graph-node classification tasks, they are generally designed to be permutation equivariant. This means that if the nodes are numbered differently, i.e., permuted in order, the predicted labels change accordingly. This is coherent with the fact that the class of a node is independent of how the graph nodes are ordered. Mathematically, this means that $\mathcal{N}_\theta\left(\mathbf{PF}, \mathbf{PAP}^\top\right) = \mathbf{P}\mathcal{N}_\theta\left(\mathbf{F}, \mathbf{A}\right)$ for every permutation matrix $\mathbf{P} \in \mathbb{R}^{d \times d}$.

The way the features are processed differs depending on the architecture one chooses. However, a standard guideline in designing these models is that the

first few layers of the network architecture aim to extract features and information from the graph. Then, the final layers aim to exploit this extracted knowledge about the graph for the downstream task of interest, such as the classification of the graph nodes. This latter part often comprises fully connected layers applied to the processed feature matrix.

1.3 Fundamentals of geometric numerical integration

This thesis builds on the connections between neural networks and dynamical systems, i.e. time-dependent Ordinary and Partial Differential Equations (ODEs and PDEs). In this section, we introduce the tools and techniques from the field of geometric numerical analysis that are used in this thesis. Geometric numerical methods aim to approximate the solutions of differential equations while preserving some of their known qualitative properties. In contrast, general-purpose numerical methods aim to obtain accurate approximations of the desired solutions without exploiting or preserving the known structure of the differential system. We focus on methods for ODEs because they are sufficient to understand the papers in this thesis. We refer the reader to [13, 27, 62, e.g.] as high-quality resources on geometric numerical methods for PDEs. We point out that we present only the techniques we use in the included papers. For a more extensive treatment, see [37].

Throughout the introduction, when we refer to the solution at a time t of

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathcal{F}\left(\mathbf{x}(t)\right) \in \mathbb{R}^d \\ \mathbf{x}(0) = \mathbf{x}_0 \end{cases},$$

we use the notation $\Phi_{\mathcal{F}}^{t}\left(\mathbf{x}_0\right)$. We denote with $\Psi_{\mathcal{F}}^{\Delta t}$ a numerical method of step size $\Delta t > 0$ approximating $\Phi_{\mathcal{F}}^{\Delta t}$.

1.3.1 Splitting methods

For our derivations, we use one numerical technique above the others: splitting methods. These methods are of interest when there is a natural or convenient way to split a vector field additively. We present the main idea assuming that the vector field is written as the sum of two terms, but the reasoning extends to the sum of more terms. For a more comprehensive treatment, see [66].

Let us consider a vector field $\mathcal{F}: \mathbb{R}^d \to \mathbb{R}^d$, with $\mathcal{F}(\mathbf{x}) = \mathcal{F}_1(\mathbf{x}) + \mathcal{F}_2(\mathbf{x})$, where $\dot{\mathbf{x}}(t) = \mathcal{F}_1\left(\mathbf{x}(t)\right)$ and $\dot{\mathbf{x}}(t) = \mathcal{F}_2\left(\mathbf{x}(t)\right)$ are differential equations that admit a

known analytical solution or whose solutions can be efficiently approximated. In general, the flows of $\mathcal{F}_1$ and $\mathcal{F}_2$ do not commute [60, Theorem 9.44], i.e.,

$$\Phi^t_{\mathcal{F}} \neq \Phi^t_{\mathcal{F}_1} \circ \Phi^t_{\mathcal{F}_2}, \quad \Phi^t_{\mathcal{F}} \neq \Phi^t_{\mathcal{F}_2} \circ \Phi^t_{\mathcal{F}_1}, \quad \Phi^t_{\mathcal{F}_1} \circ \Phi^t_{\mathcal{F}_2} \neq \Phi^t_{\mathcal{F}_2} \circ \Phi^t_{\mathcal{F}_1}.$$

However, using the Baker-Campbell-Hausdorff (BCH) formula [39, Chapter 5], it is possible to see that, for small enough t,

$$\Phi^t_{\mathcal{F}} = \Phi^t_{\mathcal{F}_1} \circ \Phi^t_{\mathcal{F}_2} + \mathcal{O}(t^2), \qquad \Phi^t_{\mathcal{F}} = \Phi^t_{\mathcal{F}_2} \circ \Phi^t_{\mathcal{F}_1} + \mathcal{O}(t^2) \tag{1.3.1}$$

$$\Phi^t_{\mathcal{F}} = \Phi^{t/2}_{\mathcal{F}_1} \circ \Phi^t_{\mathcal{F}_2} \circ \Phi^{t/2}_{\mathcal{F}_1} + \mathcal{O}(t^3), \quad \Phi^t_{\mathcal{F}} = \Phi^{t/2}_{\mathcal{F}_2} \circ \Phi^t_{\mathcal{F}_1} \circ \Phi^{t/2}_{\mathcal{F}_2} + \mathcal{O}(t^3), \tag{1.3.2}$$

leading to the Lie-Trotter splitting method, (1.3.1), and the Strang splitting method, (1.3.2). If the analytical solutions $\Phi^t_{\mathcal{F}_i}$ are unknown, one could suitably replace the exact flows with their numerical approximation and get numerical solvers again with order 1 and 2.

We adopt these methods to approximate the Hamiltonian of unconstrained systems in Chapter 6. In Chapter 2, we employ splitting methods to design nonexpansive systems and to derive some expressivity results for structured networks.

1.3.2 Volume-preserving methods

Another instance in which splitting methods have been used in this thesis is to obtain volume-preserving neural networks. Operator splitting is the strategy behind most volume-preserving integrators since a $d-$dimensional volume-preserving vector field can be split as the sum of $d-1$ systems, each being Hamiltonian in two of the d variables [52].

A diffeomorphism $\varphi : \Omega \to \mathbb{R}^d$, $\Omega \subset \mathbb{R}^d$ open, preserves the canonical volume form $\mu_{\mathbf{x}} = \mathrm{d}x_1 ... \mathrm{d}x_d$ of $\mathbb{R}^d$ if

$$\mathrm{vol}\Big(\varphi\big(\Omega\big)\Big) = \int_{\varphi(\Omega)} \mathrm{d}x_1 ... \mathrm{d}x_d = \int_{\Omega} \Big|\det\big(\partial_{\mathbf{x}}\varphi\,(\mathbf{x})\big)\Big| \,\mathrm{d}x_1 ... \mathrm{d}x_d$$

is equal to $\mathrm{vol}\big(\Omega\big)$. As a consequence, φ preserves μ if $\Big|\det\big(\partial_{\mathbf{x}}\varphi\,(\mathbf{x})\big)\Big| = 1$ for every $\mathbf{x} \in \Omega$. This characterisation immediately implies that the composition of volume-preserving maps is also volume-preserving.

Given that we do not explore this property extensively, we limit the treatment to the details needed to follow the reasoning presented in Chapter 2 to build volume-preserving neural networks. It is indeed well known, see [37, Lemma VI.9.1], that a vector field $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^d$ preserves the canonical volume form μ

if and only if the divergence of $\mathcal{F}$ associated to the volume form μ is zero. A particular class of vector fields with such a property are those partitioned like

$$\mathcal{F}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}_2) \\ f_2(\mathbf{x}_1) \end{bmatrix} = \begin{bmatrix} f_1(\mathbf{x}_2) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ f_2(\mathbf{x}_1) \end{bmatrix} =: \mathcal{F}_1(\mathbf{x}) + \mathcal{F}_2(\mathbf{x}), \quad \mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix},$$

with $\mathbf{x}_1 \in \mathbb{R}^{d_1}$, $\mathbf{x}_2 \in \mathbb{R}^{d_2}$, $f_1 : \mathbb{R}^{d_2} \to \mathbb{R}^{d_1}$, $f_2 : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$, and $d = d_1 + d_2$. This splitting is convenient since the two equations we obtain admit the analytical solutions

$$\Phi^t_{\mathcal{F}_1}(\mathbf{x}) = \begin{bmatrix} \mathbf{x}_1 + t f_1(\mathbf{x}_2) \\ \mathbf{x}_2 \end{bmatrix}, \quad \Phi^t_{\mathcal{F}_2}(\mathbf{x}) = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 + t f_2(\mathbf{x}_1) \end{bmatrix}.$$

Being $\Phi^t_{\mathcal{F}_1}$ and $\Phi^t_{\mathcal{F}_2}$ the exact solutions of divergence-free vector fields, they are both volume-preserving maps. Thus, composing these two exact flow maps, for example as in (1.3.2), is a strategy giving a consistent volume-preserving numerical approximation of the exact solution $\Phi^t_{\mathcal{F}}$.

1.3.3 Symplectic methods

Especially in physics, many systems can be described through the Lagrangian and Hamiltonian formalisms. We briefly present these two formalisms in the case of systems defined on a linear space.

The Lagrangian description of a conservative system is based on a Lagrangian function $L : T\mathbb{R}^d \to \mathbb{R}$, where $T\mathbb{R}^d \simeq \mathbb{R}^{2d}$ is the tangent bundle of $\mathbb{R}^d$. A generic point of $T\mathbb{R}^d$ is represented by its generalised coordinates $(\mathbf{q}, \dot{\mathbf{q}})$. The equations of motion of Lagrangian systems are defined by the second-order system

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\mathbf{q}}}(\mathbf{q}, \dot{\mathbf{q}})\right) - \frac{\partial L}{\partial \mathbf{q}}(\mathbf{q}, \dot{\mathbf{q}}) = 0.$$

One can then introduce the Legendre transform

$$(\mathbf{q}, \mathbf{p}) := \mathbb{F}L(\mathbf{q}, \dot{\mathbf{q}}) = \left(\mathbf{q}, \frac{\partial L}{\partial \dot{\mathbf{q}}}(\mathbf{q}, \dot{\mathbf{q}})\right),$$

which we now suppose is a diffeomorphism, and provide a change of variables between the tangent bundle $T\mathbb{R}^d$ and the cotangent bundle $T^*\mathbb{R}^d$, where the pair $(\mathbf{q}, \mathbf{p})$ belongs. We remark that $T^*\mathbb{R}^d$ can be identified with $\mathbb{R}^{2d}$ as well. This change of variables allows one to define the conjugate momentum $\mathbf{p}$ and introduce the Hamiltonian formalism based on the Hamiltonian energy

$$H(\mathbf{q}, \mathbf{p}) = \left(\mathbf{p} \cdot \dot{\mathbf{q}} - L(\mathbf{q}, \dot{\mathbf{q}})\right)\Big|_{(\mathbf{q}, \dot{\mathbf{q}}) = (\mathbb{F}L)^{-1}(\mathbf{q}, \mathbf{p})}.$$

Several systems admit a Lagrangian function that takes the form

$$L(\mathbf{q},\dot{\mathbf{q}}) = \frac{1}{2}\dot{\mathbf{q}}^\top M(\mathbf{q})\dot{\mathbf{q}} - U(\mathbf{q}),$$

where $M(\mathbf{q})$ defines a metric on $\mathbb{R}^d$. For these systems, the Hamiltonian reads

$$H(\mathbf{q},\mathbf{p}) = \frac{1}{2}\mathbf{p}^\top M^{-1}(\mathbf{q})\mathbf{p} + U(\mathbf{q}).$$

The Hamiltonian H is separable if M is independent of $\mathbf{q}$. More generally, a Hamiltonian function $H : T^*\mathbb{R}^d \to \mathbb{R}$ is called separable if it is the sum of two terms depending only on one of the two variables, i.e., either on $\mathbf{q}$ or on $\mathbf{p}$. The Hamiltonian equations of motion with respect to the canonical symplectic structure of $\mathbb{R}^{2d}$ are

$$\begin{bmatrix}\dot{\mathbf{q}}\\ \dot{\mathbf{p}}\end{bmatrix} = \mathbb{J}\nabla H(\mathbf{q},\mathbf{p}) =: X_H(\mathbf{q},\mathbf{p}), \tag{1.3.3}$$

where

$$\mathbb{J} = \begin{bmatrix} 0_d & \mathbf{I}_d \\ -\mathbf{I}_d & 0_d \end{bmatrix} \in \mathbb{R}^{2d\times 2d}$$

is the canonical symplectic matrix of $\mathbb{R}^{2d}$, and $\mathbf{I}_d, 0_d \in \mathbb{R}^{d\times d}$ are the identity and the zero matrices respectively. The solutions of (1.3.3) have several physical and geometric properties. The first one is that they preserve the energy function H, indeed

$$\frac{d}{dt}H(\mathbf{q}(t),\mathbf{p}(t)) = \nabla H(\mathbf{q}(t),\mathbf{p}(t))^\top \mathbb{J}\nabla H(\mathbf{q}(t),\mathbf{p}(t)) = 0.$$

Apart from this physical property, the flow map $\Phi^t_{X_H}$ preserves the symplectic bilinear form $\Omega(\mathbf{v},\mathbf{w}) = \mathbf{v}^\top\mathbb{J}\mathbf{w}$ defined over $T^*\mathbb{R}^d$. Indeed, one has

$$\Omega\left(\frac{\partial\Phi^t_{X_H}(\mathbf{x})}{\partial\mathbf{x}}\mathbf{v}, \frac{\partial\Phi^t_{X_H}(\mathbf{x})}{\partial\mathbf{x}}\mathbf{w}\right) = \Omega(\mathbf{v},\mathbf{w}), \quad \forall\, t\geq 0,\ \forall \mathbf{x},\mathbf{v},\mathbf{w}\in\mathbb{R}^{2d}.$$

We say that a numerical method $\Psi^{\Delta t}_{X_H}$ approximating $\Phi^{\Delta t}_{X_H}$ is symplectic if it has the same property, i.e., if

$$\Omega\left(\frac{\partial\Psi^{\Delta t}_{X_H}(\mathbf{x})}{\partial\mathbf{x}}\mathbf{v}, \frac{\partial\Psi^{\Delta t}_{X_H}(\mathbf{x})}{\partial\mathbf{x}}\mathbf{w}\right) = \Omega(\mathbf{v},\mathbf{w}), \quad \forall \mathbf{x},\mathbf{v},\mathbf{w}\in\mathbb{R}^{2d}.$$

This thesis does not consider numerical methods reproducing at a discrete level the conservation of the energy function H, for which we refer to [7, 24, 49, 76].

On the other hand, we work with methods reproducing the conservation of the bilinear form Ω, i.e., symplectic methods. There are several families of symplectic methods. We focus on splitting methods for separable Hamiltonian systems. For a more general presentation, see, e.g. [37, Chapter VI].

Let us restrict our attention to the separable Hamiltonian $H(\mathbf{q},\mathbf{p}) = K(\mathbf{p})+U(\mathbf{q})$. We may consider the Hamiltonian systems generated by K and U separately and then suitably combine their flows using a splitting method like Lie-Trotter or Strang-Splitting. Indeed, these two systems have the fundamental property of being analytically solvable. Depending on the chosen splitting strategy, one gets a different symplectic method since the composition of symplectic maps is symplectic. Two methods we work with are the symplectic Euler method

$$\Psi_{X_H}^{\Delta t}\left(\mathbf{x}_0\right) = \Phi_{X_U}^{\Delta t} \circ \Phi_{X_K}^{\Delta t}\left(\mathbf{x}_0\right), \quad \mathbf{x}_0 \in T^*\mathbb{R}^d,$$

and the Störmer-Verlet method, which takes the form

$$\Psi_{X_H}^{\Delta t}\left(\mathbf{x}_0\right) = \Phi_{X_U}^{\Delta t/2} \circ \Phi_{X_K}^{\Delta t} \circ \Phi_{X_U}^{\Delta t/2}\left(\mathbf{x}_0\right), \quad \mathbf{x}_0 \in T^*\mathbb{R}^d.$$

Both methods could be obtained by exchanging the roles of K and U, composing the flows differently, see [37, Theorems VI.3.3 and VI.3.4]. In Chapter 6, we investigate these methods as tools for approximating the Hamiltonian of unknown systems provided with observational data.

1.3.4 Projection methods

Many of the geometric properties that are worthwhile preserving when simulating differential equations can be expressed as the requirement that the solution belongs to a suitable submanifold $\mathcal{M}$ of $\mathbb{R}^d$. Indeed, if for example it is known that there exists a function $I : \mathbb{R}^d \to \mathbb{R}$ such that $\nabla I(\mathbf{x})^\top \mathcal{F}(\mathbf{x}) \equiv 0$, i.e., $I\left(\Phi_{\mathcal{F}}^t\left(\mathbf{x}_0\right)\right) = I\left(\mathbf{x}_0\right)$ for every $t \geq 0$, then a method preserving I satisfies

$$I\left(\Psi_{\mathcal{F}}^{\Delta t}\left(\mathbf{x}_0\right)\right) = I\left(\mathbf{x}_0\right)$$

for every $\mathbf{x}_0 \in \mathbb{R}^d$. A reformulation of such condition can be obtained by introducing a function $g : \mathbb{R}^d \to \mathbb{R}$ defined as $g(\mathbf{x}) = I(\mathbf{x}) - I\left(\mathbf{x}_0\right)$ and requiring

$$\Psi_{\mathcal{F}}^{\Delta t}\left(\mathbf{x}_0\right) \in \mathcal{M} := \left\{\mathbf{x} \in \mathbb{R}^d : \ g(\mathbf{x}) = 0\right\} \subset \mathbb{R}^d. \tag{1.3.4}$$

This approach is relatively general, extending to other invariants and, for example, to settings in which the solution of the differential equation $\dot{\mathbf{x}}(t) = \mathcal{F}\left(\mathbf{x}(t)\right)$ should satisfy a set of constraints. Projection methods are a class of numerical methods for which (1.3.4) holds. The main idea behind these methods is to

modify a numerical integrator that does not preserve $\mathcal{M}$ by projecting the approximation back to $\mathcal{M}$. There are several ways to formalise such a statement. We briefly present the one based on Lagrange multipliers.

Let us denote with $\varphi_{\mathcal{F}}^{\Delta t} : \mathbb{R}^d \to \mathbb{R}^d$ a generic one-step method applied to the ordinary differential equation $\dot{\mathbf{x}}(t) = \mathcal{F}\big(\mathbf{x}(t)\big)$. Suppose the solution $t \mapsto \mathbf{x}(t)$ is known to belong to a $c-$dimensional submanifold $\mathcal{M} \subset \mathbb{R}^d$ that can be expressed as the set of zeros of a constraint function $g : \mathbb{R}^d \to \mathbb{R}^{d-c}$ having functionally independent components. Then, a projection method $\Psi_{\mathcal{F}}^{\Delta t}$ based on $\varphi_{\mathcal{F}}^{\Delta t}$ and applied to a point $\mathbf{x}_n \in \mathcal{M}$ can be written as

$$\begin{aligned} \hat{\mathbf{x}}_{n+1} &= \varphi_{\mathcal{F}}^{\Delta t}\left(\mathbf{x}_n\right), \\ \mathbf{x}_{n+1} &= \hat{\mathbf{x}}_{n+1} + \Big(\partial_{\mathbf{x}} g\left(\hat{\mathbf{x}}_{n+1}\right)\Big)^{\top} \boldsymbol{\lambda}_n =: \Psi_{\mathcal{F}}^{\Delta t}\left(\mathbf{x}_n\right), \end{aligned} \tag{1.3.5}$$

where $\boldsymbol{\lambda}_n \in \mathbb{R}^{d-c}$ is chosen so that $g(\mathbf{x}_{n+1}) = 0$, i.e., $\Psi_{\mathcal{F}}^{\Delta t}(\mathbf{x}_n) \in \mathcal{M}$. In (1.3.5) the matrix $\partial_{\mathbf{x}} g(\hat{\mathbf{x}}_{n+1}) \in \mathbb{R}^{d-c\times d}$ is the Jacobian matrix of g evaluated at $\hat{\mathbf{x}}_{n+1}$. In Chapter 4, we consider a differential equation whose solutions preserve $I(\mathbf{x}) = \|\mathbf{x}\|_2^2$. In this case, we set $g(\mathbf{x}) = \|\mathbf{x}\|_2^2 - \|\mathbf{x}_0\|_2^2$, where $\mathbf{x}_0 = \mathbf{x}(0)$.

There are variants of the projection methods we have just presented, like symmetric projection methods [36], that we omit in this discussion since they are not of use in the papers included in this thesis.

1.3.5 Lie-group integrators

Lie groups are fundamental for the analysis and modelling of mechanical systems. In this section, we briefly introduce Lie groups, based on [39, 60], and then present the basics of Lie group methods.

Definition 1.1 (Lie group)**.** A Lie group is a pair $(\mathcal{G}, \cdot)$ with $\mathcal{G}$ a smooth manifold, $\cdot : \mathcal{G} \times \mathcal{G} \to \mathcal{G}$ a smooth map endowing $\mathcal{G}$ with a group structure, and the inverse map $\mathcal{G} \ni g \mapsto g^{-1} \in \mathcal{G}$ is well defined and smooth.

As with any algebraic group, Lie groups admit an identity element that we denote with e. An example of a Lie group is the special orthogonal group $SO(3)$ of 3×3 real orthogonal matrices equipped with the usual matrix-matrix product.

Definition 1.2 (Lie algebra)**.** A Lie algebra $\mathfrak{g}$ is a vector space over a field $\mathbb{K}$ endowed with an antisymmetric bilinear map $[\cdot,\cdot] : \mathfrak{g} \times \mathfrak{g} \to \mathfrak{g}$, called a Lie bracket, satisfying the Jacobi identity

$$\Big[\xi, [\nu, \eta]\Big] + \Big[\nu, [\eta, \xi]\Big] + \Big[\eta, [\xi, \nu]\Big] = 0, \ \forall \xi, \nu, \eta \in \mathfrak{g}.$$

An example of Lie algebra is the Lie algebra of smooth vector fields over a smooth manifold $\mathcal{M}$, usually denoted as $\mathfrak{X}(\mathcal{M})$. A Lie bracket for this Lie algebra is the Jacobi-Lie bracket, which is defined as

$$[X,Y](f) = X\Big(Y(f)\Big) - Y\Big(X(f)\Big), \text{ for every } f \in \mathcal{C}^\infty(\mathcal{M},\mathbb{R}),$$

where $X(f) = \mathcal{L}_X(f) : \mathcal{M} \to \mathbb{R}$ is the Lie derivative of f along X, defined as

$$\mathcal{L}_X(f)(x) = \left.\frac{d}{dt}\right|_{t=0} \left(f \circ \Phi_X^t\right)(x).$$

Each Lie group $\mathcal{G}$ has an associated Lie algebra $\mathfrak{g}$. To characterise $\mathfrak{g}$, we introduce the left translation by g, i.e., the map $L_g : \mathcal{G} \to \mathcal{G}$ with $L_g(h) = g \cdot h$.

The tangent space to a manifold can be characterised in several ways. For the purposes of this section, we opt for the following one:

Definition 1.3 (Tangent space at $h \in \mathcal{G}$)**.** The linear map $v_h : \mathcal{C}^\infty(\mathcal{G},\mathbb{R}) \to \mathbb{R}$ is a derivation at $h \in \mathcal{G}$ if for every $f, g \in \mathcal{C}^\infty(\mathcal{G},\mathbb{R})$, $v_h(fg) = f(h)\, v_h(g) + g(h)\, v_h(f)$. The set of all derivations of $\mathcal{C}^\infty(\mathcal{G},\mathbb{R})$ at h, denoted as $T_h\mathcal{G}$, is the tangent space at h to $\mathcal{G}$. Its elements are called tangent vectors at h to $\mathcal{G}$.

Definition 1.4 (Differential at $h \in \mathcal{G}$)**.** Given a smooth function $F : \mathcal{G} \to \mathcal{G}$, the differential at $h \in \mathcal{G}$ of F is the map $dF|_h : T_h\mathcal{G} \to T_{F(h)}\mathcal{G}$ defined as $dF|_h(v_h)(f) = v_h(f \circ F)$ for an arbitrary $v_h \in T_h\mathcal{G}$ and $f \in \mathcal{C}^\infty(\mathcal{G},\mathbb{R})$.

A vector field $X \in \mathfrak{X}(\mathcal{G})$ is said to be left-invariant if for every pair of points $g, h \in \mathcal{G}$, it holds that $dL_g|_h\big(X(h)\big) = X(L_g(h)) = X(g \cdot h)$, where we recall that $X(h) \in T_h\mathcal{G}$ is the evaluation of the vector field X at $h \in \mathcal{G}$. We denote as

$$\mathfrak{g} = \Big\{X \in \mathfrak{X}(\mathcal{G}) : \ X \text{ is left-invariant}\Big\}$$

the vector space of the smooth left-invariant vector fields of $\mathcal{G}$. Furthermore, for every $X \in \mathfrak{g}$, $g, h \in \mathcal{G}$, and $f : \mathcal{G} \to \mathbb{R}$ smooth, we have

$$\begin{aligned} X\Big(f \circ L_g\Big)(h) &= X(h)\Big(f \circ L_g\Big) = dL_g|_h X(h)(f) \\ &= X\Big(L_g(h)\Big)(f) = X(f)\Big(L_g(h)\Big) = X(f) \circ L_g(h), \end{aligned}$$

see [60, Proposition 8.16]. This means that the condition $X\Big(f \circ L_g\Big) = X(f) \circ L_g$ for every $g \in \mathcal{G}$ and $f \in \mathcal{C}^\infty(\mathcal{G},\mathbb{R})$ is an equivalent way to formulate the left-invariance of X. To show $\mathfrak{g}$ is a Lie algebra with respect to the Jacobi-Lie

bracket of $\mathfrak{X}(\mathcal{G})$, it is thus sufficient to show that for an arbitrary pair $X, Y \in \mathfrak{g}$, any $g \in \mathcal{G}$, and any $f \in \mathcal{C}^\infty(\mathcal{G}, \mathbb{R})$, one has

$$\begin{aligned}[X,Y]\left(f \circ L_g\right) &= XY\left(f \circ L_g\right) - YX\left(f \circ L_g\right) = X\left(Y(f) \circ L_g\right) - Y\left(X(f) \circ L_g\right) \\ &= \left(XY(f) - YX(f)\right) \circ L_g = \left([X,Y](f)\right) \circ L_g,\end{aligned}$$

and hence $[X,Y] \in \mathfrak{g}$. For an $X \in \mathfrak{g}$, the values it takes all around $\mathcal{G}$ are determined by the values at the identity element e, since $X(g) = X(g \cdot e) = dL_g|_e\big(X(e)\big)$, where $X(e) \in T_e\mathcal{G}$. This consideration implies that the space of left-invariant vector fields can be identified with $T_e\mathcal{G}$. $\mathfrak{g} \simeq T_e\mathcal{G}$ is the Lie algebra of the Lie group $\mathcal{G}$. An alternative way to see this construction is that for any $\xi \in T_e\mathcal{G}$, there is a unique left-invariant vector field $X \in \mathfrak{g}$ defined as $X(g) = dL_g|_e(\xi)$ for any $g \in \mathcal{G}$, see [60, Theorem 8.37].

It is possible to map elements of the Lie algebra $\mathfrak{g}$ to elements of $\mathcal{G}$ thanks to the exponential map $\exp : \mathfrak{g} \to \mathcal{G}$ defined as $\exp(X) = \Phi_X^1(e)$, where X is a left-invariant vector field. For the case of matrix Lie groups, the Lie group exponential map coincides with the matrix exponential $\exp(\mathbf{A}) = \sum_{j=0}^{+\infty} \mathbf{A}^j / j!$, see [60, Proposition 20.2].

Of particular importance to two of the papers included in this thesis, those in Chapters 5 and 6, is the notion of Lie group action.

Definition 1.5 (Lie group action)**.** A (left) Lie group action of $\mathcal{G}$ onto the smooth manifold $\mathcal{M}$ is a smooth map $\varphi : \mathcal{G} \times \mathcal{M} \to \mathcal{M}$ that satisfies the following properties:

- $\varphi(e, m) = m$ for every $m \in \mathcal{M}$,
- $\varphi\big(g_1, \varphi(g_2, m)\big) = \varphi\big(g_1 \cdot g_2, m\big)$ for every $g_1, g_2 \in \mathcal{G}$, and $m \in \mathcal{M}$.

The simplest example of Lie group action is the action by left-translation of $\mathcal{G}$ onto itself, i.e., $\varphi(g, h) = L_g h$. Introducing the right-translation $R_g(h) = h \cdot g$, one can also define the action by conjugation of $\mathcal{G}$ onto itself as $\varphi(g, h) = L_g \circ R_{g^{-1}}(h) = g \cdot h \cdot g^{-1}$. We will introduce in Chapter 5 a particular Lie group action of the special Euclidean group $SE(3) = SO(3) \ltimes \mathbb{R}^3$ onto $\mathbb{R}^6$, where $\ltimes$ is the semidirect product.

The orbit of a point $m \in \mathcal{M}$ for the Lie group action $\varphi : \mathcal{G} \times \mathcal{M} \to \mathcal{M}$ is defined as $\mathcal{O}(m) := \{\varphi(g, m) : g \in \mathcal{G}\} \subset \mathcal{M}$, i.e., the immersed submanifold [60, Chapter 5] of points in $\mathcal{M}$ that can be reached via the group action applied to m. We say a Lie group action is transitive if for every $m \in \mathcal{M}$, $\mathcal{O}(m) = \mathcal{M}$, i.e., for any pair of points $m_1, m_2 \in \mathcal{M}$, there is a $g \in \mathcal{G}$ such that $\varphi(g, m_1) = m_2$. An example of transitive Lie group action is the action by left translations of

$\mathcal{G}$ onto itself. We say a manifold $\mathcal{M}$ to be homogeneous if a Lie group $\mathcal{G}$ acts transitively on it. An example of a homogeneous manifold which is not a Lie group is the unitary sphere $\mathcal{S}^2 \subset \mathbb{R}^3$, which is acted transitively by $SO(3)$ as $\varphi : SO(3) \times \mathcal{S}^2 \to \mathcal{S}^2$, $(\mathbf{R}, \mathbf{q}) \to \mathbf{R}\mathbf{q}$, with $\mathbf{q} \in \mathbb{R}^3$ having $\mathbf{q}^\top \mathbf{q} = 1$.

Approximating the solutions of the ordinary differential equation defined by $X \in \mathfrak{X}(\mathcal{M})$ while maintaining the solution curves on the manifold $\mathcal{M}$ is a challenging task. When $\mathcal{M}$ is a homogeneous manifold acted by the Lie group $\mathcal{G}$ via the transitive Lie group action φ, one option is to rely on Lie group integrators. We now introduce the main idea behind a class of Lie group methods, i.e., Runge-Kutta-Munthe-Kaas (RKMK) methods. For a more detailed analysis, derivation, and use in data-driven modelling, we defer to Chapters 5 and 6. Let us consider the initial value problem

$$\begin{cases} \dot{x}(t) = X\big(x(t)\big) \in T_{x(t)}\mathcal{M} \\ x(0) = x_0 \in \mathcal{M} \end{cases} . \tag{1.3.6}$$

The solution $x(t) = \Phi_X^t(x_0)$, $t \geq 0$, lies on the manifold $\mathcal{M}$. Our goal is to design a numerical method $\Psi_X^{\Delta t} : \mathcal{M} \to \mathcal{M}$ maintaining the approximate solution on $\mathcal{M}$. Since $\mathcal{M}$ is a homogeneous manifold, there exists a group element $g \in \mathcal{G}$ such that $\varphi(g, x_0) = \Phi_X^{\Delta t}(x_0)$. The RKMK methods aim to approximate the group element g by leveraging the linear nature of the Lie algebra $\mathfrak{g}$ of the Lie group $\mathcal{G}$. The first step involves expressing the solution as $x(t) = \varphi\Big(\exp\big(\xi(t)\big), x_0\Big)$ for small enough times $t \geq 0$. Here, the curve $t \mapsto \xi(t) \in \mathfrak{g}$ solves a suitable differential equation obtained by lifting the one on $\mathcal{M}$ to $\mathfrak{g}$. Using a Runge-Kutta method, we approximate $\xi(\Delta t)$ to obtain $\xi_1 \in \mathfrak{g}$. This approximation is then used to update x_0 as $x_1 = \varphi\Big(\exp\big(\xi_1\big), x_0\Big) =: \Psi_X^{\Delta t}(x_0) \in \mathcal{M}$. Figure 1.2 shows an example where the Lie Euler method, i.e., the RKMK method based on using the explicit Euler method to find ξ_1, is applied and compared to the explicit Euler method. The ODE we consider describes the motion of a free rigid body. There are also other classes of Lie group methods, like commutator-free ones, and we refer the reader to Chapter 5 and references therein for more details.

1.3.6 Non-expansive methods

The stability of one-step schemes is one of the fundamental properties that makes them reliable. There are several notions of stability, one of which is the so-called B-stability. B-stability is also called non-linear stability, being it defined for non-linear vector fields differently from A-stability [18, 22].

For a vector field $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^d$, one can define the one-sided Lipschitz constant

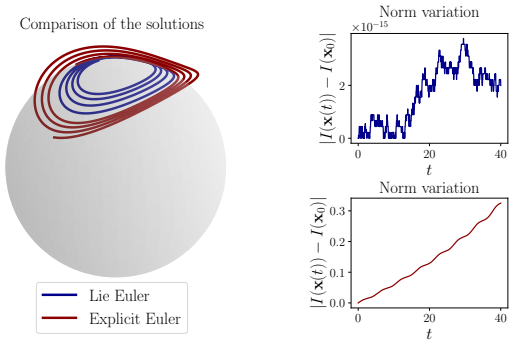


Figure 1.2: Simulation of a free rigid body preserving the norm of the angular momentum $I(\mathbf{x}) = \|\mathbf{x}\|_2^2$ with the Lie Euler method, and obtaining qualitatively inaccurate results with explicit Euler.

$\operatorname{osLip}(\mathcal{F})$ associated to the Euclidean norm of $\mathbb{R}^d$ as

$$\operatorname{osLip}(\mathcal{F}) := \sup_{\mathbf{x}\neq\mathbf{y}} \frac{\left\langle \mathbf{x}-\mathbf{y}, \mathcal{F}(\mathbf{x}) - \mathcal{F}(\mathbf{y}) \right\rangle}{\|\mathbf{x}-\mathbf{y}\|_2^2}, \tag{1.3.7}$$

where $\langle \mathbf{u}, \mathbf{v} \rangle := \mathbf{u}^\top \mathbf{v}$, $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$, is the ℓ^2 inner product of $\mathbb{R}^d$. For a continuously differentiable function $\mathcal{F}$, one can prove that

$$\operatorname{osLip}(\mathcal{F}) = \sup_{\mathbf{x}\in\mathbb{R}^d} \mu_2\left(\partial_{\mathbf{x}}\mathcal{F}(\mathbf{x})\right),$$

where, for $\mathbf{A} \in \mathbb{R}^{d\times d}$,

$$\mu_2(\mathbf{A}) := \lim_{h\to 0^+} \frac{\|\mathbf{I}_d + h\mathbf{A}\|_2 - 1}{h} = \lambda_{\max}\left(\frac{\mathbf{A}+\mathbf{A}^\top}{2}\right)$$

is the logarithmic norm induced by the ℓ^2 norm $\|\cdot\|_2$. Such a result can also be extended to less regular Lipschitz continuous functions, for which the Jacobian can be defined almost everywhere, see [21, Theorem 15]. When nothing more is specified, with osLip, we refer to the one-sided Lipschitz constant induced by the Euclidean ℓ^2 norm of $\mathbb{R}^d$, (1.3.7). We say a vector field $\mathcal{F}$ is non-expansive in the Euclidean norm if $\operatorname{osLip}(\mathcal{F}) \leq 0$, and contractive if the inequality is strict. A consequence of these properties is indeed that since

$$\begin{aligned}\frac{d}{dt}\frac{1}{2}\left\|\Phi_{\mathcal{F}}^t(\mathbf{x}) - \Phi_{\mathcal{F}}^t(\mathbf{y})\right\|_2^2 &= \left\langle \mathcal{F}\left(\Phi_{\mathcal{F}}^t(\mathbf{x})\right) - \mathcal{F}\left(\Phi_{\mathcal{F}}^t(\mathbf{y})\right), \Phi_{\mathcal{F}}^t(\mathbf{x}) - \Phi_{\mathcal{F}}^t(\mathbf{y}) \right\rangle \\ &\leq \operatorname{osLip}(\mathcal{F})\left\|\Phi_{\mathcal{F}}^t(\mathbf{x}) - \Phi_{\mathcal{F}}^t(\mathbf{y})\right\|_2^2,\end{aligned}$$

one has that for non-expansive vector fields

$$\left\|\Phi_{\mathcal{F}}^{t}(\mathbf{x})-\Phi_{\mathcal{F}}^{t}(\mathbf{y})\right\|_2 \leq \|\mathbf{x}-\mathbf{y}\|_2,$$

and for contractive ones

$$\left\|\Phi_{\mathcal{F}}^{t}(\mathbf{x})-\Phi_{\mathcal{F}}^{t}(\mathbf{y})\right\|_2 \leq e^{\mathrm{osLip}(\mathcal{F})\,t}\,\|\mathbf{x}-\mathbf{y}\|_2 < \|\mathbf{x}-\mathbf{y}\|_2$$

for every $\mathbf{x},\mathbf{y}\in\mathbb{R}^d$ and $t\geq 0$. As presented in [8], the notions of contractivity and non-expansivity extend to other norms. To do so, one can replace the ℓ^2-inner product in (1.3.7) with the inner product generating the norm of interest or with the more general notion of a weak pairing for norms which are not induced by an inner product, as for the ℓ^1 norm in Chapter 3.

Definition 1.6 (Weak pairing [8])**.** A weak-pairing $[\![\cdot,\cdot]\!]:\mathbb{R}^d\times\mathbb{R}^d\to\mathbb{R}$ is a map satisfying the following properties:

- $[\![\mathbf{x}_1+\mathbf{x}_2,\mathbf{y}]\!]\leq[\![\mathbf{x}_1,\mathbf{y}]\!]+[\![\mathbf{x}_2,\mathbf{y}]\!]$ for every $\mathbf{x}_1,\mathbf{x}_2,\mathbf{y}\in\mathbb{R}^d$,
- $[\![\cdot,\mathbf{y}]\!]:\mathbb{R}^d\to\mathbb{R}$ is continuous for every $\mathbf{y}\in\mathbb{R}^d$,
- $[\![\alpha\mathbf{x},\mathbf{y}]\!]=[\![\mathbf{x},\alpha\mathbf{y}]\!]=\alpha[\![\mathbf{x},\mathbf{y}]\!]$ and $[\![-\mathbf{x},-\mathbf{y}]\!]=[\![\mathbf{x},\mathbf{y}]\!]$ for every $\alpha\geq 0$, $\mathbf{x},\mathbf{y}\in\mathbb{R}^d$,
- $[\![\mathbf{x},\mathbf{x}]\!]>0$ for every $\mathbf{x}\in\mathbb{R}^d\setminus\{\mathbf{0}\}$,
- $|[\![\mathbf{x},\mathbf{y}]\!]|\leq[\![\mathbf{x},\mathbf{x}]\!]^{1/2}[\![\mathbf{y},\mathbf{y}]\!]^{1/2}$.

Inner products are particular examples of weak pairings. Furthermore, any norm $\|\cdot\|$ of $\mathbb{R}^d$ can be realised as $\|\mathbf{x}\|=[\![\mathbf{x},\mathbf{x}]\!]^{1/2}$ for at least one weak pairing. The example relevant for Chapter 3 is $[\![\mathbf{x},\mathbf{y}]\!]_1:=\|\mathbf{y}\|_1\operatorname{sign}(\mathbf{y})^\top\mathbf{x}$, where $\operatorname{sign}(\mathbf{y})=\begin{bmatrix}\operatorname{sign}(y_1) & \cdots & \operatorname{sign}(y_d)\end{bmatrix}^\top$, for which it holds $\|\mathbf{x}\|_1^2=[\![\mathbf{x},\mathbf{x}]\!]_1$ since

$$[\![\mathbf{x},\mathbf{x}]\!]_1=\|\mathbf{x}\|_1\operatorname{sign}(\mathbf{x})^\top\mathbf{x}=\|\mathbf{x}\|_1\sum_{i=1}^{d}\operatorname{sign}(x_i)x_i=\|\mathbf{x}\|_1\sum_{i=1}^{d}|x_i|=\|\mathbf{x}\|_1^2.$$

For the ℓ^1 norm, one can define the one-sided Lipschitz constant as

$$\mathrm{osLip}_1(\mathcal{F})=\sup_{\mathbf{x}\neq\mathbf{y}}\frac{\left[\!\left[\mathcal{F}(\mathbf{x})-\mathcal{F}(\mathbf{y}),\mathbf{x}-\mathbf{y}\right]\!\right]_1}{\|\mathbf{x}-\mathbf{y}\|_1^2}=\sup_{\mathbf{x}\neq\mathbf{y}}\frac{\operatorname{sign}(\mathbf{x}-\mathbf{y})^\top\left(\mathcal{F}(\mathbf{x})-\mathcal{F}(\mathbf{y})\right)}{\|\mathbf{x}-\mathbf{y}\|_1},$$

and again have

$$\mathrm{osLip}_1(\mathcal{F})=\sup_{\mathbf{x}\in\mathbb{R}^d}\mu_1\left(\partial_{\mathbf{x}}\mathcal{F}(\mathbf{x})\right)=\sup_{\mathbf{x}\in\mathbb{R}^d}\left(\lim_{h\to 0^+}\frac{\|\mathbf{I}_d+h\partial_{\mathbf{x}}\mathcal{F}(\mathbf{x})\|_1-1}{h}\right)$$

for differentiable functions.

Let us consider a vector field $\mathcal{F}$ on $\mathbb{R}^d$ for which

$$\left\|\Phi_{\mathcal{F}}^t(\mathbf{x}) - \Phi_{\mathcal{F}}^t(\mathbf{y})\right\| \leq \|\mathbf{x}-\mathbf{y}\| \tag{1.3.8}$$

for a generic norm $\|\cdot\|$ of $\mathbb{R}^d$, and for every $t \geq 0$ and $\mathbf{x},\mathbf{y} \in \mathbb{R}^d$. In this case, one would like to replicate such behaviour also at a discrete level, and hence work with a one-step method $\Psi_{\mathcal{F}}^{\Delta t} : \mathbb{R}^d \to \mathbb{R}^d$ for which

$$\left\|\Psi_{\mathcal{F}}^{\Delta t}(\mathbf{x}) - \Psi_{\mathcal{F}}^{\Delta t}(\mathbf{y})\right\| \leq \|\mathbf{x}-\mathbf{y}\| \tag{1.3.9}$$

for every $\mathbf{x},\mathbf{y} \in \mathbb{R}^d$ and time step $\Delta t > 0$. This 1−Lipschitz property of the map $\Psi_{\mathcal{F}}^{\Delta t}$ would guarantee the method's stability since the perturbations to the initial conditions are reduced in their norm when passing through $\Psi_{\mathcal{F}}^{\Delta t}$. In [85], it was proven that a method satisfying (1.3.9) for every norm $\|\cdot\|$ and vector field $\mathcal{F}$ for which (1.3.8) holds, can have at most order 1. Furthermore, the authors of [23] proved that implicit Euler is one of these methods. Restricting the analysis to norms generated by an inner product, as the ℓ^2 norm, the non-expansivity condition in (1.3.9) is called B-stability. For these norms, there is no order limitation, and a Runge-Kutta method with Butcher tableau defined by $(\mathbf{A},\mathbf{b},\mathbf{c})$ is B-stable if the two matrices $\mathbf{B} = \mathrm{diag}(\mathbf{b})$, and $\mathbf{M} = \mathbf{BA}+\mathbf{A}^\top\mathbf{B}-\mathbf{bb}^\top$ are positive semi-definite [9], where $\mathbf{B}$ is a diagonal matrix with diagonal given by the vector $\mathbf{b}$. Since B-stable methods are also A-stable, explicit Runge-Kutta methods can not be B-stable given that they have bounded regions of absolute stability [48, Section 4.3].

The notion of contractive and non-expansive systems is fundamental for this thesis since a considerable part of the work has been on reducing the sensitivity of neural networks to input perturbations thanks to the theory of non-expansive systems. However, implicit solvers are prohibitive when dealing with big-sized differential equations such as those we work with for image or graph-node classification tasks. Differently from B-stability, which leads to unconditional stability, i.e., no need for a restriction over the step size Δt, with explicit methods, one has to restrict the step size to achieve non-expansivity. For example, applying the explicit Euler method to a vector field $\mathcal{F}$ with $\mathrm{osLip}(\mathcal{F}) < 0$ and Lipschitz constant $L > 0$, one can derive the following step size restriction:

$$\begin{aligned}
&\left\|\Psi_{\mathcal{F}}^{\Delta t}(\mathbf{x}) - \Psi_{\mathcal{F}}^{\Delta t}(\mathbf{y})\right\|_2^2 = \left\langle \mathbf{x}-\mathbf{y}+\Delta t\left(\mathcal{F}(\mathbf{x})-\mathcal{F}(\mathbf{y})\right), \mathbf{x}-\mathbf{y}+\Delta t\left(\mathcal{F}(\mathbf{x})-\mathcal{F}(\mathbf{y})\right)\right\rangle \\
&= \|\mathbf{x}-\mathbf{y}\|_2^2 + \Delta t^2\left\|\mathcal{F}(\mathbf{x})-\mathcal{F}(\mathbf{y})\right\|_2^2 + 2\Delta t\left\langle \mathbf{x}-\mathbf{y}, \mathcal{F}(\mathbf{x})-\mathcal{F}(\mathbf{y})\right\rangle \\
&\leq \left(1+\Delta t^2 L^2 + 2\mathrm{osLip}(\mathcal{F})\Delta t\right)\|\mathbf{x}-\mathbf{y}\|_2^2 \leq \|\mathbf{x}-\mathbf{y}\|_2^2
\end{aligned}$$

if $\Delta t^2 L^2 + 2\text{osLip}(\mathcal{F})\Delta t \leq 0$ and hence $\Delta t \leq -2\text{osLip}(\mathcal{F})/L^2$. For some vector fields, as for the gradient flows we consider in Chapters 2 and 3, the theory of circle-contractivity [19] can provide the needed step size restrictions even in the non-expansive case.

1.4 Structure preserving deep learning

We now focus on Residual Neural Networks (ResNets). As introduced in Section 1.2, ResNets can be interpreted as the numerical approximation of the solutions of parametric initial value problems of the form

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathcal{F}\left(\mathbf{x}(t), \theta(t)\right) \in \mathbb{R}^d, \ \theta(t) \in \Theta \\ \mathbf{x}(0) = \mathbf{x}_0 \in \mathbb{R}^d \end{cases} . \tag{1.4.1}$$

Depending on the choice of the numerical method $\Psi_{\mathcal{F}}^{\Delta t} : \mathbb{R}^d \to \mathbb{R}^d$ and how $\mathcal{F}$ is modelled, it is possible to obtain a different neural network architecture. Furthermore, given that the dynamical system in (1.4.1) is only of interest for modelling purposes, one can assume without loss of generality that $\theta : \mathbb{R} \to \Theta$ is piecewise-constant in time. This assumption leads to a piecewise autonomous time-switching system, as we assume in Chapters 2 and 3.

Neural networks are parametric functions that aim to approximate an unknown target map $f : \mathbb{R}^d \to \mathbb{R}^c$. When a property is known to be satisfied by f, one may be interested in ensuring that the neural network also satisfies that property. An example is a neural network approximating the solution of a differential equation which is known to be on the sphere $\mathcal{S}_r^{d-1} = \left\{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\|_2^2 = r^2\right\}$ of radius r. In that case, to increase the interpretability of the approximation, the network should map points in $\mathcal{S}_r^{d-1}$ to points in $\mathcal{S}_r^{d-1}$. We consider this problem in Chapter 4, where we experimentally show that, for the case of the semi-discretised linear advection PDE, such a constraint significantly improves the stability of the network as an approximation of the flow map.

Furthermore, it might be that one does not know if the target function has a peculiar structure while still being interested in approximating it with a constrained map. This situation occurs, for example, when building neural networks that can classify images or graph nodes while being robust to input perturbations. In this case, it is not known how the perfect classifier behaves outside of the training set of input-output labelled pairs. However, we show that it is beneficial to model the neural network so that it has a small Lipschitz constant, i.e., a reduced sensitivity to input perturbations. This example is considered in Chapters 2 for image classifiers and 3 for graph-nodes classifiers.

We now introduce our methodology to impose a specific property over networks, i.e., to obtain structured neural networks. We suppose this property is closed under function compositions as, for example, when considering symplectic functions, functions from a manifold $\mathcal{M}$ onto itself, volume-preserving maps, or functions with a Lipschitz constant smaller than 1. We illustrate the strategy first for the specific property of volume preservation and then extend it to a more general setting.

The first step to designing volume-preserving neural networks is to build a family of parametric divergence-free vector fields $\mathcal{S}_\Theta$, i.e., whose solutions preserve the canonical volume form. For this derivation, we consider the partitioning of the configuration variable $\mathbf{x} \in \mathbb{R}^d$ as $\mathbf{x} = \begin{bmatrix} \mathbf{x}_1^\top & \mathbf{x}_2^\top \end{bmatrix}^\top$ with $\mathbf{x}_1 \in \mathbb{R}^{d_1}$ and $\mathbf{x}_2 \in \mathbb{R}^{d_2}$. We then introduce

$$\mathcal{S}_\Theta = \left\{ \mathcal{F}_\theta(\mathbf{x}) = \begin{bmatrix} \sigma\left(\mathbf{A}_1 \mathbf{x}_2 + \mathbf{b}_1\right) \\ \sigma\left(\mathbf{A}_2 \mathbf{x}_1 + \mathbf{b}_2\right) \end{bmatrix} \in \mathbb{R}^d : \theta = \left(\mathbf{A}_1, \mathbf{A}_2, \mathbf{b}_1, \mathbf{b}_2\right) \in \Theta \right\},$$

where, for example, $\Theta = \mathbb{R}^{d_1 \times d_2} \times \mathbb{R}^{d_2 \times d_1} \times \mathbb{R}^{d_1} \times \mathbb{R}^{d_2}$. Then, we need to choose a numerical one-step method $\Psi_{\mathcal{F}_\theta}^{\Delta t}$ which preserves the volume form, see Section 1.3.2. We rely on the Lie Trotter splitting method, which leads to network layers of the form

$$\begin{aligned} \mathbf{x}^n \mapsto \hat{\mathbf{x}}^{n+1} &= \mathbf{x}^n + \Delta t \begin{bmatrix} \sigma\left(\mathbf{A}_1 \mathbf{x}_2^n + \mathbf{b}_1\right) \\ 0 \end{bmatrix} \\ \hat{\mathbf{x}}^{n+1} \mapsto \mathbf{x}^{n+1} &= \hat{\mathbf{x}}^{n+1} + \Delta t \begin{bmatrix} 0 \\ \sigma\left(\mathbf{A}_2 \hat{\mathbf{x}}_1^{n+1} + \mathbf{b}_2\right) \end{bmatrix} =: \Psi_{\mathcal{F}_\theta}^{\Delta t}(\mathbf{x}_n). \end{aligned} \tag{1.4.2}$$

This parametrisation allows us to get volume-preserving neural networks by composing layers like the one in (1.4.2). Volume preservation is also important because it guarantees the network is invertible, which can be of interest in several applications where, for example, the network is trained to learn a change of variables.

Generalising, we obtain a procedure for structure preservation in neural networks. The first step is to design a family of parametric vector fields $\mathcal{S}_\Theta$ whose solutions satisfy the target property. Here are a few examples:

1. For the 1−Lipschitz property, one can use contractive dynamical systems like $\dot{\mathbf{x}}(t) = \mathcal{F}_\theta(\mathbf{x}(t)) = -\mathbf{A}^\top \sigma\left(\mathbf{A}\mathbf{x}(t) + \mathbf{b}\right)$, $\theta = (\mathbf{A}, \mathbf{b})$, as in Chapter 2.

2. For symplectic neural networks, one can work with parametric Hamiltonian systems like $\dot{\mathbf{x}}(t) = \mathcal{F}_\theta(\mathbf{x}(t)) = \mathbb{J}\nabla H_\theta\left(\mathbf{x}(t)\right) \in \mathbb{R}^{2d}$, for example

by setting $H_\theta(\mathbf{x}) = \mathbf{c}^\top \sigma\left(\mathbf{A}\mathbf{x}+\mathbf{b}\right)$, $\mathbf{A} \in \mathbb{R}^{h\times 2d}$, $\mathbf{b} \in \mathbb{R}^h$, and $\mathbf{c} \in \mathbb{R}^h$, with $\theta = \left(\mathbf{A}, \mathbf{b}, \mathbf{c}\right)$. We proceed in this way in Chapter 6.

3. For the preservation of a manifold $\mathcal{M} \subset \mathbb{R}^d$ one can use any family of parametric vector fields and orthogonally project them on the tangent space like $\dot{\mathbf{x}}(t) = \mathcal{F}_\theta(\mathbf{x}(t)) = P\left(\mathbf{x}(t)\right)\mathbf{B}^\top \sigma\left(\mathbf{A}\mathbf{x}(t)+\mathbf{b}\right)$, with $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{h\times d}$, $\mathbf{b} \in \mathbb{R}^h$, $\theta = \left(\mathbf{A}, \mathbf{B}, \mathbf{b}\right)$, and $P(\mathbf{x}) : \mathbb{R}^d \to T_{\mathbf{x}}\mathcal{M}$ is the orthogonal projection onto the tangent space at $\mathbf{x}$ of $\mathcal{M}$. In the case of the unit sphere $\mathcal{M} = \mathcal{S}^2$, $P(\mathbf{x}) = \mathbf{I}_3 - \mathbf{x}\mathbf{x}^\top$. We proceed in this way in Chapters 4 and 6.

4. For the preservation of a function $I : \mathbb{R}^d \to \mathbb{R}$, one can adopt the skew-gradient formulation $\dot{\mathbf{x}}(t) = \mathcal{F}_\theta(\mathbf{x}(t)) = \left(A_\theta\left(\mathbf{x}(t)\right) - A_\theta\left(\mathbf{x}(t)\right)^\top\right)\nabla I\left(\mathbf{x}(t)\right)$, see [76], where A_θ is a matrix-valued neural network. We proceed in this way for mass-preserving neural networks in Chapter 2.

Properly designing the vector fields is not enough. Indeed, one also needs to choose a numerical method $\Psi_{\mathcal{F}_\theta}^{\Delta t}$ which reproduces the desired structure at a discrete level. The techniques we adopt in the following chapters are all introduced in Section 1.3. The constrained neural network architecture can be obtained by composing single steps with $\Psi_{\mathcal{F}_\theta}^{\Delta t}$ of the structured vector fields as

$$\mathcal{N}_\theta = \Psi_{\mathcal{F}_{\theta_L}}^{\Delta t_L} \circ \cdots \circ \Psi_{\mathcal{F}_{\theta_1}}^{\Delta t_1},$$

with $\mathcal{F}_{\theta_1}, ..., \mathcal{F}_{\theta_L} \in \mathcal{S}_\Theta$ and $\theta = \left(\theta_1, ..., \theta_L\right)$.

The parametric set $\mathcal{S}_\Theta$ should be chosen so that its elements enable the design of a network that accurately solves the task at hand. While for more conventional neural networks many approximation theorems have been developed [17, 41, 46, 63, 70], less is known for more structured ones. We present results in this direction in Chapter 2, where we work with the Presnov decomposition of vector fields [74] and splitting methods to show that neural networks based on gradient flows and norm-preserving dynamical systems can approximate a broad class of target functions.

Since in Chapter 2 we do not derive explicitly the vector field decomposition, we introduce it here.

Theorem 1.1 (Presnov decomposition)**.** *Let $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^d$ be a continuously differentiable vector field. Then, there exists a unique function $\varphi : \mathbb{R}^d \to \mathbb{R}$ and a vector field $u : \mathbb{R}^d \to \mathbb{R}^d$ such that $\varphi(\mathbf{0}) = 0$, $\mathcal{F}(\mathbf{x}) = u(\mathbf{x}) + \nabla\varphi(\mathbf{x})$, and $\mathbf{x}^\top u(\mathbf{x}) = 0$ for every $x \in \mathbb{R}^d$.*

Proof. Let $\varphi(\mathbf{x}) = \int_0^1 \frac{1}{t}\sigma(t\mathbf{x})\mathrm{d}t$ where $\sigma(\mathbf{x}) = \mathbf{x}^\top \mathcal{F}(\mathbf{x})$. We then define the vector field $u(\mathbf{x}) = \mathcal{F}(\mathbf{x}) - \nabla\varphi(\mathbf{x})$. The existence of the decomposition follows

from the following calculation

$$\mathbf{x}^\top \nabla\varphi(\mathbf{x}) = \sum_{i=1}^{d} x_i \frac{\partial}{\partial x_i}\left(\int_0^1 \frac{1}{t}\sigma(t\mathbf{x})\,\mathrm{d}t\right) = \sum_{i=1}^{d} x_i \int_0^1 \frac{\partial\sigma}{\partial z_i}(\mathbf{z})\Big|_{\mathbf{z}=t\mathbf{x}}\,\mathrm{d}t$$
$$= \int_0^1 \frac{d}{dt}\sigma(t\mathbf{x})\,\mathrm{d}t = \sigma(\mathbf{x}) - \sigma(\mathbf{0}) = \mathbf{x}^\top \mathcal{F}(\mathbf{x}).$$

Let us assume there is another such decomposition $\mathcal{F}(\mathbf{x}) = v(\mathbf{x}) + \nabla\psi(\mathbf{x})$. Then,

$$0 = \mathbf{x}^\top \nabla\big(\psi(\mathbf{x}) - \varphi(\mathbf{x})\big) \implies \psi(\mathbf{x}) = \varphi(\mathbf{x})$$

since $\psi(\mathbf{0}) = \varphi(\mathbf{0}) = 0$, which also implies $u(\mathbf{x}) = v(\mathbf{x})$. □

We call $u(\mathbf{x})$ a sphere-preserving vector field since the property $\mathbf{x}^\top u(\mathbf{x}) = 0$ implies $u(\mathbf{x}) \in T_{\mathbf{x}}\mathcal{S}^{d-1}_{\|\mathbf{x}\|_2}$. This theorem allows us to show that any continuous function on a compact set can be approximated arbitrarily well in L^p norms with sphere-preserving and gradient flows.

For illustration purposes, we include Figure 1.3, where a constrained neural network built following this dynamical systems-based approach is compared to a more conventional unconstrained network. We train the two networks so they approximate the flow map $\Phi^{0.05}_{\mathcal{F}}$ of the SIR model. Such a dynamical system conserves the linear function $I(\mathbf{x}) = x_1 + x_2 + x_3$, $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^\top$. The constrained network replicates this conservation law, see Figure 1.3.

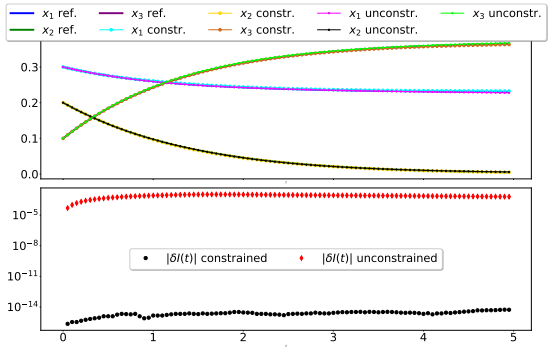


Figure 1.3: Comparison of the approximate solutions obtained with a neural network constrained to preserve the function $I(\mathbf{x}) = x_1 + x_2 + x_3$, and one where this is not imposed on the architecture. In the legend, with $|\delta I(t)|$ we refer to $\left|I(\mathbf{x}(t)) - I(\mathbf{x}_0)\right|$.

Lastly, we remark that constraining the parametric function $\mathcal{N}_\theta$ using the theory of dynamical systems is not the only possibility. For example, a well-developed framework is the Theory of Functional Connections [57], which

analyses how to modify a parametric function so it passes through a collection of target points. We work with such a theory in Chapter 7 to get neural networks satisfying the boundary conditions of Euler's elastica boundary value problem, and in Chapter 8 to impose the initial condition of a neural network-based ODE solver.

We also remark that, in machine learning, it is common to add a regularisation term to the loss function which promotes a desired structure or property over a network without modifying its architecture. Regularising the loss function does not guarantee that the network satisfies the desired structure. Throughout this thesis, we adopt various regularisation strategies and we leave their presentation to each of the separate chapters.

1.5 Solving and discovering differential equations

The research field occupied with finding approximate solutions to differential equations with neural networks and discovering a differential equation given some observed trajectories is vast and involves several challenges. We consider three problems in particular, which are

1. Approximating the right-hand side of a differential equation provided with some observed solution curves and possibly some additional information about the unknown equation (e.g., it is the semi-discretisation of a PDE of a certain type or a Hamiltonian equation).
2. Approximating the solutions of a differential equation provided with some observed solution curves.
3. Approximating the solutions of a differential equation in an unsupervised manner.

The first two problems are considered in Chapters 2, 4, 6 and 7. We analyse the third problem in Chapter 8. Problems 1 and 2 are supervised learning tasks since we need data to solve them, while the third is unsupervised since the learning occurs only based on the knowledge of the differential equation to solve. We now dedicate one subsection to each of the three problems, briefly presenting our methodology and anticipating some results.

1.5.1 Data-driven approximation of differential equations

Let us focus on the task of approximating the right-hand side of a differential equation of the form $\dot{\mathbf{x}}(t) = \mathcal{F}\big(\mathbf{x}(t)\big) \in \mathbb{R}^d$. We consider a set of N observed

trajectory segments defined as $\left\{\left(\mathbf{x}_n^0, \mathbf{x}_n^1, ..., \mathbf{x}_n^M\right)\right\}_{n=1}^{N}$, where $\mathbf{x}_n^m \approx \Phi_{\mathcal{F}}^{m\Delta t}\left(\mathbf{x}_n^0\right)$, with $m = 0, ..., M$, for a known step size $\Delta t > 0$. To approximate the map $\mathcal{F}$, we introduce a set of parametric functions $\mathcal{S}_\Theta = \left\{\mathcal{F}_\theta : \mathbb{R}^d \to \mathbb{R}^d : \theta \in \Theta\right\}$, and choose a numerical method $\Psi_{\mathcal{F}_\theta}^{\Delta t} : \mathbb{R}^d \to \mathbb{R}^d$. This setup allows us to define the loss function

$$\mathcal{L}(\theta) = \frac{1}{NM} \sum_{n=1}^{N} \sum_{m=1}^{M} \left\| \left(\Psi_{\mathcal{F}_\theta}^{\Delta t}\right)^m \left(\mathbf{x}_n^0\right) - \mathbf{x}_n^m \right\|_2^2 \to \min,$$

$$\left(\Psi_{\mathcal{F}_\theta}^{\Delta t}\right)^m := \underbrace{\Psi_{\mathcal{F}_\theta}^{\Delta t} \circ \cdots \circ \Psi_{\mathcal{F}_\theta}^{\Delta t}}_{m \text{ times}}.$$

This constitutes the starting point for all the papers studying this task and is customised for the specific problems we consider.

The first variant of the methodology, proposed in Chapter 6, is to structure the parametric functions in $\mathcal{S}_\Theta$ so that the functions that can be represented are only Hamiltonian vector fields. This restriction can be achieved, for example, by setting $\mathcal{F}_\theta(\mathbf{x}) = \mathbb{J}\nabla_{\mathbf{x}} H_\theta(\mathbf{x})$, $H_\theta : \mathbb{R}^d \to \mathbb{R}$. Another constraining strategy we adopt in Chapter 6 is to project the vector fields on the tangent space to a non-linear $c-$dimensional manifold $\mathcal{M} \subset \mathbb{R}^d$ defined as the zero set of a constraint function $g : \mathbb{R}^d \to \mathbb{R}^{d-c}$. In this case, the components of g are supposed to be functionally independent, so that $\dim(\mathcal{M}) = c$. The orthogonal projection of the vector field to the correct tangent space then leads to $\mathcal{F}_\theta$ of the form

$$\mathcal{F}_\theta(\mathbf{x}) = \left(\mathbf{I}_d - \partial_{\mathbf{x}} g(\mathbf{x})^\top \left(\partial_{\mathbf{x}} g(\mathbf{x}) \partial_{\mathbf{x}} g(\mathbf{x})^\top\right)^{-1} \partial_{\mathbf{x}} g(\mathbf{x})\right) \hat{\mathcal{F}}_\theta(\mathbf{x}) = P(\mathbf{x}) \hat{\mathcal{F}}_\theta(\mathbf{x}),$$

where $\hat{\mathcal{F}}_\theta : \mathbb{R}^d \to \mathbb{R}^d$ is a generic parametric function. We remark that $P(\mathbf{x}) : \mathbb{R}^d \to T_{\mathbf{x}}\mathcal{M} = \ker\left(\partial_{\mathbf{x}} g(\mathbf{x})\right)$ is the orthogonal projection matrix onto the tangent space at $\mathbf{x}$ of $\mathcal{M}$. In Chapter 6, we proceed in this way for vector fields that are tangent to products of cotangent bundles of $2-$spheres.

The second variant we present in the papers relates to the choice of the time stepping scheme $\Psi_{\mathcal{F}_\theta}^{\Delta t}$. Indeed, when something more is known about the underlying differential equation, choosing one method rather than another might be reasonable. For example, in Chapter 6, we consider unconstrained Hamiltonian systems and holonomically constrained Hamiltonian systems. For the first setting, we choose $\Psi_{\mathcal{F}_\theta}^{\Delta t}$ as a symplectic method, hence being more compatible with the dynamics one tries to discover. For the case of constrained systems, we instead focus on systems constrained to a homogeneous manifold and hence set $\Psi_{\mathcal{F}_\theta}^{\Delta t}$ to a Lie group integrator.

1.5.2 Data-driven approximation of the solutions of a differential equation

The methodology presented for the data-driven approximation of differential equations can be adapted to approximate the flow map of an ODE. If the goal is to find an approximation of the map sending one point of the observed trajectory segments into the following one, i.e., a map approximating $\Phi_{\mathcal{F}}^{\Delta t}$, then one can follow the same steps and the result can be found in the map $\Psi_{\mathcal{F}_\theta}^{\Delta t}$ rather than in $\mathcal{F}_\theta$ as we did before. We follow this procedure in Chapter 2 for mass-preserving dynamical systems and in Chapter 4 for PDE semi-discretisations.

Up to now, we have focused on time-dependent differential problems. In Chapter 7, we instead consider Euler's elastica, i.e., a spatial Boundary Value Problem (BVP) on a one-dimensional domain. We aim to learn approximate solutions to Euler's elastica given the boundary conditions of the differential problem. This is a supervised learning task, and the neural network approximation is found by minimising a mean squared error loss function based on data corresponding to numerical solutions to that BVP for a few boundary values. In this setting, there is no temporal dependency in the considered problem, and hence, we do not use the dynamical systems approach to design and structure neural networks. Instead, we rely on a feedforward fully connected neural network and use the theory of functional connections to structure the network so it satisfies the boundary conditions.

1.5.3 Unsupervised approximation of ODE solutions

The third problem we consider is solving a differential equation based on a neural network without training data. This line of research has grown considerably in the last few years, especially after the publication of [77], where Physics Informed Neural Networks (PINNs) have been introduced. The concept of PINNs is a revival of earlier studies from the 1990s [56]. The basic principle behind PINNs is to train a neural network to minimise the residual between the left and the right-hand sides of a differential equation on a set of collocation points. We focus on a prototypical ODE of the form

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathcal{F}\left(\mathbf{x}(t)\right) \in \mathbb{R}^d \\ \mathbf{x}(0) = \mathbf{x}_0 \in \mathbb{R}^d \end{cases}, \tag{1.5.1}$$

which we want to solve on the time interval $[0, T]$. To do so, we introduce a neural network $\mathcal{N}_\theta : \mathbb{R} \to \mathbb{R}^d$ and the loss function

$$\mathcal{L}\left(\theta\right) = \frac{1}{N} \sum_{n=1}^{N} \left\| \frac{d}{dt} \mathcal{N}_\theta(t) \Big|_{t=t_n} - \mathcal{F}\left(\mathcal{N}_\theta\left(t_n\right)\right) \right\|_2^2 + \gamma \left\| \mathcal{N}_\theta(0) - \mathbf{x}_0 \right\|_2^2 \to \min,$$

based on a set of N collocation points $t_1, ..., t_N \in [0, T]$, which can be randomly drawn from a probability distribution or chosen in other ways. We remark that the term multiplied by $\gamma \geq 0$, a parameter to specify, aims to impose the initial condition over the approximate solution. In case one desires to impose such an initial condition exactly, simple restrictions could do that, like setting $\mathcal{N}_\theta(t) = \mathbf{x}_0 + \check{\mathcal{N}}_\theta(t) - \check{\mathcal{N}}_\theta(0)$ for an unconstrained parametric function $\check{\mathcal{N}}_\theta$. From this basic methodology, different approaches have been developed, see, e.g. [50, 51]. One extension we work with in Chapter 8 is to not only solve a single initial value problem as the one in (1.5.1) but a collection of them by varying the initial condition. This methodology allows to get an approximation of the flow map $\Phi_{\mathcal{F}}^t$ for $t \in [0, T]$ and initial conditions in some compact set $\Omega \subset \mathbb{R}^d$ from which they are sampled while training the network. The basic procedure to introduce such an extension is by considering a neural network $\mathcal{N}_\theta : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, and optimising the loss function

$$\begin{aligned} \mathcal{L}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \left\| \partial_t \mathcal{N}_\theta\left(\mathbf{x}_n^0, t\right)\Big|_{t=t_n} - \mathcal{F}\left(\mathcal{N}_\theta\left(\mathbf{x}_n^0, t_n\right)\right) \right\|_2^2 \\ + \gamma \sum_{n=1}^{N} \left\| \mathcal{N}_\theta\left(\mathbf{x}_n^0, 0\right) - \mathbf{x}_n^0 \right\|_2^2 \end{aligned} \tag{1.5.2}$$

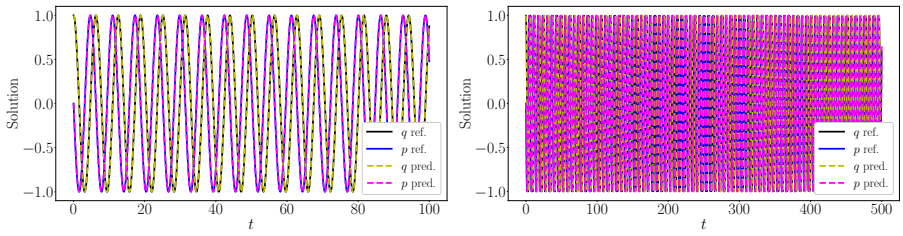
for a set of N initial conditions $\mathbf{x}_1^0, ..., \mathbf{x}_N^0 \in \Omega \subset \mathbb{R}^d$ and collocation points $t_1, ..., t_N$ in the interval $[0, T]$. Balancing the two terms in (1.5.2) is often problematic. It is thus favourable to enforce $\mathcal{N}_\theta(\mathbf{x}, 0) = \mathbf{x}$ for every $\mathbf{x} \in \mathbb{R}^d$ in the network architecture. Choosing to learn the solution operator $\mathcal{N}_\theta : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ is of more practical interest since once the expensive offline training phase is complete, one could solve several initial value problems. Furthermore, as highlighted in [89], one can also get long-time simulations by training on a narrow time interval. Indeed, supposing Ω is a forward invariant set, i.e., $\Phi_{\mathcal{F}}^t(\mathbf{x}_0) \in \Omega$ for every $\mathbf{x}_0 \in \Omega$ and $t \geq 0$, one could train the network over the time interval $[0, 1]$ and initial conditions in Ω, and then extrapolate in time as follows

$$\Phi_{\mathcal{F}}^{k+\delta t}(\mathbf{x}_0) \approx \mathcal{N}_\theta\left(\left(\mathcal{N}_\theta\right)^k(\mathbf{x}_0, 1), \delta t\right), \ \delta t \in (0, 1), \ k \in \mathbb{N},$$

where $\left(\mathcal{N}_\theta\right)^k(\mathbf{x}_0, 1) := \mathcal{N}_\theta\left(\left(\mathcal{N}_\theta\right)^{k-1}(\mathbf{x}_0, 1), 1\right)$ for $k \geq 1$, $\left(\mathcal{N}_\theta\right)^0(\mathbf{x}_0, 1) := \mathbf{x}_0$. In this case, it is fundamental to have $\mathcal{N}_\theta(\mathbf{x}, 0) = \mathbf{x}$ to get an approximate solution which is at least continuous. Figure 1.4 illustrates the use of this approach for the ODE of a harmonic oscillator. The network we consider is defined as

$$\begin{aligned} \mathcal{N}_\theta(\mathbf{x}_0, t) &= \mathbf{x}_0 + \check{\mathcal{N}}_\theta(\mathbf{x}_0, t) - \check{\mathcal{N}}_\theta(\mathbf{x}_0, 0) \\ \mathbb{R}^3 \ni \begin{bmatrix} \mathbf{x}_0^\top & t \end{bmatrix}^\top &:= \mathbf{z} \mapsto \tanh\left(\mathbf{A}_0 \mathbf{z} + \mathbf{b}_0\right) =: \mathbf{z}_1 \in \mathbb{R}^{10} \\ \mathbf{z}_1 &\mapsto \tanh\left(\mathbf{A}_1 \mathbf{z}_1 + \mathbf{b}_1\right) =: \mathbf{z}_2 \in \mathbb{R}^{10} \\ \mathbf{z}_2 &\mapsto \mathbf{A}_2 \mathbf{z}_2 + \mathbf{b}_2 =: \check{\mathcal{N}}_\theta(\mathbf{x}_0, t) \in \mathbb{R}^2. \end{aligned} \tag{1.5.3}$$

We sample collocation points in the time interval $[0, 1]$ and initial conditions in $\Omega = [-1.2, 1.2]^2 \subset \mathbb{R}^2$. The obtained approximate solution is compared with a reference one computed with Runge-Kutta (5,4).



(a) Solution curves for $T = 100$ **(b)** Solution curves for $T = 500$

Figure 1.4: Solution curves obtained with the neural network in (1.5.3) trained to minimise the loss function in (1.5.2) with $\gamma = 0$.

In Chapter 8, we consider the problem of solving an initial value problem with neural networks while also being able to provide theoretical guarantees on the obtained approximation. We start from the Parareal method [28], which is a parallel-in-time algorithm based on two one-step numerical schemes. One is called a *coarse propagator*, which is cheap to evaluate, and the other is the so-called *fine propagator*, a more expensive method that provides higher accuracy. The coarse propagator allows for parallelism in time, and we propose replacing it with an Extreme Learning Machine (ELM) to develop a neural network-based ODE solver inheriting the guarantees of the Parareal method and the efficiency of ELMs. We refer to this method as a hybrid Parareal method. We show improved efficiency using ELMs when compared to more conventional neural networks trained to approximate the flow map. The results are supported by several numerical experiments, one illustrated in Figure 1.5, where the Parareal solution, denoted as "para", is compared with a reference solution denoted as "ref".

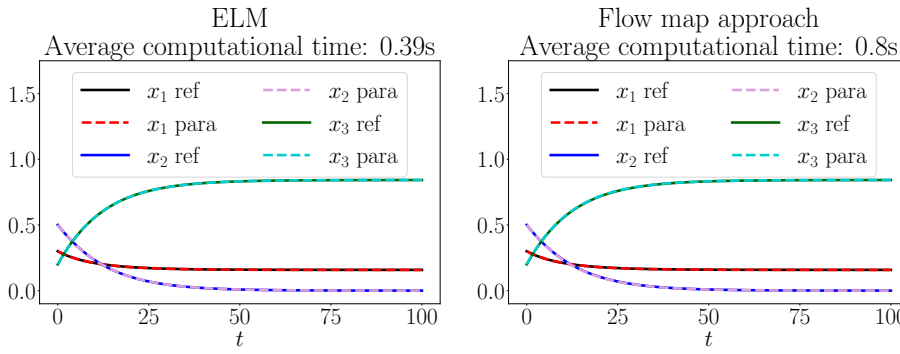


Figure 1.5: Solution of the SIR equations obtained with a hybrid Parareal method.

1.6 Summary of papers

The papers' layout and bibliographies have been adapted to have the same style. A few typos have been corrected. A few figures have been resized and placed differently to fit the B5 format. Some equations have been split to fit in the margins. No other substantial changes have been made to the papers.

Paper 1: Dynamical Systems–Based Neural Networks

Elena Celledoni, Davide Murari, Brynjulf Owren,
Carola-Bibiane Schönlieb, and Ferdia Sherry

SIAM Journal on Scientific Computing,
Vol. 45, No. 6, 2023, pp. A3071-A3094

This paper investigates techniques based on dynamical systems and geometric numerical methods to design neural networks satisfying a specific property. We apply these ideas to design volume-preserving, symplectic, mass-preserving, and 1-Lipschitz neural networks. The paper's experimental analysis focuses on convolutional neural networks (CNNs) with reduced sensitivity to adversarial input perturbations for image classification. The paper also includes a theoretical section in which we use the Presnov vector field decomposition and splitting methods to provide two universal approximation theorems.

Paper 2: Resilient Graph Neural Networks: A Coupled Dynamical Systems Approach

Moshe Eliasof, Davide Murari,
Carola-Bibiane Schönlieb, and Ferdia Sherry

Submitted

This paper investigates techniques to improve the robustness of GNNs with respect to adversarial perturbations. The focus is on classifying the nodes of an input graph whose adjacency and feature matrices have been attacked with an additive perturbation. We design a GNN based on a coupled ODE system involving both the adjacency and the feature matrices. This system of ODEs is contractive in a weighted norm, and we provide both theoretical and experimental evidence that such a design strategy allows one to obtain GNNs with reduced sensitivity to input perturbations.

Paper 3: Predictions Based on Pixel Data: Insights from PDEs and Finite Differences

Elena Celledoni, James Jackaman, Davide Murari, and Brynjulf Owren

Submitted

This paper shows that CNNs can represent finite difference semi-discretisations of several PDEs. We use such insight to provide a theoretical analysis of the approximation properties of CNNs for time sequences coming from the space-time discretisation of a PDE. We present constructive results relying on the connection between finite-difference discretisations and the convolution operation, together with the ability of some non-linear activation functions to reproduce linear and quadratic polynomials. Numerical experiments for the linear advection, heat, and Fisher equations support the theoretical analysis.

Paper 4: Lie Group integrators for mechanical systems

Elena Celledoni, Ergys Çokaj, Andrea Leone,
Davide Murari, and Brynjulf Owren

International Journal of Computer Mathematics,
Vol. 99, No. 1, 2022, pp. 58-88

This paper reviews the literature on Lie group integrators, focusing on the two classes of Runge-Kutta-Munthe-Kaas methods and commutator-free Lie group integrators. Additionally, we consider introducing step adaptivity in the discretisation algorithm. We demonstrate the theory with two applications in mechanics expressed via the Lagrangian formalism: the N-fold spherical pendulum and the dynamics of a system of two quadrotors transporting a payload.

Paper 5: Learning Hamiltonians of constrained mechanical systems

Elena Celledoni, Andrea Leone, Davide Murari, and Brynjulf Owren

Journal of Computational and Applied Mathematics,
Vol. 417, 2023, pp. 114608

In this work, we adopt Lie group integrators to approximate the Hamiltonian of holonomically constrained Hamiltonian systems. After introducing the learning procedure for unconstrained systems, we move to constrained systems. We

compare Lie group methods to Runge-Kutta integrators as a tool to approximate the Hamiltonian of constrained systems based on observed trajectory segments. One outcome of our analysis is that while to a higher order of the method consistently corresponds a higher accuracy of the approximate Hamiltonian, numerically preserving the constraint manifold does not systematically lead to similar improvements.

Paper 6: Neural networks for the approximation of Euler's elastica

Elena Celledoni, Ergys Çokaj, Andrea Leone, Sigrid Leyendecker, Davide Murari, Brynjulf Owren, Rodrigo T. Sato Martín de Almagro, Martina Stavole

Submitted

This paper presents three approaches for efficiently approximating Euler's elastica solution curves given their boundary conditions. We first present a methodology providing a discrete approximate solution over a set of spatial nodes. Given that the solution curves are continuous in space, we consider an alternative approximation strategy based on a network accepting both the boundary conditions and the arc-length parameter of the curve as inputs. We also consider an angular parametrisation of the tangent vectors so that we can enforce the normality constraints typical of this problem. We validate the three approaches with an extensive experimental analysis.

Paper 7: Parallel-in-Time Solutions with Extreme Learning Machines

Marta Betcke, Lisa Maria Kreusser, and Davide Murari

Preprint

This paper presents a hybrid parallel-in-time algorithm based on the Parareal method, where parts of the algorithm involve neural networks. First, we provide a theoretical analysis of the accuracy of neural network-based ODE solvers. This theoretical study, together with the convergence properties of the Parareal method, allows us to replace the coarse propagator of such an algorithm with an Extreme Learning Machine while retaining the convergence guarantees of the classical Parareal method. We demonstrate the effectiveness of the proposed methodology by applying it to several ODEs and a spatial semi-discretisation of the viscous Burgers' equation.

Papers not included in this thesis

My thesis includes seven papers I worked on during my PhD. I significantly contributed to both the theoretical and experimental analysis in all of these papers.

I have also collaborated on three more papers while being a PhD candidate.

Those in [11] and [69] are a conference proceeding and a non-archival report, respectively, based on two peer-reviewed papers appearing in this thesis. They are not included since they do not add substantial value to this thesis. [11] builds on Chapter 5, with experimental analysis proposing an experiment with variable step size for a chain of spherical pendula. [69] is based on Chapter 2, and its experimental section proposes a slight variation of the mass-preserving experiment we include in it.

In [83], we work with gradient flows as a tool to build non-expansive neural networks. We discuss the step size restrictions for generic Runge-Kutta methods based on the theory of circle-contractivity and investigate how these methods influence the network's performance. These 1-Lipschitz neural networks are applied to the problems of adversarial robustness, image denoising, and the inverse problem of deblurring. My contribution to this paper is limited to the theoretical analysis of the proposed architectures, which follows similar ideas of the paper in Chapter 2 of which I am the main author.

Bibliography

[1] John David Anderson and John Wendt. *Computational Fluid Dynamics*, volume 206. Springer, 1995. 2

[2] Simon Arridge, Peter Maass, Ozan Öktem, and Carola-Bibiane Schönlieb. Solving inverse problems using data-driven models. *Acta Numerica*, 28:1–174, 2019. 2, 48

[3] Francis Bach. Breaking the Curse of Dimensionality with Convex Neural Networks. *Journal of Machine Learning Research*, 18(19):1–53, 2017. 2

[4] George Keith Batchelor. *An Introduction to Fluid Dynamics*. Cambridge University Press, 1967. 2

[5] Susanne C Brenner. *The Mathematical Theory of Finite Element Methods*. Springer, 2008. 2, 252

[6] Michael Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. *arXiv preprint arXiv:2104.13478*, 2021. 9, 48

[7] Luigi Brugnano and Felice Iavernaro. Line integral methods which preserve all invariants of conservative problems. *Journal of Computational and Applied Mathematics*, 236(16):3905–3919, 2012. 14

[8] F. Bullo. *Contraction Theory for Dynamical Systems*. Kindle Direct Publishing, 1.1 edition, 2023. 21, 95, 98, 110, 119, 295, 296

[9] Kevin Burrage and John C Butcher. Stability Criteria for Implicit Runge-Kutta Methods. *SIAM Journal on Numerical Analysis*, 16(1):46–57, 1979. 22

[10] Elena Celledoni, Ergys Çokaj, Andrea Leone, Sigrid Leyendecker, Davide Murari, Brynjulf Owren, Rodrigo T Sato Martín de Almagro, and Martina Stavole. Neural networks for the approximation of Euler's elastica. *arXiv preprint arXiv:2312.00644*, 2024. 4

[11] Elena Celledoni, Ergys Çokaj, Andrea Leone, Davide Murari, and Brynjulf R Owren. Dynamics of the N-fold Pendulum in the framework of Lie Group Integrators. In *European Consortium for Mathematics in Industry*, pages 297–304. Springer, 2021. 35, 238

[12] Elena Celledoni, Ergys Çokaj, Andrea Leone, Davide Murari, and Brynjulf R Owren. Lie group integrators for mechanical systems. *International Journal of Computer Mathematics*, 99(1):58–88, 2022. 4

[13] Elena Celledoni, Volker Grimm, Robert I McLachlan, David I McLaren, D O'Neale, Brynjulf R Owren, and G Reinout W Quispel. Preserving energy resp. dissipation in numerical PDEs using the "Average Vector Field" method. *Journal of Computational Physics*, 231(20):6770–6789, 2012. 11

[14] Elena Celledoni, James Jackaman, Davide Murari, and Brynjulf R Owren. Predictions Based On Pixel Data: Insights from PDEs and Finite Differences. *arXiv preprint arXiv:2305.00723*, 2024. 4

[15] Elena Celledoni, Andrea Leone, Davide Murari, and Brynjulf R Owren. Learning Hamiltonians of constrained mechanical systems. *Journal of Computational and Applied Mathematics*, 417:114608, 2023. 4, 149

[16] Elena Celledoni, Davide Murari, Brynjulf R Owren, Carola-Bibiane Schönlieb, and Ferdia Sherry. Dynamical Systems–Based Neural Networks. *SIAM Journal on Scientific Computing*, 45(6):A3071–A3094, 2023. 3, 95, 145, 146

[17] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989. 3, 25, 246

[18] Germund G Dahlquist. A special stability problem for linear multistep methods. *BIT Numerical Mathematics*, 3(1):27–43, 1963. 19

[19] Germund G Dahlquist. Generalized disks of contractivity for explicit and implicit Runge-Kutta methods. Technical report, CM-P00069451, 1979. 23, 60

[20] Daryl J Daley and Joe Gani. *Epidemic Modelling: An Introduction*. Cambridge University Press, 2001. 2

[21] Alexander Davydov, Anton V Proskurnikov, and Francesco Bullo. Non-Euclidean Contractivity of Recurrent Neural Networks. In *2022 American Control Conference (ACC)*, pages 1527–1534. IEEE, 2022. 20

[22] Kees Dekker and Jan G Verwer. *Stability of Runge-Kutta Methods for Stiff Nonlinear Differential Equations*. North Holland, Amsterdam, 1984. 19

[23] Charles Desoer and Hiromasa Haneda. The measure of a matrix as a tool to analyze computer algorithms for circuit analysis. *IEEE Transactions on Circuit Theory*, 19(5):480–486, 1972. 22

[24] Sølve Eidnes. Order theory for discrete gradient methods. *BIT Numerical Mathematics*, 62(4):1207–1255, 2022. 14, 73

[25] Moshe Eliasof, Eldad Haber, and Eran Treister. PDE-GCN: Novel Architectures for Graph Neural Networks Motivated by Partial Differential Equations. *Advances in Neural Information Processing Systems*, 34:3836–3849, 2021. 9, 95, 99, 101, 113, 145

[26] Moshe Eliasof, Davide Murari, Ferdia Sherry, and Carola-Bibiane Schönlieb. Resilient Graph Neural Networks: A Coupled Dynamical Systems Approach. *arXiv preprint arXiv:2311.06942*, 2024. 3

[27] Erwan Faou. *Geometric Numerical Integration and Schrödinger Equations*, volume 15. European Mathematical Society, 2012. 11

[28] Martin J Gander and Ernst Hairer. Nonlinear Convergence Analysis for the Parareal Algorithm. In *Domain Decomposition Methods in Science and Engineering XVII*, pages 45–56. Springer, 2008. 31, 283, 284, 286, 287, 298

[29] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George Em Dahl. Neural Message Passing for Quantum Chemistry. In *International Conference on Machine Learning*, pages 1263–1272. PMLR, 2017. 9

[30] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep Sparse Rectifier Neural Networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011. 3

[31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 8, 159, 259

[32] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *Communications of the ACM*, 63(11):139–144, 2020. 3

[33] Philipp Grohs and Gitta Kutyniok. *Mathematical Aspects of Deep Learning*. Cambridge University Press, 2022. 3

[34] Yiqi Gu and Michael K Ng. Deep Neural Networks for Solving Large Linear Systems Arising from High-Dimensional Problems. *SIAM Journal on Scientific Computing*, 45(5):A2356–A2381, 2023. 2, 252

[35] Tianmei Guo, Jiwen Dong, Henjian Li, and Yunxing Gao. Simple convolutional neural network on image classification. In *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*, pages 721–724. IEEE, 2017. 8

[36] Ernst Hairer. Symmetric Projection Methods for Differential Equations on Manifolds. *BIT Numerical Mathematics*, 40:726–734, 2000. 16

[37] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration. Structure-Preserving Algorithms for Ordinary Differential Equations*. Springer, 2006. 2, 11, 12, 15

[38] Ernst Hairer, Syvert P Nørsett, and Gerhard Wanner. *Solving Ordinary Differential Equations I. Nonstiff Problems*, volume 8. Springer, 1993. 2, 282, 288, 289, 301, 303

[39] Brian C Hall and Brian C Hall. *Lie Groups, Lie Algebras, and Representations*. Springer, 2015. 12, 16

[40] James D Hamilton. *Time Series Analysis*. Princeton University Press, 2020. 2

[41] Juncai He, Lin Li, and Jinchao Xu. Approximation Properties of Deep ReLU CNNs. *Research in the Mathematical Sciences*, 9(3):38, 2022. 25

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 2, 3, 6, 8

[43] Herbert W Hethcote. The Mathematics of Infectious Diseases. *SIAM Review*, 42(4):599–653, 2000. 2

[44] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997. 3

[45] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982. 2

[46] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. 25

[47] David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148(3):574, 1959. 8

[48] Arieh Iserles. *A First Course in the Numerical Analysis of Differential Equations*. Number 44 in Cambridge Texts in Applied Mathematics (44). Cambridge University Press, 2009. 2, 22

[49] James Jackaman. *Finite element methods as geometric structure preserving algorithms*. PhD thesis, University of Reading, 2019. 14

[50] Ameya D Jagtap and George Em Karniadakis. Extended physics-informed neural networks (XPINNs): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Communications in Computational Physics*, 28(5), 2020. 30

[51] Ameya D Jagtap, Ehsan Kharazmi, and George Em Karniadakis. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 365:113028, 2020. 30

[52] Feng Kang and Shang Zai-Jiu. Volume-preserving algorithms for source-free dynamical systems. *Numerische Mathematik*, 71:451–463, 1995. 12

[53] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980, 2014. 3, 106

[54] Gebhard Kirchgässner, Jürgen Wolters, and Uwe Hassler. *Introduction to Modern Time Series Analysis*. Springer Science & Business Media, 2012. 2

[55] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 25, 2012. 3, 8

[56] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial Neural Networks for Solving Ordinary and Partial Differential Equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998. 29, 253, 271

[57] Carl Leake and Daniele Mortari. Deep Theory of Functional Connections: A New Method for Estimating the Solutions of Partial Differential Equations. *Machine Learning and Knowledge Extraction*, 2(1):37–55, 2020. 26

[58] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 1989. 3, 48

[59] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 3

[60] John M Lee and John M Lee. *Introduction to Smooth Manifolds*. Springer, 2012. 12, 16, 17, 18, 227, 230

[61] Taeyoung Lee, Melvin Leok, and N Harris McClamroch. *Global Formulations of Lagrangian and Hamiltonian Dynamics on Manifolds*, volume 13. Springer, 2017. 2, 226, 230, 240, 241

[62] Benedict Leimkuhler and Sebastian Reich. *Simulating Hamiltonian Dynamics*. Number 14 in Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2004. 11, 58, 70

[63] Qianxiao Li, Ting Lin, and Zuowei Shen. Deep learning via dynamical systems: An approximation perspective. *Journal of the European Mathematical Society*, 25(5):1671–1709, 2022. 25, 52, 54

[64] William A Little. The Existence of Persistent States in the Brain. *Mathematical Biosciences*, 19(1-2):101–120, 1974. 2

[65] Sebastian Lunz, Ozan Öktem, and Carola-Bibiane Schönlieb. Adversarial Regularizers in Inverse Problems. *Advances in Neural Information Processing Systems*, 31, 2018. 2

[66] Robert I McLachlan and G Reinout W Quispel. Splitting methods. *Acta Numerica*, 11:341–434, 2002. 11, 54, 57

[67] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using Deep Learning: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3523–3542, 2021. 8

[68] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*, volume 479. MIT Press, 1969. 2

[69] Davide Murari, Elena Celledoni, Brynjulf R Owren, Ferdia Sherry, and Carola-Bibiane Schönlieb. Structure preserving neural networks based on ODEs. In *NeurIPS 2022 Workshop DLDE*, 2022. 35

[70] Sebastian Neumayer, Alexis Goujon, Pakshal Bohra, and Michael Unser. Approximation of Lipschitz Functions Using Deep Spline Neural Networks. *SIAM Journal on Mathematics of Data Science*, 5(2):306–322, 2023. 25

[71] Luong Trung Nguyen, Junhan Kim, and Byonghyo Shim. Low-Rank Matrix Completion: A Contemporary Survey. *IEEE Access*, 7:94215–94237, 2019. 2

[72] Antonio Ortega, Pascal Frossard, Jelena Kovačević, José MF Moura, and Pierre Vandergheynst. Graph Signal Processing: Overview, Challenges, and Applications. *Proceedings of the IEEE*, 106(5):808–828, 2018. 9

[73] Allan Pinkus. Approximation theory of the MLP model in neural networks. *Acta Numerica*, 8:143–195, 1999. 3, 55, 155

[74] Eugene Presnov. Non-local decomposition of vector fields. *Chaos, Solitons & Fractals*, 14(5):759–764, 2002. 25, 54

[75] Alfio Quarteroni and Silvia Quarteroni. *Numerical Models for Differential Problems*, volume 2. Springer, 2009. 2

[76] G Reinout W Quispel and Grant S Turner. Discrete gradient methods for solving ODEs numerically while preserving a first integral. *Journal of Physics A: Mathematical and General*, 29(13):L341, 1996. 14, 25

[77] Maziar Raissi, Paris Perdikaris, and George Em 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. 29, 48, 252, 253, 271

[78] Frank Rosenblatt. *The Perceptron — A Perceiving and Recognizing Automaton*. Cornell Aeronautical Laboratory, 1957. 2

[79] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. 3

[80] Aliaksei Sandryhaila and José MF Moura. Discrete Signal Processing on Graphs. *IEEE Transactions on Signal Processing*, 61(7):1644–1656, 2013. 9

[81] Hanie Sedghi, Vineet Gupta, and Philip M. Long. The Singular Values of Convolutional Layers. In *International Conference on Learning Representations*, 2019. 9

[82] Ahmed A Shabana. *Dynamics of Multibody Systems*. Cambridge University Press, 2020. 2

[83] Ferdia Sherry, Elena Celledoni, Matthias J Ehrhardt, Davide Murari, Brynjulf R Owren, and Carola-Bibiane Schönlieb. Designing Stable Neural Networks using Convex Analysis and ODEs. *Physica D: Nonlinear Phenomena*, page 134159, 2024. 35, 145

[84] Robert H Shumway, David S Stoffer, and David S Stoffer. *Time series analysis and its applications*, volume 3. Springer, 2000. 2

[85] MN Spijker. Contractivity in the numerical solution of initial value problems. *Numerische Mathematik*, 42:271–290, 1983. 22

[86] Vidar Thomée. *Galerkin Finite Element Methods for Parabolic Problems*, volume 25. Springer Science & Business Media, 2007. 2

[87] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *Advances in Neural Information Processing Systems*, 30, 2017. 3

[88] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *International Conference on Learning Representations*, 2018. 9

[89] Sifan Wang and Paris Perdikaris. Long-time integration of parametric evolution equations with physics-informed DeepONets. *Journal of Computational Physics*, 475:111855, 2023. 30, 267, 300

[90] Gerhard Wanner and Ernst Hairer. *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems*, volume 375. Springer Berlin Heidelberg New York, 1996. 2

[91] John Wright and Yi Ma. *High-Dimensional Data Analysis with Low-Dimensional Models: Principles, Computation, and Applications*. Cambridge University Press, 2022. 2, 98

[92] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2020. 9

Part I: Structure Preserving Deep Learning

Dynamical Systems–Based Neural Networks

Elena Celledoni, Davide Murari, Brynjulf Owren, Carola-Bibiane Schönlieb, and Ferdia Sherry

SIAM Journal on Scientific Computing

Abstract. Neural networks have gained much interest because of their effectiveness in many applications. However, their mathematical properties are generally not well understood. If there is some underlying geometric structure inherent to the data or to the function to approximate, it is often desirable to take this into account in the design of the neural network. In this work, we start with a non-autonomous ODE and build neural networks using a suitable, structure-preserving, numerical time-discretisation. The structure of the neural network is then inferred from the properties of the ODE vector field. Besides injecting more structure into the network architectures, this modelling procedure allows a better theoretical understanding of their behaviour. We present two universal approximation results and demonstrate how to impose some particular properties on the neural networks. A particular focus is on 1-Lipschitz architectures including layers that are not 1-Lipschitz. These networks are expressive and robust against adversarial attacks, as shown for the CIFAR-10 and CIFAR-100 datasets.

2.1 Introduction

Neural networks have been employed to accurately solve many different tasks (see, e.g., [4, 12, 57, 38]). Indeed, because of their excellent approximation properties, ability to generalise to unseen data, and efficiency, neural networks are one of the preferred techniques for the approximation of functions in high-dimensional spaces.

