Scuola Superiore Meridionale
*University of Naples Federico II*


Doctoral Thesis
*in Modelling and Engineering Risk and Complexity*

# Machine learning-based Modelling and Numerical Analysis of the Emergent Dynamics of Complex Systems

*Author:*
Gianluca Fabiani

*Supervisor:*
Prof. Constantinos Siettos
*Co-advisor:*
Prof. Ioannis G. Kevrekidis

*Submitted in fulfillment of the requirements for the degree of Doctor of Philosophy in Modelling and Engineering Risk and Complexity. Coordinator: Prof. Mario di Bernardo.*


*December 11, 2024*

*To my beloved wife,*
*whose invaluable support, love, and*
*encouragement have been my constant*
*source of strength throughout this journey.*

*To my son,*
*whose arrival has filled my life with*
*immeasurable joy and inspiration.*
*May your future be as bright as the hope*
*you bring into my heart.*

*With all my love,*
*this work is dedicated to you both.*

# Abstract

The aim of this Doctoral Thesis is the development of new methodologies at the intersection of *machine learning* (ML) and *numerical analysis* to address key challenges in modeling, understanding, and forecasting the emergent dynamics of complex systems. In particular, by integrating these two fields, we overcome key limitations of traditional and deep learning approaches linked to the "curse of dimensionality" in their training, even for simple problems. The core focus of the Thesis lies in solving both forward and inverse problems, while also providing new theoretical and practical contributions to the mathematical foundations of ML.

A critical aspect of this research focuses on the advancement of ML methodologies to study complex systems, thus dealing with the so-called "curse of dimensionality". In particular, we focus on Random Projection Neural Networks (RPNNs), which relax the "curse of dimensionality" and the challenges in finding global optima for non-convex training process of fully-trained Feedforward Neural Networks (FNNs) for dynamical systems. Throughout this Thesis, we have theoretically analyzed the RPNNs for function approximation, demonstrating their exponential convergence rate for smooth functions. The Thesis has two branches: the theoretical advances and their applications for complex systems. Of course, these two are in dialectic relation. Applications drive the development of new methods, and new methods provide tools for understanding better the behavior of such systems, and so forth and so on.

From the theoretical point of view, we investigated and developed various methods for sampling the internal weights and biases of RPNNs, employing both function-agnostic and function-informed approaches while developing a geometric-based sampling algorithm for high-dimensional problems. Furthermore, we have introduced Random Projection-based Operator Networks (RandONets), which extend the use of random projections for Deep Operator Networks (DeepONets). We theoretically prove for RandONets an extension of the Chen and Chen's theorem of universal approximation for operators and empirically demonstrate that randomized and a priori fixed embeddings of both branch and trunk hidden layers can outperform traditional fully-trained DeepONets counterparts by several orders of magnitude in both computational time and numerical accuracy.

The above enhance our arsenal to solve both the forward and inverse problem in complex systems. The forward problem focuses on the numerical solution of differential equations (DEs) for complex systems and their associated challenging issues, such as

those involving stiff systems and multiscale simulations, as well as analyzing system stability through bifurcation analysis, deeply connected with the analysis of multi-stability phenomena. We rigorously evaluate ML methods, particularly Physics-Informed RPNNs (PIRPNNs), against established numerical techniques. Our results show that parsimoniously designed PIRPNNs can match or outperform traditional numerical methods and deep learning approaches in accuracy and efficiency, especially in handling stiff systems and sharp gradients. In these forward problems, by exploiting analytical derivatives of the network, solving with RPNNs reduces to finding least-squares solutions of nonlinear algebraic equations. To enhance training efficiency, we propose replacing computationally expensive gradient-based algorithms with Newton's method, combined with specialized techniques such as Moore-Penrose pseudo-inverses based on singular value decomposition (SVD) or Complete Orthogonal Decomposition (COD). We stress, that the developing of such novel algorithms and training strategies, employing for example random projections, aim to enhance the reliability and efficiency of ML for complex systems modeling. For illustration purposes, we have applied these PIRPNNs to stationary Partial Differential Equations (PDEs), including the 1D and 2D Liouville-Bratu-Gelfand and viscous Burgers' equations with mixed boundary conditions. In addition, we coupled this solution with arc-length continuation in the RPNN weight space to construct accurate bifurcation diagrams. In these experiments, we outperformed established techniques, such as finite difference (FD) and finite element methods (FEM), for the first time. Our research also extended to solving Ordinary Differential Equations (ODEs) and Differential Algebraic Equations (DAEs), including various stiff benchmark problems and stiff PDEs, including van der Pol ODEs, Robertson DAEs, Belousov-Zhabotinsky ODEs and Allen-Cahn phase field PDE discretized with FD. Notably, we surpassed a professional Matlab solver, ode15s, which is based on the backward differentiation formula (BDF) of variable order and variable step sizes.

On the other hand, the solution of the inverse problem (reconstruction of differential equations from data), is crucial for the parameter identification, system reconstruction, and discovering hidden dynamics in complex systems. Inverse problems often pose additional challenges, such as ill-posedness and sensitivity to noise, requiring robust and efficient approaches. The primary goal here is to infer underlying properties of the differential equations from observed data—specifically, identifying the macroscopic governing laws of emergent behaviors. This involves constructing ML-assisted surrogate models in the form of DEs, possibly identifying the presence of stochastic terms (SDEs), partial derivatives (PDEs), and/or integral features (IPDEs). To achieve this, a significant challenge lies in identifying coarse-grained observables that accurately describe the emergent behavior. This step is critical for capturing the key dynamics of the system while reducing the complexity and high-dimensionality of the underlying data. We employ manifold learning techniques to systematically extract relevant macroscopic observables from high-dimensional data, ensuring that these observables are both significant and representative of the system's emergent behavior. Based on the selected significant coarse-scale observables, we learn the right-hand-side (RHS) of the effective DEs using two ML schemes, namely FNNs and RPNNs. To handle challenges related to dominant spatial features, we combine these neural architectures with fixed convolutional layers

based on finite difference stencils, which can provide new feature candidates. We further refine the selection of these features through parsimonious Diffusion Maps (DMaps) and Automatic Relevance Determination (ARD) with Gaussian Processes (GPs). We contrast the advantages and limitations of different ML surrogate models, emphasizing the importance of developing task-oriented models. Specifically, we highlight the trade-off between the computational endeavor in learning infinite-dimensional PDEs operators and the simplicity of low-dimensional targeted surrogates, in the form of SDEs, particularly for tasks such as identifying tipping points (bifurcation points in dynamical systems) and performing rare event analysis in their neighborhood. In particular, for this task, we utilize both brute force Monte Carlo simulations and statistical mechanics-based methods. For the task-oriented SDE surrogate, we employ two coupled FNNs inspired by the Euler-Maruyama scheme to simultaneously learn the drift and diffusivity functions of an underlying Wiener process, based on the assumption that the data follow such a process. Notably, these low-dimensional surrogates accelerate rare event analysis by three orders of magnitude compared to full-scale microscopic simulations. This thesis presents three illustrative case studies for inverse problems: (i) the spatial propagation of action potentials in unmyelinated neurons, modeled by the FitzHugh-Nagumo PDEs, reconstructed from a mesoscopic lattice Boltzmann description; (ii) an event-driven agent-based model capturing mimesis-driven behavior; and (iii) the spread of an epidemic on an Erdős-Rényi network.