In spite of this popularity, a substantial number of results and success stories in deep learning still rely on empirical evidence and more theoretical insight is needed. Recently, a number of scientific papers on the mathematical foundations of neural networks have appeared in the literature, [9, 74, 63, 64, 69, 36]. In a similar spirit, many authors consider the design of deep learning architectures taking into account specific mathematical properties such as stability, symmetries, or constraints on the Lipschitz constant [39, 34, 29, 66, 23, 30, 70, 37, 72, 76]. Even so, the imposition of structure on neural networks is often done in an ad hoc manner, making the resulting input to output mapping $F : \mathcal{X} \to \mathcal{Y}$ hard to analyse. In this paper, we describe a general and systematic way to impose desired mathematical structure on neural networks leading to an easier approach to their analysis.

There have been multiple attempts to formulate unifying principles for the design of neural networks. We hereby mention Geometric Deep Learning (see e.g. [13, 12]), Neural ODEs (see e.g. [22, 50, 60, 75]), the continuous-in-time interpretation of Recurrent Neural Networks (see e.g. [61, 21]) and of Residual Neural Networks (see e.g. [74, 47, 18, 62, 1]). In this work, we focus on Residual Neural Networks (ResNets) and build upon their continuous interpretation.

Neural networks are compositions of parametric maps, i.e. we can characterise

a neural network as a map $\mathcal{N} = f_{\theta_k} \circ \ldots \circ f_{\theta_1} : \mathbb{R}^n \to \mathbb{R}^m$, with network layers which are $f_{\theta_i} : \mathbb{R}^{n_i} \to \mathbb{R}^{n_{i+1}}$. For ResNets, the most important parametric maps are of the form

$$x \mapsto f_{\theta_i}(x) = x + h\Lambda\left(\theta_i, x\right). \tag{2.1.1}$$

The continuous-in-time interpretation of ResNets arises from the observation that if $n_i = n_{i+1}$, f_{θ_i} coincides with one $h-$step of the explicit Euler method applied to the non-autonomous ODE $\dot{x}(t) = \Lambda(\theta(t), x(t))$. In this work, we consider piecewise-autonomous systems, i.e. we focus on time-switching systems of the form

$$\dot{x}(t) = f_{s(t)}\left(x(t)\right), \; s : [0, T] \to \left\{1, \ldots, N\right\}, \; f_i \in \mathcal{F}, \tag{2.1.2}$$

with s being piecewise constant (see e.g. [60, 44]), and $\mathcal{F}$ a family of parametric vector functions. This simplification is not restrictive and can help analyse and design neural networks, as we will clarify throughout the paper.

This interpretation of ResNets gives the skeleton of our reasoning. Indeed, we replace the explicit Euler method in (2.1.1) with suitable (structure-preserving) numerical flows of appropriate vector fields. We call the groups of layers obtained with these numerical flows "dynamical blocks". The choice of the vector field is closely related to the structure to impose. For example, to derive symplectic neural networks, we would apply symplectic time integrators to Hamiltonian vector fields. This approach enables us to derive new structured networks systematically and collocate other existing architectures into a more general setting, making their analysis easier. For instance, Section 2.2 presents a strategy to study the approximation capabilities of some structured networks. Finally, we highlight the flexibility and the benefits of this framework in Section 2.3, where we show that to obtain expressive and robust neural networks, one can also include layers that are not 1-Lipschitz.

There are multiple situations where one could be interested in networks with some prescribed property. We report three of them here, where we refer to F as the function to approximate:

1. When F has some known characterising property, e.g. F is known to be symplectic; see Section 2.4.

2. When the data we process has a particular structure, e.g. vectors whose entries sum to one, as we present in Section 2.4.

3. When we can approximate F to sufficiently high accuracy with functions in $\mathcal{G}$, a space that is well suited to model the layers of a network. An example is using the space $\mathcal{G}$ of 1-Lipschitz functions to define a classifier robust to adversarial attacks; see Section 2.3.

Thus, there are various applications where having neural networks structured in a particular way is desirable. We will delve deeper into some of them in the following sections. To be precise, we remark that all the properties we focus on are preserved under composition, such as being 1-Lipschitz or symplectic.

The paper is structured in five sections. First, in Section 2.2, we investigate the universal approximation capabilities of some neural networks, thanks to vector field decompositions, splitting methods and an embedding of the dynamics into larger dimensional spaces. We then move, in Section 2.3, to a neural network that has the property of being 1-Lipschitz. After the mathematical derivation of the architecture, we present some numerical experiments on adversarial robustness for the CIFAR-10 and CIFAR-100 image classification problems. We devote a significant part of the experimental side of this paper to examples in the well-established field of adversarial robustness, but we furthermore provide examples of other desirable structural properties that can be imposed on neural networks using connections to dynamical systems. In Section 2.4, we introduce such neural networks with specific designs. This last section aims to present in a systematic way how one can impose certain properties on the architecture. We finally conclude the paper in Section 2.5, mentioning some promising directions for further work.

Before moving on, we now report a numerical experiment that motivates our investigation of structured neural networks. The results highlight how imposing a structure does not have to degrade the approximation's quality considerably. Furthermore, this experiment suggests that not all the constraining strategies perform similarly, as we also highlight in Section 2.3. Thus, a systematic process to impose structure is essential since it allows changing the architecture in a guided manner while preserving the property of interest.

2.1.1 Classification of points in the plane

We present a numerical experiment for the classification problem of the dataset in Figure 2.1b. We consider neural networks that are 1-Lipschitz, as in Section 2.3. We define the network layers alternatingly as contractive flow maps, whose vector fields belong to $\mathcal{F}_c = \{-A^T\Sigma(Ax+b) : A^TA = I\}$, and as flows of other Lipschitz continuous vector fields in $\mathcal{F} = \{\Sigma(Ax+b) : A^TA = I\}$, with $\Sigma(z) = [\sigma(z_1), \dots, \sigma(z_n)]$ and $\sigma(s) = \max\{s, s/2\}$[1]. In Section 2.3 we expand on the choice of this activation function σ, which is called LeakyReLU and was introduced in [48]. The time steps for each vector field are network parameters, together with the matrices A and vectors b. We constrain the time steps

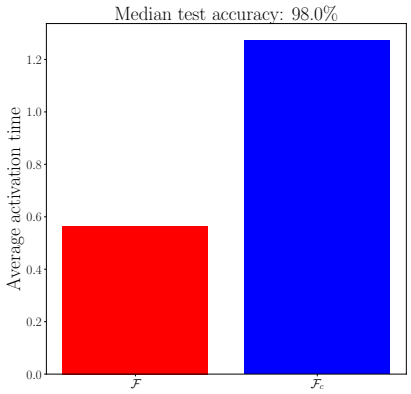
[1]To impose the weight orthogonality, we set $A = \mathrm{expm}(W - W^T)$ with expm being the matrix exponential and W a trainable matrix.

to get a 1-Lipschitz network, see Section 2.3. We report the results in Figure 2.1a and Table 2.1.

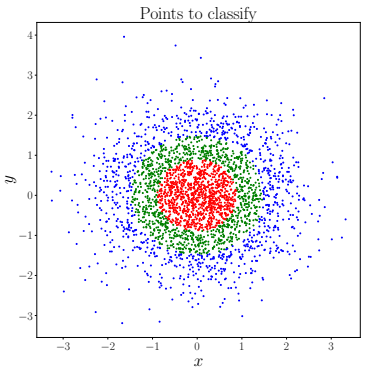
The average classification test accuracy and final integration time, in combination, get better by combining $\mathcal{F}_c$ with $\mathcal{F}$ instead of considering $\mathcal{F}_c$ alone. In particular, we see that the final integration time T with $\mathcal{F}_c \cup \mathcal{F}$ is the smallest without significantly compromising the accuracy. The parameter T quantifies how much the network layers transform the points. The larger the timestep, the further a layer is from the identity map; hence we can get a more natural and efficient solution by alternating the vector fields. In Section 2.2, we reinforce this empirical result, proving results about theoretical approximation guarantees. This renders the possibility of obtaining neural networks with prescribed properties without compromising their approximation capabilities.

Adopted family of vector fields	Median accuracy	Median of T
$\mathcal{F} \cup \mathcal{F}_c$	98.0%	1.84
$\mathcal{F}$	99.0%	7.53
$\mathcal{F}_c$	97.3%	19.82

Table 2.1: We perform 100 experiments alternating vector fields in $\mathcal{F}_c$ with those in $\mathcal{F}$, 100 using just vector fields in $\mathcal{F}_c$, and 100 with only those in $\mathcal{F}$. We work with networks with ten residual layers throughout the experiments. In the table, we report the median final time T and test accuracy for the three sets of experiments analysed



(a) Mean length of the time intervals along which the two dynamical regimes are active.



(b) Dataset studied for the classification problem

Figure 2.1: Results from the experiments alternating the vector fields of $\mathcal{F}$ and those of $\mathcal{F}_c$, together with the dataset of interest

2.2 Universal approximation properties

As introduced before, neural network layers can be modelled by discretising ordinary differential equations. In particular, this ODE-based approach can also be beneficial for imposing some structure on neural networks and providing a better theoretical understanding of their properties. In this section, we follow this principle and prove the universal approximation capabilities of two neural network architectures. Starting with the continuous-in-time interpretation of neural networks, many approaches are possible to prove such approximation properties, often based on merging the broad range of results from dynamical systems theory and numerical analysis. One can proceed, for example, in a constructive way as done in [60], where the authors investigate the dynamics of some neural networks and explicitly construct solutions to the problem of approximating a target function. Another possibility is to study the approximation capabilities of compositions of flow maps, as done in [42]. In this section, we focus on two solutions that, to the best of our knowledge, are new. The first result is based on a vector field decomposition, while the second is based on embedding vector fields into larger dimensional spaces.

The approximation results that we cover rely on approximating vector fields arbitrarily well. Consequently, this allows us to approximate their flow maps accurately. This is based on the fact that for a sufficiently regular vector field $X \in \mathrm{Lip}(\mathbb{R}^n, \mathbb{R}^n)$, if $\tilde{X} : \mathbb{R}^n \to \mathbb{R}^n$ is such that for every $x \in \mathbb{R}^n$

$$\left\| X(x) - \tilde{X}(x) \right\| < \varepsilon, \tag{2.2.1}$$

then also their flow maps are close to one another for finite time intervals. We formalise this reasoning in Proposition 2.1. In the proposition and throughout the paper, we denote with $\Phi_X^t(z)$ the time-t flow of the vector field X, applied to z.

Proposition 2.1. Let $X \in \mathrm{Lip}(\mathbb{R}^n, \mathbb{R}^n)$ and $\tilde{X} \in \mathrm{Lip}(\mathbb{R}^n, \mathbb{R}^n)$ be as in (2.2.1). Then $\|\Phi_X^t(x) - \Phi_{\tilde{X}}^t(x)\| \leq \varepsilon\, t \exp\big(\mathrm{Lip}(X)\, t\big)$, where $\mathrm{Lip}(X)$ is the Lipschitz constant of X.

Proof. We consider the integral equations of the ODEs $\dot{x}(t) = X(x(t))$ and $\dot{\tilde{x}}(t) = \tilde{X}(\tilde{x}(t))$ and study the difference of their solutions both with the same initial condition $x \in \mathbb{R}^n$

$$\begin{aligned}
\left\|\Phi_X^t(x)-\Phi_{\tilde{X}}^t(x)\right\| &= \left\|x+\int_0^t X\left(\Phi_X^s(x)\right)\mathrm{d}s - x - \int_0^t \tilde{X}\left(\Phi_{\tilde{X}}^s(x)\right)\mathrm{d}s\right\| \\
&\leq \int_0^t \left\|X\left(\Phi_X^s(x)\right)-\tilde{X}\left(\Phi_{\tilde{X}}^s(x)\right)\right\|\mathrm{d}s \\
&= \int_0^t \left\|X\left(\Phi_X^s(x)\right)-X\left(\Phi_{\tilde{X}}^s(x)\right)+X\left(\Phi_{\tilde{X}}^s(x)\right)-\tilde{X}\left(\Phi_{\tilde{X}}^s(x)\right)\right\|\mathrm{d}s \\
&\leq \mathrm{Lip}\left(X\right)\int_0^t \left\|\Phi_X^s(x)-\Phi_{\tilde{X}}^s(x)\right\|\mathrm{d}s + \varepsilon t.
\end{aligned}$$

Then we conclude that

$$\left\|\Phi_X^t(x)-\Phi_{\tilde{X}}^t(x)\right\| \leq \varepsilon t \exp\left(\mathrm{Lip}\left(X\right)t\right).$$

applying Gronwall's inequality. □

A particular consequence of this proposition is that if for every $\varepsilon > 0$ there is an $\tilde{X} \in \mathcal{F}$ making (2.2.1) true, then we can approximate the $T-$flow map of X arbitrarily well using elements of $\mathcal{F}$:

$$\left\|\Phi_X^T(x)-\Phi_{\tilde{X}}^T(x)\right\| \leq \varepsilon T \exp\left(\mathrm{Lip}\left(X\right)T\right) = c\varepsilon.$$

Because of this result, we now derive two approximation results for neural networks working at the level of modelling vector fields.

2.2.1 Approximation based on a vector field decomposition

We now aim to show that, for a particularly designed neural network, we can approximate arbitrarily well any continuous function in the L^p norm and any differentiable invertible function in the supremum norm on compact sets. We also mention how to extend this last result to generic continuous functions.

Theorem 2.1. *Let $F : \Omega \subset \mathbb{R}^n \to \mathbb{R}^n$ be a continuous function, with $\Omega \subset \mathbb{R}^n$ a compact set. Suppose that it can be approximated, with respect to some norm $\|\cdot\|$, by a composition of flow maps of $\mathcal{C}^1(\Omega, \mathbb{R}^n)$ vector fields, i.e. for any $\varepsilon > 0$, $\exists f_1, \ldots, f_k \in \mathcal{C}^1(\Omega, \mathbb{R}^n)$, such that*

$$\left\|F - \Phi_{f_k}^{h_k} \circ \ldots \circ \Phi_{f_1}^{h_1}\right\| < \varepsilon. \tag{2.2.2}$$

Then, F can be approximated arbitrarily well by composing flow maps of gradient and sphere-preserving vector fields, i.e., $\|F - \Phi_{\nabla U^k}^{h_k} \circ \Phi_{X_S^k}^{h_k} \circ \ldots \circ \Phi_{\nabla U^1}^{h_1} \circ \Phi_{X_S^1}^{h_1}\| < \varepsilon$.

By sphere-preserving vector field, we mean a vector field X_S having $z^T z$ as a first integral, i.e. such that $z^T X_S(z) = 0$ for any $z \in \mathbb{R}^n$.

The norm $\|\cdot\|$ in (2.2.2) can be any norm that is well defined for functions in $\mathcal{C}^1(\Omega, \mathbb{R}^n)$. Two typical choices in the literature are L^p norms and the supremum norm

$$\left\| F - \Phi_{f_k}^{h_k} \circ \ldots \circ \Phi_{f_1}^{h_1} \right\| := \sup_{x \in \Omega} \left\| F(x) - \Phi_{f_k}^{h_k} \circ \ldots \circ \Phi_{f_1}^{h_1}(x) \right\|. \tag{2.2.3}$$

Various works, like [11] and [42], have already proven the existence of vector fields $f_1, \ldots, f_k$ making (2.2.2) true when $\|\cdot\|$ is the L^p norm and F is a continuous function. Regarding the validity of hypothesis (2.2.2) with the norm defined in (2.2.3), we mention [68] where the authors have proven that if F is a smooth invertible map with smooth inverse, then the existence of $f_1, \ldots, f_k$ can be guaranteed.

Theorem 2.1 is a consequence of the Presnov decomposition of vector fields, introduced in [56], applied to the k vector fields $f_1, \cdots, f_k \in \mathcal{C}^1(\Omega, \mathbb{R}^n)$ appearing in (2.2.2). The Presnov decomposition is indeed a global decomposition of $\mathcal{C}^1(\mathbb{R}^n, \mathbb{R}^n)$ vector fields into the sum of a gradient and a sphere-preserving vector field. We now prove Theorem 2.1, and specialise it to the subfamilies of vector fields we implement to define neural networks.

Proof. The vector fields $f_1, \ldots, f_k$ are supposed to be continuously differentiable. Thus, they all admit a unique Presnov decomposition, i.e. they can be written as

$$f_i(x) = \nabla U^i(x) + X_S^i(x),$$

for a scalar function $U_i : \mathbb{R}^n \to \mathbb{R}$, with $U_i(0) = 0$, and a sphere-preserving vector field X_S^i. In general, the two vector fields $\nabla U^i(x)$ and $X_S^i(x)$ do not commute, i.e. the Jacobi-Lie bracket $[\nabla U^i, X_S^i]$ is not identically zero. However, because of the Baker-Campbell-Hausdorff formula (see e.g. [32, Section III.4.2]), as in splitting methods (see e.g. [51]) we can proceed with an approximation of the form

$$\Phi_{f_i}^h = \Phi_{\nabla U^i}^h \circ \Phi_{X_S^i}^h + \mathcal{O}\left(h^2\right).$$

This last equality is the local error of the Lie Trotter splitting: local order 2 and global order 1 under the hypothesis that guarantees convergence[2]. We recall that $\Phi_{f_i}^h = \Phi_{f_i}^{h/n} \circ \ldots \Phi_{f_i}^{h/n}$, where the flow maps are composed n times. Thus, up to choosing n large enough, we can approximate as accurately as desired $\Phi_{f_i}^h$ with the composition of flow maps of sphere-preserving and gradient vector fields. This concludes the proof. □

[2] We prove the convergence of the Lie-Trotter splitting formula for Lipschitz regular vector fields in Section 2.D. Such proof extends similarly to other splitting strategies.

Similar reasoning can be extended to other vector field decompositions, e.g. the Helmholtz decomposition, as long as $f_1, \dots, f_k$ admit such a decomposition. In Section 2.3, we adopt gradient vector fields whose flow maps expand and contract distances to obtain 1-Lipschitz neural networks. We now specialise Theorem 2.1 to the vector fields we use to model such neural networks.

Corollary 1. Consider the same assumptions of Theorem 2.1, and in particular the inequality (2.2.2). Then, we can approximate F arbitrarily well by composing flow maps of expansive, contractive and sphere-preserving vector fields.

We first remark that with an expansive vector field we mean a vector field X such that $\|\Phi_X^t(x) - \Phi_X^t(y)\| > \|x - y\|$ for any $t > 0$, while by contractive we mean that $\|\Phi_X^t(x) - \Phi_X^t(y)\| < \|x - y\|$. To prove the corollary, we rely on a classical universal approximation theorem with non-polynomial activation functions (see e.g. [55]). For clarity, we report it here.

Theorem 2.2 (Universal approximation, [55])**.** *Let $\Omega \subset \mathbb{R}^n$ be a compact set and $U \in \mathcal{C}^1(\mathbb{R}^n)$. Assume $\gamma \in \mathcal{C}^1(\mathbb{R})$ and γ is not a polynomial. Then for every $\varepsilon > 0$ there is*

$$\tilde{U}(x) = \boldsymbol{\alpha}^T \Gamma\left(Ax + b\right), \quad \Gamma(z) = \left[\gamma\left(z_1\right), \dots, \gamma\left(z_n\right)\right],$$

such that $\sup_{x\in\Omega} \left\|\tilde{U}(x) - U(x)\right\| < \varepsilon$, *and* $\sup_{x\in\Omega} \left\|\nabla\tilde{U}(x) - \nabla U(x)\right\| < \varepsilon$.

We now prove Corollary 1.

Proof. The proof follows the same reasoning as the one of Theorem 2.1. Indeed, we first decompose each of the $f_1, \dots, f_k$ of equation (2.2.2) via the Presnov decomposition as $f_i(x) = \nabla U^i(x) + X_S^i(x)$. Then, we approximate each of the U^i functions thanks to Theorem 2.2. To ease the notation, we focus on one of the f_i and denote it with f from now on in the proof.

Let $U : \mathbb{R}^n \to \mathbb{R}$ and X_S be so that $f(x) = \nabla U(x) + X_S(x)$. Choose then $\sigma(x) = \max\{ax, x\}$, $a \in (0, 1)$, and $\gamma(x) = \int_0^x \sigma(s)\,\mathrm{d}s$. Since γ is not a polynomial and it is continuously differentiable, Theorem 2.2 for any $\varepsilon > 0$ ensures the existence of a function

$$\tilde{U}(x) = \boldsymbol{\alpha}^T \Gamma\left(Ax + b\right),$$

that satisfies $\sup_{x\in\Omega} \|U(x) - \tilde{U}(x)\| < \varepsilon$ and $\sup_{x\in\Omega} \|\nabla U(x) - \nabla\tilde{U}(x)\| < \varepsilon$. We now split $\nabla\tilde{U}(x) = A^T \operatorname{diag}(\boldsymbol{\alpha})\Sigma(Ax + b)$ into a contractive and an expansive part, exploiting the two following properties of σ and γ:

1. σ is positively homogeneous, i.e. $\sigma(\lambda s) = \lambda\sigma(s)$ for $\lambda, s \in \mathbb{R}$, $\lambda \geq 0$,

2. γ is strongly convex.

We decompose $\boldsymbol{\alpha}$ as $\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-$, where $(\boldsymbol{\alpha}^+)_k = \max\{0, \alpha_k\}$, $(\boldsymbol{\alpha}^-)_k = -\min\{0, \alpha_k\}$ with $k = 1, \dots, n$. Because of the positive homogeneity, $\nabla \tilde{U}(x)$ can be rewritten as

$$\nabla \tilde{U}(x) = A_1^T \Sigma\left(A_1 x + b_1\right) - A_2^T \Sigma\left(A_2 x + b_2\right) = X_E(x) + X_C(x)$$

where

$$A_1 = \operatorname{diag}\left(\boldsymbol{\alpha}^+\right)^{\frac{1}{2}} A,\ A_2 = \operatorname{diag}\left(\boldsymbol{\alpha}^-\right)^{\frac{1}{2}} A,\ b_1 = \operatorname{diag}\left(\boldsymbol{\alpha}^+\right)^{\frac{1}{2}} b,\ b_2 = \operatorname{diag}\left(\boldsymbol{\alpha}^-\right)^{\frac{1}{2}} b.$$

Because of the strong convexity of γ, we have

$$\frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}t}\left\|z(t) - y(t)\right\|^2 = \left\langle X_E(z(t)) - X_E(y(t)), z(t) - y(t)\right\rangle > 0$$

with $z(t) = \Phi^t_{X_E}(z_0)$ and $y(t) = \Phi^t_{X_E}(y_0)$. This means that the flow of X_E is an expansive map. A similar reasoning shows that X_C has a contractive flow map. We can now conclude as in Theorem 2.1 since we have shown that every f_i in (2.2.2) can be approximated arbitrarily well as

$$f_i(x) \approx X_E^i(x) + X_C^i(x) + X_S^i(x).$$

□

As for the expansive and the contractive vector fields, to define neural networks based on Corollary 1 one needs to parameterise the vector field $X_S^i(z)$ that preserves spheres. Many possibilities are available, and we report a couple of them. The first is

$$\tilde{X}_S(z) = P(z) B^T \Sigma\left(Cz + d\right),\ B, C \in \mathbb{R}^{m\times n},\ d \in \mathbb{R}^m,\ P(z) = I_n - \frac{zz^T}{\|z\|^2},$$

where $P(z) : T_z\mathbb{R}^n \to T_z S^2_{\|z\|}$ is the orthogonal projection on the space $\langle z\rangle^\perp$ and $I_n \in \mathbb{R}^{n\times n}$ is the identity matrix. Another option is

$$\tilde{X}_S(z) = \Lambda\left(z, \theta\right) z$$

where $\Lambda(z,\theta) = A(z,\theta) - A(z,\theta)^T \in \mathbb{R}^{n\times n}$ with A being a strictly upper triangular matrix whose entries are modelled by $B\Sigma(Cx + b) \in \mathbb{R}^N$, $B \in \mathbb{R}^{N\times m}$, $C \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $N = \frac{n(n-1)}{2}$. These two possibilities allow us to approximate any sphere-preserving vector field arbitrarily because of classical universal approximation results like the one mentioned in Theorem 2.2. We prefer, for practical reasons, the second one in the experiments reported in Section 2.A.

We now summarise the results presented in the context of neural networks. Suppose that $\|F - \Phi^{h_k}_{f_k} \circ \ldots \circ \Phi^{h_1}_{f_1}\| < \varepsilon$ and that $f_i \approx \tilde{f}_i = X^i_C + X^i_E + \tilde{X}^i_S$ for $i = 1, \ldots, k$. In Theorem 2.1, we have worked with the exact flows of the vector fields. However, most of the times these are not available, and hence a numerical approximation is needed. This is exactly equivalent to applying a splitting numerical integrator (see e.g. [32, Chapter 2] or [51]) to approximate the h_i-flow map of $\tilde{f}_i$ (and hence also of f_i) and get

$$F(x) \approx \mathcal{N}(x) = \Psi^{h_k}_{X^k_C} \circ \Psi^{h_k}_{X^k_E} \circ \Psi^{h_k}_{X^k_S} \circ \ldots \circ \Psi^{h_1}_{X^1_C} \circ \Psi^{h_1}_{X^1_E} \circ \Psi^{h_1}_{X^1_S}(x). \tag{2.2.4}$$

Here we denote with Ψ^h_f a discrete approximation of the exact flow Φ^h_f and $\mathcal{N}$ is the neural network that approximates the target function F. Because of Corollary 1 and basic theory of numerical methods for ODEs, $\mathcal{N}$ hence can approximate arbitrarily well F in the norm $\|\cdot\|$.

We remark that the neural network $\mathcal{N}$ defined in (2.2.4) does not change the dimensionality of the input point, i.e. it is a map from $\mathbb{R}^n$ to itself. However, usually, ResNets allow for dimensionality changes thanks to linear lifting and projection layers. One can extend all the results presented in this section to the dimensionality changes, where instead of defining the whole network as the composition of discrete flow maps, just the "dynamical blocks" are characterised in that way, as represented in Figure 2.2. Consequently, one can extend

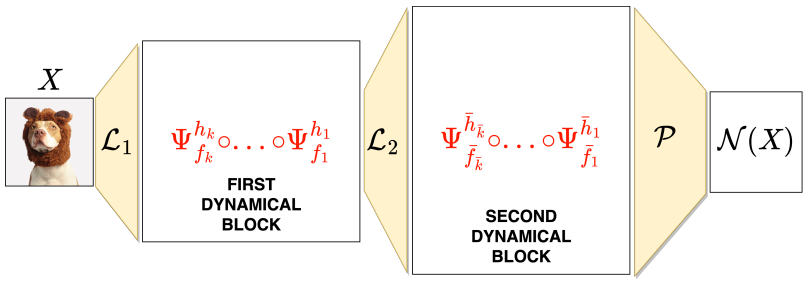


Figure 2.2: Representation of a ResNet made of two dynamical blocks, two lifting layers $\mathcal{L}_1$, $\mathcal{L}_2$ and a final projection layer $\mathcal{P}$.

the results presented in [77]. In particular, one can show that by composing flow maps of sphere-preserving and gradient vector fields, generic continuous functions can also be approximated in the sense of (2.2.3), as long as linear lifting and projection layers are allowed in the network.

In Section 2.A, we show some numerical experiments where some unknown dynamical systems and some target functions are approximated starting from the above results. We now introduce another way to get expressivity results starting from the continuous-in-time interpretation of neural networks.

2.2.2 Approximation based on Hamiltonian vector fields

Augmenting the dimensionality of the space where the dynamics is defined is a typical technique for designing deep neural networks, see Figure 2.2. Based on this idea, we now study the approximation properties of networks obtained by composing flow maps of Hamiltonian systems. For an introductory presentation of Hamiltonian systems, see [40].

We now show that for any function F for which hypothesis (2.2.2) holds, one can approximate F arbitrarily well, in the same function norm, by composing flow maps of Hamiltonian systems and linear maps. Consequently, symplectomorphisms like those defined by SympNets ([37]) can also be used approximate F arbitrarily well.

This result relies on the embedding of a vector field $f \in \mathcal{C}^1(\mathbb{R}^n, \mathbb{R}^n)$ into a Hamiltonian vector field on $\mathbb{R}^{2n}$. To do so, we first define the linear map $L : \mathbb{R}^n \to \mathbb{R}^{2n}$, as $z \mapsto L(z) = (z, 0)$. We then introduce $H_f(z, p) = p^T f(z)$, where p is the conjugate momentum of z. The gradient of such a function is

$$\nabla H_f(z,p) = \begin{bmatrix} \frac{\partial \left[p^T f(z) \right]}{\partial z} \\ f(z) \end{bmatrix}.$$

This implies that the Hamiltonian ODEs associated with H_f are

$$\begin{bmatrix} \dot{z} \\ \dot{p} \end{bmatrix} = X_{H_f}(z,p) = \mathbb{J}\nabla H_f(z,p) = \begin{bmatrix} f(z) \\ -\frac{\partial \left[p^T f(z) \right]}{\partial z} \end{bmatrix}.$$

Hence, we have $\Phi_f^h = P \circ \Phi_{X_{H_f}}^h \circ L$ where P is the projection on the first component $\mathbb{R}^{2n} \ni (z, p) \mapsto z \in \mathbb{R}^n$. This construction, with hypothesis (2.2.2), implies that

$$\left\| F - P \circ \Phi_{X_{H_{f_k}}}^{h_k} \circ L \circ P \circ \Phi_{X_{H_{f_{k-1}}}}^{h_{k-1}} \circ L \circ \ldots L \circ P \circ \Phi_{X_{H_{f_1}}}^{h_1} \circ L \right\| < \varepsilon.$$

One could be interested in such a lifting procedure and hence work with Hamiltonian systems because discretising their flow maps with symplectic methods might generate more stable networks or, as highlighted in [29], could prevent vanishing and exploding gradient problems.

2.3 Adversarial robustness and Lipschitz neural networks

In this section, we consider the problem of classifying points of a set $\mathcal{X} \subset \mathbb{R}^n$. More precisely, given a set $\mathcal{X} = \cup_{i=1}^C \mathcal{X}_i$ defined by the disjoint union of C

subsets $\mathcal{X}_1, \dots, \mathcal{X}_C$, we aim to approximate the function $\ell : \mathcal{X} \to \{1, \dots, C\}$ that assigns all the points of $\mathcal{X}$ to their correct class, i.e. $\ell(x) = i$ for all $x \in \mathcal{X}_i$ and all $i = 1, \dots, C$. Because of their approximation properties, one can often choose neural networks to solve classification problems, i.e. models that approximate the labelling function ℓ. On the other hand, there is increasing evidence that trained neural networks are sensitive to well-chosen input perturbations called adversarial attacks. The first work that points this out is [67] and, since then, numerous others (see e.g. [49, 17, 31]) have introduced both new ways to perturb the inputs (attacks) and to reduce the sensitivity of the networks (defences). We first formalise the problem of adversarial robustness from the mathematical point of view and then derive a network architecture with inherent stability properties.

Let $\mathcal{N} : \mathbb{R}^n \to \mathbb{R}^C$ be a neural network trained so that the true labelling map ℓ is well approximated by $\hat{\ell}(x) = \arg\max_{i=1,\dots,C} \mathcal{N}(x)_i$ for points $x \in \mathcal{X}$. Furthermore, let us assume

$$\|x - y\| \geq 2\varepsilon \quad \forall x, y \in \mathcal{X},\ \ell(x) \neq \ell(y)$$

for some norm $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}^+$ defined on the ambient space. With this setup, we say the network $\mathcal{N}$ is $\varepsilon-$robust if

$$\ell(x) = \hat{\ell}(x) = \hat{\ell}(x + \delta), \quad \forall \delta \in \mathbb{R}^n,\ \|\delta\| < \varepsilon.$$

In order to quantify the robustness of $\mathcal{N}$ we, first of all, consider its Lipschitz constant $\mathrm{Lip}(\mathcal{N})$, i.e. the smallest scalar value such that

$$\|\mathcal{N}(x) - \mathcal{N}(y)\|_2 \leq \mathrm{Lip}(\mathcal{N}) \|x - y\|, \quad \forall x, y \in \mathbb{R}^n,$$

where $\|\cdot\|_2 : \mathbb{R}^C \to \mathbb{R}^+$ is the ℓ^2 norm. We also need a way to quantify how certain the network predictions are. A measure of this certainty level is called margin in the literature (see e.g. [3, 7, 71]) and it is defined as

$$\mathcal{M}_{\mathcal{N}}(x) = \mathcal{N}(x)^T e_{\ell(x)} - \max_{j \neq \ell(x)} \mathcal{N}(x)^T e_j,$$

where e_i is the $i-$th vector of the canonical basis of $\mathbb{R}^C$. Combining these two quantities, in [71] the authors show that if the norm $\|\cdot\|$ considered for $\mathcal{X}$ is the ℓ^2 norm of the ambient space $\mathbb{R}^n$, then

$$\mathcal{M}_{\mathcal{N}}(x) \geq \sqrt{2}\varepsilon \,\mathrm{Lip}(\mathcal{N}) \implies \mathcal{M}_{\mathcal{N}}(x + \delta) \geq 0 \quad \forall \delta \in \mathbb{R}^n,\ \|\delta\| \leq \varepsilon. \tag{2.3.1}$$

Hence, for the points in $\mathcal{X}$ where (2.3.1) holds, the network is robust to perturbations with a magnitude not greater than ε. This result can be extended to

generic ℓ^p metrics, but, in this section, we focus on the case where $\|\cdot\|$ is the ℓ^2 metric of $\mathbb{R}^n$ and, from now on, we keep denoting it as $\|\cdot\|$.

Motivated by inequality (2.3.1), we present a strategy to constrain the Lipschitz constant of ResNets to the value of 1. Differently from [18, 76, 53], we impose such a property on the network without relying only on layers that are 1-Lipschitz. This strategy relies on the ODE-based approach that we are presenting and is motivated by the interest of getting networks that also have good expressivity capabilities. Indeed, we remark that in Section 2.2, we studied the approximation properties of networks similar to those we consider in this section. We conclude the section with extensive numerical experiments for the adversarial robustness with the CIFAR-10 and CIFAR-100 datasets to test the proposed network architectures.

2.3.1 Non-expansive dynamical blocks

Consider a scalar differentiable function $V : \mathbb{R}^n \to \mathbb{R}$ that is also strongly convex, i.e. it admits a $\mu > 0$ such that

$$\left\langle \nabla V(x) - \nabla V(y), x - y \right\rangle \geq \mu \|x - y\|^2, \tag{2.3.2}$$

see e.g. [35, Chapter 6]. We refer to a function V that is strongly convex with strong convexity constant μ as μ-strongly convex. This said, it follows that the dynamics defined by the ODE

$$\dot{x}(t) = -\nabla V\big(x(t)\big) = X\big(x(t)\big) \tag{2.3.3}$$

is contractive, since

$$\begin{aligned} \frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}t}\|x(t) - y(t)\|^2 &= -\left\langle x(t) - y(t), \nabla V\big(x(t)\big) - \nabla V\big(y(t)\big)\right\rangle \\ &\leq -\mu\|x(t) - y(t)\|^2 \\ \implies \|x(t) - y(t)\| &\leq e^{-\mu t}\|x_0 - y_0\| < \|x_0 - y_0\| \ \ \forall\, t \geq 0, \end{aligned} \tag{2.3.4}$$

where $x(t) = \Phi_X^t(x_0)$ and $y(t) = \Phi_X^t(y_0)$. A choice for V is $V(x) = \mathbf{1}^T\Gamma(Ax+b)$, where $\Gamma(x) = [\gamma(x_1), \ldots, \gamma(x_n)]$, $\gamma : \mathbb{R} \to \mathbb{R}$ is a strongly convex differentiable function, and $\mathbf{1} = [1, \ldots, 1] \in \mathbb{R}^n$. In this way, the network we generate by concatenating explicit Euler steps applied to such vector fields has layers of the type

$$x \mapsto \Psi_X^h(x) = x - hA^T\Sigma\big(Ax + b\big)$$

where $\Sigma(x) = [\sigma(x_1), \ldots, \sigma(x_n)]$ and $\sigma(s) = \gamma'(s)$.

If we discretise the ODE introduced above reproducing the non-expansive behaviour at a discrete level, as presented, for example, in [25, 18, 53], we get

that the numerical flow Ψ_X^h is non-expansive too. Consequently, we can obtain 1-Lipschitz neural networks composing these non-expansive discrete flow maps. A simple way to discretise (2.3.3) while preserving non-expansiveness is to use explicit-Euler steps with a small enough step size. Indeed, assuming $\mathrm{Lip}(\sigma) \leq 1$, a layer of the form

$$x \mapsto x - hA^T \Sigma\left(Ax + b\right), \quad h \leq \frac{2}{\|A\|^2}, \ \|A\| = \sup_{\substack{x \in \mathbb{R}^n \\ \|x\|=1}} \|Ax\|, \tag{2.3.5}$$

is guaranteed to be 1-Lipschitz. We remark that, as highlighted in [18, 53], it is not necessary to require strong convexity for γ in order to make Φ_X^t 1-Lipschitz. Indeed, it is enough to take γ convex. However, the strong convexity assumption allows us to include other layers that are not 1-Lipschitz thanks to inequality (2.3.4).

We now shortly present the reasoning behind this statement. Consider another ODE $\dot{x}(t) = Y(x(t))$ where Y is again a vector field on $\mathbb{R}^n$ and suppose that Y is L-Lipschitz. Then, we have that

$$\left\| \Phi_Y^{\bar{t}}\left(x_0\right) - \Phi_Y^{\bar{t}}\left(y_0\right) \right\| \leq \exp\left(L\bar{t}\right) \left\| x_0 - y_0 \right\|.$$

This implies that, given X as in (2.3.3), the map $\Phi_X^t \circ \Phi_Y^{\bar{t}} =: C_{\bar{t},t}$ satisfies

$$\left\| \Phi_X^t\left(\Phi_Y^{\bar{t}}\left(x_0\right)\right) - \Phi_X^t\left(\Phi_Y^{\bar{t}}\left(y_0\right)\right) \right\| \leq \exp\left(-\mu t + L\bar{t}\right) \left\| x_0 - y_0 \right\|, \tag{2.3.6}$$

so $C_{\bar{t},t}$ is Lipschitz continuous and will be 1-Lipschitz if $\exp\left(-\mu t + L\bar{t}\right) \leq 1$. This amounts to imposing $L\bar{t} \leq \mu t$ on the considered vector fields and time intervals on which corresponding flow maps are active. The map $C_{\bar{t},t}$ can be seen as the exact $(t + \bar{t})-$flow map of the switching system having a piecewise constant (in time) autonomous dynamics. In particular, such a system coincides with Y for the first time interval $[0, \bar{t})$ and with X for the time interval $[\bar{t}, \bar{t} + t)$.

We could choose Y as the gradient vector field of an L-smooth scalar potential. In other words, we ask for its gradient to be L-Lipschitz. An option is hence

$$Y(x) = A^T \Sigma\left(Ax + b\right), \ \|A\| \leq 1,$$

with σ that is $L-$Lipschitz. Thus, one possible way of building a dynamical block of layers that is 1-Lipschitz is through a consistent discretisation of the switching system

$$\dot{x}(t) = (-1)^{s(t)} A_{s(t)}^T \Sigma\left(A_{s(t)} x(t) + b_{s(t)}\right), \tag{2.3.7}$$

where $t \mapsto s(t)$ is a piecewise constant time-switching signal that, following (2.3.6), balances the expansive and contractive regimes. In (2.3.7), we are assuming that $\Sigma(z) = [\sigma(z_1), \dots, \sigma(z_n)]$ with σ which is 1−Lipschitz and $\gamma(s) = \int_0^s \sigma(t)dt$ is strongly convex. In the numerical experiment that we report at the end of the section, we design $s(t)$ that alternates between contractive and possibly non-contractive behaviours. In the following subsection, we present two possible approaches to discretise numerically the system in (2.3.7), mentioning how this extends to more general families of vector fields.

In this subsection, we have worked to obtain dynamical blocks that are non-expansive for the Euclidean metric. In Section 2.B, we show a way to extend this reasoning to more general metrics defined in the input space.

2.3.2 Non-expansive numerical discretisation

As presented in the introductory section, it is not enough to have a continuous model that satisfies some property of interest to get it at the network level. Indeed, discretising the solutions to such an ODE must also be done while preserving such a property. One approach that always works is to restrict the step sizes to be sufficiently small so that the behaviour of the discrete solution resembles the one of the exact solution. This strategy can lead to expensive network training because of the high number of time steps. On the other hand, this strategy allows weaker weight restrictions and better performances. We remark how this translates for the dynamical system introduced in (2.3.7), with $\sigma(x) = \max\{x, ax\}$. For that ODE, the one-sided Lipschitz constant of contractive layers is $\mu = a\lambda_{\min}(A^T A)$, $\lambda_{\min}$ being the smallest eigenvalue. Thus, if A is orthogonal, we get $\mu = a$. Under the same orthogonality assumption, the expansive layers in (2.3.7) have Lipschitz constant $L = 1$, and this allows to specialise the non-expansiveness condition (2.3.6) to $\bar{t} \leq at$. Thus, if we impose such a relationship and perform sufficiently small time steps, also the numerical solutions will be non-expansive.

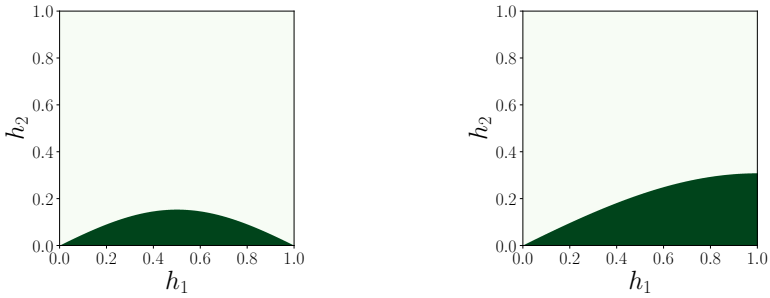
However, frequently smarter choices of discrete dynamical systems can lead to leaner architectures and faster training procedures. We focus on the explicit Euler method for this construction, although one can work with other numerical methods, like generic Runge-Kutta methods, as long as the conditions we derive are adjusted accordingly. We concentrate on two time steps applied to equation (2.3.7), but then the reasoning extends to every pair of composed discrete flows and to other families of vector fields. Let

$$\begin{aligned} \tilde{\Psi}^{h_1}(x) &= x - h_1 A_c^T \Sigma\left(A_c x + b_c\right) =: x - h_1 X\left(A_c, b_c, x\right) \\ \Psi^{h_2}(x) &= x + h_2 A_e^T \Sigma\left(A_e x + b_e\right) =: x + h_2 X\left(A_e, b_e, x\right). \end{aligned} \tag{2.3.8}$$

We remark that here the subscripts c and e stand for contractive and expansive respectively. The condition we want to have is that the map $F_h = \tilde{\Psi}^{h_1} \circ \Psi^{h_2}$ is 1-Lipschitz, or at least that this is true when A_c, A_e and Σ satisfy some well-specified properties. We first study the Lipschitz constant of both the discrete flow maps and then upper bound the one of F_h with their product. We take two points $x, y \in \mathbb{R}^n$, define $\delta X(A_c, b_c, x, y) = X(A_c, b_c, y) - X(A_c, b_c, x)$, and proceed as follows

$$\begin{aligned}
&\left\| \tilde{\Psi}^{h_1}(y) - \tilde{\Psi}^{h_1}(x) \right\|^2 \\
&= \|y-x\|^2 - 2h_1 \left\langle y-x, \delta X\left(A_c, b_c, x, y\right) \right\rangle + h_1^2 \left\| \delta X\left(A_c, b_c, x, y\right) \right\|^2 \\
&\leq \|y-x\|^2 - 2h_1 \lambda_{\min}\left(A_c^T A_c\right) a \|y-x\|^2 + h_1^2 \|A_c\|^4 \|y-x\|^2,
\end{aligned}$$

where the last inequality is because we consider $\sigma = \max\{ax, x\}$. In the experiments we present at the end of the section, we assume all the weight matrices to be orthogonal, hence $\lambda_{\min}(A_c^T A_c) = 1$. Multiple works support this weight constraint as a way to improve the generalisation capabilities, the robustness to adversarial attacks and the weight efficiency (see e.g. [72, 70]). We will detail how we get such a constraint on convolutional layers in the numerical experiments.



(a) Case $S = 1$.

(b) Case $S = 2$.

Figure 2.3: Representation of the non-expansiveness region (2.3.10) for the choice $a = 0.5$.

The orthogonality of A_c implies $\|A_c\| = 1$ and hence we get that a Lipschitz constant of $\tilde{\Psi}^{h_1}$ is $L_1 = \sqrt{1 - 2h_1 a + h_1^2}$. For the expansive flow map Ψ^{h_2}, we have

$$\begin{aligned}
\left\| \Psi^{h_2}(y) - \Psi^{h_1}(x) \right\| &\leq \|x-y\| + h_2 \operatorname{Lip}\left(A_e^T \Sigma\left(A_e z + b_e\right)\right) \|x-y\| \\
&\leq \left(1 + h_2\right) \|x-y\|
\end{aligned} \tag{2.3.9}$$

under the orthogonality assumption for A_e. The same result holds also if we just have $\|A_e\| \leq 1$. This leads to a region in the $(h_1, h_2)-$plane where $L_1 \cdot L_2 \leq 1$ that can be characterised as follows

$$\mathcal{R} = \left\{ (h_1, h_2) \in [0,1]^2 : (1+h_2)\sqrt{1-2h_1 a + h_1^2} \leq 1 \right\}. \tag{2.3.10}$$

This is represented in Figure 2.3a for the case $a = 0.5$, which is the one used also in the numerical experiment for adversarial robustness. Thus, we now have obtained a way to impose the 1-Lipschitz property on the network coming from the discretisation of the ODE (2.3.7). It is clear that the result presented here easily extends to different time-switching rules (i.e. a different choice of $s(t)$), as long as there is the possibility of balancing expansive vector fields with contractive ones. Furthermore, to enlarge the area in the $(h_1, h_2)-$plane where non-expansiveness can be obtained, one can decide to divide into sub-intervals the time intervals $[0, h_1]$ and $[0, h_2]$. Doing smaller steps, the allowed area increases. Indeed, instead of doing two single time steps of length h_1 and h_2, one can perform S time-steps all of step-length h_1/S or h_2/S. Thus, replacing $\bar{h}_1 = h_1/S$ and $\bar{h}_2 = h_2/S$ into (2.3.10) it is immediate to see that h_1 and h_2 are allowed to be larger than with the case $S = 1$. For example, if we again fix $a = 0.5$ and set $S = 2$, we get the area represented in Figure 2.3b. The choice of $a = 0.5$ and $S = 2$ is the one we adopt in the experiments reported in this section. We now conclude the section showing how the derived architecture allows us to improve the robustness against adversarial attacks for the problem of image classification.

2.3.3 Numerical experiments with adversarial robustness

We now apply the reasoning presented above to the problem of classifying images of the CIFAR-10 and CIFAR-100 datasets. The implementation is done with PyTorch and is available at the GitHub repository associated to the paper[3]. We work with convolutional neural networks, and with the activation function $\sigma(x) = \max\left\{x, \frac{x}{2}\right\}$, if not otherwise specified. We test multiple architectures and start by introducing the one coming directly from the derivation reported in the previous section. The residual layers of this network are dynamical blocks based on the discrete flow maps

$$\begin{aligned}
\tilde{\Psi}^{h_1}(x) &= x - h_1 A_c^T \Sigma\left(A_c x + b_c\right) =: x - h_1 X\left(A_c, b_c, x\right), \; A_c^T A_c = I \\
\Psi^{h_2}(x) &= x + h_2 A_e^T \Sigma\left(A_e x + b_e\right) =: x + h_2 X\left(A_e, b_e, x\right), \; A_e^T A_e = I \\
x &\mapsto \Psi^{h_2/2} \circ \tilde{\Psi}^{h_1/2} \circ \Psi^{h_2/2} \circ \tilde{\Psi}^{h_1/2}(x).
\end{aligned} \tag{2.3.11}$$

[3] https://github.com/davidemurari/StructuredNeuralNetworks

The orthogonality of the convolutional filters A_c and A_e is imposed through a regularisation strategy proposed in [72]. We comment more on this and alternative strategies later on in the description of the experimental setup. The step restriction is imposed after every training iteration, projecting back the pairs (h_1, h_2) in the region represented in Figure 2.3b if needed.

The strategy in equation (2.3.11) is defined as a "prescribed switching strategy" in the numerical experiments. It is applied both for the experiment on CIFAR-10 and CIFAR-100. To demonstrate the freedom one still has while using an alternation strategy to design the layers, we mention another switching strategy that we shall call "flexible". In this case, we have the following alternation

$$\begin{aligned}
\tilde{\Psi}^{h_1}(x) &= x - h_1 A_c^T \Sigma\left(A_c x + b_c\right) =: x - h_1 X\left(A_c, b_c, x\right),\ A_c^T A_c = I \\
\Psi^{h_2}(x) &= x + h_2 A^T \mathrm{ReLU}\left(Ax + b\right) =: x + h_2 X\left(A, b, x\right),\ A^T A = I \\
x &\mapsto \tilde{\Psi}^{h_1/2} \circ \tilde{\Psi}^{h_1/2} \circ \Psi^{h_2}(x).
\end{aligned} \tag{2.3.12}$$

Here the weight A is no longer with a subscript since the layer it defines has no guaranteed behaviour. The restriction on the step size h_1 is derived as in the previous section, while h_2 is either positive and satisfies a similar balance law as for the switching in equation (2.3.11), or it is allowed to be negative. For the dynamical block to be overall contractive, in case of a negative step h_2 we constrain it as in equation (2.3.5), i.e. we impose $|h_2| < 2$. In this way, there is not necessarily an alternation of expansive and contractive layers, but the optimiser is free to learn the switching strategy while still guaranteeing the non-expansivity of the dynamical block. For the experiments on CIFAR-100, we do not impose A to be orthogonal, but we normalise it since we have observed improved performance in this way.

We compare these alternation strategies with three other networks. The first uses only non-expansive flow maps defined in (2.3.5), with ReLU as an activation function. In the experiments, we denote this network as "non-expansive". We set the weight matrices to be orthogonal and constrain the learnable step sizes to be less than 2. We then report the results obtained with a more naïve way of constraining the Lipschitz constant of a ResNet layer. This approach relies on composing maps of the form

$$x \mapsto \frac{1}{2}\left(x + A^T \mathrm{ReLU}(Bx + b)\right),\ A^T A = B^T B = I,$$

as suggested in [43, Appendix D.1]. We noticed experimentally that this constraining strategy does not generate very expressive networks, which motivates the research for better 1-Lipschitz ResNet architectures, as proposed in this manuscript. As a general reference, we also include experiments based on a

standard ResNet that is not constrained in its weights and is composed of maps of the form

$$x \mapsto x + A^T \mathrm{ReLU}\left(Bx + b\right).$$

Before commenting on the results, we remark that the naïvely constrained network and the reference ResNet have double the parameters of the others based on dynamical systems. The rationale for this choice is to compare the networks at the level of the number of computations done per layer instead of based on the parameter count. For this reason, all the networks have the same number of layers. Furthermore, to get sufficiently accurate predictions on clean images, we did not constrain the last linear layer in all the experiments with the naïvely constrained network and all the CIFAR-100 experiments for the other networks. To jump-start the training of the networks on the CIFAR-100 dataset, we initialised all their layers, but the final projection layer, with the weights obtained on the CIFAR-10 dataset.

We implement architectures that take as inputs tensors of order three and shape $3 \times 32 \times 32$. The first dimensionality of the tensor increases to $32 - 64 - 128$ feature maps throughout the network via convolutional layers. For each fixed number of filters, we have four layers of the forms specified above. To be precise, convolutional layers replace the matrix-vector products.

The network architecture based on (2.3.11) gets close to 90% test accuracy, on the CIFAR-10 dataset, when trained with cross-entropy loss and without weight constraints. However, as presented at the beginning of this section, one could consider its Lipschitz constant and its margin at any input point of interest to get robustness guarantees for a network. For this reason, we now focus on constraining the Lipschitz constant of the architecture and introduce the loss function we adopt to promote higher margins. As in [3], we train the network architecture with the multi-class hinge loss function defined as

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq \ell(x_i)}}^{10} \max\left\{0, \mathrm{margin} - \left(\mathcal{N}\left(x_i\right)^T e_{\ell(x_i)} - \mathcal{N}\left(x_i\right)^T e_j\right)\right\},$$

where margin is a parameter to tune. We train all the networks with this loss function and with a stochastic gradient descent (SGD) optimiser. Having predictions with higher margins allows us to get more robust architectures if we fix the Lipschitz constant. Still, too high margins can lead to poor accuracy. In the experiments we test the three margin values 0.07, 0.15 and 0.3. For the networks based on dynamical systems, we report the results obtained constraining all the dynamical blocks and the final projection layer. However, we do not constrain the lifting layers. In this way, we can still control the full network's Lipschitz constant, just considering the norms of those lifting layers.

On the other hand, we leave some flexibility to the network, which can train better also when we increase the hinge-loss margin. We notice experimentally that the dynamical blocks usually get a small Lipschitz constant. Thus, even when we do not constrain all the layers, the network will still be 1-Lipschitz.

To get orthogonal convolutional filters, we apply the regularisation strategy proposed in [72]. This strategy is not the only one possible. Still, for this experiment, we preferred it to more involved ones since the main focus has been on the architecture, and the obtained results are satisfactory. Various works, e.g. [41, 54, 28], highlight how one can directly constrain the optimisation steps without having to project the weights on the right space or to add regularisation terms. We have not experimented with these kinds of strategies, and we leave them for future study. We also work with an orthogonal initialisation for the convolutional layers. The lifting layers of the networks based on dynamical systems are modelled as $x \mapsto \alpha W x$ for a convolutional filter W with $\|W\|_2 \leq 1$. To constrain the norm to 1, we add a projection step after the stochastic gradient descent (SGD) method updates the weights, i.e. we normalise the weights as $W \mapsto W / \max\{1, \|W\|_2\}$. Here, the $2-$norm of the convolutional filters is computed with the power method as described, for example, in [53]. Furthermore, we work with SGD having a learning rate scheduler that divides the learning rate after a fixed number of epochs. Finally, we generate the adversarial examples with the library "Foolbox" introduced in [58]. We focus on the ℓ^2-PGD attack and perform ten steps of it. We test different magnitudes of the adversarial perturbations.

To analyse the results of the experiments, we show how the accuracy of the networks changes as we increase the magnitude of the perturbations and the areas under these curves we get. The Area Under the Curve (AUC) metric is an informative quantity adopted to measure the adversarial robustness [8]. This metric is evaluated by computing the area below the piecewise linear curve obtained by plotting the robust accuracies as in Figure 2.4. A higher value indicates a better trade-off between accuracy and robustness. In Figure 2.4, we see that the robustness of the constrained neural networks based on dynamical systems improves compared to the baseline ResNet and the naïvely constrained one. Furthermore, we see that alternating expansive layers in the network improves the trade-off between clean accuracy and robustness than using a network with only non-expansive layers. To conclude, it is also evident from the experiments that if a more flexible alternating strategy is adopted, the results can improve because while the clean accuracy can increase, the robustness is kept unchanged or improved. In Figure 2.5, we plot the timesteps learned for the networks with a flexible alternation strategy. More precisely, given the 2 con-

secutive layers defined by

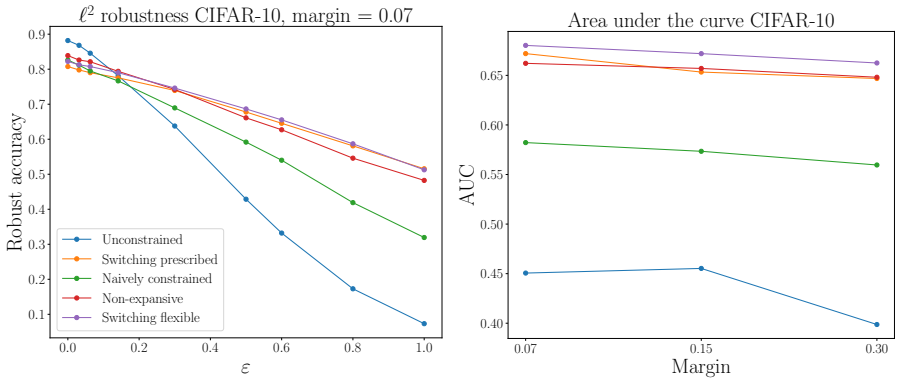
$$\tilde{\Psi}^{h_1}(x) = x - h_1 A_c^T \Sigma\left(A_c x + b_c\right) =: x - h_1 X\left(A_c, b_c, x\right),\ A_c^T A_c = I$$
$$\Psi^{h_2}(x) = x + h_2 A^T \mathrm{ReLU}\left(Ax + b\right) =: x + h_2 X\left(A, b, x\right),\ A^T A = I$$
$$x \mapsto \tilde{\Psi}^{h_1/2} \circ \tilde{\Psi}^{h_1/2} \circ \Psi^{h_2}(x),$$

we plot line segments that are as long as the steps h_1 and h_2, doing it for all the pairs of such layers. The step h_2 can also be negative, leading to non-expansive dynamics. This possibility is the main difference provided by the flexibility of the alternation approach. We notice that, especially for the CIFAR-10 dataset, a timestep is learned to be negative. This is not the case for CIFAR-100. For the case of margin = 0.15 and margin = 0.3, reported in Section 2.C, more steps are negative, especially for the CIFAR-10 experiments. On the other hand, there seems not to be a clear pattern in the step selection. These results suggest the optimiser exploits the freedom introduced due to the flexibility in the step selection and allows getting improved results in some instances. Section 2.C collects more details on how the timesteps are constrained. Furthermore, in Section 2.C, we also report the experiments for different margin values.

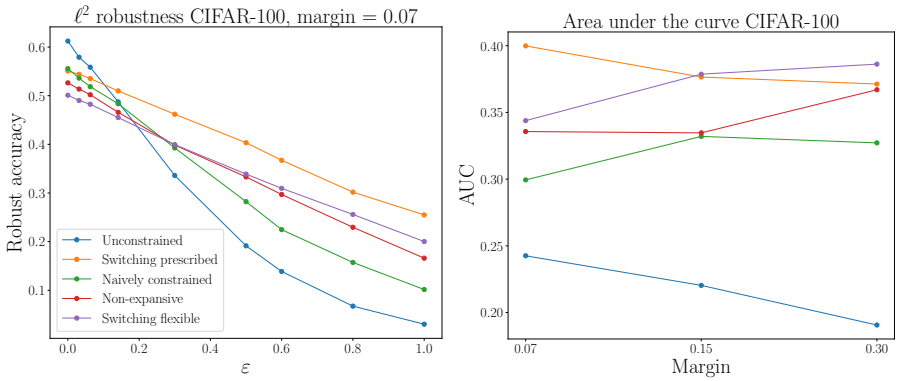
We remark that the results obtained with our proposed approach are not as good as those provided by the technique of adversarial training yet. On the other hand, our derivations lead to a more efficient training strategy that allows us to get networks with reduced sensitivity without the need to build adversarial examples in the training phase. Additionally, the results in Figure 2.4 show that the proposed constraining strategy allows considerable gains in the accuracy-robustness trade-off. The proposed framework is general enough to allow for possible improvements and reduce the performance difference with adversarial training. However, how to do so in practice still needs to be understood. We mention some possibilities in Section 2.5.

2.4 Imposition of other structure

Depending on the problem and the application where a neural network is adopted, the properties that the architecture should satisfy may be very different. We have seen in the previous section a strategy to impose Lipschitz constraints on the network to get some guarantees in terms of adversarial robustness. In that context, the property is of interest because it is possible to see that even when imposing it, we can get sufficiently accurate predictions. Moreover, this strategy allows controlling the network's sensitivity to input perturbations. As mentioned in the introduction, there are at least two other situations where structural constraints might be desirable. The first one is when one knows



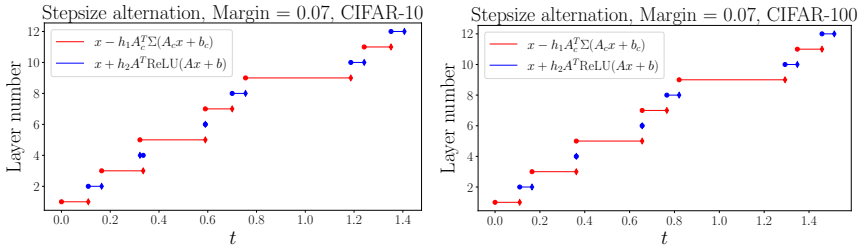
(a) Robustness plot **(b)** Area under the curve



(c) Robustness plot **(d)** Area under the curve

Figure 2.4: On the left: plots of the accuracy against the magnitude of the adversarial PGD perturbation of 1024 test clean images, comparing various perturbation magnitudes and the 5 networks introduced in the text. On the right: area under the curves to measure the actual robustness of the models. The legend is shared among the four plots, and for clarity, we omit it in the plots for the area under the curve.

that the function to approximate has some particular property. The second is when we know that the data we process induces some structure on the function we want to approximate. This section supports the claim that combining ODE models with suitable numerical integrators allows us to define structured networks. More precisely, we derive multiple architectures by putting together these two elements. Some of these have already been presented in other works, and others are new. The properties that we investigate are symplecticity, volume preservation and mass preservation. For the first two, we describe how to constrain the dynamical blocks. For the third, we propose how to structure also the linear lifting and projection layers. Moreover, for this latter example, we also report some numerical experiments. The purpose of the presented toy



(a) Learned step alternation for CIFAR-10 dataset.

(b) Learned step alternation for CIFAR-100 dataset.

Figure 2.5: Representation of the learned step sizes for the 12 layers characterising the networks giving the results reported in Figure 2.4. This is the case with margin = 0.07, and the other ones are reported in Section 2.C. The time instants corresponding to the beginning of the interval where a layer is active are denoted with dots, while the ending time instants with diamonds. This implies that if a diamond on a segment is on the left of a dot, the represented timestep is negative. The abscissa t corresponds to the sum of the timesteps characterising each layer.

example is to show that the architecture is computationally realisable and also effective.

2.4.1 Symplectic dynamical blocks

A function $F : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ is said to be symplectic if it satisfies the identity

$$\frac{\partial F(x)}{\partial x}^T \mathbb{J} \frac{\partial F(x)}{\partial x} = \mathbb{J} \quad \forall x \in \mathbb{R}^{2n}, \ \mathbb{J} = \begin{bmatrix} 0_n & I_n \\ -I_n & 0_n \end{bmatrix} \in \mathbb{R}^{2n \times 2n}$$

with $0_n, I_n \in \mathbb{R}^{n \times n}$ being the zero and the identity matrices. Symplectic maps are particularly important in classical mechanics, because the flow map Φ^t of a Hamiltonian system $\dot{x}(t) = \mathbb{J}\nabla H(x(t))$ is symplectic, see e.g. [40]. This fact implies that if one is interested in approximating such a flow map with a neural network, then structuring it to be symplectic might be desirable. In this direction, there are a considerable number of works (see e.g. [37, 23, 78, 14]). We mention in particular [37] where the authors construct layers of a network to ensure the symplectic property is satisfied. On the other hand, in [23] the authors consider a neural network as the Hamiltonian function of a system and approximate its flow map with a symplectic numerical integrator[4]. The simplest symplectic integrator is symplectic Euler, which applied to

$$H(q, p) = V(q) + K(p)$$

[4]We remark that a one-step numerical method Ψ^h is symplectic if and only if, when applied to any Hamiltonian system, it is a symplectic map.

computes updates as

$$q_{n+1} = q_n + h\partial_p K(p_n), \quad p_{n+1} = p_n - h\partial_q V(q_{n+1}).$$

We now focus on the gradient modules presented in [37], defined by alternating maps of the form

$$\mathcal{G}_1(q,p) = \begin{bmatrix} A^T \operatorname{diag}(\alpha)\,\Sigma(Ap+a) + q \\ p \end{bmatrix},$$

$$\mathcal{G}_2(q,p) = \begin{bmatrix} q \\ B^T \operatorname{diag}(\beta)\,\Sigma(Bq+b) + p \end{bmatrix}.$$

We notice that we can obtain the same map from a time-switching ODE model. We first introduce the time-dependent Hamiltonian of such a model, which is

$$H_{s(t)}(z) = \alpha_{s(t)}^T \Gamma(A_{s(t)} P_{s(t)} z + b_{s(t)}), \quad A_{s(t)} \in \mathbb{R}^{n\times n},\, b_{s(t)} \in \mathbb{R}^n, \tag{2.4.1}$$

with $s : [0,+\infty) \to \mathbb{R}^+$ being piecewise constant. We can suppose without loss of generality that $s(t) \in \{0,1,2,\dots,K\}$ and that $P_{s(t)}$ alternates between the two matrices $\Pi_1 = [I_n, 0_n] \in \mathbb{R}^{n\times 2n}$ and $\Pi_2 = [0_n, I_n] \in \mathbb{R}^{n\times 2n}$. Let now $\Gamma(z) = [\gamma(z_1),\dots,\gamma(z_n)]$, $\Sigma(z) = [\sigma(z_1),\dots,\sigma(z_n)]$, and $\gamma'(s) = \sigma(s)$. We then notice that the Hamiltonian vector field associated with $H_{s(t)}$ alternates between the following two vector fields

$$X_{H_1}(q,p) = \begin{bmatrix} A_1^T \operatorname{diag}(\alpha_1)\Sigma(A_1 p + b_1) & 0 \end{bmatrix}^T,$$

$$X_{H_2}(q,p) = \begin{bmatrix} 0 & -A_2^T \operatorname{diag}(\alpha_2)\,\Sigma(A_2 q + b_2) \end{bmatrix}^T.$$

We now conclude that if we compute the exact flow of $X_{H_{s(t)}}$ and we take $s(t)$ to be constant on every interval of length 1, we recover the gradient module in [37].

Similarly, all the network architectures presented in [23] and related works are based on defining a neural network $\mathcal{N}(q,p)$ that plays the role of the Hamiltonian function and then applying a symplectic integrator to the Hamiltonian system $\dot{z} = \mathbb{J}\nabla\mathcal{N}(z)$. The composition of discrete flow maps of the time-independent Hamiltonian gives a symplectic network with shared weights. On the other hand, if the Hamiltonian changes as in (2.4.1), one gets different weights for different layers and potentially a more expressive model as presented in [37].

2.4.2 Volume-preserving dynamical blocks

Suppose that one is interested in defining efficiently invertible and volume-preserving networks. In that case, an approach based on switching systems

and splitting methods can provide a flexible solution. Consider the switching system defined by $\dot{z}(t) = f_{s(t)}(z(t))$, $f_i \in \mathfrak{X}(\mathbb{R}^n)$, where for every value of $s(t)$, the vector field has a specific partitioning that makes it divergence-free. For example, if we have $n = 2m$,

$$f_{s(t)}(z) = \begin{bmatrix} u_{s(t)}\left(z[m:]\right) & v_{s(t)}\left(z[:m]\right) \end{bmatrix}^T$$

satisfies such a condition and its flow map will be volume-preserving. We can numerically integrate such a vector field while preserving this property. Indeed, we can apply a splitting method based on composing the exact flow maps of the two volume-preserving vector fields

$$f^1_{s(t)}(z) = \begin{bmatrix} u_{s(t)}\left(z[m:]\right) & 0 \end{bmatrix}^T, \quad f^2_{s(t)}(z) = \begin{bmatrix} 0 & v_{s(t)}\left(z[:m]\right) \end{bmatrix}^T.$$

This approach gives architectures that are close to the ones of RevNets (see e.g. [30]). The inverse of the network is efficient to compute in this case, and this translates into memory efficiency since one does not need to save intermediate activation values for the backpropagation. A particular class of these blocks can be obtained with second-order vector fields and, in particular, with second-order conservative vector fields (hence Hamiltonian):

$$\ddot{x}(t) = f_{s(t)}\left(x(t)\right), \text{ or } \ddot{x}(t) = -\nabla V_{s(t)}\left(x(t)\right),$$

where, for example,

$$V_{s(t)}(x) = \boldsymbol{\alpha}^T \Gamma\left(A_{s(t)}x + b_{s(t)}\right)$$

for some $\Gamma = [\gamma, \ldots, \gamma]$. With a similar strategy, one can also derive the volume-preserving neural networks presented in [6].

2.4.3 Mass-preserving neural networks

The final property we focus on is mass preservation. By mass preservation, we refer to the conservation of the sum of the components of a vector (see [10]). This property is typical of semi-discretisations of mass-preserving PDEs, models for chemical reactions, for population dynamics and ecology (see e.g. [33, 15, 59]). More explicitly, one could be interested in imposing such a structure if the goal is to approximate a function $F : \mathbb{R}^n \to \mathbb{R}^m$ that is known to satisfy $T_n x = \sum_{i=1}^n x_i = T_m F(x) = \sum_{i=1}^m F(x)_i$. A simple way to impose such property is by approximating the target function $F : \mathbb{R}^n \to \mathbb{R}^m$ as

$$F(x) \approx \frac{\sum_{i=1}^n x_i}{\sum_{j=1}^m \tilde{F}(x)_j} \tilde{F}(x)$$

where $\tilde{F}:\mathbb{R}^n \to \mathbb{R}^m$ is any sufficiently expressive neural network. However, this choice might lead to hard training procedures because of the denominator. Imposing this structure at the level of network layers is not so intuitive in general. Hence, we rely again on a suitable ODE formulation. A vector field $X \in \mathfrak{X}(\mathbb{R}^n)$ whose flow map preserves the sum of the components of the state vector is simply one having a linear first integral $g(x) = \mathbf{1}^T x = \sum_{i=1}^n x_i$. Thus, we can design vector fields of the form

$$\dot{y}(t) = \left(A(y) - A(y)^T\right)\mathbf{1},\ A:\mathbb{R}^n \to \mathbb{R}^{n\times n}, \tag{2.4.2}$$

and this property will be a natural consequence of the exact flow map. This mass conservation could also be extended to weighted-mass conservation, and we would just have to replace $\mathbf{1}$ with a vector of weights $\boldsymbol{\alpha}$. This extension does not, however, allow to change the dimensionality from one layer to the next one as easily. To model these vector fields, we can work with parametric functions like $\tilde{f}(x) = B^T\Sigma(Ax+b) \in \mathbb{R}^{n(n-1)/2}$ and use them to build the upper triangular matrix-valued function A in (2.4.2). As presented in [32, Chapter 4], it is also immediate to impose this property at a discrete level since every Runge-Kutta or multistep method preserves linear first integrals without time-step restrictions. Thus, a possible strategy to model mass-preserving neural networks is based on combining layers of the following types:

1. Lifting layers: $L:\mathbb{R}^k \to \mathbb{R}^{k+s}$, $L(x_1,\dots,x_k) = (x_1,\dots,x_k,0,0,\dots,0)$,
2. Projection layers:

$$P:\mathbb{R}^{k+s} \to \mathbb{R}^k,$$
$$P(x_1,\dots,x_k,x_{k+1},\dots,x_{k+s}) = (x_1+o,\dots,x_k+o),$$

   with $o = \sum_{i=1}^s x_{k+i}/s$,
3. Dynamical blocks: one-step explicit Euler discretisations of (2.4.2).

To test the neural network architecture, we focus on the approximation of the flow map of the SIR model

$$\dot{y} = \begin{bmatrix} -y_1y_2 & y_1y_2 - y_2 & y_2 \end{bmatrix}^T = X(y)^T. \tag{2.4.3}$$

This experiment relates to the research area of data-driven modelling, which has attracted a high amount of interest in recent years, especially through the tools provided by machine learning (see, e.g., [19, 26, 6, 23]). We model the neural network as discussed above. We approximate the $1-$flow map of (2.4.3)

working with pairs of the form $\{(x_i, y_i = \Psi^1_X(x_i))\}_{i=1,\dots,N}$[5]. In this context, we suppose it is not possible to integrate in time the system of ODEs because this is not available, and what is provided is just a set of observed trajectories. The plots in Figure 2.6 represent the first two components of the solution for the SIR model. All the line segments connect the components of the initial conditions with those of the time-1 updates. The considerable benefit of mass preservation as a constraint is that it allows interpretable outputs. Indeed, in this case, the components of y represent the percentages of three species in the total population, and the network we train still allows us to get this interpretation to be mass-preserving.

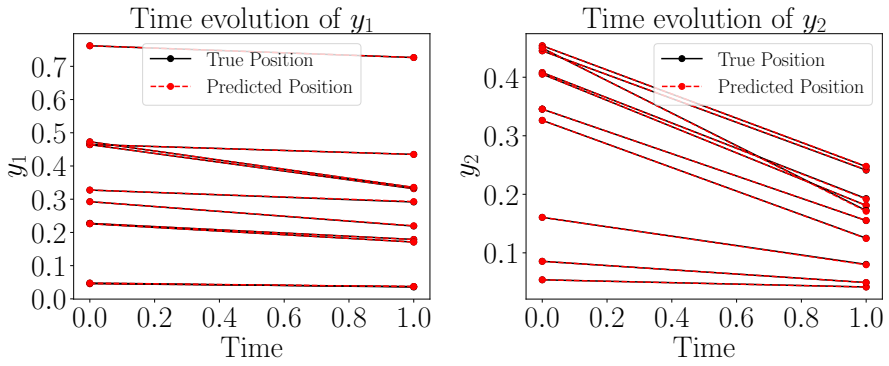


Figure 2.6: Plots of the approximation of the time 1−flow map of the SIR model (2.4.3) for 10 test initial conditions. We report the first two components of the solutions. Each point represents either an initial condition (at time 0), or a time-1 update.

2.5 Conclusion and future directions

In this work, we have introduced a framework to combine the design of ODE models with the choice of proper numerical methods to, in turn, obtain neural networks with prescribed properties. After introducing and motivating the approach, we proved two universal approximation theorems. The first one relates to sphere-preserving and gradient vector fields, while the second one involves Hamiltonian vector fields. We then obtained Lipschitz-constrained ResNets, focusing mainly on how to introduce layers that are not 1-Lipschitz. We then applied this construction to get neural networks with adversarial robustness guarantees. Finally, to show the framework's flexibility, we demonstrated how

[5]Here with $\Psi^1_X(x_i)$ we refer to an accurate approximation of the time-1 flow map of X applied to x_i

to design dynamical blocks that are symplectic, volume-preserving and mass-preserving.

The main application investigated in this manuscript is the one of adversarial robustness. Our experiments highlight that the robustness of neural networks to input perturbations can be improved using structured neural networks. However, the obtained results are competitive with other constraining strategies but not with adversarial training, which still provides state-of-the-art performance. We plan to optimise the proposed approach for higher clean accuracy, possibly by designing a better optimisation strategy or other more expressive families of expansive and contractive vector fields.

Throughout the manuscript, we have focused on explicit numerical methods as tools to generate neural network architectures. However, many geometric integrators are implicit (see e.g. [32]); thus, this remains a direction to pursue in further work so that the framework can be extended to other properties (see e.g. [5, 46]).

We have adopted the formalism of piecewise autonomous dynamical systems to design neural networks without heavily relying on the theory of time-switching systems. However, switching systems are a well-studied research area (see e.g. [44, 45, 2]), and it seems natural to study them further and their use to design neural network architectures.

Finally, we remark that imposing properties on neural networks is a promising strategy to make them more understandable, interpretable, and reliable. On the other hand, it is also clear that constraining the architecture can considerably decrease the network's expressivity in some cases. Thus, it remains to understand when it is preferable to replace hard constraints with soft constraints promoting such properties without imposing them by construction.

Acknowledgements The authors would like to thank the Isaac Newton Institute for Mathematical Sciences, Cambridge, for support and hospitality during the programmes *Mathematics of deep learning* and *Geometry, compatibility and structure preservation in computational differential equations* where work on this paper was significantly advanced. This work was supported by EPSRC grant no EP/R014604/1. EC and BO have received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 860124.

Appendix

2.A Some numerical experiments for data-driven modelling and regression

We now apply the theoretical background introduced in Section 2.2 to two approximation tasks. More precisely, we verify whether the introduced architectures are complicated to train in practice or whether they can achieve good performances. The two tasks of interest are the approximation of a continuous scalar function and the approximation of a $\mathcal{C}^1$ vector field starting from a set of training trajectories. We start with the approximation of $f(x) = x^2 + |x| + \sin x$, $x \in \mathbb{R}$, and $g(x, y) = \sqrt{x^2 + y^2}$. The goal here is to approximate them by composing flow maps of vector fields that are structured as

$$X_G(z) = A^T \operatorname{diag}(\boldsymbol{\alpha}) \Sigma \left(Az + b\right) = \nabla \left(\boldsymbol{\alpha}^T \Gamma \left(Az + b\right)\right),$$

$$X_S(z) = \left(A(z) - A(z)^T\right) z.$$

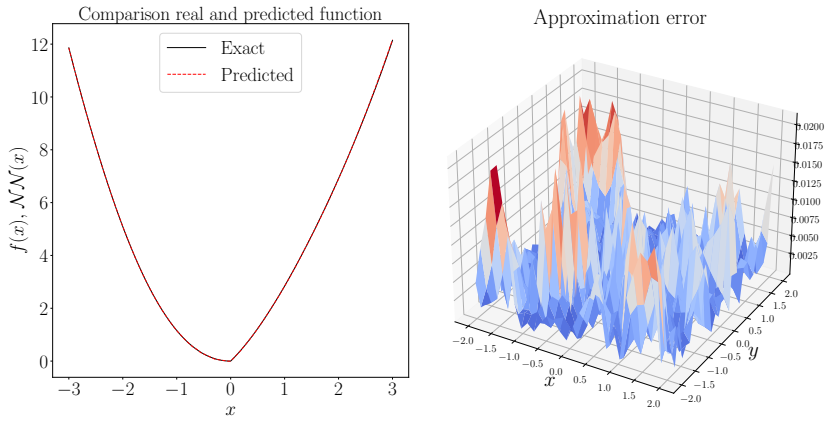
Following the result in Theorem 2.1, we compose the flow maps of X_G and X_S, maintaining the same time step for pairs of such flow maps. We report the results in Figure 2.A.1.

The second experiment that we report is the one of approximating a vector field $X \in \mathfrak{X}(\mathbb{R}^n)$ starting from a set of training pairs $\{(x_i, y_i)\}_{i=1,\dots,N}$, with $y_i = \Psi_X^h(x_i)$, for an accurate approximation Ψ_X^h of the time-h flow of X. We recall that the universal approximation result based on the Presnov decomposition allows approximating any vector field X as

$$X_\theta(z) = A^T \operatorname{diag}(\alpha) \Sigma \left(Az + b\right) + \left(B(z) - B(z)^T\right) z = X_G(z) + X_S(z),$$

as long as the weights are chosen correctly. In principle, calling Ψ^h any numerical method applied to X_θ and said $\hat{y}_i = \Psi^h(x_i)$, one could train the weights of X_θ so that they minimise

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \left\| y_i - \hat{y}_i \right\|^2.$$



(a) Approximation of $f(x)$ **(b)** Approximation of $g(x, y)$

Figure 2.A.1: Comparison between the trained networks and the true functions.

However, because of the properties of X_G and X_S, we choose to preserve them with Ψ^h and apply a splitting method. In other words, we apply to X_S and X_G two integrators Ψ^h_S and Ψ^h_G, then compose them to obtain $\Psi^h = \Psi^h_G \circ \Psi^h_S$. We choose Ψ^h_S to be an explicit method that preserves the conserved quantity $\|x\|^2$, while Ψ^h_G to be a discrete gradient method (see e.g. [27, 52]) so that it preserves the dissipative nature of X_G. As said before, this splitting strategy is not necessary in principle. However, we propose it as an alternative inspired by all the works on Hamiltonian neural networks (see e.g. [24, 20, 27]) where geometric integrators are often utilised. Furthermore, it would be interesting to understand if this or other splitting strategies give better approximation results or theoretical guarantees, but this goes beyond the scope of this work. We choose Ψ^h_S as a modified Euler-Heun method, following the derivation presented in [16], so that it is explicit and it also preserves $\|x\|^2$. For Ψ^h_G we use instead the Gonzalez discrete gradient method (see e.g. [52]). We remark that

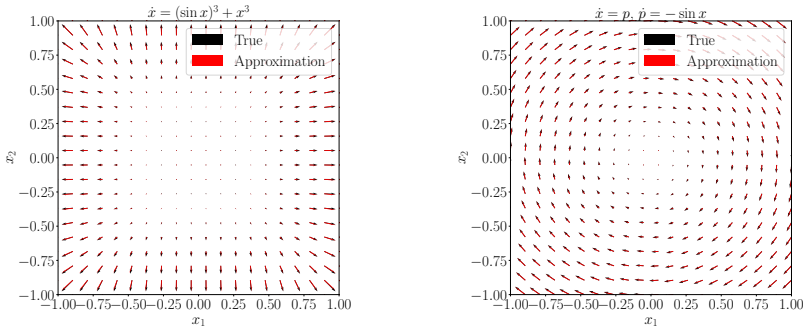


Figure 2.A.2: In the two plots, we compare the true and predicted vector fields.

discrete gradient methods applied to $\dot{x}(t) = -\nabla V(x(t))$ are of the form

$$x^{n+1} = x^n - h\overline{\nabla} V\left(x^n, x^{n+1}\right)$$

and they are, hence, implicit. However, since we have the trajectories available, i.e. the y_is are known, we do not have to solve a non-linear system of equations. Indeed, the problem of approximating X amounts to minimise the following cost function

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\left\| y_i - \left(\Psi_S^h(x_i) - h\overline{\nabla} V\left(\Psi_S^h(x_i), y_i\right)\right)\right\|^2.$$

In Figure 2.A.2, we report the results obtained for the following two vector fields

$$X_1(x) = \begin{bmatrix} (\sin x_1)^3 + x_1^3 \\ (\sin x_2)^3 + x_2^3 \\ (\sin x_3)^3 + x_3^3 \\ (\sin x_4)^3 + x_4^3 \end{bmatrix} \qquad X_2(x) = \begin{bmatrix} x_2 \\ -\sin x_1 \end{bmatrix}.$$

2.B Non-expansive networks with non-Euclidean metric

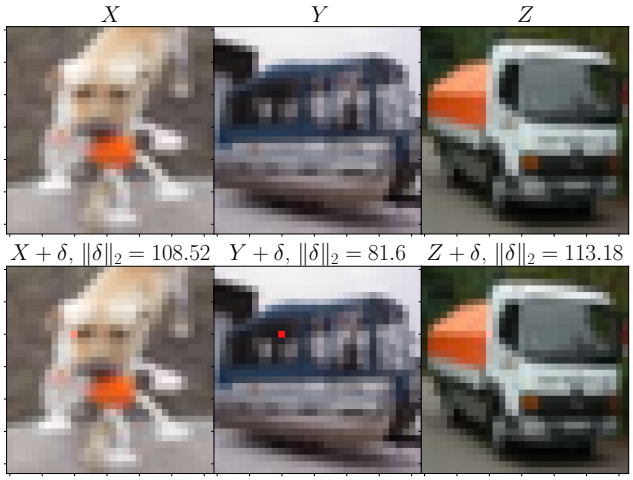


Figure 2.B.1: Three clean images from the CIFAR-10 dataset at the top and a random perturbation of the red channel in one pixel at the bottom. For humans, the images of the two rows clearly associate to the same object, even with this pixel perturbation.

In Section 2.3, we have introduced a way to generate networks that are non-expansive for the Euclidean metric on the input space. We now propose a strategy to generalise the reasoning. It is intuitive, and also evident from Figure 2.B.1, that ℓ^p norms of $\mathbb{R}^n$ are not the ones humans utilise to compare pictures. There have been many attempts to design similarity measures between images (see e.g. [73, 65]), but it is still not evident what should be the preferred choice. To make the model (2.3.7) introduced in Section 2.3 more general, we hence show how it can be made contractive for a more generic metric d defined by a symmetric and positive definite matrix $M \in \mathbb{R}^{n\times n}$. We introduce the notation $\langle v, w\rangle_M = v^T M w$ for any pair of vectors $v, w \in \mathbb{R}^n$. Let again $\Sigma(z) = [\sigma(z_1), \dots, \sigma(z_n)]$ where σ is an increasing scalar function. We focus on the autonomous dynamical system

$$\dot{z}(t) = -W^T \Sigma\big(MWz + b\big), \quad W \in \mathbb{R}^{n\times n}, b \in \mathbb{R}^n, \tag{2.B.1}$$

that is no longer a gradient vector field, but it has properties similar to the one studied above. We suppose M is constant and the same for the other involved weights. As discussed throughout the paper, how this reasoning extends to time-switching systems is quite natural and follows the procedure seen for M being the identity matrix, compare in particular Section 2.3. We now verify the contractivity of the ODE (2.B.1) with respect to the metric defined by M:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\frac{1}{2}\big\|z(t) - y(t)\big\|_M^2 &= \frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}t}\Big(\big(z(t) - y(t)\big)^T M\big(z(t) - y(t)\big)\Big)\\ &= -\Big\langle W^T\Sigma\big(MWz + b\big) - W^T\Sigma\big(MWy + b\big), z - y\Big\rangle_M\\ &= -\Big\langle \Sigma\big(MWz + b\big) - \Sigma\big(MWy + b\big), MWz - MWy\Big\rangle \le 0.\end{aligned}$$

This result implies that all the trajectories of (2.B.1) will converge to a reference trajectory if the convergence is measured using the metric defined by M. Notice that the scalar product in the last line is the canonical one of $\mathbb{R}^n$. Hence, if we have that γ is strongly convex, we can still combine these dynamics with expansive vector fields.

2.C Additional details on adversarial robustness

This section presents additional plots related to the experiment on adversarial robustness analysed in Section 2.3. We have studied the effect of using different margin values in the loss function adopted for the training of the five different neural networks. In Figure 2.C.1, we report the results obtained for two additional values of the margin parameter. The observations presented for

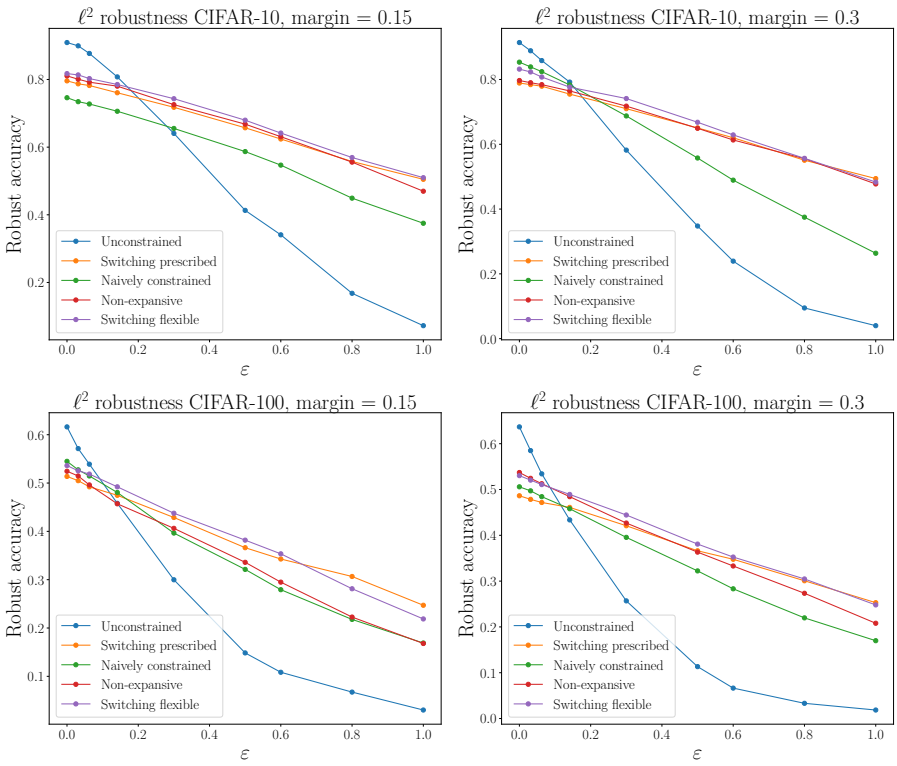


Figure 2.C.1: Behaviour of the test accuracy on 1024 images, under perturbations of different magnitude ε. On the left, we report the experiments with a margin value of 0.15, and on the right, with 0.3.

the case margin = 0.07 extend also to these ones, where the dynamically constrained networks still perform better. Moreover, the networks that allow for expansive layers still outperform those with all non-expansive layers.

We additionally provide the details on selecting the step sizes for the flexibly constrained neural networks. We recall that the flexible alternation strategy is defined by maps of the form

$$\begin{aligned}
\tilde{\Psi}^{h_1}(x) &= x - h_1 A_c^T \Sigma\left(A_c x + b_c\right) =: x - h_1 X\left(A_c, b_c, x\right), \; A_c^T A_c = I \\
\Psi^{h_2}(x) &= x + h_2 A^T \mathrm{ReLU}\left(Ax + b\right) =: x + h_2 X\left(A, b, x\right), \; A^T A = I \\
x &\mapsto \tilde{\Psi}^{h_1/2} \circ \tilde{\Psi}^{h_1/2} \circ \Psi^{h_2}(x) =: \Psi^h(x).
\end{aligned}$$

In order for the map Ψ^h to be non-expansive, we either need to have both $\tilde{\Psi}^{h_1/2}$ and Ψ^{h_2} to be 1−Lipschitz, or them to be 1−Lipschitz when composed together. For the former case, this is imposed in our implementation by the constraints $0.11 < h_1 < 1.9$ and $-1.9 < h_2 \leq 0$, since $\|A_c\| = \|A\| = 1$ and the

relevant condition is (2.3.5). In the case $h_2 > 0$, we need to impose the relation

$$(h_1, h_2) \in \mathcal{R} = \left\{ (h_1, h_2) \in \mathbb{R}^2 : (1 + h_2)\left(1 - h_1 a + h_1^2/4\right) \leq 1 \right\}.$$

Because of experimental reasons, we choose to impose it by clamping the h_1 timestep, with the PyTorch function `clamp`, so that $0.11 < h_1 < 1.9$. Then, we clamp h_2 so that the pair $(h_1, h_2) \in \mathcal{R}$, i.e. in the green area represented in Figure 2.C.2. All the constraints are imposed after each SGD step. In each

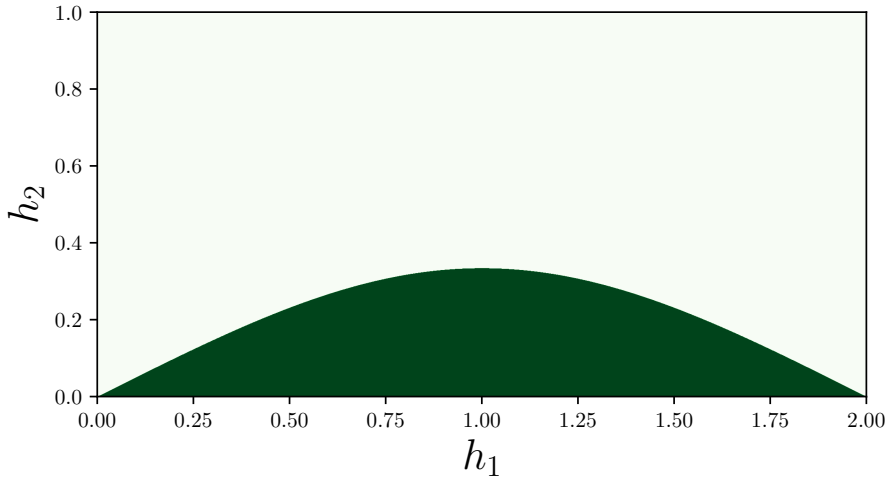
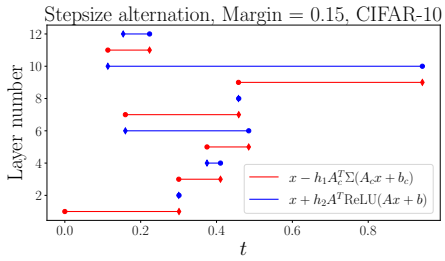


Figure 2.C.2: In green, we represent the contractivity area $\mathcal{R}$.

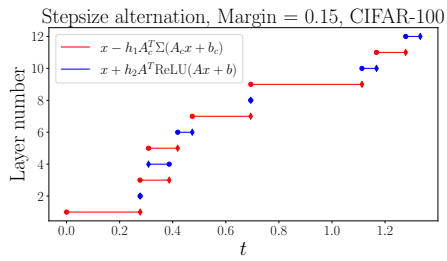
training iteration, the pair (h_1, h_2) is projected onto the green area. In Figure 2.C.3, we show the learned timesteps for the flexible training strategy that are not included in the manuscript's main text. We remark that Figure 2.C.3 reports many negative steps for the CIFAR-10 dataset, especially when the margin is 0.15, where almost all of them are. This pattern does not show up for CIFAR-100, which suggests that the increased complexity of this classification task leads to the need for more freedom in the network layers, given by expansive layers.

2.D Proof of the convergence of the splitting for Lipschitz fields

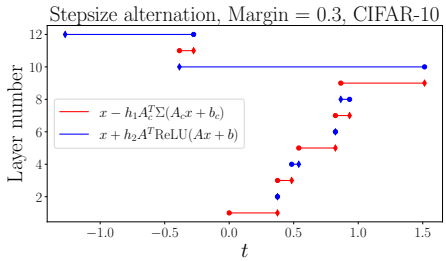
In this section, we prove the convergence of a Lie-Trotter splitting method applied to Lipschitz vector fields, as applied in the proofs of Section 2.2. This reasoning extends similarly to other splitting strategies, like Strang splitting.



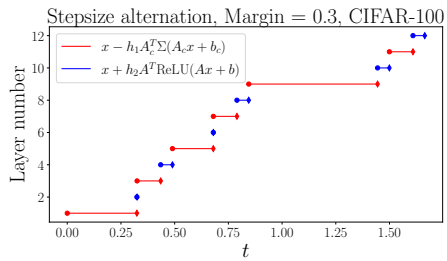
(a) Learned step alternation strategy for CIFAR-10 dataset.



(b) Learned step alternation strategy for CIFAR-100 dataset.



(c) Learned step alternation strategy for CIFAR-10 dataset.



(d) Learned step alternation strategy for CIFAR-100 dataset.

Figure 2.C.3: These figures report the learned step alternation strategies for the flexible regime, with margin values of 0.15 and 0.3

Proposition 2.2. Let $X \in \mathfrak{X}(\mathbb{R}^n)$ be a vector field that can be decomposed on the compact set $\Omega \subset \mathbb{R}^n$ as $X = f + g$ for two Lipschitz continuous vector fields $f, g \in \mathfrak{X}(\Omega)$. More precisely, let $L_f, L_g > 0$ be such that

$$\|f(x) - f(y)\| \le L_f \|x - y\|, \quad \|g(x) - g(y)\| \le L_g \|x - y\| \qquad \forall x, y \in \Omega.$$

The Lie-Trotter splitting method $\varphi^h := \Phi_g^h \circ \Phi_f^h$ is a first-order accurate approximation of the exact flow Φ_X^h.

Proof. The proof comes from applying Gronwall's inequality twice. We start considering the function

$$\gamma(t) := \left\| \Phi_g^t \circ \Phi_f^h(x_0) - \Phi_{f+g}^t(x_0) \right\|.$$

By the integral definition of the flow map, we have

$$\Phi_g^h \circ \Phi_f^h(x_0) = x_0 + \int_0^h f\left(\Phi_f^s(x_0)\right) \mathrm{d}s + \int_0^h g\left(\Phi_g^s \circ \Phi_f^h(x_0)\right) \mathrm{d}s$$

and

$$\Phi_{f+g}^h(x_0) = x_0 + \int_0^h f\left(\Phi_{f+g}^s(x_0)\right) \mathrm{d}s + \int_0^h g\left(\Phi_{f+g}^s(x_0)\right) \mathrm{d}s.$$

This means that

$$\begin{aligned}\gamma(h) &\le L_f \int_0^h \left\| \Phi_{f+g}^s(x_0) - \Phi_f^s(x_0) \right\| \mathrm{d}s + L_g \int_0^h \left\| \Phi_g^s \circ \Phi_f^h(x_0) - \Phi_{f+g}^s(x_0) \right\| \mathrm{d}s \\ &= \alpha(h) + \int_0^h \beta(s)\gamma(s)\,\mathrm{d}s\end{aligned}$$

where $\alpha(h) = L_f \int_0^h \|\Phi_{f+g}^s(x_0) - \Phi_f^s(x_0)\| \mathrm{d}s$ and $\beta(s) \equiv L_g$.