Ultimately, this thesis contributes to the advancement of both ML and numerical analysis by providing novel insights into their synergistic potential. The proposed methodologies offer promising avenues for tackling complex systems challenges across a wide range of applications, such as neuroscience, finance, and epidemiology.

# Acknowledgements

This Ph.D. journey has been a complex system in itself – a thrilling exploration marked by many tipping points, rare events, and intricate patterns hidden within the Ph.D. path. But through it all, the support of some truly amazing people kept me converging towards that sweet spot of stability.

First and foremost, I extend my profound gratitude to Professor Constantinos Siettos for being more than just a supervisor; you've been a true guide and friend. Thanks for your unconditional availability (even at midnight and beyond) and extraordinary support. Equally important is my gratitude to Professor Ioannis G. Kevrekidis, my co-advisor. A true luminary, whose inexhaustible wellspring of innovative ideas has, does, and will continue to shape the future of complexity and ML science. Your guidance throughout this research was instrumental, your passion and expertise have been an inspiration. I would like also to thank Professor Mario Di Bernardo as Coordinator of the Ph.D. program in MERC for all of his efforts for the organization of the doctoral program, his availability and help during all this period.

A special thanks to Professor Athanasios N. Yannacopoulos for his collaboration and valuable inputs on several projects. I would also like to extend my sincere thanks to Dr. Lucia Russo for her kindness, support, and collaboration. A big thanks to Hector Vargas Alvarez, Dr. Nikolaos Evangelou, and Dr. Dimitrios Patsatzis – your friendship and collaboration throughout this journey, and our teamwork on several papers, have been invaluable. I am also grateful to the Ph.D. students of the MERC, SPACE, and MPHS programs who shared this journey with me at SSM. A big thanks goes out to the Johns Hopkins University, especially the Department of Chemical and Biomolecular Engineering, where I have been for one year, with an enriching academic environment and resources. Speaking of amazing environments, I am also grateful to the Scuola Superiore Meridionale for the four-year scholarship that supported my research work. The beautiful buildings and garden in Largo San Marcellino have made this experience even more enjoyable.

To my parents and siblings, thank you for your unwavering support and belief in me throughout this journey. Your love and encouragement have been my foundation.

Finally, my deepest gratitude goes to my wife, Giovanna. Your love, patience, and understanding have been my greatest source of strength. This accomplishment is as much yours as it is mine.

# Contents

# Abbreviations

| | |
|---|---|
| **ABM** | Agent Based Model |
| **ADAM** | Adaptive Moment Estimation |
| **ARD** | Automatic Relevance Determination |
| **ANN** | Artificial Neural Network |
| **BDF** | Backward Differentiation Formula |
| **BFGS** | Broyden–Fletcher–Goldfarb–Shanno |
| **BVP** | Boundary Value Problem |
| **cdf** | cumulative distribution function |
| **COD** | Complete Orthogonal Decomposition |
| **CPU** | Central Processing Unit |
| **DAE** | Differential Algebraic Equations |
| **DE** | Differential Equations |
| **DeepONets** | Deep Operator Networks |
| **DMaps** | Diffusion Maps |
| **EF** | Equation-free |
| **e.g.** | for example (exempli gratia) |
| **ELM** | Extreme Learning Machine |
| **et al.** | and others (et alia) |
| **FD** | Finite Difference |
| **FEM** | Finite Elements Method |
| **FFT** | Fast Fourier Transform |
| **FHN** | FitzHugh-Nagumo |
| **FNN** | Feedforward Neural Network |
| **FNO** | Fourier Neural Operator |
| **FP** | Fokker-Planck |
| **GP** | Gaussian Process |
| **GPR** | Gaussian Process Regression |
| **GPU** | Graphics Processing Unit |
| **i.e.** | that is (id est) |
| **i.i.d.** | independent and identically distributed |
| **IPDE** | Integro Partial Differential Equation |
| **IVP** | Initial Value Problem |

| | |
|---|---|
| **JL** | Johnson and Lindenstrauss |
| **LBM** | Lattice Boltzmann Method |
| **LMA** | Levenberg–Marquardt algorithm |
| **ML** | Machine Learning |
| **MSE** | Mean Squared Error |
| **NN** | Neural Network |
| **ODE** | Ordinary Differential Equation |
| **PCA** | Principal Component Analysis |
| **PDE** | Partial Differential Equation |
| **pdf** | probability density function |
| **PINN** | Physics-Informed Neural Network |
| **PIRPNN** | Physics-Informed RPNN |
| **RandONets** | Random Projection-based Operator Networks |
| **RBF** | Radial Basis Function |
| **RC** | Reservoir Computing |
| **RFF** | Random Fourier Feature |
| **RFFN** | RFF Network |
| **RHS** | right-hand-side |
| **RK** | Runge-Kutta |
| **RKS** | Random Kitchen Sinks |
| **ROM** | Reduced Order Model |
| **RPNN** | Random Projection Neural Network |
| **RVFLN** | Random Vector Functional Link Network |
| **SciML** | Scientific ML |
| **SDE** | Stochastic Differential Equation |
| **SGD** | Stochastic Gradient Descent |
| **SLFNN** | Single Layer FNN |
| **SPDE** | Stochastic PDE |
| **SINDy** | Sparse Identification of Nonlinear Dynamics |
| **SVD** | Singular Value Decomposition |
| **tSVD** | truncated SVD |
| **UQ** | Uncertainty Quantification |
| **vdP** | van der Pol |
| **w.r.t.** | with respect to |

# 1 Introduction

This Thesis aims at the development of new methods at the intersection of scientific machine learning (ML) and numerical analysis to address and tackle challenging forward and inverse problems arising in complex systems. The main focus is how one can efficiently relax the “curse of dimensionality" in complex systems numerical analysis and in the training of artificial neural networks.

I highlight that part of this research was conducted in collaboration with Johns Hopkins University, Baltimore, MD, USA. During three visits spanning a total of one year (March-July 2022, February-July 2023, January 2024 and November 2024), I contributed to research within the Chemical and Biomolecular Engineering department under the guidance of Professor Ioannis G. Kevrekidis. This experience and the resulting research efforts have significantly enriched the overall research project.

In the next sections of the introduction, we will present the motivation, the relevant theoretical background and context of this research, specifically examining complex systems and their importance in modern science. We will discuss how ML and big data have transformed the study and analysis of complex systems and their emergent behavior. Next, we will review the state-of-the-art approaches and key challenges in solving forward and inverse problems in complex systems. Furthermore, we will outline the key research questions and objectives that have guided this thesis. Finally, this chapter will conclude by summarizing the contributions of this work and its importance in the context of risk and complexity.

## 1.1 Motivation and context of the research

In this section, we introduce the main motivation and context of the research, focusing on the intersection between complexity, multiscale modeling, and machine learning (ML) to tackle the “curse of dimensionality”. We highlight the importance of understanding emergent phenomena in large-scale complex systems and the role of reduced-order modeling for dealing with the “curse of dimensionality" when performing numerical analysis tasks for these phenomena. This is an open and challenging problem for both the forward and inverse problems. In fact, two Nobel Prizes in Physics in 2021 and 2024 were awarded for keystone works in complexity and machine learning for better understanding complex phenomena, and/or discovering new physical laws and emerging patterns from

data. The focus and main aim of this Thesis is the development of novel “intelligently-designed" ML methods with the aid of tailor-made numerical analysis tools, to deal with some aspects of the “curse of dimensionality" when trying to learn (the inverse problem) from high-fidelity simulators, and solve differential equations (DEs), with the use of ML. The achievement of such a goal opens the path to faster and more accurate solution of both inverse and forward problems.

### 1.1.1 A brief journey into complexity

Complex systems are ubiquitous in nature and society, captivating scientists with their intricate and emergent behaviors. Ranging, from the neural networks of the brain [1, 2] to the dynamics of financial markets [3, 4, 5, 6], and from climate patterns [7, 8, 9, 10] to the spread of epidemics [11, 12, 13], these systems exhibit complex patterns that *emerge* from the interactions of numerous components/units. Understanding, modeling, analyzing and forecasting these emergent phenomena from large-scale spatio-temporal data represent some of the most significant challenges in contemporary science [14, 15, 16, 17, 18, 19, 20, 21, 22, 23].

While complex systems are composed of interacting units (e.g., particles, agents) [14, 16], their collective behavior often transcends the properties of their individual constituents, emerging at different spatio-temporal scales than those governing the individual interactions. Unlike simpler systems, a reductionist approach, decomposing the system into its subunits, often fails to capture the holistic dynamics. This fundamental principle, eloquently expressed by Aristotle’s saying “the whole is greater than the sum of its parts", has been a central theme in both philosophical and scientific exploration for centuries. Pioneering works, such as Erwin Schrödinger’s exploration of life as a complex physical system in “What is Life? The physical aspect of the living cell" [24], and Ilya Prigogine’s Nobel Prize-winning research on the role of irreversibility in self-organizing systems [25], have laid the groundwork for our contemporary understanding. Prigogine’s concept of dissipative structures, emerging from far-from-equilibrium dynamics, underscored the importance of nonlinear interactions and time’s evolution in generating complex patterns [25]. Ultimately, the profound importance of research in complex systems was underscored by the 2021 Nobel Prize in Physics, awarded to Syukuro Manabe, Klaus Hasselmann, and Giorgio Parisi. Parisi’s contribution focused on the theory of disordered systems, glassy states and fluctuations in complex multi-scale phenomena [15]. This recognition solidified complex systems as a pivotal field in physics, demanding innovative approaches to unravel the intricate interplay between components and emergent properties.

Complex systems typically manifest multiscale phenomena, leading to unexpected and often abrupt shifts in the dominant macroscopic or mesoscopic patterns [16, 18, 19, 20, 21, 22, 23]. Interactions at the individualistic scale can trigger cascading effects, amplifying perturbations across the scales and propelling the system towards a new, potentially irreversible state. These catastrophic transitions, commonly associated with *tipping points* [18, 21, 26, 8, 23, 9], are more likely to occur near bifurcation points in the system’s emergent dynamics. Determining the frequency and probability of these transitions, as well as identifying the precursors to tipping points, is crucial for managing

and mitigating risks in various real-world applications. Such unpredictable, qualitatively different critical changes often arise from the multistable emergent behavior inherent to complex systems, where multiple emergent stable and unstable states or attractors coexist. These attractors can represent desirable or undesirable conditions, as exemplified by Earth's historical climate states, including glacial and interglacial periods, as well as the potential for a "hothouse Earth" scenario [7, 10]. In real world applications, even seemingly permanent states, such as the crystalline structure of diamond[1], are ultimately transient, subject to the relentless march of time and the influence of environmental factors. The metastability of complex systems, characterized by long periods of stability punctuated by abrupt transitions, often triggered by *rare events*, highlights the *nearly unpredictable* nature of these systems and the challenges associated with forecasting their future behavior [17, 22, 23]. Fluctuations in system parameters and the presence of noise can erode potential barriers separating different states, facilitating transitions to alternative regimes [23].