Since α is a non-decreasing function, we can apply Gronwall's integral inequality to get

$$\gamma(h) \le \alpha(h) \exp\left(\int_0^h \beta(s)\, ds \right) = \alpha(h) \exp\left(L_g h\right).$$

We need to bound the function $\alpha(h)$. We study the behaviour of

$$\lambda(s) := \left\| \Phi_{f+g}^s(x_0) - \Phi_f^s(x_0) \right\|$$

similarly to what was done above. Indeed we have

$$\begin{aligned}\Phi_{f+g}^s(x_0) - \Phi_f^s(x_0) = &\int_0^s f\left(\Phi_{f+g}^{s'}(x_0)\right) \mathrm{d}s' \\ &+ \int_0^s g\left(\Phi_{f+g}^{s'}(x_0)\right) \mathrm{d}s' - \int_0^s f\left(\Phi_f^{s'}(x_0)\right) \mathrm{d}s'\end{aligned}$$

and hence

$$\lambda(s) \leq L_f \int_0^s \lambda(s')\,\mathrm{d}s' + \int_0^s \left\| g\left(\Phi_{f+g}^{s'}(x_0)\right)\right\| \mathrm{d}s'.$$

Again by Gronwall's inequality, we can conclude

$$\lambda(s) \leq s \max_{x\in\Omega} \|g(x)\| \exp\left(L_f s\right).$$

This inequality allows finishing the proof since

$$\begin{aligned}
\gamma(h) &:= \left\| \Phi_g^h \circ \Phi_f^h(x_0) - \Phi_{f+g}^h(x_0)\right\| \\
&\leq \max_{x\in\Omega} \|g(x)\| \exp\left(L_g h\right) \int_0^h \exp\left(L_f h\right) s\,\mathrm{d}s \\
&= \frac{h^2}{2} \exp\left(\left(L_f + L_g\right) h\right) \max_{x\in\Omega} \|g(x)\| \\
&\leq \frac{h^2}{2} \exp\left(\mathrm{Lip}(X)\, h\right) \max_{x\in\Omega} \|g(x)\|.
\end{aligned}$$

Thus Lie-Trotter splitting is a first-order method for the vector field X. □

Bibliography

[1] Andrei Agrachev and Andrey Sarychev. Control on the Manifolds of Mappings with a View to the Deep Learning. *Journal of Dynamical and Control Systems*, pages 1–20, 2021. 48

[2] Tansu Alpcan and Tamer Basar. A stability result for switched systems with multiple equilibria. *Dynamics of Continuous, Discrete and Impulsive Systems Series A: Mathematical Analysis*, 17(4):949–958, 2010. 75

[3] Cem Anil, James Lucas, and Roger Grosse. Sorting Out Lipschitz Function Approximation. In *International Conference on Machine Learning*, pages 291–301. PMLR, 2019. 59, 66

[4] Simon Arridge, Peter Maass, Ozan Öktem, and Carola-Bibiane Schönlieb. Solving inverse problems using data-driven models. *Acta Numerica*, 28:1–174, 2019. 2, 48

[5] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep Equilibrium Models. *Advances in Neural Information Processing Systems*, 32, 2019. 75

[6] Jānis Bajārs. Locally-symplectic neural networks for learning volume-preserving dynamics. *arXiv preprint arXiv:2109.09151*, 2021. 72, 73

[7] Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. *Advances in Neural Information Processing Systems*, 30, 2017. 59

[8] Pouya Bashivan, Reza Bayat, Adam Ibrahim, Kartik Ahuja, Mojtaba Faramarzi, Touraj Laleh, Blake Richards, and Irina Rish. Adversarial Feature Desensitization. *Advances in Neural Information Processing Systems*, 34:10665–10677, 2021. 67

[9] Julius Berner, Philipp Grohs, Gitta Kutyniok, and Philipp Petersen. The Modern Mathematics of Deep Learning. *arXiv preprint arXiv:2105.04026*, 2021. 48

[10] Sergio Blanes, Arieh Iserles, and Shev Macnamara. Positivity-preserving methods for population models. *arXiv preprint arXiv:2102.08242*, 2021. 72

[11] Yann Brenier and Wilfrid Gangbo. l^p Approximation of maps by diffeomorphisms. *Calculus of Variations and Partial Differential Equations*, 16(2):147–164, 2003. 54

[12] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. *arXiv preprint arXiv:2104.13478*, 2021. 9, 48

[13] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017. 48

[14] Joshua William Burby, Qi Tang, and R Maulik. Fast neural Poincaré maps for toroidal magnetic fields. *Plasma Physics and Controlled Fusion*, 63(2):024001, 2020. 70

[15] Hans Burchard, Eric Deleersnijder, and Andreas Meister. A high-order conservative Patankar-type discretisation for stiff systems of production–destruction equations. *Applied Numerical Mathematics*, 47(1):1–30, 2003. 72

[16] Manuel Calvo, Domingo Hernández-Abreu, Juan I Montijano, and Luis Rández. On the preservation of invariants by explicit Runge–Kutta methods. *SIAM Journal on Scientific Computing*, 28(3):868–885, 2006. 77

[17] Nicholas Carlini and David Wagner. Towards Evaluating the Robustness of Neural Networks. In *2017 IEEE Symposium on Security and Privacy*, pages 39–57. IEEE, 2017. 59

[18] Elena Celledoni, Matthias J Ehrhardt, Christian Etmann, Robert I McLachlan, Brynjulf Owren, Carola-Bibiane Schönlieb, and Ferdia Sherry. Structure-preserving deep learning. *European Journal of Applied Mathematics*, 32(5):888–936, 2021. 48, 60, 61

[19] Elena Celledoni, Andrea Leone, Davide Murari, and Brynjulf Owren. Learning Hamiltonians of constrained mechanical systems. *Journal of Computational and Applied Mathematics*, 417:114608, 2023. 73

[20] Elena Celledoni, Andrea Leone, Davide Murari, and Brynjulf Owren. Learning Hamiltonians of constrained mechanical systems. *Journal of Computational and Applied Mathematics*, 417:114608, 2023. 77

[21] Bo Chang, Minmin Chen, Eldad Haber, and Ed H. Chi. AntisymmetricRNN: A Dynamical System View on Recurrent Neural Networks. In *International Conference on Learning Representations*, 2019. 48

[22] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural Ordinary Differential Equations. *Advances in Neural Information Processing Systems*, 31, 2018. 48

[23] Zhengdao Chen, Jianyu Zhang, Martin Arjovsky, and Léon Bottou. Symplectic Recurrent Neural Networks. In *International Conference on Learning Representations*, 2020. 48, 70, 71, 73, 149, 226, 227, 231, 235, 237

[24] Zhengdao Chen, Jianyu Zhang, Martin Arjovsky, and Léon Bottou. Symplectic Recurrent Neural Networks. In *International Conference on Learning Representations*, 2020. 77

[25] Germund Dahlquist. Generalized disks of contractivity for explicit and implicit Runge-Kutta methods. Technical report, CM-P00069451, 1979. 23, 60

[26] Sølve Eidnes. Order theory for discrete gradient methods. *BIT Numerical Mathematics*, pages 1–49, 2022. 14, 73

[27] Sølve Eidnes. Order theory for discrete gradient methods. *BIT Numerical Mathematics*, pages 1–49, 2022. 77

[28] Guilherme França, Alessandro Barp, Mark Girolami, and Michael I Jordan. Optimization on manifolds: A symplectic approach. *arXiv preprint arXiv:2107.11231*, 2021. 67

[29] Clara Lucía Galimberti, Luca Furieri, Liang Xu, and Giancarlo Ferrari-Trecate. Hamiltonian Deep Neural Networks Guaranteeing Nonvanishing Gradients by Design. *arXiv preprint arXiv:2105.13205*, 2021. 48, 58

[30] Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The Reversible Residual Network: Backpropagation Without Storing Activations. *Advances in Neural Information Processing Systems*, 30, 2017. 48, 72

[31] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. In Yoshua Bengio and Yann LeCun, editors, *ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 59

[32] Ernst Haier, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*. Springer, 2006. 54, 57, 73, 75

[33] Inga Hense and Aike Beckmann. The representation of cyanobacteria life cycle processes in aquatic ecosystem models. *Ecological Modelling*, 221(19):2330–2338, 2010. 72

[34] Johannes Hertrich, Sebastian Neumayer, and Gabriele Steidl. Convolutional proximal neural networks and Plug-and-Play algorithms. *Linear Algebra and its Applications*, 631:203–234, 2021. 48

[35] Jean-Baptiste Hiriart-Urruty and Claude Lemaréchal. *Convex Analysis and Minimization Algorithms I*, volume 305. Springer Science & Business Media, 2013. 60

[36] Kaixuan Huang, Yuqing Wang, Molei Tao, and Tuo Zhao. Why Do Deep Residual Networks Generalize Better than Deep Feedforward Networks?—A Neural Tangent Kernel Perspective. *Advances in neural information processing systems*, 33:2698–2709, 2020. 48

[37] Pengzhan Jin, Zhen Zhang, Aiqing Zhu, Yifa Tang, and George Em Karniadakis. SympNets: Intrinsic structure-preserving symplectic networks for identifying Hamiltonian systems. *Neural Networks*, 132:166–179, 2020. 48, 58, 70, 71, 146, 158

[38] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021. 48, 144

[39] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural computation*, 1(4):541–551, 1989. 3, 48

[40] Benedict Leimkuhler and Sebastian Reich. *Simulating Hamiltonian Dynamics*. Number 14 in Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2004. 11, 58, 70

[41] Benedict Leimkuhler, Tiffany J Vlaar, Timothée Pouchon, and Amos Storkey. Better Training using Weight-Constrained Stochastic Dynamics. In *Proceedings of the 38th International Conference on Machine Learning*, pages 6200–6211, 2021. 67

[42] Qianxiao Li, Ting Lin, and Zuowei Shen. Deep learning via dynamical systems: An approximation perspective. *Journal of the European Mathematical Society*, 2022. 25, 52, 54

[43] Qiyang Li, Saminul Haque, Cem Anil, James Lucas, Roger B Grosse, and Jörn-Henrik Jacobsen. Preventing Gradient Attenuation in Lipschitz Constrained Convolutional Networks. *Advances in neural information processing systems*, 32, 2019. 65

[44] Daniel Liberzon. *Switching in Systems and Control*, volume 190. Springer, 2003. 49, 75

[45] Daniel Liberzon and A Stephen Morse. Basic problems in stability and design of switched systems. *IEEE Control Systems Magazine*, 19(5):59–70, 1999. 75

[46] Andreas Look, Simona Doneva, Melih Kandemir, Rainer Gemulla, and Jan Peters. Differentiable Implicit Layers. *arXiv preprint arXiv:2010.07078*, 2020. 75

[47] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond Finite Layer Neural Networks: Bridging Deep Architectures and Numerical Differential Equations. In *International Conference on Machine Learning*, pages 3276–3285. PMLR, 2018. 48

[48] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *Proc. icml*, volume 30, page 3. Atlanta, Georgia, USA, 2013. 50

[49] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. In *International Conference on Learning Representations*, 2018. 59

[50] Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting Neural ODEs. *Advances in Neural Information Processing Systems*, 33:3952–3963, 2020. 48

[51] Robert I McLachlan and G Reinout W Quispel. Splitting methods. *Acta Numerica*, 11:341–434, 2002. 11, 54, 57

[52] Robert I McLachlan, G Reinout W Quispel, and Nicolas Robidoux. Geometric Integration Using Discrete Gradients. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 357(1754):1021–1045, 1999. 77

[53] Laurent Meunier, Blaise Delattre, Alexandre Araujo, and Alexandre Allauzen. Scalable Lipschitz Residual Networks with Convex Potential Flows. *arXiv preprint arXiv:2110.12690*, 2021. 60, 61, 67

[54] Mete Ozay and Takayuki Okatani. Optimization on Submanifolds of Convolution Kernels in CNNs. *arXiv preprint arXiv:1610.07008*, 2016. 67

[55] Allan Pinkus. Approximation theory of the MLP model in neural networks. *Acta Numerica*, 8:143–195, 1999. 3, 55, 155

[56] Eugene Presnov. Non-local decomposition of vector fields. *Chaos, Solitons & Fractals*, 14(5):759–764, 2002. 25, 54

[57] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019. 29, 48, 252, 253, 271

[58] Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox: A Python toolbox to benchmark the robustness of machine learning models. *arXiv preprint arXiv:1707.04131*, 2017. 67

[59] HH Robertson. The Solution of a Set of Reaction Rate Equations. *Numerical analysis: an introduction*, 178182, 1966. 72

[60] Domènec Ruiz-Balet and Enrique Zuazua. Neural ODE Control for Classification, Approximation, and Transport. *arXiv preprint arXiv:2104.05278*, 2021. 48, 49, 52

[61] T Konstantin Rusch and Siddhartha Mishra. UnICORNN: A recurrent model for learning very long time dependencies. In *International Conference on Machine Learning*, pages 9168–9178. PMLR, 2021. 48

[62] Lars Ruthotto and Eldad Haber. Deep Neural Networks Motivated by Partial Differential Equations. *Journal of Mathematical Imaging and Vision*, 62(3):352–364, 2020. 48, 145

[63] Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *CoRR*, abs/1312.6120, 2014. 48

[64] Ravid Shwartz-Ziv and Naftali Tishby. Opening the Black Box of Deep Neural Networks via Information. *ArXiv*, abs/1703.00810, 2017. 48

[65] Patrice Simard, Yann LeCun, and John Denker. Efficient Pattern Recognition Using a New Transformation Distance. *Advances in Neural Information Processing Systems*, 5, 1992. 79

[66] Bart Smets, Jim Portegies, Erik J Bekkers, and Remco Duits. Pde-based Group Equivariant Convolutional Neural Networks. *Journal of Mathematical Imaging and Vision*, pages 1–31, 2022. 48

[67] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, D. Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2014. 59

[68] Takeshi Teshima, Koichi Tojo, Masahiro Ikeda, Isao Ishikawa, and Kenta Oono. Universal Approximation Property of Neural Ordinary Differential Equations. *arXiv preprint arXiv:2012.02414*, 2020. 54

[69] Matthew Thorpe and Yves van Gennip. Deep Limits of Residual Neural Networks. *arXiv preprint arXiv:1810.11741*, 2018. 48

[70] Asher Trockman and J Zico Kolter. Orthogonalizing Convolutional Layers with the Cayley Transform. In *International Conference on Learning Representations*, 2021. 48, 63

[71] Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama. Lipschitz-Margin Training: Scalable Certification of Perturbation Invariance for Deep Neural Networks. *Advances in Neural Information Processing Systems*, 31, 2018. 59, 96, 123

[72] Jiayun Wang, Yubei Chen, Rudrasis Chakraborty, and Stella X Yu. Orthogonal Convolutional Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11505–11515, 2020. 48, 63, 65, 67

[73] Liwei Wang, Yan Zhang, and Jufu Feng. On the Euclidean distance of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1334–1339, 2005. 79

[74] E Weinan. A Proposal on Machine Learning via Dynamical Systems. *Communications in Mathematics and Statistics*, 1(5):1–11, 2017. 48, 147

[75] Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel Schoenholz, and Jeffrey Pennington. Dynamical Isometry and a Mean Field Theory of CNNs: How to Train 10,000-Layer Vanilla Convolutional Neural Networks. In *International Conference on Machine Learning*, pages 5393–5402. PMLR, 2018. 48

[76] Muhammad Zakwan, Liang Xu, and Giancarlo Ferrari-Trecate. On Robust Classification using Contractive Hamiltonian Neural ODEs. *arXiv preprint arXiv:2203.11805*, 2022. 48, 60, 123

[77] Han Zhang, Xi Gao, Jacob Unterman, and Tom Arodz. Approximation Capabilities of Neural ODEs and Invertible Residual Networks. In *International Conference on Machine Learning*, pages 11086–11095. PMLR, 2020. 57

[78] Aiqing Zhu, Pengzhan Jin, and Yifa Tang. Deep Hamiltonian networks based on symplectic integrators. *arXiv preprint arXiv:2004.13830*, 2020. 70, 146, 231, 243

Resilient Graph Neural Networks: A Coupled Dynamical Systems Approach

Moshe Eliasof, Davide Murari, Carola-Bibiane Schönlieb, and Ferdia Sherry

Submitted

Abstract. Graph Neural Networks (GNNs) have established themselves as a key component in addressing diverse graph-based tasks. Despite their notable successes, GNNs remain susceptible to input perturbations in the form of adversarial attacks. This paper introduces an innovative approach to fortify GNNs against adversarial perturbations through the lens of coupled dynamical systems. Our method introduces graph neural layers based on differential equations with contractive properties, which, as we show, improve the robustness of GNNs. A distinctive feature of the proposed approach is the simultaneous learned evolution of both the node features and the adjacency matrix, yielding an intrinsic enhancement of model robustness to perturbations in the input features and the connectivity of the graph. We mathematically derive the underpinnings of our novel architecture and provide theoretical insights to reason about its expected behavior. We demonstrate the efficacy of our method through numerous real-world benchmarks, reading on par, or improved performance compared to existing methods. Our source code will be published upon acceptance.

3.1 Introduction

In recent years, the emergence of Graph Neural Networks (GNNs) has revolutionized the field of graph machine learning, offering remarkable capabilities for modeling and analyzing complex graph-structured data. These networks have found applications in diverse domains and applications, from Network Analysis [15, 35] and recommendation systems, Bioinformatics [48, 13], Computer Vision [55], and more. However, the increasing prevalence of GNNs in critical decision-making processes has also exposed them to new challenges, particularly in terms of vulnerability to adversarial attacks.

In particular, it has been shown that one can design small adversarial perturbations of the input graph and its node features, that result in vastly different GNN predictions [57, 68]. Adversarial attacks received extensive attention in the context of Convolutional Neural Networks (CNNs) [23], but graph data has an added degree of freedom compared to data on a regular grid: the connectivity of the graph can be altered by adding or removing edges. Also, in natural settings, such as social network graphs, connectivity perturbations may be more realistically implementable by a potential adversary, rather than perturbations of the node features. This gives rise to hard discrete optimization problems, which necessitates the study of adversarial robustness specialized to graph data and GNNs [25].

In this paper, we propose a GNN architecture that jointly processes the adversarially attacked adjacency matrix and node features by a learnable neural dynamical system. Our approach extends the active research front that aims to design neural architectures that enjoy inherent properties and behavior, drawing inspiration from dynamical systems with similar properties [26, 58, 10,

9, 52, 17, 8]. This approach has also been used to improve the robustness of CNNs, see Appendix 3.F. Specifically, the flow map of the coupled dynamical system under consideration in this work draws inspiration from the theory of non-Euclidean contractive systems developed in [6] to offer an adversarially robust GNN. We name our method CSGNN, standing for Coupled dynamical Systems GNN. Notably, because adjacency matrices are not arbitrary matrices, their learnable neural dynamical system needs to be carefully crafted to ensure it is node-permutation equivariant, and that it yields a symmetric adjacency matrix. To the best of our knowledge, this is the first attempt at learning coupled dynamical systems that evolve both the node features and the adjacency matrix.

Main contributions. This paper offers the following advances in adversarial defense against poisoning attacks in GNNs: (i) A novel architecture, CSGNN, that jointly evolves the node features and the adjacency matrix to improve GNN robustness to input perturbations, (ii) A theoretical analysis of our CSGNN, addressing the relevance of the architecture based on the theory of contractive dynamical systems and, (iii) Improved performance on various graph adversarial defense benchmarks.

3.2 Related Work

Graph Neural Networks as Dynamical Systems. Drawing inspiration from dynamical systems models that admit desired behaviors, various GNN architectures have been proposed to take advantage of such characteristic properties. In particular, building on the interpretation of CNNs as discretizations of PDEs [10, 46], there have been multiple works that view GNN layers as time integration steps of learnable non-linear diffusion equations. Such an approach allowed exploiting this connection to understand and improve GNNs [9, 17], for example by including energy-conserving terms alongside diffusion [17, 45] or using reaction-diffusion equations [56, 12]. These approaches to designing GNNs have been shown to be of significant benefit when trying to overcome common issues such as over-smoothing [43, 7] and over-squashing [2]. Recently, it was shown that neural diffusion GNNs are robust to graph attacks [50].

We note that deep learning architectures are also often harnessed to numerically solve ODEs and PDEs or discover such dynamical systems from data [38, 3, 5]. However, in this paper, we focus on drawing links between GNNs and contractive dynamical systems to improve GNN robustness to adversar-

ial attacks. Additionally, we remark that the use of coupled dynamical systems updating both the adjacency matrix and the feature matrix jointly has also been used in [29]. However, this paper focuses on dynamic graphs for data-driven modeling purposes, while the focus of this paper it on adversarial robustness. Additionally, the parameterization of the adjacency matrix updates differs considerably between our CSGNN and [29] both in structure and the number of required parameters.

Adversarial Defense in Graph Neural Networks. Various adversarial attack algorithms have been designed for graph data, notably including nettack [67], which makes local changes to targeted nodes' features and connectivity, and metattack [68], which uses a meta-learning approach with a surrogate model, usually a graph convolutional network (GCN) [35], to generate a non-targeted global graph attack and, recently [11] proposed a novel method to create graph injection attacks.

In response to these developments, significant efforts were made to design methods that improve GNN robustness. The majority of these approaches focus on perturbations of the graph connectivity, as those are more likely and practical in social network graph datasets. Several of these methods preprocess the graph based on underlying assumptions or heuristics, for example, dropping edges where node features are not similar enough, under the assumption that the true, non-attacked, graph is homophilic [60]. Another approach, in [18], suggests truncating the singular value decomposition of the adjacency matrix, effectively eliminating its high-frequency components, based on the assumption that adversarial attacks add high-frequency perturbations to the true adjacency matrix. The aforementioned approaches are unsupervised and are typically added to existing GNN architectures while training them for a specific downstream task, such as node classification. Additionally, there are defenses that clean the attacked graph in a supervised manner, such as Pro-GNN [32], which solves a joint optimization problem for the GNN's learnable parameters, as well as for the adjacency matrix, with sparsity and low-rank regularization.

Besides methods for cleaning attacked adjacency matrices, there are also methods that aim to design robust GNN *architectures*. An example of this is given in [27], where the GCN architecture is modified to use a mid-pass filter, resulting in increased robustness. In this context, it is also natural to consider the use of Lipschitz constraints: given an upper bound on the Lipschitz constant of a classifier and a lower bound on its margin, we can issue robustness certificates [53]. This has been studied to some extent in the context of GNNs [31], although, in this case, and in contrast to our work, the Lipschitz continuity is

studied only with respect to the node features. In our work, we also consider the Lipschitz continuity with respect to the adjacency matrix. It is worth noting that the development of a defense mechanism should ideally be done in tandem with the development of an adaptive attack, although designing an appropriate adaptive attack is not generally a straightforward task [41]. In [41], a set of attacked graphs are provided as "unit tests", which have been generated using adaptive attacks for various defenses. Therefore, in our experiments, we consider both standard, long-standing benchmarks, as well as recently proposed attacks in [41]. In recent works like [50, 65], the robustness of GNNs was studied through a *node feature* dynamical systems point of view. However, in our CSGNN, we propose to learn a coupled dynamical system that involves *node features as well as the adjacency matrix*.

3.3 Preliminaries

Notations. Let $G = (V, E)$ be a graph with n nodes V and m edges E, also associated with the adjacency matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, such that $\mathbf{A}_{i,j} = 1$ if $(i, j) \in E$ and 0 otherwise, and let $\mathbf{f}_i \in \mathbb{R}^{c_{\text{in}}}$ be the input feature vector of the node $\mathbf{v}_i \in V$. In this paper, we focus on *poisoning* attacks, and we assume two types of possible attacks (perturbations) of the true data before training the GNN: (i) The features $\mathbf{f}_i$ are perturbed to $(\mathbf{f}_*)_i$, and, (ii) the adjacency matrix $\mathbf{A}$ of the graph is perturbed by adding or removing edges, inducing a perturbed adjacency matrix $\mathbf{A}_* \in \mathbb{R}^{n\times n}$. We denote by $G_* = (V_*, E_*)$ the attacked graph with the same vertices, i.e. $V = V_*$, by $\mathbf{A}_* \in \mathbb{R}^{n\times n}$ the perturbed adjacency matrix, and the perturbed node features are denoted by $(\mathbf{f}_*)_i$. We also denote by $\mathbf{F}, \mathbf{F}_* \in \mathbb{R}^{n\times c_{\text{in}}}$ the matrices collecting, as rows, the individual node features $\mathbf{f}_i, (\mathbf{f}_*)_i$.

Measuring graph attacks. To quantify the robustness of a GNN with respect to an adversarial attack, it is necessary to measure the impact of the attack. For node features, it is common to consider the Frobenius norm $\|\cdot\|_F$ to quantify the difference between the perturbed features $\mathbf{F}_*$ from the clean ones $\mathbf{F}$. However, the Frobenius norm is not a natural metric for adjacency attacks, see [4, 25] for example. Instead, it is common to measure the ℓ^0 distance between the true and attacked adjacency matrices, as follows:

$$\ell^0\left(\mathbf{A}, \mathbf{A}_*\right) = \left|\mathcal{I}\left(\mathbf{A}, \mathbf{A}_*\right)\right|, \tag{3.3.1}$$

where $\mathcal{I}(\mathbf{A}, \mathbf{A}_*) = \{i, j \in \{1, \dots, n\} : \mathbf{A}_{ij} \neq (\mathbf{A}_*)_{ij}\}$. For brevity, we refer to $\mathcal{I}(\mathbf{A}, \mathbf{A}_*)$ as $\mathcal{I}$, and by $|\mathcal{I}|$ we refer to the cardinality of $\mathcal{I}(\mathbf{A}, \mathbf{A}_*)$, as in (3.3.1).

The ℓ^0 distance measures how many entries of $\mathbf{A}$ need to be modified to obtain $\mathbf{A}_*$, and is typically used to measure budget constraints in studies of adversarial robustness of GNNs [25, 41]. Given that typical adjacency matrices consist of binary entries, it follows that:

$$\begin{aligned} \ell^0\left(\mathbf{A},\mathbf{A}_*\right) &= \left\|\operatorname{vec}\left(\mathbf{A}\right)-\operatorname{vec}\left(\mathbf{A}_*\right)\right\|_1 \\ &= \sum_{i,j=1}^{n}\left|\mathbf{A}_{ij}-\left(\mathbf{A}_*\right)_{ij}\right| = \ell^1\left(\mathbf{A},\mathbf{A}_*\right), \end{aligned} \tag{3.3.2}$$

where $\operatorname{vec}(\cdot)$ is the flattening operator, obtained by stacking the columns of $\mathbf{A}$. We refer to $\|\operatorname{vec}(\mathbf{A})-\operatorname{vec}(\mathbf{A}_*)\|_1$ as the vectorized ℓ^1 norm. That is, for binary matrices, the ℓ^0 and ℓ^1 norms coincide. However, using the ℓ^0 distance to implement constraints or regularization gives rise to computationally hard optimization problems because it is non-convex and non-smooth [59], and unfortunately, the equality in (3.3.2) is generally not true for arbitrary real-valued matrices. As shown in [59], for matrices with $\|\operatorname{vec}(\mathbf{A})\|_\infty \leq 1$, the vectorized ℓ^1 norm is the largest convex function bounded from above by the ℓ^0 norm, that is: $\|\operatorname{vec}(\mathbf{A})\|_1 \leq \|\operatorname{vec}(\mathbf{A})\|_0$. This property makes the usage of the ℓ^1 norm a common approximation of the ℓ^0 norm. Furthermore, it is also possible to relate the two norms, as follows:

$$\left\|\operatorname{vec}\left(\mathbf{A}\right)-\operatorname{vec}\left(\mathbf{A}_*\right)\right\|_1 \geq \left|\mathcal{I}\right|\cdot\min_{(i,j)\in\mathcal{I}}\left|\mathbf{A}_{ij}-\left(\mathbf{A}_*\right)_{ij}\right|, \tag{3.3.3}$$

i.e. the ℓ^1 norm can be lower bounded by the ℓ^0 norm, up to a multiplicative constant. Therefore, we can still use the ℓ^1 norm to measure the distance between arbitrary matrices as an approximation of the ℓ^0 norm. Throughout this paper, we denote the perturbed node features by $\mathbf{F}_* = \mathbf{F}+\delta\mathbf{F}$, and the perturbed adjacency matrix by $\mathbf{A}_* = \mathbf{A}+\delta\mathbf{A}$, where $\|\delta\mathbf{F}\|_F \leq \varepsilon_1$, and $\|\operatorname{vec}(\delta\mathbf{A})\|_1 \leq \varepsilon_2$.

In adversarial defense, the goal is to design a mechanism such that the output of the neural network is stable with respect to the perturbations $\delta\mathbf{F}$ and $\delta\mathbf{A}$. As discussed in Section 3.2, this goal is typically met either by modified architectures, training schemes, as well as their combinations. In Section 3.4, we present CSGNN - a defense mechanism based on a dynamical system perspective. This approach aims to reduce the sensitivity to input perturbations of the neural network and is based on the theory of contractive dynamical systems [6]. We will refer to a map as contractive with respect to a norm $\|\cdot\|$ if it is $1-$Lipschitz in such norm. Furthermore, we define contractive dynamical systems as those whose solution map is contractive with respect to the initial conditions. For completeness, in Appendix 3.A, we mathematically define and discuss contractive systems.

We start from the assumption that the best training accuracy on a given task corresponds to the clean inputs $(\mathbf{A},\mathbf{F})$. *The main idea of CSGNN is to jointly*

evolve the features $\mathbf{F}_*$ *and the adjacency matrix* $\mathbf{A}_*$*, so that even if their clean versions* $\mathbf{F}$ *and* $\mathbf{A}$ *are not known, the network would output a vector measurably similar to the one corresponding to* $(\mathbf{A}, \mathbf{F})$*, as we formulate in the following section.*

3.4 Method

3.4.1 Graph Neural Networks Inspired by Coupled Contractive Systems

We now present our CSGNN, focused on the task of robust node classification, where we wish to predict the class of each node in the graph, given attacked input data $(\mathbf{F}_*, \mathbf{A}_*)$. The goal is therefore to design and learn a map $\mathcal{D} : \mathbb{R}^{n\times c} \times \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times c} \times \mathbb{R}^{n\times n}$, that evolves the node features, as well as the adjacency matrix. To the best of our knowledge, this is the first attempt at learning a *coupled dynamical system* that considers both the node features and the adjacency matrix in the context of graph-node classification. As discussed in Section 3.2, utilizing dynamical systems–perspective in GNNs was shown to provide strong inductive bias and more predictable behavior. However, existing defense methods often limit this interpretation to *node feature updates*, while using heuristics to pre-process the adjacency matrix, if desired. Here, we advocate for the coupling of *node features and adjacency matrix* updates in a principled, data-driven, and dynamical system–based fashion.

We implement the map $\mathcal{D}$ as a composition of learnable dynamical systems inspired by contractivity theory, that simultaneously update $\mathbf{F}_*$ and $\mathbf{A}_*$. Specifically, we model $\mathcal{D}$ as an approximation of the solution, at the final time T, of the continuous dynamical system:

$$\begin{cases} \dot{F}(t) = X\left(t, F(t), A(t)\right) \in \mathbb{R}^{n\times c} \\ \dot{A}(t) = Y\left(t, A(t)\right) \in \mathbb{R}^{n\times n}, \\ \left(F(0), A(0)\right) = \left(\mathcal{K}\left(\mathbf{F}_*\right), \mathbf{A}_*\right), \end{cases} \tag{3.4.1}$$

where $\dot{F} = \mathrm{d}F/\mathrm{d}t$ denotes the first order derivative in time, and $\mathcal{K} : \mathbb{R}^{c_{\text{in}}} \to \mathbb{R}^{c}$ is a linear embedding layer. Similarly to [26, 17, 9], we assume both X and Y to be piecewise constant in time, i.e., that on a given time interval $[0, T]$, there is a partition $0 = \tau_0 < \tau_1 < \ldots < \tau_L = T$, $h_l = \tau_l - \tau_{l-1}$ for $l = 1, \ldots, L$, such that $X(t, \mathbf{F}, \mathbf{A}) = X_l(\mathbf{F}, \mathbf{A})$, $Y(t, \mathbf{A}) = Y_l(\mathbf{A})$, $\mathbf{F} \in \mathbb{R}^{n\times c}$, $\mathbf{A} \in \mathbb{R}^{n\times n}$, $t \in [\tau_{l-1}, \tau_l)$, for a pair of functions $X_l : \mathbb{R}^{n\times c} \times \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times c}$, $Y_l : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$. When referring to the approximation of $(F(\tau_l), A(\tau_l))$, we use $(\mathbf{F}^{(l)}, \mathbf{A}^{(l)})$ when we start

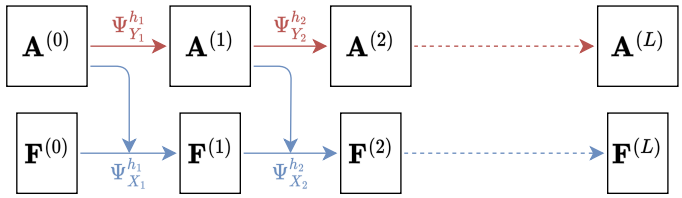


Figure 3.4.1: The coupled dynamical system $\mathcal{D}$ in CSGNN, as formulated in (3.4.2).

with the clean pair, and $(\mathbf{F}_*^{(l)}, \mathbf{A}_*^{(l)})$ with the perturbed one. To obtain a neural network, we consider the solution of (3.4.1) at time T, which is approximated using the explicit Euler method. More explicitly, we compose L explicit Euler layers each defined as $D_l((\mathbf{F},\mathbf{A})) := (\Psi_{X_l}^{h_l}(\mathbf{F},\mathbf{A}), \Psi_{Y_l}^{h_l}(\mathbf{A}))$, $l = 1,\dots,L$, where $\Psi_{X_l}^{h_l}(\mathbf{F},\mathbf{A}) := \mathbf{F} + h_l X_l(\mathbf{F},\mathbf{A})$, and $\Psi_{Y_l}^{h_l}(\mathbf{A}) := \mathbf{A} + h_l Y_l(\mathbf{A})$ are the explicit Euler steps for X_l and Y_l, respectively. The map $\mathcal{D}$ is then defined as the composition of L layers:

$$\mathcal{D} := D_L \circ \dots \circ D_1. \tag{3.4.2}$$

The coupled dynamical system encapsulated in $\mathcal{D}$ evolves both the hidden node features and the adjacency matrix for L layers. We denote the output of $\mathcal{D}$ by $(\mathbf{F}_*^{(L)}, \mathbf{A}_*^{(L)}) = \mathcal{D}((\mathcal{K}(\mathbf{F}_*), \mathbf{A}_*))$. To obtain node-wise predictions from the network to solve the downstream task, we feed the final GNN node features $\mathbf{F}_*^{(L)}$ to a classifier $\mathcal{P} : \mathbb{R}^c \to \mathbb{R}^{c_{\text{out}}}$, which is implemented by a linear layer.

To better explain the structure of CSGNN, we provide an illustration in Figure 3.4.1 and a detailed feed-forward description in Appendix 3.H.

In what follows, we describe how to characterize the functions $\Psi_{X_l}^{h_l}$ and $\Psi_{Y_l}^{h_l}$ from (3.4.2). First, in Section 3.4.2, we derive the node feature dynamical system governed by X. We show, that under mild conditions, contractivity can be achieved, allowing us to derive a bound on the influence of the attacked node features $\mathbf{F}_*$ on the GNN output. Second, in Section 3.4.3, we develop and propose a novel contractive dynamical system for the adjacency matrix, which is guided by Y.

Our motivation in designing such a coupled system stems from the nature of our considered adversarial settings. That is, we assume, that the adjacency matrix is perturbed. We note, that the adjacency matrix controls the propagation of node features. Therefore, leaving the input attacked adjacency unchanged may result in sub-par performance, as we show experimentally in Appendix 3.K. While some methods employ a pre-processing step of the attacked matrix $\mathbf{A}_*$ [18, 60], it has been shown that joint optimization of the node features and the adjacency matrix can lead to improved performance [32]. Therefore,

we develop and study novel, coupled dynamical systems that evolve both the node features and the adjacency matrix and are learned in a data-driven manner. This perspective allows to obtain favorable properties such as adjacency matrix contractivity, thereby reducing the sensitivity to adversarial adjacency matrix attacks.

3.4.2 Contractive Node Feature Dynamical System

We now describe the learnable functions $\Psi_{X_l}^{h_l}$, $l = 1, \dots, L$, that determine the node feature dynamics of our CSGNN. We build upon a diffusion-based GNN layer, [9, 17], that is known to be stable and, under certain assumptions, is contractive. More explicitly, our proposed $\Psi_{X_l}^{h_l}$ is characterized as follows:

$$\begin{aligned}
&\Psi_{X_l}^{h_l}\left(\mathbf{F}^{(l-1)}, \mathbf{A}^{(l-1)}\right) := \mathbf{F}^{(l)} \\
&:= \mathbf{F}^{(l-1)} + h_l X_l\left(\mathbf{F}^{(l-1)}, \mathbf{A}^{(l-1)}\right) \\
&= \mathbf{F}^{(l-1)} - h_l \left(\mathcal{G}^{(l-1)}\right)^\top \sigma\left(\mathcal{G}^{(l-1)} \mathbf{F}^{(l-1)} \mathbf{W}_l\right) \mathbf{W}_l^\top \tilde{\mathbf{K}}_l,
\end{aligned} \tag{3.4.3}$$

where $\mathcal{G}^{(l-1)} := \mathcal{G}(\mathbf{A}^{(l-1)})$, while $\mathbf{W}_l \in \mathbb{R}^{c \times c}$ and $\tilde{\mathbf{K}}_l = (\mathbf{K}_l + \mathbf{K}_l^\top)/2 \in \mathbb{R}^{c \times c}$ are learnable parameters which allows a gradient flow interpretation of our system, as in [22]. Also, as in [17], the map $\mathcal{G}(\mathbf{A}^{(l-1)}) : \mathbb{R}^{n \times c} \to \mathbb{R}^{n \times n \times c}$ is the gradient operator of $\mathbf{A}^{(l-1)}$ defined in Appendix 3.C, and we set $\sigma = \text{LeakyReLU}$.

Theorem 3.1 ((3.4.3) can induce stable node dynamics)**.** *Assume σ is a monotonically increasing 1-Lipschitz non-linear function. There are choices of $(\mathbf{W}_l, \mathbf{K}_l) \in \mathbb{R}^{c \times c} \times \mathbb{R}^{c \times c}$, for which the explicit Euler step in (3.4.3) is stable for a small enough $h_l > 0$, i.e. there is a convex energy $\mathcal{E}_{\mathbf{A}}$ for which*

$$\mathcal{E}_{\mathbf{A}}\left(\Psi_{X_l}^{h_l}\left(\mathbf{F}^{(l-1)}, \mathbf{A}\right)\right) \leq \mathcal{E}_{\mathbf{A}}\left(\mathbf{F}^{(l-1)}\right), \; l = 1, \dots, L. \tag{3.4.4}$$

Theorem 3.2 ((3.4.3) can induce contractive node dynamics)**.** *Assume σ is a monotonically increasing 1-Lipschitz non-linear function. There are choices of $(\mathbf{W}_l, \mathbf{K}_l) \in \mathbb{R}^{c \times c} \times \mathbb{R}^{c \times c}$, for which the explicit Euler step in (3.4.3) is contractive for a small enough $h_l > 0$, i.e.*

$$\left\| \Psi_{X_l}^{h_l}\left(\mathbf{F} + \delta\mathbf{F}, \mathbf{A}\right) - \Psi_{X_l}^{h_l}\left(\mathbf{F}, \mathbf{A}\right) \right\|_F \leq \left\| \delta\mathbf{F} \right\|_F, \tag{3.4.5}$$

for $\delta\mathbf{F} \in \mathbb{R}^{n \times c}$.

In Appendix 3.C we prove Theorems 3.1 and 3.2 for various parameterizations. One parameterization for which both theorems are satisfied corresponds

to $\mathbf{K}_l = I_c$. We have experimented with this configuration, learning only $\mathbf{W}_l \in \mathbb{R}^{c\times c}$. We found that this configuration improves several baseline results, showing the benefit of contractive node feature dynamics. We report those results in Appendix 3.K. We also found, following recent interpretations of dissipative and expanding GNNs [22], that choosing the parameterization as $\mathbf{W}_l = I_c$, and training $\mathbf{K}_l \in \mathbb{R}^{c\times c}$ leads to further improved results, as we show in our experiments in Section 3.5 and Appendix 3.K. We note that this parameterization admits stable dynamical systems, in the sense of Theorem 3.1, as discussed in Appendix 3.C.

3.4.3 Contractive Adjacency Matrix Dynamical System

As previously discussed, our CSGNN learns both node features and adjacency matrix dynamical systems to defend against adversarial attacks. We now elaborate on the latter, aiming to design and learn dynamical systems with explicit Euler approximation of the solution $\Psi_{Y_l}^{h_l}: \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$, $l = 1, \ldots, L$, such that:

$$\begin{aligned} &\left\| \mathrm{vec}\left(\Psi_{Y_l}^{h_l}\left(\mathbf{A}^{(l-1)}\right)\right) - \mathrm{vec}\left(\Psi_{Y_l}^{h_l}\left(\mathbf{A}_*^{(l-1)}\right)\right)\right\|_1 \\ &\leq \left\| \mathrm{vec}\left(\mathbf{A}^{(l-1)}\right) - \mathrm{vec}\left(\mathbf{A}_*^{(l-1)}\right)\right\|_1, \end{aligned} \tag{3.4.6}$$

where

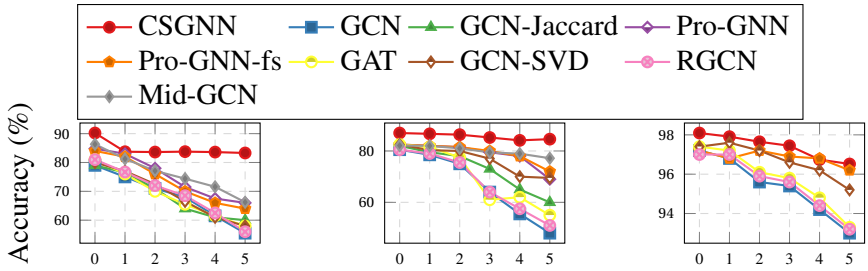
$$\Psi_{Y_l}^{h_l}\left(\mathbf{A}^{(l-1)}\right) = \mathbf{A}^{(l)} = \mathbf{A}^{(l-1)} + h_l Y_l\left(\mathbf{A}^{(l-1)}\right). \tag{3.4.7}$$

In other words, we wish to learn maps $\Psi_{Y_l}^{h_l}$ that *decrease* the vectorized ℓ^1 distance between the true and attacked adjacency matrices, thereby reducing the effect of the adjacency matrix attack.

Since we are concerned with adjacency matrices, we need to pay attention to the structure of the designed map Y_l. Specifically, we demand that (i) the learned map Y_l are *node-permutation-equivariant*. That is, relabelling (change of order) of the graph nodes should not influence the dynamical system $\Psi_{Y_l}^{h^l}$ output up to its order, and, (ii) if the input graph is symmetric, then the updated adjacency matrix $\mathbf{A}^{(l)}$ should remain symmetric. Formally, the requirement (i) demands that:

$$\Psi_{Y_l}^{h_l}\left(\mathbf{P}\mathbf{A}\mathbf{P}^\top\right) = \mathbf{P}\Psi_{Y_l}^{h_l}(\mathbf{A})\,\mathbf{P}^\top \tag{3.4.8}$$

should hold for every permutation matrix $\mathbf{P} \in \{0,1\}^{n\times n}$. The symmetry condition (ii) implies that we want $(\Psi_{Y_l}^{h_l}(\mathbf{A}))^\top = \Psi_{Y_l}^{h_l}(\mathbf{A})$ if $\mathbf{A}^\top = \mathbf{A}$. To this end, we adopt the derivations provided in [39, Appendix A], that show that in order to make the map $\Psi_{Y_l}^{h_l}$ permutation-equivariant and also symmetry preserving, we



(a) Cora **(b)** Citeseer **(c)** Polblogs

Figure 3.4.2: Node classification accuracy (%) under nettack. The horizontal axis describes the number of perturbations per node.

can set $Y_l(\mathbf{A}) = \sigma(M(\mathbf{A}))$ in (3.4.7), where $\sigma : \mathbb{R} \to \mathbb{R}$ is any non-linear activation function applied componentwise, and $M : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$ is a suitably designed linear map depending on a learnable vector $\mathbf{k} = (k_1, \cdots, k_9) \in \mathbb{R}^9$ and defined in Appendix 3.B. We now provide a theorem that validates the contractivity of the proposed adjacency matrix dynamical system, with its proof in Appendix 3.D.

Theorem 3.3 ((3.4.7) can define contractive adjacency dynamics)**.** *Let $\alpha \leq 0$, $\sigma : \mathbb{R} \to \mathbb{R}$ be a Lipschitz continuous function, with $\sigma'(s) \in [0, 1]$ almost everywhere. If $0 \leq h_l \leq 2/(2\sum_{i=2}^{9} |k_i| - \alpha)$, then the explicit Euler step*

$$\Psi_{Y_l}^{h_l}\left(\mathbf{A}^{(l-1)}\right) := \mathbf{A}^{(l-1)} + h_l \sigma\left(M\left(\mathbf{A}^{(l-1)}\right)\right), \tag{3.4.9}$$

where M is as in (3.B.1) and $k_1 = \left(\alpha - \sum_{i=2}^{9} |k_i|\right)$, is contractive in the vectorized ℓ^1 norm.

In our experiments, $\alpha \leq 0$ is a non-positive hyperparameter of the network.

In this Section, we presented the contractive node and adjacency updates in our CSGNN that allow for reduced sensitivity to adversarial perturbed inputs in a learnable fashion with respect to the downstream performance of the task at hand, which is node classification in this paper. Notably, it is important to distinguish between extreme cases of contractivity, such as in the case of multi-layer perceptron (MLP), which holds no sensitivity to the adjacency matrix by design, and a network with *learnable* sensitivity through a contractive behavior, as presented in our CSGNN.

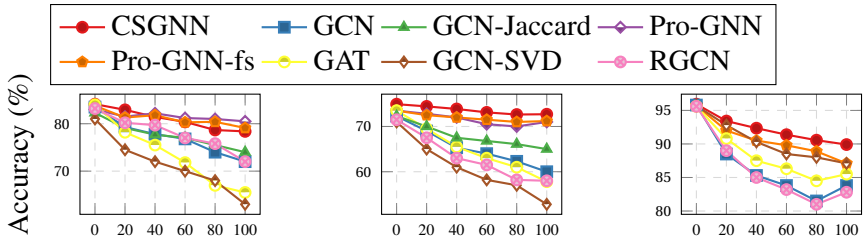
Dataset	Ptb Rate (%)	0	5	10	15	20	25
Cora	GCN	83.50±0.44	76.55±0.79	70.39±1.28	65.10±0.71	59.56±2.72	47.53±1.96
	GAT	83.97±0.65	80.44±0.74	75.61±0.59	69.78±1.28	59.94±0.92	54.78±0.74
	RGCN	83.09±0.44	77.42±0.39	72.22±0.38	66.82±0.39	59.27±0.37	50.51±0.78
	GCN-Jaccard	82.05±0.51	79.13±0.59	75.16±0.76	71.03±0.64	65.71±0.89	60.82±1.08
	GCN-SVD	80.63±0.45	78.39±0.54	71.47±0.83	66.69±1.18	58.94±1.13	52.06±1.19
	Pro-GNN-fs	83.42±0.52	82.78±0.39	77.91±0.86	76.01±1.12	68.78±5.84	56.54±2.58
	Pro-GNN	82.98±0.23	82.27±0.45	79.03±0.59	76.40±1.27	73.32±1.56	69.72±1.69
	Mid-GCN	**84.61±0.46**	**82.94±0.59**	80.14±0.86	77.77±0.75	76.58±0.29	72.89±0.81
	CSGNN	84.12±0.31	82.20±0.65	**80.43±0.74**	**79.32±1.04**	**77.47±1.22**	**74.46±0.99**
Citeseer	GCN	71.96±0.55	70.88±0.62	67.55±0.89	64.52±1.11	62.03±3.49	56.94±2.09
	GAT	73.26±0.83	72.89±0.83	70.63±0.48	69.02±1.09	61.04±1.52	61.85±1.12
	RGCN	71.20±0.83	70.50±0.43	67.71±0.30	65.69±0.37	62.49±1.22	55.35±0.66
	GCN-Jaccard	72.10±0.63	70.51±0.97	69.54±0.56	65.95±0.94	59.30±1.40	59.89±1.47
	GCN-SVD	70.65±0.32	68.84±0.72	68.87±0.62	63.26±0.96	58.55±1.09	57.18±1.87
	Pro-GNN-fs	73.26±0.38	73.09±0.34	72.43±0.52	70.82±0.87	66.19±2.38	66.40±2.57
	Pro-GNN	73.28±0.69	72.93±0.57	72.51±0.75	72.03±1.11	70.02±2.28	68.95±2.78
	Mid-GCN	74.17±0.28	74.31±0.42	73.59±0.29	73.69±0.29	71.51±0.83	69.12±0.72
	CSGNN	**74.93±0.52**	**74.91±0.33**	**73.95±0.35**	**73.82±0.61**	**73.01±0.77**	**72.94±0.56**
Polblogs	GCN	95.69±0.38	73.07±0.80	70.72±1.13	64.96±1.91	51.27±1.23	49.23±1.36
	GAT	95.35±0.20	83.69±1.45	76.32±0.85	68.80±1.14	51.50±1.63	51.19±1.49
	RGCN	95.22±0.14	74.34±0.19	71.04±0.34	67.28±0.38	59.89±0.34	56.02±0.56
	GCN-SVD	95.31±0.18	89.09±0.22	81.24±0.49	68.10±3.73	57.33±3.15	48.66±9.93
	Pro-GNN-fs	93.20±0.64	93.29±0.18	89.42±1.09	86.04±2.21	79.56±5.68	63.18±4.40
	CSGNN	**95.87±0.26**	**95.79±0.15**	**93.21±0.16**	**92.08±0.39**	**90.10±0.37**	**87.37±0.66**

Table 3.5.1: Node classification performance (accuracy±std) under a non-targeted attack (metattack) with varying perturbation rates.

3.5 Experiments

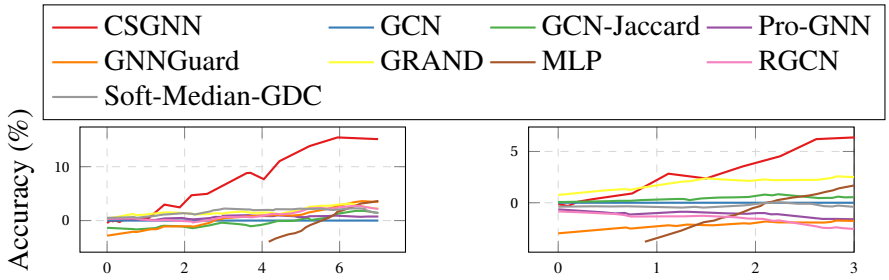
We now study the effectiveness of CSGNN against different graph adversarial attacks. In Section 3.5.1 we discuss our experimental settings. The experimental analysis we propose focuses on poisoning based on modifying the true structure of the graph, by adding/removing edges between existent nodes or perturbing their node features. The presented mathematical setup is not limited to this class of attacks, but we focus on them so as to compare our performances to similar techniques for improving the network robustness. In Section 3.5.2, we report our experimental results and observations on several benchmarks, with additional results and an ablation study in Appendix 3.K. The results presented in Section 3.5.2 focus on adjacency matrix attacks, which are the most popular in the literature [61]. To provide a comprehensive evaluation of our CSGNN, we also perform a set of experiments that include attacked node features with nettack.

Since we follow the attacks evaluated in the literature, which utilize different training/validation/test splits for different types of attacks, the reported results for perturbation rate 0 can be different under different attacks.



(a) Cora **(b)** Citeseer **(c)** Polblogs

Figure 3.5.1: Node classification accuracy (%) under a random adjacency matrix attack. The horizontal axis describes the attack percentage.



(a) Cora-ML **(b)** Citeseer

Figure 3.5.2: Node classification accuracy (%) using unit-tests from [41]. Results are relative to a baseline GCN. The horizontal axis shows the attack budget (%).

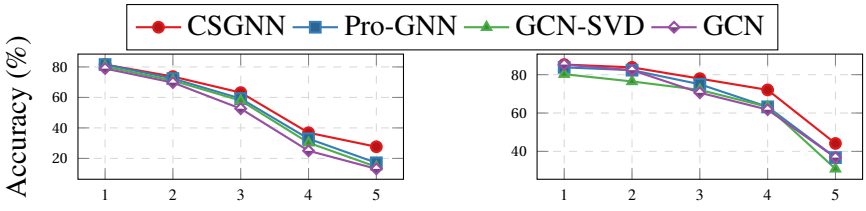
3.5.1 Experimental settings

Datasets. Following [67, 68], we validate the proposed approach on four benchmark datasets, including three citation graphs, i.e., Cora, Citeseer, Pubmed, and one blog graph, Polblogs. The statistics of the datasets are shown in Appendix 3.I. Note that in the Polblogs graph, node features are not available. In this case, we follow Pro-GNN [32] and set the input node features to a $n \times n$ identity matrix.

Baselines. We demonstrate the efficacy of CSGNN by comparing it with popular GNNs and defense models, as follows: **GCN** [35]: Is one of the most commonly used GNN architectures, consisting of feature propagation according to the symmetric normalized Laplacian and channel mixing steps. **GAT** [54]: Graph Attention Networks (GAT) employ an attention mechanism to learn edge weights for the feature propagation step. **RGCN** [66]: RGCN models node features as samples from Gaussian distributions, and modifies GCN to propagate both the mean and the variance. In the neighborhood aggregation

operation, high-variance features are down-weighted to improve robustness. **GCN-Jaccard** [60]: This is an unsupervised pre-processing method that relies on binary input node features, based on the assumption that the true graph is homophilic. Edges between nodes with features whose Jaccard similarity is below a certain threshold are removed. **GCN-SVD** [18]: GCN-SVD is also an unsupervised pre-processing step. Based on the observation that nettack tends to generate high-rank perturbations to the adjacency matrix, it is suggested to truncate the SVD of the adjacency matrix before it is used to train a GNN. **Pro-GNN** [32]: Pro-GNN attempts to jointly optimize GCN weights and a corrected adjacency matrix using a loss function consisting of a downstream supervised task-related loss function and low-rank and sparsity regularization. In Pro-GNN-fs, an additional feature smoothing regularization is used. **Mid-GCN** [27]: Mid-GCN modifies the standard GCN architecture to utilize a mid-pass filter, unlike the typical low-pass filter in GCN. **GNNGuard** [64]: GNNGuard modifies message-passing GNNs to include layer-dependent neighbor importance weights in the aggregation step. The neighbor importance weights are designed to favor edges between nodes with similar features, encoding an assumption of homophily. **GRAND** [20]: In this method, multiple random graph data augmentations are generated, which are then propagated through the GNN. The GNN is trained using a task-related loss and a consistency regularization that encourages similar outputs for the different augmented graphs. **Soft-Median-GDC** [21]: This approach first preprocesses the adjacency matrix using graph diffusion convolution [36], after which a GNN that uses soft median neighborhood aggregation function is trained. **GARNET** [16]: This method suggests wiring the graph using weighted spectral embeddings, which are shown to be related to the original, clean graph. **HANG** [65]: This approach is based on conservative Hamiltonian neural flows, used to process node features and for improved robustness. The comparisons with GNNGuard, GARNET and HANG are reported in Appendix 3.K.

Training and Evaluation. We follow the same experimental settings as in [32]. Put precisely, and unless otherwise specified, for each dataset, we randomly choose 10% of the nodes for training, 10% of the nodes for validation, and the remaining 80% nodes for testing. For each experiment, we report the average node classification accuracy of 10 runs. The hyperparameters of all the models are tuned based on the validation set accuracy. In all experiments, the objective function to be minimized is the cross-entropy loss, using the Adam optimizer [34]. Note that another benefit of our CSGNN is the use of downstream loss only, compared to other methods that utilize multiple losses to learn adjacency matrix updates. In Appendix 3.J, we discuss the hyperparameters of CSGNN. A complexity and runtime discussion is given in Appendix 3.L.



(a) Cora **(b)** Citeseer

Figure 3.5.3: Node classification accuracy (%) under targeted attack with nettack to both node features and adjacency matrix. The horizontal axis describes the number of perturbations per node.

3.5.2 Adversarial Defense Performance

We evaluate the node classification performance of CSGNN against four types of poisoning attacks: (i) non-targeted attack, (ii) targeted attack, (iii) random attack, and, (iv) unit tests. Below we elaborate on the results obtained on each type of attack.

Robustness to Non-Targeted Adversarial Attacks. We evaluate the node classification accuracy of our CSGNN and compare it with the baseline methods after using the non-targeted adversarial attack metattack [68]. We follow the publicly available attacks and splits in [32]. We experiment with varying perturbation rates, i.e., the ratio of changed edges, from 0 to 25% with a step size of 5%. We report the average accuracy, as well as the obtained standard deviation over 10 runs in Table 3.5.1. The best-performing method is highlighted in bold. Except for a few cases, our CSGNN consistently improves or offers on-par performance with other methods.

Robustness to Targeted Adversarial Attacks. In this experiment, we use nettack [67] as a targeted attack. Following [66], we vary the number of perturbations made on every targeted node from 1 to 5 with a step size of 1. The nodes in the test set with a degree larger than 10 are set as target nodes. Here, we also use the publicly available splits in [32]. The node classification accuracy on target nodes is shown in Figure 3.4.2. From the figure, we can observe that when the number of perturbations increases, the performance of CSGNN is better than other methods on the attacked target nodes in most cases.

Robustness to targeted attacks to node features and adjacency matrix. We now provide additional experiments, where not only the connectivity struc-

ture of the graph is attacked, but also the node features, demonstrated on the Cora and Citeseer datasets. To generate the attacked versions of these datasets, we follow the same protocol as in Pro-GNN [32]. The attacks are based on nettack [67], which applies a targeted attack to the test nodes of the clean graph having a degree larger than 10. To attack all these nodes, we iterate through the target nodes and iteratively update the feature and adjacency matrices attacking the previously obtained one at the next target node. We work with different attack intensities, applying 1 to 5 perturbations per targeted node, with a step of 1. The results are reported in Figure 3.5.3, where we compare the performance of CSGNN, with those of GCN, GCN-SVD, and Pro-GNN. As we can see, CSGNN outperforms all of the compared models on this task.

Robustness to Random Attacks. In this experimental setting, we evaluate the performance of CSGNN when the adjacency matrix is attacked by adding random fake edges, from 0% to 100% of the number of edges in the true adjacency matrix, with a step size of 20%. The results are reported in Figure 3.5.1. It can be seen, that CSGNN is on par with or better than the considered baselines.

Unit tests. We utilize the recently suggested *unit tests* from [41]. This is a set of perturbed citation datasets, which are notable for the fact that the perturbations were not generated using standard attack generation procedures that focus only on attacks like nettack or metattack. Instead, 8 adversarial defense methods were studied. Then, bespoke, adaptive attack methods were designed for each of them. These attack methods were applied to the citation datasets to generate the "unit tests". We experiment with those attacks as they offer a challenging benchmark, that further highlights the contribution of our CSGNN. We present the results in Figure 3.5.2, showing the relative performance of CSGNN and other baselines compared to GCN. We see that our CSGNN performs better than other considered models. This result further highlights the robustness of CSGNN under different adversarial attack scenarios, on several datasets. In Figure 3.K.3 of Appendix 3.K, we also provide absolute performance results.

3.6 Summary and Discussion

In this paper, we present CSGNN, a novel GNN architecture inspired by contractive dynamical systems for graph adversarial defense. Our CSGNN learns a coupled dynamical system that updates both the node features as well as the

adjacency matrix to reduce the impact of input perturbations, thereby defending against graph adversarial attacks. We provide a theoretical analysis of our CSGNN, to gain insights into its characteristics and expected behavior. Our profound experimental study of CSGNN reveals the importance of employing the proposed coupled dynamical system to reduce attack influence on the model's accuracy. Namely, our results verify both the efficacy compared to existing methods, as well as the necessity of each of the dynamical systems in CSGNN. Since our approach presents a novel way to model both the node features and adjacency matrix through the lens of dynamical systems, we believe that our findings and developments will find further use in graph adversarial defense and attacks, as well as other applications of GNNs.

Appendix

Remark. In the Appendices, we often work with the Jacobian matrix of a piecewise smooth function, like $\sigma(x) = \mathrm{LeakyReLU}(x)$. This is done to bound the Lipschitz constant $\mathrm{Lip}(f)$ of a map, or its one-sided Lipschitz constant $\mathrm{osLip}(f)$. For Lipschitz-continuous maps, by Rademacher's Theorem [19, Theorem 3.1.6], the Jacobian is defined almost everywhere, and one can still obtain the convenient relations

$$\begin{aligned} \mathrm{Lip}\big(f\big) \leq L &\iff \big\|Df(x)\big\| \leq L \text{ a.e.,} \\ \mathrm{osLip}\big(f\big) \leq c &\iff \mu\big(Df(x)\big) \leq c \text{ a.e.} \end{aligned}$$

for a Lipschitz continuous function $f : \Omega \to \mathbb{R}^n$, where Ω is an open convex subset of $\mathbb{R}^n$. The second relation can be found, for example, in [14, Theorem 16]. For simplicity, we do not say all the quantities are defined almost everywhere throughout.

3.A Contractive Systems

This appendix defines continuous contractive dynamical systems and provides a background on the properties of such systems. We focus on contractivity with respect to a norm $\|\cdot\|$ on $\mathbb{R}^k$ induced by an inner product $\langle\cdot,\cdot\rangle : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$, i.e. so that $\|\mathbf{x}\|^2 = \langle \mathbf{x}, \mathbf{x}\rangle$ for every $\mathbf{x} \in \mathbb{R}^k$. This extends thanks to the less restrictive notion of weak pairing considered in [6]. Following this more general approach, the reasoning extends naturally to the ℓ^1 norm, i.e., the one used in the case of the dynamical system we propose for the adjacency matrix. Let us consider the dynamical system

$$\begin{cases} \dot{x}(t) = f\big(x(t)\big) \in \mathbb{R}^k, \\ x\big(t_0\big) = \mathbf{x}_0 \in \mathbb{R}^k. \end{cases} \tag{3.A.1}$$

Consider the convex set $\Omega \subset \mathbb{R}^k$. We say $f : \Omega \to \mathbb{R}^k$ satisfies the one-sided Lipschitz inequality on Ω with constant $\mathrm{osLip}(f) \in \mathbb{R}$ if for every $\mathbf{x}, \mathbf{y} \in \Omega$

$$\left\langle f(\mathbf{x}) - f(\mathbf{y}), \mathbf{x} - \mathbf{y} \right\rangle \leq \mathrm{osLip}(f) \left\| \mathbf{x} - \mathbf{y} \right\|^2. \tag{3.A.2}$$

Additionally, we remark that if f is $\mathrm{Lip}(f)-$Lipschitz continuous, then its one-sided constant $\mathrm{osLip}(f) \leq \mathrm{Lip}(f)$ since one has

$$\left\langle f(\mathbf{x}) - f(\mathbf{y}), \mathbf{x} - \mathbf{y} \right\rangle \leq \left\| f(\mathbf{x}) - f(\mathbf{y}) \right\| \cdot \left\| \mathbf{x} - \mathbf{y} \right\| \leq \mathrm{Lip}(f) \left\| \mathbf{x} - \mathbf{y} \right\|^2.$$

However, while the Lipschitz constant can only be non-negative, the one-sided Lipschitz constant can also be strictly negative. For example, if $f(\mathbf{x}) = -\mathbf{x}$ we get $\mathrm{osLip}(f) = -1$.

Definition 3.1 (Contractive dynamical system)**.** Let $\Omega \subset \mathbb{R}^k$ be a convex set. We say the dynamical system in (3.A.1) is strictly contractive on Ω if it satisfies the one-sided Lipschitz inequality (3.A.2) with $\mathrm{osLip}(f) < 0$ for every $\mathbf{x}, \mathbf{y} \in \Omega$, and contractive if $\mathrm{osLip}(f) \leq 0$.

The motivation behind definition 3.1 comes from a relatively simple derivation. Let $\mathbf{x}(t)$ and $\mathbf{y}(t)$ be two analytical solutions to (3.A.1), respectively with $\mathbf{x}(t_0) = \mathbf{x}_0$ and $\mathbf{y}(t_0) = \mathbf{y}_0$. Then one has

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{1}{2}\left\|\mathbf{x}(t) - \mathbf{y}(t)\right\|^2\right) = \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{1}{2}\left\langle \mathbf{x}(t) - \mathbf{y}(t), \mathbf{x}(t) - \mathbf{y}(t)\right\rangle\right) \\
&= \left\langle f\left(\mathbf{x}(t)\right) - f\left(\mathbf{y}(t)\right), \mathbf{x}(t) - \mathbf{y}(t)\right\rangle \leq \mathrm{osLip}\left(f\right)\left\|\mathbf{x}(t) - \mathbf{y}(t)\right\|^2.
\end{aligned}$$

By Gronwall's inequality [24], one hence gets

$$\left\|\mathbf{x}(t) - \mathbf{y}(t)\right\| \leq \left\|\mathbf{x}_0 - \mathbf{y}_0\right\| e^{\mathrm{osLip}(f)(t - t_0)}, \quad \forall\, t \geq t_0.$$

As a consequence, $\|\mathbf{x}(t) - \mathbf{y}(t)\|$ tends to 0 exponentially fast as $t \to +\infty$ if $\mathrm{osLip}(f) < 0$, while it changes in a stable way when $\mathrm{osLip}(f) \leq 0$ since

$$\left\|\mathbf{x}(t) - \mathbf{y}(t)\right\| \leq \left\|\mathbf{x}_0 - \mathbf{y}_0\right\|, \quad \forall\, t \geq t_0.$$

It hence follows that $\mathrm{osLip}(f)$ provides a contraction rate of a generic pair of trajectories, one towards the other. To conclude this section, we focus briefly on the dynamical systems considered for the features. This is of the form

$$\begin{cases} \dot{x}(t) = -\nabla V\left(x(t)\right) \\ x\left(t_0\right) = \mathbf{x}_0 \end{cases}$$

for a convex function $V : \mathbb{R}^k \to \mathbb{R}$. The properties of convex functions guarantee

$$-\left\langle \nabla V(\mathbf{x}) - \nabla V\left(\mathbf{y}\right), \mathbf{x} - \mathbf{y}\right\rangle \leq 0$$

and hence that $\mathrm{osLip}(-\nabla V) \leq 0$. This control on the expansivity of the dynamics is the main motivation for our proposed forward update in Section 3.4.

3.B Expression for the linear equivariant layer in the adjacency dynamics

We now write the explicit expression for the linear permutation equivariant layer adopted in the updates of the adjacency matrix. This layer depends on 9 learnable scalar parameters $k_1, \dots, k_9 \in \mathbb{R}$ and takes the form

$$\begin{aligned} M(\mathbf{A}) = k_1\mathbf{A} + k_2\mathrm{diag}\big(\mathrm{diag}(\mathbf{A})\big) + \frac{k_3}{2n}\Big(\mathbf{A}\mathbf{1}_n\mathbf{1}_n^\top + \mathbf{1}_n\mathbf{1}_n^\top\mathbf{A}\Big) + k_4\mathrm{diag}\big(\mathbf{A}\mathbf{1}_n\big) \\ + \frac{k_5}{n^2}\Big(\mathbf{1}_n^\top\mathbf{A}\mathbf{1}_n\Big)\mathbf{1}_n\mathbf{1}_n^\top + \frac{k_6}{n}\Big(\mathbf{1}_n^\top\mathbf{A}\mathbf{1}_n\Big)I_n + \frac{k_7}{n^2}\Big(\mathbf{1}_n^\top\mathrm{diag}(\mathbf{A})\Big)\mathbf{1}_n\mathbf{1}_n^\top \\ + \frac{k_8}{n}\Big(\mathbf{1}_n^\top\mathrm{diag}(\mathbf{A})\Big)I_n + \frac{k_9}{2n}\Big(\mathrm{diag}(\mathbf{A})\mathbf{1}_n^\top + \mathbf{1}_n\big(\mathrm{diag}(\mathbf{A})\big)^\top\Big), \end{aligned} \tag{3.B.1}$$

where $I_n \in \mathbb{R}^{n\times n}$ denotes the identity matrix. The operator diag acts both on matrices and vectors, and is defined as

$$\begin{aligned} &\mathrm{diag}: \mathbb{R}^{n\times n} \to \mathbb{R}^n,\ \mathrm{diag}(\mathbf{A}) = \sum_{i=1}^n \Big(\mathbf{e}_i^\top\mathbf{A}\mathbf{e}_i\Big)\mathbf{e}_i, \\ &\mathrm{diag}: \mathbb{R}^n \to \mathbb{R}^{n\times n},\ \mathrm{diag}(\mathbf{a}) = \sum_{i=1}^n \Big(\mathbf{a}^\top\mathbf{e}_i\Big)\mathbf{e}_i\mathbf{e}_i^\top, \end{aligned}$$

with $\mathbf{e}_i \in \mathbb{R}^n$ a one-hot vector with 1 in the $i-$th entry. For matrix input, the main diagonal is extracted. For vector input, its values are placed on the diagonal of a matrix.

3.C Contractivity of the feature updating rule

Before proving the contractivity of the feature update rule, we report the definition of the graph gradient operator $\mathcal{G}$, for completeness.

Definition 3.2 (Graph gradient operator)**.** We define the graph gradient operator $\mathcal{G}(\mathbf{A}) : \mathbb{R}^{n\times c} \to \mathbb{R}^{n\times n\times c}$, as follows:

$$\big(\mathcal{G}(\mathbf{A})\,\mathbf{F}\big)_{ijk} = \mathbf{A}_{ij}\Big(\mathbf{F}_{ik} - \mathbf{F}_{jk}\Big),\ i,j \in \{1,\dots,n\}, k \in \{1,\dots,c\},$$

and its transpose $\mathcal{G}(\mathbf{A})^\top : \mathbb{R}^{n\times n\times c} \to \mathbb{R}^{n\times c}$ as

$$\Big(\mathcal{G}(\mathbf{A})^\top\mathbf{O}\Big)_{ik} = \sum_{j=1}^n \Big(\mathbf{A}_{ij}\mathbf{O}_{ijk} - \mathbf{A}_{ji}\mathbf{O}_{jik}\Big),\ i \in \{1,\dots,n\}, k \in \{1,\dots,c\}.$$

Note that in practice, we compute it only if the entry $\mathbf{A}_{ij} \neq 0$, and that this gradient operator is just a spatial difference operation applied channel-wise as is common in previous methods [9, 17].

We now turn to prove that the node feature update rule is contractive. First of all we notice that $X_l(\mathbf{0}_{n\times c}, \mathbf{A}) = \mathbf{0}_{n\times c}$ for every $\mathbf{A} \in \mathbb{R}^{n\times n}$, and hence since X_l is L-Lipschitz for some $L > 0$, we also can conclude

$$\left\| X_l\left(\mathbf{F}, \mathbf{A}\right) - X_l\left(\mathbf{0}_{n\times c}, \mathbf{A}\right) \right\|_F = \left\| X_l\left(\mathbf{F}, \mathbf{A}\right) \right\|_F \leq L \left\|\mathbf{F}\right\|_F. \tag{3.C.1}$$

We remark that since both $\mathcal{G}(\mathbf{A})$ and $\mathcal{G}(\mathbf{A})^\top$ are linear operations, one could equivalently introduce them as matrices acting on the vectorization $\mathrm{vec}(\mathbf{F})$. We call the matrix version of the gradient operator $\widehat{\mathcal{G}}(\mathbf{A})$ and similarly for its transpose. Using this notation, we see that

$$\mathrm{vec}\left(\mathcal{G}\left(\mathbf{A}\right)\mathbf{F}\mathbf{W}_l\right) = \widehat{\mathcal{G}}\left(\mathbf{A}\right)\mathrm{vec}\left(\mathbf{F}\mathbf{W}_l\right) = \widehat{\mathcal{G}}\left(\mathbf{A}\right)\left(\mathbf{W}_l^\top \otimes I_n\right)\mathrm{vec}\left(\mathbf{F}\right).$$

Hence,

$$\mathrm{vec}\left(X_l\left(\mathbf{F}, \mathbf{A}\right)\right) = -\left(\tilde{\mathbf{K}}_l \otimes I_n\right)\left(\mathbf{W}_l \otimes I_n\right)\widehat{\mathcal{G}}\left(\mathbf{A}\right)^\top \sigma\left(\widehat{\mathcal{G}}\left(\mathbf{A}\right)\left(\mathbf{W}_l^\top \otimes I_n\right)\mathrm{vec}\left(\mathbf{F}\right)\right).$$

We then introduce the energy

$$\mathcal{E}_{\mathbf{A}}(\mathbf{F}) = \mathbf{1}^\top \gamma\left(\widehat{\mathcal{G}}(\mathbf{A})\left(\mathbf{W}_l^\top \otimes I_n\right)\mathrm{vec}\left(\mathbf{F}\right)\right),\ \gamma'(s) = \sigma(s),$$

where $\mathbf{1} \in \mathbb{R}^{n\cdot n\cdot c}$, is a vector of all ones. This energy is convex in $\mathbf{F}$ since it is obtained by composing convex functions, given that σ is non-decreasing.

Notice that since the gradient of $\mathcal{E}_{\mathbf{A}}$ with respect to $\mathbf{f} := \mathrm{vec}(\mathbf{F})$ writes

$$\nabla_{\mathbf{f}}\mathcal{E}_{\mathbf{A}}\left(\mathbf{F}\right) = \left(\mathbf{W}_l \otimes I_n\right)\widehat{\mathcal{G}}\left(\mathbf{A}\right)^\top \sigma\left(\widehat{\mathcal{G}}\left(\mathbf{A}\right)\left(\mathbf{W}_l^\top \otimes I_n\right)\mathrm{vec}\left(\mathbf{F}\right)\right),$$

we can express $\mathrm{vec}(X_l(\mathbf{A}, \mathbf{F}))$ in the simpler form

$$\hat{X}_l\left(\mathbf{F}, \mathbf{A}\right) := \mathrm{vec}\left(X_l\left(\mathbf{F}, \mathbf{A}\right)\right) = -\left(\tilde{\mathbf{K}}_l \otimes I_n\right)\nabla_{\mathbf{f}}\mathcal{E}_{\mathbf{A}}(\mathbf{F}).$$

This allows us to prove both Theorems 3.1 and 3.2 for two interesting configurations. First, we notice that if $\tilde{\mathbf{K}}_l$ is positive definite, when $\mathbf{F} \neq \mathbf{0}_{n\times c}$ and $\nabla_{\mathbf{f}}\mathcal{E}_{\mathbf{A}}(\mathbf{F}) \neq \mathbf{0}_{n\times c}$, we have

$$\nabla_{\mathbf{f}}\mathcal{E}_{\mathbf{A}}\left(\mathbf{F}\right)^\top \mathrm{vec}\left(X_l\left(\mathbf{A}, \mathbf{F}\right)\right) \leq \lambda_{\max}\left(-\tilde{\mathbf{K}}_l\right)\left\|\nabla_{\mathbf{f}}\mathcal{E}_{\mathbf{A}}\left(\mathbf{F}\right)\right\|_2^2 \leq L^2\lambda_{\max}\left(-\tilde{\mathbf{K}}_l\right)\left\|\mathbf{F}\right\|_F^2 < 0.$$

Here, with $\lambda_{\max}(-\tilde{\mathbf{K}}_l)$ we denote the maximum eigenvalue of $-\tilde{\mathbf{K}}_l$, which is negative since $\tilde{\mathbf{K}}_l$ is positive definite. We then have

$$\mathcal{E}_{\mathbf{A}}\left(\Psi_{X_l}^{h_l}\left(\mathbf{F}, \mathbf{A}\right)\right) \leq \mathcal{E}_{\mathbf{A}}\left(\mathbf{F}\right)$$

for small enough $h_l > 0$ since $\hat{X}_l$ locally provides a descent direction for $\mathcal{E}_{\mathbf{A}}$. This result guarantees that the updates $\mathbf{F}^{(l)}$ will remain bounded.

To prove Theorem 3.2, we first notice that

$$\|\mathbf{P}\|_F = \|\mathrm{vec}\,(\mathbf{P})\|_2 \quad \forall \mathbf{P} \in \mathbb{R}^{n\times c},$$

and thus prove the result for the vectorization of $X_l(\mathbf{A},\mathbf{F})$. We focus on the case $\mathbf{K}_l = \lambda I_c$, $\lambda > 0$, which is the one tested in the experiments of Appendix 3.K. Given that in this case $\mathrm{vec}(X_l(\mathbf{F},\mathbf{A})) = -\lambda\nabla_{\mathbf{f}}(\mathcal{E}_{\mathbf{A}}(\mathbf{F},\mathbf{A}))$, we can immediately conclude. Indeed, we can apply the results in [49], for example, to prove the desired result for every $\mathbf{W}_l \in \mathbb{R}^{n\times n}$. This is just a direct consequence of the properties of convex functions with Lipschitz gradient.

3.D Proofs for the contractivity of the adjacency matrix updates

This section aims to provide a detailed proof of Theorem 3.3. This is divided into various steps. We first provide the vectorized version of such theorem, which we then prove. The theorem then follows directly by definition of the vectorized ℓ^1 norm.

Let $M(\mathbf{A})$ be defined as in (3.B.1). It is clear that M is linear in $\mathbf{A} \in \mathbb{R}^{n\times n}$, and thus that there exists a matrix $\mathbf{T} \in \mathbb{R}^{n^2\times n^2}$ such that

$$\mathrm{vec}\left(M\left(\mathbf{A}^{(l)}\right)\right) = \mathbf{T}\,\mathrm{vec}\left(\mathbf{A}^{(l)}\right). \tag{3.D.1}$$

We now characterize the explicit expression of such $\mathbf{T}$.

Theorem 3.4. *The matrix* $\mathbf{T}$ *can be written as follows:*

$$\begin{aligned}\mathbf{T} = {} & k_1 I_{n^2} + k_2 \sum_{i=1}^{n}\left(\mathbf{e}_i\mathbf{e}_i^\top\right)\otimes\left(\mathbf{e}_i\mathbf{e}_i^\top\right) + \frac{k_3}{2n}\left(\mathbf{1}_n\mathbf{1}_n^\top\otimes I_n + I_n\otimes\mathbf{1}_n\mathbf{1}_n^\top\right)\\ & + k_4\sum_{i=1}^{n}\left(\mathbf{e}_i\mathbf{1}_n^\top\right)\otimes\left(\mathbf{e}_i\mathbf{e}_i^\top\right) + \frac{k_5}{n^2}\left(\mathbf{1}_n\mathbf{1}_n^\top\right)\otimes\left(\mathbf{1}_n\mathbf{1}_n^\top\right)\\ & + \frac{k_6}{n}\sum_{i=1}^{n}\left(\mathbf{e}_i\mathbf{1}_n^\top\right)\otimes\left(\mathbf{e}_i\mathbf{1}_n^\top\right) + \frac{k_7}{n^2}\sum_{i=1}^{n}\left(\mathbf{1}_n\mathbf{e}_i^\top\right)\otimes\left(\mathbf{1}_n\mathbf{e}_i^\top\right)\\ & + \frac{k_8}{n}\sum_{i,j=1}^{n}\left(\mathbf{e}_j\mathbf{e}_i^\top\right)\otimes\left(\mathbf{e}_j\mathbf{e}_i^\top\right) + \frac{k_9}{2n}\sum_{i=1}^{n}\left(\left(\mathbf{1}_n\mathbf{e}_i^\top\right)\otimes\left(\mathbf{e}_i\mathbf{e}_i^\top\right) + \left(\mathbf{e}_i\mathbf{e}_i^\top\right)\otimes\left(\mathbf{1}_n\mathbf{e}_i^\top\right)\right)\end{aligned} \tag{3.D.2}$$

where $\mathbf{e}_i \in \mathbb{R}^{n^2}$ *is the* $i-$*th element of the canonical basis.*

Proof. We do it by focusing on the separate 9 pieces defining $\mathbf{T}$. The main result needed to complete the derivation is that

$$\operatorname{vec}(\mathbf{ABC}) = \left(\mathbf{C}^\top \otimes \mathbf{A}\right) \operatorname{vec}(\mathbf{B}). \tag{3.D.3}$$

We omit the part with k_1 since it is trivial. Let us start with

$$\begin{aligned}\operatorname{diag}(\operatorname{diag}(\mathbf{A})) &= \sum_{i=1}^{n} \left(\mathbf{e}_i^\top A \mathbf{e}_i\right) \mathbf{e}_i \mathbf{e}_i^\top = \sum_{i=1}^{n} \mathbf{e}_i \left(\mathbf{e}_i^\top A \mathbf{e}_i\right) \mathbf{e}_i^\top \\ &= \sum_{i=1}^{n} \left(\mathbf{e}_i \mathbf{e}_i^\top\right) A \left(\mathbf{e}_i \mathbf{e}_i^\top\right)\end{aligned}$$

which allows us to conclude the proof by the linearity of the vec operator, and (3.D.3). Moving to the k_3 part we have that by (3.D.3)

$$\operatorname{vec}\left(\mathbf{A}\mathbf{1}_n \mathbf{1}_n^\top + \mathbf{1}_n \mathbf{1}_n^\top \mathbf{A}\right) = \left(\mathbf{1}_n \mathbf{1}_n^\top \otimes I_n + I_n \otimes \mathbf{1}_n \mathbf{1}_n^\top\right) \operatorname{vec}(A).$$

The k_4 part can be rewritten as

$$\operatorname{diag}(\mathbf{A}\mathbf{1}_n) = \sum_{i=1}^{n} \left(\mathbf{e}_i^\top \mathbf{A}\mathbf{1}_n\right) \mathbf{e}_i \mathbf{e}_i^\top = \sum_{i=1}^{n} \mathbf{e}_i \mathbf{e}_i^\top \mathbf{A}\mathbf{1}_n \mathbf{e}_i^\top$$

which gives

$$\operatorname{vec}\left(\operatorname{diag}(\mathbf{A}\mathbf{1}_n)\right) = \left(\sum_{i=1}^{n} \left(\mathbf{e}_i \mathbf{1}_n^\top\right) \otimes \left(\mathbf{e}_i \mathbf{e}_i^\top\right)\right) \operatorname{vec}(\mathbf{A}).$$

Term for k_5 follows immediately from (3.D.3), while for k_6 we can write

$$\left(\mathbf{1}_n^\top \mathbf{A}\mathbf{1}_n\right) I_n = \left(\mathbf{1}_n^\top \mathbf{A}\mathbf{1}_n\right) \sum_{i=1}^{n} \mathbf{e}_i \mathbf{e}_i^\top = \sum_{i=1}^{n} \mathbf{e}_i \left(\mathbf{1}_n^\top \mathbf{A}\mathbf{1}_n\right) \mathbf{e}_i^\top$$

which implies the desired expression. For k_7 it holds

$$\left(\mathbf{1}_n^\top \operatorname{diag}(\mathbf{A})\right) \mathbf{1}_n \mathbf{1}_n^\top = \sum_{i=1}^{n} \left(\mathbf{e}_i^\top \mathbf{A}\mathbf{e}_i\right) \mathbf{1}_n \mathbf{1}_n^\top = \sum_{i=1}^{n} \mathbf{1}_n \mathbf{e}_i^\top \mathbf{A}\mathbf{e}_i \mathbf{1}_n^\top$$

which allows us to conclude. We now move to

$$\left(\mathbf{1}_n^\top \operatorname{diag}(\mathbf{A})\right) I_n = \sum_{i=1}^{n} \left(\mathbf{e}_i^\top \mathbf{A}\mathbf{e}_i\right) \sum_{j=1}^{n} \mathbf{e}_j \mathbf{e}_j^\top = \sum_{i,j=1}^{n} \mathbf{e}_j \mathbf{e}_i^\top \mathbf{A}\mathbf{e}_i \mathbf{e}_j^\top$$

and hence

$$\operatorname{vec}\left(\left(\mathbf{1}_n^\top \operatorname{diag}(\mathbf{A})\right) I_n\right) = \left(\sum_{i,j=1}^{n} \left(\mathbf{e}_j \mathbf{e}_i^\top\right) \otimes \left(\mathbf{e}_j \mathbf{e}_i^\top\right)\right) \operatorname{vec}(\mathbf{A}).$$

We now consider the term multiplying k_9, which writes

$$\begin{aligned}
&\operatorname{diag}(\mathbf{A})\,\mathbf{1}_n^\top + \mathbf{1}_n \operatorname{diag}(\mathbf{A})^\top \\
&= \sum_{i=1}^{n} \left(\mathbf{e}_i^\top \mathbf{A} \mathbf{e}_i\right) \mathbf{e}_i \mathbf{1}_n^\top + \mathbf{1}_n \sum_{i=1}^{n} \left(\mathbf{e}_i^\top \mathbf{A} \mathbf{e}_i\right) \mathbf{e}_i^\top, \\
&\operatorname{vec}\left(\operatorname{diag}(\mathbf{A})\,\mathbf{1}_n^\top + \mathbf{1}_n \operatorname{diag}(\mathbf{A})^\top\right) \\
&= \left(\sum_{i=1}^{n} \left(\mathbf{1}_n \mathbf{e}_i^\top\right) \otimes \left(\mathbf{e}_i \mathbf{e}_i^\top\right) + \left(\mathbf{e}_i \mathbf{e}_i^\top\right) \otimes \left(\mathbf{1}_n \mathbf{e}_i^\top\right)\right) \operatorname{vec}(\mathbf{A}),
\end{aligned}$$

and concludes the proof. □

We now want to evaluate how large can $h_l > 0$ be so that the condition

$$\left\| D\Psi_{\hat{Y}_l}^{h_l} \right\|_1 \leq 1$$

is satisfied. Let us note here that the norm under consideration is the *matrix* ℓ^1 norm, not the usual *vector* ℓ^1 norm, which has previously occurred in the main text. Recall that the matrix ℓ^1 norm of a matrix $\mathbf{T} \in \mathbb{R}^{n^2 \times n^2}$ is given as the maximum of the absolute column sums:

$$\|\mathbf{T}\|_1 = \max_{1 \leq j \leq n^2} \sum_{i=1}^{n^2} \left|\mathbf{T}_{ij}\right|.$$

Theorem 3.5. *The matrix* $\mathbf{T} - k_1 I_{n^2}$*, with* $\mathbf{T}$ *defined in (3.D.2), has* ℓ^1 *norm bounded by* $\sum_{i=2}^{9} |k_i|$.

Proof. Given that one can bound the norm of the sum with the sum of the norms, it is enough to show that the norms of all the contributions in (3.D.2) can be bounded by the absolute value of the respective constant k_i. The proof follows from multiplying from the left every term by $\mathbf{1}_{n^2}$ and using the linearity of the sum. □

For compactness, we now denote $\operatorname{vec}(Y_l(\mathbf{A}))$ with $\hat{Y}_l(\mathbf{a})$, where $\mathbf{a} = \operatorname{vec}(\mathbf{A})$. Furthermore, the map $\Psi_{\hat{Y}_l}^{h_l}$ is defined as $\Psi_{\hat{Y}_l}^{h_l}(\mathbf{a}) := \operatorname{vec}(\Psi_{Y_l}^{h_l}(\mathbf{A}))$.

Theorem 3.6. *Let* $\alpha \leq 0$*,* $\sigma : \mathbb{R} \to \mathbb{R}$ *be a Lipschitz continuous function, with* $\sigma'(s) \in [0, 1]$*. Then if*

$$0 \leq h_l \leq \frac{2}{2\sum_{i=2}^{9} |k_i| - \alpha},$$

the explicit Euler step

$$\mathbf{a}^{(l)} = \Psi_{\hat{Y}_l}^{h_l}\left(\mathbf{a}^{(l-1)}\right) := \mathbf{a}^{(l-1)} + h_l \sigma\left(\mathbf{T}\mathbf{a}^{(l-1)}\right), \quad k_1 = \left(\alpha - \sum_{i=2}^{9} |k_i|\right), \tag{3.D.4}$$

is contractive in the ℓ^1 norm.

For the proof and the successive derivations, we denote with $D\hat{Y}_l(\mathbf{a}) \in \mathbb{R}^{n^2 \times n^2}$ the Jacobian matrix of the vector field $\hat{Y}_l : \mathbb{R}^{n^2} \to \mathbb{R}^{n^2}$, having entries

$$\left(D\hat{Y}_l(\mathbf{a})\right)_{ij} = \frac{\partial\left(\hat{Y}_l(\mathbf{a})\right)_i}{\partial \mathbf{a}_j}. \tag{3.D.5}$$

The Jacobian matrix is needed because, for functions that are almost everywhere continuously differentiable, the contractivity condition is equivalent to

$$\left\| D\Psi^{h_l}_{\hat{Y}_l}(\mathbf{a}) \right\|_1 \leq 1 \tag{3.D.6}$$

almost everywhere.

Proof. For compactness, we drop the superscript and denote $\mathbf{a}^{(l-1)}$ as $\mathbf{a}$, since the provided estimates are independent of the evaluation point. The map $\Psi^{h_l}_{\hat{Y}_l}$ is differentiable almost everywhere and hence the result simplifies to proving

$$\left\| D\Psi^{h_l}_{\hat{Y}_l}(\mathbf{a}) \right\|_1 \leq 1.$$

We thus compute

$$D\Psi^{h_l}_{\hat{Y}_l}(\mathbf{a}) = I_{n^2} + h_l \operatorname{diag}\left(\sigma'(\mathbf{Ta})\right)\mathbf{T}.$$

To simplify the proof, we introduce the matrix $\mathbf{S} = \mathbf{T} - k_1 I_{n^2}$. This means that the update in (3.D.4) can be written as

$$\Psi^{h_l}_{\hat{Y}_l}(\mathbf{a}) = \mathbf{a} + h_l \sigma\left(\mathbf{Sa} + \left(\alpha - \sum_{i=2}^{9} |k_i|\right)\mathbf{a}\right).$$

We call $d_1, \ldots, d_{n^2}$ the diagonal entries of the matrix $\operatorname{diag}(\sigma'(\cdot))$. It follows

$$\begin{aligned}
\left(D\Psi^{h_l}_{\hat{Y}_l}(\mathbf{a})\right)_{ii} &= 1 + h_l d_i \left(\mathbf{S}_{ii} + \left(\alpha - \sum_{i=2}^{9} |k_i|\right)\right) \\
\left(D\Psi^{h_l}_{\hat{Y}_l}(\mathbf{a})\right)_{ji} &= h_l d_j \mathbf{S}_{ji}.
\end{aligned}$$

If $(D\Psi^{h_l}_{\hat{Y}_l}(\mathbf{a}))_{ii} \geq 0$, one gets

$$\sum_{j=1}^{n^2} \left| \left(D\Psi^{h_l}_{\hat{Y}_l}(\mathbf{a})\right)_{ji} \right| = 1 + h_l d_i \left(\mathbf{S}_{ii} + \left(\alpha - \sum_{i=2}^{9} |k_i|\right)\right) + h_l \sum_{j \neq i} d_j |\mathbf{S}_{ji}| \leq 1.$$

This inequality holds whenever

$$d_i \mathbf{S}_{ii} + \sum_{j \neq i} d_j \left|\mathbf{S}_{ji}\right| + \left(\alpha - \sum_{i=2}^{9} \left|k_i\right|\right) \leq 0,$$

which is always true since

$$\begin{aligned} d_i \mathbf{S}_{ii} + \sum_{j \neq i} d_j \left|\mathbf{S}_{ji}\right| + \left(\alpha - \sum_{i=2}^{9} \left|k_i\right|\right) &\leq d_i \left|\mathbf{S}_{ii}\right| + \sum_{j \neq i} d_j \left|\mathbf{S}_{ji}\right| + \left(\alpha - \sum_{i=2}^{9} \left|k_i\right|\right) \\ &\leq \left\|\mathbf{S}\right\|_1 - \sum_{i=2}^{9} \left|k_i\right| + \alpha \leq \alpha \leq 0. \end{aligned}$$

It hence only remains to study the case $\left(D\Psi^{h_l}_{\hat{Y}_l}(\mathbf{a})\right)_{ii} < 0$, which leads to

$$\sum_{j=1}^{n^2} \left|\left(D\Psi^{h_l}_{\hat{Y}_l}(\mathbf{a})\right)_{ji}\right| = -1 - h_l d_i \left(\mathbf{S}_{ii} + \left(\alpha - \sum_{i=2}^{9} \left|k_i\right|\right)\right) + h_l \sum_{j \neq i} d_j \left|\mathbf{S}_{ji}\right| \leq 1.$$

We move to a more stringent condition which is given by bounding $-h_l d_i \mathbf{S}_{ii} \leq h_l d_i |\mathbf{S}_{ii}| \leq h_l |\mathbf{S}_{ii}|$ so that we get

$$\begin{aligned} \sum_{j=1}^{n^2} \left|\left(D\Psi^{h_l}_{\hat{Y}_l}(\mathbf{a})\right)_{ji}\right| &\leq -1 + h_l \left|\mathbf{S}_{ii}\right| - h_l d_i \left(\alpha - \sum_{i=2}^{9} \left|k_i\right|\right) + h_l \sum_{j \neq i} \left|\mathbf{S}_{ji}\right| \\ &\leq -1 + h_l \left\|\mathbf{S}\right\|_1 - h_l d_i \left(\alpha - \sum_{i=2}^{9} \left|k_i\right|\right) \leq 1. \end{aligned}$$

This holds true when

$$h_l \leq \frac{2}{\left\|\mathbf{S}\right\|_1 - d_i \alpha + d_i \sum_{i=2}^{9} \left|k_i\right|}.$$

Now since $\|\mathbf{S}\|_1 \leq \sum_{i=2}^{9} |k_i|$, and $-d_i \alpha \in [0, -\alpha]$, we have

$$\frac{2}{2\sum_{i=1}^{9} \left|k_i\right| - \alpha} \leq \frac{2}{\left\|\mathbf{S}\right\|_1 - d_i \alpha + d_i \sum_{i=1}^{9} \left|k_i\right|},$$

which allows us to conclude that if

$$0 \leq h_l \leq \frac{2}{2\sum_{i=2}^{9} \left|k_i\right| - \alpha}$$

then the contractivity condition is satisfied. □

By definition of the vectorized ℓ^1 norm, one can hence conclude that Theorem 3.6 is equivalent to Theorem 3.3.

3.E Proof of the contractivity of the coupled dynamical system

In this section, we work with the coupled system

$$\begin{cases} \dot{F}(t) = -\mathcal{G}\left(A(t)\right)^{\top} \sigma\left(\mathcal{G}\left(A(t)\right) F(t)\, W(t)\right) W(t)^{\top} \\ \dot{A}(t) = \sigma\left(M\left(A(t)\right)\right), \end{cases} \tag{3.E.1}$$

where M is defined as in Appendix 3.B, and hence defines an equivariant system which is also contractive in vectorized ℓ^1 norm. Consider only the bounded time interval $t \in [0, \bar{T}]$, where we assume that the graph defined by $t \mapsto A(t)$ starts and remains connected. This guarantees $\mathcal{G}(A(t))F(t) = 0$ if and only if $F(t) \equiv F(0)$, and $F(0)$ has coinciding rows. We suppose this does not happen at time $t = 0$.

We assume that the activation function is $\sigma = \text{LeakyReLU}$. In the adjacency dynamical system, we set $\alpha < 0$. Furthermore, let W_l be non-singular. We now show that (3.E.1) provides a contractive continuous dynamical system in a suitable norm.