### 1.1.2 The Low-Dimensional Manifold Hypothesis in Multiscale Modeling of Emergent Phenomena

Given the inherent challenges posed by complex systems, including multistability, metastability, and the occurrence of rare events, as well as their evident high-dimensionality, nonlinearity, and multiscale nature, it might seem counterintuitive that their essential characteristics could be captured by a limited set of macroscopic observables [19, 27, 28, 29, 30, 31, 32], for example, density or momentum, as in the Navier-Stokes equations. However, this principle, often termed the *Manifold Hypothesis*, is a central hypothesis that underpins much of complex systems research [33, 29]. It asserts that the system's dynamics are effectively confined to a lower-dimensional subspace, as a consequence of the intricate interactions between system components. While the potential for irreducible complexity exists, the assumption of emergent behavior, often associated with the formation of collective patterns, is fundamental to the study of complex systems. This principle stems from the observation that the interactions between system components often induce correlations, effectively projecting/driving the system's high-dimensional dynamics onto a lower-dimensional manifold. Besides, *Self-organization* within complex systems leads to the formation of dissipative structures, from which macroscopic properties emerge [14, 16, 25]. The concept of a *Slow Manifold*, where system trajectories rapidly converge, provides a theoretical framework for this phenomenon [34, 29, 31]. This convergence is similar to the behavior observed in *singular perturbed systems*, where the system's dynamics separate into *fast* and *slow* components. Crucially, the separation of time scales inherent to complex systems leads to dependencies between higher-order and lower-order statistical moments. This characteristic underscores the potential for *dimensionality reduction* techniques. By focusing on a reduced set of carefully selected *coarse-grained* variables, one may effectively capture the system's essential behavior, thus developing *reduced order models* (ROMs) capable of simulating the collective

[1] Over an extremely long period of time, at standard temperature and pressure, diamond is thermodynamically unstable and will gradually transform into graphite.

dynamics of complex systems efficiently [35, 27, 36, 34, 19, 37, 11, 12, 28, 38, 39, 40].

To further clarify this concept, consider the behavior of a gas. While the individual motions of atoms within a gas are highly complex and occur in a high-dimensional space, the emergent behavior can be described using a single macroscopic variable: temperature. Temperature, in this case, represents the average kinetic energy of all the molecules in the gas, effectively reducing the dimensionality of the system's description [41]. Therefore, we can say that the emergent behavior in this case become apparent and physically explainable. Similarly to gases, fluid dynamics offers a canonical example of multiscale (and coarse-graining) modelling, where systems can be described at various levels of detail (granularity) [41]. The fundamental constituents are atoms/molecules, and the behavior of the fluid can be described through classical mechanics using Newton's equations of motion. This particle-based perspective forms the foundation for understanding fluid dynamics. At the macroscopic level, fluid flow is governed by the Navier-Stokes equations, a set of partial differential equations (PDEs) that describe the motion of fluid substances. These equations are fundamental to fluid mechanics and are used to model a wide range of phenomena, from weather patterns to blood flow [42, 41]. They capture the conservation of mass, momentum, and energy within the fluid, considering factors such as pressure, velocity, temperature, and fluid properties like viscosity [41]. Bridging the gap between this atomic view and the macroscopic behavior we observe requires intermediate levels of description. The *mesoscopic* scale, often modeled using the Boltzmann transport equation, represents one such intermediate level, where collective particle properties are considered [43, 35, 27, 44, 40, 45]. This approach, which tracks the evolution of particle density distributions, effectively represents particles as collective entities. Beyond the mesoscopic and macroscopic scales, even coarser representations are possible. Simplified models, such as the Lorenz system for turbulent flows [42], or the Doi model for liquid crystals [46, 44] can capture essential dynamics. In fact, the Lorenz system is a classic example of how one can construct ROM that can capture complex behaviors such as multistability and chaos, where small changes in initial conditions lead to vastly different outcomes. Despite the inherent unpredictability in chaotic regimes, it is remarkable that such complex dynamics can still be captured by low-dimensional models. In the case of the Lorenz attractor, three ordinary differential equations (ODEs) suffice to describe the system's chaotic behavior. This example highlights how even highly sensitive emergent behavior, originating from complex interactions at the high-dimensional microscopic scale, may often be reduced to simpler, physically meaningful low-dimensional macroscopic descriptions.

At the microscopic level, instead of particles in complex fluids, diverse individual agents or constituents and intricate interaction mechanisms contribute to complexity to different phenomena across fields. For instance, in neuroscience, neurons interactions enable cognitive functions [1, 2]. In ecology, predator-prey dynamics influence ecosystem balance [26]. In epidemiology, infected individuals may spread disease across a population [11, 12, 13], while in finance, traders and investors affect market fluctuations [3, 6, 28]. The physical and dynamical state of the entire complex system is high-dimensional, encompassing variables such as the position and velocity of agents in a crowd or the distribution of opinions in a complex social network. Individual agents

interact through a variety of mechanisms, which may include complex networks, spatial interactions, or alternative topologies such as hypergraphs. These interactions can be multifaceted; for example, agents may reside within multi-layer networks where they perform different functions and adhere to distinct interaction schemes, influencing their behavior in various systems. The nature of interactions is not solely determined by the topology; they can also be governed by stochastic processes, decision-making rules, environmental influences, or event-driven mechanisms, where the occurrence of specific events can strengthen, alter, or rewire the network of interactions.

A diverse array of computational modeling techniques, including Brownian dynamics [27], Monte Carlo simulations [38, 31], Agent-Based Models (ABM) [3, 11, 28, 13, 26, 6], molecular dynamics [36], cellular automata [47] and social-force models for crowd dynamics [48, 49], are employed to simulate these microscopic processes. The specific choice of technique depends on the nature of the system being studied, as each method is tailored to address different scales and complexities. These simulators often serve as digital twins, "creating surrogate versions of real-world complex systems inside our computing machines, changing the way we do science" [50]. Moreover, our need for understanding (and controlling) such phenomena has made microscopic simulators, such as ABMs, key modeling tools in digital twin modeling in domains ranging from ecology [26, 47] and epidemics [11, 13], to finance and economy [3, 4, 5, 28, 6]. However, while powerful, microscopic simulations can be computationally expensive due to the high dimensionality of the system. For example, simulating just one physical second of molecular motion can require days of computational time, making long-term predictions infeasible. This slowness often necessitates parallelized efforts and specialized hardware, such as GPUs, to manage the extensive computations involved. This challenge becomes even more pronounced when conducting systems-level analysis, such as localizing tipping points or performing rare event analysis. These tasks frequently rely on extensive, brute-force temporal simulations to estimate the frequency distribution of abrupt transitions [4, 5, 51, 52, 6]. However, such an approach confronts the "curse of dimensionality": the computational cost rises exponentially with the number of degrees of freedom [5, 6]. Such a direct simulation scenarios approach is therefore neither systematic nor computationally efficient for high-dimensional simulators. Furthermore, it often does not provide physical insight regarding the mechanisms that drive the transitions.