The focus is now on the time-independent case, i.e. only on X_l and Y_l, since the more general case follows naturally. We thus specify the expression in (3.E.1) for this setting:

$$\begin{cases} \dot{F}(t) = -\mathcal{G}\left(A(t)\right)^{\top} \sigma\left(\mathcal{G}\left(A(t)\right) F(t)\, W_l\right) W_l^{\top} =: X_l\left(F(t), A(t)\right) \\ \dot{A}(t) = \sigma\left(M\left(A(t)\right)\right) =: Y_l\left(A(t)\right). \end{cases} \tag{3.E.2}$$

Using the concepts described in Appendix 3.A, it is possible to prove that the two separate equations are strictly contracting in their respective norms, that is

$$\begin{aligned} \left\| F(t) - F_*(t) \right\|_F &\leq e^{-\nu_1 t} \left\| \mathbf{F}^{(0)} - \mathbf{F}_*^{(0)} \right\|_F, \ \nu_1 > 0, \\ \left\| \text{vec}\left(A(t)\right) - \text{vec}\left(A_*(t)\right) \right\|_1 &\leq e^{-\nu_2 t} \left\| \text{vec}\left(\mathbf{A}^{(0)}\right) - \text{vec}\left(\mathbf{A}_*^{(0)}\right) \right\|_1, \ \nu_2 > 0, \end{aligned}$$

where $t \in [0, \bar{T}]$, while $(F(t), A(t))$ and $(F_*(t), A_*(t))$ are solutions of (3.E.2), when considered separately, and starting respectively at $(\mathbf{F}^{(0)}, \mathbf{A}^{(0)})$ and $(\mathbf{F}_*^{(0)}, \mathbf{A}_*^{(0)})$. For more details on this result see [6]. We remark that in the first inequality, the $\mathbf{A}^{(0)}$ matrix is seen as a parameter, and hence does not evolve. These two conditions thus say that the two systems are strictly contracting when considered separately. In [51], the author shows that when these conditions hold and the mixed Jacobian matrix

$$J\left(\mathbf{F}, \mathbf{A}\right) = \frac{\partial \text{vec}\left(X_l\left(\mathbf{F}, \mathbf{A}\right)\right)}{\partial \text{vec}\left(\mathbf{A}\right)} \in \mathbb{R}^{nc \times n^2} \tag{3.E.3}$$

is bounded on a closed and convex set Ω in the norm

$$\left\| J\left(\mathbf{F},\mathbf{A}\right)\right\|_{1,2} = \max_{\substack{v\in\mathbb{R}^{n^2}\\ \|v\|_1=1}} \left\| J\left(\mathbf{F},\mathbf{A}\right) v\right\|_2 = \max_{j=1,\dots,n^2} \left\| J\left(\mathbf{F},\mathbf{A}\right)\mathbf{e}_j\right\|_2,$$

there is a pair of constants $m_1, m_2 > 0$ such that the system in (3.E.1) is contractive with respect to the weighted norm

$$\begin{aligned} &d_{m_1,m_2}\left(\left(\mathbf{F}^{(0)},\mathbf{A}^{(0)}\right),\left(\mathbf{F}_*^{(0)},\mathbf{A}_*^{(0)}\right)\right)\\ &= m_1\left\|\mathbf{F}^{(0)}-\mathbf{F}_*^{(0)}\right\|_F + m_2\left\|\mathrm{vec}\left(\mathbf{A}^{(0)}\right)-\mathrm{vec}\left(\mathbf{A}_*^{(0)}\right)\right\|_1. \end{aligned}$$

Now, $(F(t),A(t))$ and $(F_*(t),A_*(t))$ are solutions of (3.E.2) considered as a coupled system, i.e. solving jointly the two equations. Notice also that since $Y_l(\mathbf{0}_{n\times n}) = \mathbf{0}_{n\times n}$, and such system is contractive, it follows that any solution of (3.E.1) has $A(t)$ which is bounded uniformly in time by the norm of the initial condition $\|\mathrm{vec}(\mathbf{A}^{(0)})\|_1$. Similarly, we notice that for every $\mathbf{A}$, one also has $X_l(\mathbf{0}_{n\times c},\mathbf{A}) = \mathbf{0}_{n\times c}$ and hence also $\|F(t)\|_F$ is bounded by $\|\mathbf{F}^{(0)}\|_F$. To get more concise derivations, we use the conventional notation $[k] := \{1,\dots,k\}$ for indices. To compute the Jacobian and its norm, we first define $\mathcal{G}$ and $\mathcal{G}^\top$, as matrix operators, by specifying their components when acting on generic $\mathbf{F}\in\mathbb{R}^{n\times c}$ and $\mathbf{O}\in\mathbb{R}^{n\times n\times c}$:

$$\begin{aligned} \left(\mathcal{G}\left(\mathbf{A}\right)\mathbf{F}\right)_{ijk} &= \mathbf{A}_{ij}\left(\mathbf{F}_{ik}-\mathbf{F}_{jk}\right),\ i,j\in[n],\ k\in[c],\\ \left(\mathcal{G}\left(\mathbf{A}\right)^\top\mathbf{O}\right)_{ik} &= \sum_{j=1}^{n}\left(\mathbf{A}_{ij}\mathbf{O}_{ijk}-\mathbf{A}_{ji}\mathbf{O}_{jik}\right),\ i\in[n],\ k\in[c]. \end{aligned}$$

Here, $\mathcal{G}(\mathbf{A})^\top\mathbf{O}$ is obtained thanks to the relation

$$\mathrm{vec}\left(\mathcal{G}\left(\mathbf{A}\right)\mathbf{F}\right)^\top\mathrm{vec}\left(\mathbf{O}\right) = \mathrm{vec}\left(\mathbf{F}\right)^\top\mathrm{vec}\left(\mathcal{G}\left(\mathbf{A}\right)^\top\mathbf{O}\right).$$

We derive the desired Jacobian working on its components. First, let us compute

$$\frac{\partial\left(X_l\left(\mathbf{F}\right)\right)_{im}}{\partial\mathbf{A}_{rs}},$$

so that the norm we are interested in can be obtained as

$$\max_{r,s\in[n]}\sum_{i\in[n]}\sum_{m\in[c]}\left(\frac{\partial\left(X_l\left(\mathbf{F}\right)\right)_{im}}{\partial\mathbf{A}_{rs}}\right)^2.$$

We first denote $\hat{\mathbf{F}} := \mathbf{F}\mathbf{W} \in \mathbb{R}^{n\times c}$, and focus on

$$\frac{\partial\left(\mathcal{G}(\mathbf{A})^\top \sigma\left(\mathcal{G}(\mathbf{A})\hat{\mathbf{F}}\right)\right)_{ik}}{\partial \mathbf{A}_{rs}} = \frac{\partial}{\partial \mathbf{A}_{rs}}\left(\sum_{j\in[n]}\left[\mathbf{A}_{ij}\sigma\left(\mathcal{G}(\mathbf{A})\hat{\mathbf{F}}\right)_{ijk} - \mathbf{A}_{ji}\sigma\left(\mathcal{G}(\mathbf{A})\hat{\mathbf{F}}\right)_{jik}\right]\right)$$
$$= \frac{\partial}{\partial \mathbf{A}_{rs}}\left(\sum_{j\in[n]}\left[\mathbf{A}_{ij}\sigma\left(\mathbf{A}_{ij}\left(\hat{\mathbf{F}}_{ik}-\hat{\mathbf{F}}_{jk}\right)\right) - \mathbf{A}_{ji}\sigma\left(-\mathbf{A}_{ji}\left(\hat{\mathbf{F}}_{ik}-\hat{\mathbf{F}}_{jk}\right)\right)\right]\right).$$

We remark that for the case $\sigma(x) = \mathrm{ReLU}(x)$, one has $\sigma(x) - \sigma(-x) = x$, and hence when $\mathbf{A} = \mathbf{A}^\top$ we would have a simpler setting to deal with from now on. However, we aim to be general so we do not restrict to this case throughout the derivation. We can now proceed with the differentiation, and get

$$\begin{aligned}
&\delta_{ir}\sigma\left(\mathbf{A}_{is}\left(\hat{\mathbf{F}}_{ik}-\hat{\mathbf{F}}_{sk}\right)\right) - \delta_{is}\sigma\left(-\mathbf{A}_{ri}\left(\hat{\mathbf{F}}_{ik}-\hat{\mathbf{F}}_{rk}\right)\right)\\
&+\delta_{ir}\mathbf{A}_{is}\sigma'\left(\mathbf{A}_{is}\left(\hat{\mathbf{F}}_{ik}-\hat{\mathbf{F}}_{sk}\right)\right)\left(\hat{\mathbf{F}}_{ik}-\hat{\mathbf{F}}_{sk}\right)\\
&+\delta_{is}\mathbf{A}_{ri}\sigma'\left(-\mathbf{A}_{ri}\left(\hat{\mathbf{F}}_{ik}-\hat{\mathbf{F}}_{rk}\right)\right)\left(\hat{\mathbf{F}}_{ik}-\hat{\mathbf{F}}_{rk}\right),
\end{aligned}$$

where δ_{ij} is the Kronecker delta, which is 1 when $i = j$, and 0 otherwise. This means that the Jacobian matrix

$$\partial_{\mathbf{A}_{rs}}\left(\mathcal{G}(\mathbf{A})^\top \sigma\left(\mathcal{G}(\mathbf{A})\hat{\mathbf{F}}\right)\right) \in \mathbb{R}^{n\times c},$$

has all zero entries but those in the two rows r and s that are of the form

$$\begin{aligned}
&\mathbf{e}_r^\top \partial_{\mathbf{A}_{rs}}\left(\mathcal{G}(\mathbf{A})^\top \sigma\left(\mathcal{G}(\mathbf{A})\hat{\mathbf{F}}\right)\right)\\
&= \sigma\left(\mathcal{G}(\mathbf{A})_{rs}\left(\mathbf{e}_r-\mathbf{e}_s\right)^\top\hat{\mathbf{F}}\right) + \mathbf{A}_{rs}\mathrm{diag}\left(\sigma'\left(\mathbf{A}_{rs}\left(\mathbf{e}_r-\mathbf{e}_s\right)^\top\hat{\mathbf{F}}\right)\right)\left(\mathbf{e}_r-\mathbf{e}_s\right)^\top\hat{\mathbf{F}}\\
&\mathbf{e}_s^\top \partial_{\mathbf{A}_{rs}}\left(\mathcal{G}(\mathbf{A})^\top \sigma\left(\mathcal{G}(\mathbf{A})\hat{\mathbf{F}}\right)\right)\\
&= -\sigma\left(\mathbf{A}_{rs}\left(\mathbf{e}_r-\mathbf{e}_s\right)^\top\hat{\mathbf{F}}\right) - \mathbf{A}_{rs}\mathrm{diag}\left(\sigma'\left(\mathbf{A}_{rs}\left(\mathbf{e}_r-\mathbf{e}_s\right)^\top\hat{\mathbf{F}}\right)\right)\left(\mathbf{e}_r-\mathbf{e}_s\right)^\top\hat{\mathbf{F}}\\
&= -\mathbf{e}_r^\top \partial_{\mathbf{A}_{rs}}\left(\mathcal{G}(\mathbf{A})^\top \sigma\left(\mathcal{G}(\mathbf{A})\hat{\mathbf{F}}\right)\right).
\end{aligned}$$

As a consequence, when $r = s$, the matrix is the zero matrix. We conclude that

$$\begin{aligned}\left\|\frac{\partial \operatorname{vec}\left(X_l(\mathbf{FA})\right)}{\partial \operatorname{vec}(\mathbf{A})}\right\|_{1,2} &= \sqrt{2}\max_{r,s\in[n]}\left\|\mathbf{e}_r^\top \partial_{\mathbf{A}_{rs}}\left(\mathcal{G}(\mathbf{A})^\top \sigma\left(\mathcal{G}(\mathbf{A})\hat{\mathbf{F}}\right)\mathbf{W}^\top\right)\right\|_2 \\ &\le \sqrt{2}\,\|\mathbf{W}\|_2 \max_{r,s\in[n]}\left\|\mathbf{e}_r^\top \partial_{\mathbf{A}_{rs}}\left(\mathcal{G}(\mathbf{A})^\top \sigma\left(\mathcal{G}(\mathbf{A})\hat{\mathbf{F}}\right)\right)\right\|_2 \\ &\le 2\sqrt{2}\,\|\mathbf{W}\|_2^2 \max_{r,s\in[n]}\left\|\mathbf{A}_{rs}\left(\mathbf{e}_r-\mathbf{e}_s\right)^\top \mathbf{F}\right\|_2 \\ &= 2\sqrt{2}\,\|\mathbf{W}\|_2^2 \max_{r,s\in[n]}\left\|\left[\mathcal{G}(\mathbf{A})\mathbf{F}\right]_{rs}\right\|_2,\end{aligned}$$

where we assumed $\sigma(0) = 0$ and $\operatorname{Lip}(\sigma) \le 1$, as quite typical for neural network activation functions.

Following the proof of [51, Theorem 3], if we take $m_1, m_2 > 0$ such that

$$-\nu_2 - \frac{m_2}{m_1} 2\sqrt{2}\,\|\mathbf{W}\|_2^2 \max_{r,s\in[n]}\left\|\left[\mathcal{G}(\mathbf{A})\mathbf{F}\right]_{rs}\right\|_2 > 0$$

the coupled system in (3.E.1) is contractive with respect to the norm

$$\left\|(\mathbf{A},\mathbf{F})\right\|_{m_1,m_2} := m_1\,\|\operatorname{vec}(\mathbf{A})\|_1 + m_2\,\|\mathbf{F}\|_F.$$

3.F Related works: adversarial robustness via dynamical systems and Lipschitz regularity

The approach of improving the robustness of neural networks through Lipschitz constraints and techniques typical of the stability theory of dynamical systems has attracted much interest in recent years. This research direction has been investigated especially for classification tasks based on structured grids, i.e. images. For completeness in the presentation, we mention a few relevant contributions based on these insights and briefly comment on them. In [62], the authors work with evasion attacks and start by observing that not all input perturbations lead to changes in the predicted class. They then provide robustness guarantees based on the stability theory of dynamical systems. In [33], the authors again propose to enhance the robustness of a neural network based on Lyapunov stability. More precisely, they design a loss function promoting the closeness of output predictions to stable equilibria of the differential equation ruling the network, and also that these equilibria are as far as possible when different classes are considered. In [28], the authors work with neural-controlled ODEs and introduce a framework based on forward invariance. They propose a strategy to turn a desired function into a Lyapunov function for the ODE

driving the neural network, hence having its sublevel sets be forward invariant. This technique and some sampling strategies allow them to get certifiably robust models. In [49, 63], the authors exploit the connection between ResNet architectures and numerical methods, as in this manuscript, to design contractive residual neural networks based on the discretization of contractive dynamical systems. The experimental setup of both these works is again based on evasion attacks for image-based classification problems. Together with these mentioned works, which rely on dynamical systems theory, a big body of literature proposes to constrain the Lipschitz constant of the network as a way of reducing its sensitivity to input perturbations, see [53, 44, 37, 30].

3.G Lipschitz constant of the map $\mathcal{D}$

We now consider the bound on the Lipschitz constant of the network part $\mathcal{D}$ obtained by composing explicit Euler steps of dynamical systems. We consider the setting described in Appendix 3.E and assume all the steps $h_1, \dots, h_L$ to be small enough so that these explicit Euler steps are also contractive, see Theorems 3.2 and 3.3. These maps are contractive when considered in isolation, but, in general, their composition is not. We recall the system of differential equations to consider:

$$\begin{cases} \dot{F}(t) = X\left(t, F(t), A(t)\right) \\ \dot{A}(t) = Y\left(t, A(t)\right). \end{cases} \tag{3.G.1}$$

To specify the Lipschitz constant of the map $\mathcal{D}$, we introduce the two following maps:

$$X_{l,\mathbf{A}} : \mathbb{R}^{n\times c} \to \mathbb{R}^{n\times c},\ X_{l,\mathbf{A}}(\mathbf{F}) := X_l\left(\mathbf{F}, \mathbf{A}\right) = X\left(\tau_{l-1}, \mathbf{F}, \mathbf{A}\right), \tag{3.G.2}$$

$$X_{l,\mathbf{F}} : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times c},\ X_{l,\mathbf{F}}(\mathbf{A}) := X_l\left(\mathbf{F}, \mathbf{A}\right) = X\left(\tau_{l-1}, \mathbf{F}, \mathbf{A}\right). \tag{3.G.3}$$

Let us first recall the expression for $\mathcal{D}$, which is

$$\mathcal{D} = \mathcal{D}_L \circ \ldots \circ \mathcal{D}_1,$$

where

$$\mathcal{D}_l\left(\left(\mathbf{F}^{(l-1)}, \mathbf{A}^{(l-1)}\right)\right) = \left(\Psi^{h_l}_{X_{l,\mathbf{A}^{(l-1)}}}\left(\mathbf{F}^{(l-1)}\right), \Psi^{h_l}_{Y_l}\left(\mathbf{A}^{(l-1)}\right)\right), \quad l = 1, \dots, L.$$

We consider the two pairs of initial conditions $(\mathbf{F}^{(0)},\mathbf{A}^{(0)})$, $(\mathbf{F}_*^{(0)},\mathbf{A}_*^{(0)})$, and update first the $\mathbf{F}^{(0)}$ component with $\Psi^{h_1}_{X_{1,\mathbf{A}^{(0)}}}$, and $\mathbf{F}_*^{(0)}$ with $\Psi^{h_1}_{X_{1,\mathbf{A}_*^{(0)}}}$ to get

$$\mathbf{F}^{(1)} = \Psi^{h_1}_{X_{1,\mathbf{A}^{(0)}}}\left(\mathbf{F}^{(0)}\right) = \mathbf{F}^{(0)} + h_1 X_{1,\mathbf{A}^{(0)}}\left(\mathbf{F}^{(0)}\right) \tag{3.G.4}$$

$$\mathbf{F}_*^{(1)} = \Psi^{h_1}_{X_{1,\mathbf{A}_*^{(0)}}}\left(\mathbf{F}_*^{(0)}\right) = \mathbf{F}_*^{(0)} + h_1 X_{1,\mathbf{A}_*^{(0)}}\left(\mathbf{F}_*^{(0)}\right) \tag{3.G.5}$$

$$\begin{aligned}
&\left\|\mathbf{F}^{(1)} - \mathbf{F}_*^{(1)}\right\|_F \qquad (3.G.6)\\
&= \left\|\Psi^{h_1}_{X_{1,\mathbf{A}^{(0)}}}\left(\mathbf{F}^{(0)}\right) + \left(\Psi^{h_1}_{X_{1,\mathbf{A}_*^{(0)}}}\left(\mathbf{F}^{(0)}\right) - \Psi^{h_1}_{X_{1,\mathbf{A}_*^{(0)}}}\left(\mathbf{F}^{(0)}\right)\right) - \Psi^{h_1}_{X_{1,\mathbf{A}_*^{(0)}}}\left(\mathbf{F}_*^{(0)}\right)\right\|_F\\
&\leq \left\|\Psi^{h_1}_{X_{1,\mathbf{A}^{(0)}}}\left(\mathbf{F}^{(0)}\right) - \Psi^{h_1}_{X_{1,\mathbf{A}_*^{(0)}}}\left(\mathbf{F}^{(0)}\right)\right\|_F + \left\|\mathbf{F}^{(0)} - \mathbf{F}_*^{(0)}\right\|_F\\
&\leq h_1\left\|X_{1,\mathbf{F}^{(0)}}\left(\mathbf{A}^{(0)}\right) - X_{1,\mathbf{F}^{(0)}}\left(\mathbf{A}_*^{(0)}\right)\right\|_F + \varepsilon_1\\
&\leq h_1 \mathrm{Lip}\left(X_{1,\mathbf{F}^{(0)}}\right)\left\|\mathrm{vec}\left(\mathbf{A}^{(0)}\right) - \mathrm{vec}\left(\mathbf{A}_*^{(0)}\right)\right\|_1 + \varepsilon_1\\
&\leq h_1 \mathrm{Lip}\left(X_{1,\mathbf{F}^{(0)}}\right)\varepsilon_2 + \varepsilon_1. \qquad (3.G.7)
\end{aligned}$$

By $\mathrm{Lip}(X_{1,\mathbf{F}^{(0)}})$ we refer to the Lipschitz constant of the map $X_{1,\mathbf{F}^{(0)}} : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times c}$, where the first space has the vectorized ℓ^1 norm, and the second has the Frobenius norm.

Then one can update the adjacency matrices $\mathbf{A}^{(0)}$ and $\mathbf{A}_*^{(0)}$ to

$$\mathbf{A}^{(1)} = \Psi^{h_1}_{Y_1}\left(\mathbf{A}^{(0)}\right),\ \mathbf{A}_*^{(1)} = \Psi^{h_1}_{Y_1}\left(\mathbf{A}_*^{(0)}\right),$$

for which we have already proven $\|\mathrm{vec}(\mathbf{A}^{(1)}) - \mathrm{vec}(\mathbf{A}_*^{(1)})\|_1 \leq \varepsilon_2$. To get the general form of the Lipschitz constant of $\mathcal{D}$, we update again the features so it is easier to generalize:

$$\mathbf{F}^{(2)} = \Psi^{h_2}_{X_{2,\mathbf{A}^{(1)}}}\left(\mathbf{F}^{(1)}\right) = \mathbf{F}^{(1)} + h_2 X_{2,\mathbf{A}^{(1)}}\left(\mathbf{F}^{(1)}\right) \tag{3.G.8}$$

$$\mathbf{F}_*^{(2)} = \Psi^{h_2}_{X_{2,\mathbf{A}_*^{(1)}}}\left(\mathbf{F}_*^{(1)}\right) = \mathbf{F}_*^{(1)} + h_2 X_{2,\mathbf{A}_*^{(1)}}\left(\mathbf{F}_*^{(1)}\right) \tag{3.G.9}$$

$$\left\|\mathbf{F}^{(2)} - \mathbf{F}_*^{(2)}\right\|_F \leq h_2 \mathrm{Lip}\left(X_{2,\mathbf{F}^{(1)}}\right)\varepsilon_2 + h_1 \mathrm{Lip}\left(X_{1,\mathbf{F}^{(0)}}\right)\varepsilon_2 + \varepsilon_1. \tag{3.G.10}$$

This leads to the general bound on the expansivity of $\mathcal{D}$ when it is composed

of L layers, which is

$$\begin{aligned} d\left(\mathcal{D}\left(\mathbf{F}^{(0)},\mathbf{A}^{(0)}\right),\mathcal{D}\left(\mathbf{F}_*^{(0)},\mathbf{A}_*^{(0)}\right)\right) &:= \left\|\mathrm{vec}(\mathbf{A}^{(L)})-\mathrm{vec}(\mathbf{A}_*^{(L)})\right\|_1 + \left\|\mathbf{F}^{(L)}-\mathbf{F}_*^{(L)}\right\|_F \\ &\leq \varepsilon_1+\varepsilon_2\left(1+\sum_{i=1}^{L}\mathrm{Lip}\left(X_{i,\mathbf{F}^{(i-1)}}\right)h_i\right) \\ &=: \varepsilon_1+c\left(h_1,\ldots,h_L\right)\varepsilon_2. \end{aligned} \tag{3.G.11}$$

We remark that in case the adjacency matrix is not perturbed, i.e., $\varepsilon_2 = 0$, the map $\mathcal{D}$ can be controlled by the perturbation magnitude to the feature matrix, i.e., ε_1. On the other hand, even if the features are not perturbed, i.e. $\varepsilon_1 = 0$, the feature updates can not be bounded simply with ε_2, since their update depends on different adjacency matrices. We have already commented on this aspect in Section 3.4 since this interconnection is also the reason why we have proposed to jointly update the feature and the adjacency matrices. The important aspect of this analysis is that constraining the map $\mathcal{D}$ so this contractive setup occurs allows getting the bound in (3.G.11), which quantifies how sensitive the network is to perturbations. We also remark that the derivation in the previous section allows one to get practical bounds on the Lipschitz constants $\mathrm{Lip}(X_{i,\mathbf{F}^{(i)}})$.

3.H Architecture

We describe in Algorithm 1 the architecture of CSGNN.

3.I Datasets

We provide the statistics of the datasets used in our experiments in Table 3.I.1.

3.J Hyperparameters

All the hyperparameters were determined by grid search, and the ranges and sampling distributions are provided in Table 3.J.1.

Algorithm 1 CSGNN Architecture

Input: Attacked node features $\mathbf{F}_* \in \mathbb{R}^{n \times c_{\text{in}}}$ and adjacency matrix $\mathbf{A}_* \in \{0,1\}^{n \times n}$.

Output: Predicted node labels $\tilde{\mathbf{Y}} \in \mathbb{R}^{n \times c_{\text{out}}}$.

procedure CSGNN
  $\mathbf{F}_* \leftarrow \text{Dropout}\left(\mathbf{F}_*, p\right)$
  $\mathbf{F}_*^{(0)} = \mathcal{K}\left(\mathbf{F}_*\right)$; $\mathbf{A}_*^{(0)} = \mathbf{A}_*$
  for $l = 1 \ldots L$ **do**
    $\mathbf{F}_*^{(l-1)} \leftarrow \text{Dropout}\left(\mathbf{F}_*^{(l-1)}, p\right)$
    Node Feature Update: $\mathbf{F}_*^{(l)} = \Psi_{X_l}^{h_l}\left(\mathbf{F}_*^{(l-1)}, \mathbf{A}_*^{(l-1)}\right)$
    Adjacency Update: $\mathbf{A}_*^{(l)} = \Psi_{Y_l}^{h_l}\left(\mathbf{A}_*^{(l-1)}\right)$
  end for
  $\mathbf{F}_*^{(L)} \leftarrow \text{Dropout}\left(\mathbf{F}_*^{(L)}, p\right)$
  $\tilde{\mathbf{Y}} = \mathcal{P}\left(\mathbf{F}_*^{(L)}\right)$
  Return $\tilde{\mathbf{Y}}$
end procedure

3.K Experimental Results

Results on Pubmed. We now provide our results on the Pubmed [42] dataset, with three types of attacks: (i) non-targeted using metattack, reported in Table 3.K.1, (ii) targeted attack using nettack in Figure 3.K.1, and (iii) random adjacency matrix attack in Figure 3.K.2. Those experiments are done under the same settings as those in Section 3.5 in the main paper. Overall, we see that our CSGNN achieves similar or better results compared with other baselines. Specifically, we see that CSGNN outperforms all the considered baselines under targeted attack (using nettack), and similar performance when no perturbations occur, as can be depicted in Figure 3.K.1.

Absolute performance results on Unit tests. In Section 3.5.2, we provide the obtained relative node classification accuracy (%) with respect to the accuracy of GCN. Here, we also provide the absolute results, for an additional perspective on the performance of CSGNN. Our results are reported in Figure 3.K.3.

Enforcing contractive node dynamics improves baseline performance. As discussed in Section 3.4.2, we draw inspiration from contractive dynamical systems and, therefore, propose a contractivity-inspired node feature dynami-

Dataset	$\mathbf{N_{LCC}}$	$\mathbf{E_{LCC}}$	**Classes**	**Features**
Cora [40]	2,485	5,069	7	1,433
Citeseer [47]	2,110	3,668	6	3,703
Polblogs [1]	1,222	16,714	2	/
Pubmed [42]	19,717	44,338	3	500

Table 3.I.1: Datasets Statistics. Following [67, 68], we consider only the largest connected component (LCC).

Hyperparameter	Range	Distribution
input/output embedding learning rate	$[10^{-5}, 10^{-2}]$	uniform
node dynamics learning rate	$[10^{-5}, 10^{-2}]$	uniform
adjacency dynamics learning rate	$[10^{-5}, 10^{-2}]$	uniform
input/output embedding weight decay	$[5 \cdot 10^{-8}, 5 \cdot 10^{-2}]$	log uniform
node dynamics weight decay	$[5 \cdot 10^{-8}, 5 \cdot 10^{-2}]$	log uniform
adjacency dynamics weight decay	$[5 \cdot 10^{-8}, 5 \cdot 10^{-2}]$	log uniform
input/output embedding dropout	$[0, 0.6]$	uniform
node dynamics dropout	$[0, 0.6]$	uniform
share weights between time steps	{yes, no}	discrete uniform
step size h	$[10^{-2}, 1]$	log uniform
adjacency contractivity parameter α	$[-2, 0]$	uniform
#layers L	$\{2, 3, 4, 5\}$	discrete uniform
#channels c	$\{8, 16, 32, 64, 128\}$	discrete uniform

Table 3.J.1: Hyperparameter ranges

cal system. For our results in the main paper, in Section 3.5, we use a more general definition of the node dynamical system that can admit both contractive and non-contractive dynamics in a data-driven fashion that generalizes the contractive behavior described in Theorem 3.2. To motivate our choice and inspiration from such systems, we now show that by enforcing contractive dynamics only (i.e., ensuring $\tilde{\mathbf{K}}_l$ is positive definite), improved results are also achieved, in addition to our results with CSGNN in the main paper. To this end, we will denote the *Enforced* CSGNN variant of our method by ECSGNN. We present the performance of ECSGNN under the non-targeted metattack in Table 3.K.2, targeted nettack in Figure 3.K.4, and random attacks in Figure 3.K.5. Overall, we see that ECSGNN typically offers improved performance compared to several baselines and, in some cases, outperforms all of the considered models. Also, we find that its extension, non-enforced CSGNN tends to yield further performance improvements, as shown in the main paper.

Dataset	Ptb Rate (%)	0	5	10	15	20	25
Pubmed	GCN	87.19±0.09	83.09±0.13	81.21±0.09	78.66±0.12	77.35±0.19	75.50±0.17
	GAT	83.73±0.40	78.00±0.44	74.93±0.38	71.13±0.51	68.21±0.96	65.41±0.77
	RGCN	86.16±0.18	81.08±0.20	77.51±0.27	73.91±0.25	71.18±0.31	67.95±0.15
	GCN-Jaccard	87.06±0.06	86.39±0.06	85.70±0.07	84.76±0.08	83.88±0.05	83.66±0.06
	GCN-SVD	83.44±0.21	83.41±0.15	83.27±0.21	83.10±0.18	83.01±0.22	82.72±0.18
	Pro-GNN-fs	87.33±0.18	**87.25±0.09**	**87.25±0.09**	**87.20±0.09**	87.09±0.10	86.71±0.09
	Pro-GNN	87.26±0.23	87.23±0.13	87.21±0.13	87.20±0.15	**87.15±0.15**	**86.76±0.19**
	CSGNN	**87.36±0.02**	87.16±0.10	87.08±0.09	87.06±0.08	86.59±0.18	86.63±0.08

Table 3.K.1: Node classification performance (accuracy±std) under non-targeted attack (metattack) on the Pubmed dataset with varying perturbation rates.

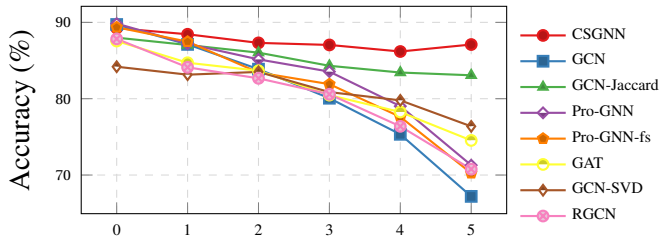


Figure 3.K.1: Node classification accuracy (%) on the Pubmed dataset using nettack as an attack method. The horizontal axis describes the number of perturbations per node.

Our conclusion from this experiment is that node feature contractivity helps to improve robustness to adversarial attacks.

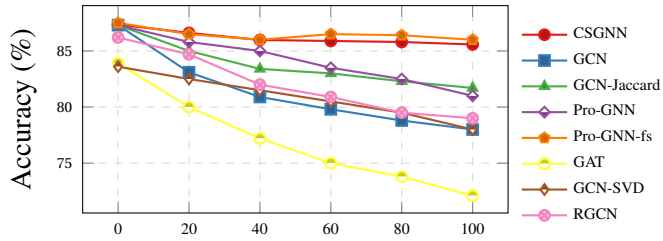
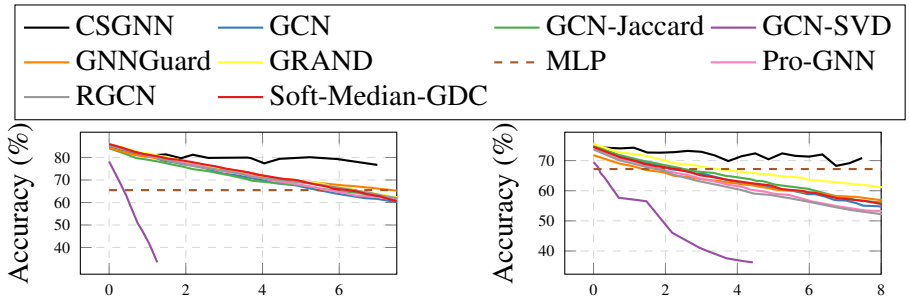


Figure 3.K.2: Node classification accuracy (%) on the Pubmed dataset with a random adjacency matrix attack. The horizontal axis describes the attack percentage.

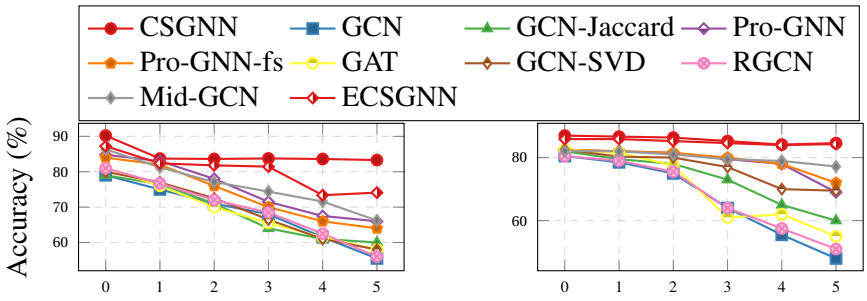


(a) Cora-ML **(b)** Citeseer

Figure 3.K.3: Absolute results version of Figure 3.5.2. The horizontal axis describes the attack budget (%) as defined in [41].

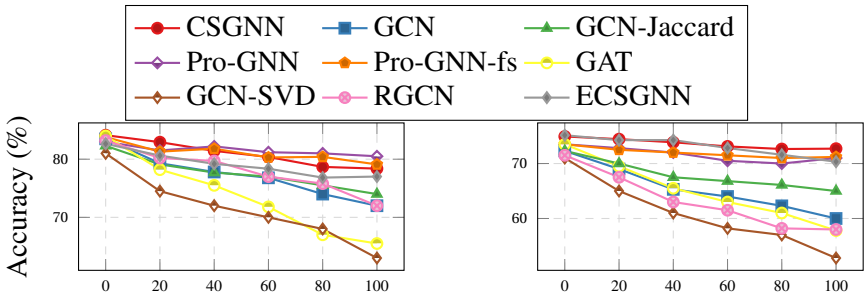
Dataset	Ptb Rate (%)	0	5	10	15	20	25
Cora	GCN	83.50±0.44	76.55±0.79	70.39±1.28	65.10±0.71	59.56±2.72	47.53±1.96
	GAT	83.97±0.65	80.44±0.74	75.61±0.59	69.78±1.28	59.94±0.92	54.78±0.74
	RGCN	83.09±0.44	77.42±0.39	72.22±0.38	66.82±0.39	59.27±0.37	50.51±0.78
	GCN-Jaccard	82.05±0.51	79.13±0.59	75.16±0.76	71.03±0.64	65.71±0.89	60.82±1.08
	GCN-SVD	80.63±0.45	78.39±0.54	71.47±0.83	66.69±1.18	58.94±1.13	52.06±1.19
	Pro-GNN-fs	83.42±0.52	82.78±0.39	77.91±0.86	76.01±1.12	68.78±5.84	56.54±2.58
	Pro-GNN	82.98±0.23	82.27±0.45	79.03±0.59	76.40±1.27	73.32±1.56	69.72±1.69
	Mid-GCN	84.61±0.46	82.94±0.59	80.14±0.86	77.77±0.75	76.58±0.29	72.89±0.81
	CSGNN	84.12±0.31	82.20±0.65	80.43±0.74	79.32±1.04	77.47±1.22	74.46±0.99
	ECSGNN	82.79±0.33	80.59±0.61	79.19±0.81	76.29±0.96	73.88±0.84	72.27±0.78
Citeseer	GCN	71.96±0.55	70.88±0.62	67.55±0.89	64.52±1.11	62.03±3.49	56.94±2.09
	GAT	73.26±0.83	72.89±0.83	70.63±0.48	69.02±1.09	61.04±1.52	61.85±1.12
	RGCN	71.20±0.83	70.50±0.43	67.71±0.30	65.69±0.37	62.49±1.22	55.35±0.66
	GCN-Jaccard	72.10±0.63	70.51±0.97	69.54±0.56	65.95±0.94	59.30±1.40	59.89±1.47
	GCN-SVD	70.65±0.32	68.84±0.72	68.87±0.62	63.26±0.96	58.55±1.09	57.18±1.87
	Pro-GNN-fs	73.26±0.38	73.09±0.34	72.43±0.52	70.82±0.87	66.19±2.38	66.40±2.57
	Pro-GNN	73.28±0.69	72.93±0.57	72.51±0.75	72.03±1.11	70.02±2.28	68.95±2.78
	Mid-GCN	74.17±0.28	74.31±0.42	73.59±0.29	73.69±0.29	71.51±0.83	69.12±0.72
	CSGNN	74.93±0.52	74.91±0.33	73.95±0.35	73.82±0.61	73.01±0.77	72.94±0.56
	ECSGNN	75.01±0.28	74.97±0.38	73.97±0.29	73.67±0.45	72.92±0.97	72.89±0.90

Table 3.K.2: Node classification performance (accuracy±std) of ECSGNN and other baselines, under non-targeted attack (metattack) with varying perturbation rates.



(a) Cora **(b)** Citeseer

Figure 3.K.4: Node classification accuracy (%) of ECSGNN and other baselines, under a targeted attack generated by nettack. The horizontal axis describes the number of perturbations per node.



(a) Cora **(b)** Citeseer

Figure 3.K.5: Node classification accuracy (%) of ECSGNN and other baselines, under a random adjacency matrix attack. The horizontal axis describes the attack percentage.

Learning contractive adjacency dynamics is beneficial. As our CSGNN is composed of two learnable coupled dynamical systems that evolve both the node features and the adjacency matrix, it is interesting to quantify the importance of learning the adjacency matrix dynamics. Therefore, we now report the node classification accuracy (%) on the Cora and Citeseer datasets, under three different attack settings - non-targeted with metattack, targeted with nettack, and a random adjacency matrix attack. We choose the strongest attack under each setting, out of our experiments in Section 3.5.2. Namely, for metattack, we choose the highest perturbation rate of 25%, for nettack we choose the maximal number of node perturbations of 5, and for random adjacency matrix attack we randomly add edges that amount to 100% of the edges in the original, clean graph. We report the results in Table 3.K.3. We denote the CSGNN variant that does not employ an adjacency dynamical system by $\text{CSGNN}_{\text{noAdj}}$. As can be seen from the table, there is a positive impact when adding the learnable adjacency matrix component to our CSGNN, highlighting its importance to the learned dynamical system.

Method	Cora			Citeseer		
	nettack	metattack	random	nettack	metattack	random
$\text{CSGNN}_{\text{noAdj}}$	81.90	70.25	77.19	82.20	70.17	71.28
CSGNN	83.29	74.46	78.38	84.60	72.94	72.70

Table 3.K.3: The influence of learning the adjacency dynamical system $\Psi_{Y_l}^{h_l}$. The results show the node classification accuracy (%) with and without learning the adjacency dynamical system.

Comparison with GNNGuard. We now provide a comparison of our CSGNN with GNNGuard [64]. We report results, both on Cora and Citeseer, for Metattack with a 20% perturbation rate and for 5 targeted nodes using Nettack, as reported in [64]. These results further highlight the significance of our method, given that in most cases, CSGNN outperforms GNNGuard.

Method	Cora		Citeseer	
	metattack	nettack	metattack	nettack
GNNGuard	72.20	77.50	71.10	86.50
CSGNN (Ours)	77.47	83.20	73.00	84.60

Table 3.K.4: Comparison of our proposed GNN architecture with GNNGuard, based on the experimental setup proposed in [64]. The results show the node classification accuracy (%). Metattack is considered with a 20% perturbation rate, and Nettack with 5 targeted nodes.

Ablation study for the number of network layers. One of the hyperparam-

eters of the networks we consider is the number of network layers. We now report an ablation study of the performance of CSGNN versus the number of layers on the Cora dataset. The study has a varying number of layers, between 1 and 32. To provide a comprehensive understanding, we provide results both on 5% and 25% perturbation rate attacks using metattack. Our results are reported in Table 3.K.5, and they show that CSGNN achieves optimal results in the range of 2-5 layers, which is also what we used as a hyperparameter range in our experiments.

Number of Layers	1	2	3	4	5	8	16	32
5% Ptb. Rate	82.57	83.98	84.12	84.10	84.12	84.08	83.99	84.05
25% Ptb. Rate	70.06	71.88	73.93	74.46	74.33	74.21	74.30	74.01

Table 3.K.5: Ablation study over the number of layers for CSGNN. The tests are done over the Cora dataset, attacked with metattack.

Comparison with GARNET. We now provide a comparison of the CSGNN with GARNET proposed in [16]. The reported results correspond to the best results shown in [16]. The tables report test node classification accuracies, in %, where they were available in the original paper. As can be seen from Tables 3.K.6 and 3.K.7, our method performs competitively with GARNET on both datasets.

Ptb rate (%)	0	5	10	15	20	25
CSGNN	87.36	87.16	87.08	87.06	86.59	86.63
GARNET	86.86	N/A	86.24	N/A	85.69	N/A

Table 3.K.6: Metattack on Pubmed. Comparison between CSGNN and GARNET [16].

Ptb rate (%)	0	5	10	15	20	25
CSGNN	84.12	82.20	80.43	79.32	77.47	74.46
GARNET	82.67	N/A	82.17	N/A	81.34	N/A

Table 3.K.7: Metattack on Cora. Comparison between CSGNN and GARNET [16].

Comparison with HANG. This paragraph reports the comparison of CSGNN with HANG, presented in [65]. As for GARNET, we only include the results available in the original paper of HANG. Tables 3.K.8 and 3.K.9 display competitive results of CSGNN with respect to the best ones reported in [65].

Ptb rate (%)	0	5	10	15	20	25
CSGNN	87.36	87.16	87.08	87.06	86.59	86.63
HANG	85.23	85.12	85.17	85.15	85.20	85.06

Table 3.K.8: Metattack on Pubmed. Comparison between CSGNN and HANG [65].

Ptb rate (%)	0	5	10	15	20	25
CSGNN	95.87	95.79	93.21	92.08	90.10	87.37
HANG	94.63	94.38	92.46	90.85	89.19	86.89

Table 3.K.9: Metattack on Polblogs. Comparison between CSGNN and HANG [65].

3.L Complexity and Runtimes

This appendix analyzes the complexity and runtimes of our proposed graph neural network, accounting for the additional overhead cost due to updating the adjacency matrix. Inference times and memory consumption are based on experiments run on the Cora dataset with an Nvidia RTX-3090 GPU and 24GB of memory. In theory, the added runtime complexity for a CSGNN adjacency matrix learning layer is of $O(9 \cdot n^2)$ where n is the number of nodes. Thus, assuming L layers, the overall complexity of a CSGNN network is $O(L(n \cdot c^2 + m \cdot c + 9 \cdot n^2))$, whereas node-based ODE systems usually are of runtime complexity $O(L(n \cdot c^2 + m \cdot c))$. We recall that n is the number of nodes, m is the number of edges in the graph, and the term of c^2 stems from the channel mixing term in (3.4.3). The factor of 9 stems from the 9 parameters to be learned in Equation (3.B.1). Below, we report the runtimes and memory consumption of our CSGNN and compare it with $\text{CSGNN}_{\text{noAdj}}$ (that is, CSGNN without the adjacency matrix update), and Pro-GNN for reference. These experiments were run on the Cora dataset with networks with 64 channels and 2 layers. It can be seen that both CSGNN and Pro-GNN require more resources. However, such an approach offers improved performance.

Also, we note that while both our CSGNN and Pro-GNN propose methods to evolve the adjacency matrix, our CSGNN shows significantly lower training computational time, due to our solution taking the form of a learned neural ODE system for the adjacency matrix, while Pro-GNN solves an optimization problem with feedback to the downstream task (which is also an end-to-end solution, although different than ours). For inference, Pro-GNN uses a fixed adjacency matrix that is found by training, while our CSGNN evolves the input adjacency matrix using the learned network. This can also be seen as an

Method	Time (ms)		Memory (MB)	Classification accuracy (%)	
	Training	Inference		metattack	nettack
Pro-GNN	1681.17	1.01	1989	69.72	66.21
$\text{CSGNN}_{\text{noAdj}}$	3.29	1.32	891	70.25	81.90
CSGNN	9.24	5.96	1623	74.46	83.29

Table 3.L.1: Comparison of our proposed GNN architecture with Pro-GNN in terms of runtime in milliseconds (measured per epoch) and memory consumption in megabytes, as well as the accuracy (%) of the models under different attacks. The cost of the training refers to one epoch. Metattack is considered with a 25% perturbation rate, and Nettack targets 5 nodes.

advantage of CSGNN, as it can be used for inference on different kinds of attacks and, therefore, can potentially generalize to different attacks, and this is a future research direction.

To conclude, we remark that the investigation of techniques that allow for a reduction of the computational costs in the updates of the adjacency matrices is an important and interesting topic on its own that is left for future work.

Bibliography

[1] Lada A Adamic and Natalie Glance. The political blogosphere and the 2004 US election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, pages 36–43, 2005. 127

[2] Uri Alon and Eran Yahav. On the Bottleneck of Graph Neural Networks and its Practical Implications. In *International Conference on Learning Representations*, 2021. 95

[3] Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P. Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019. 95

[4] Aleksandar Bojchevski and Stephan Günnemann. Adversarial Attacks on Node Embeddings via Graph Poisoning. In *International Conference on Machine Learning*, pages 695–704. PMLR, 2019. 97

[5] Johannes Brandstetter, Daniel E. Worrall, and Max Welling. Message Passing Neural PDE Solvers. In *International Conference on Learning Representations*, 2022. 95

[6] Francesco Bullo. *Contraction Theory for Dynamical Systems*. Kindle Direct Publishing, 1.1 edition, 2023. 21, 95, 98, 110, 119, 295, 296

[7] Chen Cai and Yusu Wang. A Note on Over-Smoothing for Graph Neural Networks. *arXiv preprint arXiv:2006.13318*, 2020. 95

[8] Elena Celledoni, Davide Murari, Brynjulf Owren, Carola-Bibiane Schönlieb, and Ferdia Sherry. Dynamical Systems–Based Neural Networks. *SIAM Journal on Scientific Computing*, 45(6):A3071–A3094, 2023. 3, 95, 145, 146

[9] Benjamin Paul Chamberlain, James Rowbottom, Maria Gorinova, Stefan Webb, Emanuele Rossi, and Michael M Bronstein. GRAND: Graph Neu-

ral Diffusion. In *International Conference on Machine Learning (ICML)*, pages 1407–1418. PMLR, 2021. 95, 99, 101, 113

[10] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural Ordinary Differential Equations. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6572–6583, 2018. 95

[11] Yongqiang Chen, Han Yang, Yonggang Zhang, Kaili Ma, Tongliang Liu, Bo Han, and James Cheng. Understanding and Improving Graph Injection Attack by Promoting Unnoticeability. *arXiv preprint arXiv:2202.08057*, 2022. 96

[12] Jeongwhan Choi, Seoyoung Hong, Noseong Park, and Sung-Bae Cho. GREAD: Graph Neural Reaction-Diffusion Equations. *arXiv preprint arXiv:2211.14208*, 2022. 95

[13] Gabriele Corso, Hannes Stärk, Bowen Jing, Regina Barzilay, and Tommi S. Jaakkola. DiffDock: Diffusion Steps, Twists, and Turns for Molecular Docking. In *The Eleventh International Conference on Learning Representations*, 2023. 94

[14] Alexander Davydov, Anton V Proskurnikov, and Francesco Bullo. Non-Euclidean Contraction Analysis of Continuous-Time Neural Networks. *arXiv preprint arXiv:2110.08298*, 2021. 110

[15] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016. 94

[16] Chenhui Deng, Xiuyu Li, Zhuo Feng, and Zhiru Zhang. GARNET: Reduced-Rank Topology Learning for Robust and Scalable Graph Neural Networks. In *Learning on Graphs Conference*, pages 3–1. PMLR, 2022. 106, 132

[17] Moshe Eliasof, Eldad Haber, and Eran Treister. PDE-GCN: Novel Architectures for Graph Neural Networks Motivated by Partial Differential Equations. *Advances in Neural Information Processing Systems*, 34, 2021. 9, 95, 99, 101, 113, 145

[18] Negin Entezari, Saba A Al-Sayouri, Amirali Darvishzadeh, and Evangelos E Papalexakis. All You Need Is Low (Rank): Defending Against Adversarial Attacks on Graphs. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 169–177, 2020. 96, 100, 106

[19] Herbert Federer. *Geometric Measure Theory*. Springer, 2014. 110

[20] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. Graph Random Neural Networks for Semi-Supervised Learning on Graphs. *Advances in neural information processing systems*, 33:22092–22103, 2020. 106

[21] Simon Geisler, Tobias Schmidt, Hakan Sirin, Daniel Zügner, Aleksandar Bojchevski, and Stephan Günnemann. Robustness of Graph Neural Networks at Scale. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 7637–7649, 2021. 106

[22] Francesco Di Giovanni, James Rowbottom, Benjamin Paul Chamberlain, Thomas Markovich, and Michael M. Bronstein. Understanding convolution on graphs via energies. *Transactions on Machine Learning Research*, 2023. 101, 102

[23] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 94

[24] Thomas Hakon Gronwall. Note on the Derivatives with Respect to a Parameter of the Solutions of a System of Differential Equations. *Annals of Mathematics*, pages 292–296, 1919. 111

[25] Stephan Günnemann. Graph Neural Networks: Adversarial Robustness. In Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 149–176. Springer Nature Singapore, Singapore, 2022. 94, 97, 98

[26] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, December 2017. 95, 99

[27] Jincheng Huang, Lun Du, Xu Chen, Qiang Fu, Shi Han, and Dongmei Zhang. Robust Mid-Pass Filtering Graph Convolutional Networks. In

Ying Ding, Jie Tang, Juan F. Sequeda, Lora Aroyo, Carlos Castillo, and Geert-Jan Houben, editors, *Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*, pages 328–338. ACM, 2023. 96, 106

[28] Yujia Huang, Ivan Dario Jimenez Rodriguez, Huan Zhang, Yuanyuan Shi, and Yisong Yue. FI-ODE: Certifiably Robust Forward Invariance in Neural ODEs. *arXiv preprint arXiv:2210.16940*, 2022. 122

[29] Zijie Huang, Yizhou Sun, and Wei Wang. Coupled Graph ODE for Learning Interacting System Dynamics. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, 2021. 96

[30] Todd Huster, Cho-Yu Jason Chiang, and Ritu Chadha. Limitations of the Lipschitz Constant as a Defense Against Adversarial Examples. In *ECML PKDD 2018 Workshops: Nemesis 2018, UrbReas 2018, SoGood 2018, IWAISe 2018, and Green Data Mining 2018, Dublin, Ireland, September 10-14, 2018, Proceedings 18*, pages 16–29. Springer, 2019. 123

[31] Yaning Jia, Dongmian Zou, Hongfei Wang, and Hai Jin. Enhancing Node-Level Adversarial Defenses by Lipschitz Regularization of Graph Neural Networks. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, page 951–963, New York, NY, USA, 2023. Association for Computing Machinery. 96

[32] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph Structure Learning for Robust Graph Neural Networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 66–74, 2020. 96, 100, 105, 106, 107, 108

[33] Qiyu Kang, Yang Song, Qinxu Ding, and Wee Peng Tay. Stable Neural ODE with Lyapunov-Stable Equilibrium Points for Defending Against Adversarial Attacks. *Advances in Neural Information Processing Systems*, 34:14925–14937, 2021. 122

[34] Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014. 3, 106

[35] Thomas N Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. *International Conference on Learning Representations (ICLR)*, 2017. 94, 96, 105

[36] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion Improves Graph Learning. In Hanna M. Wallach, Hugo

Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 13333–13345, 2019. 106

[37] Sheng Liu, Xiao Li, Yuexiang Zhai, Chong You, Zhihui Zhu, Carlos Fernandez-Granda, and Qing Qu. Convolutional Normalization: Improving Deep Convolutional Network Robustness and Training. *Advances in neural information processing systems*, 34:28919–28928, 2021. 123

[38] Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. PDE-net: Learning PDEs from data, 2018. 95

[39] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and Equivariant Graph Networks. *ICLR*, 2018. 102

[40] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the Construction of Internet Portals with Machine Learning. *Information Retrieval*, 3(2):127–163, 2000. 127

[41] Felix Mujkanovic, Simon Geisler, Stephan Günnemann, and Aleksandar Bojchevski. Are Defenses for Graph Neural Networks Robust? *Advances in Neural Information Processing Systems*, 35, 2022. 97, 98, 105, 108, 129

[42] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and U Edu. Query-driven Active Surveying for Collective Classification. In *10th International Workshop on Mining and Learning with Graphs*, volume 8, page 1, 2012. 126, 127

[43] Kenta Oono and Taiji Suzuki. Graph Neural Networks Exponentially Lose Expressive Power for Node Classification. In *International Conference on Learning Representations*, 2020. 95

[44] Patricia Pauli, Anne Koch, Julian Berberich, Paul Kohler, and Frank Allgöwer. Training robust neural networks using Lipschitz bounds. *IEEE Control Systems Letters*, 6:121–126, 2021. 123

[45] T Konstantin Rusch, Ben Chamberlain, James Rowbottom, Siddhartha Mishra, and Michael Bronstein. Graph-Coupled Oscillator Networks. In *International Conference on Machine Learning*. PMLR, 2022. 95

[46] Lars Ruthotto and Eldad Haber. Deep Neural Networks Motivated by Partial Differential Equations. *Journal of Mathematical Imaging and Vision*, 62:352–364, 2020. 95

[47] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective Classification in Network Data. *AI magazine*, 29(3):93–93, 2008. 127

[48] Andrew W. Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander W. R. Nelson, Alex Bridgland, Hugo Penedones, Stig Petersen, Karen Simonyan, Steve Crossan, Pushmeet Kohli, David T. Jones, David Silver, Koray Kavukcuoglu, and Demis Hassabis. Improved protein structure prediction using potentials from deep learning. *Nature*, 577, 2020. 94

[49] Ferdia Sherry, Elena Celledoni, Matthias J Ehrhardt, Davide Murari, Brynjulf Owren, and Carola-Bibiane Schönlieb. Designing Stable Neural Networks using Convex Analysis and ODEs. *arXiv preprint arXiv:2306.17332*, 2023. 114, 123

[50] Yang Song, Qiyu Kang, Sijie Wang, Kai Zhao, and Wee Peng Tay. On the Robustness of Graph Neural Diffusion to Topology Perturbations. *Advances in Neural Information Processing Systems*, 35:6384–6396, 2022. 95, 97

[51] Eduardo D Sontag. Contractive Systems with Inputs. In *Perspectives in Mathematical System Theory, Control, and Signal Processing: A Festschrift in Honor of Yutaka Yamamoto on the Occasion of his 60th Birthday*, pages 217–228. Springer, 2010. 119, 122

[52] Matthew Thorpe, Tan Minh Nguyen, Hedi Xia, Thomas Strohmer, Andrea Bertozzi, Stanley Osher, and Bao Wang. GRAND++: Graph Neural Diffusion with A Source Term. In *International Conference on Learning Representations*, 2022. 95

[53] Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama. Lipschitz-Margin Training: Scalable Certification of Perturbation Invariance for Deep Neural Networks. *Advances in neural information processing systems*, 31, 2018. 59, 96, 123

[54] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. 105

[55] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic Graph CNN for Learning on Point Clouds. *arXiv preprint arXiv:1801.07829*, 2018. 94

[56] Yuelin Wang, Kai Yi, Xinliang Liu, Yu Guang Wang, and Shi Jin. ACMP: Allen-Cahn Message Passing for Graph Neural Networks with Particle Phase Transition. *arXiv preprint arXiv:2206.05437*, 2022. 95

[57] Marcin Waniek, Tomasz P Michalak, Michael J Wooldridge, and Talal Rahwan. Hiding individuals and communities in a social network. *Nature Human Behaviour*, 2(2):139–147, 2018. 94

[58] E Weinan. A Proposal on Machine Learning via Dynamical Systems. *Communications in Mathematics and Statistics*, 5(1):1–11, March 2017. 95

[59] John Wright and Yi Ma. *High-Dimensional Data Analysis with Low-Dimensional Models: Principles, Computation, and Applications*. Cambridge University Press, 2022. 2, 98

[60] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. Adversarial Examples for Graph Data: Deep Insights into Attack and Defense. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, 2019. 96, 100, 106

[61] Ying Xu, Michael Lanier, Anindya Sarkar, and Yevgeniy Vorobeychik. Attacks on Node Attributes in Graph Neural Networks. *arXiv preprint arXiv:2402.12426*, 2024. 104

[62] Runing Yang, Ruoxi Jia, Xiangyu Zhang, and Ming Jin. Certifiably Robust Neural ODE With Learning-Based Barrier Function. *IEEE Control Systems Letters*, 2023. 122

[63] Muhammad Zakwan, Liang Xu, and Giancarlo Ferrari-Trecate. Robust Classification using Contractive Hamiltonian Neural ODEs. *IEEE Control Systems Letters*, 7:145–150, 2022. 48, 60, 123

[64] Xiang Zhang and Marinka Zitnik. GNNGuard: Defending Graph Neural Networks against Adversarial Attacks. In *Advances in Neural Information Processing Systems 33*, 2020. 106, 131

[65] Kai Zhao, Qiyu Kang, Yang Song, Rui She, Sijie Wang, and Wee Peng Tay. Adversarial Robustness in Graph Neural Networks: A Hamiltonian Approach. *Advances in Neural Information Processing Systems*, 36, 2024. 97, 106, 132, 133

[66] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. Robust Graph Convolutional Networks Against Adversarial Attacks. In *Proceedings of*

the 25th ACM SIGKDD international conference on knowledge discovery & data mining, pages 1399–1407, 2019. 105, 107

[67] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial Attacks on Neural Networks for Graph Data. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2847–2856, 2018. 96, 105, 107, 108, 127

[68] Daniel Zügner and Stephan Günnemann. Adversarial Attacks on Graph Neural Networks via Meta Learning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. 94, 96, 105, 107, 127

Predictions Based on Pixel Data: Insights from PDEs and Finite Differences

Elena Celledoni, James Jackaman, Davide Murari, and Brynjulf Owren

Submitted

Abstract. As supported by abundant experimental evidence, neural networks are state-of-the-art for many approximation tasks in high-dimensional spaces. Still, there is a lack of a rigorous theoretical understanding of what they can approximate, at which cost, and at which accuracy. One network architecture of practical use, especially for approximation tasks involving images, is (residual) convolutional networks. However, due to the locality of the linear operators involved in these networks, their analysis is more complicated than that of fully connected neural networks. This paper deals with approximation of time sequences where each observation is a matrix. We show that with relatively small networks, we can represent exactly a class of numerical discretizations of PDEs based on the method of lines. We constructively derive these results by exploiting the connections between discrete convolution and finite difference operators. Our network architecture is inspired by those typically adopted in the approximation of time sequences. We support our theoretical results with numerical experiments simulating the linear advection, heat, and Fisher equations.

4.1 Introduction

When endeavoring to understand the world around us, continuum modeling through partial differential equations (PDEs) [25, 61] has proven a fundamental tool for a plethora of applications. Indeed, any process one may envision can be modeled by a differential equation through an appropriate simplification, be the focus fluid flows [65, e.g.], mathematical biology [23], chemical engineering [29], and most physical process. In practice, such models are typically solved using the rich toolbox established by numerical analysis and scientific computing. Popular techniques include finite difference methods [38], finite element methods [12], and spectral methods [11], and have proven excellent tools for simulating PDEs. However, there is a significant limitation, which is intrinsic to the inaccuracy of the PDE model itself. In other words, a PDE can only describe macroscopic behaviors that are sufficiently understood in terms of physical principles, and it can not capture phenomena at the molecular level exactly or incorporate information from observed data.

Over the past decade(s), there has been an explosion of research interest in artificial intelligence and machine learning, with applications in almost every area imaginable, like self-driving cars, chemistry, natural language processing, medical imaging, robotics, or weather forecasting [53, 9, 39, 35, 18, 5, 13]. One increasingly popular application for machine learning is in the approximation of PDEs. Indeed, this has led to the development of several new technologies, such as physics-informed neural networks (PINNs) [48, 20]. Such models are powerful tools for approximating PDE solutions. They typically build some PDE structure into the network and can be used both to solve PDEs and to learn PDEs from data. Besides PINNs, significant progress has been made

in solving the inverse problem of discovering the underlying PDE [50, 7, 69]. Additional connections between PDEs and neural networks appear in the design of neural networks inspired by discrete numerical PDEs [51, 57, 24].

Our primary goal in this paper is to understand how accurately two-layer convolutional neural networks (CNNs) can approximate space-time discretizations of PDEs. An essential step towards incorporating physical knowledge into more standard networks is to construct neural networks capturing the structure of families of PDEs while being trained on data that is only an approximation of their solutions. In [40], the authors develop a methodology for expressing finite difference approximations with convolutional layers. This connection between discrete convolution and finite differences has also been used in [63, 46, 21] and is fundamental to the results of the present paper. The main difference between our work and the results presented in [40] is the purpose of the study. In [40], the authors aim to discover a PDE given some snapshots of its solutions. They constrain their convolutional neural network architecture so that it can be seen as the semi-discretization of a PDE. Instead, in our work we study how deep convolutional filters have to be to represent the spatial semi-discretization of certain classes of PDEs. We do not impose restrictions on the entries of the filters, but we require specific choices of activation functions. See section 4.4 for our analysis. We prove that for linear PDEs, a two-layer CNN with ReLU activation function and two channels can provide a second-order accurate semi-discretization of the PDE. A similar result holds for nonlinear PDEs with quadratic interaction terms.

In this paper, we consider space-time discretizations of PDEs. We restrict to two-dimensional spatial domains and take discrete snapshots in time where each observation is a matrix. While extending to higher dimensions and to tensors is not difficult, it is omitted here for the ease of notation. Sequences of matrices or tensors can be viewed as videos, and the task of learning the map from one snapshot to the next is an instance of the more general problem of next-frame prediction. In this context, unrestricted CNNs have been proven efficient [41, 43, 26, 68]. Similarly, in [8, 28, 4, 36], the authors consider videos of dynamical systems, such as a physical pendulum, and approximate the underlying temporal evolution with a neural network inspired by ODEs and PDEs.

Reliable numerical methods for differential equations must be both accurate and stable. Stability is a known issue also for neural networks, which are inherently sensitive to input perturbations such as adversarial attacks [62]. The investigation of techniques to reduce networks' sensitivity to changes in the inputs is an active area of research [16, 56, 42, e.g.]. We analyze two techniques to enhance network stability in the context of next-frame predictions,

intended as its ability to make reliable predictions further in the future. The strategy that we propose relies on incorporating physical knowledge of the problem, specifically by preserving the norm of the PDE's initial condition, in the spirit of geometric numerical integration [31]. It is also feasible to work directly with other properties of the PDE (such as conserved quantities) [34, 6, 19, 16, 67, 66, 70, 14]. Incorporating such properties into the network often results in improved stability as we will see in the numerical experiments for the linear advection equation in the case of norm preservation.

We will test our method on several PDEs, including the Fisher nonlinear reaction-diffusion equation. We anticipate the results for this PDE in figure 4.1.1, with details found in subsection 4.6.3.

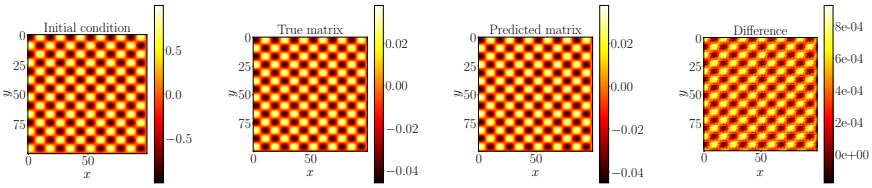


Figure 4.1.1: Visual representation of the prediction provided by the neural network. We consider one specific test initial condition, and snapshots coming from the space-time discretization of the reaction-diffusion equation (4.A.2). The network and PDE parameters can be found respectively in subsection 4.6.3 and appendix 4.A. From the left, we have the initial condition, the true space discretization of the solution at time $40\delta t$, the network prediction, and the difference between the last two matrices.

The remainder of this paper is structured as follows: In section 4.2, we discuss residual neural networks inspired by dynamical systems by introducing the fundamental tools used to build our method. In section 4.3 we provide an analysis of the error terms involved in the approximation of time-sequences generated by PDE discretizations. Section 4.4, specializes this analysis to PDEs on a two-dimensional spatial domain, presenting constructive approximation results. Section 4.5 introduces two methodologies that we use to improve the stability of the learned map, i.e., injecting noise while training the model and preserving the norm of the solution when the underlying PDE is known to do so. In section 4.6, we perform numerical experiments verifying the good dynamical behavior of the proposed methods. In section 4.7, we make concluding remarks and comment on future extensions of this research.

4.2 Residual neural networks for time sequences

We now present the considered problem for time sequences defined on a generic Euclidean space $\mathbb{R}^H$.

Problem: Let $\{(U_n^0, U_n^1, ..., U_n^M)\}_{n=1}^N$ be a set of N observed sequences each containing $M+1$ temporal snapshots represented by $U_n^m \in \mathbb{R}^H$. The superscript $m = 0, ..., M$ represents the observation time $t_m = m\delta t$, and the subscript $n = 1, ..., N$ refers to the considered initial condition $U_n^0 \in \mathbb{R}^H$. Suppose there is a map $\Phi : \mathbb{R}^H \to \mathbb{R}^H$ such that

$$U_n^{m+1} = \Phi\left(U_n^m\right) \quad n = 1, ..., N,\ m = 0, \cdots, M-1. \tag{4.2.1}$$

We seek an accurate approximation of Φ that can reproduce unseen time sequences generated by the same dynamics (i.e., on a test set)[1].

We consider sequences arising from numerical discretizations of scalar PDEs. Since a dynamical system generates our data, it is natural to approximate the map Φ with a ResNet, which results from composing L functions of the form

$$x \mapsto x + F_{\theta_i}(x), \quad \theta_i \in \mathcal{P},\ i = 1, ..., L, \tag{4.2.2}$$

where $\mathcal{P}$ is the space of admissible parameters, F_{θ_i} is a vector field on $\mathbb{R}^H$, $x \in \mathbb{R}^H$, and L is the number of layers. One can see (4.2.2) as an explicit Euler step of size 1 for the parametric vector field $x \mapsto F_{\theta_i}(x)$, [22, 30].

Throughout, we denote with Φ_F^t the exact flow, at time t, of the vector field $F : \mathbb{R}^H \to \mathbb{R}^H$, and we use $\Psi_F^{\delta t}$ to denote one of its numerical approximations at time δt, for example $\Psi_F^{\delta t}(x) = x + \delta t F(x)$, $\delta t = 1$, in (4.2.2).

We follow three steps for the approximation of Φ:

1. Define a family of vector fields $\{F_\theta : \theta \in \mathcal{P}\}$ capable of providing approximations of a spatial semi-discretization of the unknown PDE that is both qualitatively and quantitatively accurate.

2. Choose a one-step numerical method $\Psi_{F_\theta}^{\delta t}$ that is compatible with the dynamics to be approximated.

3. Choose a suitable loss function and minimize the discrepancy between the model predictions and the training data.

[1] When we refer to a generic sequence, we suppress the subscript n and write $\{U^m\}_{m=0,...,M}$.

4.2.1 Characterization of the vector field

We study parametrizations of F_θ of the form $F_\theta(U) = \mathcal{L}_2\sigma\left(\mathcal{L}_1(U)\right)$ where $\mathcal{L}_1, \mathcal{L}_2$ are suitable linear maps, and σ is a non-linear activation function applied entry-wise. The linear maps we consider are represented by convolution operations with unrestricted filters. Our choice of the activation functions is so that F_θ can represent products between derivative discretizations that typically appear in PDEs. The details for this characterization of F_θ are provided in section 4.4. Our construction is inspired by [3, 41, 40].

4.2.2 Numerical time integrator

Given a parametric vector field $F_\theta : \mathbb{R}^H \to \mathbb{R}^H$, we define a neural network $\mathcal{N}_\theta(U) := \Psi^{\delta t}_{F_\theta}(U)$ by using the numerical method $\Psi^{\delta t}_{F_\theta}$. In our experiments, the map $\Psi^{\delta t}_{F_\theta}$ will generally be defined by composing k substeps with a numerical method $\varphi^{\delta t/k}_{F_\theta}$ of time step $\delta t/k$. Thus, we can express $\mathcal{N}_\theta$ as

$$\mathcal{N}_\theta(U) = \Psi^{\delta t}_{F_\theta}(U) = \underbrace{\varphi^{\delta t/k}_{F_\theta} \circ \ldots \circ \varphi^{\delta t/k}_{F_\theta}}_{k \text{ times}}(U).$$

This can be seen as a k-layer neural network with shared weights. In most of our experiments, $\varphi^{\delta t/k}_{F_\theta}$ corresponds to the explicit Euler method. The chosen numerical method can also be structure-preserving in case the considered dataset has some property that is worth preserving. Experimentally, we shall consider one such problem in the linear advection equation, which is norm preserving. We will see that preserving such an invariant leads to improved stability in the predictions. We compare this to noise injection [10], see section 4.5.

4.2.3 Optimization problem to solve

We conclude this section with the third step, reporting the loss function optimized to find the final approximate function $\mathcal{N}_\theta$. Since this optimization step is not the main focus of the paper, we adopt the standard mean squared error loss function defined for the full dataset as

$$\mathcal{L}(\theta, Q) = \frac{1}{N \cdot Q} \sum_{n=1}^{N} \sum_{q=1}^{Q} \left\| \mathcal{N}_\theta^q\left(U_n^0\right) - \Phi^q\left(U_n^0\right) \right\|^2, \tag{4.2.3}$$

where $Q < M$ is the number of steps we perform while training, and $\mathcal{N}_\theta^q = \underbrace{\mathcal{N}_\theta \circ \cdots \circ \mathcal{N}_\theta}_{q \text{ steps}}$. We note that $\Phi^q(U_n^0) = U_n^q$, as seen in (4.2.1). We use the Adam

optimizer to minimize the loss in our experiments.

The strategy of composing q times the neural network while optimizing its weights helps reduce the accumulation of error due to the iterative application of the neural network, see [17, 15]. This strategy, hence, positively affects the network's temporal stability.

4.3 Error bounds for network-based approximations of PDE solutions

Let us restrict our focus to seeking solutions $u : \mathbb{R} \times \Omega \to \mathbb{R}$, $\Omega \subset \mathbb{R}^d$, which satisfy PDEs of the form

$$\partial_t u = \mathcal{L}u + \sum_{i=1}^{I} \beta_i \mathcal{D}_i^a u \cdot \mathcal{D}_i^b u, \quad \beta_i \in \mathbb{R},\ t \geq 0,\ x \in \Omega, \tag{4.3.1}$$

where $\mathcal{L}, \mathcal{D}_i^a, \mathcal{D}_i^b$ are linear differential operators in the spatial variable $x \in \mathbb{R}^d$, and I is the number of quadratic interactions in the PDE.

A commonly adopted strategy to find approximate solutions to (4.3.1) is to use the method of lines (see, e.g., [55, 54]). We introduce a spatial discretization of the differential operators based on H spatial nodes in Ω and obtain an ODE of the form

$$\dot{U}(t) = LU(t) + \sum_{i=1}^{I} \beta_i \left(D_i^a U(t)\right) \odot \left(D_i^b U(t)\right) = F\left(U(t)\right) \in \mathbb{R}^H, \tag{4.3.2}$$

where the components of U are approximations of u in the grid-points of the spatial discretization, $\odot$ is the entry-wise product, and LU, $D_i^a U$, and $D_i^b U$ are spatial discretizations of $\mathcal{L}u$, $\mathcal{D}_i^a u$, and $\mathcal{D}_i^b u$. Going forward, we assume that the dataset $\{(U^0, U^1, ..., U^M)\}$, where $U^m \in \mathbb{R}^H$, provides an approximate solution to (4.3.2). That is to say that by introducing a time step δt and defining $t_m = m\delta t$, we have $U^m \approx U(t_m)$ for $t \mapsto U(t) \in \mathbb{R}^H$ which is the exact solution of (4.3.2).

Let x_h, $h = 1, ..., H$, be a generic point on the spatial grid over Ω. Then, $(U^m)_h \approx u(t_m, x_h)$ where $\mathbb{R} \times \Omega \ni (t, x) \mapsto u(t, x) \in \mathbb{R}$ is the analytical solution of (4.3.1). To account for possible measurement errors, we introduce the function $e : \mathbb{R} \times \Omega \to \mathbb{R}$ representing the difference between the data and the true solution u, which allows to write

$$\left(U^0\right)_h = u\left(0, x_h\right) \tag{4.3.3}$$

$$\left(U^m\right)_h = u\left(t_m, x_h\right) + e\left(t_m, x_h\right), \quad m = 1, ..., M. \tag{4.3.4}$$

4.3.1 Splitting of the approximation errors

Our goal is to find a map $\mathcal{N}_\theta$ which accurately approximates the function

$$U^m \mapsto \Phi\left(U^m\right) = U^{m+1}, \quad m = 0, ..., M-1.$$

We focus on the case $m = 0$, but the same analysis can be done for the other values of m. Let $\Phi_F^{\delta t}$ be the exact flow map of the ODE in (4.3.2). Considering the spatial error due to the discretization, which we assume is of a generic order k, (4.3.4) implies

$$U^1 = \Phi_F^{\delta t}\left(U^0\right) + \mathcal{O}\left(\delta x^k\right) + \varepsilon^1,$$

where $\varepsilon^1 \in \mathbb{R}^H$ is defined such that $\left(\varepsilon^1\right)_h = e\left(t_1, x_h\right)$, δx quantifies the mesh size of the spatial grid, and k is the order of the method used to spatially discretize the PDE (4.3.1) to obtain (4.3.2). We focus on approximating the unknown ODE (4.3.2). More precisely, we will evaluate how well we can approximate the flow map $\Phi_F^{\delta t}$ of F given a parametric set of functions

$$\mathcal{F} = \left\{ F_\theta\left(U\right) = \mathcal{L}_2\left(\theta\right)\sigma\left(\mathcal{L}_1\left(\theta\right)\left(U\right)\right) \in \mathbb{R}^H : \theta \in \mathcal{P} \right\} \tag{4.3.5}$$

for a set of admissible parameters $\mathcal{P}$ inducing mappings of the form

$$U \mapsto \Psi_{F_\theta}^{\delta t}\left(U\right),\ F_\theta \in \mathcal{F}.$$

For simplicity, we will suppress the dependency of $\mathcal{L}_1$ and $\mathcal{L}_2$ on θ writing $F_\theta(U) = \mathcal{L}_2\sigma(\mathcal{L}_1 U)$ for two linear maps $\mathcal{L}_1 = \mathcal{L}_1(\theta)$ and $\mathcal{L}_2 = \mathcal{L}_2(\theta)$. Through splitting the local error, starting from position U^0, we may write

$$\begin{aligned}
&\left\| U^1 - \Psi_{F_\theta}^{\delta t}\left(U^0\right) \right\| \\
&= \left\| \varepsilon^1 + \mathcal{O}\left(\delta x^k\right) + \Phi_F^{\delta t}\left(U^0\right) - \Psi_{F_\theta}^{\delta t}\left(U^0\right) \right\| \\
&\leq \left\| \varepsilon^1 \right\| + \mathcal{O}\left(\delta x^k\right) + \left\| \Phi_F^{\delta t}\left(U^0\right) - \Psi_F^{\delta t}\left(U^0\right) + \Psi_F^{\delta t}\left(U^0\right) - \Psi_{F_\theta}^{\delta t}\left(U^0\right) \right\| \\
&\leq \underbrace{\left\| \varepsilon^1 \right\|}_{\text{measurement error}} + \underbrace{\mathcal{O}\left(\delta x^k\right)}_{\text{spatial error}} + \underbrace{\left\| \Phi_F^{\delta t}\left(U^0\right) - \Psi_F^{\delta t}\left(U^0\right) \right\|}_{\text{classical error estimate}} \\
&\quad + \underbrace{\left\| \Psi_F^{\delta t}\left(U^0\right) - \Psi_{F_\theta}^{\delta t}\left(U^0\right) \right\|}_{\text{network approximation}}.
\end{aligned} \tag{4.3.6}$$

In this chain of inequalities, $\|\cdot\|$ denotes the Euclidean norm on $\mathbb{R}^H$. The estimate's classical error term only depends on the local truncation error of the

numerical method $\Psi^{\delta t}$. Indeed, this term is $\mathcal{O}(\delta t^{r+1})$ if $\Psi^{\delta t}$ is a method of order r, allowing us to write $\Psi_F^{\delta t}\left(U^0\right) = \Phi_F^{\delta t}\left(U^0\right) + \mathcal{O}\left(\delta t^{r+1}\right)$, which leads to

$$\left\| U^1 - \Psi_{F_\theta}^{\delta t}\left(U^0\right)\right\| \leq \mathcal{O}\left(\delta x^k\right) + \mathcal{O}\left(\delta t^{r+1}\right) + \left\|\varepsilon^1\right\| + \left\|\Phi_F^{\delta t}\left(U^0\right) - \Phi_{F_\theta}^{\delta t}\left(U^0\right)\right\|.$$

We assume that for all the considered initial conditions U_n^0 and time instants $t \in [0, \delta t]$, the vectors $\Phi_F^t(U_n^0)$ and $\Phi_{F_\theta}^t(U_n^0)$ belong to a compact set $\Omega \subset \mathbb{R}^H$. Restricting to Ω, the vector field F in (4.3.2) is Lipschitz continuous, with a Lipschitz constant denoted as $\mathrm{Lip}(F)$. To handle the second term on the right-hand side of (4.3.6), we work with Gronwall's inequality applied to the integral representation of the flow map:

$$\begin{aligned}
&\left\|\Phi_F^{\delta t}\left(U^0\right) - \Phi_{F_\theta}^{\delta t}\left(U^0\right)\right\| \leq \int_0^{\delta t}\left\|F\left(\Phi_F^s\left(U^0\right)\right) - F_\theta\left(\Phi_{F_\theta}^s\left(U^0\right)\right)\right\| \mathrm{d}\, s \\
&= \int_0^{\delta t}\left\|F\left(\Phi_F^s\left(U^0\right)\right) - F\left(\Phi_{F_\theta}^s\left(U^0\right)\right) + F\left(\Phi_{F_\theta}^s\left(U^0\right)\right) - F_\theta\left(\Phi_{F_\theta}^s\left(U^0\right)\right)\right\| \mathrm{d}\, s \\
&\leq \mathrm{Lip}\left(F\right)\int_0^{\delta t}\left\|\Phi_F^s\left(U^0\right) - \Phi_{F_\theta}^s\left(U^0\right)\right\| \mathrm{d}\, s + \delta t \sup_{V\in\Omega}\left\|F_\theta\left(V\right) - F\left(V\right)\right\| \\
&\implies \left\|\Phi_F^{\delta t}\left(U^0\right) - \Phi_{F_\theta}^{\delta t}\left(U^0\right)\right\| \leq \delta t \exp\left(\mathrm{Lip}\left(F\right)\delta t\right)\sup_{V\in\Omega}\left\|F_\theta\left(V\right) - F\left(V\right)\right\|.
\end{aligned} \tag{4.3.7}$$

Thus, to control this quantity from above, we need to understand the approximation properties of $\mathcal{F}$. In particular, we want to quantify how much complexity $\mathcal{F}$ requires to guarantee the existence of an $F_\theta \in \mathcal{F}$ leading to

$$\sup_{V\in\Omega}\left\|F_\theta\left(V\right) - F\left(V\right)\right\| < \delta t^q \tag{4.3.8}$$

for some $q \geq 1$. We note that if $q = r$, such a result would guarantee that the approximation of the map Φ provided by $\Psi_{F_\theta}^{\delta t}$ can be as accurate as the one provided by the numerical method $\Psi_F^{\delta t}$ directly applied to the exact vector field F. We now characterize the space $\mathcal{F}$ so that it can exactly represent F, i.e., so that there exists an element $F_\theta \in \mathcal{F}$ with $F = F_\theta$. In practice, this exact representation will never be obtained, and the presented derivation allows the quantification of the approximation error. In the next section, we will focus on two-dimensional PDEs, i.e., $d = 2$, and analyze (4.3.8) for that setting.

4.4 Error analysis for PDEs on a two-dimensional spatial domain

We now focus on PDEs defined on a two-dimensional spatial domain. These PDEs are discretized on a uniform grid with p grid points along both the axes over the spatial domain $\Omega = [0,1]^2 \subset \mathbb{R}^2$, i.e., we set $d = 2$ and $H = p^2$. In this two-dimensional setting, we denote the grid points by (x_h, y_k), $h, k = 1, ..., p$. Since the grid is uniform, we have $x_{h+1} - x_h = y_{k+1} - y_k =: \delta x$. For convenience, we represent an element in $\mathbb{R}^{p^2}$ as a matrix in $\mathbb{R}^{p \times p}$. The only immediate consequence is that the previously derived estimates involving the Euclidean norm of $\mathbb{R}^H$ now hold in the Frobenius norm of $\mathbb{R}^{p \times p}$. We assume that the PDE is at most of the second order and that the spatial semi-discretization in (4.3.2) comes from a second-order accurate finite differences scheme, i.e., $k = 2$ in (4.3.6). By restricting ourselves to the two-dimensional case, second-order PDEs, and second-order accurate finite differences, we significantly simplify the exposition. Our reasoning may be extended to higher-dimensional domains discretized with regular grids and higher-order finite differences.

For our derivations, we use tensors of orders three and four. We denote a tensor of order four as

$$\boldsymbol{\mathcal{X}} = \left[\boldsymbol{X}_1, ..., \boldsymbol{X}_J\right] \in \mathbb{R}^{J \times K \times R \times S} \tag{4.4.1}$$

where $\boldsymbol{X}_j \in \mathbb{R}^{K \times R \times S}$ is a tensor of order three for every $j = 1, ..., J$ defined as

$$\boldsymbol{X}_j = \left[X_{j,1}, ..., X_{j,K}\right] \in \mathbb{R}^{K \times R \times S}, \tag{4.4.2}$$

where $X_{j,k} \in \mathbb{R}^{R \times S}$ for every $k = 1, ..., K$. We may access the components of fourth-order tensors using the notation $\boldsymbol{\mathcal{X}}_j = \boldsymbol{X}_j \in \mathbb{R}^{K \times R \times S}$, and $\boldsymbol{\mathcal{X}}_{j,k} = X_{j,k} \in \mathbb{R}^{R \times S}$, for $j = 1, ..., J$ and $k = 1, ..., K$. When $\boldsymbol{X} = [X_1] \in \mathbb{R}^{1 \times R \times S}$, we will often contract the first dimensionality and refer to it as $X_1 \in \mathbb{R}^{R \times S}$.

We now show that any function with the same structure as F, as described in (4.3.2), can be represented by the parametric space of functions $\mathcal{F}$ defined in (4.3.5) with linear maps realized by convolution operations.

4.4.1 Convolutional layers as finite differences

The discrete convolution operation is the foundation of many successful machine learning algorithms, particularly for approximation tasks involving images. This work focuses on "same" convolutions, i.e., convolution operations that do not change the input dimension. In this case, the input matrix has to be padded compatibly with the application of interest. Specifically, the padding

strategy in our setting should relate to how the PDE solution u behaves outside the domain $[0,1]^2$, i.e., to the boundary conditions. We focus on the case of periodic boundary conditions, as considered in the numerical experiments.

For the specific situation $U \in \mathbb{R}^{3\times 3}$, i.e., $p = 3$, and 3×3 convolutional filters, the periodic padding leads to

$$U_P = \begin{bmatrix} u_{33} & u_{31} & u_{32} & u_{33} & u_{31} \\ u_{13} & u_{11} & u_{12} & u_{13} & u_{11} \\ u_{23} & u_{21} & u_{22} & u_{23} & u_{21} \\ u_{33} & u_{31} & u_{32} & u_{33} & u_{31} \\ u_{13} & u_{11} & u_{12} & u_{13} & u_{11} \end{bmatrix}.$$

Regardless of the value of p, for 3×3 "same" convolutions, one has to add two rows and two columns around the matrix U. The convolution operation $K * U$ defined by a 3×3 filter K is a linear map obtained by computing the Frobenius inner product of K with all the contiguous 3×3 submatrices of the padded input U_P. We show this procedure in the following example:

$$U_P = \begin{bmatrix} u_{33} & u_{31} & u_{32} & u_{33} & u_{31} \\ u_{13} & u_{11} & u_{12} & u_{13} & u_{11} \\ u_{23} & u_{21} & u_{22} & u_{23} & u_{21} \\ u_{33} & u_{31} & u_{32} & u_{33} & u_{31} \\ u_{13} & u_{11} & u_{12} & u_{13} & u_{11} \end{bmatrix}, \quad r_{11} = \operatorname{trace}\left(\begin{bmatrix} u_{33} & u_{31} & u_{32} \\ u_{13} & u_{11} & u_{12} \\ u_{23} & u_{21} & u_{22} \end{bmatrix}^T K \right),$$

where r_{11} is the first entry of the output R obtained convolving K with U. This operation is local as it only considers the entries of 3×3 submatrices.

The same locality property holds for finite difference operators (see, e.g., [64, 47]), which are discrete approximations of the derivatives of a function given its nodal values sampled on a grid. For the PDE (4.3.1), if the solution u is regular enough in the spatial variables, there exists a $K \in \mathbb{R}^{3\times 3}$ such that

$$(K * U)_{hk} = \mathcal{L}u\left(\bar{t}, x_h, y_k\right) + \mathcal{O}\left(\delta x^2\right),$$

where $U \in \mathbb{R}^{p\times p}$ is defined as $U_{hk} = u(\bar{t}, x_h, y_k)$ for a fixed $\bar{t} \geq 0$. An explicit example is the well-known 5−point formula (see [1, Formula 25.3.30]) for approximating the Laplace operator Δ, which is defined as

$$\frac{1}{\delta x^2}\left(\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} * U \right)_{hk} = \Delta u\left(\bar{t}, x_h, y_k\right) + \mathcal{O}\left(\delta x^2\right).$$

Similarly, any partial derivative of second-order or lower can be approximated to second-order accuracy with a 3×3 convolution. Consequentially, one may

observe that (4.3.2) with L, D_i^a, and D_i^b realized by 3×3 convolution operations, can provide a second-order accurate spatial discretization of the PDE (4.3.1).

The convolution operator can be extended from matrices to higher-order tensors. Let $\boldsymbol{U} \in \mathbb{R}^{C_i \times p \times p}$ be a generic third-order tensor and $\boldsymbol{\mathcal{K}} \in \mathbb{R}^{C_o \times C_i \times K \times K}$ a fourth-order tensor representing the set of filters defining the convolution operation. We denote with $\boldsymbol{R} \in \mathbb{R}^{C_o \times p \times p}$ the result of the convolution operation $\boldsymbol{\mathcal{K}} * \boldsymbol{U} = \boldsymbol{R}$. The components of $\boldsymbol{R}$ can be characterized as

$$\boldsymbol{R}_i = \sum_{j=1}^{C_i} \boldsymbol{\mathcal{K}}_{i,j} * \boldsymbol{U}_j \in \mathbb{R}^{p \times p}, \quad i = 1, ..., C_o.$$

As in the PyTorch library [44], we adopt the convention $\boldsymbol{\mathcal{K}} \in \mathbb{R}^{C_o \times C_i \times K \times K}$ where C_i and C_o are the numbers of input and output channels respectively, while the convolutional filters $\boldsymbol{\mathcal{K}}_{i,j}$ are of shape $K \times K$.

4.4.2 Error analysis for F_θ based on convolution operations

Based on the connections between finite differences and discrete convolution, we now show that building F_θ as

$$F_\theta(U) = \mathcal{L}_2 \sigma\left(\mathcal{L}_1(U)\right) \tag{4.4.3}$$

with σ a suitable activation function applied entry-wise, $\mathcal{L}_1(U) = \boldsymbol{\mathcal{K}} * U + b_1$, and $\mathcal{L}_2\left(\sigma\left(\mathcal{L}_1(U)\right)\right) = \boldsymbol{\mathcal{H}} * \sigma\left(\mathcal{L}_1(U)\right) + b_2$ allows to exactly represent the right-hand side F for a second-order accurate semi-discretization of the PDE (4.3.1), where b_1, b_2 are bias terms added to the convolved inputs. In this case, $\theta = (\boldsymbol{\mathcal{K}}, \boldsymbol{\mathcal{H}}, b_1, b_2)$ represents the set of parameters defining F_θ.

A common choice for σ is the rectified linear unit $\sigma_1(x) = \mathrm{ReLU}(x) := \max\{0, x\}$. Some publications also considered powers of σ_1 (see, e.g., [59, 37]), calling them Rectified Power Units (RePUs). We first present a theoretical derivation based on $\sigma_2(x) = \mathrm{ReLU}^2(x)$. Then, we demonstrate a simplification of this result for linear partial differential equations (PDEs) based on $\sigma_1(x) = \mathrm{ReLU}(x)$. These two activation functions satisfy the important property

$$x^q = \mathrm{ReLU}^q(x) + (-1)^q \mathrm{ReLU}^q(-x), \; q \in \mathbb{N}. \tag{4.4.4}$$

This identity allows for polynomials of degrees 1 and 2 to be represented by composing suitable linear functions and the two activation functions σ_1 and σ_2. These two activation functions are not polynomials, which allows them to be included in networks that can approximate sufficiently regular functions

as accurately as desired, see [45]. Therefore, they are more appealing than polynomials when designing neural network architectures.

As an immediate consequence of (4.4.4), one can derive the following identities

$$D * U = \sigma_1\left(D * U\right) - \sigma_1\left(-D * U\right), \tag{4.4.5a}$$

$$\begin{aligned} D * U &= \frac{1}{2}\left(\left(D * U + 1\right)^2 - \left(D * U\right)^2\right) - \frac{1}{2} \\ &= \frac{1}{2}\Big(\sigma_2\left(D * U + 1\right) + \sigma_2\left(-D * U - 1\right) \\ &\quad -\sigma_2\left(D * U\right) - \sigma_2\left(-D * U\right)\Big) - \frac{1}{2}, \end{aligned} \tag{4.4.5b}$$

$$\begin{aligned} \left(D_1 * U\right) \odot \left(D_2 * U\right) &= \frac{1}{2}\left(\left(\left(D_1 + D_2\right) * U\right)^2 - \left(D_1 * U\right)^2 - \left(D_2 * U\right)^2\right) \\ &= \frac{1}{2}\Big(\sigma_2\left(\left(D_1 + D_2\right) * U\right) + \sigma_2\left(-\left(D_1 + D_2\right) * U\right) \\ &\quad -\sigma_2\left(D_1 * U\right) - \sigma_2\left(-D_1 * U\right) \\ &\quad -\sigma_2\left(D_2 * U\right) - \sigma_2\left(-D_2 * U\right)\Big). \end{aligned} \tag{4.4.5c}$$

These identities show how one can handle the linear term and the quadratic non-linearities arising in (4.3.2) using parametrizations like those in (4.4.3). More explicitly, (4.4.5a) and (4.4.5b) show how to handle the linear term $L * U$, and (4.4.5c) the quadratic interactions, as formalized in the following theorem.

Theorem 4.1. *Let $U \in \mathbb{R}^{p \times p}$ and*

$$F\left(U\right) = L * U + \sum_{i=1}^{I} \beta_i \left(D_{2i-1} * U\right) \odot \left(D_{2i} * U\right) \in \mathbb{R}^{p \times p},$$

for $L, D_1, D_2, \ldots, D_{2I} \in \mathbb{R}^{3 \times 3}$. Further, let F_θ be the parametric map defined by

$$F_\theta\left(U\right) = \boldsymbol{\mathcal{H}} * \sigma_2\left(\boldsymbol{\mathcal{K}} * U + b_1\right) + b_2. \tag{4.4.6}$$

Then, F_θ can represent F for suitably chosen parameters

$$\boldsymbol{\mathcal{K}} \in \mathbb{R}^{4+6I \times 1 \times 3 \times 3},\ \boldsymbol{\mathcal{H}} \in \mathbb{R}^{1 \times 4+6I \times 1 \times 1},\ b_1 \in \mathbb{R}^{4+6I},\ b_2 \in \mathbb{R}.$$

Proof. The proof is constructive since we report the exact expression of a family of weights that achieves the desired goal. We only specify the parts of the convolutional filters that are non-zero, which follow from (4.4.5). We first fix the bias terms as

$$b_1 = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

and $b_2 = -1/2$. For the first convolutional filter, we instead set

$$\mathcal{K}_{1,1} = -\mathcal{K}_{2,1} = \mathcal{K}_{3,1} = -\mathcal{K}_{4,1} = L$$

taking care of the linear part and

$$\mathcal{K}_{4+6i-5,1} = -\mathcal{K}_{4+6i-4,1} = D_{2i-1} + D_{2i},\ i = 1, ..., I,$$

$$\mathcal{K}_{4+6i-3,1} = -\mathcal{K}_{4+6i-2,1} = D_{2i-1},\ i = 1, ..., I,$$

$$\mathcal{K}_{4+6i-1,1} = -\mathcal{K}_{4+6i,1} = D_{2i},\ i = 1, ..., I,$$

which allows us to deal with quadratic interactions. This choice lets us get

$$\begin{aligned}\mathcal{K} * U + b_1 = \Big[& L * U + 1, -L * U - 1, L * U, -L * U, \\ & \big(D_1 + D_2\big) * U, -\big(D_1 + D_2\big) * U, D_1 * U, -D_1 * U, \\ & D_2 * U, -D_2 * U, ..., \big(D_{2I-1} + D_{2I}\big) * U, -\big(D_{2I-1} + D_{2I}\big) * U, \\ & D_{2I-1} * U, -D_{2I-1} * U, D_{2I} * U, -D_{2I} * U \Big].\end{aligned}$$

Based again on (4.4.5), we can conclude that, by setting

$$\mathcal{H}_{1,1} = \mathcal{H}_{1,2} = -\mathcal{H}_{1,3} = -\mathcal{H}_{1,4} = \frac{1}{2},$$

$$\begin{aligned}\mathcal{H}_{1,4+6i-5} &= \mathcal{H}_{1,4+6i-4} = -\mathcal{H}_{1,4+6i-3} \\ &= -\mathcal{H}_{1,4+6i-2} = -\mathcal{H}_{1,4+6i-1} = -\mathcal{H}_{1,4+6i} = \frac{\beta_i}{2},\ i = 1, ..., I,\end{aligned}$$

the result follows. □

We remark that the considered F_θ is not limited to representing only functions with the same structure as F, and this is the primary motivation behind the choice of not explicitly defining F_θ as

$$F_\theta\big(U\big) = L * U + \sum_{i=1}^{I} \beta_i \big(D_{2i-1} * U\big) \odot \big(D_{2i} * U\big).$$

Indeed, it is generally hard to know if some temporal observations come from the discretization of a PDE. For this reason, we work with a more general neural network architecture. We note that any space of parametric functions $\mathcal{F}$ that contains parametric functions as those in theorem 4.1 can represent F. That is to say that many overparametrized networks can be used while maintaining the theoretical guarantees.

We now simplify the construction for the case of linear PDEs.

Theorem 4.2. *Let $U \in \mathbb{R}^{p\times p}$ and*

$$F(U) = L * U \in \mathbb{R}^{p\times p},$$

for $L \in \mathbb{R}^{3\times 3}$. Further, let F_θ be the parametric map defined by

$$F_\theta(U) = \boldsymbol{\mathcal{H}} * \sigma_1(\boldsymbol{\mathcal{K}} * U + b_1) + b_2. \tag{4.4.7}$$

Then, F_θ can represent F for suitably chosen parameters

$$\boldsymbol{\mathcal{K}} \in \mathbb{R}^{2\times 1\times 3\times 3}, \ \boldsymbol{\mathcal{H}} \in \mathbb{R}^{1\times 2\times 1\times 1}, \ b_1 \in \mathbb{R}^2, \ b_2 \in \mathbb{R}.$$

Proof. One can set $b_1 = [0,0]$, $b_2 = 0$, $\boldsymbol{\mathcal{K}}_{1,1} = -\boldsymbol{\mathcal{K}}_{2,1} = L$, and $\boldsymbol{\mathcal{H}}_{1,2} = -\boldsymbol{\mathcal{H}}_{1,2} = 1$, which allows to conclude the proof. □

These two theorems can be extended to any activation function satisfying (4.4.4). More explicitly, what is essential is to be able to represent linear maps and quadratic interactions composing σ with suitable linear maps. Apart from polynomial activation functions, the LeakyReLU activation function also allows the representation of the right-hand side for linear PDEs. Indeed, such activation function is defined as $\text{LeakyReLU}(x;a) = \max\{ax, x\}$, $a \in (0,1)$, and theorem 4.2 extends to this activation function since

$$x = \frac{1}{1+a}\Big(\text{LeakyReLU}(x;a) - \text{LeakyReLU}(-x;a)\Big).$$

We conclude with a corollary combining this section's derivations with those in section 4.3, providing the central insight into our theoretical analysis.

Corollary 2. Let $\Psi^{\delta t}$ be a numerical method of order r. Let $U^0 \mapsto \Phi\left(U^0\right)$ be the target one-step map obtained on a uniform mesh of $\Omega = [0,1]^2$ and with time step δt, for the PDE (4.3.1). Further, assume the measurement error is either zero or of order equal to or higher than r in time and 2 in space. Then, the map $U^0 \mapsto \Psi^{\delta t}_{F_\theta}\left(U^0\right)$ can provide an approximation of Φ accurate to order 2 in space and r in time if F_θ is defined as in theorem 4.1.

We remark that the same result holds for linear PDEs and parametric spaces of functions as in theorem 4.2.

Proof. The proof immediately follows by combining the error splitting in (4.3.6) and theorem 4.1. Indeed, one can get

$$\left\|\Phi(U^0) - \Psi_{F_\theta}^{\delta t}\left(U^0\right)\right\| \leq \mathcal{O}\left(\delta x^2\right) + \mathcal{O}\left(\delta t^{r+1}\right) + \left\|\varepsilon^1\right\| + \left\|\Psi_F^{\delta t}\left(U^0\right) - \Psi_{F_\theta}^{\delta t}\left(U^0\right)\right\|$$
$$= \mathcal{O}\left(\delta x^2\right) + \mathcal{O}\left(\delta t^{r+1}\right) + \left\|\varepsilon^1\right\|$$

since there exists a $\theta \in \mathcal{P}$ such that $\mathcal{F} \ni F_\theta = F$. □

This result ensures the possibility of getting approximations of PDE solutions that are second-order accurate in space and r in time, with networks having a number of parameters growing linearly with the quadratic interactions in (4.3.1).

Remark. *Our expressivity results rely on showing that the neural networks we consider can exactly represent a class of classical numerical methods and, hence, inherit their approximation rates. This reasoning is common when analyzing neural networks, e.g., [2, 34]. In the experiments, the parametric function F_θ is never forced to reproduce the spatial semi-discretization of a PDE, meaning that the presented derivations are intended as an upper bound for how poor the found approximation can be. If there is a better approximation of Φ that can be provided given the available data, the training phase will aim at that target.*

4.5 Improving the stability of predictions

The stability of an iterative method is often as crucial as its quantitative accuracy. When predicting the next frame of a time sequence, we consider a map stable if it is not overly sensitive to input perturbations. In our setting, iteratively applying the network leads to artifacts in the predictions, resulting in data points on which the network has not been trained. The goal is to minimize the impact of these artifacts, preventing a significant degradation in the accuracy of subsequent predictions.