Ascending to a coarser grain, the mesoscopic level encompasses groups of particles or agents, often referred to as mesoscale structures. These structures can be characterized by probability density functions of statistical moments, providing a statistical description of the population. Mathematical frameworks, relying on kinetic theory and statistical mechanics, like Fokker-Planck (FP) equations [53, 44, 3], Langevin equations [54], lattice Boltzmann [35, 40, 45], dissipative particle dynamics [55] and wavelet analysis [56] can be employed to model these mesoscopic dynamics.

At the macroscopic scale, the system's emergent behavior becomes apparent. By adopting a continuum perspective, the system can be described by a limited set of coarse-grained observables governed by PDEs, possibly including stochastic terms (SPDEs) or integral terms (IPDEs). Alternatively, extremely-coarse mean-field approximations can offer some scenario-oriented or essential emergent description at a spatiotemporal-scale

of interest, thus ROMs composed of few ODEs, could represent effectively medium to long time evolution of the emergent behavior.

The potential of PDEs to capture the complexity of these systems is particularly intriguing. Nonlinear reaction-diffusion equations, such as Turing's model for morphogenesis [57], the Allen-Cahn equations for microstructure evolution and diffusive interfaces [58], and Prigogine's Brusselator diffusion PDEs [59], have demonstrated the ability to generate complex patterns and instabilities [57, 59]. Moreover, PDE-based models have been successfully applied to biological and socio-economic systems [60, 3]. The Keller-Segel model describes chemotaxis in bacterial populations [60], while FP equations have been used to model agent behavior in financial markets [3]. These applications underscore the versatility of PDEs in capturing emergent phenomena of the complex system dynamics.

However, for many *real-world* complex systems such equations in a closed form, i.e., good macroscopic descriptions, are in general unavailable [34, 19]. Besides, even the "correct" macroscopic variables for the description of the emergent behavior may be also not known a priori [37]. In certain situations, there could be physical observables with significant meaning. However, it becomes necessary to assess their significance within the context of a dynamical system and determine if they provide sufficient information to create closures [40]. Therefore, the reconstruction of "accurate" coarse-grained dynamical models from data is becoming a fundamental problem in complex systems theory and its various applications.

### 1.1.3 Machine Learning and Big Data analysis meet Complex systems

We live in an era where the volume, speed, and variety of *big data* are rapidly increasing, and its analysis is becoming increasingly important [61, 62, 63]. This is not only due to the fascination of extracting valuable, reliable, and understandable information from vast datasets, but also because these data are rapidly transforming our society and our interactions with technology [61]. Big data power services that were unimaginable just a few years ago, profoundly changing our daily lives and habits [61, 63]. In this context, *data mining*, a collection of mathematical techniques and methodologies aimed at efficiently extracting useful information and meaningful patterns from large datasets, has become essential [62]. This allows us to create models that can interpret data behavior and predict future trends. The tools underlying this process are diverse and multidisciplinary, including functional analysis, numerical analysis, algebra, statistics, and artificial intelligence.

Recent developments at the junction between numerical analysis and ML have revolutionized the way we think about modeling and analysis from big data, thus opening the way to a new direction, where the data themselves constitute the focus of interest and the generation of hypotheses arises from data mining and analysis [64, 65, 66, 67].

ML, a branch of artificial intelligence, focuses on methods that can "learn" from experience, thus adaptively improve performance based on data. ML is therefore highly suited for tasks such as classification, prediction, detection, optimization, and recognition, thus leading to new strategies for analysis, identification, and control of complex systems. In the context of complex systems, ML techniques have been applied to uncover

system structures (e.g., network topologies) [68] and analyze the dynamics of nonlinear behaviors, including the prediction of system evolution [40, 66], the reconstruction of bifurcation diagrams [69, 45], the design of controllers and observers [70, 71], the identification of Hamiltonian system [72] and the computation of Lyapunov exponents [73, 74]. On the other side, certain ML methods, like reservoir computing [75, 74] and long short-term memory networks (LSTMs) [76, 74], are themselves dynamical systems. This has led to new theoretical studies of these methods using tools from dynamical systems' theory. ML has proven effective in modeling complex physical phenomena [66], including climate networks [77], phase-field modelling [78], spatiotemporal chaos [73, 74], and neural circuits [79]. However, many real-world applications of complex systems – ranging from neuroscience and engineering to social sciences and economics – pose additional challenges, such as incomplete or noisy data and the difficulty of deriving macroscopic models from microscopic dynamics [80]. These challenges have sparked a growing interest in data-driven approaches to reconstructing effective models, especially when traditional equations are unavailable or too costly to solve [66].

Moreover, while ML offers powerful tools for handling such problems, it is often limited by data scarcity, especially in cases where collecting large, high-quality datasets is impractical [80]. This is particularly critical in complex systems research, where observational data can be sparse, incomplete, or highly uncertain. Addressing these limitations requires the development of methods that are both data-efficient and capable of incorporating domain knowledge [66]. Equally important is ensuring that the resulting models are robust and interpretable, with proper uncertainty quantification to assess the reliability of predictions [81]. The intersection of ML and complex systems continues to expand, offering new opportunities for both understanding and controlling complex systems. Promising future directions include hybrid data-driven and physics-informed models, explainable AI for decision-making, and real-time applications such as ML-assisted early-warning systems and online control.

### 1.1.4 Solving Forward and Inverse Problems in Complex Systems Using Machine Learning

The two fundamental applications of ML in complex systems regards the solution of the inverse and forward problem. In the forward problem, the mathematical model – typically described by ODEs or PDEs – is known, including the initial and boundary conditions. The unknowns are the solution fields, which correspond to the spatio-temporal behavior of the dynamical system. On the other hand, the inverse problem deals with the opposite situation: the data, typically collected by a finite number of sensors distributed across the spatial domain and recording at discrete time intervals, are known, but the underlying model or governing laws are unknown. These data often represent spatio-temporal realizations of the system's trajectories, which may be available either in their entirety (e.g., high-dimensional states of all agents) or through composite observations of these states [82].

Thus, the primary objectives of ML in this context are mainly two. The first is solving the inverse problem, which involves identifying or discovering the underlying

macroscopic laws. This includes learning nonlinear operators and constructing coarse-scale dynamical models, such as ODEs and PDEs along with their closures, using data from large-scale microscopic simulations or multi-fidelity observations [83, 65, 72, 84, 40, 74, 85, 86, 87, 66]. Second, once these coarse-scale models are constructed, ML can be used to systematically investigate the system's dynamics by efficiently solving the corresponding DEs, especially for challenging problems like stiffness and high-dimensional PDEs [88, 89, 83, 65, 69, 85, 90, 91, 92, 93, 66, 94, 95]. In addition to these primary objectives, a new paradigm is emerging that focuses on the identification of general nonlinear operators from data and/or adequately sampled functional spaces [86, 96, 97].