We work on improving the stability of our networks on two levels. The spatial stability of the model is linked to how the parametric vector field F_θ, which provides an approximate spatial semi-discretization of the PDE, processes the input matrices. For this, we have increased the size of the convolutional filters from 3×3 to 5×5 so that the pixel perturbations are better averaged out due to the wider window of action of the convolutional layers. For the temporal stability, we explore two strategies, both detailed in this section. The first is noise injection, and the second involves building norm-preserving neural networks when dealing with data coming from norm-preserving PDEs. In the

experiments associated with linear advection, which is known to be norm preserving, we will show that preserving the norm results in the most significant stability improvements, followed by the noise injection strategy. Both these approaches improve on the results obtained without applying any temporal stability-enhancing strategy.

4.5.1 Noise injection

A technique often used in the literature to improve the training of a neural network is to introduce noise into the training set before sending the data through the network. This is generally additive noise with an expected value of zero. This helps to reduce the chance of overfitting the dataset. It has been shown, for example in [10], that introducing noise in the inputs is equivalent to regularizing the network weights. Weight regularization is a commonly used technique in machine learning to improve the generalization capabilities of these parametric models, as discussed in [27, Chapter 7].

To introduce noise, we modify the loss function to the form

$$\mathcal{L}_\varepsilon(\theta, Q) = \frac{1}{N \cdot Q} \sum_{n=1}^{N} \sum_{q=1}^{Q} \left\| \mathcal{N}_\theta^q \left(U_n^0 + \delta_n \right) - \Phi^q \left(U_n^0 \right) \right\|^2, \tag{4.5.1}$$

where $\delta_n \sim \mathcal{U}(-\varepsilon, \varepsilon)^{p \times p}$ are independent identically distributed uniform random variables. More precisely, a new perturbation δ_n is generated at each training iteration. In the context of approximating the dynamics of unknown PDEs, one can also think of this noise injection strategy as a way to reduce the sensitivity of the learned dynamical system to perturbations in the initial condition. Indeed, (4.5.1) ensures that the trajectories of a neighborhood of the initial condition U_n^0 are pushed towards the trajectory of U_n^0.

4.5.2 Norm preservation

We now move to the next technique we consider: incorporating in the neural network architecture a conservation law that the PDE is known to have. We analyze the linear advection equation $\partial_t u = b \cdot \nabla u$, $\nabla \cdot b = 0$, with periodic boundary conditions on $\Omega = [0,1]^2$. For this PDE, the $L^2(\Omega)$ norm of u is preserved since

$$\begin{aligned} \frac{d}{dt} \frac{1}{2} \int_\Omega u^2 \, \mathrm{d}x \, \mathrm{d}y &= \int_\Omega u \partial_t u \, \mathrm{d}x \, \mathrm{d}y = \int_\Omega u \left(b \cdot \nabla u \right) \mathrm{d}x \, \mathrm{d}y \\ &= \int_\Omega b \cdot \nabla \left(\frac{u^2}{2} \right) \mathrm{d}x \, \mathrm{d}y = \int_{\partial\Omega} n \cdot \left(b \frac{u^2}{2} \right) \mathrm{d}s = 0, \end{aligned}$$

where n is the outer pointing normal vector. To generate the training data, we use a norm-preserving numerical method, see appendix 4.A for details about it.

To design a neural network that preserves the Frobenius norm of the input matrix, we first define a parametric space $\mathcal{F}$ of vector fields whose solutions preserve the Frobenius norm and then use a norm-preserving numerical method $\Psi^{\delta t}$. We define $\mathcal{F}$ as

$$\mathcal{F} = \left\{ F_\theta(U) = \mathcal{L}_2\sigma\left(\mathcal{L}_1(U)\right) - U\frac{\operatorname{trace}\left(U^T\mathcal{L}_2\sigma\left(\mathcal{L}_1(U)\right)\right)}{\operatorname{trace}\left(U^TU\right)} : \theta \in \mathcal{P} \right\}.$$

$F_\theta(U)$ is the orthogonal projection of $\mathcal{L}_2\sigma(\mathcal{L}_1(U))$ onto the tangent space at U of the manifold of $p \times p$ matrices having the same Frobenius norm of U. As a consequence, for every $U_0 \in \mathbb{R}^{p\times p}$ and $F_\theta \in \mathcal{F}$, one has

$$\frac{d}{dt}\left\|\Phi^t_{F_\theta}(U_0)\right\|^2 = 2\cdot\operatorname{trace}\left(\Phi^t_{F_\theta}(U_0)^T F_\theta\left(\Phi^t_{F_\theta}(U_0)\right)\right) = 0,$$

and hence $\left\|\Phi^t_{F_\theta}(U_0)\right\| = \|U_0\|$ for every $t \geq 0$.

We now present the numerical method $\Psi^{\delta t}$ used in the experiments. Such an integrator is a correction of the explicit Euler method aiming to preserve the norm through a Lagrange multiplier. The method is described as follows

$$\begin{aligned}\tilde{U}^{m+1} &= U^m + \delta t F_\theta\left(U^m\right)\\ U^{m+1} &= \tilde{U}^{m+1} + \lambda\tilde{U}^{m+1} =: \Psi^{\delta t}_{F_\theta}\left(U^m\right),\end{aligned}$$

where $\lambda \in \mathbb{R}$ is chosen so that $\|U^{m+1}\|^2 = \|U^m\|^2$. For this relatively simple constraint, λ can be exactly computed as follows

$$\left\|U^{m+1}\right\|^2 = (1+\lambda)^2\left\|\tilde{U}^{m+1}\right\|^2 = \left\|U^m\right\|^2 \implies \lambda = -1 \pm \frac{\left\|U^m\right\|}{\left\|\tilde{U}^{m+1}\right\|}.$$

Given that when $\delta t = 0$ one wants to have $\lambda = 0$, the physical choice for λ is the one with the plus sign, hence leading to

$$U^{m+1} = \Psi^{\delta t}_{F_\theta}\left(U^m\right) = \frac{U^m + \delta t F_\theta\left(U^m\right)}{\left\|U^m + \delta t F_\theta\left(U^m\right)\right\|}\left\|U^m\right\|. \tag{4.5.2}$$

4.6 Numerical experiments

This section collects numerical experiments supporting the network architecture introduced in section 4.4. All neural networks are implemented with the PyTorch library [44] and are trained with the Adam optimizer. Our implementation can be found in [33]. We consider the three following problems:

1. linear advection equation, $\partial_t u = b \cdot \nabla u = \partial_x u + \partial_y u$,
2. heat equation, $\partial_t u = \alpha \Delta u = \alpha \left(\partial_{xx} u + \partial_{yy} u \right)$, and
3. Fisher equation, $\partial_t u = \alpha \Delta u + u(1-u)$.

All these PDEs are considered with doubly periodic spatial boundary conditions and solved on $\Omega = [0,1]^2 \subset \mathbb{R}^2$. We have not used the finite difference method to generate the training data to reduce the bias introduced by our data-generation technique. Indeed, we obtain the space-time observations from finite element simulations as described in appendix 4.A. These simulations yield a local truncation error of $\mathcal{O}(\delta t^3 + \delta x^2)$, allowing us to quantify our measurement error $\|\varepsilon^m\|$ within this section. Further, when we refer to a numerical method $\Psi^{\delta t}$, we perform 5 sub-steps of step size $\delta t/5$ and omit this to simplify the notation.

In subsection 4.6.1, we consider linear advection and compare the results obtained with two networks. These have the same number of parameters, but one is corrected for norm preservation, as presented in section 4.5.2, and based on the Lagrange multiplier method presented there, while the other is without these correction and projection steps. As with the other PDEs, we also compare the effects of noise injection on the error accumulation of the learned models.

In subsection 4.6.2, we deal with the heat equation with a neural network based on the presented theoretical derivations and choose the explicit Euler method as $\Psi^{\delta t}$. Finally, in subsection 4.6.3, we report results for the Fisher equation based on the explicit Euler method.

To remain consistent with the results in section 4.4, we conduct numerical experiments using networks with the same number of channels as those we theoretically studied. However, based on numerical evidence, we employ 5×5 convolutional filters rather than 3×3 and 1×1. The expressivity results still apply for these filters since they can represent convolutions with smaller filters, and, in practice, this choice leads to improved temporal stability of the network as a next-frame predictor.

Two out of the three PDEs we consider are linear. Hence, we adopt the more efficient parametrization provided by theorem 4.2, i.e., for the linear advection and heat equations, we define

$$F_\theta\left(U\right) = \boldsymbol{\mathcal{H}} * \mathrm{ReLU}\left(\boldsymbol{\mathcal{K}} * U + b_1\right) + b_2 \tag{4.6.1}$$

where $\boldsymbol{\mathcal{K}} \in \mathbb{R}^{2\times 1\times 5\times 5}$, $\boldsymbol{\mathcal{H}} \in \mathbb{R}^{1\times 2\times 5\times 5}$, $b_1 \in \mathbb{R}^2$ and $b_2 \in \mathbb{R}$. For the Fisher equation, we use the parametrization based on $\mathrm{ReLU}^2(x)$ as in theorem 4.1.

To demonstrate the network's accuracy, we present figures showing the evolution of three metrics as the network makes predictions over 40 time steps. These metrics rely on 30 test initial conditions and are defined as

$$\texttt{maxE}\left(m\right) = \max\left\{\left|\left(\mathcal{N}_\theta^m\left(U_n^0\right) - U_n^m\right)_{hk}\right| : n = 1,...,30, h, k \in \{1,...,p\}\right\}, \tag{4.6.2a}$$

$$\texttt{mse}\left(m\right) = \frac{1}{30}\sum_{n=1}^{30}\left(\frac{1}{p^2}\left\|\mathcal{N}_\theta^m\left(U_n^0\right) - U_n^m\right\|^2\right), \tag{4.6.2b}$$

$$\texttt{rE}\left(m\right) = \frac{1}{30}\sum_{n=1}^{30}\left(\frac{\left\|\mathcal{N}_\theta^m(U_n^0) - U_n^m\right\|}{\left\|U_n^m\right\|}\right), \tag{4.6.2c}$$

$$\mathcal{N}_\theta^m = \underbrace{\mathcal{N}_\theta \circ ... \circ \mathcal{N}_\theta}_{m \text{ times}}, \quad m = 1,...,40.$$

We refer to (4.6.2a) as the maximum absolute error, to (4.6.2b) as the mean squared error (MSE), and to (4.6.2c) as the average relative error. All experiments are conducted with $p = 100$, i.e., with matrices of size 100×100.

To train the networks, we optimize either the function $\mathcal{L}(\theta, M)$ in (4.2.3) or $\mathcal{L}_\varepsilon(\theta, M)$ in (4.5.1). For noise injection, we set the noise magnitude to $\varepsilon = 10^{-2}$. We adopt a training procedure which, as described in algorithm 2, pretrains the network on shorter sequences of snapshots, decreasing the learning rate as we increase the sequence length, as in curriculum learning [60]. The algorithm is presented for the complete set of training initial conditions and uses a step learning rate scheduler, dividing the learning rate by 10 every 135 epochs. In practice, we implement a mini-batch version of this algorithm with a cyclic learning rate scheduler, [58]; however, the training procedure follows the same logic. All experiments use batches of size 32, i.e., 32 different initial conditions.

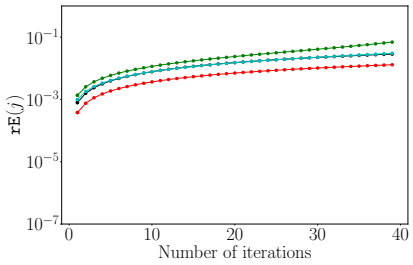
Algorithm 2 Training with the full training set and step learning rate scheduler

Initialize $\mathcal{N}_\theta$
`Epochs` $\leftarrow 300$
$\ell \leftarrow 5 \cdot 10^{-3}$ ▷ Set the starting learning rate
for $M \in [2,3,4]$ **do**
    `lr` $\leftarrow \ell$
    while `e<Epochs` **do**
        One optimization step of $\mathcal{L}(\theta, M)$ with learning rate `lr`
        if `e` $\in [135, 270]$ **then**
            `lr` $\leftarrow$ `lr`$/10$ ▷ Learning rate scheduler
        end if
        `e` $\leftarrow$ `e` $+1$
    end while
    $\ell \leftarrow \ell/2$ ▷ Now we add another time step, but we do smaller optimization steps
end for

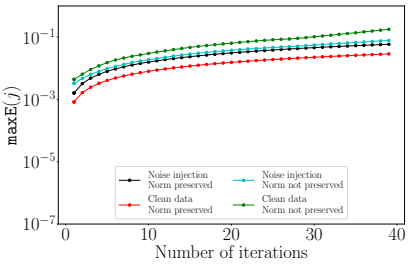
4.6.1 Linear advection equation

Before presenting the numerical results, we briefly recall the neural networks we consider. We define two neural networks: one corresponds to the explicit Euler method applied to the vector field F_θ in (4.6.1), and the other corresponds to the projected version presented in subsection 4.5.2.

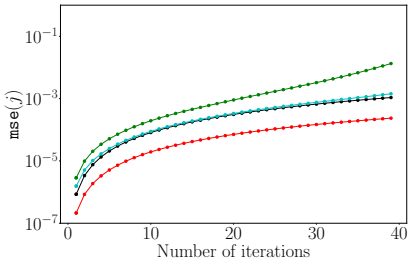
In these experiments, we compare the effects of noise injection and norm preservation on the accuracy and stability of the time series approximations provided by the trained neural network. The results are presented in figure 4.6.1. We note that, as expected, even though the neural networks we consider have the same number of parameters, their different arrangements considerably change the results we recover. Firstly, injecting noise while training the network and preserving the norm of the initial condition both improve the stability of the predictions since the error accumulates at a lower speed than without these changes. Secondly, even if we carefully specify the correct value of the norm to preserve in the experiments, combining the two strategies does not improve the results beyond only injecting noise. To conclude, we highlight that these experiments imply that such small networks not only have the potential to be expressive enough to represent the desired target map Φ, but that one can also find a set of weights leading to an accurate and stable solution.



(a) Average relative error over a batch of 30 test initial conditions.



(b) Maximum absolute error over a batch of 30 test initial conditions.



(c) Average of MSE over a batch of 30 test initial conditions.

Figure 4.6.1: Test errors for the linear advection equation.

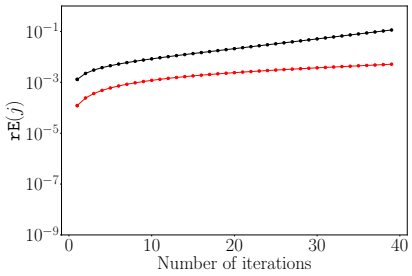
4.6.2 Heat equation

For the heat equation, we consider the same neural network architecture used for linear advection. The time step is imposed following the Courant–Friedrichs–Lewy (CFL) condition[2] and is

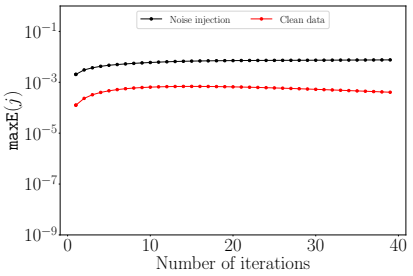
$$\delta t = 0.24 \cdot \frac{\delta x^2}{\alpha} \approx 2.445 \cdot 10^{-3}.$$

We report the results of this experiment in figure 4.6.2. The network is based on the explicit Euler method. Unlike what occurs for the linear advection equation, we notice that introducing additive noise in the training procedure worsens the results. This behavior is primarily a consequence of the inability of the training phase to find a good set of weights. While with clean data we can consistently reach a loss value of the order of 10^{-8}, the additive injection of noise leads to a final training loss of the order of 10^{-5}. The dynamics of the heat equation are dissipative, leading to the need for a more complex regularization strategy than additive noise.

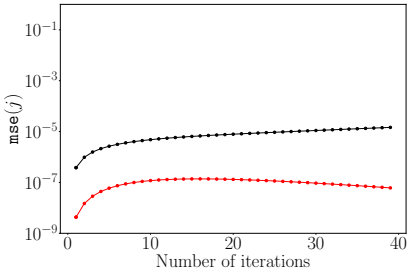
[2]The details on the values of the diffusivity constant α can be found in appendix 4.A.1



(a) Average relative error over a batch of 30 test initial conditions.



(b) Maximum absolute error over a batch of 30 test initial conditions.



(c) Average of MSE over a batch of 30 test initial conditions.

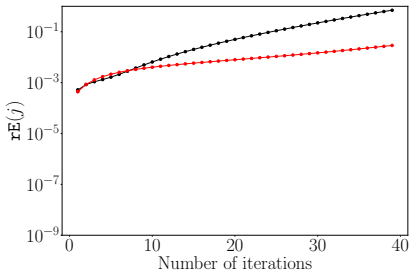
Figure 4.6.2: Test errors for the heat equation.

4.6.3 Fisher equation

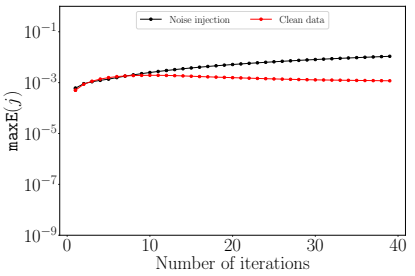
The Fisher equation is a nonlinear PDE with $I = 1$ quadratic nonlinear interactions (see theorem 4.1). For this reason, the theoretical derivations in section 4.4 guarantee that a CNN F_θ with two layers, ReLU^2 as activation function, and ten channels is sufficient to represent the semi-discretization of the PDE corresponding to centered finite differences of the second order. This architecture is precisely the one we use in the experiments.

The dynamics of this system are more complicated than those of the heat equation. As presented in appendix 4.A.3, we generate the initial conditions to train and test the network similarly to the heat equation, and the time step δt has the same value. To obtain the results in figure 4.6.3, we select only the training and test initial conditions for which $\|U^0\|_F > 10$. We choose the value 10 to obtain a more uniform dataset and avoid working with inputs of entirely different scales. This change is due to the low frequency of the initial conditions with small norms obtained through random data generation. Overall, the numerical results in figure 4.6.3 align with those we get for the other two PDEs. As with the heat equation, the dissipative nature of this PDE makes noise injection less effective than it is for linear advection, and possibly different regularization

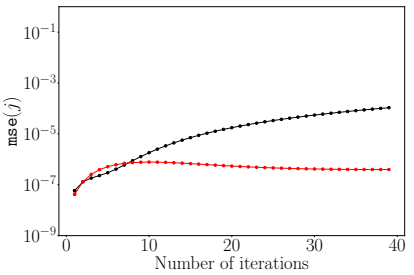
strategies are needed to improve the stability of the network.



(a) Average relative error over a batch of 30 test initial conditions.



(b) Maximum absolute error over a batch of 30 test initial conditions.



(c) Average of MSE over a batch of 30 test initial conditions.

Figure 4.6.3: Test errors for the Fisher equation.

4.7 Conclusion and further work

In this manuscript, we presented expressivity results for two-layer CNNs used for approximating temporal sequences. We focused on PDE space-time observations, examining CNNs' ability to represent PDE solutions for PDEs with quadratic nonlinear terms.

Exploiting the connections between finite difference operators and discrete convolution, we showed that it is sufficient to consider relatively small two-layer networks, where the size increases linearly in the number of quadratic interactions in the unknown model. Moreover, the networks that we investigated can represent broader classes of sequence data than just PDE solutions.

We also experimentally analyzed the effects of norm preservation and noise injection. Norm preservation as a means to improve the network stability can be beneficial for other PDE-generated datasets, such as for the Schrödinger equation, see [52]. Furthermore, the effects of other conservation laws could

also be considered. Our proposed approach to leveraging the conservation laws relies on projection methods, which extend naturally to other conserved properties. However, it would be interesting to understand if stability is enhanced by applying these other conservation properties and if the simplicity of projection methods becomes a limiting factor in the extension to more general conservation laws or if it remains a valuable strategy.

While our theoretical results rely on specific choices of activation functions, in practice, many more activation functions are successfully included in neural networks for time-series approximations. Hence, it is of interest to extend our results to other classes of functions.

Acknowledgments

This work was partially supported by a grant from the Simons Foundation (DM), the ERCIM Alain Bensoussan Fellowship Programme and Marie Skłodowska-Curie grant agreement No 101108679 (JJ). Additional funding was received from the European Union's Horizon 2020 research and innovation program under the MSCA-ETN grant agreement No 860124 and MSCA-SE grant agreement 101131557, from the Research Council of Norway, and from the Trond Mohn Foundation (EC) and (BO).

Appendix

4.A Data generation

We have relied on finite element discretizations to generate data throughout our numerical experiments in section 4.6. This dataset can be found in [32]. For completeness, we now briefly outline the methods used. Throughout, spatially, we utilize piecewise continuous linear Lagrange finite elements which, to a reasonable extent, respect the PDE dynamics. Crucially, any generated data must be interpolated (preferably in a structure-preserving manner) as a set of matrices. Recall that the matrix entries represent points in two-dimensional space, with each matrix corresponding to a discrete "snapshot" of the solution in time. With this matrix structure in mind, we choose to mesh our domain with regular quadrilaterals (as opposed to the more typical triangulation) and utilize bi-quadratic basis functions. Indeed, by doing so with linear elements, the degrees of freedom of the methods will be represented precisely by the matrix values. In particular, we shall write this finite element space as $\mathbb{V}$, which implicitly depends on the meshing of our domain. Throughout, we fix our matrix dimension to be $\mathbb{R}^{100\times 100}$, which fixes our mesh resolution to be $p = 100$, where p corresponds to the number of subrectangles in both the x and y direction. Temporally, we utilize second-order implicit time-stepping methods and define the temporal evolution through the time step δt. For ease of implementation, we use the finite element library Firedrake [49], and our implementation can be found in [33]. Each simulation is initialized by "random" initial data, which is problem-dependent, as described below.

4.A.1 Linear advection

Let $u = u(t, x, y)$, where $(x, y) \in \Omega := [0, 1]^2$ doubly periodic and $t \in [0, T]$. Then, we may express linear advection as

$$\begin{cases} \partial_t u = b \cdot \nabla u, \\ u\left(0, x, y\right) = u_0\left(x, y\right) \end{cases}$$

where $b \in \mathbb{R}^2$ is some specified constant. To clarify exposition, we shall focus on the first step of the method, which may be easily extrapolated at all times. Let U_0 be given by the interpolation of the initial data u_0 into the finite element space, then the solution at the next time step is given by seeking $U_1 \in \mathbb{V}$ such that

$$\int_\Omega \left(\frac{U_1 - U_0}{\delta t} + b \cdot \nabla U_{\frac{1}{2}} \right) \phi \, \mathrm{d}x \, \mathrm{d}y \quad \forall \phi \in \mathbb{V},$$

where $U_{\frac{1}{2}} = \frac{1}{2}\left(U_0 + U_1\right)$.

One fundamental property of this method is that it preserves both a discrete mass and momentum. To be more concise, the mass $\int_\Omega U_1 \, \mathrm{d}x \, \mathrm{d}y = \int_\Omega U_0 \, \mathrm{d}x \, \mathrm{d}y$ is constant over time, as can be observed by the periodic boundary conditions and after choosing $\phi = 1$. More importantly, the momentum $\int_\Omega U_1^2 \, \mathrm{d}x \, \mathrm{d}y = \int_\Omega U_0^2 \, \mathrm{d}x \, \mathrm{d}y$ is conserved, as may be observed after choosing $\phi = U_{\frac{1}{2}}$. Conservation of momentum is equivalent to preserving the norm of the underlying matrix.

In section 4.6.1, our random initial conditions are generated by

$$u_0 = \sin\left(2\pi\alpha_1\left(x - x_s\right)\right)\cos\left(2\pi\alpha_2\left(y - y_s\right)\right) + 1,$$

where $\alpha_i \sim U(\{5, 6, 7, 8\})$ are independently sampled with equal likelihood and $x_s, y_s \sim U([0, 1])$ are sampled from a unitary uniform distribution. Further, we fix the time step to be $\delta t = 0.02$ and $\vec{b} = (1, 1)$ throughout the experiments.

4.A.2 Heat equation

Here, we utilize the same setup as the heat equation. That is to say, we let $u = u(t, x, y)$ where $(x, y) \in \Omega = [0, 1]^2$ (doubly periodic) and $t \in [0, T]$, be given by

$$\begin{cases} u_t = \alpha \Delta u \\ u(0, x, y) = u_0(x, y), \end{cases}$$

where $\alpha \in \mathbb{R}$ is the dissipation constant. Letting U_0 be the interpolation of some given randomized initial data into the finite element space $\mathbb{V}$, the numerical method is described by seeking $U_1 \in \mathbb{V}$ such that

$$\int_\Omega \left(\left(\frac{U_1 - U_0}{\delta t} \right) \phi + \alpha \nabla U_{\frac{1}{2}} \cdot \nabla \phi \right) \mathrm{d}x \, \mathrm{d}y = 0 \quad \forall \phi \in \mathbb{V}.$$

In section 4.6.2, we exploit the dissipative behavior of this equation. Indeed, this numerical method respects the physical rate of dissipation. By choosing

$\phi = U^{\frac{1}{2}}$, we observe

$$\int_\Omega U_1^2 \,\mathrm{d}\,x\,\mathrm{d}\,y = \int_\Omega \left(U_0^2 - \alpha \nabla U_{\frac{1}{2}}^2\right) \mathrm{d}\,x\,\mathrm{d}\,y,$$

which is consistent with the true rate of dissipation in the heat equation.

In section 4.6.2, our random initial conditions are generated by

$$u_0 = \sin\left(k\pi\left(x - x_p\right)\right)\sin\left(k\pi\left(y - y_p\right)\right), \tag{4.A.1}$$

where $k \sim U(\{2,3,4,5,6,7\})$ is randomly selected and $x_p, y_p \sim \mathcal{N}(1, 0.5)$ are normally distributed with mean 1 and variance 0.5. Further, throughout our experiments, we fix $\delta t = 0.024$ and $\alpha = 0.01$.

4.A.3 Fisher equation

By modifying the heat equation (maintaining our doubly periodic spatial domain), let us consider a nonlinear reaction-diffusion equation. In particular, let $u = u(t, x, y)$ solve the reaction-diffusion equation

$$\begin{cases} u_t = \alpha \Delta u + u\,(1 - u) \\ u(0, x, y) = u_0(x, y), \end{cases} \tag{4.A.2}$$

where u_0 represents our randomized initial data. Letting U_0 be given by the interpolation of u_0 into the finite element space, the first step of the method is given by seeking $U_1 \in \mathbb{V}$ such that

$$\int_\Omega \left(\frac{U_1 - U_0}{\delta t}\right)\phi + \alpha \nabla U_{\frac{1}{2}} \cdot \nabla \phi - \left(U_{\frac{1}{2}} + \frac{1}{3}\left(U_1^2 + U_1 U_0 + U_0^2\right)\right)\phi \,\mathrm{d}\,x\,\mathrm{d}\,y = 0,$$

for all $\phi \in \mathbb{V}$. Note here that the choice of temporal discretization for the nonlinear term is not unique, and we have chosen a second-order accurate temporal discretization. In section 4.6.3, the random initial conditions generated are the same as for the heat equation due to similarities in the setup (see (4.A.1)).

Bibliography

[1] Milton Abramowitz, Irene A Stegun, and Robert H Romer. Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables, 1988. 153

[2] Ben Adcock, Simone Brugiapaglia, Nick Dexter, and Sebastian Moraga. Near-optimal learning of Banach-valued, high-dimensional functions via deep neural networks. *arXiv preprint arXiv:2211.12633*, 2022. 158

[3] Antonio Alguacil, Wagner Gonçalves Pinto, Michael Bauerheim, Marc C Jacob, and Stéphane Moreau. Effects of boundary conditions in fully convolutional networks for learning spatio-temporal dynamics. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 102–117. Springer, 2021. 148

[4] Christine Allen-Blanchette, Sushant Veer, Anirudha Majumdar, and Naomi Ehrich Leonard. LagNetViP: A Lagrangian Neural Network for Video Prediction. *arXiv preprint arXiv:2010.12932*, 2020. 145

[5] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020. 144

[6] Jānis Bajārs. Locally-symplectic neural networks for learning volume-preserving dynamics. *Journal of Computational Physics*, page 111911, 2023. 146

[7] Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019. 145

[8] Tom Bertalan, Felix Dietrich, Igor Mezić, and Ioannis G Kevrekidis. On learning Hamiltonian systems from data. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 29(12), 2019. 145

[9] Alex Bihlo. A generative adversarial network approach to (ensemble) weather prediction. *Neural Networks*, 139:1–16, jul 2021. 144

[10] Chris M Bishop. Training with Noise is Equivalent to Tikhonov Regularization. *Neural Computation*, 7(1):108–116, 1995. 148, 159

[11] John P Boyd. *Chebyshev and Fourier Spectral Methods*. Courier Corporation, 2001. 144

[12] Susanne Brenner and Ridgway Scott. *The Mathematical Theory of Finite Element Methods*, volume 15. Springer Science & Business Media, 2007. 144

[13] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020. 144

[14] Elsa Cardoso-Bihlo and Alex Bihlo. Exactly conservative physics-informed neural networks and deep operator networks for dynamical systems. *arXiv preprint arXiv:2311.14131*, 2023. 146

[15] Elena Celledoni, Andrea Leone, Davide Murari, and Brynjulf Owren. Learning Hamiltonians of constrained mechanical systems. *Journal of Computational and Applied Mathematics*, 417:114608, 2023. 4, 149

[16] Elena Celledoni, Davide Murari, Brynjulf Owren, Carola-Bibiane Schönlieb, and Ferdia Sherry. Dynamical Systems–Based Neural Networks. *SIAM Journal on Scientific Computing*, 45(6):A3071–A3094, 2023. 3, 95, 145, 146

[17] Zhengdao Chen, Jianyu Zhang, Martín Arjovsky, and Léon Bottou. Symplectic Recurrent Neural Networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020. 48, 70, 71, 73, 149, 226, 227, 231, 235, 237

[18] Syed OwaisAli Chishti, Sana Riaz, Muhammad BilalZaib, and Mohammad Nauman. Self-Driving Cars Using CNN and Q-Learning. In *2018 IEEE 21st International Multi-Topic Conference (INMIC)*, pages 1–7. IEEE, 2018. 144

[19] Taco Cohen and Max Welling. Group Equivariant Convolutional Networks. In *International Conference on Machine Learning*, pages 2990–2999. PMLR, 2016. 146

[20] Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific Machine Learning Through Physics–Informed Neural Networks: Where we are and What's Next. *Journal of Scientific Computing*, 92(3), jul 2022. 144

[21] Bin Dong, Qingtang Jiang, and Zuowei Shen. Image Restoration: Wavelet Frame Shrinkage, Nonlinear Evolution PDEs, and Beyond. *Multiscale Modeling & Simulation*, 15(1):606–660, 2017. 145

[22] Weinan E. A Proposal on Machine Learning via Dynamical Systems. *Communications in Mathematics and Statistics*, 1(5):1–11, 2017. 48, 147

[23] Leah Edelstein-Keshet. *Mathematical Models in Biology*. SIAM, 2005. 144

[24] Moshe Eliasof, Eldad Haber, and Eran Treister. PDE-GCN: Novel Architectures for Graph Neural Networks Motivated by Partial Differential Equations. *Advances in Neural Information Processing Systems*, 34:3836–3849, 2021. 9, 95, 99, 101, 113, 145

[25] Lawrence C. Evans. *Partial Differential Equations*, volume 19 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 1998. 144

[26] Stathi Fotiadis, Eduardo Pignatelli, Mario Lino Valencia, Chris Cantwell, Amos Storkey, and Anil A Bharath. Comparing recurrent and convolutional neural networks for predicting wave propagation. *arXiv preprint arXiv:2002.08981*, 2020. 145

[27] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. 8, 159, 259

[28] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian Neural Networks. *Advances in Neural Information Processing Aystems*, 32, 2019. 145, 226, 227, 237

[29] Bartosz A Grzybowski. *Chemistry in Motion: Reaction-Diffusion Systems for Micro - and Nanotechnology*. John Wiley & Sons, 2009. 144

[30] Eldad Haber and Lars Ruthotto. Stable Architectures for Deep Neural Networks. *Inverse problems*, 34(1):014004, 2017. 147

[31] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*. Springer, 2006. 146

[32] James Jackaman and Davide Murari. Data used in Predictions Based on Pixel Data: Insights from PDEs and Finite Differences. 10.5281/zenodo.11549488, 2024. 168

[33] James Jackaman and Davide Murari. Implementation used in 'Predictions Based on Pixel Data: Insights from PDEs and Finite Differences'. 10.5281/zenodo.11619634, 2024. 161, 168

[34] Pengzhan Jin, Zhen Zhang, Aiqing Zhu, Yifa Tang, and George Em Karniadakis. SympNets: Intrinsic structure-preserving symplectic networks for identifying Hamiltonian systems. *Neural Networks*, 132:166–179, 2020. 48, 58, 70, 71, 146, 158

[35] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021. 48, 144

[36] Asif Khan and Amos J Storkey. Hamiltonian Latent Operators for content and motion disentanglement in image sequences. *Advances in Neural Information Processing Systems*, 35:7250–7263, 2022. 145

[37] Jason M Klusowski and Andrew R Barron. Approximation by Combinations of ReLU and Squared ReLU Ridge Functions With ℓ^1 and ℓ^0 Controls. *IEEE Transactions on Information Theory*, 64(12):7649–7656, 2018. 154

[38] Randall J LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. SIAM, 2007. 144

[39] Qing Li, Weidong Cai, Xiaogang Wang, Yun Zhou, David Dagan Feng, and Mei Chen. Medical image classification with convolutional neural network. In *2014 13th international Conference on Control Automation Robotics & Vision (ICARCV)*, pages 844–848. IEEE, 2014. 144

[40] Zichao Long, Yiping Lu, and Bin Dong. PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925, 2019. 145, 148

[41] Michaël Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *CoRR*, abs/1511.05440, 2015. 145, 148

[42] Laurent Meunier, Blaise J Delattre, Alexandre Araujo, and Alexandre Allauzen. A Dynamical System Perspective for Lipschitz Neural Networks. In *International Conference on Machine Learning*, pages 15484–15500. PMLR, 2022. 145

[43] Sergiu Oprea, Pablo Martinez-Gonzalez, Alberto Garcia-Garcia, John Alejandro Castro-Vargas, Sergio Orts-Escolano, Jose Garcia-Rodriguez, and Antonis Argyros. A Review on Deep Learning Techniques for Video Prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(6):2806–2826, 2020. 145

[44] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 154, 161

[45] Allan Pinkus. Approximation theory of the MLP model in neural networks. *Acta numerica*, 8:143–195, 1999. 3, 55, 155

[46] William K Pratt. *Digital Image Processing: PIKS Scientific Inside*, volume 4. Wiley Online Library, 2007. 145

[47] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical Mathematics*, volume 37. Springer Science & Business Media, 2010. 153

[48] 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, feb 2019. 144

[49] Florian Rathgeber, David A. Ham, Lawrence Mitchell, Michael Lange, Fabio Luporini, Andrew T. T. McRae, Gheorghe-Teodor Bercea, Graham R. Markall, and Paul H. J. Kelly. Firedrake: Automating the Finite Element Method by Composing Abstractions. *ACM Trans. Math. Software*, 43(3):Art. 24, 27, 2017. 168

[50] Samuel Rudy, Alessandro Alla, Steven L Brunton, and J Nathan Kutz. Data-Driven Identification of Parametric Partial Differential Equations. *SIAM Journal on Applied Dynamical Systems*, 18(2):643–660, 2019. 145

[51] Lars Ruthotto and Eldad Haber. Deep Neural Networks Motivated by Partial Differential Equations. *Journal of Mathematical Imaging and Vision*, 62(3):352–364, 2020. 48, 145

[52] JM Sanz-Serna. Methods for the numerical solution of the nonlinear schrödinger equation. *mathematics of computation*, 43(167):21–27, 1984. 166

[53] Sebastian Scher and Gabriele Messori. Predicting Weather Forecast Uncertainty With Machine Learning. *Quarterly Journal of the Royal Meteorological Society*, 144(717):2830–2841, oct 2018. 144

[54] William E Schiesser. *The Numerical Method of Lines: Integration of Partial Differential Equations*. Elsevier, 2012. 149

[55] William E Schiesser and Graham W Griffiths. *A Compendium of Partial Differential Equation Models: Method of Lines Analysis with Matlab*. Cambridge University Press, 2009. 149

[56] Ferdia Sherry, Elena Celledoni, Matthias J Ehrhardt, Davide Murari, Brynjulf Owren, and Carola-Bibiane Schönlieb. Designing stable neural networks using convex analysis and ODEs. *Physica D: Nonlinear Phenomena*, 463:134159, 2024. 35, 145

[57] Bart MN Smets, Jim Portegies, Erik J Bekkers, and Remco Duits. PDE-based Group Equivariant Convolutional Neural Networks. *Journal of Mathematical Imaging and Vision*, 65(1):209–239, 2023. 145

[58] Leslie N Smith. Cyclical Learning Rates for Training Neural Networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472. IEEE, 2017. 162

[59] David So, Wojciech Mańke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V Le. Searching for Efficient Transformers for Language Modeling. *Advances in Neural Information Processing Systems*, 34:6010–6022, 2021. 154

[60] Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. Curriculum Learning: A Survey. *International Journal of Computer Vision*, 130(6):1526–1565, 2022. 162

[61] Walter A. Strauss. *Partial Differential Equations: An Introduction*. John Wiley & Sons, Ltd., Chichester, second edition, 2008. 144

[62] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR*, 2014. 145

[63] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer Nature, 2022. 145

[64] James William Thomas. *Numerical Partial Differential Equations: Finite Difference Methods*, volume 22. Springer Science & Business Media, 2013. 153

[65] Geoffrey K Vallis. *Atmospheric and Oceanic Fluid Dynamics: Fundamentals and Large-Scale Circulation*. Cambridge University Press, 2017. 144

[66] Rui Wang, Karthik Kashinath, Mustafa Mustafa, Adrian Albert, and Rose Yu. Towards Physics-informed Deep Learning for Turbulent Flow Prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1457–1466, 2020. 146

[67] Rui Wang, Robin Walters, and Rose Yu. Incorporating Symmetry into Deep Dynamics Models for Improved Generalization. In *International Conference on Learning Representations*, 2021. 146

[68] Yunbo Wang, Haixu Wu, Jianjin Zhang, Zhifeng Gao, Jianmin Wang, S Yu Philip, and Mingsheng Long. PredRNN: A Recurrent Neural Network for Spatiotemporal Predictive Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):2208–2225, 2022. 145

[69] Hao Xu, Haibin Chang, and Dongxiao Zhang. DL-PDE: Deep-Learning Based Data-Driven Discovery of Partial Differential Equations from Discrete and Noisy Data. *Communications in Computational Physics*, 29(3):698–728, 2021. 145

[70] Aiqing Zhu, Pengzhan Jin, and Yifa Tang. Deep Hamiltonian networks based on symplectic integrators. *arXiv preprint arXiv:2004.13830*, 2020. 70, 146, 231, 243

Part II: Solving and Discovering Differential Equations

Lie Group integrators for mechanical systems

Elena Celledoni, Ergys Çokaj, Andrea Leone, Davide Murari, and Brynjulf Owren

International Journal of Computer Mathematics

Abstract. Since their introduction in the 1990s, Lie group integrators have become a method of choice in many application areas. These include multibody dynamics, shape analysis, data science, image registration, and biophysical simulations. Two important classes of intrinsic Lie group integrators are the Runge–Kutta–Munthe–Kaas methods and the commutator-free Lie group integrators. We give a short introduction to these classes of methods. The Hamiltonian framework is attractive for many mechanical problems, and in particular, we shall consider Lie group integrators for problems on cotangent bundles of Lie groups where a number of different formulations are possible. There is a natural symplectic structure on such manifolds and through variational principles one may derive symplectic Lie group integrators. We also consider the practical aspects of the implementation of Lie group integrators, such as adaptive time stepping. The theory is illustrated by applying the methods to two nontrivial applications in mechanics. One is the N-fold spherical pendulum where we introduce the restriction of the adjoint action of the group $SE(3)$ to TS^2, the tangent bundle of the two-dimensional sphere. Finally, we show how Lie group integrators can be applied to model the controlled path of a payload being transported by two rotors. This problem is modeled on $\mathbb{R}^6 \times \big(SO(3) \times \mathfrak{so}(3)\big)^2 \times (TS^2)^2$ and put in a format where Lie group integrators can be applied.

5.1 Introduction

In many physical problems, including multibody dynamics, the configuration space is not a linear space but rather consists of a collection of rotations and translations. A simple example is the free rigid body, whose configuration space consists of 3D rotations. A more advanced example is the simplified model of the human body, where the skeleton at a given time is described as a system of interacting rods and joints. Mathematically, the structure of such problems is usually best described as a manifold. Since manifolds, by definition, can be equipped with local coordinates, one can always describe and simulate such systems locally as if they were linear spaces. There are of course many choices of local coordinates, for rotations some famous ones are: Euler angles, the Tait-Bryan angles commonly used in aerospace applications, the unit length quaternions, and the exponentiated skew-symmetric 3×3-matrices. Lie group integrators represent a somewhat different strategy. Rather than specifying a choice of local coordinates from the outset, in this approach, the model and the numerical integrator are expressed entirely in terms of a Lie group and its action on the phase space. This often leads to a more abstract and simpler formulation of the mechanical system and of the numerical schemes, deferring further details to the implementation phase.

In the literature, one can find many different types and formats of Lie group integrators. Some of these are completely general and intrinsic, meaning that

they only make use of inherent properties of Lie groups and manifolds as was suggested in [11, 40, 6]. However, many numerical methods have been suggested that add structure or utilize properties that are specific to a particular Lie group or manifold. Notable examples of this are the methods based on canonical coordinates of the second kind [45], and the methods based on the Cayley transformation [31, 13], applicable e.g. to the rotation groups and Euclidean groups. In some applications, e.g., in multibody systems, it may be useful to formulate the problem as a mix between Lie groups and kinematic constraints, introducing, for instance, Lagrange multipliers. Sometimes, this may lead to more practical implementations where a basic general setup involving Lie groups can be further equipped with different constraints, depending on the particular application. Such constrained formulations are outside the scope of the present paper. It should also be noted that the Lie group integrators devised here do not make any a priori assumptions about how the manifold is represented.

The applications of Lie group integrators for mechanical problems also have a long history. Two of the early important contributions were the Newmark methods of Simo and Vu–Quoc [49] and the symplectic and energy-momentum methods by Lewis and Simo [31]. Mechanical systems are often described as Euler–Lagrange equations or as Hamiltonian systems on manifolds, with or without external forces, [28]. Important ideas for the discretization of mechanical systems also originated from the work of Moser and Veselov [51, 37] on discrete integrable systems. This work served as motivation for further developments in the field of geometric mechanics and for the theory of (Lie group) discrete variational integrators [27, 20, 29]. The majority of Lie group methods found in the literature are one-step type generalizations for classical methods, such as Runge–Kutta type formulas. In mechanical engineering, the classical BDF methods have played an important role and were recently generalized [54] to Lie groups. Similarly, the celebrated α-method for linear spaces proposed by Hilber, Hughes, and Taylor [22] has been popular for solving problems in multibody dynamics, and in [1, 2, 4] this method is generalized to a Lie group integrator.

The literature on Lie group integrators is rich and diverse, and the interested reader may consult the surveys [26, 10, 7, 44] and Chapter 4 of the monograph [18] for further details.

In this paper, we discuss different ways of applying Lie group integrators to simulate the dynamics of mechanical multibody systems. Our point of departure is the formulation of the models as differential equations on manifolds. Assuming to be given either a Lie group acting transitively on the manifold $\mathcal{M}$ or a set of frame vector fields on $\mathcal{M}$, we use them to describe the mechanical

system and further to build the numerical integrator. We shall here mostly consider schemes of the types commonly known as Crouch–Grossman methods [11], Runge–Kutta–Munthe–Kaas methods [39, 40] and Commutator-free Lie group methods [6].

The choice of Lie group action is often not unique and thus the same mechanical system can be described in different equivalent ways. Under numerical discretization, the different formulations can lead to the conservation of different geometric properties of the mechanical system. In particular, we explore the effect of these different formulations on a selection of examples in multibody dynamics. Lie group integrators have been successfully applied for the simulation of mechanical systems, and in problems of control, bio-mechanics and other engineering applications, see for example [46], [27] [9], [25]. The present work is motivated by applications in modeling and simulation of slender structures like Cosserat rods and beams [49], and one of the examples presented here is the application to a chain of pendula. Another example considers an application for the controlled dynamics of a multibody system.

In section 5.2, we give a review of the methods using only the essential intrinsic tools of Lie group integrators. The algorithms are simple and amenable to a coordinate-free description suited to object-oriented implementations. In section 5.3, we discuss Hamiltonian systems on Lie groups, and we present three different Lie group formulations of the heavy top equations. These systems (and their Lagrangian counterpart) often arise in applications as building blocks of more realistic systems, which also comprise damping and control forces. In section 5.4, we discuss some ways of adapting the integration step size in time. In section 5.5, we consider the application to a chain of pendula. Section 5.6 considers the application of a multibody system of interest in the simulation and control of drone dynamics.

5.2 Lie group integrators

5.2.1 The formulation of differential equations on manifolds

Lie group integrators solve differential equations whose solutions evolve on a manifold $\mathcal{M}$. For ease of notation, we restrict the discussion to the case of autonomous vector fields, although allowing for explicit t-dependence could easily have been included. This means that we seek a curve $y(t) \in \mathcal{M}$ whose tangent at any point coincides with a vector field $F \in \mathcal{X}(\mathcal{M})$ and passing through a designated initial value y_0 at $t = t_0$

$$\dot{y}(t) = F|_{y(t)}, \qquad y(t_0) = y_0. \tag{5.2.1}$$

Before addressing numerical methods for solving (5.2.1), it is necessary to introduce a convenient way of representing the vector field F. There are different ways of doing this. One is to furnish $\mathcal{M}$ with a transitive action $\psi : G \times \mathcal{M} \to \mathcal{M}$ by some Lie group G of dimension $d \geq \dim \mathcal{M}$. We denote the action of g on m as $g \cdot m$, i.e. $g \cdot m = \psi(g, m)$. Let $\mathfrak{g}$ be the Lie algebra of G, and denote by $\exp : \mathfrak{g} \to G$ the exponential map. We define $\psi_* : \mathfrak{g} \to \mathcal{X}(\mathcal{M})$ to be the infinitesimal generator of the action, i.e.

$$F_\xi\Big|_m = \psi_*(\xi)\Big|_m = \frac{d}{dt}\bigg|_{t=0} \psi\Big(\exp(t\xi), m\Big) \tag{5.2.2}$$

The transitivity of the action now ensures that $\psi_*(\mathfrak{g})\big|_m = T_m\mathcal{M}$ for any $m \in \mathcal{M}$, such that any tangent vector $v_m \in T_m\mathcal{M}$ can be represented as $v_m = \psi_*(\xi_v)\big|_m$ for some $\xi_v \in \mathfrak{g}$ (ξ_v may not be unique). Consequently, for any vector field $F \in \mathcal{X}(\mathcal{M})$ there exists a map $f : \mathcal{M} \to \mathfrak{g}$[1] such that

$$F|_m = \psi_*\big(f(m)\big)\Big|_m, \quad \text{for all } m \in \mathcal{M} \tag{5.2.3}$$

This is the original tool [40] for representing a vector field on a manifold with a group action. Another approach was used in [11] where a set of *frame vector fields* $E_1, \ldots, E_d$ in $\mathcal{X}(\mathcal{M})$ was introduced assuming that for every $m \in \mathcal{M}$,

$$\operatorname{span}\Big\{E_1\big|_m, \ldots, E_d\big|_m\Big\} = T_m\mathcal{M}.$$

Then, for any vector field $F \in \mathcal{X}(\mathcal{M})$ there are, in general non-unique, functions $f_i : \mathcal{M} \to \mathbb{R}$, which can be chosen with the same regularity as F, such that

$$F|_m = \sum_{i=1}^{d} f_i(m)\, E_i\big|_m.$$

A fixed vector $\xi \in \mathbb{R}^d$ will define a vector field F_ξ on $\mathcal{M}$ similar to (5.2.2)

$$F_\xi\Big|_m = \sum_{i=1}^{d} \xi_i E_i|_m \tag{5.2.4}$$

If $\xi_i = f_i(p)$ for some $p \in \mathcal{M}$, the corresponding F_ξ will be a vector field in the linear span of the frame which coincides with F at the point p. Such a vector field was named by [11] as a *the vector field frozen at* p.

The two formulations just presented are, in many cases, connected and can then be used in an equivalent manner. Suppose that $e_1, \ldots, e_d$ is a basis of the

[1] If the Lie group action is smooth, a map f of the same regularity as F can be found [53]

Lie algebra $\mathfrak{g}$, then we can simply define frame vector fields as $E_i = \psi_*(e_i)$ and the vector field we aim to describe is,

$$F|_m = \psi_*\left(f(m)\right)\Big|_m = \psi_*\left(\sum_i f_i(m)\, e_i\right)\Bigg|_m = \sum_i f_i\, E_i|_m .$$

As mentioned above, there is a non-uniqueness issue when defining a vector field by means of a group action or a frame. A more fundamental description can be obtained using the machinery of connections. The assumption is that the simply connected manifold $\mathcal{M}$ is equipped with a connection which is flat and has constant torsion. Then F_p, the frozen vector field of F at p defined above, can be defined as the unique element $F_p \in \mathcal{X}(\mathcal{M})$ satisfying

1. $F_p|_p = F|_p$
2. $\nabla_X F_p = 0$ for any $X \in \mathcal{X}(M)$.

So F_p is the vector field that coincides with F at p and is parallel transported to any other point on $\mathcal{M}$ by the connection ∇. Since the connection is flat, the parallel transport from the point p to another point $m \in \mathcal{M}$ does not depend on the chosen path between the two points. For further details, see e.g. [32].

Example 1. For mechanical systems on Lie groups, two important constructions are the adjoint and coadjoint representations. For every $g \in G$ there is an automorphism $\mathrm{Ad}_g : \mathfrak{g} \to \mathfrak{g}$ defined as

$$\mathrm{Ad}_g(\xi) = TL_g \circ TR_{g^{-1}}(\xi)$$

where L_g and R_g are the left and right multiplications respectively, $L_g(h) = gh$ and $R_g(h) = hg$. Since Ad is a representation, i.e., $\mathrm{Ad}_{gh} = \mathrm{Ad}_g \circ \mathrm{Ad}_h$, it also defines a left Lie group action by G on $\mathfrak{g}$. From this definition and a duality pairing $\langle \cdot, \cdot \rangle$ between $\mathfrak{g}$ and $\mathfrak{g}^*$, we can also derive a representation on $\mathfrak{g}^*$ denoted Ad^*_g, simply by

$$\left\langle \mathrm{Ad}^*_g(\mu), \xi \right\rangle = \left\langle \mu, \mathrm{Ad}_g(\xi) \right\rangle, \quad \xi \in \mathfrak{g},\ \mu \in \mathfrak{g}^*.$$

The action $g \cdot \mu = \mathrm{Ad}^*_{g^{-1}}(\mu)$ has infinitesimal generator given as

$$\psi_*(\xi)\Big|_\mu = -\mathrm{ad}^*_\xi \mu$$

Following [34], for a Hamiltonian $H : T^*G \to \mathbb{R}$, define H^- to be its restriction to $\mathfrak{g}^*$. Then, the Lie-Poisson reduction of the dynamical system is defined on $\mathfrak{g}^*$ as

$$\dot{\mu} = -\mathrm{ad}^*_{\frac{\partial H^-}{\partial \mu}} \mu$$

and this vector field is precisely of the form (5.2.3) with $f(\mu) = \frac{\partial H^-}{\partial \mu}(\mu)$. A side effect of this is that the integral curves of these Lie-Poisson systems preserve coadjoint orbits, making the coadjoint action an attractive choice for Lie group integrators.

Let us now detail the situation for the very simple case where $G = SO(3)$. The Lie algebra $\mathfrak{so}(3)$ can be modeled as 3×3 skew-symmetric matrices, and via the standard basis, we identify each such matrix $\hat{\xi}$ by a vector $\xi \in \mathbb{R}^3$, this identification is known as the hat map

$$\hat{\xi} = \begin{bmatrix} 0 & -\xi_3 & \xi_2 \\ \xi_3 & 0 & -\xi_1 \\ -\xi_2 & \xi_1 & 0 \end{bmatrix}. \tag{5.2.5}$$

Now, we also write the elements of $\mathfrak{so}^*(3)$ as vectors in $\mathbb{R}^3$ with duality pairing $\langle \mu, \xi \rangle = \mu^T \xi$. With these representations, we find that the coadjoint action can be expressed as

$$g \cdot \mu = \psi(g, \mu) = \mathrm{Ad}^*_{g^{-1}} \mu = g\mu$$

the rightmost expression being a simple matrix-vector multiplication. Since g is orthogonal, it follows that the coadjoint orbits foliate 3-space into spherical shells, and the coadjoint action is transitive on each of these orbits. The free rigid body can be cast as a problem on $T^*SO(3)$ with a left-invariant Hamiltonian, which reduces to the function

$$H^-(\mu) = \frac{1}{2} \left\langle \mu, \mathbb{I}^{-1} \mu \right\rangle$$

on $\mathfrak{so}^*(3)$ where $\mathbb{I} : \mathfrak{so}(3) \to \mathfrak{so}^*(3)$ is the inertia tensor. From this, we can now set $f(\mu) = \partial H^- / \partial \mu = \mathbb{I}^{-1} \mu$. We then recover the Euler free rigid body equation as

$$\dot{\mu} = \psi_* \left(f(\mu) \right) \Big|_{\mu} = -\mathrm{ad}^*_{\mathbb{I}^{-1}\mu} \mu = -\mathbb{I}^{-1} \mu \times \mu$$

where the last expression involves the cross product of vectors in $\mathbb{R}^3$.

5.2.2 Two classes of Lie group integrators

The simplest numerical integrator for linear spaces is the explicit Euler method. Given an initial value problem $\dot{y} = F(y)$, $y(0) = y_0$ the method is defined as $y_{n+1} = y_n + hF(y_n)$ for some step size h. In the spirit of the previous section, one could think of the Euler method as the h-flow of the constant vector field $F_{y_n}(y) = F(y_n)$, that is

$$y_{n+1} = \exp\left(hF_{y_n} \right) y_n.$$

This definition of the Euler method also makes sense when F is replaced by a vector field on some manifold. In this general situation it is known as the Lie–Euler method.

We shall here consider the two classes of methods known as Runge–Kutta–Munthe–Kaas (RKMK) methods and Commutator-free Lie group methods.

For RKMK methods, the underlying idea is to transform the problem from the manifold $\mathcal{M}$ to the Lie algebra $\mathfrak{g}$, take a time step, and map the result back to $\mathcal{M}$. The transformation we use is

$$y(t) = \exp(\sigma(t)) \cdot y_0, \quad \sigma(0) = 0.$$

The transformed differential equation for $\sigma(t)$ makes use of the derivative of the exponential mapping. The reader should consult [40] for details about the derivation. We give the final result

$$\dot{\sigma}(t) = \operatorname{dexp}^{-1}_{\sigma(t)} \Big(f\left(\exp(\sigma(t)) \cdot y_0\right) \Big). \tag{5.2.6}$$

The map $v \mapsto \operatorname{dexp}_u(v)$ is linear and invertible when u belongs to some sufficiently small neighborhood of $0 \in \mathfrak{g}$. It has an expansion in nested Lie brackets [21]. Using the operator $\operatorname{ad}_u(v) = [u, v]$ and its powers $\operatorname{ad}^2_u v = [u, [u, v]]$ etc, one can write

$$\operatorname{dexp}_u(v) = \left.\frac{e^z - 1}{z}\right|_{z=\operatorname{ad}_u} (v) = v + \frac{1}{2}[u, v] + \frac{1}{6}[u, [u, v]] + \cdots \tag{5.2.7}$$

and the inverse is

$$\operatorname{dexp}^{-1}_u(v) = \left.\frac{z}{e^z - 1}\right|_{z=\operatorname{ad}_u} (v) = v - \frac{1}{2}[u, v] + \frac{1}{12}[u, [u, v]] + \cdots. \tag{5.2.8}$$

The RKMK methods are now obtained simply by applying some standard Runge–Kutta method to the transformed equation (5.2.6) with a time step h, using the initial value $\sigma(0) = 0$. This leads to an output $\sigma_1 \in \mathfrak{g}$ and one simply sets $y_1 = \exp(\sigma_1) \cdot y_0$. Then one repeats the procedure replacing y_0 by y_1 in the next step etc. While solving (5.2.6), one needs to evaluate $\operatorname{dexp}^{-1}_u(v)$ as a part of the process. This can be done by truncating the series (5.2.8) since $\sigma(0) = 0$ implies that we always evaluate $\operatorname{dexp}^{-1}_u$ with $u = \mathcal{O}(h)$, and thus, the kth iterated commutator $\operatorname{ad}^k_u = \mathcal{O}(h^k)$. For a given Runge–Kutta method, there are some clever tricks that can be done to minimize the total number of commutators to be included from the expansion of $\operatorname{dexp}^{-1}_u v$, see [5, 41]. We give

here one concrete example of an RKMK method proposed in [5]

$$
\begin{aligned}
f_{n,1} &= hf\left(y_n\right),\\
f_{n,2} &= hf\left(\exp\left(\tfrac{1}{2}f_{n,1}\right)\cdot y_n\right),\\
f_{n,3} &= hf\left(\exp\left(\tfrac{1}{2}f_{n,2}-\tfrac{1}{8}\left[f_{n,1},f_{n,2}\right]\right)\cdot y_n\right),\\
f_{n,4} &= hf\left(\exp\left(f_{n,3}\right)\cdot y_n\right),\\
y_{n+1} &= \exp\left(\tfrac{1}{6}\left(f_{n,1}+2f_{n,2}+2f_{n,3}+f_{n,4}-\tfrac{1}{2}\left[f_{n,1},f_{n,4}\right]\right)\right)\cdot y_n.
\end{aligned}
$$

The other option is to compute the exact expression for $\operatorname{dexp}_u^{-1}(v)$ for the particular Lie algebra we use. For instance, it was shown in [8] that for the Lie algebra $\mathfrak{so}(3)$ one has

$$
\operatorname{dexp}_u^{-1}(v) = v - \frac{1}{2}u\times v + \alpha^{-2}\left(1-\tfrac{\alpha}{2}\cot\tfrac{\alpha}{2}\right)u\times(u\times v).
$$

We will present the corresponding formula for $\mathfrak{se}(3)$ in Section 5.2.3.

The second class of Lie group integrators to be considered here are the commutator-free methods, named this way in [6] to emphasize the contrast to RKMK schemes which usually include commutators in the method format. These schemes include the Crouch-Grossman methods [11] and they have the format

$$
\begin{aligned}
Y_{n,r} &= \exp\left(h\sum_k \alpha^k_{r,J} f_{n,k}\right)\cdots\exp\left(h\sum_k \alpha^k_{r,1} f_{n,k}\right)\cdot y_n\\
f_{n,r} &= f\left(Y_{n,r}\right)\\
y_{n+1} &= \exp\left(h\sum_k \beta^k_J f_{n,k}\right)\cdots\exp\left(h\sum_k \beta^k_1 f_{n,k}\right)\cdot y_n.
\end{aligned}
$$

Here the Runge–Kutta coefficients $\alpha^k_{r,j}$, β^r_j are related to a classical Runge–Kutta scheme with coefficients a^k_r, b_r in that $a^k_r = \sum_j \alpha^k_{r,j}$ and $b_r = \sum_j \beta^r_j$. The $\alpha^k_{r,j}$, β^r_j are usually chosen to obtain computationally inexpensive schemes with the highest possible order of convergence. The computational complexity of the above schemes depends on the cost of computing an exponential as well as of evaluating the vector field. Therefore, it makes sense to keep the number of exponentials J in each stage as low as possible, and possibly also the number of stages s. A trick proposed in [6] was to select coefficients that make it possible to reuse exponentials from one stage to another. This is perhaps

best illustrated through the following example from [6], a generalization of the classical 4th-order Runge–Kutta method.

$$\begin{aligned}
Y_{n,1} &= y_n \\
Y_{n,2} &= \exp\left(\tfrac{1}{2}hf_{n,1}\right)\cdot y_n \\
Y_{n,3} &= \exp\left(\tfrac{1}{2}hf_{n,2}\right)\cdot y_n \\
Y_{n,4} &= \exp\left(hf_{n,3} - \tfrac{1}{2}hf_{n,1}\right)\cdot Y_{n,2} \\
y_{n+\frac{1}{2}} &= \exp\left(\tfrac{1}{12}h\left(3f_{n,1} + 2f_{n,2} + 2f_{n,3} - f_{n,4}\right)\right)\cdot y_n \\
y_{n+1} &= \exp\left(\tfrac{1}{12}h\left(-f_{n,1} + 2f_{n,2} + 2f_{n,3} + 3f_{n,4}\right)\right)\cdot y_{n+\frac{1}{2}}
\end{aligned} \tag{5.2.9}$$

where $f_{n,i} = f(Y_{n,i})$. Here, we see that one exponential is saved in computing $Y_{n,4}$ by making use of $Y_{n,2}$.

5.2.3 An exact expression for $\operatorname{dexp}_u^{-1}(v)$ in $\mathfrak{se}(3)$

As an alternative to using a truncated version of the infinite series for $\operatorname{dexp}_u^{-1}$ (5.2.8), one can consider exact expressions obtained for certain Lie algebras. Since $\mathfrak{se}(3)$ is particularly important in applications to mechanics, we give here its exact expression. For this, we represent elements of $\mathfrak{se}(3)$ as a pair $(A, a) \in \mathbb{R}^3 \times \mathbb{R}^3 \cong \mathbb{R}^6$, the first component corresponding to a skew-symmetric matrix $\hat{A}$ via (5.2.5) and a is the translational part. Now, let $\varphi(z)$ be a real analytic function at $z = 0$. We define

$$\varphi_+(z) = \frac{\varphi(iz) + \varphi(-iz)}{2}, \qquad \varphi_-(z) = \frac{\varphi(iz) - \varphi(-iz)}{2i}$$

We next define the four functions

$$g_1(z) = \frac{\varphi_-(z)}{z},\ \tilde{g}_1(z) = \frac{g_1'(z)}{z}, \quad g_2(z) = \frac{\varphi(0) - \varphi_+(z)}{z^2},\ \tilde{g}_2(z) = \frac{g_2'(z)}{z}$$

and the two scalars $\rho = A^T a$, $\alpha = \|A\|_2$. One can show that for any (A, a) and (B, b) in $\mathfrak{se}(3)$, it holds that

$$\varphi\left(\operatorname{ad}_{(A,a)}\right)(B, b) = (C, c)$$

where

$$\begin{aligned}
C &= \varphi(0)B + g_1(\alpha)\,A \times B + g_2(\alpha)\,A \times \left(A \times B\right) \\
c &= \varphi(0)b + g_1(\alpha)\left(a \times B + A \times b\right) + \rho\tilde{g}_1(\alpha)\,A \times B + \rho\tilde{g}_2(\alpha)\,A \times \left(A \times B\right) \\
&\quad + g_2(\alpha)\left(a \times \left(A \times B\right) + A \times \left(a \times B\right) + A \times \left(A \times b\right)\right).
\end{aligned}$$

Considering for instance (5.2.8), we may now use $\varphi(z) = \frac{z}{e^z-1}$ to calculate

$$g_1(z) = -\frac{1}{2},\ \tilde{g}_1(z) = 0,\ g_2(z) = \frac{1 - \frac{z}{2}\cot\frac{z}{2}}{z^2},\ \tilde{g}_2(z) = \frac{1}{z}\frac{d}{dz}g_2(z),\ \varphi(0) = 1,$$

and thereby obtain an expression for $\mathrm{dexp}^{-1}_{(A,a)}(B,b)$ with the formula above.

Similar types of formulas are known for computing the matrix exponential as well as functions of the ad-operator for several other Lie groups of small and medium dimension. For instance in [38] a variety of coordinate mappings for rigid body motions are discussed. For Lie algebras of larger dimension, both the exponential mapping and dexp_u^{-1} may become computationally infeasible. For these cases, one may benefit from replacing the exponential with some other coordinate map for the Lie group $\phi : \mathfrak{g} \to G$. One option is to use canonical coordinates of the second kind [45]. Then, for some Lie groups, such as the orthogonal, unitary, and symplectic groups, there exist other maps that can be used and which are computationally less expensive. A popular choice is the Cayley transformation [13].

5.3 Hamiltonian systems on Lie groups

In this section, we consider Hamiltonian systems on Lie groups. These systems (and their Lagrangian counterpart) often appear in mechanics applications as building blocks for more realistic systems with additional damping and control forces. We consider canonical systems on the cotangent bundle of a Lie group and Lie-Poisson systems which can arise by symmetry reduction or otherwise. We illustrate the various cases with different formulations of the heavy top system.

5.3.1 Semidirect products

The coadjoint action by G on $\mathfrak{g}^*$ is denoted Ad^*_g defined for any $g \in G$ as

$$\left\langle \mathrm{Ad}^*_g \mu, \xi \right\rangle = \left\langle \mu, \mathrm{Ad}_g \xi \right\rangle, \quad \forall \xi \in \mathfrak{g}, \tag{5.3.1}$$

where $\mathrm{Ad} : \mathfrak{g} \to \mathfrak{g}$ is the adjoint representation and for a duality pairing $\langle \cdot, \cdot \rangle$ between $\mathfrak{g}^*$ and $\mathfrak{g}$.

We consider the cotangent bundle of a Lie group G, T^*G and identify it with $G \times \mathfrak{g}^*$ using the right multiplication $R_g : G \to G$ and its tangent mapping $R_{g*} :=$

TR_g. The cartesian product $G \times \mathfrak{g}^*$ can be given a semidirect product structure that turns it into a Lie group $\mathbf{G} := G \ltimes \mathfrak{g}^*$ where the group multiplication is

$$\left(g_1, \mu_1\right) \cdot \left(g_2, \mu_2\right) = \left(g_1 \cdot g_2, \mu_1 + \mathrm{Ad}^*_{g_1^{-1}} \mu_2\right). \tag{5.3.2}$$

Acting by left multiplication any vector field $F \in \mathcal{X}(\mathbf{G})$ is expressed by means of a map $f \colon \mathbf{G} \to T_e\mathbf{G}$,

$$F\left(g, \mu\right) = T_e R_{(g,\mu)} f\left(g, \mu\right) = \left(R_{g*} f_1, f_2 - \mathrm{ad}^*_{f_1} \mu\right), \tag{5.3.3}$$

where $f_1 = f_1(g, \mu) \in \mathfrak{g}$, $f_2 = f_2(g, \mu) \in \mathfrak{g}^*$ are the two components of f.

5.3.2 Symplectic form and Hamiltonian vector fields

The right trivialized[2] symplectic form pulled back to $\mathbf{G}$ reads

$$\begin{aligned} &\omega_{(g,\mu)} \left(\left(R_{g*} \xi_1, \delta \nu_1\right), \left(R_{g*} \xi_2, \delta \nu_2\right) \right) \\ &= \left\langle \delta \nu_2, \xi_1 \right\rangle - \left\langle \delta \nu_1, \xi_2 \right\rangle - \left\langle \mu, [\xi_1, \xi_2] \right\rangle, \qquad \xi_1, \xi_2 \in \mathfrak{g}. \end{aligned} \tag{5.3.4}$$

See [31] for more details, proofs, and for the left trivialized symplectic form. The vector field F is a Hamiltonian vector field if it satisfies

$$\mathrm{i}_F \omega = dH,$$

for some Hamiltonian function $H : T^*G \to \mathbb{R}$, where i_F is defined as $\mathrm{i}_F(X) := \omega(F, X)$ for any vector field X. This implies that the map f for such a Hamiltonian vector field gets the form

$$f(g, \mu) = \left(\frac{\partial H}{\partial \mu}(g, \mu), -R_g^* \frac{\partial H}{\partial g}(g, \mu) \right). \tag{5.3.5}$$

The following is a one-parameter family of symplectic Lie group integrators on T^*G:

[2] $\omega_{(g,\mu)}$ is obtained from the natural symplectic form on T^*G (which is a differential two-form), defined as

$$\Omega_{\left(g, p_g\right)} \left(\left(\delta v_1, \delta \pi_1\right), \left(\delta v_2, \delta \pi_2\right) \right) = \left\langle \delta \pi_2, \delta v_1 \right\rangle - \left\langle \delta \pi_1, \delta v_2 \right\rangle,$$

by right trivialization.

$$M_\theta = \mathrm{dexp}^*_{-\xi}(\mu_0 + \mathrm{Ad}^*_{\exp(\theta\xi)}(\bar{n})) - \theta \mathrm{dexp}^*_{-\theta\xi}\mathrm{Ad}^*_{\exp(\theta\xi)}(\bar{n}), \tag{5.3.6}$$

$$(\xi, \bar{n}) = hf\left(\exp(\theta\xi)\cdot g_0, M_\theta\right), \tag{5.3.7}$$

$$(g_1, \mu_1) = \left(\exp(\xi), \mathrm{Ad}^*_{\exp((\theta-1)\xi)}\bar{n}\right)\cdot(g_0, \mu_0). \tag{5.3.8}$$

For higher-order integrators of this type and a complete treatment, see [3].

5.3.3 Reduced equations Lie Poisson systems

A mechanical system formulated on the cotangent bundle T^*G with a left or right invariant Hamiltonian can be reduced to a system on $\mathfrak{g}^*$ [33]. In fact for a Hamiltonian H right invariant under the left action of G, $\frac{\partial H}{\partial g} = 0$, and from (5.3.3) and (5.3.5) we get for the second equation

$$\dot{\mu} = \mp \mathrm{ad}^*_{\frac{\partial H}{\partial \mu}}\mu, \tag{5.3.9}$$

where the positive sign is used in case of left invariance (see e.g. section 13.4 in [35]). The solution to this system preserves coadjoint orbits, thus using the Lie group action

$$g\cdot\mu = \mathrm{Ad}^*_{g^{-1}}\mu,$$

to build a Lie group integrator results in preservation of such coadjoint orbits. Lie group integrators for this interesting case were studied in [15].

The Lagrangian counterpart to these Hamiltonian equations are the Euler–Poincaré equations[3], [24].

5.3.4 Three different formulations of the heavy top equations

The heavy top is a simple test example for illustrating the behavior of Lie group methods. We will consider three different formulations for this mechanical system. The first formulation is on $T^*SO(3)$ where the equations are canonical Hamiltonian, a second point of view is that the system is a Lie–Poisson system on $\mathfrak{se}^*(3)$, and finally it is canonical Hamiltonian on a larger group with a quadratic Hamiltonian function. The three different formulations suggest the use of different Lie group integrators.

[3] The Euler–Poincaré equations are Euler–Lagrange equations with respect to a Lagrange–d'Alembert principle obtained taking constraint variations.

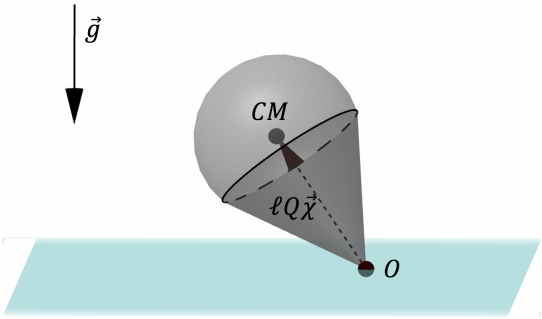


Figure 5.3.1: Illustration of the heavy top, where CM is the center of mass of the body, O is the fixed point, $\vec{g}$ is the gravitational acceleration vector, and $\ell, Q, \vec{\chi}$ follow the notation introduced in Section 5.3.4

Heavy top equations on $T^*SO(3)$.

The heavy top is a rigid body with a fixed point in a gravitational field. The phase space of this mechanical system is $T^*SO(3)$, where the equations of the heavy top are in canonical Hamiltonian form. Assuming (Q, p) are coordinates for $T^*SO(3)$, $\Pi = (T_eL_Q)^*(p)$ is the left trivialized or body momentum. The Hamiltonian of the heavy top is given in terms of (Q, Π) as

$$H\colon SO(3) \ltimes \mathfrak{so}^*(3) \to \mathbb{R}, \quad H(Q,\Pi) = \frac{1}{2}\left\langle \Pi, \mathbb{I}^{-1}\Pi \right\rangle + Mg\ell\,\Gamma\cdot\mathcal{X}, \quad \Gamma = Q^{-1}\Gamma_0,$$

where $\mathbb{I}\colon \mathfrak{so}(3) \to \mathfrak{so}^*(3)$ is the inertia tensor, here represented as a diagonal 3×3 matrix, $\Gamma = Q^{-1}\Gamma_0$, where $\Gamma_0 \in \mathbb{R}^3$ is the axis of the spatial coordinate system parallel to the direction of gravity but pointing upwards, M is the mass of the body, g is the gravitational acceleration, $\mathcal{X}$ is the body fixed unit vector of the oriented line segment pointing from the fixed point to the center of mass of the body, ℓ is the length of this segment. The equations of motion on $SO(3)\ltimes \mathfrak{so}^*(3)$ are

$$\dot{\Pi} = \Pi \times \mathbb{I}^{-1}\Pi + Mg\ell\,\Gamma \times \mathcal{X}, \tag{5.3.10}$$

$$\dot{Q} = Q\widehat{\mathbb{I}^{-1}\Pi}. \tag{5.3.11}$$

The identification of $T^*SO(3)$ with $SO(3)\ltimes\mathfrak{so}^*(3)$ via right trivialization leads to the spatial momentum variable $\pi = (T_eR_Q)^*(p) = Q\Pi$. The equations written in the space variables (Q, π) get the form

$$\dot{\pi} = Mg\ell\,\Gamma_0 \times Q\mathcal{X}, \tag{5.3.12}$$

$$\dot{Q} = \hat{\omega}Q \qquad \omega = Q\mathbb{I}^{-1}Q^T\pi, \tag{5.3.13}$$

where the first equation states that the component of π parallel to Γ_0 is constant in time. These equations can be obtained from (5.3.3) and (5.3.5) on the right trivialized $T^*SO(3)$, $SO(3) \ltimes \mathfrak{so}^*(3)$, with the heavy top Hamiltonian and the symplectic Lie group integrators (5.3.7)-(5.3.8) can be applied in this case. Similar methods were proposed in [31] and [48].

Heavy top equations on $\mathfrak{se}^*(3)$

The Hamiltonian of the heavy top is not invariant under the action of $SO(3)$, so the equations (5.3.10)-(5.3.11) given in Section 5.3.4 cannot be reduced to $\mathfrak{so}^*(3)$. Nevertheless the heavy top equations are Lie–Poisson on $\mathfrak{se}^*(3)$, [52, 17, 47].

Observe that the equations of the heavy top on $T^*SO(3)$ (5.3.10)-(5.3.11) can be easily modified eliminating the variable $Q \in SO(3)$ and replacing it with $\Gamma \in \mathbb{R}^3$ $\Gamma = Q^{-1}\Gamma_0$ to obtain

$$\dot{\Pi} = \Pi \times \mathbb{I}^{-1}\Pi + Mg\ell\,\Gamma \times \mathcal{X}, \tag{5.3.14}$$

$$\dot{\Gamma} = \Gamma \times \left(\mathbb{I}^{-1}\Pi\right). \tag{5.3.15}$$

We will see that the solutions of these equations evolve on $\mathfrak{se}^*(3)$. In what follows, we consider elements of $\mathfrak{se}^*(3)$ to be pairs of vectors in $\mathbb{R}^3$, e.g. (Π, Γ). Correspondingly the elements of $SE(3)$ are represented as pairs $(g, \mathbf{u})$ with $g \in SO(3)$ and $\mathbf{u} \in \mathbb{R}^3$. The group multiplication in $SE(3)$ is then

$$\left(g_1, \mathbf{u}_1\right) \cdot \left(g_2, \mathbf{u}_2\right) = \left(g_1 g_2, g_1 \mathbf{u}_2 + \mathbf{u}_1\right),$$

where $g_1 g_2$ is the product in $SO(3)$ and $g_1\mathbf{u}$ is the product of a 3×3 orthogonal matrix with a vector in $\mathbb{R}^3$. The coadjoint representation and its infinitesimal generator on $\mathfrak{se}^*(3)$ take the form

$$\begin{aligned}
\mathrm{Ad}^*_{(g,\mathbf{u})}\left(\Pi, \Gamma\right) &= \left(g^{-1}\left(\Pi - \mathbf{u} \times \Gamma\right), g^{-1}\Gamma\right), \\
\mathrm{ad}^*_{(\xi,\mathbf{u})}\left(\Pi, \Gamma\right) &= \left(-\xi \times \Pi - \mathbf{u} \times \Gamma, -\xi \times \Gamma\right).
\end{aligned}$$

Using this expression for $\mathrm{ad}^*_{(\xi,\mathbf{u})}$ with $(\xi = \frac{\partial H}{\partial \Pi}, \mathbf{u} = \frac{\partial H}{\partial \Gamma})$, it can be easily seen that the equations (5.3.9) in this setting reproduce the heavy top equations (5.3.14)-(5.3.15). Therefore the equations are Lie–Poisson equations on $\mathfrak{se}^*(3)$. However, since the heavy top is a rigid body with a fixed point and there are no translations, these equations do not arise from a reduction of $T^*SE(3)$. Moreover, the Hamiltonian on $\mathfrak{se}^*(3)$ is not quadratic, and the equations are not geodesic equations. Implicit and explicit Lie group integrators applicable to this formulation of the heavy top equations and preserving coadjoint orbits were discussed in [15], for a variable step size integrator applied to this formulation of the heavy top see [12].

Heavy top equations with quadratic Hamiltonian.

We rewrite the heavy top equations one more time considering the constant vector $\mathbf{p} = -Mg\ell\mathcal{X}$ as a momentum variable conjugate to the position $\mathbf{q} \in \mathbb{R}^3$ and where $\mathbf{p} = Q^{-1}\Gamma_0 + \dot{\mathbf{q}}$, and the Hamiltonian is a quadratic function of Π, Q, $\mathbf{p}$ and $\mathbf{q}$:

$$H\colon T^*SO(3) \times \mathbb{R}^{3^*} \times \mathbb{R}^3 \to \mathbb{R},$$
$$H((\Pi, Q), (\mathbf{p}, \mathbf{q})) = \frac{1}{2}\left\langle \Pi, \mathbb{I}^{-1}\Pi \right\rangle + \frac{1}{2}\left\| \mathbf{p} - Q^{-1}\Gamma_0 \right\|^2 - \frac{1}{2}\left\| Q^{-1}\Gamma_0 \right\|^2,$$

see [23, section 8.5]. This Hamiltonian is invariant under the left action of $SO(3)$. The corresponding equations are canonical on $T^*S \equiv S \ltimes \mathfrak{s}^*$ where $S = SO(3) \times \mathbb{R}^3$ with Lie algebra $\mathfrak{s} := \mathfrak{so}(3) \times \mathbb{R}^3$ and T^*S can be identified with $T^*SO(3) \times \mathbb{R}^{3^*} \times \mathbb{R}^3$. The equations are

$$\dot{\Pi} = \Pi \times \mathbb{I}^{-1}\Pi - \left(Q^{-1}\Gamma_0\right) \times \mathbf{p}, \tag{5.3.16}$$
$$\dot{Q} = Q\widehat{\mathbb{I}^{-1}\Pi}, \tag{5.3.17}$$
$$\dot{\mathbf{p}} = \mathbf{0}, \tag{5.3.18}$$
$$\dot{\mathbf{q}} = \mathbf{p} - Q^{-1}\Gamma_0, \tag{5.3.19}$$

and in the spatial momentum variables

$$\dot{\pi} = -\Gamma_0 \times Q\mathbf{p}, \tag{5.3.20}$$
$$\dot{Q} = \hat{\omega}Q, \quad \omega = Q\mathbb{I}^{-1}Q^T\pi, \tag{5.3.21}$$
$$\dot{\mathbf{p}} = \mathbf{0}, \tag{5.3.22}$$
$$\dot{\mathbf{q}} = \mathbf{p} - Q^{-1}\Gamma_0. \tag{5.3.23}$$

Similar formulations were considered in [30] for the stability analysis of an underwater vehicle. A similar but different formulation of the heavy top was considered in [4].

Numerical experiments.

We apply various implicit Lie group integrators to the heavy top system. The test problem we consider is the same as in [4], where $Q(0) = I$, $\ell = 2$, $M = 15$ $\mathbb{I} = \mathrm{diag}(0.234375, 0.46875, 0.234375)$, $\pi(0) = \mathbb{I}(0, 150, -4.61538)$, $\mathcal{X} = (0, 1, 0)$ $\Gamma_0 = (0, 0, -9.81)$.

In Figure 5.3.2 we report the performance of the symplectic Lie group integrators (5.3.6)-(5.3.8) applied both on the equations (5.3.12)-(5.3.13) with $\theta = 0$,

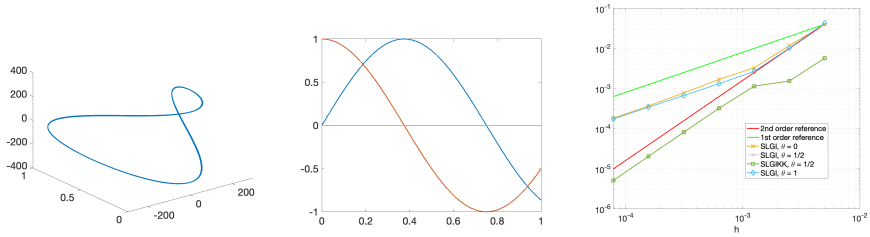


Figure 5.3.2: Symplectic Lie group integrators integration on the time interval $[0,1]$. Left: 3D plot of $M\ell Q^{-1}\Gamma_0$. Center: components of $Q\mathcal{X}$. The left and center plots are computed with the same step size. Right: verification of the order of the methods.

$\theta = \frac{1}{2}$ and $\theta = 1$ (SLGI), and to the equations (5.3.20)-(5.3.23) with $\theta = \frac{1}{2}$ (SLGIKK). The methods with $\theta = \frac{1}{2}$ attain order 2. In Figure 5.3.3, we show the energy error for the symplectic Lie group integrators with $\theta = \frac{1}{2}$ and $\theta = 0$ integrating with step size $h = 0.01$ for 6000 steps.

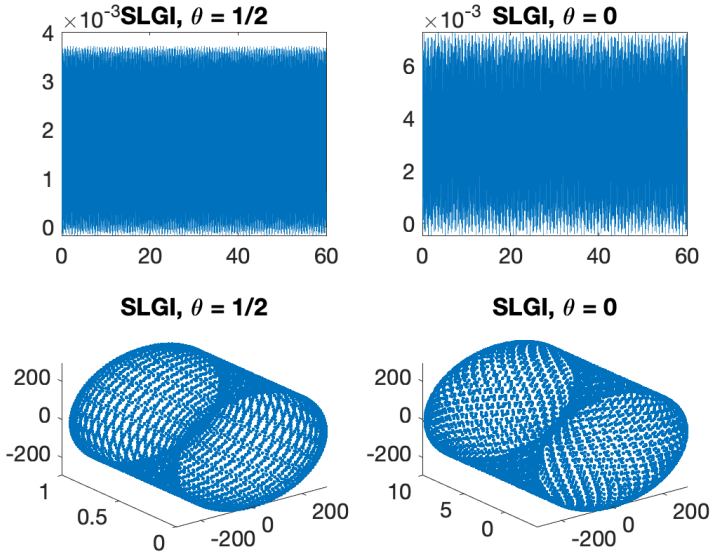


Figure 5.3.3: Symplectic Lie group integrators, long time integration, $h = 0.01$, 6000 steps.. Top: energy error, bottom 3D plot of $M\ell Q^{-1}\Gamma_0$.

5.4 Variable step size

One approach for varying the step size is based on the use of an embedded Runge–Kutta pair. This principle can be carried from standard Runge–Kutta methods in vector spaces to the present situation with RKMK and commutator-free schemes via minor modifications. We briefly summarize the main principle of embedded pairs before giving more specific details for the case of Lie group integrators. This approach is very well documented in the literature and goes back to Merson [36], and a detailed treatment can be found in [19, p. 165–168].

An embedded pair consists of a main method used to propagate the numerical solution, together with some auxiliary method that is only used to obtain an estimate of the local error. This local error estimate is, in turn, used to derive a step size adjustment formula that attempts to keep the local error estimate approximately equal to some user-defined tolerance tol in every step. Suppose the main method is of order p and the auxiliary method is of order $\tilde{p} \neq p$. [4] Both methods are applied to the input value y_n and yields approximations y_{n+1} and $\tilde{y}_{n+1}$ respectively, using the same step size h_{n+1}. Now, some distance measure[5] between y_{n+1} and $\tilde{y}_{n+1}$ provides an estimate e_{n+1} for the size of the local truncation error. Thus, $e_{n+1} = Ch_{n+1}^{\tilde{p}+1} + \mathcal{O}(h^{\tilde{p}+2})$. Aiming at $e_{n+1} \approx \text{tol}$ in every step, one may use a formula of the type

$$h_{n+1} = \theta \left(\frac{\text{tol}}{e_{n+1}} \right)^{\frac{1}{\tilde{p}+1}} h_n \tag{5.4.1}$$

where θ is a 'safety factor', typically chosen between 0.8 and 0.9. In case the step is rejected because $e_n > \text{tol}$, we can redo the step with a step size obtained by the same formula. We summarize the approach in the following algorithm

Given y_n, h_n, tol
Let $h := h_n$
repeat
  Compute y_{n+1}, $\tilde{y}_{n+1}$, e_{n+1} from y_n, h
  Update step size $h := \theta \left(\frac{\text{tol}}{e_{n+1}} \right)^{\alpha} h$
  accepted := $e_{n+1} < \text{tol}$
  if accepted

[4] In this paper, we will assume $\tilde{p} < p$ in which case the local error estimate is relevant for the approximation $\tilde{y}_{n+1}$

[5] There are many options for how to do this in practice, and the choice may also depend on the application. For example, a Riemannian metric is a natural and robust alternative.

update step index: $n := n+1$
$h_n := h$
until accepted

Here we have used again the safety factor θ, and the parameter α is generally chosen as $\alpha = \frac{1}{1+\min(p,\tilde{p})}$.

5.4.1 RKMK methods with variable step size

We need to specify how to calculate the quantity e_{n+1} in each step. For RKMK methods, the situation is simplified by the fact that we are solving the local problem (5.2.6) in the linear space $\mathfrak{g}$, where the known theory can be applied directly. So any standard embedded pair of Runge–Kutta methods described by coefficients $(a_{ij}, b_i, \tilde{a}_{ij}, \tilde{b}_i)$ of orders $(p, \tilde{p})$ can be applied to the full dexpinv-equation (5.2.6) to obtain local Lie algebra approximations σ_1, $\tilde{\sigma}_1$ and one uses e.g. $e_{n+1} = \|\sigma_1 - \tilde{\sigma}_1\|$ (note that the equation itself depends on y_n). For methods that use a truncated version of the series for dexp_u^{-1}, one may also try to optimize performance by including commutators that are shared between the main method and the auxiliary scheme.

5.4.2 Commutator-free methods with variable step size

For the commutator-free methods of section 5.2.2, the situation is different since such methods do not have a natural local representation in a linear space. One can still derive embedded pairs, and this can be achieved by studying order conditions [43] as was done in [12]. Now one obtains after each step two approximations y_{n+1} and $\tilde{y}_{n+1}$ on $\mathcal{M}$ both by using the same initial value y_n and step size h_n. One must also have access to some metric d to calculate $e_{n+1} = d(y_{n+1}, \tilde{y}_{n+1})$ We give a few examples of embedded pairs.

Pairs of order $\left(p, \tilde{p}\right) = (3,2)$

It is possible to obtain embedded pairs of order 3(2) which satisfy the requirements above. We present two examples from [12]. The first one reuses the

second stage exponential in the update

$$\begin{aligned}
Y_{n,1} &= y_n \\
Y_{n,2} &= \exp\left(\tfrac{1}{3}hf_{n,1}\right)\cdot y_n \\
Y_{n,3} &= \exp\left(\tfrac{2}{3}hf_{n,2}\right)\cdot y_n \\
y_{n+1} &= \exp\left(h\left(-\tfrac{1}{12}f_{n,1}+\tfrac{3}{4}f_{n,3}\right)\right)\cdot Y_{n,2} \\
\tilde{y}_{n+1} &= \exp\left(\tfrac{1}{2}h\left(f_{n,2}+f_{n,3}\right)\right)\cdot y_n.
\end{aligned}$$

One could also have reused the third stage $Y_{n,3}$ in the update rather than $Y_{n,2}$.

$$\begin{aligned}
Y_{n,1} &= y_n \\
Y_{n,2} &= \exp\left(\tfrac{2}{3}hf_{n,1}\right)\cdot y_n \\
Y_{n,3} &= \exp\left(h(\tfrac{5}{12}f_{n,1}+\tfrac{1}{4}f_{n,2}\right)\cdot y_n \\
y_{n+1} &= \exp\left(h\left(-\tfrac{1}{6}f_{n,1}-\tfrac{1}{2}f_{n,2}+f_{n,3}\right)\right)\cdot Y_{n,3} \\
\tilde{y}_{n+1} &= \exp\left(\tfrac{1}{4}h\left(f_{n,1}+3f_{n,3}\right)\right)\cdot y_n.
\end{aligned}$$

It is always understood that the frozen vector fields are $f_{n,i} := f_{Y_{n,i}}$.

Order $(4,3)$

The procedure of deriving efficient pairs becomes more complicated as the order increases. In [12] a low cost pair of order $(4,3)$ was derived, in the sense that one attempted to minimize the number of stages and exponentials in the embedded pair as a whole. This came, however, at the expense of a relatively large error constant. So rather than presenting the method from that paper, we suggest a simpler procedure at the cost of some more computational work per step, we simply furnish the commutator-free method of section 5.2 by a third order auxiliary scheme. It can be described as follows:

1. Compute $Y_{n,i}$, $i=1\ldots,4$ and y_{n+1} from (5.2.9)
2. Compute an additional stage $\bar{Y}_{n,3}$ and then $\tilde{y}_{n+1}$ as

$$\begin{aligned}
\bar{Y}_{n,3} &= \exp\left(\tfrac{3}{4}hf_{n,2}\right)\cdot y_n \\
\tilde{y}_{n+1} &= \exp\left(\tfrac{h}{9}\left(-f_{n,1}+3f_{n,2}+4\bar{f}_{n,3}\right)\right)\cdot\exp\left(\tfrac{h}{3}f_{n,1}\right)\cdot y_n.
\end{aligned} \tag{5.4.2}$$

5.5 The N-fold 3D pendulum

In this section, we present a model for a system of N connected 3-dimensional pendulums. The modeling part comes from [28], and here we study the vector field describing the dynamics in order to re-frame it into the Lie group integrators setting described in the previous sections. The model we use is not completely realistic since, for example, it neglects possible interactions between pendulums and assumes ideal spherical joints between them. However, this is still a relevant example from the point of view of geometric numerical integration. More precisely, we show a possible way to work with a configuration manifold which is not a Lie group, applying the theoretical instruments introduced before. The Lagrangian we consider is a function from $(TS^2)^N$ to $\mathbb{R}$.

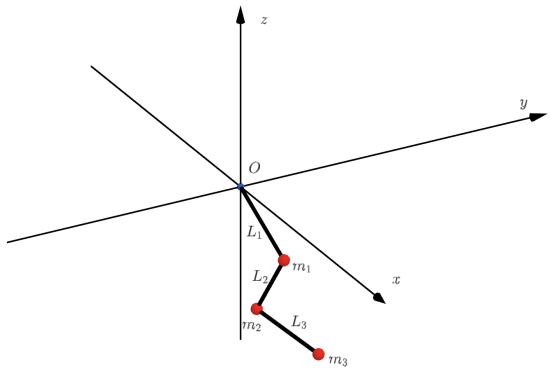


Figure 5.5.1: 3−fold pendulum at a fixed time instant, with fixed point placed at the origin.

Instead of the coordinates $(q_1, ..., q_N, \dot{q}_1, ..., \dot{q}_N)$, where $\dot{q}_i \in T_{q_i}S^2$, we choose to work with the angular velocities. Precisely,

$$T_{q_i}S^2 = \left\{ v \in \mathbb{R}^3 : \, v^T q_i = 0 \right\} = \langle q_i \rangle^{\perp} \subset \mathbb{R}^3,$$

and hence for any $\dot{q}_i \in T_{q_i}S^2$ there exist $\omega_i \in \mathbb{R}^3$ such that $\dot{q}_i = \omega_i \times q_i$, which can be interpreted as the angular velocity of q_i. So we can assume without loss of generality that $\omega_i^T q_i = 0$ (i.e. $\omega_i \in T_{q_i}S^2$) and pass to the coordinates $(q_1, \omega_1, q_2, \omega_2, ..., q_N, \omega_N) \in (TS^2)^N$ to describe the dynamics. In this section, we denote the masses of the pendulums with $m_1, ..., m_N$ and their lengths with $L_1, ..., L_N$. Figure 5.5.1 shows the case $N = 3$. We organize the section into three parts:

1. We define the transitive Lie group action used to integrate this model numerically,

2. We show a possible way to express the dynamics in terms of the infinitesimal generator of this action for the general case of N joint pendulums,
3. We focus on the case $N = 2$, as a particular example. For this setting, we present some numerical experiments comparing various Lie group integrators and some classical numerical integrators. Then, we conclude with numerical experiments on variable step size.

5.5.1 Transitive group action on $\left(TS^2\right)^N$

We characterize a transitive action for $(TS^2)^N$, starting with the case $N = 1$ and generalizing it to $N > 1$. The action we consider is based on the identification between $\mathfrak{se}(3)$, the Lie algebra of $SE(3)$, and $\mathbb{R}^6$. We start from the Ad-action of $SE(3)$ on $\mathfrak{se}(3)$ (see [23]), which writes

$$\mathrm{Ad} : SE(3) \times \mathfrak{se}(3) \to \mathfrak{se}(3),$$
$$\mathrm{Ad}\Big(\big(R, r\big), \big(u, v\big)\Big) = \big(Ru, Rv + \hat{r}Ru\big).$$

Since $\mathfrak{se}(3) \simeq \mathbb{R}^6$, the Ad-action allows us to define the following Lie group action on $\mathbb{R}^6$

$$\psi : SE(3) \times \mathbb{R}^6 \to \mathbb{R}^6, \ \psi\Big(\big(R, r\big), \big(u, v\big)\Big) = \big(Ru, Rv + \hat{r}Ru\big).$$

We can think of ψ as a Lie group action on TS^2 since, for any $q \in \mathbb{R}^3$, it maps points of

$$TS^2_{|q|} := \left\{\big(\tilde{q}, \tilde{\omega}\big) \in \mathbb{R}^3 \times \mathbb{R}^3 : \ \tilde{\omega}^T \tilde{q} = 0, \ |\tilde{q}| = |q|\right\} \subset \mathbb{R}^6$$

into other points of $TS^2_{|q|}$. Moreover, with standard arguments (see [42]), it is possible to prove that the orbit of a generic point $m = (q, \omega) \in \mathbb{R}^6$ with $\omega^T q = 0$ coincides with

$$\mathrm{Orb}(m) = TS^2_{|q|}.$$

In particular, when $q \in \mathbb{R}^3$ is a unit vector (i.e. $q \in S^2$), ψ allows us to define a transitive Lie group action on $TS^2 = TS^2_{|q|=1}$ which writes

$$\psi : SE(3) \times TS^2 \to TS^2$$
$$\psi\Big(\big(A, a\big), \big(q, \omega\big)\Big) := \psi_{(A,a)}\big(q, \omega\big) = \big(Aq, A\omega + \hat{a}Aq\big) = \big(\bar{q}, \bar{\omega}\big).$$

To conclude the description of the action, we report here its infinitesimal generator, which is fundamental in the Lie group integrator setting

$$\psi_*\Big(\big(u, v\big)\Big)\Big|_{(q,\omega)} = \big(\hat{u}q, \hat{u}\omega + \hat{v}q\big).$$

We can extend this construction to the case $N > 1$ in a natural way, i.e., through the action of a Lie group obtained from cartesian products of $SE(3)$ and equipped with the direct product structure. More precisely, we consider the group $G = (SE(3))^N$, and by direct product structure, we mean that for any pair of elements

$$\delta^{(1)} = (\delta_1^{(1)}, ..., \delta_N^{(1)}), \quad \delta^{(2)} = (\delta_1^{(2)}, ..., \delta_N^{(2)}) \in G,$$

denoted with $*$ the semidirect product of $SE(3)$, we define the product $\circ$ on G as

$$\delta^{(1)} \circ \delta^{(2)} := \left(\delta_1^{(1)} * \delta_1^{(2)}, ..., \delta_N^{(1)} * \delta_N^{(2)}\right) \in G.$$

With this group structure defined, we can generalize the action introduced for $N = 1$ to larger Ns as follows

$$\psi : \left(SE(3)\right)^N \times \left(TS^2\right)^N \to \left(TS^2\right)^N,$$

$$\psi\left(\left(A_1, a_1, ..., A_N, a_n\right), \left(q_1, \omega_1, ..., q_N, \omega_N\right)\right) =$$

$$= \left(A_1 q_1, A_1\omega_1 + \hat{a}_1 A_1 q_1, ..., A_N q_N, A_N\omega_N + \hat{a}_N A_N q_N\right),$$

whose infinitesimal generator writes

$$\psi_*\left(\xi\right)\Big|_m = \left(\hat{u}_1 q_1, \hat{u}_1\omega_1 + \hat{v}_1 q_1, ..., \hat{u}_N q_N, \hat{u}_N\omega_N + \hat{v}_N q_N\right),$$

where $\xi = [u_1, v_1, ..., u_N, v_N] \in \mathfrak{se}(3)^N$ and $m = (q_1, \omega_1, ..., q_N, \omega_N) \in (TS^2)^N$. We now have the only group action we need to deal with the $N-$fold spherical pendulum. In the following part of this section, we work on the vector field describing the dynamics and adapting it to the Lie group integrators setting.

5.5.2 Full chain

We consider the vector field $F \in \mathfrak{X}((TS^2)^N)$, describing the dynamics of the N-fold 3D pendulum, and we express it in terms of the infinitesimal generator of the action defined above. More precisely, we find a function $F : (TS^2)^N \to \mathfrak{se}(3)^N$ such that

$$\psi_*\left(f(m)\right)\Big|_m = F|_m, \ \forall m \in \left(TS^2\right)^N.$$

We omit the derivation of F starting from the Lagrangian of the system, which can be found in the section devoted to mechanical systems on $(S^2)^N$ of [28]. The configuration manifold of the system is $(S^2)^N$, while the Lagrangian, expressed in terms of the variables $(q_1, \omega_1, ..., q_N, \omega_N) \in (TS^2)^N$, writes

$$L\left(q, \omega\right) = T\left(q, \omega\right) - U\left(q\right) = \frac{1}{2}\sum_{i,j=1}^{N}\left(M_{ij}\omega_i^T \hat{q}_i^T \hat{q}_j \omega_j\right) - \sum_{i=1}^{N}\left(\sum_{j=i}^{N} m_j\right) g L_i e_3^T q_i,$$

where

$$M_{ij} = \left(\sum_{k=\max\{i,j\}}^{N} m_k \right) L_i L_j I_3 \in \mathbb{R}^{3\times 3}$$

is the inertia matrix of the system, I_3 is the 3×3 identity matrix, and $e_3 = [0,0,1]^T$. Noticing that when $i = j$ we get

$$\omega_i^T \hat{q}_i^T \hat{q}_i \omega_i = \omega_i^T \left(I_3 - q_i q_i^T \right) \omega_i = \omega_i^T \omega_i,$$

we simplify the notation writing

$$T\left(q,\omega\right) = \frac{1}{2} \sum_{i,j=1}^{N} \left(\omega_i^T R(q)_{ij} \omega_j \right)$$

where $R(q) \in \mathbb{R}^{3N\times 3N}$ is a symmetric block matrix defined as

$$R(q)_{ii} = \left(\sum_{j=i}^{N} m_j \right) L_i^2 I_3 \in \mathbb{R}^{3\times 3},$$

$$R(q)_{ij} = \left(\sum_{k=j}^{N} m_k \right) L_i L_j \hat{q}_i^T \hat{q}_j \in \mathbb{R}^{3\times 3} = R\left(q\right)_{ji}^T,\ i < j.$$

The vector field on which we need to work defines the following first-order ODE

$$\dot{q}_i = \omega_i \times q_i,\ i = 1,...,N,$$

$$R(q)\dot{\omega} = \left[\sum_{\substack{j=1 \\ j\neq i}}^{N} M_{ij} |\omega_j|^2 \hat{q}_i q_j - \left(\sum_{j=i}^{N} m_j \right) g L_i \hat{q}_i e_3 \right]_{i=1,...,N} \in \mathbb{R}^{3N}$$

By direct computation, it is possible to see that, for any $q = (q_1,...,q_N) \in (S^2)^N$ and $\omega \in T_{q_1} S^2 \times ... \times T_{q_N} S^2$, we have

$$\left(R\left(q\right)\omega \right)_i \in T_{q_i} S^2.$$

Therefore, there is a well-defined linear map

$$A_q : T_{q_1} S^2 \times ... \times T_{q_N} S^2 \to T_{q_1} S^2 \times ... \times T_{q_N} S^2, A_q\left(\omega\right) := R\left(q\right)\omega.$$

We can even notice that $R(q)$ defines a positive-definite bilinear form on this linear space since

$$\omega^T R\left(q\right)\omega = \sum_{i,j=1}^{N} \omega_i^T \hat{q}_i^T M_{ij} \hat{q}_j \omega_j = \sum_{i,j=1}^{N} \left(\hat{q}_i \omega_i \right)^T M_{ij} \left(\hat{q}_j \omega_j \right) = v^T M v > 0.$$

The last inequality holds because M is the inertia matrix of the system and hence it defines a symmetric positive-definite bilinear form on $T_{q_1}S^2 \times ... \times T_{q_N}S^2$, see, e.g., [16][6]. This implies the map A_q is invertible and hence we are ready to express the vector field in terms of the infinitesimal generator. We can rewrite the ODEs for the angular velocities as follows:

$$\dot{\omega} = A_q^{-1}\left(\left[g_1, ..., g_N\right]^T\right) = \begin{bmatrix} h_1(q,\omega) \\ \vdots \\ h_N(q,\omega) \end{bmatrix} = \begin{bmatrix} a_1(q,\omega) \times q_1 \\ \vdots \\ a_N(q,\omega) \times q_N \end{bmatrix}$$

where

$$g_i = g_i(q,\omega) = \sum_{\substack{j=1 \\ j\neq i}}^{N} M(q)_{ij} \left|\omega_j\right|^2 \hat{q}_i q_j - \left(\sum_{j=i}^{N} m_j\right) g L_i \hat{q}_i e_3,\ i = 1, ..., N$$

and $a_1, ..., a_N : (TS^2)^N \to \mathbb{R}^3$ are N functions whose existence is guaranteed by the analysis done above. Indeed, we can set $a_i(q,\omega) := q_i \times h_i(q,\omega)$ and conclude that a mapping f from $(TS^2)^N$ to $(\mathfrak{se}(3))^N$ such that

$$\psi_*\left(f(q,\omega)\right)\Big|_{(q,\omega)} = F|_{(q,\omega)}$$

is the following one

$$f(q,\omega) = \begin{bmatrix} \omega_1 \\ q_1 \times h_1 \\ \vdots \\ \omega_N \\ q_N \times h_N \end{bmatrix} \in \mathfrak{se}(3)^N \simeq \mathbb{R}^{6N}.$$

We will not go into the Hamiltonian formulation of this problem; however, we remark that a similar approach works even in that situation. Indeed, following the derivation presented in [28], we see that for a mechanical system on $(S^2)^N$ the conjugate momentum writes

$$T^*_{q_1}S^2 \times ... \times T^*_{q_N}S^2 \ni \pi = (\pi_1, ..., \pi_N),\ \text{where } \pi_i = -\hat{q}_i^2 \frac{\partial L}{\partial \omega_i}$$

[6] It follows from the definition of the inertia tensor, i.e.,

$$0 \leq \tilde{T}(q,\dot{q}) = \frac{1}{2}\sum_{i=1}^{N}\left(\sum_{j\geq i} m_j\right) L_i L_j \dot{q}_i^T \dot{q}_j := \frac{1}{2}\dot{q}^T M \dot{q}.$$

Moreover, in this situation, it is even possible to explicitly find the Cholesky factorization of the matrix M with an iterative algorithm.

and its components are still orthogonal to the respective base points $q_i \in S^2$. Moreover, Hamilton's equations take the form

$$\dot{q}_i = \frac{\partial H(q,\pi)}{\partial \pi_i} \times q_i,$$
$$\dot{\pi}_i = \frac{\partial H(q,\pi)}{\partial q_i} \times q_i + \frac{\partial H(q,\pi)}{\partial \pi_i} \times \pi_i,$$

which implies that setting

$$f(q,\pi) = \left[\partial_{q_1} H(q,\pi), \quad \partial_{\pi_1} H(q,\pi), \quad \ldots, \quad \partial_{q_N} H(q,\pi), \quad \partial_{\pi_N} H(q,\pi)\right]$$

we can represent even the Hamiltonian vector field of the $N-$fold 3D pendulum in terms of this group action.

Case $N = 2$

We have seen how it is possible to turn the equations of motion of a $N-$chain of pendulums into the Lie group integrators setting. Now, we focus on the example with $N = 2$ pendulums. The equations of motion write

$$\dot{q}_1 = \hat{\omega}_1 q_1, \quad \dot{q}_2 = \hat{\omega}_2 q_2,$$

$$R(q)\begin{bmatrix}\dot{\omega}_1 \\ \dot{\omega}_2\end{bmatrix} = \begin{bmatrix}\left(-m_2 L_1 L_2 \left|\omega_2\right|^2 \hat{q}_2 + \left(m_1+m_2\right) g L_1 \hat{e}_3\right) q_1 \\ \left(-m_2 L_1 L_2 \left|\omega_1\right|^2 \hat{q}_1 + m_2 g L_2 \hat{e}_3\right) q_2\end{bmatrix}, \tag{5.5.1}$$

where

$$R(q) = \begin{bmatrix}\left(m_1+m_2\right) L_1^2 I_3 & m_2 L_1 L_2 \hat{q}_1^T \hat{q}_2 \\ m_2 L_1 L_2 \hat{q}_2^T \hat{q}_1 & m_2 L_2^2 I_3\end{bmatrix}.$$

As presented above, the matrix $R(q)$ defines a linear invertible map of the space $T_{q_1}S^2 \times T_{q_2}S^2$ onto itself:

$$A_{(q_1,q_2)} : T_{q_1}S^2 \times T_{q_2}S^2 \to T_{q_1}S^2 \times T_{q_2}S^2,\ \left[\omega_1,\omega_2\right]^T \to R(q)\left[\omega_1,\omega_2\right]^T.$$

We can easily see that it is well-defined since

$$R(q)\begin{bmatrix}\omega_1 \\ \omega_2\end{bmatrix} = \begin{bmatrix}\left(m_1+m_2\right) L_1^2 I_3 & m_2 L_1 L_2 \hat{q}_1^T \hat{q}_2 \\ m_2 L_1 L_2 \hat{q}_2^T \hat{q}_1 & m_2 L_2^2 I_3\end{bmatrix}\begin{bmatrix}\hat{v}_1 q_1 \\ \hat{v}_2 q_2\end{bmatrix} = \begin{bmatrix}\hat{r}_1 q_1 \\ \hat{r}_2 q_2\end{bmatrix} \in \left(TS^2\right)^2$$

with

$$r_1(q,\omega) := \left(m_1+m_2\right) L_1^2 v_1 + m_2 L_1 L_2 \hat{q}_2 \hat{v}_2 q_2,$$
$$r_2(q,\omega) := m_2 L_1 L_2 \hat{q}_1 \hat{v}_1 q_1 + m_2 L_2^2 v_2.$$

This map guarantees that if we rewrite the pair of equations for the angular velocities in (5.5.1) as

$$\dot{\omega} = R^{-1}(q) \begin{bmatrix} \left(-m_2 L_1 L_2 |\omega_2|^2 \hat{q}_2 + (m_1 + m_2) g L_1 \hat{e}_3\right) q_1 \\ \left(-m_2 L_1 L_2 |\omega_1|^2 \hat{q}_1 + m_2 g L_2 \hat{e}_3\right) q_2 \end{bmatrix} = R^{-1}(q)\, b =$$

$$= A^{-1}_{(q_1,q_2)}(b) = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} \in T_{q_1} S^2 \times T_{q_2} S^2,$$

then we are assured that there exists a pair of functions $a_1, a_2 : TS^2 \times TS^2 \to \mathbb{R}^3$ such that

$$\dot{\omega} = \begin{bmatrix} a_1(q,\omega) \times q_1 \\ a_2(q,\omega) \times q_2 \end{bmatrix} = \begin{bmatrix} h_1(q) \\ h_2(q) \end{bmatrix}.$$

Since we want $a_i \times q_i = h_i$, we just impose $a_i = q_i \times h_i$ and hence the whole vector field can be rewritten as

$$\begin{bmatrix} \dot{q}_1 \\ \dot{\omega}_1 \\ \dot{q}_2 \\ \dot{\omega}_2 \end{bmatrix} = \begin{bmatrix} \omega_1 \times q_1 \\ (q_1 \times h_1) \times q_1 \\ \omega_2 \times q_2 \\ (q_2 \times h_2) \times q_2 \end{bmatrix} = F|_{(q,\omega)},$$

with $h_i = h_i(q,\omega)$ and

$$\begin{bmatrix} h_1(q,\omega) \\ h_2(q,\omega) \end{bmatrix} = R^{-1}(q) \begin{bmatrix} \left(-m_2 L_1 L_2 |\omega_2|^2 \hat{q}_2 + (m_1 + m_2) g L_1 \hat{e}_3\right) q_1 \\ \left(-m_2 L_1 L_2 |\omega_1|^2 \hat{q}_1 + m_2 g L_2 \hat{e}_3\right) q_2 \end{bmatrix}.$$

Therefore, we can express the whole vector field in terms of the infinitesimal generator of the action of $SE(3) \times SE(3)$ as

$$\psi_* \left(f(q,\omega) \right) \Big|_{(q,\omega)} = F|_{(q,\omega)}$$

through the function

$$f : TS^2 \times TS^2 \to \mathfrak{se}(3) \times \mathfrak{se}(3) \simeq \mathbb{R}^{12}, \ (q,\omega) \to (\omega_1, q_1 \times h_1, \omega_2, q_2 \times h_2).$$

5.5.3 Numerical experiments

In this section, we present some numerical experiments for the $N-$chain of pendulums. We start by comparing the various Lie group integrators that we have tested (with the choice $N = 2$) and conclude by analyzing an implementation of variable step size. Lie group integrators allow us to keep the evolution

of the solution in the correct manifold, which is $TS^2 \times TS^2$ when $N = 2$. Hence, we briefly report two sets of numerical experiments. In the first one, we show the convergence rate of all the Lie group integrators tested on this model. In the second one, we check how they behave in terms of preserving the two following relations:

- $q_i(t)^T q_i(t) = 1$, i.e. $q_i(t) \in S^2$, $i = 1, 2$,
- $q_i(t)^T \omega_i(t) = 0$, i.e. $\omega_i(t) \in T_{q_i(t)} S^2$, $i = 1, 2$,

completing the analysis with a comparison with the classical Runge–Kutta 4 and with ODE45 of MATLAB. The Lie group integrators used to obtain the following experiments are Lie Euler, Lie Euler Heun, three versions of Runge–Kutta–Munthe–Kaas methods of order four and one of order three. The RKMK4 with two commutators mentioned in the plots, is the one presented in Section 5.2, while the other schemes can be found for example in [7].

Figure 5.5.2 presents the plots of the errors, in logarithmic scale, obtained considering as a reference solution the one given by the ODE45 method, with strict tolerance. Here, we used an exact expression for the $\mathrm{dexp}_\sigma^{-1}$ function. However, we could obtain the same results with a truncated version of this function, keeping a sufficiently high number of commutators, or after some clever manipulations of the commutators (as with RKMK4 with two commutators, see Section 5.2.2). The schemes show the right convergence rates, so we can move to the analysis of the time evolution on $TS^2 \times TS^2$.

In Figure 5.5.3, we can see the comparison of the time evolution of the $2-$norms of $q_1(t)$ and $q_2(t)$, for $0 \leq t \leq T = 5$. As highlighted above, unlike classical numerical integrators like the one implemented in ODE45 or the Runge–Kutta 4, the Lie group methods preserve the norm of the base components of the solutions, i.e., $|q_1(t)| = |q_2(t)| = 1 \ \forall t \in [0, T]$. Therefore, as expected, these integrators preserve the configuration manifold. However, to complete this analysis, we show the plots making a similar comparison but with the tangentiality conditions.

Indeed, in Figure 5.5.4, we compare the time evolution of the inner products $q_1(t)^T \omega_1(t)$ and $q_2(t)^T \omega_2(t)$ for $t \in [0, 5]$, i.e., we see if these integrators preserve the geometry of the whole phase space $TS^2 \times TS^2$. As we can see, while for Lie group methods these inner products are of the order of 10^{-14} and 10^{-15}, the ones obtained with classical integrators show that the tangentiality conditions are not preserved with the same accuracy.

We now move to some experiments on variable step size. In this last part, we focus on the RKMK pair coming from the Dormand–Prince method (DOPRI

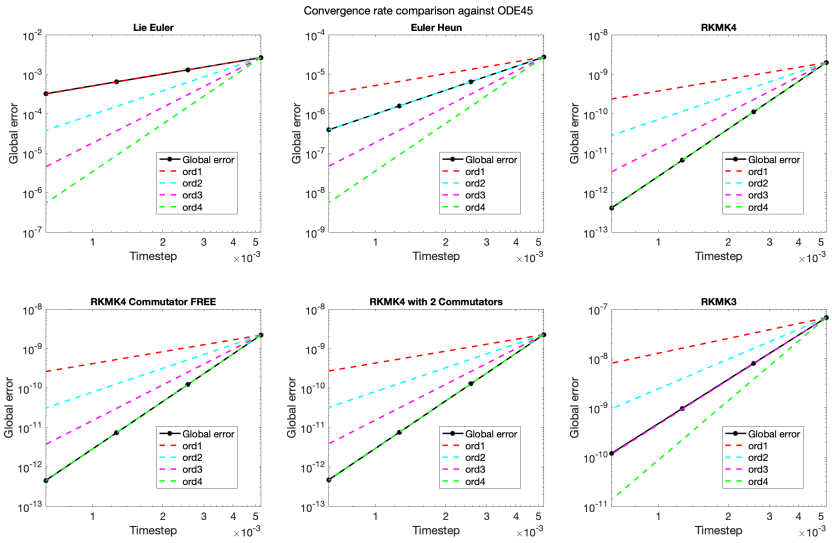


Figure 5.5.2: Convergence rate of the implemented Lie group integrators, based on global error considering as a reference solution the one of ODE45, with strict tolerance.

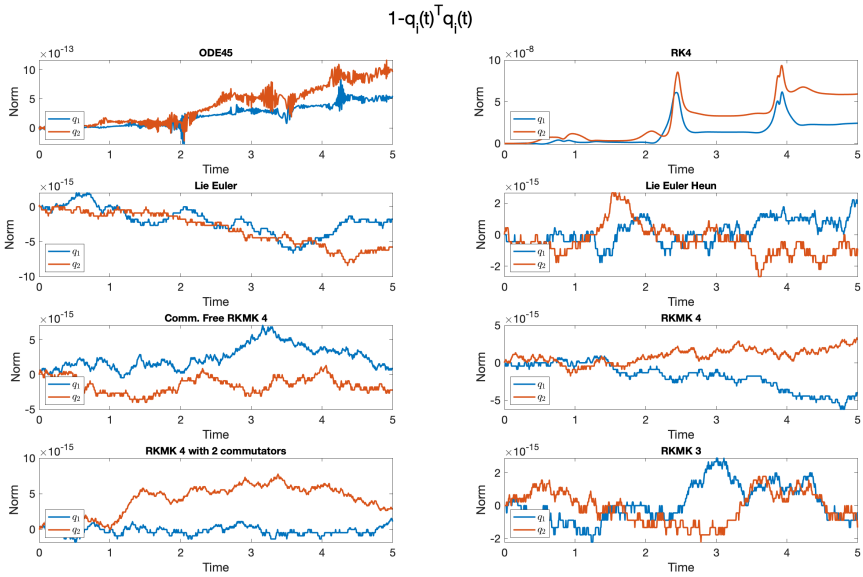


Figure 5.5.3: Visualization of the quantity $1 - q_i(t)^T q_i(t)$, $i = 1,2$, for time $t \in [0,5]$. These plots focus on the preservation of the geometry of S^2.

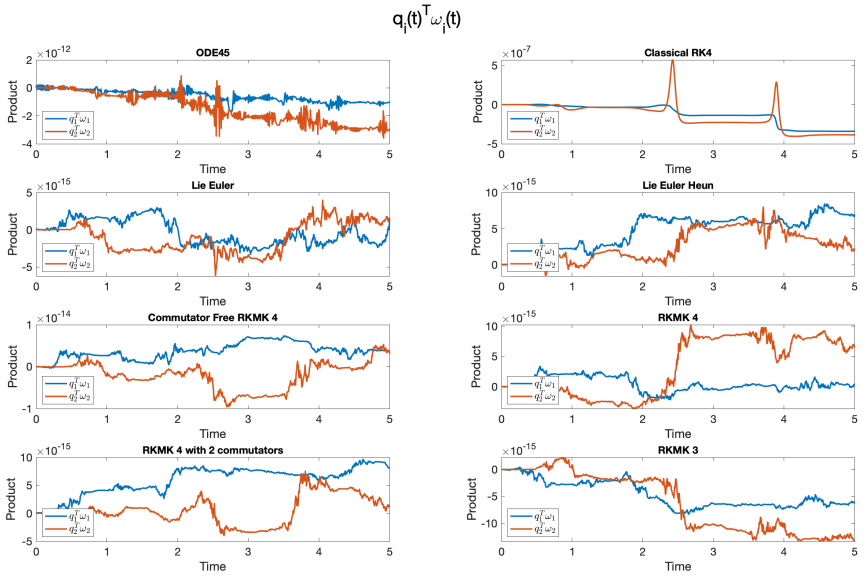


Figure 5.5.4: Visualization of the inner product $q_i(t)^T\omega_i(t)$, $i = 1,2$, for $t \in [0,5]$. These plots focus on the preservation of the geometry of $T_{q_i(t)}S^2$.

5(4) [14]), which we denote with RKMK(5,4). The aim of the plots we show is to compare the same schemes, both with constant and variable step size. We start by setting a tolerance and solving the system with the RKMK(5,4) scheme. Using the same number of time steps, we solve it again with RKMK of order 5. These experiments show that, for some tolerance and some initial conditions, the step size's adaptivity improves the numerical approximation accuracy. Since we do not have an available analytical solution to quantify these two schemes' accuracy, we compare them with the solution obtained with a strict tolerance and ODE45. We compute such accuracy, at time $T = 3$, by means of the Euclidean norm of the ambient space $\mathbb{R}^{6N}$.

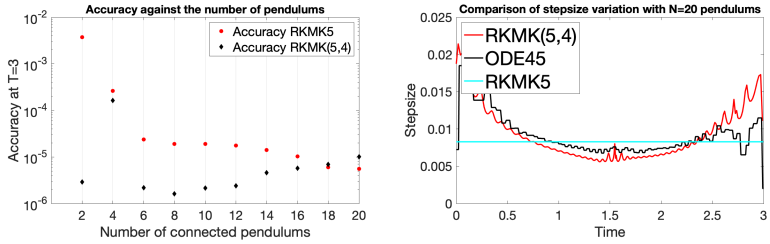
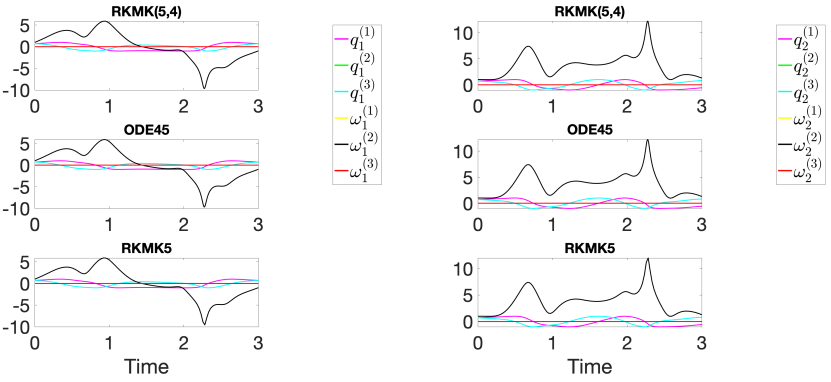


Figure 5.5.5: Comparison of accuracy at the final time (on the left) and step adaptation for the case $N = 20$ (on the right), with all pendulums of length $L_i = 1$.

In Figure 5.5.5, we compare the performance of the constant and variable step size methods, where the structure of the initial condition is always the same, but what changes is the number of connected pendulums. The considered initial condition is $(q_i, \omega_i) = \left(\sqrt{2}/2, 0, \sqrt{2}/2, 0, 1, 0\right), \quad \forall i = 1, ..., N$, and all the masses and lengths are set to 1. From these experiments, we can notice situations where the variable step size beats the constant one in terms of accuracy at the final time, like the case $N = 2$, which we discuss in more detail afterward.

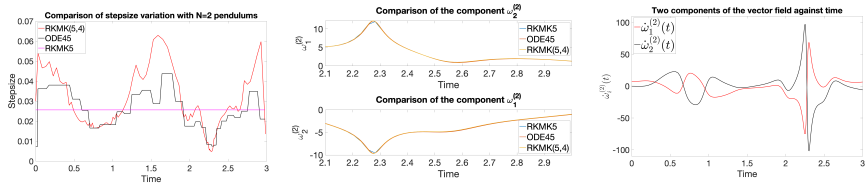
The results presented in Figure 5.5.5 (left) do not aim to highlight any particular relation between how the number of pendulums increases and the regularity of the solution. Indeed, as we add more pendulums, we keep incrementing the total length of the chain since $\sum_{i=1}^{N} L_i = N$. Thus, here, we do not have any appropriate limiting behavior in the solution as $N \to +\infty$. The behavior presented in that figure seems to highlight an improvement in accuracy for the RKMK5 method as N increases. However, this is biased by the fact that when we increase N to achieve the fixed tolerance of 10^{-6} with RKMK(5,4), we need more time steps in the discretization. Thus, this plot does not say that as N increases, the dynamics become more regular; it suggests that the number of required timesteps increases faster than the “degree of complexity" of the dynamics.
For the case $N = 2$, we notice a relevant improvement passing to variable step size. In Figures 5.5.6 and 5.5.7, we can see that, for this choice of the parameters, the solution behaves smoothly in most of the time interval, but then there is a peak in the second component of the angular velocities of both the masses, at $t \approx 2.2$. We can observe this behavior both in the plots of Figure 5.5.6, where we project the solution on the twelve components, and even in Figure 5.5.7c. In the latter, we plot two of the vector field components, i.e., the second components of the angular accelerations $\dot{\omega}_i(t)$, $i = 1, 2$. They show an abrupt change in the vector field in correspondence to $t \approx 2.2$, where the step is considerably restricted. Thus, to summarize, the gain we see with variable step size when $N = 2$ is motivated by the unbalance in the length of the time intervals with no abrupt changes in the dynamics and those where they appear. Indeed, we see that apart from a neighborhood of $t \approx 2.2$, the vector field does not change quickly. On the other hand, for the case $N = 20$, this is the case. Thus, the adaptivity of the step size does not bring relevant improvements in the latter situation.

The motivating application behind our choice of this mechanical system has been some intuitive relation with a beam model, as highlighted in the introduction of this work. However, for this limiting behavior to make sense, we should fix the length of the entire chain of pendulums to some L (the length of the beam at rest) and then set the size of each pendulum to $L_i = L/N$. In this



(a) $(q_1(t), \omega_1(t))$ **(b)** $(q_2(t), \omega_2(t))$

Figure 5.5.6: In these plots we represent the six components of the solution describing the dynamics of the first mass (on the left) and of the second mass (on the right), for the case $N = 2$. We compare the behavior of the solution obtained with constant step size RKMK5, the variable step size RKMK(5,4), and ODE45.



(a) Step adaptation **(b)** Zoom at final times **(c)** Values of $\dot{\omega}_i^{(2)}(t)$

Figure 5.5.7: On the left, we compare the adaptation of the step size of RKMK(5,4) with the one of ODE45 and with the constant step size of RKMK5. In the center, we plot the second component of the angular velocities $\omega_i^{(2)}$, $i = 1, 2$, and we zoom in on the last time interval $t \in [2.1, 3]$ to see that the variable step size version of the method better reproduces the reference solution. On the right, we visualize the speed of variation of the second component of the angular velocities.

case, keeping the same tolerance of 10^{-6} for RKMK(5,4), we get the results presented in Figure 5.5.8. We do not investigate more in details this approach, which might be relevant for further work, however we highlight that here the step adaptivity improves the results as we expected.

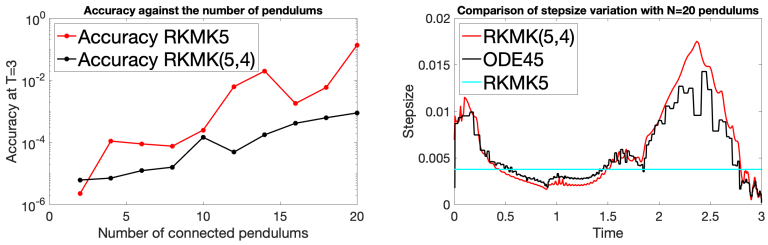


Figure 5.5.8: Comparison of accuracy at final time (on the left) and step adaptation for the case $N = 20$ (on the right), with all pendulums of length $L_i = 5/N$.

5.6 Dynamics of two quadrotors transporting a mass point

In this section, we consider a multibody system made of two cooperating quadrotor unmanned aerial vehicles (UAV) connected to a point mass (suspended load) via rigid links. This model is described in [28, 50].

We consider an inertial frame whose third axis goes in the direction of gravity but with opposite orientation, and we denote with $y \in \mathbb{R}^3$ the mass point and with $y_1, y_2 \in \mathbb{R}^3$ the two quadrotors. We assume that the links between the two quadrotors and the mass point are of a fixed length $L_1, L_2 \in \mathbb{R}^+$. The configuration variables of the system are: the position of the mass point in the inertial frame, $y \in \mathbb{R}^3$, the attitude matrices of the two quadrotors, $(R_1, R_2) \in (SO(3))^2$ and the directions of the links which connect the center of mass of each quadrotor respectively with the mass point, $(q_1, q_2) \in (S^2)^2$. The configuration manifold of the system is $Q = \mathbb{R}^3 \times (SO(3))^2 \times (S^2)^2$. In order to present the equations of motion of the system, we start by identifying $TSO(3) \simeq SO(3) \times \mathfrak{so}(3)$ via left-trivialization. This choice allows us to write the kinematic equations of the system as

$$\dot{R}_i = R_i \hat{\Omega}_i, \quad \dot{q}_i = \hat{\omega}_i q_i \qquad i = 1, 2, \tag{5.6.1}$$

where $\Omega_1, \Omega_2 \in \mathbb{R}^3$ represent the angular velocities of each quadrotor, respectively, and ω_1, ω_2 express the time derivatives of the orientations $q_1, q_2 \in S^2$, respectively, in terms of angular velocities, expressed with respect to the body-

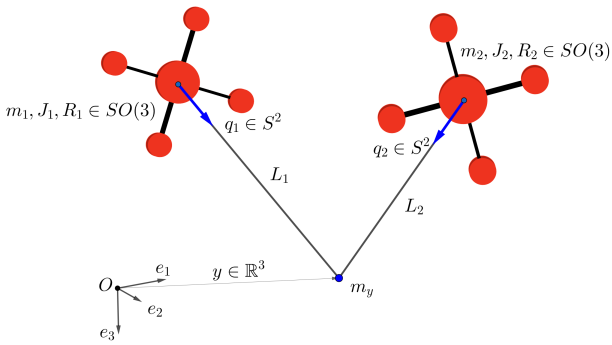


Figure 5.6.1: Two quadrotors connected to the mass point m_y via massless links of lengths L_i.

fixed frames. From these equations, we define the trivialized Lagrangian

$$L\left(y, \dot{y}, R_1, \Omega_1, R_2, \Omega_2, q_1, \omega_1, q_2, \omega_2\right) : \mathbb{R}^6 \times \left(SO(3) \times \mathfrak{so}(3)\right)^2 \times \left(TS^2\right)^2 \to \mathbb{R},$$

as the difference of the total kinetic energy of the system and the total potential (gravitational) energy, $L = T - U$, with:

$$T = \frac{1}{2} m_y \left\| \dot{y} \right\|^2 + \frac{1}{2} \sum_{i=1}^{2} \left(m_i \left\| \dot{y} - L_i \hat{\omega}_i q_i \right\|^2 + \Omega_i^T J_i \Omega_i \right),$$

and

$$U = -m_y g e_3^T y - \sum_{i=1}^{2} m_i g e_3^T \left(y - L_i q_i \right),$$

where $J_1, J_2 \in \mathbb{R}^{3\times 3}$ are the inertia matrices of the two quadrotors and $m_1, m_2 \in \mathbb{R}^+$ are their respective total masses. In this system, each of the two quadrotors generates a thrust force, which we denote with $u_i = -T_i R_i e_3 \in \mathbb{R}^3$, where T_i is the magnitude, while e_3 is the direction of this vector in the $i-$th body-fixed frame, $i = 1, 2$. The presence of these forces makes it a nonconservative system. Moreover, the rotors of the two quadrotors generate a moment vector, and we denote with $M_1, M_2 \in \mathbb{R}^3$ the cumulative moment vector of each of the two quadrotors. To derive the Euler–Lagrange equations, a possible approach is through Lagrange–d'Alambert's principle, as presented in [28]. We write them in matrix form as

$$A(z)\dot{z} = h(z) \tag{5.6.2}$$

where

$$z = \left[y, v, \Omega_1, \Omega_2, \omega_1, \omega_2 \right]^T \in \mathbb{R}^{18},$$

$$A(z) = \begin{bmatrix} I_3 & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & M_q & 0_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & J_1 & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & J_2 & 0_3 & 0_3 \\ 0_3 & -\frac{1}{L_1}\hat{q}_1 & 0_3 & 0_3 & I_3 & 0_3 \\ 0_3 & -\frac{1}{L_2}\hat{q}_2 & 0_3 & 0_3 & 0_3 & I_3 \end{bmatrix},$$

$$h(z) = \begin{bmatrix} h_1(z) \\ h_2(z) \\ h_3(z) \\ h_4(z) \\ h_5(z) \\ h_6(z) \end{bmatrix} = \begin{bmatrix} v \\ -\sum_{i=1}^{2} m_i L_i \|\omega_i\|^2 q_i + M_q g e_3 + \sum_{i=1}^{2} u_i^{\parallel} \\ -\Omega_1 \times J_1\Omega_1 + M_1 \\ -\Omega_2 \times J_2\Omega_2 + M_2 \\ -\frac{1}{L_1} g \hat{q}_1 e_3 - \frac{1}{m_1 L_1} q_1 \times u_1^{\perp} \\ -\frac{1}{L_2} g \hat{q}_2 e_3 - \frac{1}{m_2 L_2} q_2 \times u_2^{\perp} \end{bmatrix},$$

where $M_q = m_y I_3 + \sum_{i=1}^{2} m_i q_i q_i^T$, and $u_i^{\parallel}, u_i^{\perp}$ are respectively the orthogonal projection of u_i along q_i and to the plane $T_{q_i}S^2$, $i = 1,2$, i.e. $u_i^{\parallel} = q_i q_i^T u_i$, $u_i^{\perp} = (I - q_i q_i^T) u_i$. These equations, coupled with the kinematic equations in (5.6.1), describe the dynamics of a point

$$P = \left[y,\ v,\ R_1,\ \Omega_1,\ R_2,\ \Omega_2,\ q_1,\ \omega_1,\ q_2,\ \omega_2 \right] \in M = TQ.$$

Since the matrix $A(z)$ is invertible, we pass to the following set of equations

$$\dot{z} = A^{-1}(z)h(z) := \tilde{h}(z) := \bar{h}(P) = [\bar{h}_1(P), ..., \bar{h}_6(P)]^T. \tag{5.6.3}$$

5.6.1 Analysis via transitive group actions

We identify the phase space M with $M \simeq T\mathbb{R}^3 \times \left(TSO(3)\right)^2 \times \left(TS^2\right)^2$. The group we consider is

$$\bar{G} = \mathbb{R}^6 \times \left(TSO(3)\right)^2 \times \left(SE(3)\right)^2,$$

where the groups are combined with a direct-product structure and $\mathbb{R}^6$ is the additive group. For a group element

$$g = \Big((a_1, a_2), \big((B_1, b_1), (B_2, b_2) \big), \big((C_1, c_1), (C_2, c_2) \big) \Big) \in \bar{G}$$

and a point $P \in M$ in the manifold, we consider the following left action

$$\psi_g(P) = \Big[y + a_1,\ v + a_2,\ B_1 R_1,\ \Omega_1 + b_1,\ B_2 R_2,\ \Omega_2 + b_2,$$
$$C_1 q_1,\ C_1\omega_1 + c_1 \times C_1 q_1,\ C_2 q_2,\ C_2\omega_2 + c_2 \times C_2 q_2 \Big].$$

The well-definiteness and transitivity of this action come from standard arguments, see for example [42]. The infinitesimal generator associated to

$$\xi = \left[\xi_1,\ \xi_2,\ \eta_1,\ \eta_2,\ \eta_3,\ \eta_4,\ \mu_1,\ \mu_2,\ \mu_3,\ \mu_4\right] \in \bar{\mathfrak{g}},$$

where $\bar{\mathfrak{g}} = T_e\bar{G}$, writes

$$\psi_*(\xi)\Big|_P = \Big[\xi_1,\ \xi_2,\ \hat{\eta}_1 R_1,\ \eta_2,\ \hat{\eta}_3 R_2,\ \eta_4,\ \hat{\mu}_1 q_1,\ \hat{\mu}_1\omega_1 + \hat{\mu}_2 q_1,\ \hat{\mu}_3 q_2,\ \hat{\mu}_3\omega_2 + \hat{\mu}_4 q_2\Big].$$

We can now focus on the construction of the function $f : M \to \bar{\mathfrak{g}}$ such that $\psi_*(f(P))|_P = F|_P$, where

$$F|_P = \Big[\bar{h}_1(P),\ \bar{h}_2(P),\ R_1\hat{\Omega}_1,\ \bar{h}_3(P),\ R_2\hat{\Omega}_2,\ \bar{h}_4(P),\ \hat{\omega}_1 q_1,\ \bar{h}_5(P),\ \hat{\omega}_2 q_2,\ \bar{h}_6(P)\Big] \in T_P M$$

is the vector field obtained combining the equations (5.6.1) and (5.6.3). We have

$$f(P) = \Big[\bar{h}_1(P),\ \bar{h}_2(P),\ R_1\Omega_1,\ \bar{h}_3(P),\ R_2\Omega_2,\ \bar{h}_4(P),\ \omega_1,\ q_1 \times \bar{h}_5(P),\ \omega_2,\ q_2 \times \bar{h}_6(P)\Big] \in \bar{\mathfrak{g}}.$$

We have obtained the local representation of the vector field $F \in \mathfrak{X}(M)$ in terms of the infinitesimal generator of the transitive group action ψ, hence we can solve for one time step Δt the IVP

$$\begin{cases} \dot{\sigma}(t) = \operatorname{dexp}^{-1}_{\sigma(t)}\left(f\left(\psi\left(\exp\left(\sigma(t)\right), P(t)\right)\right)\right) \\ \sigma(0) = 0 \in \bar{\mathfrak{g}} \end{cases}$$

and then update the solution $P(t+\Delta t) = \psi(\exp(\sigma(\Delta t)), P(t))$.

The above construction is completely independent of the control functions $\{u_i^{\parallel}, u_i^{\perp}, M_i\}_{i=1,2}$, and hence it is compatible with any choice of these parameters.

5.6.2 Numerical experiments

We tested Lie group numerical integrators for a load transportation problem presented in [50]. The control inputs $\{u_i^{\parallel}, u_i^{\perp}, M_i\}_{i=1,2}$ are constructed such

that the point mass asymptotically follows a given desired trajectory $y_d \in \mathbb{R}^3$, given by a smooth function of time, and the quadrotors maintain a prescribed formation relative to the point mass. In particular, the parallel components $u_i^{\parallel}$ are designed such that the payload follows the desired trajectory y_d (load transportation problem), while the normal components $u_i^{\perp}$ are designed such that q_i converge to desired directions q_{id} (tracking problem in S_2). Finally, M_i are designed to control the attitude of the quadrotors.

In this experiment, we focus on a simplified dynamics model, i.e., we neglect the construction of the controllers M_i for the attitude dynamics of the quadrotors. However, the full dynamics model can also be easily integrated once the expressions for the attitude controllers are available.

In Figure 5.6.2, we show the convergence rate of four different RKMK methods compared with the reference solution obtained with ODE45 in MATLAB.

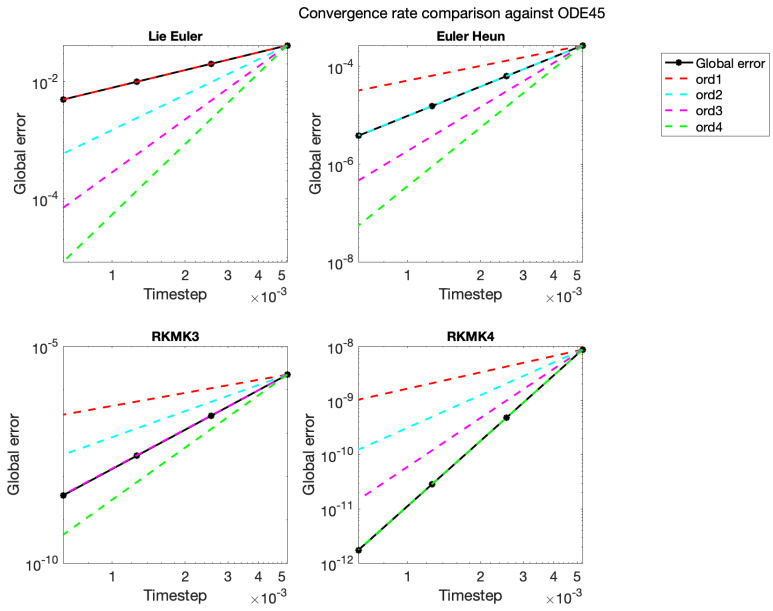


Figure 5.6.2: Convergence rate of the numerical schemes compared with ODE45

In Figures 5.6.3-5.6.7, we show results in the tracking of a parabolic trajectory obtained by integrating the system in (5.6.2) with a RKMK method of order 4.

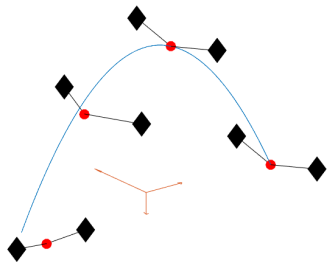


Figure 5.6.3: Snapshots at $0 \leq t \leq 5$.

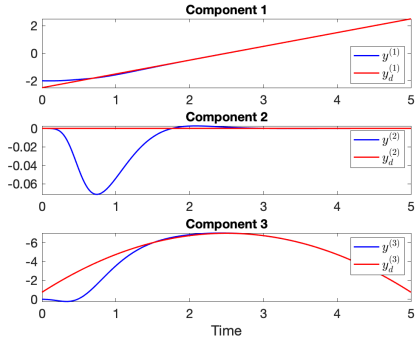


Figure 5.6.4: Components of the load position (in blue) and the desired trajectory (in red) as a function time.

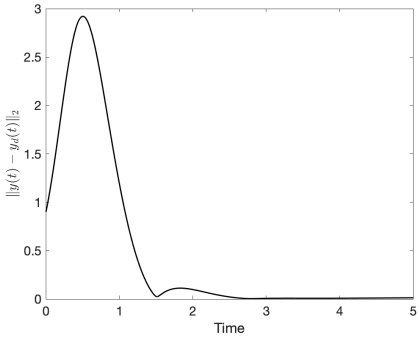


Figure 5.6.5: Deviation of the load position from the target trajectory.

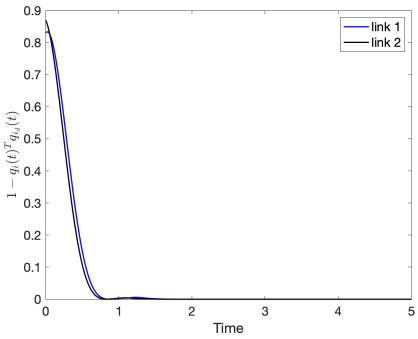


Figure 5.6.6: Direction error of the links.

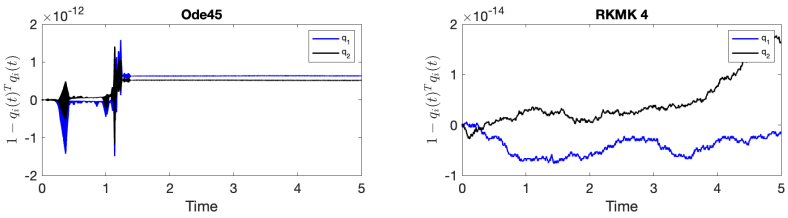


Figure 5.6.7: Preservation of the norms of $q_1, q_2 \in S^2$.

5.7 Summary and outlook

In this paper, we have considered Lie group integrators with a particular focus on problems from mechanics. In mathematical terms, this means that the Lie groups and manifolds of particular interest are $SO(n),\ n = 2,3$, $SE(n),\ n = 2,3$ as well as the manifolds S^2 and TS^2. The abstract formulations by, e.g., Crouch and Grossman [11], Munthe-Kaas [40], and Celledoni et al. [6] have often been demonstrated on small toy problems in the literature, such as the free rigid body or the heavy top systems. But in papers like [4], hybrid versions of Lie group integrators have been applied to more complex beam and multibody problems. The present paper is attempting to move in the direction of more relevant examples without causing the numerical solution to depend on how the manifold is embedded in an ambient space or the choice of local coordinates.

It will be the subject of future work to explore more examples and to aim for a more systematic approach to applying Lie group integrators to mechanical problems. In particular, it is of interest to the authors to consider models of beams that could be seen as a generalization of the N-fold pendulum discussed here.

Funding

This work was supported by Marie Skłodowska-Curie [860124].

Bibliography

[1] M. Arnold and O. Brüls. Convergence of the generalized-α scheme for constrained mechanical systems. *Multibody Syst. Dyn.*, 18(2):185–202, 2007. 183

[2] M. Arnold, O. Brüls, and A. Cardona. Error analysis of generalized-α Lie group time integration methods for constrained mechanical systems. *Numer. Math.*, 129(1):149–179, 2015. 183

[3] G. Bogfjellmo and H. Marthinsen. High-Order Symplectic Partitioned Lie Group Methods. *Foundations of Computational Mathematics*, pages 1–38, 2015. 193

[4] O. Bruls and A. Cardona. On the Use of Lie Group Time Integrators in Multibody Dynamics. *J. Computational Nonlinear Dynamics*, 5(3):031002, 2010. 183, 196, 219

[5] F. Casas and B. Owren. Cost Efficient Lie Group Integrators in the RKMK Class. *BIT Numerical Mathematics*, 43(4):723–742, 2003. 188, 189

[6] E. Celledoni, A. Marthinsen, and B. Owren. Commutator-free Lie group methods. *Future Generation Computer Systems*, 19:341–352, 2003. 183, 184, 189, 190, 219

[7] E. Celledoni, H. Marthinsen, and B. Owren. An introduction to Lie group integrators—basics, new developments and applications. *J. Comput. Phys.*, 257(part B):1040–1061, 2014. 183, 208

[8] E. Celledoni and B. Owren. Lie group methods for rigid body dynamics and time integration on manifolds. *Comput. Methods Appl. Mech. Engrg.*, 192(3-4):421–438, 2003. 189

[9] J. Ćesić, V. Jukov, I. Petrovic, and D. Kulić. Full body human motion estimation on lie groups using 3D marker position measure-

ments. *In Proceedings of the IEEE-RAS International Conference on HumanoidRobotics, Cancun, Mexico, 15–17 November 2016*, 2016. 184

[10] S.H. Christiansen, H.Z. Munthe-Kaas, and B. Owren. Topics in structure-preserving discretization. *Acta Numerica*, 20:1–119, 2011. 183

[11] P. E. Crouch and R. Grossman. Numerical Integration of Ordinary Differential Equations on Manifolds. *J. Nonlinear Sci.*, 3:1–33, 1993. 183, 184, 185, 189, 219

[12] C. Curry and B. Owren. Variable step size commutator free Lie group integrators. *Numer. Algorithms*, 82(4):1359–1376, 2019. 195, 199, 200

[13] F. Diele, L. Lopez, and R. Peluso. The Cayley transform in the numerical solution of unitary differential systems. *Adv. Comput. Math.*, 8(4):317–334, 1998. 183, 191

[14] J. R Dormand and P. J Prince. A family of embedded Runge-Kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980. 210

[15] K. Engø and S. Faltinsen. Numerical integration of Lie–Poisson systems while preserving coadjoint orbits and energy. *SIAM J. Numer. Anal.*, 39(1):128–145, 2001. 193, 195

[16] H. Goldstein, C.P. Poole, and J. Safko. *Classical Mechanics*. Pearson, 2013. 205

[17] V. Guillemin and S. Sternberg. The moment map and collective motion. *Ann. Physics*, 127:220–253, 1980. 195

[18] E. Hairer, Ch. Lubich, and G. Wanner. *Geometric Numerical Integration*, volume 31 of *Springer Series in Computational Mathematics*. Springer, Heidelberg, 2010. Structure-Preserving Algorithms for Ordinary Differential Equations, Reprint of the second (2006) edition. 183

[19] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I, Nonstiff Problems*. Springer-Verlag, Second revised edition, 1993. 198

[20] J. Hall and M. Leok. Lie Group Spectral Variational Integrators. *Found. Comput. Math.*, 17(1):199–257, 2017. 183

[21] F. Hausdorff. Die symbolische Exponential Formel in der Gruppen theorie. *Leipziger Ber.*, 58:19–48, 1906. 188

[22] H. M. Hilber, T. J. R. Hughes, and R. L. Taylor. Improved numerical dissipation for time integration algorithms in structural dynamics. *Earthquake Engineering & Structural Dynamics*, 5(3):283–292, 1977. cited By 1683. 183

[23] D. Holm. *Geometric Mechanics: Part II: Rotating, Translating and Rolling*. World Scientific Publishing Company, 2008. 196, 202

[24] D. Holm, J. Marsden, and T. Ratiu. The Euler–Poincaré Equations and Semidirect Products with Applications to Continuum Theories. *Adv. in Math.*, 137:1–81, 1998. 193

[25] S. Holzinger and J. Gerstmayr. Time integration of rigid bodies modelled with three rotation parameters. *Multibody Sys Dyn*, 2021. 184

[26] A. Iserles, H. Z. Munthe-Kaas, S. P. Nørsett, and A. Zanna. Lie-group methods. *Acta Numerica*, 9:215–365, 2000. 183

[27] T. Lee, M. Leok, and N. H. McClamroch. Lie group variational integrators for the full body problem. *Comput. Methods Appl. Mech. Engrg.*, 196(29-30):2907–2924, 2007. 183, 184

[28] T. Lee, M. Leok, and N. H. McClamroch. *Global Formulations of Lagrangian and Hamiltonian Dynamics on Manifolds*. Interaction of Mechanics and Mathematics. Springer, Cham, 2018. A geometric approach to modeling and analysis. 183, 201, 203, 205, 213, 214

[29] T. Leitz and S. Leyendecker. Galerkin Lie-group variational integrators based on unit quaternion interpolation. *Comput. Methods Appl. Mech. Engrg.*, 338:333–361, 2018. 183

[30] N. E. Leonard and J. E. Marsden. Stability and drift of underwater vehicle dynamics: Mechanical systems with rigid motion symmetry. *Physica D*, 105(1–3):130–162, 1997. 196

[31] D. Lewis and J. C. Simo. Conserving algorithms for the dynamics of Hamiltonian systems of Lie groups. *J. Nonlinear Sci.*, 4:253–299, 1994. 183, 192, 195

[32] A. Lundervold and H. Z. Munthe-Kaas. On algebraic structures of numerical integration on vector spaces and manifolds. In *Faà di Bruno Hopf algebras, Dyson-Schwinger equations, and Lie-Butcher series*, volume 21 of *IRMA Lect. Math. Theor. Phys.*, pages 219–263. Eur. Math. Soc., Zürich, 2015. 186

[33] J. E. Marsden, T. Ratiu, and A. Weinstein. Semi-direct products and reduction in mechanics. *Transactions of the American Mathematical Society*, 281:147–77, 1884. 193

[34] J. E. Marsden and T. S. Ratiu. *Introduction to Mechanics and Symmetry*. Springer-Verlag, 1994. 186

[35] J. E. Marsden and T. S. Ratiu. *Introduction to Mechanics and Symmetry*. Number 17 in Texts in Applied Mathematics. Springer-Verlag, second edition, 1999. 193

[36] R. H. Merson. An operational method for the study of integration processes. In *Proc. Symp. Data Processing*, 1957. 198

[37] J. Moser and A. P. Veselov. Discrete versions of some classical integrable systems and factorization of matrix polynomials. *Comm. Math. Phys.*, 139(2):217–243, 1991. 183

[38] A. Müller. Coordinate mappings for Rigid Body Motions. *ASME Journal of Computational and Nonlinear Dynamics*, 12, 2017. 191

[39] H. Munthe-Kaas. Runge–Kutta methods on Lie groups. *BIT*, 38(1):92–111, 1998. 184

[40] H. Munthe-Kaas. High order Runge–Kutta methods on manifolds. *Appl. Numer. Math.*, 29:115–127, 1999. 183, 184, 185, 188, 219, 239

[41] H. Munthe-Kaas and B. Owren. Computations in a free Lie algebra. *Phil. Trans. Royal Soc. A*, 357:957–981, 1999. 188

[42] P. J. Olver. *Applications of Lie Groups to Differential Equations*, volume 107. Springer Science & Business Media, 2000. 202, 216

[43] B. Owren. Order conditions for commutator-free Lie group methods. *J. Phys. A*, 39(19):5585–5599, 2006. 199

[44] B. Owren. Lie group integrators. In *Discrete mechanics, geometric integration and Lie-Butcher series*, volume 267 of *Springer Proc. Math. Stat.*, pages 29–69. Springer, Cham, 2018. 183

[45] B. Owren and A. Marthinsen. Integration methods based on canonical coordinates of the second kind. *Numer. Math.*, 87(4):763–790, 2001. 183, 191

[46] J. Park and W. Chung. Geometric integration on Euclidean group with application to articulated multibody systems. *J. CAM*, 21, 2005. 184

[47] T. Ratiu. Euler-Poisson equations on Lie algebras and the N-dimensional heavy rigid body. *Proc. Nat. Acad. Sci. USA*, pages 1327–1328, 1981. 195

[48] A. Saccon. Midpoint rule for variational integrators on Lie groups. *Internat. J. Numer. Methods Engrg.*, 78(11):1345–1364, 2009. 195

[49] J. C. Simo and L. Vu-Quoc. On the dynamics of finite-strain rods undergoing large motions — a geometrically exact approach. *Comput. Methods Appl. Mech. Engrg.*, 66:125–161, 1988. 183, 184

[50] Lee T., K. Sreenath, and V. Kumar. Geometric control of cooperating multiple quadrotor UAVs with a suspended payload. *52nd IEEE Conference on Decision and Control*, pages 5510–5515, 2013. 213, 216

[51] A. P. Veselov. Integrable systems with discrete time, and difference operators. *Funktsional. Anal. i Prilozhen.*, 22(2):1–13, 96, 1988. 183

[52] A. M. Vinogradov and B. Kupershmidt. The structure of Hamiltonian mechanics. *Funktsional. Anal. i Prilozhen.*, 32:177–243, 1977. 195

[53] F. W. Warner. *Foundations of Differentiable Manifolds and Lie Groups*. GTM 94. Springer-Verlag, 1983. 185

[54] V. Wieloch and M. Arnold. BDF integrators for constrained mechanical systems on Lie groups. *J. CAM*, 2019. In press. 183

Learning Hamiltonians of constrained mechanical systems

Elena Celledoni, Andrea Leone, Davide Murari, Brynjulf Owren

Journal of Computational and Applied Mathematics

Abstract. Recently, there has been an increasing interest in the modeling and computation of physical systems with neural networks. Hamiltonian systems are an elegant and compact formalism in classical mechanics, where the dynamics is fully determined by one scalar function, the Hamiltonian. The solution trajectories are often constrained to evolve on a submanifold of a linear vector space. In this work, we propose new approaches for the accurate approximation of the Hamiltonian function of constrained mechanical systems given sample data information of their solutions. We focus on the importance of the preservation of the constraints in the learning strategy by using both explicit Lie group integrators and other classical schemes.

6.1 Introduction

Neural networks have been proven to be effective in learning patterns from data in many different contexts. Recently there has been an increasing interest in applying neural networks to learn physical models from data, for example models of classical mechanics. For Hamiltonian systems, multiple approaches have been proposed to approximate the energy function, see, e.g., [9, 14, 26, 13, 23]. Building on these results, we propose an improved learning procedure. Our main contribution is an approach to learning the Hamiltonian for systems defined on the cotangent bundle $T^*\mathcal{Q}$ of some manifold $\mathcal{Q}$ embedded in a vector space. Under the assumption that $T^*\mathcal{Q}$ is homogeneous, we show how to do that while preserving the geometry during the learning phase. In this paper, by preservation of the geometry, we mean the accurate conservation of the constraints rather than of other geometric features such as symplecticity, energy, or other first integrals of the system.

As in [13], we express the dynamics of constrained systems by embedding the problem in a vector space of larger dimension, but in our approach we do not make use of Lagrange multipliers. With the aim of understanding the importance of the geometry in this approximation problem, we compare learning procedures based on numerical integrators that preserve the phase space of the system with others that do not. We restrict to homogeneous spaces where Lie group methods can preserve the geometry up to machine accuracy (see, e.g., [8]). For example, multi-body lumped mass systems fall naturally in this setting [20, Chapter 2]. This restriction still includes systems with the configuration manifold that is a Lie group, as in some problems of rigid body and rod dynamics, but we will not consider these applications here. The experiments show that there are specific problems where approximating the Hamiltonian using a Lie group method can be relevant. Surprisingly, in many other settings classical Runge–Kutta integrators produce comparable results.

The main focus of the present paper is to learn an approximation of a Hamiltonian system where the training data are given as a set of trajectory segments.

To do so, one could learn the dynamics either by approximating the Hamiltonian vector field or the Hamiltonian function as done in our work. Another relevant difference in the learning framework consists of considering in the training procedure either one time step of the flow map (see, e.g., [14]) or a sequence of successive time steps as proposed in [9]. The latter work shows experimental evidence that taking into account temporal dependencies improves performance. We follow the second strategy when dealing with unconstrained systems, whereas we test both of them with our approach to constrained systems.

In principle, the Hamiltonian can be any differentiable function. However, for mechanical systems, it is often made by the sum of a (quadratic) kinetic energy and a potential energy, [25, 15, 21]. Following [26], we make the ansatz that the kinetic energy is characterized by a symmetric and positive definite matrix, and hence we aim to estimate it.

We conclude this Section with a more precise definition of the problem of interest. In the second Section, we introduce the Hamiltonian formalism for both unconstrained and constrained systems. In the third Section, we focus on unconstrained systems, presenting the general learning procedure that will be extended to constrained systems in the fourth Section. We also discuss how additional known information about the dynamical system can be included in the network training procedure. The experimental results show that physics-based regularization could be helpful to improve the extrapolation capability of the network and its stability in the presence of noise. In the last Section, we formalize the problem of learning a constrained Hamiltonian mechanical system and discuss the importance of the geometry for this class of problems. Finally, we complete this Section with numerical experiments in the `PyTorch` framework, showing how the predicted Hamiltonian depends on some training parameters and on the presence of noise. The numerical implementations are available in the GitHub repository associated to the paper[1].

6.1.1 Description of the problem

Suppose to be given a set of N sampled trajectories coming from a Hamiltonian system defined on a submanifold $\mathcal{M} = T^*\mathcal{Q}$ of $\mathbb{R}^{2n}$, where $T^*\mathcal{Q}$ is the cotangent bundle of the configuration manifold $\mathcal{Q}$ (see [19, Chapter 11] for more details). Moreover, assume that each of these trajectories contains M

[1] https://github.com/davidemurari/learningConstrainedHamiltonians

equispaced (in time) points. In other words, suppose that

$$\left\{\left(x_i, \bar{y}_i^2, ..., \bar{y}_i^M\right)\right\}_{i=1,...,N}, \ \bar{y}_i^j = \Phi_{X_H}^{(j-1)\Delta t}\left(x_i\right) \tag{6.1.1}$$

as a training set, where $\Phi_{X_H}^t$ is the time t-flow of the exact, unknown Hamiltonian system. In practice, we never have access to the exact trajectories but to either a noisy version of them or a numerical approximation.

The approach we use aims to approximate the vector field $X_H \in \mathfrak{X}(\mathcal{M})$ that governs the dynamics, where by $\mathfrak{X}(\mathcal{M})$ we denote the collection of all smooth vector fields on $\mathcal{M}$. However, we know that such a vector field is Hamiltonian, i.e., there exists a scalar function $H : \mathcal{M} \to \mathbb{R}$ which, together with the geometry given by $\mathcal{M}$, characterizes the dynamics completely. For this reason, we do not need to directly approximate X_H, but just H and then eventually recover X_H.

The problem under consideration can be described as an inverse problem since we want to infer the function H from trajectory data of the corresponding dynamical system rather than from samples of the function H itself. This description of the problem motivates how we measure the accuracy of our approximation, denoted by a parametric model H_Θ. Indeed, the target is not to approximate the trajectories of the given Hamiltonian system with some neural network but to approximate the Hamiltonian. Thus, the quality of the approximation can be computed in at least two ways. First, one can compare some measured trajectories with those obtained from the approximation. More precisely, we randomly generate $\tilde{N}$ initial conditions $z_i \in \mathcal{M}$, their $\tilde{M}$ time updates, and compute

$$\mathcal{E}_1\left(\left\{u_i^j\right\}_{i=1,...,\tilde{N}}^{j=1,...,\tilde{M}}, \left\{v_i^j\right\}_{i=1,...,\tilde{N}}^{j=1,...,\tilde{M}}\right) = \frac{1}{\tilde{N}\tilde{M}} \sum_{j=1}^{\tilde{M}} \sum_{i=1}^{\tilde{N}} \left\| u_i^j - v_i^j \right\|^2, \tag{6.1.2}$$

where $\|\cdot\|$ is the Euclidean norm of $\mathbb{R}^{2n}$, $u_i^1 = z_i$, $v_i^1 = z_i$, $u_i^{j+1} = \Psi_{X_H}^h(u_i^j)$ and $v_i^{j+1} = \Psi_{X_{H_\Theta}}^h(v_i^j)$ for a numerical integrator Ψ^h of choice. One can randomly generate these initial conditions for academic examples where the true Hamiltonian is actually known. In this case, $\tilde{N}$ and $\tilde{M}$ can be specified arbitrarily, usually with $\tilde{N}$ less than the number of training trajectories N. On the other hand, in more realistic applications, one has to work with the initial conditions for which the related trajectory segments are known. In this case $\tilde{N}$ and $\tilde{M}$ are constrained by the available data, in particular the number of total trajectories is split into N for training and $\tilde{N}$ for test. In our experiments, we adopted the `SciPy` implementation of the Dormand-Prince pair of order (5,4) with a strict tolerance. In fact, following the PyTorch implementation of the mean squared error, $\mathcal{E}_1$ is actually divided by $2n$. Alternatively, as introduced in [11], one

can compare pointwise values of the approximated and the true Hamiltonian, when known. This gives

$$\mathcal{E}_2(H, H_\Theta) = \frac{1}{\tilde{N}} \sum_{i=1}^{\tilde{N}} \left| H\left(z_i\right) - H_\Theta\left(z_i\right) - \frac{1}{\tilde{N}} \sum_{l=1}^{\tilde{N}} \left(H\left(z_l\right) - H_\Theta\left(z_l\right) \right) \right|, \qquad (6.1.3)$$

where $\mathcal{E}_2$ handles the fact that Hamiltonians differing, on $\mathcal{M}$, by a constant generate the same vector field. Indeed, $\mathcal{E}_2(H, H + c) = 0$.

6.2 Hamiltonian mechanical systems

In this work, we focus on Hamiltonian mechanical systems based on a configuration manifold $\mathcal{Q} \subseteq \mathbb{R}^n$. We now introduce some basic elements of the theory of unconstrained Hamiltonian dynamics on $\mathbb{R}^{2n}$, which corresponds to the case $\mathcal{Q} = \mathbb{R}^n$. Then, we extend this formulation to constrained systems on $T^*\mathcal{Q} \subset \mathbb{R}^{2n}$.

The Hamiltonian formalism gives a particular class of conservative vector fields which, in contrast to the Lagrangian one, can always be expressed with a system of first-order ordinary differential equations. For the unconstrained case, the equations are of the form $\dot{x}(t) = \mathbb{J}\nabla H(x(t)) := X_H(x(t))$ where $x(t) = [q(t), p(t)] \in \mathbb{R}^{2n}$ comprises the configuration variables and their conjugate momenta. Here, $H : \mathbb{R}^{2n} \to \mathbb{R}$ is a smooth function called the Hamiltonian of the system, and $\mathbb{J} \in \mathbb{R}^{2n \times 2n}$ is the symplectic matrix.

In this work, we focus on Hamiltonian systems whose energy function is of the form

$$H\left(q, p\right) = \frac{1}{2} p^T M^{-1}\left(q\right) p + V\left(q\right)$$

where $M(q)$ is the mass matrix of the system, possibly depending on the configuration $q \in \mathbb{R}^n$, and $V(q)$ is the potential energy of the system. This is not a too restrictive assumption since it still includes a quite broad family of systems. For unconstrained systems, we will further restrict to the case where M is a constant matrix and the Hamiltonian is separable. This assumption allows to implement symplectic numerical integration without needing implicit updates. On the other hand, in the constrained setting, we aim at preserving the geometry of the numerical flow map rather than other properties such as symplecticity. As a consequence, we can work with variable mass matrices still using explicit numerical integrators as in the unconstrained case.

We now briefly formalize how to extend this formulation to Hamiltonian systems that are holonomically constrained on some configuration manifold $\mathcal{Q} =$

$\{q \in \mathbb{R}^n : g(q) = 0\}$ embedded in $\mathbb{R}^n$ (for a more detailed derivation of this formalism we refer to [20, Chapter 8]). Many mechanical systems relevant for applications are characterized by the presence of some constraints that are coupled to the ODE defining the dynamics. One way to model this kind of problems is based on Lagrange multipliers, which lead to differential algebraic equations (DAEs). There has been some work in the direction of extending the Hamiltonian neural network's framework to constrained systems (see, e.g., [13] in which this strategy of introducing Lagrange multipliers is applied).

In this manuscript, we want to present an alternative approach based on the assumption that the constrained manifold $\mathcal{Q}$ is embedded in some linear space $\mathbb{R}^n$. This is actually not a restriction, since Whitney's embedding theorem always guarantees the existence of such an ambient space (see, e.g., [19, Chapter 6]). More explicitly, because of this embedding property, constrained multibody systems can be modeled by means of some projection operator and the vector field is written in such a way that it directly respects the constraints, without the addition of algebraic equations.

Furthermore, we assume that the components $g_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, ..., m$, are functionally independent on the zero level set, so that the Hamiltonian is defined on the $(2n-2m)$ dimensional cotangent bundle $\mathcal{M} = T^*\mathcal{Q}$. Working with elements of the tangent space at q, $T_q\mathcal{Q}$, as vectors in $\mathbb{R}^n$, we introduce a linear operator that defines the orthogonal projection of an arbitrary vector $v \in \mathbb{R}^n$ onto $T_q\mathcal{Q}$, i.e.

$$\forall\, q \in \mathcal{Q}, \text{ we set } P(q) : \mathbb{R}^n \to T_q\mathcal{Q}, \;\; v \mapsto P(q)\, v.$$

$P(q)^T$ can be seen as a map sending vectors of $\mathbb{R}^n$ into covectors in $T_q^*\mathcal{Q}$. If $g(q)$ is differentiable, assuming $G(q)$ is the Jacobian matrix of $g(q)$, we have $T_q\mathcal{Q} = \operatorname{Ker} G(q)$, and so $P(q) = I_n - G(q)\left(G(q)^T G(q))\right)^{-1} G(q)^T$, where $I_n \in \mathbb{R}^{n\times n}$ is the identity matrix. This projection map allows us to define Hamilton's equations as follows

$$\begin{cases} \dot{q} = P(q)\, \partial_p H(q,p) \\ \dot{p} = -P(q)^T \partial_q H(q,p) + W(q,p)\, \partial_p H(q,p), \end{cases} \tag{6.2.1}$$

where

$$W(q,p) = P(q)^T \Lambda(q,p)^T P(q) + \Lambda(q,p) P(q) - P(q)^T \Lambda(q,p)^T,$$

$$\text{with} \quad \Lambda(q,p) = \frac{\partial P(q)^T p}{\partial q}.$$

It is important to remark that since $T^*\mathcal{Q} \subset \mathbb{R}^{2n}$, we can work with the coordinates of the ambient space in the subsequent development. We notice that

when $\mathcal{Q} = \mathbb{R}^n$, we can set $P(q) = I$ and recover the unconstrained formulation. These equations of motion can be derived by the standard Hamilton's variational principle on the phase space or by the Legendre transform applied to the Euler-Lagrange equations. However, due to the geometry of the system, the variations need to be constrained to the right spaces and this is done with the projection map $P(q)$. We will focus on the case $Q = S^2 \times ... \times S^2 = (S^2)^k$ in Section 6.4.2, where the mass matrix $M(q)$ and equation (6.2.1) take a structured form, with S^2 the unit sphere in $\mathbb{R}^3$.

6.3 Learning unconstrained systems

As in [9], we base the training on a recurrent approach, that is graphically described in Figure 6.3.1.

As mentioned in Subsection 6.1.1, we work with numerically generated training trajectories that we denote by

$$\left\{\left(x_i, y_i^2, ..., y_i^M\right)\right\}_{i=1,...,N}.$$

We limit the treatment of noisy training data to Subsection 6.3.2. To obtain an approximation of the Hamiltonian H, we define a parametric model H_Θ and look for a Θ so that the trajectories generated by H_Θ resemble the given ones. H_Θ in principle can be any parametric function depending on the parameters Θ. In our approach, Θ will collect a factor of the mass matrix and the weights of a neural network, as specified in equation (6.3.3). We use a numerical one-step method $\Psi^{\Delta t}_{X_{H_\Theta}}$ to generate the trajectories

$$\hat{y}_i^j(\Theta) := \Psi^{\Delta t}_{X_{H_\Theta}}\left(\hat{y}_i^{j-1}(\Theta)\right), \quad \hat{y}_i^1(\Theta) := x_i, \quad j = 2, \dots, M,\ i = 1, \dots, N. \tag{6.3.1}$$

For unconstrained problems we use symplectic numerical integrators, since they can take an explict form and their adoption in the training procedure allows to have a target modified Hamiltonian to approximate (see, e.g., [27]). We then optimize a loss function measuring the distance between the given trajectories y_i^j and the generated ones $\hat{y}_i^j$, defined as

$$\mathcal{L}(\Theta) := \frac{1}{2n}\frac{1}{NM}\sum_{i=1}^{N}\mathcal{L}_i(\Theta) = \frac{1}{2n}\frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}\left\|\hat{y}_i^j(\Theta) - y_i^j\right\|^2, \tag{6.3.2}$$

where $\|\cdot\|$ is the Euclidean metric of $\mathbb{R}^{2n}$. This is implemented with the PyTorch `MSELoss` loss function. Such a training procedure resembles the one of Recurrent Neural Networks (RNNs), introduced in [24], as shown for the

forward pass of a single training trajectory in Figure 6.3.1. Indeed, the weight sharing principle of RNNs is reproduced by the time steps in the numerical integrator which are all based on the same approximation of the Hamiltonian, and hence on the same weights Θ. Finally, in Algorithm 3 we report one training epoch for a batch of data points.

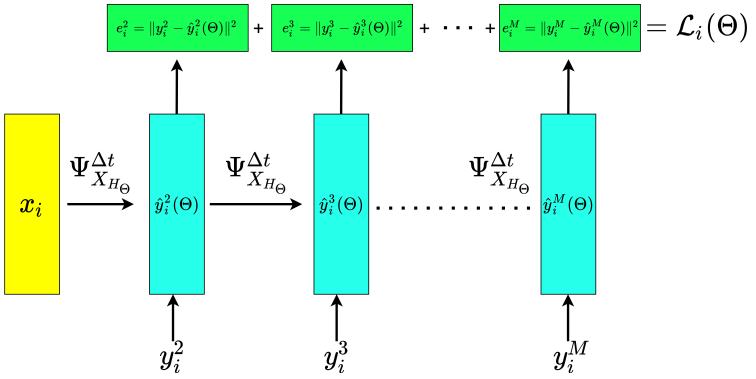


Figure 6.3.1: Forward pass of an input training trajectory $(x_i, y_i^2, ..., y_i^M)$. The picture highlights the resemblance to an unrolled version of a Recurrent Neural Network. The network outputs $(\hat{y}_i^2, ..., \hat{y}_i^M)$.

Algorithm 3 One epoch of the recurrent approximation of the Hamiltonian.

Choose a numerical integrator (s stages)
$\hat{N} \leftarrow$ `batch size`, Loss $\leftarrow 0$
for $i = 1, \dots, \hat{N}$ **do**
  $\hat{y}_i^1 \leftarrow x_i$
  for $j = 1, \dots, M$ **do**
    $\hat{y}_i^{j,[1]} \leftarrow \hat{y}_i^j$
    for $k = 1, \dots, s-1$ **do**
      Compute current value of Hamiltonian $H_\Theta\left(\hat{y}_i^{j,[k]}\right)$
      Compute $\nabla H_\Theta\left(\hat{y}_i^{j,[k]}\right)$ ▷ With automatic differentiation
      Compute stage $\hat{y}_i^{j,[k+1]}$
    end for
    Compute $\hat{y}_i^{j+1}$
    Increase Loss following equation (6.3.2)
  end for
end for
Optimize Loss

6.3.1 Architecture of the network

In this work, the role of the neural network is to model the Hamiltonian, i.e., a scalar function defined on the phase space $\mathbb{R}^{2n}$. Thus, the starting and arrival spaces are fixed. For unconstrained systems we assume that

$$H(q,p) = \frac{1}{2} p^T M^{-1} p + V(q) = K(p) + V(q)$$

is separable. Here, the kinetic energy is a quadratic form defined by the symmetric positive definite matrix M^{-1}. It can hence be modeled through a learnable matrix A, $K(p) \approx K_A(p)$, by replacing M^{-1} or M with $A^T A$ during the learning procedure. This modeling choice improves extrapolation properties since it allows to learn local (on a compact set) information that is valid on a larger domain, i.e., the mass matrix. In Section 6.4 we extend this reasoning to some configuration dependent mass matrices, where $M(q)$ is modeled through a constant symmetric and positive definite matrix. Recalling that $A^T A$ can even be singular or close to singular, one can promote the positive definiteness of the modeled matrix adding a positive definite perturbation matrix to $A^T A$. Notice that, in principle, the imposition of the positive (semi)definiteness of the matrix defining the kinetic energy is not necessary, but it allows to get more interpretable results. Indeed, it is known that the kinetic energy should define a metric on $\mathbb{R}^n$ and the assumption we are making guarantees such a property. For constrained systems we proceed in a similar way, as shown in equation (6.4.3). For the potential energy, a possible modeling strategy is to work with standard feedforward neural networks, and hence to define

$$V(q) \approx V_\theta(q) = f_{\theta_m} \circ \ldots \circ f_{\theta_1}(q),$$

$$\theta_i = (W_i, b_i) \in \mathbb{R}^{n_i \times n_{i-1}} \times \mathbb{R}^{n_i},\ \theta := [\theta_1, \ldots, \theta_m],$$

$$f_{\theta_i}(u) := \Sigma(W_i u + b_i),\ \mathbb{R}^n \ni z \mapsto \Sigma(z) = \left[\sigma(z_1), \ldots, \sigma(z_n)\right] \in \mathbb{R}^n,$$

for example with $\sigma(x) = \tanh(x)$. In particular applications, where some additional information is known about the system, one can impose more structure on the architecture modelling $V(q)$. For example, in the case of odd potential or rotationally symmetric potential, one can define respectively an odd neural network V_θ or a rotationally equivariant one (see, e.g., [4]). Therefore, we have that

$$\Theta = [A, \theta], \quad H(q,p) \approx H_\Theta(q,p) = K_A(p) + V_\theta(q). \tag{6.3.3}$$

We remark that the Hamiltonian does not need to be approximated by a neural network, and hence in a compositional way. Many other parametrizations are possible. For example, starting from the sparse identification of dynamical systems approach presented in [3], in [12] it is proposed to parametrize

the Hamiltonian with a dictionary of functions, for example polynomials and trigonometric functions. In our work, however, we opt for standard feedforward neural networks as the modeling assumption.

We now provide further details on the extrapolation capabilities of this network model. The learning procedure presented above is based on extracting temporal information coming from a set of trajectories belonging to a compact subset $\Omega \subset \mathbb{R}^{2n}$. In general, there is no reason why the Hamiltonian should be accurate outside of this set. To be more precise, denoting by $T > 0$ the largest time at which we know the trajectories, we have that

1. given enough samples in set Ω, distributed in order to capture the behavior of the dynamical system, the prediction of the network is expected to be accurate in $\Omega_{[0,T]} := \{\Phi^t_{X_H}(x) : t \in [0, T],\ x \in \Omega\}$, i.e., for any $z_0 \in \Omega_{[0,T]}$ and any $\bar{t} > 0$ such that $\Phi^t(z_0) \in \Omega_{[0,T]}$ for all $t \in [0, \bar{t}]$,
2. outside $\Omega_{[0,T]}$ one cannot guarantee that the prediction will be accurate.

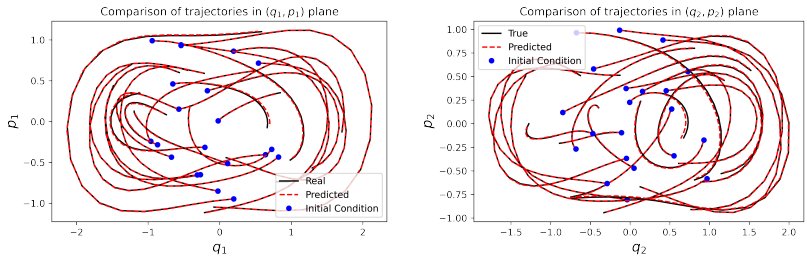
If we think of classical regression problems or even classification ones, it seems reasonable not to have information about the approximated quantity outside the sampled area. In those cases, with generalization we mean being sufficiently accurate close to the training points but still inside the sampled domain. However, here we know that the inferred function $H(q, p)$ has physical meaning and properties, so we might incorporate global known information about it to extend the applicability of the predictions.

This discussion supports the architectural choice for the kinetic energy suggested before (as in [26]). Indeed, supposing the Hamiltonian is separable, we know that the variable p appears in the energy function only via the quadratic form $\frac{1}{2} p^T M^{-1} p$. Thus, our modeling assumption allows us to approximate the mass matrix M just from a set of trajectories, hence capturing the dependency of H on the variable p also outside $\Omega_{[0,T]}$. Other possible improvements can be obtained when some symmetry structure is known for the Hamiltonian. On a similar direction, in Subsection 6.3.2, we add some regularization based on other prior physical knowledge.

We present in Figure 6.3.2 the comparison between ten learned trajectories and the corresponding exact ones of the Hamiltonian system $X_H \in \mathfrak{X}(\mathbb{R}^4)$ with Hamiltonian

$$H(q,p) = \frac{1}{2}\begin{bmatrix} p_1 & p_2 \end{bmatrix}^T \begin{bmatrix} 5 & -1 \\ -1 & 5 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} + \frac{q_1^4 + q_2^4}{4} + \frac{q_1^2 + q_2^2}{2}. \tag{6.3.4}$$

The training procedure of the network is based on 900 trajectories, sampled uniformly in six time instants, on the interval $0 \le t \le 0.3$. We remark that



(a) Projection on (q_1, p_1) **(b)** Projection on (q_2, p_2)

Figure 6.3.2: Comparison of real and predicted test trajectories for the Hamiltonian (6.3.4). In this case, for the potential energy we used a feedforward network with three hidden layers having respectively 100, 50 and 50 neurons and tanh as activation function. The training integrator is Störmer-Verlet, with $M = 6$ and final time $T = 0.3$ and we use the Adam optimizer. The test trajectories, at $\tilde{M} = 20$ uniformly distributed points in the time interval $[0, 1]$, are obtained with ODE(5,4). These trajectories correspond to $\tilde{N} = 100$ initial conditions on which the network has not been trained.

the training initial conditions are carefully chosen so that their associated trajectory segments well-capture the dynamics of interest. Figure 6.3.2 collects test trajectories corresponding to the time interval $[0, 1]$. Since we are interested in approximating the Hamiltonian and not directly the trajectories, we are not constrained to evaluate the quality of the approximation with the same time integrator as the one used for training. In fact, these test trajectories have been generated with an embedded Runge–Kutta pair of order (5,4), with same relative and absolute accuracies for both the real and learned systems. Experimentally, it is clear that the qualitative behavior of the Hamiltonian is well captured, as we can see from Figure 6.3.2. To quantify the agreement of the prediction with the true Hamiltonian we report the $\mathcal{E}_1$ metric, as defined in (6.1.2), that is $6.59 \cdot 10^{-5}$. Furthermore, the training loss is $4.62 \cdot 10^{-7}$.

6.3.2 Robustness to noise and regularization

In real-world applications, data is contaminated by noise, which usually comes from the measurement process. Thus, we need to test the robustness of the learning framework to the presence of noise in the training trajectories. To do so, we synthetically generate the trajectories as before, and then add random normal noise to all the points except the initial condition (for an averaging strategy that allows to deal even with perturbed initial conditions, see, e.g., [9]). By construction, the network necessarily learns a Hamiltonian function, that is expected to generate trajectories close to the noisy ones. Since the training does not rely on clean trajectories, it is reasonable not to expect neither

a loss value which is as small as in the absence of noise, nor a too accurate approximation of the Hamiltonian and the trajectories. Nevertheless, we aim for a learned Hamiltonian with level sets close to the exact ones, hence giving trajectories that resemble the true ones. One way of improving the quality of the neural networks proposed here, is to make use of a priori known physical properties of the dynamical system. We use an approach based on soft constraints which means that we take the known physical properties into account by adding a regularization term in the cost function. An example of such a property could be one or more known conserved quantities, so called first integrals. Hamiltonian systems always have at least one first integral, namely the Hamiltonian function itself, but there might be additional independent ones. Enforcing the first integrals to be preserved or nearly preserved seems to be a reasonable strategy for obtaining improved qualitative behavior of the resulting approximation as shown in the following example.

Consider a Hamiltonian system with Hamiltonian function $H : \mathbb{R}^{2n} \to \mathbb{R}$, and a functionally independent first integral G, i.e. $\nabla H(x)$ and $\nabla G(x)$ are never parallel. Consider the numerical integration $\hat{y}_k^j$, $j = 1, ..., M$ of the approximated Hamiltonian vector field X_{H_Θ}, starting at $\hat{y}_k^1 = x_k$. In the ideal case in which the learned Hamiltonian H_Θ coincides with H and the numerical flow is replaced with the exact one, both H and G should be conserved. For this reason, we suggest adding to the loss function in equation (6.3.2) the following "regularization" term:

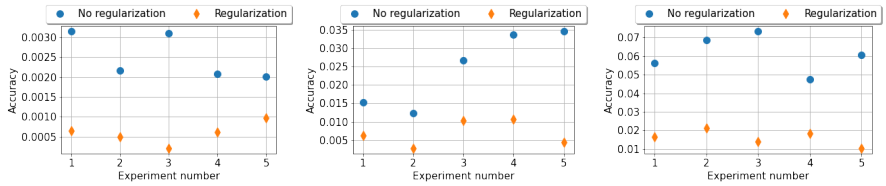
$$\mu \sum_{j \in \mathcal{I}} \left(G\left(\hat{y}_k^j\right) - G\left(x_k\right) \right)^2$$

for all the training points x_k. Here $\mathcal{I}$ is a subset of indices contained in $\{1, ..., M\}$, and μ is a regularization parameter that balances the importance of the preservation of the additional first integral against the perfect fitting of the training trajectories. We test this regularization procedure with the Hamiltonian system $X_H \in \mathfrak{X}(\mathbb{R}^4)$ defined by

$$H\left(q_1, q_2, p_1, p_2\right) = \frac{q_1^2 + p_1^2}{2} + \frac{p_2^2}{2} + \frac{1}{2} q_2^2 + \frac{1}{4} q_2^4 = h_1\left(q_1, p_1\right) + h_2\left(q_2, p_2\right).$$

This system has $G(q, p) := h_1(q_1, p_1)$ as an additional independent first integral other than H. We report in Figure 6.3.3 some plots of the obtained $\mathcal{E}_1$ values as defined in (6.1.2). In these experiments, we add some random noise of the form $\varepsilon\delta$ to the points y_i^j of the numerical trajectories, where $\delta \sim \mathcal{N}(0, 1)$ follows a standard normal distribution. The same experiment is run 5 times, and for each of these we plot the obtained $\mathcal{E}_1$ value. For each experiment, we generate new training and test trajectories, and these are used for both the regularized training and the non-regularized one. Furthermore, each experiment has a different random initialization of the weights, which is however shared between

the regularized and non regularized networks. We notice that with regularization we can consistently get a better error in terms of the $\mathcal{E}_1$ measure. There is not a huge difference between the results, however. This suggests that when prior information is known, it might be important to experiment with this kind of regularizing terms. To conclude the Section, we highlight how remarkable



(a) Case $\varepsilon = 0.1$ **(b)** Case $\varepsilon = 0.3$ **(c)** Case $\varepsilon = 0.5$

Figure 6.3.3: 5 repeated experiments for each perturbation regime. We plot on the y axis the average accuracy, in terms of the $\mathcal{E}_1$ measure, obtained with the trained network, when compared with the real (non-noisy) trajectories.

it is that even without the regularization term, the trajectories are qualitatively well captured by the network and hence the test error is quite low. This is mostly due to the prior physical knowledge we impose on the learning procedure, i.e. that the vector field should be Hamiltonian. Indeed, since in the worst case the network approximates the wrong Hamiltonian, we always expect that it does not overfit the noisy trajectories, since they can not be learned exactly. On the other hand, without the prior knowledge of the Hamiltonian nature of the system, all the overfitting problems of standard neural networks reoccur and the risk of being closer to an interpolant of the noisy trajectories is higher.

6.4 Learning constrained Hamiltonian systems

The approximation of the Hamiltonians of constrained mechanical systems with neural networks has already been studied in the literature. Two main approaches can be identified. One of them is based on local coordinates on the constrained manifold (see, e.g., [9, 14]) and the other uses ambient space coordinates and Lagrange multipliers (see [13]). In principle, both the formulations apply to any constrained Hamiltonian system. However, as remarked in [13], the choice of a redundant system of coordinates usually gives a simpler expression for the Hamiltonian. This results in a more data-efficient training procedure. In the second approach, an embedded Runge–Kutta pair of order (5,4) is used to train the network. This choice inevitably leads to a drift from the constrained manifold during the training, even if it can be reduced by setting the tolerances of the integrator. However, in this way the cost of the

integrator increases, hence this is not the most efficient way to preserve the constraints.

In this work, we use an alternative global formulation of the dynamics, as introduced in Section 6.2. In principle this formulation adapts to any constrained Hamiltonian system whose configuration manifold is a submanifold of $\mathbb{R}^n$. Coupling this description of the dynamics with the learning framework introduced in Section 6.3, their Hamiltonian functions can be approximated. To be more precise, one can use any numerical integrator to discretize the constrained trajectories and compare them with the training data. For example, Runge–Kutta 4 method can be used and this experimentally gives fast training procedures and accurate approximations of the Hamiltonian, as shown in the experiments of Subsection 6.4.3.

We remark that in general numerical integrators do not preserve the geometry of the system and there might be a drift from the constrained manifold (see, e.g., [15, Chapter 7]). Experimentally this does not seem to have a great impact on the quality of the predicted Hamiltonian in most of the cases. However, as we present in the numerical experiments with Lie group integrators, there might be situations in which one benefits from training the Hamiltonian with an integrator preserving the phase space. Notice that the Hamiltonian that defines the dynamics has non-unique extension outside the phase space $\mathcal{M} = T^*\mathcal{Q}$. This is due to the projection matrix $P(q) = I_n - G(q)\left(G(q)^T G(q)\right)^{-1} G(q)^T$ appearing in equation (6.2.1), where $G(q)$ is the Jacobian matrix of the constraint function $g(q)$ defining $\mathcal{Q}$. This justifies investigating the importance of the preservation of the manifold $T^*\mathcal{Q}$ in the training procedure.

As introduced in Section 6.2, in this work we assume that the constrained configuration manifold $\mathcal{Q}$ is known. Referring to equation (6.2.1), we notice that once the geometry is known, it is enough to specify the Hamiltonian function $H : T^*\mathcal{Q} \subseteq \mathbb{R}^{2n} \to \mathbb{R}$ in order to characterize the dynamics of a system. We show a setting in which the geometry can be preserved by Lie group integrators (see, [18, 6, 8]) focusing on the case $T^*\mathcal{Q}$ is homogeneous[2]. We see this even as an opportunity to study the behavior of this class of methods in an applied framework and combined with neural networks. This geometric setup applies, for example, when $\mathcal{Q}$ is a homogeneous manifold and the transitive action $\psi : G \times \mathcal{Q} \to \mathcal{Q}$ defines, for any $q \in \mathcal{Q}$, a submersion $\psi_q : G \to \mathcal{Q}$ at the identity element $e \in G$ (see, e.g., [2, 7]). Cartesian products of homogeneous manifolds are homogeneous too. Usually, multibody systems have constrained configuration manifolds given by cartesian products of S^2, $\mathbb{R}^k$, $SO(3)$ and $SE(3)$, which

[2] A smooth manifold $\mathcal{M}$ is homogeneous if for any pair of points $m_1, m_2 \in \mathcal{M}$ there is $g \in G$ such that $\psi(g, m_1) = m_2$, where $\psi : G \times \mathcal{M} \to \mathcal{M}$ is a Lie group action. In other words, ψ is a transitive action.

are respectively the special orthogonal and Euclidean groups. These are all homogeneous manifolds and so are their tangent and cotangent bundles.

6.4.1 Lie group methods and neural networks

Among the various classes of Lie group methods, we consider the Runge–Kutta–Munthe–Kaas (RKMK) methods and the commutator-free ones (see, e.g., [22, 5]). The underlying idea of RKMK methods, applied to $F \in \mathfrak{X}(\mathcal{M})$, with $\mathcal{M}$ an homogeneous manifold, is to express F as $F|_m = \psi_*(f(m))|_m$. Here ψ_* is the infinitesimal generator of ψ, a transitive Lie group action of G on $\mathcal{M}$, and $f : \mathcal{M} \to \mathfrak{g}$ is a function that locally lifts the dynamics to the Lie algebra $\mathfrak{g}$ of G. On this linear space, we can perform a time step integration. We then map the result back to $\mathcal{M}$, and repeat this up to the final integration time. More explicitly, let Δt be the size of the uniform time step of the discretization, we then update $y_n \in \mathcal{M}$ to y_{n+1} by

$$\begin{cases} \gamma(0) = 0 \in \mathfrak{g}, \\ \dot{\gamma}(t) = \mathrm{dexp}^{-1}_{\gamma(t)} \circ f \circ \psi\left(\exp\left(\gamma(t)\right), y_n\right) \in T_{\gamma(t)}\mathfrak{g}, \\ y_{n+1} = \psi\left(\exp\left(\gamma_1\right), y_n\right) \in \mathcal{M}, \end{cases} \tag{6.4.1}$$

where $\gamma_1 \approx \gamma(\Delta t) \in \mathfrak{g}$ is computed with a Runge–Kutta method, and dexp^{-1} is the inverse of the differential of the exponential map $\exp : \mathfrak{g} \to G$ as defined, for example, in [18, Section 2.6]. We do not go into the details of commutator-free methods, but the following development applies to them as well. In particular, the function f still plays a fundamental role.

We now present a natural way to combine the learning framework typical of unconstrained systems with Lie group integrators. This is done by introducing a Lie group method during the learning procedure. Indeed, since we want to apply a Lie group integrator to deal with nonlinear geometries, we set $\Psi^{\Delta t}$, defined in equation (6.3.1), to be the Δt update given by some RKMK method. In other words, using the notation of equation (6.4.1), we get $\Psi^{\Delta t}(z) = \psi(\exp(\gamma_1), z)$ with $\gamma_1 \in \mathfrak{g}$.

The setting presented above for generic vector fields on homogeneous manifolds simplifies considerably in the presence of Hamiltonian systems. Indeed, for this type of systems, what is needed to fully determine the dynamics is the geometry given by $\mathcal{M} = T^*\mathcal{Q}$ and the scalar Hamiltonian function $H : \mathcal{M} \to \mathbb{R}$. In other words, we can think of the function $f : \mathcal{M} \to \mathfrak{g}$, that allows to express the vector field in terms of the infinitesimal generator of the action, as the result of an operator $F : C^1(\mathcal{M}, \mathbb{R}) \to \{\mathcal{M} \to \mathfrak{g}\}$ acting on a scalar function H. More explicitly, we can write $f = F[H]$ where F and H encode, respectively,

the geometry and the dynamics of the system. This operator is not really necessary, but it clarifies considerably how the neural network comes into play in the learning framework. Indeed, because of this construction, we can write the numerical flow $\Psi^{\Delta t}$ as the map sending y_n into $y_{n+1} = \psi(\exp(\gamma_{\Delta t, y_n}), y_n)$ with $\gamma_{\Delta t, y_n}$ being an approximation of the solution $\gamma(\Delta t)$ of the following initial value problem

$$\begin{cases} \dot{\gamma}(t) = \operatorname{dexp}^{-1}_{\gamma(t)} \circ F\left[H_\Theta\right] \circ \psi\left(\exp\left(\gamma(t)\right), y_n\right) \in T_{\gamma(t)}\mathfrak{g}, \\ \gamma(0) = 0 \in \mathfrak{g}. \end{cases}$$

Here H_Θ is the approximation of the Hamiltonian given by the current weights Θ of the neural network. Thus, applying a particular family of geometric numerical integrators, we can directly study some constrained systems with the same ideas coming from learning unconstrained ones. Since following this procedure the geometry is preserved, one can consider replacing the Euclidean distance in the loss function defined in equation (6.3.2) with a Riemannian metric of the constrained manifold. This would bring to distances between points that correspond to the length of the minimal geodesic connecting them, which is, in general, different from the length of the segment in the ambient space having them as extrema. In the remaining part of the Section, we specialize this reasoning to mechanical systems defined on copies of T^*S^2. We focus on a chain of spherical pendula, but the geometric setting also applies to other systems (see, e.g., [20, Section 10.5]).

6.4.2 Mechanical systems on $\left(T^*S^2\right)^k$

As anticipated in the introductory section, in this geometric setting, we do not involve symplectic integrators, and we do not assume we have a separable Hamiltonian anymore. Thus, we now model a more general family of Hamiltonians as

$$H(q,p) = \frac{1}{2} p^T M^{-1}(q)\, p + V(q). \tag{6.4.2}$$

We model the potential energy as before. However, we need an alternative strategy for the inverse of the mass matrix, which is no longer assumed to be constant. Based on the problem, one can choose various parametrizations of the mass matrix or its inverse. We decide to specialize the architecture based on the fact that the geometry of the system is known to be $\mathcal{M} = (T^*S^2)^k$, where $S^2 \subset \mathbb{R}^3$. We coordinatize $\mathcal{M}$ with $(q,p) = (q_1, \dots, q_k, p_1, \dots, p_k) \in \mathbb{R}^{6k}$. In this case, when $p \in \mathbb{R}^{3k}$ is intended as the vector of linear momenta, the matrix

$M(q)$ in equation (6.4.2) is a block matrix, with

$$i,j=1,\dots,k,\quad \mathbb{R}^{3\times 3}\ni M(q)_{ij}=\begin{cases} m_{ii}I_3, & i=j \\ m_{ij}(I_3-q_iq_i^T), & \text{otherwise,}\end{cases}$$

see [20, Section 8.3.3] for further details. Here, the matrix having constant entries m_{ij} is symmetric and positive definite. For this reason, we leverage this form of the kinetic energy and learn a constant matrix $A\in\mathbb{R}^{k\times k}$ and a vector $b\in\mathbb{R}^k$ so that

$$\begin{bmatrix} m_{11} & \dots & m_{1k} \\ m_{21} & \dots & m_{2k} \\ \vdots & \vdots & \vdots \\ m_{k1} & \dots & m_{kk}\end{bmatrix}\approx A^TA+\begin{bmatrix}\tilde{b}_1 & 0 & \dots & 0 \\ 0 & \tilde{b}_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \tilde{b}_k\end{bmatrix} \tag{6.4.3}$$

where $\tilde{b}_i:=\max(0,b_i)$ are terms added to promote the positive definiteness of the right-hand side. We also tested elevating to the second power the b_i instead of taking the maximum with 0, but we got better results with the choice presented in equation (6.4.3). The matrix on the left-hand side of equation (6.4.3) is exactly the one appearing in Hamiltonian formulations with Cartesian coordinates, as the one used in [13].