In the next section, we briefly review the state-of-the-art for the solution of forward and inverse problems.

## 1.2 State-of-the-art of ML for complex systems

The interest in using ML as an alternative to classical numerical analysis methods [98, 99, 100, 101, 102, 103, 104, 105, 106] for the solution of the inverse [107, 108, 109, 110, 111, 112, 27], and forward [113, 114, 115, 116] problems in DEs can be traced back three decades ago. These early efforts demonstrated the potential of using neural networks for forward tasks such as solving ODEs and PDEs [113, 114, 115, 116], feedback control [117], and reproducing numerical schemes [108, 112], as well as for inverse tasks such as system identification based on macroscopic observations and derivation of normal forms [107, 108, 109, 118, 112, 119]. Today, this interest has been boosted together with our need to better understand and analyze the emergent dynamics of complex multi-physics/ multiscale dynamical systems of fundamental theoretical and technological importance [66]. Moreover, advancements in computational power and new theoretical methods have driven the development of ML techniques for solving complex forward problems in PDEs, including nonlinearity, stiffness, sharp gradients, complex geometry and high-dimensionality [89, 88, 120, 121, 90, 122, 69, 95, 123].

More details on specific ML methods for forward and inverse problems are presented in the two following subsections.

### 1.2.1 State-of-the-art for the solution of the forward problem with ML

In recent years, significant advancements in ML have broadened our computational toolkit with the ability to solve the forward problem in DEs and multiscale/complex systems. For the forward problem, ML algorithms such as GPR and PINNs are trained to approximate the solutions of nonlinear DEs, with a particular interest in stiff and high-dimensional systems [88, 89, 83, 65, 93, 69, 95, 94, 91], as well as for the solution of nonlinear functional equations [66, 86, 124, 70, 125, 71].

Since the 1990s, there has been a notable increase in research focusing on solving numerical analysis problems [113, 108, 114, 117, 115, 112]. Early work by Lee and Kang (1990) [113] employed a Hopfield Neural Network to address the numerical solution of DEs. Rico-Martinez et al. (1992) [108] introduced a recursive multilayer ANNs

architecture, emulating the implementation of the 4th-order Runge-Kutta scheme, for the identification of continuous-time ODEs and the construction of their bifurcation diagram from experimental data. Meade and Fernandez (1994) [114] handle the resolution of linear ODEs with ANNs, applying the Galerkin weighted-residuals method. Yeşildirek and Lewis (1995) [117] employed a neural network-based controller for the feedback linearization of dynamical systems. Gerstberger and Rentrop (1997) [115] utilized feedforward neural networks (FNNs) addressing stiff ODEs, as well as DAEs. Lagaris et al. (1998) [116] systematically investigate the use of FNNs for the solution of linear and nonlinear DEs, addressing a range of scenarios from initial and boundary value problems. The method is based on the construction of appropriate trial functions, the analytical derivation of the gradient of the error with respect to the network parameters and collocation. The training of the FNN was achieved iteratively with the quasi-Newton BFGS method.

Nowadays, the exponentially increasing – over the last decades – computational power and recent theoretical advances, have fueled further developments at the intersection between ML and numerical analysis. In particular, on the side of the numerical solution of PDEs, the development of systematic and robust ML methodologies targeting at the solution of large scale systems of nonlinear problems with steep gradients constitutes an open and challenging problem in the area. Notably, PIML methods, such as those explored by Raissi et al. (2018, 2019), Han et al. (2018), and Karniadakis et al. (2021), have become a central focus [83, 65, 89, 66, 88, 126, 121, 122, 90]. In particular, the term “Physics-Informed Neural Networks" (PINNs) was coined [65] to describe ANNs that are trained to solve the forward and inverse problem for DEs, incorporating information about the governing equations, initial and boundary conditions (for PDEs), thus providing analytically the necessary derivatives that are needed for the training phase, using for example automatic differentiation [127, 93].

Raissi et al. (2018) [83], addressed the concept of numerical Gaussian Processes for the numerical solution of linear and nonlinear time-dependent differential operators. The proposed approach is demonstrated through the one-dimensional nonlinear Burgers, the Schrödinger, and the Allen–Cahn equations. Han et al. (2018) [89] used Deep Learning to solve high-dimensional nonlinear parabolic PDEs including the Black–Scholes, the Hamilton–Jacobi–Bellman and the Allen–Cahn equation. Wei et al. (2018) [88] used DNN to solve modified high-dimensional diffusion equations. The training of the DNN is achieved iteratively using an unsupervised universal ML solver. Regazzoni et al. (2019) [126] used FNNs to develop and solve ROMs from data. The approach was demonstrated using two low-dimensional non-linear systems of ODEs modelling a nonlinear pendulum and a nonlinear transmission lie circuit and two linear PDEs, namely the heat and the wave equation. Samaniego et al. (2020) [121] used Deep learning to solve coupled PDEs arising in phase-field models in mechanical problems. The approach is based on collocation while the training is achieved minimizing the energy of the system by exploiting the variational structure that may arise in some of these problems. Tang et al. (2021) [122] used ANN for the numerical solution and coupling of PDEs at their interfaces. The approach was demonstrated via the computation of coupled Poisson and advection-diffusion equations. Chen et al. (2021) [90] used Gaussian Processes

for the numerical solution of PDEs as well as the solution of the inverse problem. The efficiency of the scheme was validated through a nonlinear elliptic PDE, the 1D time-dependent Burgers and the Eikonal PDE. The optimization of the unknown parameters was performed using the Gauss-Newton method, which was found to converge in a small number of iterations for the specific problems.

While ML offers innovative approaches, it's important to acknowledge the significant contributions of classical numerical analysis methods. For spatial discretization, techniques such as FD, FEM, Finite Volume, and Spectral methods based on Fourier series or Chebyshev polynomials have been widely used for decades [106, 102, 105]. These methods provide robust and efficient solutions for a variety of PDEs, thanks to their solid theoretical underpinnings, in terms of convergence and stability, and proven performance in practical applications. For the time integration of initial value problems (IVPs), numerical analysis has developed a rich arsenal of techniques. The fundamental idea is to approximate the right-hand side (RHS) of ODEs with polynomials and intermediate approximation stages, and then integrate the resulting interpolating polynomial. This approach has led to methods like single-step Runge-Kutta methods, and Multi-step methods such as Adams-Bashforth and Adams-Moulton [98, 99, 100, 103, 104]. Specialized codes for ODEs often incorporate stable methods and adaptive time-stepping to enhance accuracy and efficiency. In `MATLAB`, composite methods with variable order and step size, like `ode45` and `ode15s`, are popular choices for both nonstiff and stiff problems [103].