For the spherical pendulum, we have $k=1$, and hence, the Hamiltonian dynamics is defined on the cotangent bundle T^*S^2, which is a homogeneous manifold. This can be obtained thanks to the transitivity of the group action

$$\Psi: SE(3)\times T^*S^2\to T^*S^2,\ \left(\left(R,r\right),\left(q,p^T\right)\right)\mapsto\left(Rq,\left(Rp+r\times Rq\right)^T\right),$$

where the transpose comes from the usual interpretation of covectors as row vectors. As in [16, Chapter 6], we represent a generic element of the special Euclidean group $G=SE(3)$ as an ordered pair (R,r), where $R\in SO(3)$ is a rotation matrix and $r\in\mathbb{R}^3$ is a vector. With this specific choice of the geometry, the formulation presented in equation (6.2.1) simplifies considerably. Indeed $P(q)=I_3-qq^T$ which implies $W(q,p)=pq^T-qp^T$. Replacing these expressions in (6.2.1) and using the triple product rule, we end up with the following set of ODEs

$$\begin{cases}\dot{q} &= \left(I-qq^T\right)\partial_pH\left(q,p\right) \\ \dot{p} &= -\left(I-qq^T\right)\partial_qH\left(q,p\right)+\partial_pH\left(q,p\right)\times\left(p\times q\right).\end{cases} \tag{6.4.4}$$

This vector field $X(q,p)$ can be expressed as $\psi_*(F[H](q,p))(q,p)$ with

$$\psi_*\left(\left(\xi,\eta\right)\right)\left(q,p\right)=\left(\xi\times q,\xi\times p+\eta\times q\right),\quad \left(\xi,\eta\right)\in\mathfrak{g}=\mathfrak{se}(3)$$

and

$$F[H](q,p) = (\xi,\eta) = \left(q \times \frac{\partial H(q,p)}{\partial p}, \frac{\partial H(q,p)}{\partial q} \times q + \frac{\partial H(q,p)}{\partial p} \times p \right).$$

A similar reasoning can be extended to a chain of k connected pendula and hence to a system on $(T^*S^2)^k$. The main idea is to replicate both the equations in (6.4.4) and the expression $F[H]$ for all the k copies of T^*S^2. A more detailed explanation can be found in [8].

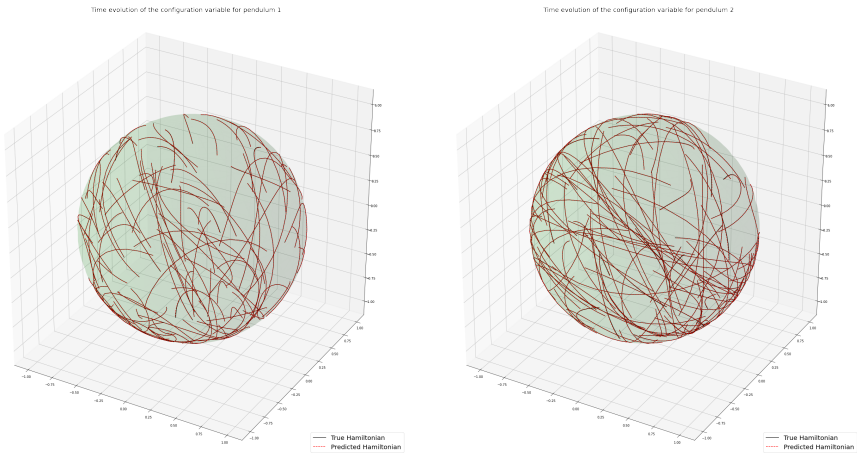


Figure 6.4.1: Comparison between 100 test trajectories obtained with the true Hamiltonian H and the predicted one H_Θ. To train H_Θ, a Lie group method is used. This gives $\mathcal{E}_1 = 2.65 \cdot 10^{-6}$ and a final training loss of $1.6 \cdot 10^{-9}$.

We present in Figure 6.4.1 the results obtained for the training of a double pendulum, i.e. $k = 2$. To train the network, we generate a set of $N = 500$ training trajectories with the embedded Runge–Kutta pair of order (5,4) of `SciPy`. The final integration time is $T = 0.1$ and $M = 5$. To model the potential energy, we use a feedforward network with three hidden layers of 100 neurons each. In the plots, we show the configuration variables, $q_1, q_2 \in S^2$, obtained for 100 test trajectories in the time interval $[0,1]$, where the network H_Θ has been trained with a commutator-free method of order 4.

6.4.3 Experimental study of the learning procedure

We investigate the influence of the training setup on the error measures $\mathcal{E}_1$, $\mathcal{E}_2$, defined in (6.1.3), and on the training loss. More precisely, we test how the parameters M, N, the noise magnitude, and the training integrator affect the

performance of the network. We quantify the magnitude of noise in the training trajectories with a parameter $\varepsilon > 0$, as in Subsection 6.3.2. The integrators that we study are Lie Euler, explicit Euler (both of order 1), commutator-free, and Runge–Kutta (both of order 4). In particular, Lie Euler and commutator-free methods preserve the phase space $\mathcal{M}$ up to machine accuracy. To get a sufficient sample of experiments, we repeat all the tests five times and look at the medians and geometric means[3] of the obtained results. To be precise, we test $N \in \{50, 500, 1000, 1500\}$, $M \in \{2, 3, 5\}$, and $\varepsilon \in \{0, 0.001, 0.01, 0.1\}$. Therefore, we perform a total of 960 experiments, and also here the potential energy is modeled with a feedforward network of 3 hidden layers having 100 neurons each. Furthermore, for the four experiments performed varying just the integrator and with the other parameters fixed, the network's weights are initialized to be the same, and the training and test initial conditions are the same. For all these experiments, we focus on the single spherical pendulum, we keep the final training time to $T = 0.1$, and we do not use regularization terms. The training trajectories have been generated with the `SciPy` implementation of the Dormand-Prince pair of order (5,4) with strict tolerance. As shown in Table 6.4.1, the order of the numerical integrator used to train the network plays an important role. Indeed, we get results that are similar for methods of the same order, but there is a noticeable decay in the errors and in the loss when we increase the order from one to four. As highlighted in [27], this effect can be explained with a standard argument of backward error analysis, see, e.g., [15, Chapter 9]. From the results reported in Table 6.4.1, we see that the local error of the integrator is more important than the preservation of the geometry. Therefore, even if, from a theoretical point of view, it seems relevant to remain on the manifold during the training, in practice, this does not seem to be very important in the particular experiment considered here. In Figure 6.4.2, we plot the dependencies of $\mathcal{E}_1$, $\mathcal{E}_2$ and the training loss, on N, M, ε and the integrator. We notice that values of $\mathcal{E}_1$ below a threshold of 10^{-7} can be reached only with

Order	Integrator	$\mathcal{E}_1$	$\mathcal{E}_2$	Training Loss
1	EE	5.7e-5	1.13e-2	2.12e-6
1	LE	4.9e-5	1.07e-2	1.17e-6
4	RK4	1.12e-5	3.83e-3	2.63e-7
4	CF4	1.12e-5	3.85e-3	2.64e-7

Table 6.4.1: In this table, we report the geometric means of the quantities $\mathcal{E}_1$, $\mathcal{E}_2$, and the training loss. Here we average over all the 240 experiments that have the same integrator. We denote the four integrators with EE (explicit Euler), LE (Lie Euler), RK4 (Runge–Kutta 4), and CF4 (commutator-free 4).

[3] The choice of geometric means is because of the exponential nature of the error measures and the training loss.

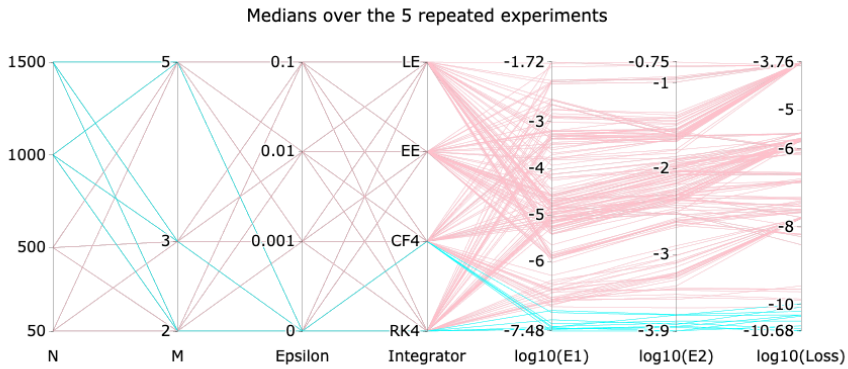


Figure 6.4.2: This is a parallel coordinate plot reporting the dependencies of $\mathcal{E}_1$, $\mathcal{E}_2$ and the training loss on the parameters N, M, ε and on the integrator. Each colored polyline corresponds to the median over five experiments, i.e., same N, M, ε, and same integrator. The lines in cyan color represent the combinations giving $\mathcal{E}_1 < 10^{-7}$.

integrators of order four and with the smallest value of ε. The interplay of N, M, and ε is further investigated in Table 6.4.2. An interactive version of Figure 6.4.2, together with other parallel coordinate plots, can be found at the GitHub Page https://davidemurari.github.io/learningConstrainedHamiltonians/, while the dataset is available in the GitHub repository associated to the paper.

We conclude this parameter study considering separately the case with and without noise, $\varepsilon > 0$ and $\varepsilon = 0$ respectively. The results are reported in Table 6.4.2. In general, the lowest values of $\mathcal{E}_1$ are obtained with high N. For the model under consideration, $N = 1000$ seems already high enough to achieve good results. Regarding M, Table 6.4.2 shows that to achieve lower values of $\mathcal{E}_1$ in the presence of noise, one needs to adopt a higher M. On the other hand, in the absence of noise, it seems important to have a high M only for low-order integrators. Finally, as may be expected, even if this Table does not distinguish among the different magnitudes of the noise, we see that with $\varepsilon = 0$ better results can be achieved.

We also point out that the experiments were performed for short integration times, where not only symplectic integrators can generate physically meaningful trajectories. It would be interesting to explore the performance of symplectic and constraint-preserving integrators in this setting (see, e.g., [1]), and we defer this to further work.

Besides the theoretical aspect of the non-uniqueness of the extension of the dynamics outside of $\mathcal{M} \subset \mathbb{R}^{2n}$, we now report a numerical experiment where the preservation of the geometry during the training is beneficial. We consider

Without noise									
Integrator of order 1					Integrator of order 4				
N	M	Int.	$\mathcal{E}_1$	$\mathcal{E}_2$	N	M	Int.	$\mathcal{E}_1$	$\mathcal{E}_2$
1500	5	LE	1e-6	2.4e-3	1500	2	CF4	3.2e-8	1.3e-4
1000	5	LE	1e-6	2.3e-3	1500	5	RK4	3.3e-8	1.4e-4
1000	5	EE	1e-6	2.4e-3	1500	3	RK4	3.4e-8	1.4e-4
1500	5	EE	1e-6	2.5e-3	1500	2	RK4	3.6e-8	1.5e-4
500	5	LE	2e-6	2.6e-3	1500	3	CF4	3.7e-8	1.5e-4
With noise									
Integrator of order 1					Integrator of order 4				
N	M	Int.	$\mathcal{E}_1$	$\mathcal{E}_2$	N	M	Int.	$\mathcal{E}_1$	$\mathcal{E}_2$
1500	5	LE	1.2e-5	6.6e-3	1500	5	RK4	5e-6	3.4e-3
1500	5	EE	1.2e-5	6.3e-3	1500	5	CF4	6e-6	4.1e-3
1000	5	LE	1.5e-5	6.6e-3	1000	5	CF4	7e-6	3.8e-3
1000	5	EE	1.6e-5	6.8e-3	1000	5	RK4	8e-6	4.3e-3
500	5	LE	1.8e-5	6.7e-3	1000	3	RK4	8e-6	4.2e-3

Table 6.4.2: In this Table, we report the combinations that give the five best values of $\mathcal{E}_1$, together with the corresponding value $\mathcal{E}_2$. These are the geometric means among all the experiments. The two tables compare the performance on data with and without noise.

again a simple spherical pendulum, and we assume to know that the potential energy is linear. We hence impose this prior information on the architecture of the network. Due to the problem's simplicity, we aim to reach very low $\mathcal{E}_1$ and $\mathcal{E}_2$ values. Training the same architecture for 200 epochs, both with Runge–Kutta and commutator-free methods of order 4, we get the results in Table 6.4.3. Indeed, the geometric integrator outperforms the classical Runge-Kutta method in this experiment.

Numerical method in the training	$\mathcal{E}_1$	$\mathcal{E}_2$
Runge-Kutta of order 4	4.2e-12	1.5e-6
Commutator-free of order 4	1.1e-14	2.5e-7

Table 6.4.3: Comparison of the accuracy measures $\mathcal{E}_1$ and $\mathcal{E}_2$ obtained with the two integrators. These results are obtained by imposing the linear structure of the potential energy on the network modeling the Hamiltonian of the spherical pendulum. The kinetic energy has been modeled as in previous experiments.

This experiment suggests that the choice of an integrator that does not fully exploit the available information, like the geometry, might limit the quality

of the obtained approximations. For those cases in which one is interested in predictions that are as accurate as possible, this might be a relevant issue.

The experiments performed lead to the conclusion that modeling multi-body systems with neural networks can be a valuable approach. However, to better leverage the approximation capabilities of machine learning techniques (see, e.g., [17, 10]), we believe that a deeper investigation and understanding of how they interface with physical models is necessary.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 860124.

The authors are grateful to Ergys Çokaj for the valuable discussions in the early stage of this work. The authors would like to thank the Isaac Newton Institute for Mathematical Sciences, Cambridge, for support and hospitality during the program "Mathematics of Deep Learning".

Bibliography

[1] Hans C Andersen. Rattle: A "velocity" version of the shake algorithm for molecular dynamics calculations. *Journal of computational Physics*, 52(1):24–34, 1983. 244

[2] RW Brockett and HJ Sussmann. Tangent bundles of homogeneous spaces are homogeneous spaces. In *Proc. Amer. Math. Soc*, volume 35, pages 550–551, 1972. 238

[3] 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. 233

[4] Elena Celledoni, Matthias Joachim Ehrhardt, Christian Etmann, Brynjulf Owren, Carola-Bibiane Schonlieb, and Ferdia Sherry. Equivariant neural networks for inverse problems. *Inverse Problems*, 37, 2021. 233

[5] Elena Celledoni, Arne Marthinsen, and Brynjulf Owren. Commutator-free Lie group methods. *Future Generation Computer Systems*, 19(3):341–352, 2003. 239

[6] Elena Celledoni, Håkon Marthinsen, and Brynjulf Owren. An introduction to Lie group integrators–basics, new developments and applications. *Journal of Computational Physics*, 257:1040–1061, 2014. 238

[7] Elena Celledoni, Ergys Çokaj, Andrea Leone, Davide Murari, and Brynjulf Owren. Dynamics of the N-fold Pendulum in the framework of Lie Group Integrators. *arXiv preprint arXiv:2109.12325*, 2021. 35, 238

[8] Elena Celledoni, Ergys Çokaj, Andrea Leone, Davide Murari, and Brynjulf Owren. Lie group integrators for mechanical systems. *International Journal of Computer Mathematics*, 0(0):1–31, 2021. 226, 238, 242

[9] Zhengdao Chen, Jianyu Zhang, Martin Arjovsky, and Léon Bottou. Symplectic Recurrent Neural Networks. In *International Conference on*

Learning Representations, 2020. 48, 70, 71, 73, 149, 226, 227, 231, 235, 237

[10] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989. 3, 25, 246

[11] Marco David and Florian Méhats. Symplectic Learning for Hamiltonian Neural Networks. *arXiv preprint arXiv:2106.11753*, 2021. 228

[12] Daniel M. DiPietro, Shiying Xiong, and Bo Zhu. Sparse Symplectically Integrated Neural Networks. In *Advances in Neural Information Processing Systems 34*. Curran Associates, Inc., 2020. 233

[13] Marc Finzi, Alex Wang, and Andrew Gordon Wilson. Simplifying Hamiltonian and Lagrangian Neural Networks via Explicit Constraints. *NeurIPS*, 2020. 226, 230, 237, 241

[14] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian Neural Networks. *Advances in Neural Information Processing Systems*, 32:15379–15389, 2019. 145, 226, 227, 237

[15] E. Hairer, Ch. Lubich, and G. Wanner. *Geometric Numerical Integration*, volume 31 of *Springer Series in Computational Mathematics*. Springer, Heidelberg, 2010. Structure-Preserving Algorithms for Ordinary Differential Equations, Reprint of the second (2006) edition. 227, 238, 243

[16] Darryl D Holm. *Geometric Mechanics-Part II: Rotating, Translating and Rolling*. World Scientific, 2011. 241

[17] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991. 246

[18] Arieh Iserles, Hans Z Munthe-Kaas, Syvert P Nørsett, and Antonella Zanna. Lie-group methods. *Acta numerica*, 9:215–365, 2000. 238, 239

[19] John M. Lee. *Introduction to Smooth Manifolds*. Graduate Texts in Mathematics. Springer New York, NY, 2012. 12, 16, 17, 18, 227, 230

[20] T. Lee, M. Leok, and N H. McClamroch. Global formulations of Lagrangian and Hamiltonian Dynamics on Manifolds. *Springer*, 13:31, 2017. 2, 226, 230, 240, 241

[21] Jerrold E Marsden and Tudor S Ratiu. Introduction to Mechanics and Symmetry. *Physics Today*, 48(12):65, 1995. 227

[22] H. Munthe-Kaas. High order Runge–Kutta Methods on Manifolds. *Appl. Num. Math.*, 29:115–127, 1999. 183, 184, 185, 188, 219, 239

[23] Christian Offen and Sina Ober-Blöbaum. Symplectic integration of learned Hamiltonian systems. *arXiv preprint arXiv:2108.02492*, 2021. 226

[24] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning Internal Representations by Error Propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985. 231

[25] E. T. Whittaker. *A Treatise on the Analytical Dynamics of Particles and Rigid Bodies*. Cambridge University Press, 1993. Fourth edition. 227

[26] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ODE-Net: Learning Hamiltonian Dynamics with Control. *arXiv preprint arXiv:1909.12077*, 2019. 226, 227, 234

[27] Aiqing Zhu, Pengzhan Jin, and Yifa Tang. Deep hamiltonian networks based on symplectic integrators. *arXiv preprint arXiv:2004.13830*, 2020. 70, 146, 231, 243

Neural networks for the approximation of Euler's elastica

Elena Celledoni, Ergys Çokaj, Andrea Leone, Sigrid Leyendecker, Davide Murari, Brynjulf Owren, Rodrigo T. Sato Martín de Almagro, Martina Stavole

Submitted

Abstract. Euler's elastica is a classical model of flexible slender structures relevant in many industrial applications. Static equilibrium equations can be derived via a variational principle. The accurate approximation of solutions to this problem can be challenging due to nonlinearity and constraints. We present two neural network-based approaches for simulating Euler's elastica. Starting from a data set of solutions of the discretised static equilibria, we train the neural networks to produce solutions for unseen boundary conditions. We present a *discrete* approach learning discrete solutions from the discrete data. We then consider a *continuous* approach using the same training data set but learning continuous solutions to the problem. We present numerical evidence that the proposed neural networks can effectively approximate configurations of the planar Euler's elastica for a range of different boundary conditions.

7.1 Introduction

Modelling of mechanical systems is relevant in various branches of engineering. Typically, it leads to the formulation of variational problems and differential equations, whose solutions are approximated with numerical techniques. The efficient solution of linear and nonlinear systems resulting from the discretisation of mechanical problems has been a persistent challenge of applied mathematics. While classical solvers are characterised by a well-established and mature body of literature [42, 36, 32, 18, 19, 3, 39], the past decade has witnessed a surge in the use of novel machine learning-assisted techniques [8, 5, 40, 44, 10, 17, 21, 26, 34, 29, 30, 6, 25, 45, 9, 13, 35]. These approaches aim at enhancing solution methods by leveraging the wealth of available data and known physical principles. The use of deep learning techniques to improve the performance of traditional numerical algorithms in terms of efficiency, accuracy, and computational scalability [5], is becoming increasingly popular also in computational mechanics (see, e.g. [27]). Examples include a wide range of problems that require the approximation of functions, as well as efficient reduced order modelling [4] or more specific numerical tasks such as optimising the quadrature rule for computing the finite element stiffness matrix [52] or the investigation of data-driven numerical frameworks for the bifurcation analysis of partial differential equations [12, 15]. This recent literature is evidence that neural networks can be used successfully as surrogate models for the solution operators of various differential equations.

In the context of ordinary and partial differential equations, two main trends can be identified. The first one aims at providing a machine learning-based approximation to the discrete solutions of differential problems on a specific space-time grid, for example, by solving linear or nonlinear systems efficiently and accelerating convergence of iterative schemes [25, 6, 26, 21, 17]. The second one provides instead solutions to the differential problem as continuous

(and differentiable) functions of the temporal and spatial variables. Depending on the context, conditions on such approximate solutions are provided by the differential problem itself, the initial values and boundary conditions, and the available data. The idea of providing approximate solutions as functions defined on the space-time domain and parametrised as neural networks was proposed in the nineties [24] and was recently revived in the framework of Physics-Informed Neural Networks in [40]. Since then, such an approach has attracted much interest and developed in many directions [8, 45, 23].

In this work, we use neural networks to approximate the configurations of highly flexible slender structures modelled as beams. Such models are of great interest in industrial applications like cable car ropes, diverse types of wires or endoscopes [37, 47, 31, 43]. Notwithstanding their ingenious and simple mathematical formulation, slender structure models can accurately reproduce complex mechanical behaviour and, for this reason, their numerical discretisation is often challenging. Furthermore, the use of 3-dimensional models requires high computational time. Due to the fact that slender deformable structures have one dimension (length) being orders of magnitude larger than their other dimensions (cross-section), it is possible to reduce the complexity of the problem from a 3-dimensional elastic continuum to a 1-dimensional beam. A beam is modelled as a centerline curve, $\mathbf{q} : [0, L] \to \mathbb{R}^n, s \mapsto \mathbf{q}(s)$, with $n = 2$ or $n = 3$, along which a rigid cross-section $\Sigma(s)$ is attached. The main model assumption is that the diameter of $\Sigma(s)$ is small compared with the undeformed length L. The complexity of the model depends on factors such as the dimension of the problem, the translational and rotational degrees of freedom (DOF) at each node of the beam, and the analysis, i.e., static or dynamic. Exploring the numerous beam models documented in the literature, we choose to approach the challenge of approximating beam deformations using a simple yet widely employed model, i.e., the 2-dimensional *Euler's elastica* [11]. The cross-section $\Sigma(s)$ is assumed to have unchanged geometrical and material properties and be orthogonal to the centerline $\mathbf{q}(s)$. The latter is an inextensible curve and solution of a bending energy minimisation problem [28, 33, 46] for given boundary conditions.

Although the 2-dimensional Euler's elastica is relatively simple compared to more comprehensive models, it can robustly represent interesting real-world phenomena. For instance, the elastica model appropriately captures the high bending deformations of flexible endoscopes, complex medical devices, during surgeries [47]. The approximation of the elastica through neural networks can help predict the deformed configuration of the beam for endoscopy simulations, particularly when the beam encounters constraints in confined spaces.

When approximating static equilibria of Euler's elastica via neural networks,

a key issue is to ensure the inextensibility of the curve (having unit norm tangents) as well as the boundary conditions. Two main approaches can be found in the literature [23, 45, 41]. One is the weak imposition of constraints and boundary conditions by adding appropriate extra terms to the loss function. The other is a strong imposition strategy consisting in shaping the network architectures to satisfy the constraints by construction. We show examples of both the approaches in Sections 7.4 and 7.5.

The paper is organised as follows. In Section 7.2, we present the mathematical model of the planar Euler's elastica, including its continuous and discrete equilibrium equations. We describe the approach used to generate the data sets for the numerical experiments. In Section 7.3, we introduce some basic theory and notation for neural networks that we shall use in the succeeding sections. Starting from general theory, we specialise in the task of approximating configurations of Euler's elastica. In Section 7.4, we introduce the *discrete* approach, which aims to approximate precomputed numerical discretisations of Euler's elastica. This represents the natural approach to approximate the discrete solution trajectories with a parametric method. We discuss some drawbacks associated with this approach and then propose an alternative approximation strategy in Section 7.5, that leverages the fact that we are approximating a continuous curve on a spatial grid. The *continuous* approach consists of computing an arc length parametrisation of the beam configuration. We provide insights into two additional networks and analyse how the test accuracy changes with varying constraints, such as boundary conditions or tangent vector norms. Data and codes for the numerical experiments are available in the GitHub repository associated with the paper[1].

Main contributions: This paper presents advancements in the approximation of beam static configurations using neural networks. These advancements include: (i) A detailed experimental analysis of approximating numerical discretisations of Euler's elastica configurations through what we call *discrete network*, (ii) Identification and discussion of the limitations associated with this discrete approach, and (iii) Introduction of a new parametrisation strategy called *continuous network* to address some of these drawbacks.

7.2 Euler's elastica model

We consider an inextensible beam model in which the cross-section $\Sigma(s)$ is assumed to be constant along the arc length s and perpendicular to the centerline $\mathbf{q}(s)$, which means that no shear deformation can occur. Thus, the deformation

[1] https://github.com/ergyscokaj/LearningEulersElastica

Nomenclature	
$\mathcal{L}$	continuous Lagrangian function
$\mathcal{S}$	continuous action functional
$\mathcal{L}_d$	discrete Lagrangian function
$\mathcal{S}_d$	discrete action functional
$\mathbf{q}$	configuration of the beam
$\mathbf{q}'$	first spatial derivative of $\mathbf{q}$
θ	tangential angle
s	arc length parameter
κ	curvature
L	length of the undeformed beam
EI	bending stiffness
$\hat{\mathbf{q}}$	numerical approximation of $\mathbf{q}$
$N+1$	number of discretisation nodes, with N the number of intervals
h	space step (length of each interval)
$q^{\mathrm{d}}_{\boldsymbol{\rho}}$	discrete neural network
$q^{\mathrm{c}}_{\boldsymbol{\rho}}$	continuous neural network approximating the curve $\boldsymbol{q}(s)$
$\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$	continuous neural network approximating the function $\theta(s)$
$\boldsymbol{\rho}$	parameters of the neural network
ℓ	number of layers in the neural network
σ	activation function
M	number of training data
B	size of one training batch
MSE	mean squared error
MLP	multi layer perceptron
MULT	multiplicative neural network
$\mathcal{D}$	differential operator
$\mathcal{I}$	quadrature operator

Table 7.1.1: List of abbreviations and notations.

of the centerline is a pure bending problem, precisely Euler's elastica curve. In the following, we assume $\mathbf{q} \in C^2([0, L], \mathbb{R}^2)$, i.e., the curve is planar and twice continuously differentiable with length L. If s denotes the arc length parameter, then $\|\mathbf{q}'(s)\| = 1$, where $' = \frac{d}{ds}$, for all $s \in [0, L]$. The elastica problem consists of minimising the following Euler-Bernoulli energy functional

$$\int_0^L \kappa(s)^2 \mathrm{d}s,$$

where $\kappa(s)$ denotes the curvature of $\mathbf{q}(s)$, [33]. Given the arc length parametrisation, then $\kappa(s) = \|\mathbf{q}''(s)\|$.

We can reformulate this problem as a constrained Lagrangian problem as follows. Consider the second-order Lagrangian $\mathcal{L} : T^{(2)}Q \to \mathbb{R}$, where $T^{(2)}Q$ denotes the second-order tangent bundle [7] of the configuration manifold Q,

which in this case is $\mathbb{R}^2$:

$$\mathcal{L}(\mathbf{q}, \mathbf{q}', \mathbf{q}'') = \frac{1}{2} EI \left\| \mathbf{q}'' \right\|^2 . \tag{7.2.1}$$

Here, abusing the notation, $'$ denotes a spatial derivative, but we do not initially assume arc length parametrisation. The parameter EI is the bending stiffness, which governs the response of the elastica under bending. This mechanical parameter consists of material and geometric properties, where E is Young's modulus and I is the second moment of area of the cross-section Σ. For simplicity, these parameters are assumed to be constant along the length of the beam.

In order to recover the solutions of the elastica, the Lagrangian in (7.2.1) must be supplemented with the constraint equation

$$\Phi(\mathbf{q}, \mathbf{q}') = \left\| \mathbf{q}' \right\|^2 - 1 = 0. \tag{7.2.2}$$

This imposes arc length parametrisation of the curve $\mathbf{q}(s)$ and leads to the augmented Lagrangian $\widetilde{\mathcal{L}} : T^{(2)}Q \times \mathbb{R} \to \mathbb{R}$

$$\widetilde{\mathcal{L}}(\mathbf{q}, \mathbf{q}', \mathbf{q}'', \Lambda) = \mathcal{L}(\mathbf{q}, \mathbf{q}', \mathbf{q}'') + \Lambda \Phi(\mathbf{q}, \mathbf{q}'), \tag{7.2.3}$$

where $\Lambda(s)$ is a Lagrange multiplier, see [46]. The Lagrangian function coincides with the total elastic energy over solutions of the corresponding Euler-Lagrange equations. The internal bending moment is directly related to the curvature $\kappa(s)$.

The continuous action functional $\mathcal{S}$ is defined as:

$$\mathcal{S}[\mathbf{q}] = \int_0^L \widetilde{\mathcal{L}}(\mathbf{q}, \mathbf{q}', \mathbf{q}'', \Lambda) \, \mathrm{d}s. \tag{7.2.4}$$

Applying Hamilton's principle of stationary action, $\delta \mathcal{S} = 0$, yields the Euler-Lagrange equations

$$\begin{aligned} \frac{\mathrm{d}^2}{\mathrm{d}s^2}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{q}''}\right) - \frac{\mathrm{d}}{\mathrm{d}s}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{q}'}\right) + \frac{\partial \mathcal{L}}{\partial \mathbf{q}} &= \frac{\mathrm{d}}{\mathrm{d}s}\left(\frac{\partial \Phi}{\partial \mathbf{q}'}\Lambda\right) - \frac{\partial \Phi}{\partial \mathbf{q}}\Lambda, \\ \left\| \mathbf{q}' \right\|^2 - 1 &= 0, \end{aligned} \tag{7.2.5}$$

which need to be satisfied together with the boundary conditions on positions and tangents, i.e., $(\mathbf{q}(0), \mathbf{q}'(0)) = (\mathbf{q}_0, \mathbf{q}_0')$ and $(\mathbf{q}(L), \mathbf{q}'(L)) = (\mathbf{q}_N, \mathbf{q}_N')$.

7.2.1 Space discretisation of the elastica

The continuous augmented Lagrangian $\widetilde{\mathcal{L}}$ in (7.2.3) and the action integral $\mathcal{S}$ in (7.2.4) are discretised over the beam length L using constant step size $h =$

L/N, with $N+1$ the number of the resulting equidistant nodes $0 = s_0 < s_1 < \ldots < s_{N-1} < s_N = L$. In second-order systems, the discrete Lagrangian is a function $\widetilde{\mathcal{L}}_d : TQ \times TQ \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. In this study, we refer to a discretisation of the Lagrangian function proposed in [14] based on the trapezoidal rule:

$$\begin{aligned}&\widetilde{\mathcal{L}}_d\left(\mathbf{q}_k, \mathbf{q}'_k, \mathbf{q}_{k+1}, \mathbf{q}'_{k+1}, \Lambda_k, \Lambda_{k+1}\right)\\&= \frac{h}{2}\left[\widetilde{\mathcal{L}}\left(\mathbf{q}_k, \mathbf{q}'_k, \left(\mathbf{q}''_k\right)^-, \Lambda_k\right) + \widetilde{\mathcal{L}}\left(\mathbf{q}_{k+1}, \mathbf{q}'_{k+1}, \left(\mathbf{q}''_{k+1}\right)^+, \Lambda_{k+1}\right)\right],\end{aligned}$$

where $\mathbf{q}_k$, $\mathbf{q}'_k$, and Λ_k are approximations of $\mathbf{q}(s_k)$, $\mathbf{q}'(s_k)$, and $\Lambda(s_k)$, and the curvature on the interval $[s_k, s_{k+1}]$ is approximated in terms of lower order derivatives as follows

$$\mathbf{q}''(s_k) \approx \left(\mathbf{q}''_k\right)^- = \frac{\left(-2\mathbf{q}'_{k+1} - 4\mathbf{q}'_k\right)h + 6\left(\mathbf{q}_{k+1} - \mathbf{q}_k\right)}{h^2},$$

$$\mathbf{q}''(s_{k+1}) \approx \left(\mathbf{q}''_{k+1}\right)^+ = \frac{\left(4\mathbf{q}'_{k+1} + 2\mathbf{q}'_k\right)h - 6\left(\mathbf{q}_{k+1} - \mathbf{q}_k\right)}{h^2}.$$

This amounts to a piece-wise linear and discontinuous approximation of the curvature on $[0, L]$.

The action integral in (7.2.4) along the exact solution $\mathbf{q}$ with boundary conditions $\left(\mathbf{q}_0, \mathbf{q}'_0\right)$ and $\left(\mathbf{q}_N, \mathbf{q}'_N\right)$ is approximated by

$$\mathcal{S}_d = \sum_{k=0}^{N-1} \widetilde{\mathcal{L}}_d\left(\mathbf{q}_k, \mathbf{q}'_k, \mathbf{q}_{k+1}, \mathbf{q}'_{k+1}, \Lambda_k, \Lambda_{k+1}\right). \tag{7.2.6}$$

The discrete variational principle $\delta\mathcal{S}_d = 0$ leads to the following discrete Euler-Lagrange equations:

$$\begin{aligned}&D_3\widetilde{\mathcal{L}}_d\left(\mathbf{q}_{k-1}, \mathbf{q}'_{k-1}, \mathbf{q}_k, \mathbf{q}'_k, \Lambda_{k-1}, \Lambda_k\right) + D_1\widetilde{\mathcal{L}}_d\left(\mathbf{q}_k, \mathbf{q}'_k, \mathbf{q}_{k+1}, \mathbf{q}'_{k+1}, \Lambda_k, \Lambda_{k+1}\right) = 0,\\&D_4\widetilde{\mathcal{L}}_d\left(\mathbf{q}_{k-1}, \mathbf{q}'_{k-1}, \mathbf{q}_k, \mathbf{q}'_k, \Lambda_{k-1}, \Lambda_k\right) + D_2\widetilde{\mathcal{L}}_d\left(\mathbf{q}_k, \mathbf{q}'_k, \mathbf{q}_{k+1}, \mathbf{q}'_{k+1}, \Lambda_k, \Lambda_{k+1}\right) = 0,\\&D_6\widetilde{\mathcal{L}}_d\left(\mathbf{q}_{k-1}, \mathbf{q}'_{k-1}, \mathbf{q}_k, \mathbf{q}'_k, \Lambda_{k-1}, \Lambda_k\right) + D_5\widetilde{\mathcal{L}}_d\left(\mathbf{q}_k, \mathbf{q}'_k, \mathbf{q}_{k+1}, \mathbf{q}'_{k+1}, \Lambda_k, \Lambda_{k+1}\right) = 0,\end{aligned} \tag{7.2.7}$$

for $k = 1, \ldots, N-1$, which approximate the equilibrium equations of the beam in (7.2.5) and can be solved together with the boundary conditions. Here, D_i for $i = 1, \ldots, 6$ denotes the differentiation with respect to the i-th argument.

7.2.2 Data generation

The elastica was one of the first examples displaying elastic instability and bifurcation phenomena [48, 2]. Elastic instability implies that small pertur-

bations of the boundary conditions might lead to large changes in the beam configuration, which results in unstable equilibria. Under certain boundary conditions, bifurcation can appear, leading to a multiplicity of solutions [33]. In particular, this means that the numerical problem may display history dependence and converge to solutions that do not minimise the bending energy. In order to generate a physically meaningful data set, avoiding unstable and non-unique solutions is essential. Thus, in addition to the minimisation of the discrete action S_d in (7.2.6), we ensure the fulfilment of the discrete Euler-Lagrange equations (7.2.7), which can be seen as necessary conditions for the stationarity of the discrete action. We exclude from the data set numerical solutions computed with boundary conditions where minimisation of (7.2.6) and accurate solution of (7.2.7) can not be simultaneously achieved.

In particular, we consider a curve of length $L = 3.3$ and bending stiffness $EI = 10$, divided into $N = 50$ intervals. We fix the endpoints $\mathbf{q}_0 = (0,0)$, $\mathbf{q}_N = (3,0)$. The units of measurement are deliberately omitted as they have no impact on the results of this work. We impose boundary conditions on the tangents in the following two variants:

1. the angle of the tangents with respect to the x-axis at the boundary, θ_0 and θ_N, is prescribed in the range $[0, 2\pi]$, in a specular symmetric fashion, i.e., $\theta_N = \pi - \theta_0$. Hereafter, we refer to this case as *both-ends*,

2. the angle of the left tangent is left fixed as $\theta_0 = 0$, and the angle of the right tangent, θ_N, varies in the range of $[0, 2\pi]$. We refer to this case as *right-end*.

Based on these parameters and boundary values, and using cubic splines as initial guess, we generate a data set of 2000 trajectories (1000 trajectories for each case) by minimising the particular action in (7.2.6), with the `trust-constr` solver of the `optimize.minimize` procedure provided in `SciPy` [49]. We check the resulting solutions by using them as initial guesses for the `root` method of `SciPy.optimize`, solving the discrete Euler-Lagrange equations (7.2.7).

The learning problem we consider relies on numerically generated solution curves. This choice allows us to work with data points that are quantifiably close to the analytical solution of Euler's elastica. Consequently, showing that the neural networks we propose can accurately approximate these curves translates into their ability to approximate the analytical solution accurately. The motivation of this strategy is not to improve on the numerical solver we use but to use its accuracy to train a model that is able to extrapolate to unseen boundary conditions and generate their solution curves more efficiently than

the numerical method itself. The chosen supervised learning setting is independent of the fact that we use numerical solutions as data. Indeed, if one had another reliable approximation of the analytical solution, for example, based on realistic measurements, those could also be used or combined with numerically generated trajectories. Using numerical solutions as data is not an inherent limitation of the proposed procedure but a choice we make to quotient out the issue of not having reliable input data. Furthermore, we mainly focus on the development of neural networks able to approximate such input data with high accuracy.

7.3 Approximation with neural networks

We start by providing a concise overview of neural networks, which also serves to define the notation used in Sections 7.4 and 7.5. We refer to [20, 23, 16] and references therein for a more extensive introduction. A neural network is a parametric function $f_{\boldsymbol{\rho}} : \mathcal{I} \to \mathcal{O}$ with parameters $\boldsymbol{\rho} \in \Psi$ given as a composition of multiple transformations,

$$f_{\boldsymbol{\rho}} := f_\ell \circ \cdots \circ f_j \circ \cdots \circ f_1, \tag{7.3.1}$$

where each f_j represents the j-th layer of the network, with $j = 1, \dots, \ell$, and ℓ is the number of layers. For example, multi-layer perceptrons (MLPs) have each layer f_j defined as

$$f_j^{\text{MLP}}(\mathbf{x}) = \sigma\left(\mathbf{A}_j \mathbf{x} + \mathbf{b}_j\right) \in \mathbb{R}^{n_j}, \tag{7.3.2}$$

where n_j is the dimension of the output of the j-th layer, $\mathbf{x} \in \mathbb{R}^{n_{j-1}}$, and $\mathbf{A}_j \in \mathbb{R}^{n_j \times n_{j-1}}$, $\mathbf{b}_j \in \mathbb{R}^{n_j}$ are the parameters of the j-th layer, i.e., $\boldsymbol{\rho} = \{\mathbf{A}_j, \mathbf{b}_j\}_{j=1}^{\ell}$. The activation function σ is a continuous nonlinear scalar function, which acts component-wise on vectors. The architecture of the neural network is prescribed by the layers f_j in (7.3.1) and determines the space of functions $\mathcal{F} = \{f_{\boldsymbol{\rho}} : \mathcal{I} \to \mathcal{O},\ \boldsymbol{\rho} \in \Psi\}$ that can be represented. The weights $\boldsymbol{\rho}$ are chosen such that $f_{\boldsymbol{\rho}}$ approximates accurately enough a map of interest $f : \mathcal{I} \to \mathcal{O}$. Usually, this choice follows from minimising a purposely designed loss function $\text{Loss}(\boldsymbol{\rho})$.

In supervised learning, we are given a data set $\Omega = \{\mathbf{x}^i, \mathbf{y}^i\}_{i=1}^M$ consisting of M pairs $\left(\mathbf{x}^i, \mathbf{y}^i = f\left(\mathbf{x}^i\right)\right)$. The loss function measures the distance between the network predictions $f_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)$ and the desired outputs $\mathbf{y}^i$ in some appropriate norm $\|\cdot\|$,

$$\text{Loss}\left(\boldsymbol{\rho}\right) = \frac{1}{M} \sum_{i=1}^{M} \left\| f_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right) - \mathbf{y}^i \right\|^2.$$

The training of the network is the process of minimising $\text{Loss}(\boldsymbol{\rho})$ with respect to $\boldsymbol{\rho}$ and it is usually done with gradient descent (GD):

$$\boldsymbol{\rho}^{(k)} \mapsto \boldsymbol{\rho}^{(k)} - \eta \nabla \text{Loss}\left(\boldsymbol{\rho}^{(k)}\right) =: \boldsymbol{\rho}^{(k+1)}.$$

The scalar value η is known as the learning rate. The iteration process is often implemented using subsets of data $\mathcal{B} \subset \Omega$ of cardinality $B = |\mathcal{B}|$ (batches). In this paper, we use an accelerated version of GD known as Adam [22].

During training, we evaluate the model's prediction accuracy using inputs in a validation set. This helps to prevent overfitting on the training data and may serve as a stopping criterion if the training loss diminishes, but the validation error rises. Once the training is complete, we assess the model's accuracy in predicting the correct output for new inputs included in a test set composed of boundary conditions outside the training and validation sets. In the following, we measure the accuracy on the training, validation, and test data using the mean squared error of the difference between the predicted trajectories and the true ones.

We now turn to the task of approximating the static equilibria of the planar elastica introduced in Section 7.2, i.e., approximating a family of curves $\{\mathbf{q}^i : [0, L] \mapsto \mathbb{R}^2\}$ determined by boundary conditions,

$$\left\{\mathbf{q}^i(0) = \mathbf{q}_0^i,\ \mathbf{q}^i(L) = \mathbf{q}_N^i,\ \left(\mathbf{q}^i\right)'(0) = \left(\mathbf{q}_0^i\right)',\ \left(\mathbf{q}^i\right)'(L) = \left(\mathbf{q}_N^i\right)'\right\}, \tag{7.3.3}$$

where $\left(\mathbf{q}_0^i, \mathbf{q}_N^i, \left(\mathbf{q}_0^i\right)', \left(\mathbf{q}_N^i\right)'\right) \in \mathbb{R}^8$. To tackle this problem, we require a set of evaluations $\{\mathbf{q}_k^i, (\mathbf{q}_k^i)'\}$ on the nodes $s_k \in [0, L]$ of a discretisation. More precisely, in our setting, the data set includes numerical approximations $\hat{\mathbf{q}}$ of the solution $\mathbf{q}(s)$ and its spatial derivative $\mathbf{q}'(s)$ at the $N-1$ discrete locations $s_k = \frac{kh}{L}$ in the interval $[0, L]$, for M pairs of boundary conditions, as described in Section 7.2.2.

7.4 The discrete network

The discretisation of Euler's elastica presented in Section 7.2.1 provides discrete solutions on a set of nodes along the curve. These solutions can sometimes be hard to obtain since a global optimisation problem needs to be solved, and the number of nodes can be large. This motivates using neural networks to learn the approximate solution on the internal nodes for a given set of boundary conditions. The data set Ω consists of M precomputed discrete solutions

$$\Omega = \left\{\left(\mathbf{x}^i, \mathbf{y}^i\right)\right\}_{i=1}^{M},$$

where

$$\mathbf{x}^i = \left(\mathbf{q}_0^i, \left(\mathbf{q}_0^i\right)', \mathbf{q}_N^i, \left(\mathbf{q}_N^i\right)'\right) \in \mathbb{R}^8$$

are the input boundary conditions and

$$\mathbf{y}^i = \left(\hat{\mathbf{q}}_1^i, \left(\hat{\mathbf{q}}_1^i\right)', \ldots, \hat{\mathbf{q}}_{N-1}^i, \left(\hat{\mathbf{q}}_{N-1}^i\right)'\right) \in \mathbb{R}^{4(N-1)}$$

are the computed solutions at the internal nodes that serve as output data for the network's training.

For any symmetric positive definite matrix W, we define the weighted norm $\|\mathbf{x}\|_W^2 = \mathbf{x}^\top W \mathbf{x}$. The weighted MSE loss

$$\text{Loss}(\boldsymbol{\rho}) = \frac{1}{4M(N-1)} \sum_{i=1}^{M} \left\| q_{\boldsymbol{\rho}}^{\mathrm{d}}\left(\mathbf{x}^i\right) - \mathbf{y}^i \right\|_W^2 \tag{7.4.1}$$

will be used to learn the input-to-output map $q_{\boldsymbol{\rho}}^{\mathrm{d}} : \mathbb{R}^8 \to \mathbb{R}^{4(N-1)}$, where the superscript d stands for discrete. One should be aware that there is a numerical error in $\mathbf{y}^i$ compared to the exact solution, and the size of this error will pose a limit to the accuracy of the neural network approximation.

7.4.1 Numerical experiments

This section provides experimental support to the proposed learning framework using the machine learning library `PyTorch` [38]. The experiments of this section are run on a CPU machine. We perform a series of experiments varying some hyperparameters in the training procedure. We fix the batch size B to 32 and use the Adam optimiser [22] for the training with learning rate 10^{-3} and weight decay set to 0. In (7.4.1) we use the weight matrix

$$W = I + \gamma G^\top G,$$

where $G = S^4 - I$ with S the forward shift operator on vectors of $\mathbb{R}^{4(N-1)}$. This choice of G allows us to compute differences between corresponding entries of the input associated with neighbouring nodes. We determine the number of epochs for training both the discrete and continuous networks based on experimental evidence. We fix a high enough number which allows us to achieve qualitatively accurate predictions and ensure that both training and validation losses start to plateau a few epochs before the set maximum. We consider a multi-layer perceptron with the hyperbolic tangent as an activation function, and we vary the number of layers and the number of hidden nodes in each layer. We also test different values of the parameter γ in the weight matrix

W. We rely on the software framework `Optuna` [1], which employs Bayesian optimisation methods to automate and efficiently conduct the search for the combination that yields the best result. We collect in Table 7.B.1 the hyperparameters with the corresponding ranges and in Table 7.B.2 the selected values. The resulting training error on the *both-end* data set is $1.14 \cdot 10^{-7}$, the validation error is $2.151 \cdot 10^{-7}$, and the test error is $4.009 \cdot 10^{-7}$. Figure 7.4.1 compares test trajectories for $\mathbf{q}$ and $\mathbf{q}'$. We remark that, as already clear from the low value of the training and test errors, the network can accurately replicate the behaviour of the training and test data. Furthermore, we have zero errors at the end nodes since the network is trained only on the internal nodes, and the boundary values are appended to the predicted solution in a post-processing phase. On the other hand, since this discrete approach does not relate the components as evaluations of a smooth curve, there is no regular behaviour in the error.

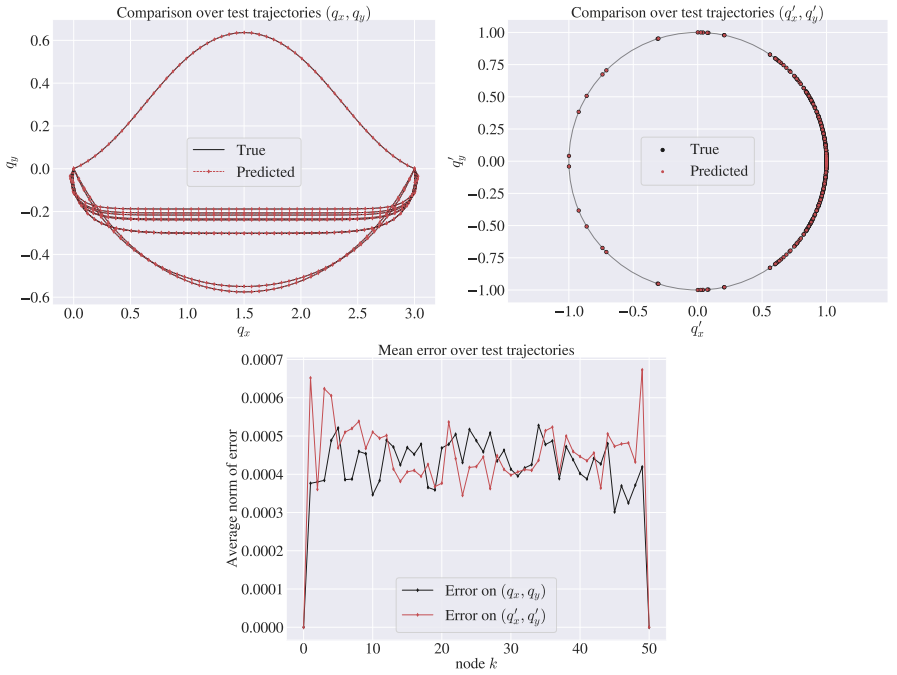


Figure 7.4.1: Comparison over test trajectories for $\mathbf{q}$ and $\mathbf{q}'$ for the discrete network $q^{\mathrm{d}}_{\boldsymbol{\rho}}$ tested on the *both-ends* data set with 80% – 10% – 10% splitting into training, validation, and test sets. The mean squared error on the test set equals $4.009 \cdot 10^{-7}$. For presentation purposes, only ten randomly selected trajectories are considered in the first two plots.

As an additional evaluation of the deep learning framework's behaviour, we conduct experiments to assess how the learning process performs when the

number of training data varies, i.e., with different splittings of the data set into training, validation, and test sets. We report the results in Table 7.4.1 and summarise the corresponding hyperparameters in Table 7.B.2 of the Appendix.

Data set splitting Training - validation - test	Training accuracy	Validation accuracy	Test accuracy
10% - 10% - 10%	$2.331 \cdot 10^{-5}$	$3.874 \cdot 10^{-5}$	$8.545 \cdot 10^{-4}$
20% - 10% - 10%	$1.852 \cdot 10^{-6}$	$1.327 \cdot 10^{-6}$	$1.361 \cdot 10^{-4}$
40% - 10% - 10%	$4.802 \cdot 10^{-7}$	$4.793 \cdot 10^{-7}$	$1.295 \cdot 10^{-6}$
80% - 10% - 10%	$1.140 \cdot 10^{-7}$	$2.151 \cdot 10^{-7}$	$4.009 \cdot 10^{-7}$

Table 7.4.1: Behaviour of the discrete network $q^{\mathrm{d}}_{\boldsymbol{\rho}}$ tested on the *both-ends* data set with fewer training data points. The size of the training set varies, while that of the validation and the test sets is fixed. The last row corresponds to the results in Figure 7.4.1.

We also report results obtained by merging the *both-end* and the *right-end* trajectories, with $80\% - 10\% - 10\%$ splitting of the whole new data set into training, validation, and test sets. The results are shown in Figure 7.4.2 and are obtained with 3 layers, 616 hidden nodes, and $\gamma = 7.323 \cdot 10^{-3}$. The resulting training, validation, and test errors are, respectively, $9.893 \cdot 10^{-8}$, $1.126 \cdot 10^{-7}$, and $7.854 \cdot 10^{-8}$.

7.5 The continuous network

The approach described in the previous section shows accurate results, given a large enough amount of beam discretisations with a fixed number of nodes $N+1$, equally distributed in $[0, L]$. It seems reasonable to expect the parametric model's approximation quality to improve when the number of discretisation nodes increases. However, in this approach, the dimension of the predicted vector grows with N, and hence minimising the loss function (7.4.1) becomes more difficult. In addition, the fact that the discrete network approach depends on the spatial discretisation of the training data restricts the output dimension to a specific number of nodes. Consequently, there would be two main options to assess the solution at different locations: training the network once more, or interpolating the previously obtained approximation. These limitations make such a discrete approach less appealing and suggest that having a neural network that is a smooth function of the arc length coordinate s can be beneficial. This modelling assumption would also be compatible with different discretisations of the curve and would not suffer from the curse of dimensionality if more nodes were added. In this setting, the discrete node s_k at which an ap-

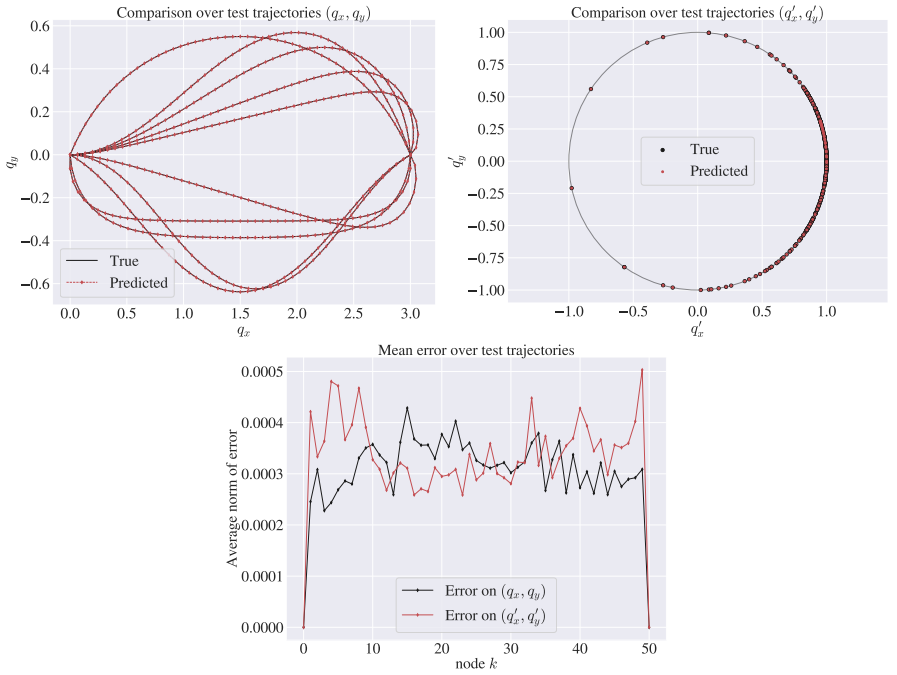


Figure 7.4.2: Comparison over test trajectories for $\mathbf{q}$ and $\mathbf{q}'$ for the discrete network $q^{\mathrm{d}}_{\boldsymbol{\rho}}$ tested on the *both-ends* + *right-end* data set with $80\% - 10\% - 10\%$ splitting into training, validation, and test sets. The mean squared error on the test set equals $7.854 \cdot 10^{-8}$. For presentation purposes, only ten randomly selected trajectories are considered in the first two plots.

proximation of the solution is available, is included in the input data together with the boundary conditions. As a result, we work with the following data set

$$\Omega = \left\{ \left(s_k, \mathbf{x}^i \right), \mathbf{y}^i_k \right\}^{i=1,\dots,M}_{k=0,\dots,N},$$

where, as in the previous section,

$$\mathbf{x}^i = \left(\mathbf{q}^i_0, \left(\mathbf{q}^i_0 \right)', \mathbf{q}^i_N, \left(\mathbf{q}^i_N \right)' \right) \in \mathbb{R}^8,$$

and

$$\mathbf{y}^i_k = \left(\hat{\mathbf{q}}^i_k, \left(\hat{\mathbf{q}}^i_k \right)' \right).$$

Here $\hat{\mathbf{q}}^i_k$ is the numerical solution $\hat{\mathbf{q}}$ on the node s_k, satisfying the i-th boundary conditions in (7.3.3). Let us introduce the neural network

$$q^{\mathrm{c}}_{\boldsymbol{\rho}} : \mathbb{R}^8 \to \mathcal{C}^\infty \left(\left[0, L \right], \mathbb{R}^2 \right),$$

and the differential operator

$$\mathcal{D}:\mathcal{C}^\infty\left([0,L],\mathbb{R}^2\right)\to\mathcal{C}^\infty\left([0,L],\mathbb{R}^2\right),\ \mathcal{D}\left(q^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)\right)(s_k)=\frac{d}{ds}\left(q^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)\right)(s)\Big|_{s=s_k},$$

so that we can define

$$y_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s_k):=\left(q^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s_k),\mathcal{D}\left(q^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)\right)(s_k)\right).$$

To train the network $q^{\mathrm{c}}_{\boldsymbol{\rho}}$, we define the loss function

$$\begin{aligned}\mathrm{Loss}(\boldsymbol{\rho})=\frac{1}{4M(N+1)}\sum_{i=1}^{M}\sum_{k=0}^{N}\Bigg(&\left\|y_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s_k)-\boldsymbol{y}^i_k\right\|_2^2\\&+\gamma\left(\left(\left\|\pi_{\mathcal{D}}\left(y_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s_k)\right)\right\|_2^2-1\right)^2\right)\Bigg),\end{aligned}\tag{7.5.1}$$

where $\pi_{\mathcal{D}}:\mathbb{R}^8\to\mathbb{R}^4$ is the projection on the second component $\mathcal{D}(q^{\mathrm{c}}_{\boldsymbol{\rho}}(\mathbf{x}^i))(s_k)$, and $\gamma\geq 0$ weighs the violation of the normality constraint. The map $q^{\mathrm{c}}_{\boldsymbol{\rho}}$ is now a neural network that associates each set of boundary conditions $\mathbf{x}^i$ with a smooth curve $q^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right):[0,L]\to\mathbb{R}^2$ that can be evaluated at every point $s\in[0,L]$. We denote this network with the superscript c since this curve is, in particular, continuous. The outputs $q^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s)\in\mathbb{R}^2$ are approximations of the configuration of the beam at $s\in[0,L]$.

We point out that, contrary to the discrete case, we learn approximations of $\mathbf{q}(s)$ also on the end nodes, i.e., at $s=0$ and $s=L$. This is because we do not impose the boundary conditions by construction. Even though there are multiple approaches to embed them into the network architecture, the one we try in our experiments made the optimisation problem too complex, thus we only impose the boundary conditions weakly in the loss function.

Another strategy is to compute the angles θ_k between the tangents $(\hat{\mathbf{q}}_k)'$ and the x-axis and to use them as training data. To this end, we define the neural network

$$\theta^{\mathrm{c}}_{\boldsymbol{\rho}}:\mathbb{R}^8\to\mathcal{C}^\infty\left([0,L],\mathbb{R}\right)$$

as $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}=\hat{\theta}^{\mathrm{c}}_{\boldsymbol{\rho}}\circ\pi$, where

$$\hat{\theta}^{\mathrm{c}}_{\boldsymbol{\rho}}:\mathbb{R}^2\to\mathcal{C}^\infty\left([0,L],\mathbb{R}\right)\tag{7.5.2}$$

is a neural network, and the function $\pi:\mathbb{R}^8\to\mathbb{R}^2$ extracts the tangential angles from the boundary conditions, i.e., $\pi\left(\mathbf{x}^i\right)=\left(\theta^i_0,\theta^i_N\right)$. Such a network should

approximate the angular function $\theta : [0, L] \ni s \to \mathbb{R}$, so that

$$\tau^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s) := \left(\cos\left(\theta^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s)\right), \sin\left(\theta^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s)\right)\right) \in \mathbb{R}^2 \tag{7.5.3}$$

gets close to the tangent vector $\mathbf{q}'(s)$. As a result, the constraint on the unit norm of the tangents is satisfied by construction, and the inextensibility of the elastica is guaranteed. The curve

$$\mathbf{q}(s) = \mathbf{q}_0 + \int_0^s \mathbf{q}'(\bar{s})\mathrm{d}\bar{s}$$

can then be approximated through the reconstruction formula

$$q^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s) = \mathbf{q}_0 + \mathcal{I}\left(\tau^{\mathrm{c}}_{\rho}\left(\mathbf{x}^i\right)\right)(s), \tag{7.5.4}$$

where the operator $\mathcal{I} : \mathcal{C}^\infty\left([0, L], \mathbb{R}^2\right) \to \mathcal{C}^\infty\left([0, L], \mathbb{R}^2\right)$ is such that

$$\mathcal{I}\left(\tau^{\mathrm{c}}_{\rho}\left(\mathbf{x}^i\right)\right)(s) \approx \int_0^s \tau^{\mathrm{c}}_{\rho}\left(\mathbf{x}^i\right)(\bar{s})\,\mathrm{d}\bar{s}.$$

In the numerical experiments, $\mathcal{I}$ is based on the 3-point Gaussian quadrature formula applied to a partition of the interval $[0, L]$, see [39, Chapter 9]. As done previously, we define the vector

$$y_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s_k) := \left(q^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s_k), \tau^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s_k)\right), \tag{7.5.5}$$

with components defined as in (7.5.3) and (7.5.4). This allows us to train the network $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$ by minimising the same loss function as in (7.5.1), where this time $y^{\mathrm{c}}_{\boldsymbol{\rho}}$ is given by (7.5.5). Furthermore, since by construction this case satisfies $\left\|\pi_{\mathcal{D}}\left(y^{\mathrm{c}}_{\boldsymbol{\rho}}(\mathbf{x}^i)(s)\right)\right\|_2 = \left\|\tau^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s)\right\|_2 \equiv 1$, we set $\gamma = 0$. We present numerical experiments for the two proposed continuous networks $q^{\mathrm{c}}_{\boldsymbol{\rho}}$ and $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$. In the latter case, by neural network architecture, we refer to $\hat{\theta}^{\mathrm{c}}_{\boldsymbol{\rho}}$ rather than $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$ in what follows. We analyse $q^{\mathrm{c}}_{\boldsymbol{\rho}}$ more thoroughly in Section 7.5.1, mirroring most of the discrete case experiments. In Section 7.5.2, we study how the results are affected when we impose the arc length parametrisation and enforce the boundary conditions to be exactly satisfied by the network $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$.

7.5.1 Numerical experiments with $q^{\mathrm{c}}_{\boldsymbol{\rho}}$

As for the case of the discrete network, we perform an in-depth investigation of this learning setting. In this case, the experiments are run on a GPU-P100 machine. For this continuous setup, the standard MLP architecture does not provide accurate results even after a hyperparameter optimisation routine. Given

that the simple MLP architecture does not seem to be flexible enough to capture the complexity of the elastica solution in this continuous framework, we move to a different architecture that we call MULT for the presence of multiplicative interactions in its architecture. This network has demonstrated superior performance to standard fully connected neural networks in the context of operator learning, see e.g. [50]. Details on this architecture can be found in Appendix 7.A. We fix the learning rate η to $5{\cdot}10^{-3}$ and only vary the number of layers and of hidden nodes in the training procedure, with the range of options reported in Appendix 7.B, Table 7.B.3. In this case, we define the loss as in (7.5.1), with $\gamma = 10^{-2}$. The weight decay is systematically set to 0. For the *both-ends* data set, this leads to a training error equal to $3.554 \cdot 10^{-6}$, a validation error equal to $4.779 \cdot 10^{-6}$, and a test error equal to $4.354 \cdot 10^{-6}$. In Figure 7.5.1, the comparison over test trajectories for $\mathbf{q}$ and $\mathbf{q}'$ is shown. As we can see in the plot showing the mean error over the trajectories, the error on the end nodes is nonzero since we are not imposing boundary conditions by construction. This is in contrast to the corresponding plot for the discrete network in Figure 7.4.1.

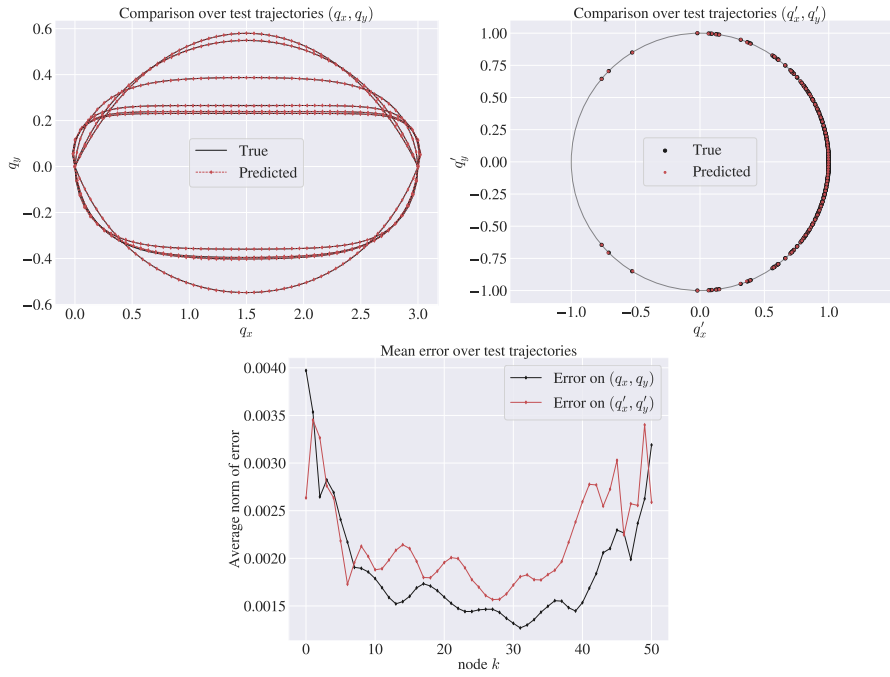


Figure 7.5.1: Comparison over test trajectories for $\mathbf{q}$ and $\mathbf{q}'$ for the continuous network $q^{\mathrm{c}}_{\boldsymbol{\rho}}$ tested on the *both-ends* data set with 80% – 10% – 10% splitting into training, validation, and test sets. The mean squared error on the test set equals $4.354 \cdot 10^{-6}$. For presentation purposes, only ten randomly selected trajectories are considered in the first two plots.

Also in this case, we examine the behaviour of the learning process with different splittings of the data set into training and test sets. We display the results in Table 7.5.1 and summarise the corresponding hyperparameters in Appendix 7.B, Table 7.B.4.

Data set splitting Training - validation - test	Training accuracy	Validation accuracy	Test accuracy
10% - 10% - 10%	$2.146 \cdot 10^{-4}$	$1.252 \cdot 10^{-3}$	$8.811 \cdot 10^{-4}$
20% - 10% - 10%	$4.187 \cdot 10^{-5}$	$4.239 \cdot 10^{-5}$	$6.279 \cdot 10^{-5}$
40% - 10% - 10%	$7.037 \cdot 10^{-6}$	$8.357 \cdot 10^{-6}$	$8.434 \cdot 10^{-6}$
80% - 10% - 10%	$3.554 \cdot 10^{-6}$	$4.779 \cdot 10^{-6}$	$4.354 \cdot 10^{-6}$

Table 7.5.1: Behaviour of the continuous network $q^{\mathrm{c}}_{\boldsymbol{\rho}}$ tested on the *both-ends* data set with fewer training data points. The size of the training set varies, while that of the validation and the test sets is fixed. The last row corresponds to the results in Figure 7.5.1.

7.5.2 Numerical experiments with $\theta^{\mathbf{c}}_{\boldsymbol{\rho}}$

Here we consider a neural network approximation of the angle $\theta(s)$ that parametrises the tangent vector $\mathbf{q}'(s) = \big(\cos\theta(s), \sin\theta(s)\big)$. By design, the approximation $\tau^{\mathrm{c}}_{\boldsymbol{\rho}}$ of the tangent vector $\mathbf{q}'$ satisfies the constraint $\|\tau^{\mathrm{c}}_{\boldsymbol{\rho}}(\mathbf{x}^i)(s)\|_2 = 1$ for every $s \in [0, L]$ and $\mathbf{x}^i \in \mathbb{R}^8$. We also analyse how the neural network approximation behaves when the boundary conditions $\tau^{\mathrm{c}}_{\boldsymbol{\rho}}(\mathbf{x}^i)(0) = \mathbf{q}'(0)$ and $\tau^{\mathrm{c}}_{\boldsymbol{\rho}}(\mathbf{x}^i)(L) = \mathbf{q}'(L)$ are imposed by construction. To do so, we model the parametric function $\hat{\theta}^{\mathrm{c}}_{\boldsymbol{\rho}}$, defined in (7.5.2), in one of the two following ways:

$$\hat{\theta}^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s) = f_{\boldsymbol{\rho}}\left(s, \theta^i_0, \theta^i_N\right), \tag{7.5.6}$$

$$\begin{aligned}\hat{\theta}^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s) = f_{\boldsymbol{\rho}}\left(s, \theta^i_0, \theta^i_N\right) + \left(\theta^i_0 - f_{\boldsymbol{\rho}}\left(0, \theta^i_0, \theta^i_N\right)\right) e^{-100 s^2} \\ + \left(\theta^i_N - f_{\boldsymbol{\rho}}\left(L, \theta^i_0, \theta^i_N\right)\right) e^{-100 (s-L)^2},\end{aligned} \tag{7.5.7}$$

where $f_{\boldsymbol{\rho}} : \mathbb{R}^3 \to \mathbb{R}$ is any neural network, and we recall that $\pi(\mathbf{x}^i) = (\theta^i_0, \theta^i_N)$. We remark that in the case of the parameterisation in (7.5.7), one gets $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}(\mathbf{x}^i)(0) = \theta^i_0$ and $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}(\mathbf{x}^i)(L) = \theta^i_N$ up to machine precision, due to the fast decay of the Gaussian function. As in the previous sections, we collect the hyperparameter and architecture options with the respective range of choices in Table 7.B.5, and we report the results without imposing the boundary conditions in Figure 7.5.2, while those imposing them in Figure 7.5.3, in both

cases using the *both-ends* data set, with 80% – 10% – 10% splitting into training, validation, and test sets. The results shown in the two figures correspond respectively to training errors of $6.288 \cdot 10^{-6}$ and $5.301 \cdot 10^{-6}$, validation errors $5.874 \cdot 10^{-6}$ and $5.065 \cdot 10^{-6}$, and test errors of $5.089 \cdot 10^{-6}$ and $4.385 \cdot 10^{-6}$. The best-performing hyperparameter combinations can be found in Table 7.B.6.

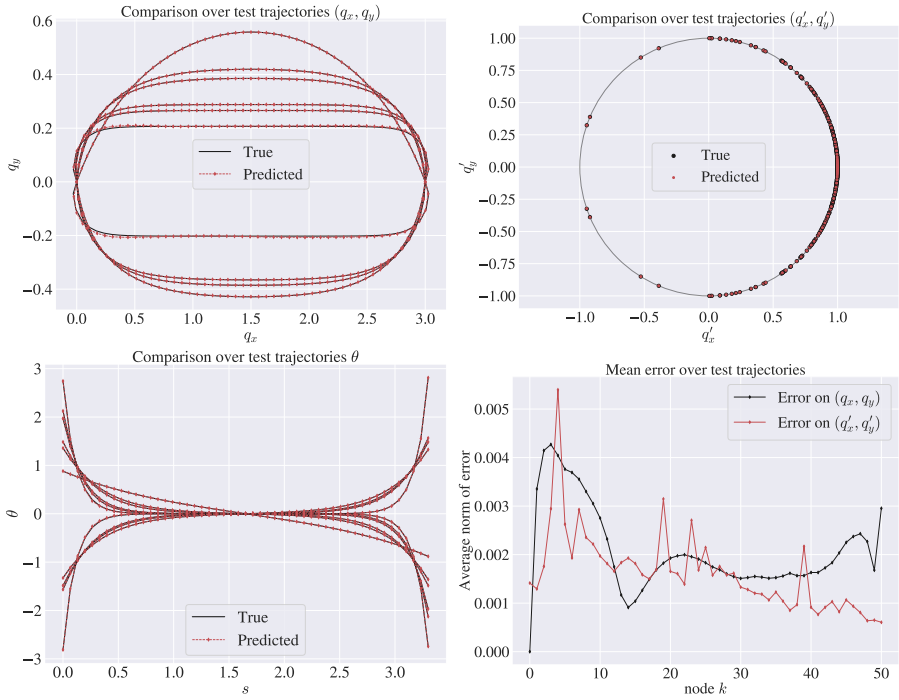


Figure 7.5.2: Comparison over test trajectories for $\mathbf{q}$ and $\mathbf{q}'$, for the case $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$ is modelled as in (7.5.6), with 80% – 10% – 10% splitting of the *both-ends* data set into training, validation, and test sets. The mean squared error on the test set equals $6.289 \cdot 10^{-6}$. For presentation purposes, only ten randomly selected trajectories are considered in the first two plots.

7.6 Discussion

The results in Figures 7.5.2 and 7.5.3 are comparable, especially looking at the mean error plots. This suggests that the imposition of the boundary conditions, in the proposed way, is not limiting the expressivity of the considered network. Thus, given the boundary value nature of our problem, these figures advocate the enforcement of the boundary conditions on the network $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$. However, due to the chosen reconstruction procedure in (7.5.4) for the variable $\mathbf{q}$, we are able to impose the boundary conditions on $\mathbf{q}$ only on the left node. Other more

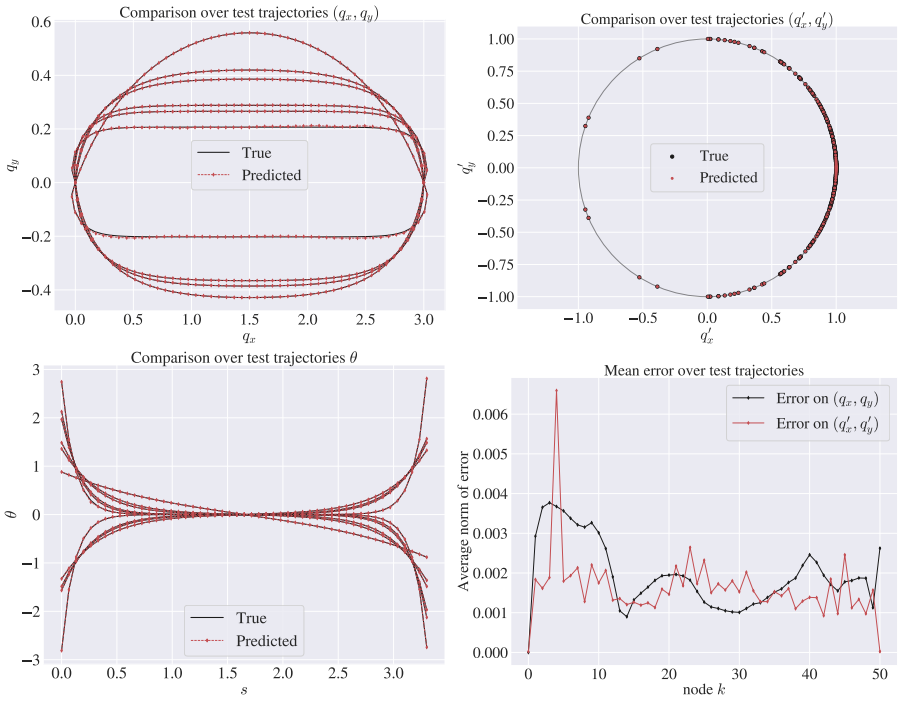


Figure 7.5.3: Comparison over test trajectories for $\mathbf{q}$ and $\mathbf{q}'$, for the case $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$ is modelled as in (7.5.7), with 80% – 10% – 10% splitting of the *both-ends* data set into training, validation, and test sets. The mean squared error on the test set equals $4.385 \cdot 10^{-6}$. For presentation purposes, only ten randomly selected trajectories are considered in the first two plots.

symmetric reconstruction procedures can be adopted, but the proposed one has provided better experimental results.

Comparing the results related to $q^{\mathrm{c}}_{\boldsymbol{\rho}}$ with those of $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$, we notice similar performances in terms of training and test errors. In both cases, they have one order of magnitude more than the corresponding training and test errors of the discrete network $q^{\mathrm{d}}_{\boldsymbol{\rho}}$. Thus, as a result of our experiments, we can conclude that

- if the accuracy and the efficient evaluation of the model at the discrete nodes are of interest, the discrete network is the best option;
- for a more flexible model, not restricted to the discrete nodes, the continuous network is a better choice; among the two proposed modelling strategies, working with $q^{\mathrm{c}}_{\boldsymbol{\rho}}$ is more suitable for an easy parametrisation of both $\mathbf{q}$ and $\mathbf{q}'$, while $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$ is more suitable to impose geometrical structure and constraints.

The total accuracy error of a neural network model can be defined by splitting it into three components: approximation error, optimisation error, and generalisation error (see e.g. [29]). To achieve excellent agreement between predicted and reference trajectories, it is crucial to select the appropriate architecture and fine-tune the model hyperparameters. Our results demonstrate that we can construct a network that is expressive enough to provide a small approximation error and has very good generalisation capability.

Lastly, we have compared the time cost of the Neural Network prediction against the traditional approach with numerical solvers as described in Section 7.2.2. The discrete and continuous approaches outperformed the traditional solvers with an average speedup of 105.000 times and 260.000 times, respectively, across the test trajectories. The training time of the continuous network is 1.25 times larger than that of the discrete network. It's important to note that these results are subject to certain limitations, such as the specific choice of the hyperparameters or the machine used to train and test the network. These findings suggest that using neural networks to predict new solutions of the elastica for unseen boundary conditions is much more time efficient than the classical numerical methods, although requiring intensive offline training.

7.6.1 Future work

In the methods presented in this paper, the mathematical problem and the neural network model do not interact once the data set is created. To improve the results presented here, one could include Euler's elastica model directly into the training process. This could be done either by directly imposing in the loss function that $\mathbf{q}(s)$ satisfies the differential equations (7.2.5), or one could add the constrained action integral from (7.2.4) into the loss function that is minimised, see e.g. [24, 40, 10, 44].

There are many promising directions to follow up on this work. One is to consider 3D versions of Euler's elastica, another is to look at the dynamical problem, and finally one may examine industrial applications, as mentioned in the introduction.

Acknowledgments. This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 860124. This work was partially supported by a grant from the Simons Foundation (DM). This contribution reflects only the authors' view, and the Research Executive Agency and the European Commission are not responsible for any use that may be made of the information it contains.

Appendix

7.A Architecture for the continuous network

We provide the expression of the forward propagation of the multiplicative network MULT used for the experiments in Section 7.5:

$$\mathbf{U} = \sigma\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1\right), \quad \mathbf{V} = \sigma\left(\mathbf{W}_2\mathbf{x} + \mathbf{b}_2\right) \tag{7.A.1}$$

$$\mathbf{H}_1 = \sigma\left(\mathbf{W}_3\mathbf{x} + \mathbf{b}_3\right) \tag{7.A.2}$$

$$\mathbf{Z}_j = \sigma\left(\mathbf{W}_j^z\mathbf{H}_j + \mathbf{b}_j^z\right), \; j = 1, \dots, \ell \tag{7.A.3}$$

$$\mathbf{H}_{j+1} = \left(1 - \mathbf{Z}_j\right) \odot \mathbf{U} + \mathbf{Z}_j \odot \mathbf{V}, \; j = 1, \dots, \ell \tag{7.A.4}$$

$$f_{\boldsymbol{\rho}}^{\text{MULT}}(\mathbf{x}) = \mathbf{W}\mathbf{H}_{\ell+1} + \mathbf{b}, \tag{7.A.5}$$

where $\odot$ denotes the component-wise multiplications. In this case,

$$\boldsymbol{\rho} = \left\{\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b}_2, \mathbf{W}_3, \mathbf{b}_3, \left(\mathbf{W}_j^z, \mathbf{b}_j^z\right)_{j=1}^{\ell}, \mathbf{W}, \mathbf{b}\right\},$$

and the weight matrices and biases have shapes that allow for the expressions (7.A.1)-(7.A.5) to be well-defined. This architecture is inspired by neural attention mechanisms and was introduced in [51] to improve the gradient behaviour. A further motivation for our choice of including this architecture is experimental since it has proven effective in solving the task of interest while still having a similar number of parameters to the MLP architecture. Throughout the paper, we refer to this architecture as *multiplicative* since it includes component-wise multiplications, which help capture multiplicative interactions between the variables.

7.B Details on hyperparameter optimisation

We provide here further details on the hyperparameter optimisation strategy with `Optuna`, relative to the results in Sections 7.4 and 7.5. The tables be-

low display the hyperparameters we optimise for in each of the networks, the ranges and the distribution, as well as the selected combinations used to perform the experiments in the paper.

Hyperparameter	Range	Distribution
# layers ℓ	$\{0, ..., 10\}$	discrete uniform
# hidden nodes	$[10, 1000] \cap \mathbb{N}$	discrete uniform
γ	$[0, 1 \cdot 10^{-2}]$	uniform

Table 7.B.1: Hyperparameter ranges for the discrete network $q^{\mathrm{d}}_{\boldsymbol{\rho}}$. The first column of the table reports the hyperparameters we test. The second describes the set of allowed values for each, while the third specifies how such values are explored through `Optuna`.

Hyperparameter	% of trajectories of the whole dataset in the training set			
	10%	20%	40%	80%
# layers ℓ	4	4	4	4
# hidden nodes	950	978	997	985
γ	$7.044 \cdot 10^{-3}$	$6.336 \cdot 10^{-3}$	$9.004 \cdot 10^{-3}$	$3.853 \cdot 10^{-3}$

Table 7.B.2: Choice of hyperparameters for the training of the discrete network $q^{\mathrm{d}}_{\boldsymbol{\rho}}$ tested on the *both-ends* data set with different sizes of the training set, with the validation and test sets each containing 10% of trajectories of the dataset.

Hyperparameter	Range	Distribution
# layers ℓ	$\{5, \ldots, 10\}$	discrete uniform
# hidden nodes	$[10, 250] \cap \mathbb{N}$	discrete uniform

Table 7.B.3: Hyperparameter ranges for the continuous network $q^{\mathrm{c}}_{\boldsymbol{\rho}}$. The first column of the table reports the hyperparameters we test. The second describes the set of allowed values for each, while the third specifies how such values are explored through `Optuna`.

Hyperparameter	% of trajectories of the whole dataset in the training set			
	10%	20%	40%	80%
# layers ℓ	6	7	8	6
# hidden nodes	139	185	181	106

Table 7.B.4: Choice of hyperparameters for the training of the continuous network $q^{\mathrm{c}}_{\boldsymbol{\rho}}$ tested on the *both-ends* data set with different sizes of the training set, with the validation and test sets each containing 10% of trajectories of the dataset.

Hyperparameter	Range	Distribution
# layers ℓ	$\{1,\dots,10\}$	discrete uniform
# hidden nodes	$[50,200]\cap\mathbb{N}$	discrete uniform

Table 7.B.5: Hyperparameter ranges for the continuous network $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$.The first column reports the hyperparameters we test. The second describes the set of allowed values for each, while the third specifies how such values are explored through `Optuna`.

Hyperparameter	$\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$ as in (7.5.6)	$\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$ as in (7.5.7)
# layers ℓ	8	8
# hidden nodes	93	58

Table 7.B.6: Choice of the hyperparameters for the training of the continuous network $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$ tested on the both-ends data set with 80%−10%−10% splitting. The second column shows the combination of hyperparameters yielding the best result corresponding to Figure 7.5.2, while the third column that corresponding to Figure 7.5.3.

Bibliography

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2623–2631, 2019. 262

[2] Davide Bigoni. *Nonlinear Solid Mechanics: Bifurcation Theory and Material Instability*. Cambridge University Press, (2012). 257

[3] Susanne C Brenner. *The Mathematical Theory of Finite Element Methods*. Springer, (2008). 2, 252

[4] S. L. Brunton and J. N. Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2 edition, 2022. 252

[5] Steven L. Brunton and J. Nathan Kutz. Machine Learning for Partial Differential Equations. *arXiv:2303.17078*, 2023. 252

[6] Samuel Chevalier, Jochen Stiasny, and Spyros Chatzivasileiadis. Accelerating Dynamical System Simulations with Contracting and Physics-Projected Neural-Newton Solvers. In *Learning for Dynamics and Control Conference*, pages 803–816, PMLR, 2022. 252

[7] Leonardo Colombo, Sebastián Ferraro, and David Martín de Diego. Geometric Integrators for Higher-Order Variational Systems and Their Application to Optimal Control. *Journal of Nonlinear Science*, 26:1615–1650, 2016. 255

[8] Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific Machine Learning Through Physics–Informed Neural Networks: Where we are and What's Next. *Journal of Scientific Computing*, 92(3):88, 2022. 252, 253

[9] Mario De Florio, Enrico Schiassi, and Roberto Furfaro. Physics-informed neural networks and functional interpolation for stiff chemical kinetics. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 32(6), 2022. 252, 284, 294

[10] W. E and B. Yu. The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018. 252, 271

[11] L. Euler. *De Curvis Elasticis*. Additamentum in Methodus inveniendi lineas curvas maximi minimive proprietate gaudentes, sive solutio problematis isoperimetrici lattissimo sensu accepti, Lausanne, 1744. 253

[12] G. Fabiani, F. Calabrò, L. Russo, and C. Siettos. Numerical solution and bifurcation analysis of nonlinear partial differential equations with extreme learning machines. *Journal of Scientific Computing*, 89(2):44, 2021. 252

[13] Gianluca Fabiani, Evangelos Galaris, Lucia Russo, and Constantinos Siettos. Parsimonious physics-informed random projection neural networks for initial value problems of ODEs and index-1 DAEs. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 33(4), 2023. 252, 284

[14] Sebastián J Ferraro, David Martín de Diego, and Rodrigo T Sato Martín de Almagro. Parallel iterative methods for variational integration applied to navigation problems. *IFAC-PapersOnLine*, 54(19):321–326, 2021. 257

[15] E. Galaris, G. Fabiani, I. Gallos, I. Kevrekidis, and C. Siettos. Numerical Bifurcation Analysis of PDEs From Lattice Boltzmann Model Simulations: a Parsimonious Machine Learning Approach. *Journal of Scientific Computing*, 92(2):34, 2022. 252

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT press, (2016). 8, 159, 259

[17] Yiqi Gu and Michael K Ng. Deep Neural Networks for Solving Large Linear Systems Arising from High-Dimensional Problems. *SIAM Journal on Scientific Computing*, 45(5):A2356–A2381, 2023. 2, 252

[18] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I, Nonstiff Problems*. Springer-Verlag, Second revised edition edition, (1993). 252

[19] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II, Stiff and Differential-Algebraic Problems*. Springer-Verlag, Second revised edition edition, (1996). 252

[20] Catherine F Higham and Desmond J Higham. Deep Learning: An Introduction for Applied Mathematicians. *Siam review*, 61(4):860–891, 2019. 259

[21] JCS Kadupitiya, Geoffrey C Fox, and Vikram Jadhao. Solving Newton's equations of motion with large timesteps using recurrent neural networks based operators. *Machine Learning: Science and Technology*, 3(2):025002, 2022. 252

[22] Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015. 260, 261

[23] Stefan Kollmannsberger, Davide D'Angella, Moritz Jokeit, and Leon Herrmann. *Deep Learning in Computational Mechanics*. Springer, (2021). 253, 254, 259

[24] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial Neural Networks for Solving Ordinary and Partial Differential Equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998. 29, 253, 271

[25] Ying Li, Zuojia Zhou, and Shihui Ying. DeLISA: Deep learning based iteration scheme approximation for solving PDEs. *Journal of Computational Physics*, 451:110884, 2022. 252

[26] Yuying Liu, J Nathan Kutz, and Steven L Brunton. Hierarchical deep learning of multiscale differential equation time-steppers. *Philosophical Transactions of the Royal Society A*, 380(2229):20210200, 2022. 252

[27] Alexander Humer Loc Vu-Quoc. Deep Learning Applied to Computational Mechanics: A Comprehensive Review, State of the Art, and the Classics. *Computer Modeling in Engineering & Sciences*, 137(2):1069–1343, 2023. 252

[28] A. E. H. Love. *A Treatise on the Mathematical Theory of Elasticity*. Cambridge University Press, Cambridge, 1863 - 1940. 253

[29] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature machine intelligence*, 3(3):218–229, 2021. 252, 271

[30] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM review*, 63(1):208–228, 2021. 252

[31] Davide Manfredo, Vanessa Dörlich, Joachim Linn, and Martin Arnold. Data based constitutive modelling of rate independent inelastic effects in composite cables using preisach hysteresis operators. *Multibody System Dynamics*, pages 1–16, 2023. 253

[32] Jerrold E Marsden and Matthew West. Discrete mechanics and variational integrators. *Acta numerica*, 10:357–514, 2001. 252

[33] Shigeki Matsutani. Euler's Elastica and Beyond. *Journal of Geometry and Symmetry in Physics*, 17:45–86, 2010. 253, 255, 258

[34] Marios Mattheakis, David Sondak, Akshunna S Dogra, and Pavlos Protopapas. Hamiltonian neural networks for solving equations of motion. *Physical Review E*, 105(6):065305, 2022. 252

[35] Daniele Mortari, Hunter Johnston, and Lidia Smith. High accuracy least-squares solutions of nonlinear differential equations. *Journal of computational and applied mathematics*, 352:293–307, 2019. 252, 284

[36] Jorge Nocedal and Stephen J Wright. *Numerical Optimization*. Springer, (1999). 252, 294

[37] Konstantina Ntarladima, Michael Pieber, and Johannes Gerstmayr. A model for contact and friction between beams under large deformation and sheaves. *Nonlinear Dynamics*, pages 1–18, 2023. 253

[38] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in neural information processing systems*, 32, 2019. 261

[39] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical Mathematics*, volume 37. Springer Science & Business Media, (2006). 252, 266, 288

[40] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019. 29, 48, 252, 253, 271

[41] Franz M. Rohrhofer, Stefan Posch, Clemens Gößnitzer, and Bernhard C Geiger. On the Role of Fixed Points of Dynamical Systems in Training Physics-Informed Neural Networks. *Transactions on Machine Learning Research*, 2022. 254

[42] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, (2003). 252

[43] Mohammad Ali Saadat and Damien Durville. A mixed stress-strain driven computational homogenization of spiral strands. *Computers & Structures*, 279:106981, 2023. 253

[44] Esteban Samaniego, Cosmin Anitescu, Somdatta Goswami, Vien Minh Nguyen-Thanh, Hongwei Guo, Khader Hamdia, X Zhuang, and T Rabczuk. An Energy Approach to the Solution of Partial Differential Equations in Computational Mechanics via Machine Learning: Concepts, Implementation and Applications. *Computer Methods in Applied Mechanics and Engineering*, 362:112790, 2020. 252, 271

[45] Enrico Schiassi, Roberto Furfaro, Carl Leake, Mario De Florio, Hunter Johnston, and Daniele Mortari. Extreme theory of functional connections: A fast physics-informed neural network method for solving ordinary and partial differential equations. *Neurocomputing*, 457:334–356, 2021. 252, 253, 254

[46] David A Singer. Lectures on Elastic Curves and Rods. In *AIP Conference Proceedings*, volume 1002, pages 3–32, American Institute of Physics, 2008. 253, 256

[47] Martina Stavole, Rodrigo T Sato Martín de Almagro, Margus Lohk, and Sigrid Leyendecker. Homogenization of the constitutive properties of composite beam cross-sections. In *ECCOMAS Congress 2022-8th European Congress on Computational Methods in Applied Sciences and Engineering*, 2022. 253

[48] Stephen P Timoshenko and James M Gere. *Theory of Elastic Stability*. McGraw-Hill Book Company, (1961). 257

[49] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. 258

[50] Sifan Wang and Paris Perdikaris. Long-time integration of parametric evolution equations with physics-informed DeepONets. *Journal of Computational Physics*, 475:111855, 2023. 30, 267, 300

[51] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and Mitigating Gradient Flow Pathologies in Physics-Informed Neural Networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021. 272

[52] Genki Yagawa and Atsuya Oishi. *Computational Mechanics with Deep Learning: An Introduction*. Springer Nature, (2022). 252

Parallel-in-Time Solutions with Extreme Learning Machines

Marta Betcke, Lisa Maria Kreusser, and Davide Murari

Preprint

Abstract. This paper considers one of the fundamental parallel-in-time methods for the solution of ordinary differential equations, Parareal, and extends it by adopting a neural network as a coarse propagator. We provide a theoretical analysis of the convergence properties of the proposed algorithm and show its effectiveness for several examples, including Lorenz and Burgers' equations. In our numerical simulations, we further specialize the underpinning neural architecture to Extreme Learning Machines (ELMs), a 2−layer neural network where the first layer weights are drawn at random rather than optimized. This restriction substantially increases the efficiency of fitting ELM's weights in comparison to a standard feedforward network without negatively impacting the accuracy, as demonstrated in the SIR system example.

8.1 Introduction

In this paper, we consider initial value problems expressed as a system of first-order ordinary differential equations (ODEs). This wide class of problems arises in many social and natural sciences applications, including semi-discretized, time-dependent partial differential equations. We express a generic system of such differential equations as

$$\begin{cases} \mathbf{x}'(t) = \mathcal{F}\big(\mathbf{x}(t)\big) \in \mathbb{R}^d, \\ \mathbf{x}(0) = \mathbf{x}_0, \end{cases} \tag{8.1.1}$$

which will be our reference problem. Here, $'$ denotes the derivative with respect to the time variable. To guarantee the existence and uniqueness of its solutions, we assume that $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^d$ is a Lipschitz-continuous vector field and $t \in [0, T]$ for some $T > 0$. Solving an initial value problem like (8.1.1) analytically is generally not a possibility, and hence one needs to rely on numerical approximations to the solution curve $t \mapsto \mathbf{x}(t)$. Numerical techniques rely on introducing a time discretization $0 < t_1 < \cdots < t_N = T$ of the interval $[0, T]$, with steps $\Delta t_n = t_{n+1} - t_n$, and computing approximations $\mathbf{x}_n$ of the solution $\mathbf{x}(t_n)$ at the nodes t_n, i.e., $\mathbf{x}_n \approx \mathbf{x}(t_n)$. A popular and established option is provided by one-step methods, such as Runge–Kutta schemes, which relate $\mathbf{x}_{n+1}$ to $\mathbf{x}_n$ in terms of a map $\varphi_{\mathcal{F}}^{\Delta t_n}$ of the form $\mathbf{x}_{n+1} = \varphi_{\mathcal{F}}^{\Delta t_n}(\mathbf{x}_n)$. Collocation methods are a subset of Runge–Kutta methods [16, Section II.7] with particular relevance to this paper. These methods aim to approximate the solution on each interval $[t_n, t_{n+1}]$ with a real polynomial $\tilde{\mathbf{x}}$ of a sufficiently high degree and coefficients in $\mathbb{R}^d$. The updated solution is then computed as $\mathbf{x}_{n+1} = \varphi_{\mathcal{F}}^{\Delta t_n}(\mathbf{x}_n)$ evaluating the polynomial at $t = t_{n+1}$ as $\mathbf{x}_{n+1} = \tilde{\mathbf{x}}(t_{n+1}) \approx \mathbf{x}(t_{n+1})$. To determine the coefficients of the polynomial $\tilde{\mathbf{x}}(t)$, one needs to solve the system of algebraic equations $\tilde{\mathbf{x}}'(t_{n,c}) = \mathcal{F}(\tilde{\mathbf{x}}(t_{n,c}))$ for a set of C collocation points $t_n \le t_{n,1} < t_{n,2} < \cdots < t_{n,C} \le t_{n+1}$.

As initial value problems define causal processes, many time-stepping schemes are sequential by nature, in the sense that to compute $\mathbf{x}_{n+1}$, one has to compute $\mathbf{x}_n$ first. Nonetheless, multiple successful approaches such as Parareal [26], PFASST [9], and MGRIT [11] have introduced some notion of parallel-in-time solution of initial value problems (8.1.1), see for instance [13] for an overview of existing methods.

In this work, we build upon the Parareal algorithm [26]. The speedup in Parareal is achieved by coupling a fine time step integrator with a coarse step integrator. In each iteration, the coarse integrator updates the initial conditions of initial value problems on time subintervals, which can be solved in parallel and only entail fine step time integration over a short time. The elegance and strong theoretical grounding of the idea (see [14, 15], for instance) led to a number of variants of the Parareal algorithm, including combinations of Parareal with neural networks [25, 20, 21].

In recent years, solving differential equations with machine learning approaches gained in popularity; see, for instance, [22] for a review. For learned methods to become staple solvers, understanding their properties and ensuring they reproduce the qualitative behavior of the solutions is paramount. The problem of convergence and generalization for neural network-based PDE solvers has been considered in [29, 7, 5], for instance. An analysis of the approximation properties of neural networks in the context of PDE solutions is provided in [32, 23]. In the context of ODEs, there is an increasing interest in developing deep neural networks to learn time-stepping schemes unrestricted by constraints of the local Taylor series, including approaches based on flow maps [27], model order reduction [36], and spectral methods [24].

In the context of combining Parareal with neural networks, Parareal with a physics-informed neural network as a coarse propagator was suggested in [20]. In [25], the authors introduced a parallel (deep) neural network based on parallelizing the forward propagation following similar principles to those behind Parareal. In [21], it was proposed to learn a coarse propagator by parameterizing its stability function and optimizing the associated weights to minimize an analytic convergence factor of the Parareal method for parabolic PDEs.

Neural networks are generally considered as a composition of parametric maps whose weights are all optimized so that a task of interest is solved with sufficient accuracy. The common choice of the optimization procedure is gradient-based algorithms, which start from a random set of initial weights and update them iteratively until the stopping criterion has been reached. A class of neural networks where some of the weights are not updated at all is often called Extreme Learning Machines (ELMs) [19, 18]. Despite their seemingly reduced capability of approximating functions, these neural networks retain most of the

approximation results of more conventional neural networks. For example, as derived in [19, Theorem 2.1], ELMs with two layers and H hidden neurons, where only the last layer is optimized while all other weights are independently sampled from any interval according to any continuous probability distribution, can interpolate with probability one any set of H distinct input-to-output pairs. Their expressivity properties, see e.g. [34, 35], make them suitable for the approximation of solutions of ODEs which were successfully considered in [10, 30, 37, 8, 4], yielding accurate approximations in a fraction of the training time when compared to more conventional networks.

8.1.1 Contributions

In this work, we build a hybrid numerical method based on the Parareal framework, where an ELM constitutes the coarse time stepping scheme. We first derive an a-posteriori error estimate for general neural network-based solvers of ODEs. This theoretical result allows us to replace the coarse integrator of the Parareal method with an ELM while preserving its convergence guarantees. The ELMs are trained online during the Parareal iterations. There are several benefits to the proposed procedure. First, our hybrid approach comes with theoretical guarantees and allows us to solve a differential equation such that the produced solution is accurate to a certain degree. Additionally, using ELMs rather than a more conventional neural network leads to a significant speedup in the algorithm without sacrificing its capabilities. Indeed, as we show for the SIR problem, using ELMs leads to about half of the computational time of the other method, even without accounting for the offline training phase of the more conventional network. Further, we demonstrate the effectiveness of the proposed approach, together with the timings of the components of the algorithm, and apply it to several examples in Section 8.6.

8.1.2 Outline

The outline of the paper is as follows. We start with introducing the Parareal algorithm and its convergence properties in Section 8.2. Section 8.3 presents the theoretical derivation of an a-posteriori error estimate for neural network-based solvers. This result relies on a nonlinear variation of the constants formula, also called the Gröbner-Alekseev Theorem. In Section 8.4, we propose a hybrid algorithm combining the Parareal framework with the ELM-based coarse propagator. We study the convergence properties of this hybrid algorithm in Section 8.5. The effectiveness of the proposed method is tested in Section 8.6 on the benchmark dynamical systems studied in [14] with the addition of the

SIR and ROBER problems. We conclude with the summary and analysis of the obtained results in Section 8.7.

8.2 Parareal method

This section introduces the Parareal algorithm [26] and presents a convergence result needed for our derivations.

8.2.1 The method

The Parareal algorithm builds on two one-step methods that we call $\varphi_F^{\Delta t}, \varphi_C^{\Delta t} : \mathbb{R}^d \to \mathbb{R}^d$, denoting the fine and coarse integrators with timestep Δt, respectively. There are multiple options to design such maps, one being to use the same one-step method but with finer or coarser timesteps, e.g.,

$$\varphi_F^{\Delta t} := \varphi_C^{\Delta t/M} \circ \cdots \circ \varphi_C^{\Delta t/M} = \left(\varphi_C^{\Delta t/M}\right)^M, \ M \in \mathbb{N}.$$

This strategy motivates the subscripts of the two maps since these methods rely on a fine and a coarse mesh. Another option to define $\varphi_F^{\Delta t}$ and $\varphi_C^{\Delta t}$ is to use methods of different orders, hence different levels of accuracy with the same timestep Δt. Regardless of how we define these two methods, the map $\varphi_F^{\Delta t}$ is more expensive to evaluate than $\varphi_C^{\Delta t}$. The goal of the Parareal algorithm is to get an approximate solution $\{\mathbf{x}_n\}_{n=0}^N$ over the mesh $t_0 = 0 < t_1 < \cdots < t_N = T$, $\Delta t_n = t_{n+1} - t_n$, with the same degree of accuracy as the one obtained with $\varphi_F^{\Delta t_n}$ but in a shorter time. This is achieved by transforming (8.1.1) into a collection of initial value problems on a shorter time interval by using $\varphi_C^{\Delta t_n}$. This zeroth iterate of the method consists of finding intermediate initial conditions $\mathbf{x}_n^0$ by integrating (8.1.1) with $\varphi_C^{\Delta t_n}$ to get

$$\mathbf{x}_0^0 = \mathbf{x}_0, \ \mathbf{x}_{n+1}^0 = \varphi_C^{\Delta t_n}\left(\mathbf{x}_n^0\right), \ n = 0, \ldots, N-1, \ \Delta t_n = t_{n+1} - t_n,$$

and define the N initial value problems on the subintervals

$$\begin{cases} \mathbf{x}'(t) = \mathcal{F}\left(\mathbf{x}(t)\right), \\ \mathbf{x}\left(t_n\right) = \mathbf{x}_n^0, \end{cases} \qquad t \in \left[t_n, t_{n+1}\right], \ n = 0, \ldots, N-1. \tag{8.2.1}$$

These problems can now be solved in parallel using the fine integrator $\varphi_F^{\Delta t_n}$, which constitutes the parallel step in all successive Parareal iterates. A predictor-corrector scheme is used to iteratively update the initial conditions on the

subintervals $[t_n, t_{n+1}]$. Parareal iteration $i+1$ reads

$$\mathbf{x}_{n+1}^{i+1} = \varphi_F^{\Delta t_n}\left(\mathbf{x}_n^i\right) + \varphi_C^{\Delta t_n}\left(\mathbf{x}_n^{i+1}\right) - \varphi_C^{\Delta t_n}\left(\mathbf{x}_n^i\right),\ n = 0, \ldots, N-1. \tag{8.2.2}$$

A common choice of a stopping criterion is $\max_{n=1,\ldots,N} \|\mathbf{x}_n^{i+1} - \mathbf{x}_n^i\|_2 < \varepsilon$ for some tolerance $\varepsilon > 0$. The parallel speedup is achieved if this criterion is met with far less iterates than the number of time intervals N.

8.2.2 Interpretation of the correction term

Following [14], we provide the interpretation of (8.2.2) as an approximation of the Newton step for matching the exact flow at the time discretization points $t_0 = 0, \ldots, t_N = T$. We consider

$$\mathcal{H}(\mathbf{y}) := \begin{bmatrix} \mathbf{y}_0 - \mathbf{x}_0 \\ \mathbf{y}_1 - \phi_{\mathcal{F}}^{\Delta t_0}(\mathbf{y}_0) \\ \mathbf{y}_2 - \phi_{\mathcal{F}}^{\Delta t_1}(\mathbf{y}_1) \\ \vdots \\ \mathbf{y}_N - \phi_{\mathcal{F}}^{\Delta t_{N-1}}(\mathbf{y}_{N-1}) \end{bmatrix} = 0, \quad \mathbf{y} = \begin{bmatrix} \mathbf{y}_0 \\ \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_N \end{bmatrix} \in \mathbb{R}^{d\cdot(N+1)},$$

where $\phi_{\mathcal{F}}^{\Delta t}(\mathbf{x}_n)$ with $\mathbf{x}_n \in \mathbb{R}^d$ is the exact solution $\mathbf{x}(\Delta t)$ of the initial value problem

$$\begin{cases} \mathbf{x}'(t) = \mathcal{F}\left(\mathbf{x}(t)\right), \\ \mathbf{x}(0) = \mathbf{x}_n. \end{cases}$$

Linearizing $\mathcal{H}$ at the ith iterate, $\mathbf{x}^i$, equating it to 0 and solving for the $i+1$st iterate, $\mathbf{x}^{i+1}$, we arrive at the Newton update

$$\mathbf{x}_{n+1}^{i+1} = \phi_{\mathcal{F}}^{\Delta t_n}\left(\mathbf{x}_n^i\right) + \partial_{\mathbf{x}}\left(\phi_{\mathcal{F}}^{\Delta t_n}\right)\left(\mathbf{x}_n^i\right)\left(\mathbf{x}_n^{i+1} - \mathbf{x}_n^i\right), \quad n = 0, \ldots, N-1$$

for the solution of the system $\mathcal{H}(\mathbf{y}) = 0$. The idea behind Parareal is then to approximate the unknown $\phi_{\mathcal{F}}^{\Delta t_n}(\mathbf{x}_n^i)$ with $\varphi_F^{\Delta t_n}(\mathbf{x}_n^i)$, and the first order term with

$$\partial_{\mathbf{x}}\left(\phi_{\mathcal{F}}^{\Delta t_n}\right)\left(\mathbf{x}_n^i\right)\left(\mathbf{x}_n^{i+1} - \mathbf{x}_n^i\right) \approx \varphi_C^{\Delta t_n}\left(\mathbf{x}_n^{i+1}\right) - \varphi_C^{\Delta t_n}\left(\mathbf{x}_n^i\right),$$

which yields (8.2.2).

8.2.3 Convergence

Convergence of the Parareal iterations was proven in [14] under the assumption that the fine integrator $\varphi_F^{\Delta t}$ and the exact flow map $\phi_{\mathcal{F}}^{\Delta t}$ coincide.

Theorem 8.1 (Theorem 1 in [14]). *Let us consider the initial value problem (8.1.1) and partition the time interval* $[0,T]$ *into* N *intervals of size* $\Delta t = T/N$ *using a grid of nodes* $t_n = n\Delta t$. *Assume that the fine integrator* $\varphi_F^{\Delta t}$ *coincides with the exact flow map* $\phi_{\mathcal{F}}^{\Delta t}$, *i.e.* $\varphi_F^{\Delta t} = \phi_{\mathcal{F}}^{\Delta t}$. *Furthermore, suppose that there exist* $p \in \mathbb{N}$, *a set of continuously differentiable functions* $c_{p+1}, c_{p+2}, \cdots$, *and* $\alpha > 0$ *such that*

$$\begin{aligned} &\varphi_F^{\Delta t}(\mathbf{x}) - \varphi_C^{\Delta t}(\mathbf{x}) = c_{p+1}(\mathbf{x})(\Delta t)^{p+1} + c_{p+2}(\mathbf{x})(\Delta t)^{p+2} + \cdots, \quad \text{and} \\ &\left\| \varphi_F^{\Delta t}(\mathbf{x}) - \varphi_C^{\Delta t}(\mathbf{x}) \right\|_2 \leq \alpha (\Delta t)^{p+1} \end{aligned} \tag{8.2.3}$$

for every $\mathbf{x} \in \mathbb{R}^d$, *and also that there exists* $\beta > 0$ *such that*

$$\left\| \varphi_C^{\Delta t}(\mathbf{x}) - \varphi_C^{\Delta t}(\mathbf{y}) \right\|_2 \leq (1 + \beta \Delta t) \left\| \mathbf{x} - \mathbf{y} \right\|_2 \tag{8.2.4}$$

for every $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$. *Then there exists a positive constant* γ *such that, at the* $i-th$ *iterate of the Parareal method, the following bound holds*

$$\left\| \mathbf{x}(t_n) - \mathbf{x}_n^i \right\|_2 \leq \frac{\alpha}{\gamma} \frac{\left(\gamma (\Delta t)^{p+1}\right)^{i+1}}{(i+1)!} (1 + \beta \Delta t)^{n-i-1} \prod_{j=0}^{i} (n-j).$$

This result guarantees that as the iteration progresses, the method provides an increasingly accurate solution. Furthermore, when $i = n$, the last product on the right-hand side vanishes, which corresponds to the worst-case scenario of the sequential solution, a.k.a. at the nth iterate, the above idealized Parareal method replicates the analytical solution for the time subintervals up to t_n.

We take advantage of this convergence result in Section 8.4, constructing the coarse propagator as a neural network satisfying the assumptions of Theorem 8.1.

8.3 A-posteriori error estimate for solvers based on neural networks

We aim to design a hybrid parallel-in-time solver for (8.1.1) based on the Parareal algorithm. This procedure consists of the Parareal iteration where the coarse propagator $\varphi_C^{\Delta t}$ is replaced by a neural network. In Section 8.4, we will focus on a particular class of neural networks called Extreme Learning Machines (ELMs). For now, however, we do not specify the structure of the neural network and define it as a map $\mathcal{N}_\theta : [0, \Delta t] \times \mathbb{R}^d \to \mathbb{R}^d$, parametrized by weights θ, and satisfying the initial condition of the ODE, $\mathcal{N}_\theta(0; \mathbf{x}_0) = \mathbf{x}_0$.

In the classical Parareal iteration, the coarse propagator $\varphi_C^{\Delta t_n}$ is a map satisfying $\mathbf{x}(t_{n+1}) \approx \varphi_C^{\Delta t_n}(\mathbf{x}(t_n))$, where $\mathbf{x}(t)$ solves (8.1.1). The coarse propagator balances the cost versus accuracy of the approximation, with the sweet spot yielding optimal parallel speedup. With this in mind, we design our replacement to be a continuous function of time and to allow longer steps than commonly taken by single-step numerical methods as employed by $\varphi_C^{\Delta t_n}$. Motivated by collocation methods [16, Chapter II.7], we choose the weights of the neural network $\mathcal{N}_\theta$ so that it satisfies the differential equation (8.1.1) at some collocation points in the interval $[0, \Delta t]$. More explicitly, given a set $\{t_1, \ldots, t_C\} \subset [0, \Delta t]$, we look for a set of weights θ minimizing the loss function

$$\mathcal{L}(\theta, \mathbf{x}_0) := \sum_{c=1}^{C} \left\| \mathcal{N}_\theta'(t_c; \mathbf{x}_0) - \mathcal{F}\left(\mathcal{N}_\theta(t_c; \mathbf{x}_0)\right) \right\|_2^2. \tag{8.3.1}$$

Consistent with our convention, in (8.3.1) $'$ denotes the time derivative, i.e., the derivative with respect to the first component.

In the following, we propose an error analysis for the approximate solution $\mathcal{N}_\theta$. This error analysis allows us to provide a-posteriori theoretical guarantees on both the accuracy of the network $\mathcal{N}_\theta(\Delta t; \mathbf{x}_0)$ as a continuous approximation of the solution, as well as its potential as a replacement of $\varphi_C^{\Delta t}(\mathbf{x}_0)$ while keeping intact the convergence guarantees of Parareal. We focus on a practical error estimate based on quadrature rules. For an, albeit less practical, alternative estimate based on defect control, see Appendix 8.A.

Assumption 1. Assume that the collocation points $\{t_1, \ldots, t_C\} \subset [0, \Delta t]$, with $t_1 < \cdots < t_C$, define a Lagrange quadrature rule exact up to order p for some given $p \geq 1$, i.e., there is a set of weights $\rho_1, \ldots, \rho_C$ for which

$$\int_0^{\Delta t} f(t)\,\mathrm{d}t = \sum_{c=1}^{C} \rho_c f(t_c) =: \mathcal{I}_p(f; 0, \Delta t), \ \forall f \in \mathbb{P}_{p-1}, \tag{8.3.2}$$

where $\mathbb{P}_{p-1}$ is the set of real polynomials of degree $p-1$.

For a set of collocation points satisfying Assumption 1 and any scalar $p-$times continuously differentiable function $f \in \mathcal{C}^p(\mathbb{R}, \mathbb{R})$, it holds [33, Chapter 9]

$$\left| \mathcal{I}_p(f; 0, \Delta t) - \int_0^{\Delta t} f(t)\,\mathrm{d}t \right| \leq \kappa (\Delta t)^{p+1} \max_{\xi \in [0, \Delta t]} \left| f^{(p)}(\xi) \right|, \kappa > 0, \tag{8.3.3}$$

where $f^{(p)}$ is the derivative of f of order p.

We can now formulate a quadrature-based a-posteriori error estimate for the *continuous* approximation $\mathcal{N}_\theta(t; \mathbf{x}_0)$ that only requires the defect to be sufficiently small at the collocation points.

Theorem 8.2 (Quadrature-based a-posteriori error estimate)**.** *Let* $\mathbf{x}(t)$ *be the solution of the initial value problem (8.1.1) with* $\mathcal{F} \in \mathcal{C}^{p+1}(\mathbb{R}^d, \mathbb{R}^d)$*. Suppose that Assumption 1 on the C collocation points* $0 \le t_1 < \cdots < t_C \le \Delta t$ *is satisfied and assume that* $\mathcal{N}_\theta(\cdot; \mathbf{x}_0) : [0, \Delta t] \to \mathbb{R}^d$ *is smooth and satisfies the collocation conditions up to some error of magnitude* ε*, i.e.*

$$\left\| \mathcal{N}'_\theta(t_c; \mathbf{x}_0) - \mathcal{F}\left(\mathcal{N}_\theta(t_c; \mathbf{x}_0)\right) \right\|_2 \le \varepsilon, \ c = 1, \dots, C. \tag{8.3.4}$$

Then, there exist two constants $\alpha, \beta > 0$ *such that, for all* $t \in [0, \Delta t]$*,*

$$\left\| \mathbf{x}(t) - \mathcal{N}_\theta(t; \mathbf{x}_0) \right\|_2 \le \alpha (\Delta t)^{p+1} + \beta \varepsilon. \tag{8.3.5}$$

The proof of Theorem 8.2 is based on the Gröbner-Alekseev formula [16, Theorem 14.5] that we now state for completeness.

Theorem 8.3 (Gröbner-Alekseev)**.** *For* $\mathcal{F} \in \mathcal{C}^1(\mathbb{R}^d, \mathbb{R}^d)$ *and* $\mathcal{G} : \mathbb{R}^d \to \mathbb{R}^d$ *consider the solutions* $\mathbf{x}(t)$ *and* $\mathbf{y}(t)$ *of the two ODEs*

$$\begin{cases} \mathbf{x}'(t) = \mathcal{F}\left(\mathbf{x}(t)\right), \\ \mathbf{x}(0) = \mathbf{x}_0, \end{cases} \qquad \begin{cases} \mathbf{y}'(t) = \mathcal{F}\left(\mathbf{y}(t)\right) + \mathcal{G}\left(\mathbf{y}(t)\right), \\ \mathbf{y}(0) = \mathbf{x}_0, \end{cases}$$

assuming they both have a unique solution. For any times $0 \le s \le t$*, let* $\phi_{\mathcal{F}}^{s,t}(\mathbf{y}(s))$ *be the exact solution of the initial value problem*

$$\begin{cases} \mathbf{x}'(t) = \mathcal{F}\left(\mathbf{x}(t)\right), \\ \mathbf{x}(s) = \mathbf{y}(s). \end{cases}$$

Then, for any $t \ge 0$*, one has*

$$\mathbf{y}(t) = \mathbf{x}(t) + \int_0^t \left. \frac{\partial \phi_{\mathcal{F}}^{s,t}(\mathbf{x}_0)}{\partial \mathbf{x}_0} \right|_{\mathbf{x}_0 = \mathbf{y}(s)} \mathcal{G}(\mathbf{y}(s)) \mathrm{d}s. \tag{8.3.6}$$

We now prove the a-posteriori error estimate in Theorem 8.2 using Theorem 8.3.

Proof of Theorem 8.2. Let $\mathbf{x}(t)$ be the solution of the initial value problem (8.1.1). Further note that $t \mapsto \mathcal{N}_\theta(t; \mathbf{x})$ satisfies the initial value problem

$$\begin{cases} \mathcal{N}'_\theta(t; \mathbf{x}_0) = \mathcal{F}\left(\mathcal{N}_\theta(t; \mathbf{x}_0)\right) + \left[\mathcal{N}'_\theta(t; \mathbf{x}_0) - \mathcal{F}\left(\mathcal{N}_\theta(t; \mathbf{x}_0)\right) \right], \\ \mathcal{N}_\theta(0; \mathbf{x}_0) = \mathbf{x}_0. \end{cases}$$

Setting $\mathcal{G}\left(\mathcal{N}_\theta\left(t;\mathbf{x}_0\right)\right) = \mathcal{N}'_\theta\left(t;\mathbf{x}_0\right) - \mathcal{F}\left(\mathcal{N}_\theta\left(t;\mathbf{x}_0\right)\right)$, from (8.3.6) we obtain

$$\begin{aligned}\left\|\mathbf{x}(t) - \mathcal{N}_\theta\left(t;\mathbf{x}_0\right)\right\|_2 &= \left\| \int_0^t \left.\frac{\partial \phi_{\mathcal{F}}^{s,t}\left(\mathbf{y}_0\right)}{\partial \mathbf{y}_0}\right|_{\mathbf{y}_0=\mathcal{N}_\theta(s;\mathbf{x}_0)} \mathcal{G}\left(\mathcal{N}_\theta\left(s;\mathbf{x}_0\right)\right) \mathrm{d}s \right\|_2 \\ &\leq \delta \left\| \int_0^t \left[\mathcal{N}'_\theta\left(s;\mathbf{x}_0\right) - \mathcal{F}\left(\mathcal{N}_\theta\left(s;\mathbf{x}_0\right)\right)\right] \mathrm{d}s \right\|_2,\end{aligned}$$

where $0 < \delta < \infty$ bounds the norm of the Jacobian matrix of $\phi_{\mathcal{F}}^{s,t}$ for $0 \leq s \leq t \leq \Delta t$ by virtue of $\mathcal{F} \in \mathcal{C}^1(\mathbb{R}^d, \mathbb{R}^d)$. Approximating the integral with the quadrature and subsequently bounding the residual at the collocation points, we obtain

$$\begin{aligned}\left\|\mathbf{x}(t) - \mathcal{N}_\theta\left(t;\mathbf{x}_0\right)\right\|_2 &\leq \delta \left(\left\| \sum_{c=1}^{C} \rho_c \left[\mathcal{N}'_\theta\left(t_c;\mathbf{x}_0\right) - \mathcal{F}\left(\mathcal{N}_\theta\left(t_c;\mathbf{x}_0\right)\right)\right] \right\|_2 + \bar{\kappa}(\Delta t)^{p+1} \right) \\ &\leq \delta \left(\varepsilon \sum_{c=1}^{C} \left|\rho_c\right| + \bar{\kappa}(\Delta t)^{p+1} \right),\end{aligned}$$

where $t \in [0, \Delta t]$ and

$$\bar{\kappa} := \kappa \cdot \left(\max_{t\in[0,\Delta t]} \left\| \frac{d^p}{dt^p} \left[\mathcal{N}'_\theta\left(t;\mathbf{x}_0\right) - \mathcal{F}\left(\mathcal{N}_\theta\left(t;\mathbf{x}_0\right)\right)\right] \right\|_2 \right) > 0,$$

the right-hand side of (8.3.3). To conclude the proof we set $\alpha = \bar{\kappa}\delta$, $\beta = \delta \sum_{c=1}^{C} \left|\rho_c\right|$. □

While for the proof it suffices that δ is finite, more practical bounds based on the one-sided Lipschitz constant of the vector field can be obtained. We derive such a bound in Appendix 8.B.

Given $\beta\varepsilon \ll (\Delta t)^{p+1}$, Theorem 8.2 implies that the approximation provided by the neural network is as accurate as the one provided by a pth order one-step method with step size Δt. This result allows us to replace the coarse integrator $\varphi_C^{\Delta t}$ with a neural network-based solver maintaining the convergence properties of Parareal.

8.4 Parareal method based on Extreme Learning Machines

The theoretical results presented in Section 8.3 hold for generic continuous approximate solutions, particularly those provided by any neural network $\mathcal{N}_\theta$.

We now restrict the neural architecture to Extreme Learning Machines (ELMs), which allow a more efficient, hence faster, solution of the optimization problem (8.3.1) as we will highlight in Section 8.6.

8.4.1 Architecture design

ELMs are feedforward neural networks composed of two layers, with trainable parameters confined to the outermost layer. We draw the weights of the first layer from the continuous uniform distribution $\mathcal{U}(\underline{\omega}, \overline{\omega})$, for a lower bound $\underline{\omega}$ and an upper bound $\overline{\omega}$ which are set to -1 and 1, respectively, in all our experiments. We then aim to approximate the solution of (8.1.1) at a time t with the parametric function

$$\begin{aligned}\mathcal{N}_\theta\left(t;\mathbf{x}_0\right) &= \mathbf{x}_0 + \sum_{h=1}^{H} \mathbf{w}_h\left(\varphi_h(t) - \varphi_h(0)\right) = \mathbf{x}_0 + \sum_{h=1}^{H} \mathbf{w}_h\left(\sigma\left(a_h t + b_h\right) - \sigma\left(b_h\right)\right) \\ &= \mathbf{x}_0 + \theta^\top \sigma\left(\mathbf{a}\,t + \mathbf{b}\right),\ \mathbf{a} = \begin{bmatrix} a_1 \\ \vdots \\ a_H \end{bmatrix},\ \mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_H \end{bmatrix} \in \mathbb{R}^H,\ \theta = \begin{bmatrix} \mathbf{w}_1^\top \\ \vdots \\ \mathbf{w}_H^\top \end{bmatrix} \in \mathbb{R}^{H\times d},\end{aligned} \tag{8.4.1}$$

by training the weights collected in the matrix θ. Here, $\varphi_h(t) = \sigma(a_h t + b_h) \in \mathbb{R}$, $h = 1,\dots,H$, is a given set of basis functions with $a_h, b_h \sim \mathcal{U}(\underline{\omega}, \bar{\omega})$, and $\sigma \in \mathcal{C}^\infty(\mathbb{R},\mathbb{R})$ a smooth activation function. In the numerical experiments, we always choose $\sigma(\cdot) = \tanh(\cdot)$. The architecture in (8.4.1) satisfies the initial condition of (8.1.1), i.e., $\mathcal{N}_\theta\left(0;\mathbf{x}_0\right) = \mathbf{x}_0$. In addition, $t \mapsto \mathcal{N}_\theta\left(t;\mathbf{x}_0\right)$ and σ have the same regularity.

8.4.2 Algorithm design

Our method closely mimics the Parareal algorithm but with the network (8.4.1) deployed as a coarse propagator in the Parareal update (8.2.2). While in the classical Parareal algorithm the coarse propagator $\varphi_C^{\Delta t_n}$ is assumed to be known for all sub-intervals and Parareal iterations, we do not make this assumption in our approach. Instead, we determine individual weights for the update of each of the initial conditions featuring in the update (8.2.2) to allow for a better adaptation to the local behavior of the approximated solution. Furthermore, our neural coarse integrator $\varphi_C^{\Delta t_n}$ is not known ahead of time but is recovered and changing in the course of the Parareal iteration. Learning the coarse integrator involves training an ELM on each of the sub-intervals to solve the

ODE (8.1.1) at a set of fixed collocation points in the sub-interval. This procedure would be prohibitively expensive for generic neural networks trained with gradient-based methods. However, for ELMs, estimating the matrix θ in (8.4.1) is considerably cheaper and comparable with classical collocation approaches, striking a balance between the computational efficiency, desirable behavior, and flexibility of the integrator. Finally, in Section 8.5 we demonstrate that our approach is provably convergent.

8.4.3 Training strategy

The neural coarse propagator for solution (8.1.1) on the time interval $[0, T]$ is obtained by splitting the interval into N sub-intervals, $\Delta t_n = t_{n+1} - t_n$, $t_0 = 0 < t_1 < \cdots < t_N = T$, and training N individual ELMs in sequence. On the nth sub-interval $[t_n, t_{n+1}]$ we train an ELM of the form (8.4.1) to solve the ODE system (8.1.1) approximately on this sub-interval. The initial condition at time t_n is obtained by the evaluation of the previous Parareal correction step. Since the vector field $\mathcal{F}$ in (8.1.1) does not explicitly depend on the time variable, we can restrict our presentation to a solution on a sub-interval $[0, \Delta t_n]$

$$\begin{cases} \mathbf{x}'(t) = \mathcal{F}\left(\mathbf{x}(t)\right), \\ \mathbf{x}(0) = \mathbf{x}_n^i, \end{cases} \tag{8.4.2}$$

where the superscript i refers to ith Parareal iterate.

To train an ELM (8.4.1) on the sub-interval $[0, \Delta t_n]$, we introduce C collocation points $0 \leq t_{n,1} < \cdots < t_{n,C} \leq \Delta t_n$, where the subscript n keeps track of the interval length Δt_n emphasizing the independent choice of collocation points on each sub-interval. For a given initial condition $\mathbf{x}_n^i$, we find a matrix θ_n^i such that $\mathcal{N}_{\theta_n^i}$ approximately satisfies (8.4.2) for all $t_{n,c}$, $c = 1, \ldots, C$, by solving the optimization problem

$$\theta_n^i = \arg\min_{\theta \in \mathbb{R}^{H \times d}} \sum_{c=1}^{C} \left\| \mathcal{N}_\theta'\left(t_{n,c}; \mathbf{x}_n^i\right) - \mathcal{F}\left(\mathcal{N}_\theta\left(t_{n,c}; \mathbf{x}_n^i\right)\right) \right\|_2^2. \tag{8.4.3}$$

This hybrid Parareal method returns approximations of the solution at the time nodes $t_0, \ldots, t_N$, which we call $\mathbf{x}_0, \ldots, \mathbf{x}_N$. Furthermore, since the ELMs on sub-intervals are smooth functions of time, one could also access a piecewise smooth approximation of the curve $[0, T] \ni t \mapsto \mathbf{x}(t)$ by evaluating the individual ELMs upon convergence of the Parareal iteration

$$\tilde{\mathbf{x}}(t) = \mathcal{N}_{\theta_n}\left(t - t_n; \mathbf{x}_n\right), \; t \in \left[t_n, t_{n+1}\right), \; n = 0, \ldots, N-1. \tag{8.4.4}$$

Here, θ_n and $\mathbf{x}_n$ are the weight matrix and the initial condition at the time t_n in the final Parareal iteration. Note that even though the points $\mathbf{x}_n^i$ are updated

in each Parareal iteration (8.2.2), they do not tend to change drastically, and we can initialize θ_n^{i+1} in (8.4.3) with the previous iterate θ_n^i to speedup convergence. We terminate the Parareal iteration when either the maximum number of iterations is reached or the difference between two consecutive iterates satisfies a given tolerance.

Algorithm 4 Hybrid Parareal algorithm based on ELMs

Inputs : $\mathbf{x}_0$, `tol`, `max_it`
`error` $\leftarrow$ `tol` $+1$, $i \leftarrow 1$, $\mathbf{x}_0^0 \leftarrow \mathbf{x}_0$
for $n = 0$ **to** $N-1$ **do** ▷ Zeroth iterate
  Find $\theta_n^0 = \arg\min_{\theta \in \mathbb{R}^{H\times d}} \sum_{c=1}^{C} \left\| \mathcal{N}'_\theta\left(t_{n,c}; \mathbf{x}_n^0\right) - \mathcal{F}\left(\mathcal{N}_\theta\left(t_{n,c}; \mathbf{x}_n^0\right)\right) \right\|_2^2$
  $\mathbf{x}_{n+1}^0 \leftarrow \mathcal{N}_{\theta_n^0}\left(\Delta t_n; \mathbf{x}_n^0\right)$, $\mathbf{x}_{n+1}^{S,-1} \leftarrow \mathcal{N}_{\theta_n^0}\left(\Delta t_n; \mathbf{x}_n^0\right)$
end for
while $i <$ `max_it` **and** `error` $>$ `tol` **do**
  `error` $\leftarrow 0$
  for $n = 0$ **to** $N-1$ **do** ▷ Fine integrator, **Parallel** For Loop
    $\mathbf{x}_{n+1}^F \leftarrow \varphi_F^{\Delta t_n}(\mathbf{x}_n^{i-1})$
  end for
  $\mathbf{x}_0^i \leftarrow \mathbf{x}_0$
  for $n = 0$ **to** $N-1$ **do**
    Find $\theta_n^i = \arg\min_{\theta \in \mathbb{R}^{H\times d}} \sum_{c=1}^{C} \left\| \mathcal{N}'_\theta\left(t_{n,c}; \mathbf{x}_n^i\right) - \mathcal{F}\left(\mathcal{N}_\theta\left(t_{n,c}; \mathbf{x}_n^i\right)\right) \right\|_2^2$
    $\mathbf{x}_{n+1}^S \leftarrow \mathcal{N}_{\theta_n^i}\left(\Delta t_n; \mathbf{x}_n^i\right)$ ▷ Next coarse approximation
    $\mathbf{x}_{n+1}^i \leftarrow \mathbf{x}_{n+1}^F + \mathbf{x}_{n+1}^S - \mathbf{x}_{n+1}^{S,-1}$ ▷ Parareal correction
    $\mathbf{x}_{n+1}^{S,-1} \leftarrow \mathbf{x}_{n+1}^S$
    `error` $\leftarrow \max\left\{\text{error}, \left\|\mathbf{x}_{n+1}^i - \mathbf{x}_{n+1}^{i-1}\right\|_2\right\}$
  end for
  $i \leftarrow i+1$
end while
return $\left\{\mathbf{x}_0^{i-1}, \ldots, \mathbf{x}_N^{i-1}\right\}$, $\left\{\theta_0^{i-1}, \ldots, \theta_{N-1}^{i-1}\right\}$

8.4.4 Implementation details

Our hybrid Parareal method is described in Algorithm 4, and the Python code can be found in the associated GitHub repository[1]. The zeroth iterate of the

[1] https://github.com/davidemurari/ELMHybridParareal

method, starting in line 3, only relies on ELMs to get intermediate initial conditions $\mathbf{x}_n^0$, $n = 0, \dots, N-1$. These initial conditions are then used to solve with the fine integrators $\varphi_F^{\Delta t_n}$ the N initial value problems in parallel, see line 10. These approximations are subsequently updated in the Parareal correction step of line 16.

The Algorithm 4 relies on solving a *nonlinear* optimization problem in lines 4 and 14 to update the weights θ_n^i. For all systems studied in Section 8.6 but the Burgers' equation, we use the Levenberg–Marquardt algorithm [31, Chapter 10]. For Burgers' equation, we rely on the Trust Region Reflective algorithm [2] to exploit the sparsity of the Jacobian matrix. The optimization algorithms are implemented with the `least_squares` method of `scipy.optimize`. In both cases, we provide an analytical expression of the Jacobian of the loss function with respect to the weight θ, derived in Appendix 8.C. Additionally, all the systems but the ROBER problem are solved on a uniform grid, i.e., $t_n = nT/N$. For the ROBER problem, we work with a non-uniform grid, refined in $[0, 1]$, to capture the spike in the solution occurring at small times.

As common in neural network-based approaches for solving differential equations, see, e.g., [4], we opt for C equispaced collocation points in each time interval. We also tested Lobatto quadrature points in the Lorenz example in subsection 8.6.3. In all experiments, we set $C = 5$ and the number of hidden neurons $H = C = 5$ to match.

8.5 Convergence of the ELM-based Parareal method

In this section, we study the convergence properties of Algorithm 4. Following the Parareal analysis in Theorem 8.1, we only need to consider the time interval $[0, \Delta t]$ and collocation points $0 < t_1 < \cdots < t_C < \Delta t$ satisfying Assumption 1.

We write our solution ansatz, (8.4.1), and its time derivative evaluated at the collocation points as the matrices

$$\tilde{\mathbf{X}}_\theta(\mathbf{x}, \Delta t) = \begin{bmatrix} \mathcal{N}_\theta(t_1; \mathbf{x})^\top \\ \vdots \\ \mathcal{N}_\theta(t_C; \mathbf{x})^\top \end{bmatrix} \in \mathbb{R}^{C \times d}, \quad \tilde{\mathbf{X}}'_\theta(\mathbf{x}, \Delta t) = \begin{bmatrix} \mathcal{N}'_\theta(t_1; \mathbf{x})^\top \\ \vdots \\ \mathcal{N}'_\theta(t_C; \mathbf{x})^\top \end{bmatrix} \in \mathbb{R}^{C \times d},$$

and shorthand the evaluation of the vector field $\mathcal{F}$ on the rows of the matrix

$\mathbf{X} \in \mathbb{R}^{C\times d}$

$$\mathbf{F}(\mathbf{X}) = \begin{bmatrix} \mathcal{F}\left(\mathbf{X}^\top \mathbf{e}_1\right)^\top \\ \vdots \\ \mathcal{F}\left(\mathbf{X}^\top \mathbf{e}_C\right)^\top \end{bmatrix} \in \mathbb{R}^{C\times d},$$

with $\mathbf{e}_1, \dots, \mathbf{e}_C \in \mathbb{R}^C$ the canonical basis of $\mathbb{R}^C$.

We further rewrite the ansatz as $\tilde{\mathbf{X}}_\theta(\mathbf{x}, \Delta t) = \mathbf{1}_C \mathbf{x}^\top + (\mathbf{H} - \mathbf{H}_0)\theta$, where $\mathbf{1}_C = \begin{bmatrix} 1 & \dots & 1 \end{bmatrix}^\top \in \mathbb{R}^C$,

$$\mathbf{H} = \begin{bmatrix} \sigma\left(\mathbf{a}^\top t_1 + \mathbf{b}^\top\right) \\ \vdots \\ \sigma\left(\mathbf{a}^\top t_C + \mathbf{b}^\top\right) \end{bmatrix} \in \mathbb{R}^{C\times H}, \quad \mathbf{H}' = \begin{bmatrix} \sigma'\left(\mathbf{a}^\top t_1 + \mathbf{b}^\top\right) \odot \mathbf{a}^\top \\ \vdots \\ \sigma'\left(\mathbf{a}^\top t_C + \mathbf{b}^\top\right) \odot \mathbf{a}^\top \end{bmatrix} \in \mathbb{R}^{C\times H},$$

with $\mathbf{a}^\top = \begin{bmatrix} a_1 & \cdots & a_H \end{bmatrix}$, $\mathbf{b}^\top = \begin{bmatrix} b_1 & \cdots & b_H \end{bmatrix}$, and $\sigma \in \mathcal{C}^\infty(\mathbb{R}, \mathbb{R})$ evaluated componentwise while $\odot$ denotes the componentwise product. As for the experiments, we restrict ourselves to the case $C = H$ for which one can prove, see [19, Theorem 2.1], that with probability one $\mathbf{H}$ and $\mathbf{H}'$ are invertible for $\mathbf{a}, \mathbf{b}$ drawn from any continuous probability distribution. Finally, $\mathbf{H}_0 = \mathbf{1}_C \sigma\left(\mathbf{b}^\top\right) \in \mathbb{R}^{C\times H}$ in $\tilde{\mathbf{X}}_\theta(\mathbf{x}, \Delta t)$, accounts for the initial condition.

Theorem 8.4 (Existence and regularity of the solution)**.** *For the loss function*

$$\mathcal{L}(\theta, \mathbf{x}) := \left\| \tilde{\mathbf{X}}'_\theta(\mathbf{x}, \Delta t) - \mathbf{F}\left(\tilde{\mathbf{X}}_\theta(\mathbf{x}, \Delta t)\right) \right\|_F^2 \tag{8.5.1}$$

with $\mathcal{N}_\theta$ in $\tilde{\mathbf{X}}_\theta$ defined as in (8.4.1), σ a smooth $1-$Lipschitz activation function, and a choice of step size

$$\Delta t \in \left(0, \left(\left\| (\mathbf{H}')^{-1} \right\|_2 \mathrm{Lip}(\mathcal{F}) \sqrt{C} \|\mathbf{a}\|_2 \right)^{-1} \right), \tag{8.5.2}$$

there exists a unique Lipschitz continuous function $\mathbb{R}^d \ni \mathbf{x} \mapsto \theta(\mathbf{x}) \in \mathbb{R}^{H\times d}$ such that $\mathcal{L}(\theta(\mathbf{x}), \mathbf{x}) = 0$ for every $\mathbf{x} \in \mathbb{R}^d$.

We remark that the loss function in (8.5.1) using the Frobenius norm $\|\cdot\|_F$ is a reformulation of (8.3.1) in a matrix form. We now prove Theorem 8.4 using a parameterized version of Banach Contraction Theorem presented in [3, Lemma 1.9].

Proof. The requirement $\mathcal{L}(\theta,\mathbf{x}) = 0$ implies that the ansatz $\tilde{\mathbf{X}}_\theta(\mathbf{x},\Delta t) = \mathbf{1}_C\mathbf{x}^\top + (\mathbf{H}-\overline{\mathbf{H}})\theta$ and its derivative, $\tilde{\mathbf{X}}'_\theta(\mathbf{x},\Delta t) = \mathbf{H}'\theta$, satisfy the ODE (8.4.2),

$$\tilde{\mathbf{X}}'_\theta(\mathbf{x},\Delta t) = \mathbf{F}\left(\tilde{\mathbf{X}}_\theta(\mathbf{x},\Delta t)\right),$$

which can be equivalently written as $\mathbf{H}'\theta = \mathbf{F}\left(\mathbf{1}_C\mathbf{x}^\top + \left(\mathbf{H}-\overline{\mathbf{H}}\right)\theta\right)$. We introduce the fixed point map

$$T(\theta,\mathbf{x}) = \left(\mathbf{H}'\right)^{-1}\mathbf{F}\left(\mathbf{1}_C\mathbf{x}^\top + \left(\mathbf{H}-\overline{\mathbf{H}}\right)\theta\right) \in \mathbb{R}^{H\times d},$$

and, when not differently specified, we denote with $\mathrm{Lip}(f)$ the Lipschitz constant of a Lipschitz continuous function f with respect to the ℓ^2 norm. Since $\mathrm{Lip}(\sigma) \le 1$, we have

$$\left\|\mathbf{H}-\overline{\mathbf{H}}\right\|_F^2 = \sum_{c=1}^{C}\left\|\sigma\left(\mathbf{a}^\top t_c + \mathbf{b}^\top\right) - \sigma\left(\mathbf{b}^\top\right)\right\|_2^2 \le C\|\mathbf{a}\|_2^2(\Delta t)^2$$

as $t_c \in (0,\Delta t)$. Furthermore,

$$\left\|\mathbf{F}(\mathbf{X}) - \mathbf{F}(\mathbf{Y})\right\|_F^2 = \sum_{c=1}^{C}\left\|\mathcal{F}(\mathbf{X}^\top\mathbf{e}_c) - \mathcal{F}(\mathbf{Y}^\top\mathbf{e}_c)\right\|_2^2 \le \mathrm{Lip}\left(\mathcal{F}\right)^2\|\mathbf{X}-\mathbf{Y}\|_F^2$$

for any $\mathbf{X},\mathbf{Y} \in \mathbb{R}^{C\times d}$. Setting $\ell_\theta = \left\|\left(\mathbf{H}'\right)^{-1}\right\|_2 \mathrm{Lip}(\mathcal{F})\sqrt{C}\|\mathbf{a}\|_2\Delta t$ we conclude that $T(\cdot,\mathbf{x})$ is Lipschitz continuous with constant $\ell_\theta < 1$ for Δt satisfying (8.5.2), as

$$\begin{aligned}\left\|T(\theta_2,\mathbf{x}) - T(\theta_1,\mathbf{x})\right\|_F &\le \left\|\left(\mathbf{H}'\right)^{-1}\right\|_2 \mathrm{Lip}(\mathcal{F})\sqrt{C}\|\mathbf{a}\|_2\Delta t\left\|\theta_2-\theta_1\right\|_F \\ &= \ell_\theta\left\|\theta_2-\theta_1\right\|_F\end{aligned}$$

for any $\theta_1,\theta_2 \in \mathbb{R}^{H\times d}$. We note that the 2−norm of $\left(\mathbf{H}'\right)^{-1}$ can be used since for any pair of matrices $\mathbf{A},\mathbf{B}$ of compatible dimensions, it holds $\|\mathbf{AB}\|_F \le \|\mathbf{A}\|_2\|\mathbf{B}\|_F$. Furthermore, $T(\theta,\cdot)$ is Lipschitz continuous with Lipschitz constant given by $\ell_{\mathbf{x}} = \left\|\left(\mathbf{H}'\right)^{-1}\right\|_2 \mathrm{Lip}\left(\mathcal{F}\right)\sqrt{C}$, since

$$\begin{aligned}\left\|T(\theta,\mathbf{x}_2) - T(\theta,\mathbf{x}_1)\right\|_F &\le \left\|\left(\mathbf{H}'\right)^{-1}\right\|_2 \mathrm{Lip}\left(\mathcal{F}\right)\left\|\mathbf{1}_C\left(\mathbf{x}_2-\mathbf{x}_1\right)^\top\right\|_F, \quad \forall \mathbf{x}_1,\mathbf{x}_2 \in \mathbb{R}^d \\ &= \left\|\left(\mathbf{H}'\right)^{-1}\right\|_2 \mathrm{Lip}\left(\mathcal{F}\right)\sqrt{C}\left\|\mathbf{x}_2-\mathbf{x}_1\right\|_2 = \ell_{\mathbf{x}}\left\|\mathbf{x}_2-\mathbf{x}_1\right\|_2.\end{aligned}$$

By [3, Lemma 1.9], we can hence conclude that, provided Δt satisfies (8.5.2), there is a well-defined Lipschitz continuous function $\theta : \mathbb{R}^d \to \mathbb{R}^{H\times d}$, with

$$\mathrm{Lip}\left(\theta\right) \le \frac{\ell_{\mathbf{x}}}{1-\ell_\theta} = \frac{\left\|\left(\mathbf{H}'\right)^{-1}\right\|_2 \mathrm{Lip}\left(\mathcal{F}\right)\sqrt{C}}{1-\left\|\left(\mathbf{H}'\right)^{-1}\right\|_2 \mathrm{Lip}(\mathcal{F})\sqrt{C}\|\mathbf{a}\|_2\Delta t},$$

such that $\theta(\mathbf{x}) = T(\theta(\mathbf{x}),\mathbf{x})$ for every $\mathbf{x} \in \mathbb{R}^d$, or equivalently $\mathcal{L}(\theta(\mathbf{x}),\mathbf{x}) = 0$. □

Proposition 8.1 (Convergence of the hybrid Parareal method)**.** Consider the initial value problem (8.1.1) with $\mathcal{F}:\mathbb{R}^d \to \mathbb{R}^d$ a smooth Lipschitz continuous vector field. Suppose that the time interval $[0,T]$ is partitioned into N intervals of size $\Delta t = T/N$ such that Δt satisfies (8.5.2) and choose the C collocation points $0 \le t_1 < \cdots < t_C \le \Delta t$ to satisfy the Assumption 1. Let σ be a smooth 1−Lipschitz function. Then for the coarse integrator $\varphi_C^{\Delta t}(\mathbf{x}) = \mathbf{x} + \theta(\mathbf{x})^T\left(\sigma(\mathbf{a}\Delta t + \mathbf{b}) - \sigma(\mathbf{b})\right)$ with $\theta(\mathbf{x})$ as in Theorem 8.4 and the fine integrator $\varphi_F^{\Delta t} = \phi_{\mathcal{F}}^{\Delta t}$, there exist positive constants α, γ, β such that, at the i−th iterate of the hybrid Parareal method, the following bound holds

$$\left\|\mathbf{x}(t_n) - \mathbf{x}_n^i\right\|_2 \le \frac{\alpha}{\gamma}\frac{\left(\gamma(\Delta t)^{p+1}\right)^{i+1}}{(i+1)!}\left(1+\beta\Delta t\right)^{n-i-1}\prod_{j=0}^{i}\left(n-j\right). \tag{8.5.3}$$

Proof. The proof is based on showing that our network satisfies assumptions (8.2.3) and (8.2.4) of Theorem 8.1. Theorem 8.4 guarantees that, for Δt satisfying (8.5.2), $\mathbf{x} \mapsto \theta(\mathbf{x})$ is Lipschitz continuous with Lipschitz constant $\mathrm{Lip}\left(\theta\right)$. Further noting that $\left\|\sigma(\mathbf{a}\Delta t + \mathbf{b}) - \sigma(\mathbf{b})\right\|_2 \le \|\mathbf{a}\|_2 \Delta t$ as $\mathrm{Lip}(\sigma) \le 1$ we can write

$$\begin{aligned}\left\|\varphi_C^{\Delta t}\left(\mathbf{x}_2\right) - \varphi_C^{\Delta t}\left(\mathbf{x}_1\right)\right\|_2 &\le \left\|\mathbf{x}_2 - \mathbf{x}_1\right\|_2 + \|\mathbf{a}\|_2\,\Delta t\,\mathrm{Lip}\left(\theta\right)\left\|\mathbf{x}_2 - \mathbf{x}_1\right\|_2 \\ &= \left(1+\beta\Delta t\right)\left\|\mathbf{x}_2 - \mathbf{x}_1\right\|_2\end{aligned}$$

for any $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^d$, where $\beta := \mathrm{Lip}\left(\theta\right)\|\mathbf{a}\|_2$, and hence condition (8.2.4) is satisfied. Given that $\theta(\mathbf{x}) = T(\theta(\mathbf{x}), \mathbf{x})$, as guaranteed by Theorem 8.4, satisfies the collocation conditions exactly, Theorem 8.2 ensures that there exists $\alpha > 0$ such that $\left\|\varphi_F^{\Delta t}(\mathbf{x}) - \varphi_C^{\Delta t}(\mathbf{x})\right\|_2 \le \alpha\left(\Delta t\right)^{p+1}$. Because of the smoothness of $\mathcal{F}$ and σ, one can also Taylor expand in time and guarantee the existence of continuously differentiable functions $c_{p+1}, c_{p+2}, \ldots$ such that

$$\phi_{\mathcal{F}}^{\Delta t}(\mathbf{x}) - \mathcal{N}_\theta\left(\Delta t; \mathbf{x}\right) = c_{p+1}(\mathbf{x})(\Delta t)^{p+1} + c_{p+2}(\mathbf{x})(\Delta t)^{p+2} + \cdots.$$

This allows concluding that (8.2.3) holds, and the hybrid Parareal satisfies (8.5.3). □

As for the classical Parareal method, at the nth iterate, our hybrid Parareal method with the exact fine integrator replicates the analytical solution at the time instants $t_0, \ldots, t_n$.

In practice, as presented in the previous section, we do not have access to the function $\mathbf{x} \mapsto \theta(\mathbf{x})$, but we only approximate its value at the points involved in the hybrid Parareal iterates, i.e., $\theta_n^i \approx \theta(\mathbf{x}_n^i)$. Let us denote by $\hat{\theta}:\mathbb{R}^d \to \mathbb{R}^{H\times d}$

the function approximating θ, so that $\theta_n^i = \hat{\theta}(\mathbf{x}_n^i)$. This function is typically provided by a convergent iterative method minimizing (8.5.1). Under the smoothness assumptions of Proposition 8.1 and supposing the map $x \mapsto \hat{\theta}(x)$ is Lipschitz continuous, i.e., the adopted optimization method depends regularly on the parameter $\mathbf{x} \in \mathbb{R}^d$, the convergence in Proposition 8.1 also holds for the approximate case. To see this, note that condition (8.2.3) also holds for the approximate case as long as $\mathcal{F}$ is smooth enough and the collocation conditions are solved sufficiently accurately. In practice, based on (8.3.4), it suffices to have

$$\max_{c=1,\dots,C} \left\| \left(\tilde{\mathbf{X}}'_{\theta_n^i}(\mathbf{x}, \Delta t) - \mathbf{F}\left(\tilde{\mathbf{X}}_{\theta_n^i}(\mathbf{x}, \Delta t) \right) \right)^\top \mathbf{e}_c \right\|_2 \leq \tilde{\alpha} \left(\Delta t \right)^{p+1}$$

for an $\tilde{\alpha} > 0$, and every $n = 0, \dots, N-1$ and iterate i. Furthermore, assumption (8.2.4) follows from the Lipschitz regularity of the approximate function $\hat{\theta}$.

8.6 Numerical results

This section collects several numerical tests that support our theoretical derivations. We consider six dynamical systems, four of which come from the experimental section in [14], to which we add the SIR model and the ROBER problem. We assume that, for each of these systems, a single initial value problem is of interest and explore how ELM-based coarse propagators perform for that initial value problem. For the one-dimensional Burgers' equation, we consider a semi-discretization with centered finite differences and provide the experimental results for different initial conditions, imposing homogeneous Dirichlet boundary conditions on the domain $[0, 1]$.

The chosen fine integrators are classical Runge–Kutta methods with a smaller timestep than the coarse one Δt. More explicitly, we assume that the coarse timestep Δt is a multiple of the fine timestep δt and one coarse integrator step Δt, corresponds to $\Delta t / \delta t$ steps of the size δt of the fine integrator. In all experiments, we use equispaced time collocation points, and for the Lorenz system, we also use Lobatto points. For stiff problems such as Burgers' and ROBER's, we use the implicit Euler method (IE), with update $\mathbf{x}_{n+1} = \mathbf{x}_n + \delta t \mathcal{F}(\mathbf{x}_{n+1})$, as a fine integrator, while for the others we found Runge-Kutta (RK4),

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{\delta t}{6} \left(k_1 + 2k_2 + 2k_3 + k_4 \right)$$

with

$$k_1 = \mathcal{F}\left(\mathbf{x}_n\right), \; k_2 = \mathcal{F}\left(\mathbf{x}_n + \delta t \frac{k_1}{2}\right), \; k_3 = \mathcal{F}\left(\mathbf{x}_n + \delta t \frac{k_2}{2}\right), \; k_4 = \mathcal{F}\left(\mathbf{x}_n + \delta t k_3\right).$$

to provide accurate solutions with moderately small step sizes. We specify the adopted timesteps in the dedicated sections below.

The purpose of this paper is to demonstrate that our hybrid Parareal method based on ELMs is theoretically motivated and practically effective, rather than the high-performance implementation. Thus, most of our experiments are run on a single processor where the parallel speedup would result from parallel execution of the fine integrators on the sub-intervals in proportion to the number of cores used. To demonstrate the principle in hardware we run the ROBER's problem on five processors available to us and compare to the serial application of the fine integrator, however Parareal benefits will scale up with the number of cores. For Burgers' equation, we again use five processors for convenience since this allows us to do 100 repeated experiments faster.

In all plots, the label "para" refers to the hybrid methodology with neural networks as coarse propagators, while "ref" refers to the reference solution obtained by the sequential application of the fine solver. We always plot the piecewise smooth Hybrid Parareal approximant constructed as (8.4.4). We run the Hybrid Parareal until the difference between two consecutive iterates was at most $\texttt{tol} = 10^{-4}$. As a safeguard, we put a hard limit, $\texttt{max\_it} = 20$, on the iteration number, which was, however, not triggered in any of our experiments. All experiments were run on a MacBook Pro 2020 with an Intel i5 processor, and all the computational times averaged over 100 runs per experiment. For each experiment, we report an average time per update of the coarse integrator on a sub-interval, which is also averaged over the number of sub-intervals. We measure the timing when computing the zeroth iterate in lines 3-6 of Algorithm 4 to isolate the effects of warm starts used in Parareal update in later iterations. We also report a total average time to compute the solution, including the above mentioned coarse integrator updates along with possibly parallel execution of the fine step integrators.

8.6.1 SIR

The SIR model is one of the simplest systems considered in mathematical biology to describe the spread of viral infections. SIR consists of three coupled ODEs for $\mathbf{x} = [x_1, x_2, x_3]^\top$ with parameters $\beta = 0.1$, and $\gamma = 0.1$:

$$\begin{cases} x_1'(t) = -\beta x_1(t)\, x_2(t), \\ x_2'(t) = \beta x_1(t)\, x_2(t) - \gamma x_2(t), \\ x_3'(t) = \gamma x_2(t), \\ \mathbf{x}(0) = \begin{bmatrix} 0.3 & 0.5 & 0.2 \end{bmatrix}^\top. \end{cases} \tag{8.6.1}$$

We use this example to compare two different types of coarse propagators,

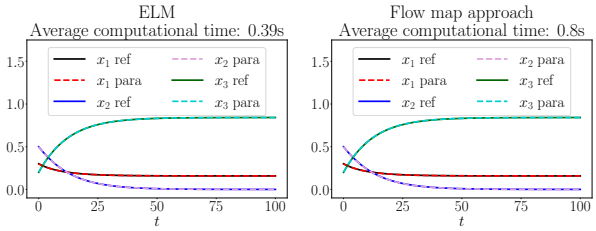


Figure 8.6.1: SIR: Hybrid Parareal solution with (left) an ELM-based coarse propagator, (right) flow map coarse propagator.

the ELM-based approach with a neural operator-type *flow map* trained to approximate the solutions of the dynamical system described by (8.6.1) for initial conditions in the compact set $\Omega = [0,1]^3$, and times in $[0,1]$, see also [12, 39]. Given that the Parareal method needs to evaluate the coarse propagator on several initial conditions, the learned flow map is the most natural neural network-based alternative, while a standard Physics Informed Neural Network, which needs to be fitted for each initial condition, would be computationally too expensive. Both coarse propagators use the same coarse timestep $\Delta t = 1$, while the fine solver timestep is $\delta t = 10^{-2}$. The piecewise smooth approximations computed with both methods are plotted in Figure 8.6.1. We report the corresponding timings in Table 8.6.1.

Timing breakdown	ELM	Flow
Offline training phase	0s	~20 minutes
Average cost coarse step in the zeroth iterate	0.0009773s	0.0002729s
Average cost to produce the solution	0.3940s	0.8047s

Table 8.6.1: SIR: Computational time for the ELM and flow map based Hybrid Parareal variants on a single core.

The ELM-based approach took an average of 0.3940 seconds to converge to the final solution over 100 repeated experiments, while the flow map approach took an average time of 0.8047 seconds. The reason behind the higher cost of the flow map approach is that ELMs minimize the residual more accurately than the flow map approach since they are trained for the specific initial conditions of interest, leading to a faster convergence of the Parareal method. If the offline training phase is accounted for, about 20 more minutes must be considered for the flow map approach, while no offline training is required for the ELM-based approach. The offline training cost depends on the chosen architecture and training strategy. These details are provided in Appendix 8.D.

Given the reported results, it is clear that while both methods are comparable in terms of accuracy, the distribution of the costs is considerably different. The

flow map approach has a high training cost and a low evaluation cost but is also less accurate, hence needing more Parareal iterations. On the other hand, the ELM strategy, having no offline training phase and yielding more accurate solutions and hence needing fewer Parareal iterations, saves substantial time. For this reason, we will only focus on the ELM-based approach in the following experiments.

8.6.2 ROBER

The ROBER problem is a prototypical stiff system of coupled ODEs with parameters $k_1 = 0.04$, $k_2 = 3 \cdot 10^7$, and $k_3 = 10^4$,

$$\begin{cases} x_1'(t) = -k_1 x_1(t) + k_3 x_2(t) x_3(t), \\ x_2'(t) = k_1 x_1(t) - k_2 x_2^2(t) - k_3 x_2(t) x_3(t), \\ x_3'(t) = k_2 x_2^2(t), \\ \mathbf{x}(0) = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^\top. \end{cases} \tag{8.6.2}$$

As ROBER's solution spikes for short times, the usual approach is to discretize the time non-uniformly. Therefore we choose the coarse step size to be $\Delta t = 10^{-2}$ for times in $[0,1]$ and $\Delta t = 3$ for times in $[1,100]$. The fine integrator timestep is $\delta t = 10^{-4}$. We remark that ROBER's problem is commonly solved using a variable step-size method, for example, based on an embedded Runge-Kutta method [16, Section II.4]. Fixing the step size allows us to understand how the proposed hybrid method performs on stiff equations without extra complication of step adaptivity. A variable step Parareal method (regardless if the coarse propagator is learned or classical), would involve adaptivity in both coarse and fine step and is beyond scope of this work.

Timing breakdown	ELM	Sequential IE
Average cost coarse step in the zeroth iterate	0.001881s	
Average cost to produce the solution	179.8280s	263.2613s

Table 8.6.2: ROBER: Computational time for Hybrid Parareal using five cores versus sequential application of IE with fine step δt.

We report the obtained approximate solutions in Figure 8.6.2 and the timings in Table 8.6.2. In these experiments, the fine integrators were executed in parallel on five cores. Thus, the total average time to compute the solution reflects the parallel speed up, albeit for a small number of cores. Given this stiff problem requires an implicit fine integrator, we expect the computational costs of the update of the coarse integrator, 0.001881s, and one step of the fine integrator, 0.000263s, to be closer than when using an explicit scheme as it is the case

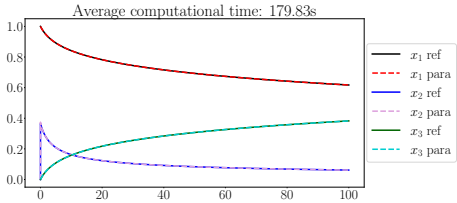


Figure 8.6.2: ROBER: Components of the Hybrid Parareal solution. To plot all components on the same scale, $\mathbf{x}_2$ was scaled by a factor of 10^4.

in the remaining examples. Additionally, to cover one coarse step, the fine integrator needs to perform at least 100 steps, given our choices for δt and Δt. These respective costs help to optimally balance the choice of the number of sub-intervals versus the number of fine steps in each sub-interval, along with practical considerations like the number of cores available.

8.6.3 Lorenz

For weather forecasts, real-time predictions are paramount, rendering parallel-in-time solvers highly relevant in this context. Lorenz's equations

$$\begin{cases} x_1'(t) = -\sigma x_1(t) + \sigma x_2(t), \\ x_2'(t) = -x_1(t)\, x_3(t) + r\, x_1(t) - x_2(t), \\ x_3'(t) = x_1(t)\, x_2(t) - b x_3(t), \\ \mathbf{x}(0) = \begin{bmatrix} 20 & 5 & -5 \end{bmatrix}^\top, \end{cases} \tag{8.6.3}$$

describe one simple model for weather prediction. Different parameter values give rise to considerably different trajectories for this system. We set $\sigma = 10$, $r = 28$, and $b = 8/3$ to have chaotic behavior. We compute an approximate solution up to time $T = 10$, using ELMs as a coarse propagator with $\Delta T = T/250$ and RK4 with step $\delta t = T/14500$ as a fine integrator.

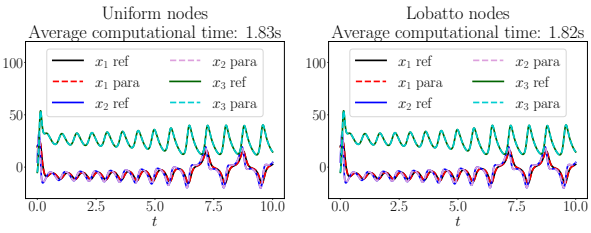


Figure 8.6.3: Lorenz: Hybrid Parareal solution with (left) uniform collocation points, (right) Lobatto collocation points.

To show that the algorithm is not overly sensitive to the choice of the collocation points, we repeated the simulations using Lobatto collocation points. The

qualitative behavior of the produced solutions for one choice of trained weights is reported in Figure 8.6.3 and the corresponding timings in Table 8.6.3. Although the Lorenz system is chaotic, the proposed hybrid solver provides an accurate approximate solution on the considered interval. Additionally, the average cost of one evaluation of the coarse ELM-based integrator does not appear to depend strongly on the system's complexity but mostly on its dimension. Indeed, the average cost of one ELM evaluation is comparable with the one for the SIR problem, see Table 8.6.1.

Timing breakdown	Uniform	Lobatto
Average cost coarse step in the zeroth iterate	0.0009430s	0.0009371s
Average cost to produce the solution	1.8312s	1.8184s

Table 8.6.3: Lorenz: Computational time for the ELM-based Hybrid Parareal with uniform and Lobatto nodes on a single core.

8.6.4 Arenstorf orbit

The three-body problem is a well-known problem in physics that pertains to the time evolution of three bodies interacting because of their gravitational forces. Changing the ratios between the masses, their initial conditions, and velocities can starkly alter the system's time evolution, and many configurations have been thoroughly studied. One of them is the stable Arenstorf orbit, which arises when one of the masses is negligible, and the other two masses orbit in a plane. The equations of motion for this specific instance of the three-body problem are

$$\begin{cases} x_1''(t) = x_1(t) + 2x_2'(t) - b\frac{x_1+a}{D_1} - a\frac{x_1'(t)-b}{D_2}, \\ x_2''(t) = x_2(t) - 2x_1'(t) - b\frac{x_2(t)}{D_1} - a\frac{x_2(t)}{D_2}, \\ \begin{bmatrix} x_1(0) & x_1'(0) & x_2(0) & x_2'(0) \end{bmatrix}^\top = \begin{bmatrix} 0.994 & 0 & 0 & v_2^0 \end{bmatrix}^\top, \end{cases} \tag{8.6.4}$$

$$D_1 = \left((x_1(t)+a)^2 + x_2(t)^2\right)^{3/2}, \quad D_2 = \left((x_1(t)-b)^2 + x_2(t)^2\right)^{3/2},$$

$v_2^0 = -2.00158510637908252240537862224$, $a = 0.12277471$, and $b = 1 - a$. This configuration leads to a periodic orbit of period 17.06521656015796 [16]. In practice, we transform (8.6.4) into a first order system via the velocity variables $v_1(t) := x_1'(t)$ and $v_2(t) := x_2'(t)$. We include the plot of the obtained solution for time up to $T = 17$ and timesteps $\Delta t = T/125$, and $\delta t = T/80000$, in Figure 8.6.4 and the timings in Table 8.6.4.

This experiment serves to illustrate the benefits of using a Parareal-like correction of the neural network-based solution. Indeed, the approximate solution

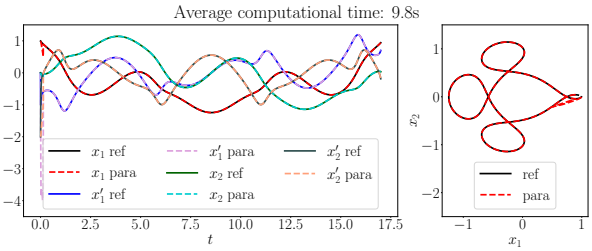


Figure 8.6.4: Arenstorf: Components of the Hybrid Parareal solution (left), and the orbit of the initial condition (right).

for short times does not accurately follow the correct trajectory. One possible remedy would be to restrict the step size Δt as was done for the ROBER's problem. However, even for this larger time step choice, after just one step Δt, the Parareal correction resets the initial condition for the next interval, bringing the solution back onto the stable orbit. Thus, not relying solely on a network-based solution allows us to compute an accurate solution for later times, even though initially, the solution departs the orbit.

Timing breakdown	ELM
Average cost coarse step in the zeroth iterate	0.001912s
Average cost to produce the solution	9.7957s

Table 8.6.4: Arenstorf: Computational time for Hybrid Parareal using a single core.

8.6.5 Viscous Burgers' equation

Most of the systems considered up to now are low-dimensional. A natural way to test the method's performance on higher-dimensional systems is to work with spatial semi-discretizations of PDEs, where the mesh over which the spatial discretization is defined determines the system's dimension. We consider the one-dimensional Burgers' equation

$$\begin{cases} \partial_t u(x,t) + u(x,t)\,\partial_x u(x,t) = \nu \partial_{xx} u(x,t), & x \in \Omega = [0,1], \\ u(x,0) = \sin(2\pi x), & x \in \Omega, \\ u(0,t) = u(1,t) = 0, & t \geq 0. \end{cases} \tag{8.6.5}$$

In this section, we only report the results for the initial condition in Equation (8.6.5), but we include results for two more choices of initial conditions in Appendix 8.F. All the experiments were run on five cores. In all tests, we work with viscosity parameter $\nu = 1/50$, a uniform spatial grid of 51 points in $\Omega = [0,1]$ and coarse and fine step sizes $\Delta t = 1/50$ and $\delta t = 1/500$, respectively.

The spatial semi-discretization with centered finite differences writes

$$\begin{cases} \mathbf{u}'(t) = -\mathbf{u}(t) \odot \left(D_1 \mathbf{u}(t)\right) + \nu D_2 \mathbf{u}(t), \\ \mathbf{u}(0) = \sin(2\pi \mathbf{x}) \in \mathbb{R}^{51}, \end{cases}$$

where $\mathbf{x} = \begin{bmatrix} x_0 & x_1 & \dots & x_{50} \end{bmatrix}^\top$, $\mathbf{x}_i = i\Delta x$, $\Delta x = 1/50$, $i = 0, \dots, 50$, $\odot$ is the component-wise product, and $D_1, D_2 \in \mathbb{R}^{51\times 51}$ are the centered finite difference matrices of first and second order, respectively, suitably corrected to impose the homogeneous Dirichlet boundary conditions on $t \mapsto \mathbf{u}(t)$.

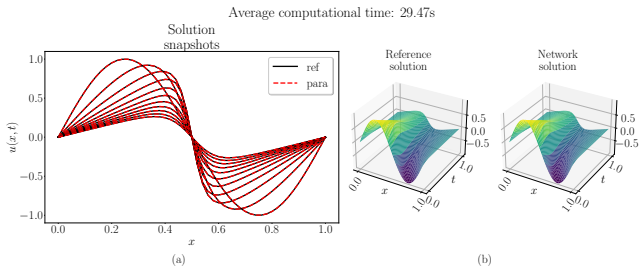


Figure 8.6.5: Burgers: Snapshots of the solution obtained with Hybrid Parareal (left), comparison of the solution surfaces between Hybrid Parareal and the fine integrator applied serially (right). Solution corresponding to $u_0(x) = \sin(2\pi x)$.

We report the qualitative behavior of the solutions in Figure 8.6.5. Subfigure (a) tracks the solution at ten equally spaced time instants in the interval $[0, T = 1]$. Subfigure (b) shows the solution surfaces obtained with the IE method on the left and the hybrid Parareal for one set of trained parameters on the right. We include the timings in Table 8.6.5. We observe that the cost of the presented hybrid Parareal method grows with the dimensionality d of the problem. However, we remark that for each of the 51 components of the solution, the displayed results were obtained with only $H = 5$ coefficients.

Timing breakdown	ELM
Average cost coarse step in the zeroth iterate	0.2098s
Average cost to produce the solution	29.4740s

Table 8.6.5: Burgers: Computational time for Hybrid Parareal using five cores, and initial condition $u_0(x) = \sin(2\pi x)$.

8.7 Conclusions and future extensions

In this manuscript, we proposed a hybrid parallel-in-time algorithm to solve initial value problems using a neural network as a coarse propagator within

the Parareal framework. We derived an a-posteriori error estimate for generic neural network-based approximants. Based on these theoretical results, we defined a hybrid Parareal algorithm involving ELMs as coarse propagators which inherits the theoretical guarantees of the Parareal algorithm.

We compared our hybrid Parareal solver based on ELMs with one based on the flow map approach on the SIR problem. We demonstrated that our approach led to lower computational costs and no offline training phase. We reserve the judgment of flow map performance. However, we also tested it for other examples, including the Brusselator, where we noticed that the offline training phase can be very intricate. This is due to the fact that the flow map approach relies on the existence of a forward invariant subset Ω of $\mathbb{R}^d$, or equivalently for every $\mathbf{x}_0 \in \Omega$, also $\mathbf{x}(t) \in \Omega$ for all $t \in [0, \Delta t]$ where $\mathbf{x}(0) = \mathbf{x}_0$.

The most promising extension of this work is to include a mechanism allowing for time-adaptivity in the algorithm, i.e., for coarsening or refinement of the temporal grid based on the local behavior of the solution. It would also be interesting to test our approach on higher-dimensional systems with high-performance computing hardware.

Appendix

8.A A-posteriori error estimate based on the defect

We now derive an alternative a-posteriori estimate for network-based approximate solutions based on defect control.

Lemma 8.1. Consider the initial value problem (8.1.1), given by

$$\begin{cases} \mathbf{x}'(t) = \mathcal{F}\big(\mathbf{x}(t)\big), \\ \mathbf{x}(0) = \mathbf{x}_0, \end{cases}$$

where $\mathcal{F}: \mathbb{R}^d \to \mathbb{R}^d$ is continuously differentiable and admits a unique solution. Let $\mathbf{y}: \mathbb{R} \to \mathbb{R}^d$ satisfy

$$\begin{cases} \mathbf{y}'(t) = \mathcal{F}\big(\mathbf{y}(t)\big) + \mathbf{d}(t), \quad \mathbf{d}: \mathbb{R} \to \mathbb{R}^d, \\ \mathbf{y}(0) = \mathbf{x}_0. \end{cases}$$

Then $\mathbf{z}(t) := \mathbf{y}(t) - \mathbf{x}(t)$ satisfies the linear differential equation

$$\begin{cases} \mathbf{z}'(t) = A(t)\mathbf{z}(t) + \mathbf{d}(t), \\ \mathbf{z}(0) = 0, \end{cases} \tag{8.A.1}$$

where

$$A(t) = \int_0^1 \mathrm{D}\mathcal{F}(\mathbf{x}(t) + s\mathbf{z}(t))\mathrm{d}s$$

and $\mathrm{D}\mathcal{F}$ is the Jacobian matrix of $\mathcal{F}$.

Proof. To prove the lemma, it suffices to highlight that

$$\mathcal{F}\big(\mathbf{y}(t)\big) - \mathcal{F}\big(\mathbf{x}(t)\big) = \int_0^1 \frac{d}{ds}\mathcal{F}\big(\mathbf{x}(t) + s\mathbf{z}(t)\big)\,\mathrm{d}s = A(t)\mathbf{z}(t).$$

□

The solution to the linear problem (8.A.1) satisfies the following bound:

Lemma 8.2 (Theorem 1 in [38])**.** Let $\mathbf{z}(t)$ solve the initial value problem in (8.A.1). Suppose that $\|\mathbf{d}(t)\|_2 \leq \varepsilon$ for $t \geq 0$. Then,

$$\|\mathbf{z}(t)\|_2 \leq \varepsilon \int_0^t \exp\left(\int_s^t \mu_2\left(A(\tau)\right) \mathrm{d}\tau\right) \mathrm{d}s,$$

where

$$\mu_2(A) = \lambda_{\max}\left(\frac{A + A^\top}{2}\right)$$

is the logarithmic 2−norm of A.

For the proof of this lemma, see [17, Theorem 10.6].

As we are interested in solving (8.1.1), we set $\mathbf{y}(t) := \mathcal{N}_\theta\left(t; \mathbf{x}_0\right)$ and introduce the defect function

$$\mathbf{d}\,(t) := \mathcal{N}_\theta'\left(t; \mathbf{x}_0\right) - \mathcal{F}\left(\mathcal{N}_\theta\left(t; \mathbf{x}_0\right)\right).$$

We remark that the definition of $\mathbf{d}$ is of the same form as the loss (8.3.1). If it was known that $\|\mathbf{d}(t)\|_2 \leq \varepsilon$ for a tolerance $\varepsilon > 0$ and all $t \in [0, \Delta t]$, then by Lemma 8.2 we could conclude that

$$\left\|\mathbf{x}(t) - \mathcal{N}_\theta\left(t; \mathbf{x}_0\right)\right\|_2 \leq \varepsilon \int_0^t \exp\left(\int_s^t \mu_2\left(A(\tau)\right) \mathrm{d}\tau\right) \mathrm{d}s, \quad t \in \left[0, \Delta t\right].$$

Given that the solution $\mathbf{x}(t)$ is unknown, $A(\tau)$ and its logarithmic norm cannot be computed exactly. Thus, for a more practical error estimate, we introduce an assumption on the existence of a compact subset $\Omega \subset \mathbb{R}^d$ such that $\mathbf{x}(t) + s\mathbf{z}(t) \in \Omega$ for $(s, t) \in [0, 1] \times [0, \Delta t]$. Then, we can proceed with the inequality chain as

$$\left\|\mathbf{x}(t) - \mathcal{N}_\theta\left(t; \mathbf{x}_0\right)\right\|_2 \leq \varepsilon \int_0^t e^{M(t-s)} \mathrm{d}s = \varepsilon \frac{e^{Mt} - 1}{M}, \tag{8.A.2}$$

where $M := \max_{\mathbf{z} \in \Omega} \mu_2(\mathrm{D}\mathcal{F}(\mathbf{z})) \in \mathbb{R}$. Note that the right-hand side of (8.A.2) is nonnegative for all $t \geq 0$. In particular, (8.A.2) implies that a neural network $\mathcal{N}_\theta$ can be employed to approximate the solution of (8.1.1) which is as accurate as a classical coarse solver $\varphi_C^{\Delta t}$ provided the norm of the defect $\|\mathbf{d}(t)\|_2$ is sufficiently small.

8.B Bound on the norm of the sensitivity matrix

In this appendix, we provide a practical bound for the norm of the Jacobian of the flow map of a vector field $\mathcal{F}$, assumed to be continuously differentiable

with respect to the initial condition. For this, we differentiate the initial value problem (8.1.1), given by

$$\begin{cases} \frac{d}{dt}\phi_{\mathcal{F}}^{s,t}(\mathbf{x}_0) = \mathcal{F}\left(\phi_{\mathcal{F}}^{s,t}(\mathbf{x}_0)\right) \in \mathbb{R}^d, \\ \phi_{\mathcal{F}}^{s,s}(\mathbf{x}_0) = \mathbf{x}_0, \end{cases} \tag{8.B.1}$$

with respect to $\mathbf{x}_0$ and obtain

$$\begin{cases} \frac{d}{dt}\left(\frac{\partial \phi_{\mathcal{F}}^{s,t}(\mathbf{x}_0)}{\partial \mathbf{x}_0}\right) = \mathrm{D}\mathcal{F}\left(\phi_{\mathcal{F}}^{s,t}(\mathbf{x}_0)\right)\frac{\partial \phi_{\mathcal{F}}^{s,t}(\mathbf{x}_0)}{\partial \mathbf{x}_0} \in \mathbb{R}^{d\times d}, \\ \frac{\partial \phi_{\mathcal{F}}^{s,s}(\mathbf{x}_0)}{\partial \mathbf{x}_0} = I_d, \end{cases} \tag{8.B.2}$$

where $I_d \in \mathbb{R}^{d\times d}$ is the identity matrix. Equation (8.B.2) is generally known as the variational equation of (8.B.1). This ODE is a non-autonomous linear differential equation in the unknown matrix $\partial_{\mathbf{x}_0}\phi_{\mathcal{F}}^{s,t}(\mathbf{x}_0)$. In practice, (8.B.2) should be solved jointly with (8.B.1). However, for the purpose of bounding the Euclidean norm $\|\partial_{\mathbf{x}_0}\phi_{\mathcal{F}}^{s,t}(\mathbf{x}_0)\|_2$, it is not necessary to solve them. Following [6, Chapter 2], we assume that $\phi_{\mathcal{F}}^{s,t}(\mathbf{x}_0) \in \Omega$ for $\Omega \subset \mathbb{R}^d$ compact and all $0 \le s \le t \le \Delta t$. This is not a restrictive assumption on compact time intervals given the assumed regularity for $\mathcal{F}$. Then, one can get

$$\begin{aligned} \left\|\partial_{\mathbf{x}_0}\phi_{\mathcal{F}}^{s,t}(\mathbf{x}_0)\right\|_2 &\le \left\|\partial_{\mathbf{x}_0}\phi_{\mathcal{F}}^{s,s}(\mathbf{x}_0)\right\|_2 \exp\left(\int_s^t \mu_2\left(\mathrm{D}\mathcal{F}\left(\phi_{\mathcal{F}}^{s,s'}(\mathbf{x}_0)\right)\right)\mathrm{d}s'\right) \\ &= \exp\left(\int_s^t \mu_2\left(\mathrm{D}\mathcal{F}\left(\phi_{\mathcal{F}}^{s,s'}(\mathbf{x}_0)\right)\right)\mathrm{d}s'\right) \le \exp\left(M\Delta t\right), \end{aligned}$$

where $M = \max_{\mathbf{z}\in\Omega}\mu_2(D\mathcal{F}(\mathbf{z}))$. We conclude that the constant δ in the proof of Theorem 8.2 can be set to $\exp(M\Delta t)$, with M positive or negative depending on $\mathcal{F}$.

8.C The Jacobian matrix of the loss function

In this subsection, we consider the loss function (8.5.1) and its gradient. Note that (8.5.1) can be expressed as (8.3.1), which in turn can be related to the solution of the nonlinear matrix equation

$$\tilde{\mathbf{X}}'_\theta(\mathbf{x}, \Delta t) = \mathbf{F}\left(\tilde{\mathbf{X}}_\theta(\mathbf{x}, \Delta t)\right).$$

More explicitly, we have

$$\begin{aligned} &\tilde{\mathbf{X}}_\theta(\mathbf{x}, \Delta t) = \mathbf{1}_C\mathbf{x}^\top + \left(\mathbf{H} - \overline{\mathbf{H}}\right)\theta,\ \mathbf{1}_C = \begin{bmatrix} 1 & \dots & 1 \end{bmatrix}^\top \in \mathbb{R}^C, \\ &\tilde{\mathbf{X}}'_\theta(\mathbf{x}, \Delta t) = \mathbf{H}'\theta. \end{aligned}$$

To minimize the loss function (8.3.1), we need the Jacobian of the matrix-valued function $\mathbf{G}_\theta(\mathbf{x}, \Delta t) = \tilde{\mathbf{X}}'_\theta(\mathbf{x}, \Delta t) - \mathbf{F}(\tilde{\mathbf{X}}_\theta(\mathbf{x}, \Delta t))$. As $\mathbf{G}_\theta$ is a matrix-valued function with matrix inputs, we rely on the vectorization operator, denoted by vec, using the machinery of matrix-calculus introduced, for example, in [28]. We hence compute $\frac{\partial \mathrm{vec}(\mathbf{G}_\theta(\mathbf{x},\Delta t))}{\partial \mathrm{vec}(\theta)} \in \mathbb{R}^{Cd \times Hd}$, given by

$$\begin{aligned}
\frac{\partial \mathrm{vec}\left(\mathbf{G}_\theta\left(\mathbf{x}, \Delta t\right)\right)}{\partial \mathrm{vec}\left(\theta\right)} &= I_d \otimes \mathbf{H}' - \frac{\partial \mathrm{vec}\left(\mathbf{F}\left(\tilde{\mathbf{X}}_\theta\left(\mathbf{x}, \Delta t\right)\right)\right)}{\partial \mathrm{vec}\left(\theta\right)} \\
&= I_d \otimes \mathbf{H}' - \frac{\partial \mathrm{vec}\left(\mathbf{F}\left(\mathbf{X}\right)\right)}{\partial \mathrm{vec}\left(\mathbf{X}\right)}\Big|_{\mathbf{X}=\tilde{\mathbf{X}}_\theta(\mathbf{x},\Delta t)} \frac{\partial \mathrm{vec}\left(\tilde{\mathbf{X}}_\theta\left(\mathbf{x}, \Delta t\right)\right)}{\partial \mathrm{vec}\left(\theta\right)} \\
&= I_d \otimes \mathbf{H}' - \frac{\partial \mathrm{vec}\left(\mathbf{F}\left(\mathbf{X}\right)\right)}{\partial \mathrm{vec}\left(\mathbf{X}\right)}\Big|_{\mathbf{X}=\tilde{\mathbf{X}}_\theta(\mathbf{x},\Delta t)} \left(I_d \otimes \left(\mathbf{H} - \overline{\mathbf{H}}\right)\right),
\end{aligned}$$

where $I_d \in \mathbb{R}^{d \times d}$ is the identity matrix, $\otimes$ is the Kronecker product, and vec stacks the columns of the input matrix into a column vector. The Jacobian of $\mathbf{F}$ in the last line depends on the vector field $\mathcal{F}$, while the other terms do not.

Most of the dynamical systems we consider in the numerical experiments in Section 8.6 are of low dimension. For this reason, for all the cases but Burgers' equation, we assemble the Jacobian case by case, following this construction. For Burgers' equation, we instead implement it as a linear operator, specifying its action and the action of its transpose onto input vectors. For the Burgers' equation, we have

$$\mathcal{F}(\mathbf{u}) = -\mathbf{u} \odot \left(\mathbf{D}_1 \mathbf{u}\right) + \nu \mathbf{D}_2 \mathbf{u} \in \mathbb{R}^d,$$

and hence $\mathbf{F}(\mathbf{X}) = -\mathbf{X} \odot (\mathbf{X}\mathbf{D}_1^\top) + \nu \mathbf{X}\mathbf{D}_2^\top \in \mathbb{R}^{C \times d}$. This expression implies that

$$\frac{\partial \mathrm{vec}\left(\mathbf{F}\left(\mathbf{X}\right)\right)}{\partial \mathrm{vec}\left(\mathbf{X}\right)} = -\mathrm{diag}\left(\mathrm{vec}\left(\mathbf{X}\mathbf{D}_1^\top\right)\right) - \mathrm{diag}\left(\mathrm{vec}\left(\mathbf{X}\right)\right)\left(\mathbf{D}_1 \otimes I_C\right) + \nu \mathbf{D}_2 \otimes I_C.$$

8.D Details on the network for the flow map approach

In this section, we provide details on the network for the flow map approach required for the comparison of the training costs presented in Table 8.6.1. The network used for the coarse propagator is based on the parametrization

$$\begin{aligned}
\mathbf{z} := \left[\mathbf{x}_0^\top, t\right]^\top &\mapsto \tanh\left(\mathbf{A}_0 \mathbf{z} + \mathbf{a}_0\right) =: \mathbf{h}_1 \in \mathbb{R}^{10}, \\
\mathbf{h}_\ell &\mapsto \tanh\left(\mathbf{A}_\ell \mathbf{h}_\ell + \mathbf{a}_\ell\right) =: \mathbf{h}_{\ell+1} \in \mathbb{R}^{10},\ \ell = 1, \cdots, 4, \\
\mathbf{h}_5 &\mapsto \mathbf{x}_0 + \left(1 - e^{-t}\right) \mathbf{P}\mathbf{h}_5 =: \mathcal{N}_\theta\left(t; \mathbf{x}_0\right) \in \mathbb{R}^3,
\end{aligned}$$

where $\theta = \{\mathbf{A}_\ell, \mathbf{a}_\ell, \mathbf{P}\}_{\ell=0}^4$. To train the network, implemented with PyTorch, we use the Adam optimizer for 10^5 epochs, with each epoch consisting of minimizing the ODE residual over 500 different randomly sampled collocation points $(t^i, \mathbf{x}_0^i) \in [0,1] \times [0,1]^3$.

8.E Experiment for Brusselator's equation

This section collects numerical experiments for the Brusselator, which is a system of two scalar differential equations modeling a chain of chemical reactions [1]. The equations write

$$\begin{cases} x_1'(t) = A + x_1^2(t)\, x_2(t) - \big(B+1\big)\, x_1(t), \\ x_2'(t) = B x_1(t) - x_1^2(t)\, x_2(t), \\ \mathbf{x}(0) = \begin{bmatrix} 0 & 1 \end{bmatrix}^\top, \end{cases} \tag{8.E.1}$$

where we choose the parameters $A = 1$, $B = 3$. In this setting, one can prove to have a limit cycle in the dynamics. We simulate this system on the time interval $[0, T = 12]$, with a fine timestep $\delta t = T/640$ and a coarse one of size $\Delta T = T/32$. We repeat the simulation 100 times, reporting the average cost

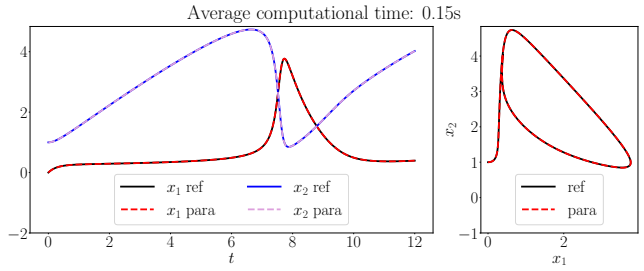


Figure 8.E.1: Brusselator: Components of the Hybrid Parareal solution (left), and the orbit of the initial condition (right).

of one coarse timestep in Table 8.E.1, together with the average total cost of the hybrid Parareal solver. Figure 8.E.1 shows the approximate solution and a reference solution. We also remark that, as desired, the hybrid method recovers the limit cycle.

Timing breakdown	ELM
Average cost coarse step in the zeroth iterate	0.001012s
Average cost to produce the solution	0.1469s

Table 8.E.1: Brusselator: Computational time for Hybrid Parareal using a single core.

8.F Additional experiments for Burgers' equation

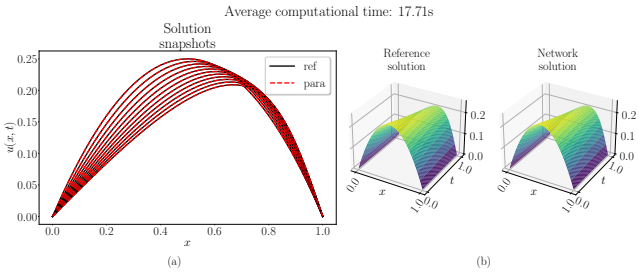


Figure 8.F.1: Burgers: Snapshots of the solution obtained with Hybrid Parareal (left), comparison of the solution surfaces between Hybrid Parareal and the fine integrator applied serially (right). Solution corresponding to $u_0(x) = x(1-x)$.

Timing breakdown	ELM
Average cost coarse step in the zeroth iterate	0.1695s
Average cost to produce the solution	17.7069s

Table 8.F.1: Burgers: Computational time for Hybrid Parareal using five cores, and initial condition $u_0(x) = x(1-x)$.

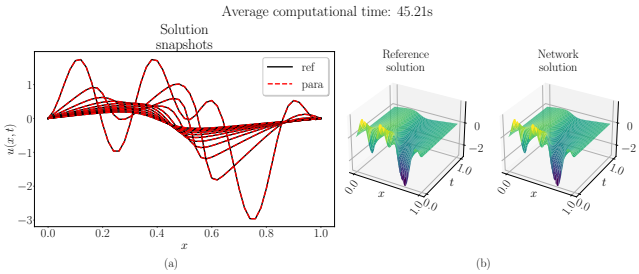


Figure 8.F.2: Burgers: Snapshots of the solution obtained with Hybrid Parareal (left), comparison of the solution surfaces between Hybrid Parareal and the fine integrator applied serially (right). Solution corresponding to $u_0(x) = \sin(2\pi x) + \cos(4\pi x) - \cos(8\pi x)$.

Timing breakdown	ELM
Average cost coarse step in the zeroth iterate	0.3356s
Average cost to produce the solution	45.2056s

Table 8.F.2: Burgers: Computational time for Hybrid Parareal using five cores, and initial condition $u_0(x) = \sin(2\pi x) + \cos(4\pi x) - \cos(8\pi x)$.

In this section, we report the simulation results for the Burgers' equation with two more initial conditions. The setup of the network and the partition of the time domain are the same as for the initial condition included in Section 8.6.5. In Figure 8.F.1, we work with the initial condition $u_0(x) = x(1-x)$, while in

Figure 8.F.2 with $u_0(x) = \sin(2\pi x) + \cos(4\pi x) - \cos(8\pi x)$. The timings are included in Tables 8.F.1 and 8.F.2, respectively. As expected, the time to obtain the full solution grows with the complexity of the initial condition. Indeed, there are about 10 seconds of difference between the fastest, corresponding to the quadratic initial condition in Figure 8.F.2, to the second fastest, the one with $u_0(x) = \sin(2\pi x)$, and the slowest in Figure 8.F.2. The reason behind this observed behavior is that, for more complicated solutions, the coarse predictions need to be corrected with the Parareal correction step more often, and the optimization problems to solve to get the coarse propagator get more expensive.

Bibliography

[1] Shaun Ault and Erik Holmgreen. Dynamics of the Brusselator. *Math 715 Projects (Autumn 2002)*, 2, 2003. 311

[2] Mary Ann Branch, Thomas F Coleman, and Yuying Li. A Subspace, Interior, and Conjugate Gradient Method for Large-Scale Bound-Constrained Minimization Problems. *SIAM Journal on Scientific Computing*, 21(1):1–23, 1999. 294

[3] F. Bullo. *Contraction Theory for Dynamical Systems*. Kindle Direct Publishing, 1.1 edition, 2023. 21, 95, 98, 110, 119, 295, 296

[4] Mario De Florio, Enrico Schiassi, and Roberto Furfaro. Physics-informed neural networks and functional interpolation for stiff chemical kinetics. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 32(6), 2022. 252, 284, 294

[5] Tim De Ryck and Siddhartha Mishra. Error analysis for physics-informed neural networks (PINNs) approximating Kolmogorov PDEs. *Advances in Computational Mathematics*, 48(6):79, 2022. 283

[6] Charles A Desoer and Mathukumalli Vidyasagar. *Feedback Systems: Input–Output Properties*. SIAM, 2009. 309

[7] Nathan Doumèche, Gérard Biau, and Claire Boyer. Convergence and error analysis of PINNs. *arXiv preprint arXiv:2305.01240*, 2023. 283

[8] Vikas Dwivedi and Balaji Srinivasan. Physics Informed Extreme Learning Machine (PIELM)–A rapid method for the numerical solution of partial differential equations. *Neurocomputing*, 391:96–118, 2020. 284

[9] Matthew Emmett and Michael L. Minion. Toward an efficient parallel in time method for partial differential equations. *Communications in Applied Mathematics and Computational Science*, 7(1):105–132, 2012. 283

[10] Gianluca Fabiani, Evangelos Galaris, Lucia Russo, and Constantinos Siettos. Parsimonious physics-informed random projection neural networks for initial value problems of ODEs and index-1 DAEs. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 33(4):043128, 2023. 252, 284

[11] R. D. Falgout, S. Friedhoff, Tz. V. Kolev, S. P. Maclachlan, and J. B. Schroder. Parallel Time Integration with Multigrid. *SIAM Journal on Scientific Computing*, 36(6):c635–c661, 2014. 283

[12] Cedric Flamant, Pavlos Protopapas, and David Sondak. Solving Differential Equations Using Neural Network Solution Bundles. *arXiv preprint arXiv:2006.14372*, 2020. 300

[13] Martin J. Gander. 50 Years of Time Parallel Time Integration. In Thomas Carraro, Michael Geiger, Stefan Körkel, and Rolf Rannacher, editors, *Multiple Shooting and Time Domain Decomposition Methods*, pages 69–113, Cham, 2015. Springer International Publishing. 283

[14] Martin J. Gander and Ernst Hairer. Nonlinear Convergence Analysis for the Parareal Algorithm. In Ulrich Langer, Marco Discacciati, David E. Keyes, Olof B. Widlund, and Walter Zulehner, editors, *Domain Decomposition Methods in Science and Engineering XVII*, pages 45–56, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. 31, 283, 284, 286, 287, 298

[15] Martin J. Gander and Stefan Vandewalle. Analysis of the Parareal Time-Parallel Time-Integration Method. *SIAM Journal on Scientific Computing*, 29(2):556–578, 2007. 283

[16] Ernst Hairer, Syvert P Nørsett, and Gerhard Wanner. *Solving Ordinary Differential Equations I, Nonstiff Problems*. Springer-Vlg, 1993. 2, 282, 288, 289, 301, 303

[17] Ernst Hairer, Syvert P Nørsett, and Gerhard Wanner. *Solving Ordinary Differential Equations I, Nonstiff Problems*. Springer-Vlg, 1993. 308

[18] Gao Huang, Guang-Bin Huang, Shiji Song, and Keyou You. Trends in extreme learning machines: A review. *Neural Networks*, 61:32–48, 2015. 283

[19] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1-3):489–501, 2006. 283, 284, 295

[20] Abdul Qadir Ibrahim, Sebastian Götschel, and Daniel Ruprecht. Parareal with a Physics-Informed Neural Network as Coarse Propagator. In José Cano, Marios D. Dikaiakos, George A. Papadopoulos, Miquel Pericàs, and Rizos Sakellariou, editors, *Euro-Par 2023: Parallel Processing*, pages 649–663, Cham, 2023. Springer Nature Switzerland. 283

[21] Bangti Jin, Qingle Lin, and Zhi Zhou. Learning Coarse Propagators in Parareal Algorithm. arXiv preprint arXiv:2311.15320, 2023. 283

[22] George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021. 283

[23] Gitta Kutyniok, Philipp Petersen, Mones Raslan, and Reinhold Schneider. A Theoretical Analysis of Deep Neural Networks and Parametric PDEs. *Constructive Approximation*, 55(1):73–125, 2022. 283

[24] Henning Lange, Steven L. Brunton, and J. Nathan Kutz. From Fourier to Koopman: Spectral Methods for Long-term Time Series Prediction. *J. Mach. Learn. Res.*, 22(1), jan 2021. 283

[25] Youngkyu Lee, Jongho Park, and Chang-Ock Lee. Parareal Neural Networks Emulating a Parallel-in-Time Algorithm. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–12, 2022. 283

[26] J.-L. Lions, Yvon Maday, and Gabriel Turinici. A "parareal" in time discretization of PDE's. *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, 332:661–668, 2001. 283, 285

[27] Yuying Liu, J. Nathan Kutz, and Steven L. Brunton. Hierarchical deep learning of multiscale differential equation time-steppers. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 380(2229):20210200, 2022. 283

[28] Jan R Magnus and Heinz Neudecker. *Matrix Differential Calculus with Applications in Statistics and Econometrics*. John Wiley & Sons, 2019. 310

[29] Siddhartha Mishra and Roberto Molinaro. Estimates on the generalization error of physics-informed neural networks for approximating a class of inverse problems for PDEs. *IMA Journal of Numerical Analysis*, 42(2):981–1022, 2022. 283

[30] Daniele Mortari, Hunter Johnston, and Lidia Smith. High accuracy least-squares solutions of nonlinear differential equations. *Journal of computational and applied mathematics*, 352:293–307, 2019. 252, 284

[31] Jorge Nocedal and Stephen J Wright. *Numerical Optimization*. Springer, 1999. 252, 294

[32] Joost AA Opschoor, Philipp C Petersen, and Christoph Schwab. Deep ReLU networks and high-order finite element methods. *Analysis and Applications*, 18(05):715–770, 2020. 283

[33] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical Mathematics*, volume 37. Springer Science & Business Media, 2006. 252, 266, 288

[34] Ali Rahimi and Benjamin Recht. Uniform Approximation of Functions with Random Bases. In *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, pages 555–561, 2008. 284

[35] Ali Rahimi and Benjamin Recht. Weighted Sums of Random Kitchen Sinks: Replacing minimization with randomization in learning. *Advances in neural information processing systems*, 21, 2008. 284

[36] F. Regazzoni, L. Dedè, and A. Quarteroni. Machine learning for fast and reliable solution of time-dependent differential equations. *Journal of Computational Physics*, 397:108852, 2019. 283

[37] Enrico Schiassi, Mario De Florio, Andrea D'Ambrosio, Daniele Mortari, and Roberto Furfaro. Physics-Informed Neural Networks and Functional Interpolation for Data-Driven Parameters Discovery of Epidemiological Compartmental Models. *Mathematics*, 9(17):2069, 2021. 284

[38] Gustaf Söderlind. On nonlinear difference and differential equations. *BIT Numerical Mathematics*, 24:667–680, 1984. 308

[39] Sifan Wang and Paris Perdikaris. Long-time integration of parametric evolution equations with physics-informed deeponets. *Journal of Computational Physics*, 475:111855, 2023. 30, 267, 300

ISBN 978-82-326-8318-5 (printed ver.)
ISBN 978-82-326-8317-8 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

NTNU

Norwegian University of
Science and Technology