Numerical analysis has made remarkable progress in solving DEs, as evidenced by W. Gear's influential SIAM review (1981), "Numerical solution of Ordinary Differential Equations: is there anything left to do?" [100]. For relatively simple problems, it is clear that nearly all challenges have been addressed, and specialized numerical solvers are highly efficient, to the extent that probably ML approaches may never surpass them in performance. However, when faced with complex and stiff problems, sharp gradients can pose significant challenges for traditional methods such as FD and spectral schemes, which may struggle to accurately capture the rapid changes in the solution. In addition, FD schemes are less developed, and FEM tessellations can become computationally infeasible for intricate geometries or high-dimensional PDEs in dimensions beyond three. In these scenarios, traditional numerical methods may encounter limitations due to the complexity of the geometry and the computational cost associated with mesh generation [89, 88].

Recent results in the literature, as reviewed above, suggest that PINNs hold significant promise for tackling such challenging scenarios, particularly in the context of stiff systems, sharp gradients, and high-dimensional PDEs. Nevertheless, it is important to recognize that PINNs are still in the early stages of development. Key challenges remain, including mitigating the high computational cost associated with the "curse of dimensionality" and overcoming the difficulties posed by the non-convex optimization inherent in training PINNs. Furthermore, establishing rigorous theoretical foundations for the convergence rates and stability of PINN methods is essential for their widespread adoption in practical applications.

### 1.2.2 State-of-the-art for the solution of the inverse problem with ML

The discovery of physical laws and the solution of the inverse problem in complex systems modelling, i.e., the construction of PDEs for the emergent dynamics from data and consequently the systematic analysis of their dynamics with established numerical analysis techniques is a holy grail in the study of complex systems and has been the focus of intense research efforts over the last years [66, 128, 97, 65].

The goal of the inverse problem is to identify the model, whether discrete or continuous, that governs the observed dynamics. This task is challenging due to the inherent ill-posedness of the problem, where multiple models may fit the data equally well. These models can vary in form, scale, and accuracy, ranging from black-box models (which rely on fully nonlinear representations) to gray-box models that incorporate prior knowledge about the system's physics. In other cases, the focus may be on parameter estimation, which could involve fitting functional parameters dependent on the states and their derivatives in inhomogeneous settings.

A systematic analysis of the framework to solve the so-called inverse problem requires three essential tasks. First comes the discovery of an appropriate low-dimensional set of collective variables (observables) that can be used to describe the evolution of the emergent dynamics [32, 30]. Such coarse-scale variables may, or may not, be *a priori* available, depending on how much physical insight we have about the problem. Indeed, for complex systems, such "good" macroscopic observables that can be used effectively for modeling the dynamics of the emergent patterns are not always directly available. Thus, such an appropriate set of "hidden" macroscopic variables have to be identified from data. Such data can be available either directly from experiments or from detailed simulations using for example molecular dynamics, ABMs, lattice Boltzmann methods and Monte-Carlo methods. For this task, various manifold/ML methods have been proposed, including diffusion maps (DMAPs) [129, 130, 40, 45, 131], ISOMAP [132, 133] and local linear embedding (LLE) [134, 135], but also autoencoders (AE) [136, 137, 72, 138, 139].

The second task focuses on identifying the appropriate type of model required to address the problem. A key consideration is whether the identified quantities can be modeled using ODEs or PDEs, or if global or integral features necessitate the use of IPDEs. This process involves determining whether a parabolic or hyperbolic time evolution is needed, or if mixed terms are appropriate. Additionally, it is crucial to identify which spatial partial derivatives are relevant—whether they represent diffusion, transport, reaction, advection, or higher-order processes. Feature selection poses a significant challenge in this context, as the relevant local or global (PDEs or IPDEs) spatial features are often unknown beforehand [40, 45, 67].

Based on this initial analysis, the third task pertains to the construction of appropriate ROMs, in order to parsimoniously perform useful numerical tasks and –hopefully– obtain additional physical insight. One (traditional) option is the construction of ROMs "by paper and pencil", using the tools of statistical mechanics [30, 32]. However, restrictive assumptions, that are made in order to obtain explicit closures bias the estimation of the actual location of tipping points, as well as the statistics and the uncertainty quantification (UQ) of the associated catastrophic shifts [32]. Moreover, the derivation of analytical

ROMs requires *a priori* knowledge of the "correct" physical variables, of the (first-principles) laws of the interactions between units and eventually the topology/network of interactions in high details.

Another option is the direct, data-driven identification of surrogate models in the form of ordinary, stochastic or partial DEs via ML. From the early '90s, exploiting both theoretical and technological advances, researchers employed ML algorithms for system identification using macroscopic observations, i.e., assuming that we already know the set of coarse variables to model the underlying dynamics and the derivation of normal forms ([107, 108, 109, 118, 112, 119]). The solution of the inverse problem leverages the ability of ML algorithms to learn the physical laws, their parameters, and closures among scales from data [140, 65, 40, 66, 45, 67, 141, 86, 124]. To the best of our knowledge, the first neural network-based solution of the inverse problem for identifying the evolution law (the right-hand-side) of parabolic PDEs, using spatial partial derivatives as basis functions, was presented in Gonzalez et al. (1998) [112]. In the same decade, such inverse identification problems for PDEs, were investigated through ROMs for PDEs, using data-driven Proper Orthogonal Decomposition (POD) basis functions [109] and Fourier basis functions (in a context of approximate inertial manifolds) in [119].

More recently, a growing number of techniques have been proposed to identify the governing laws of complex systems. Such approaches include, to name a few, sparse identification of nonlinear dynamical systems (SINDy) [142], GPR [40, 90], FNNs [108, 112, 40, 84, 45, 141, 139], RPNNs [45, 67], recursive neural networks (RvNN) [112, 138], reservoir computing (RC) [74], neural ODEs [143, 87], autoencoders [72, 138, 139], as well as DeepONet [86]. However, their approximation accuracy clearly depends very strongly on the available training data, especially around the tipping points, where the dynamics can even blow up in finite time.

In the early 2000s, the Equation-Free and Variable-Free multiscale framework [35, 19, 27, 37] provided a systematic framework for the numerical analysis (numerical bifurcation analysis, design of controllers, optimization, rare-events analysis) of the emergent dynamics as well as for the acceleration of microscopic simulations, by bridging the microscale where the physical laws may be known and the macroscopic scale where the emergent dynamics evolve. This bridging is achieved via the concept of the "coarse time steppers", i.e., the construction of a black-box map on the macroscopic scale. By doing so, one can perform multiscale numerical analysis, even for microscopically large-scale systems tasks by exploiting the algorithms (toolkit) of matrix-free methods in the Krylov subspace [144, 145, 28], thus bypassing the need to construct explicitly models in the form of PDEs. In the case when the macroscopic variables are not known a priori, one can resort to non-linear manifold learning algorithms such as Diffusion maps (DMaps) [129, 130, 146] to identify the intrinsic dimension of the slow manifold where the emergent dynamics evolve. If the coarse-variables are known, the Equation-free (EF) approach [19] offers an efficient alternative for learning "on demand" *local* black-box coarse-grained maps for the emergent dynamics on an embedded low-dimensional subspace; this bypasses the need to construct (global, generalizable) surrogate models. This approach can be particularly useful when conducting numerical bifurcation analysis, or designing controllers for ABMs [131]. However, even with a knowledge of good

coarse-scale variables, constructing the necessary *lifting operator* (going from coarse scale descriptions to consistent fine scale ones) is far from trivial [131].

In the next section, we describe the main challenges associated to the solution of forward and inverse problems.

## 1.3 Challenges of traditional and ML approaches for complex Systems: the curse of dimensionality

Despite numerous efforts in numerical analysis, ML, as well as theoretical and analytical approaches, significant open challenges persist in solving the inverse and forward problems in DEs. These challenges include: (a) improving closures beyond mean-field approximations by solving inverse problems from data generated by microscopic simulations, and, (b) efficiently solving high-dimensional PDEs in the forward problem, which is nontrivial, particularly when dealing with sharp gradients, stiffness, and complex geometries or free/moving boundaries.

Traditional numerical techniques, such as Finite Difference (FD) or Finite Element Method (FEM), often struggle with high-dimensional parameterized PDEs due to the "curse of dimensionality", leading to an exponential increase in computational resources. These methods rely on fixed grids or meshes, which can hinder their applicability in complex geometries, free-boundaries or problems characterized by steep gradients, necessitating adaptive approaches. Polynomial-based methods, while effective in low dimensions, often do not generalize well in higher dimensions, where ML methods, particularly meshless approaches like neural networks, can potentially excel. Besides, traditional numerical methods often face difficulties in incorporating noisy data, strong nonlinearities and stiffness introducing multiple sources of uncertainty [81].

ML has emerged as a promising avenue and potential solution for some of the above challenges [88, 89, 66, 69, 86, 67, 45, 91], and have led to a growing interest for solving more efficiently large-scale difficult numerical analysis problems.

However, ML itself is not without its challenges: failures may arise at the training phase, especially in deep learning formulations. This occurs because training deep neural networks, for both forward and inverse problems, can be slow and computationally expensive, exacerbated by the non-convexity of optimization landscapes and difficulties in escaping local minima [147, 148, 149, 150, 151, 66]. Also, training deep neural networks requires substantial amounts of data, which may not always be available for real-world problems. Furthermore, despite recent theoretical advancements, rigorous theoretical numerical analysis of their performance in terms of accuracy, stability, and robustness is still lacking, as well as the uncertainty quantification of the ML models is still limited in the literature [81, 152]. The uncertainty and interpretability of ML models present significant challenges, particularly in scientific contexts where understanding underlying dynamics is as crucial as predictive accuracy. In complex systems, additional challenges arise when modeling and identifying tipping points and rare events, which necessitate robust surrogate models capable of managing stiffness and inherent uncertainties [67].

Finally, while it is often believed that ML methods, such as ANNs, are less suscep-

tible to the “curse of dimensionality” due to their theoretically better scalability [153], the curse still affects both the data and model complexity. Training in high-dimensional parameter spaces, such as those found in deep learning models with numerous weights, biases, and hyperparameters (like the number of neurons, hidden layers, learning rates, and batch sizes), becomes exceedingly difficult. The vast number of options and decisions required—such as choosing the right architecture, optimization algorithm, and hyperparameters—makes finding an optimal solution highly challenging.

The above challenges highlights that the solution of inverse problems involving the identification of “hidden physics" from data often remains prohibitively expensive, and the theoretical tools are still at their infancy [83, 65, 66]. Furthermore, solving real-world physical problems with missing, gappy or noisy boundary conditions through traditional approaches is currently infeasible.

Two additional practical challenges for the solution of the inverse problem, that are worth to mention, involve *emergent spaces* and *parameter non-identifiability* [154, 155]. The first challenge lies in the identification of emergent spaces for complex systems, such as networks of interacting agents, where no obvious spatial coordinates exist to define effective evolution laws in the form of PDEs [154]. In such systems, embedding coordinates must be learned from time-series data through manifold learning techniques, which then allow for the discovery of effective PDEs in these emergent spaces.

The second challenge arises in learning parametric PDEs. The specific parameter, whose changes or perturbations cause sudden and dramatic changes in the system’s behavior, may remain unknown [155]. While some system parameters may be known, the effective bifurcation parameter might remain hidden. This issue becomes even more apparent when data is structured not around explicit, tunable parameters, but rather as realizations from different populations. This scenario often arises in real-world applications where data is collected from diverse sources or under different conditions.

## 1.4 Key research questions

Central to our research is whether Machine Learning (ML) can effectively complement or even replace traditional numerical methods in solving both forward and inverse problems. Some key research questions that this Thesis addresses are here listed:

- To what extent can ML methods complement or replace traditional numerical techniques in solving both forward and inverse problems within the context of complex systems?
- Can ML be employed to uncover novel insights into the underlying dynamics of complex systems, particularly with respect to identifying tipping points and analyzing rare events?
- Can we overcome the difficult issues associated with the “curse of dimensionality" and the non-convexity when optimizing large-scale deep neural networks?

The research questions posed are framed within the context of complex systems due to the intrinsic challenges of capturing their emergent behaviors, high-dimensionality,

and nonlinearity. Additionally, the research questions posed arise from the recognition of significant limitations in both traditional numerical methods and modern ML approaches when applied to complex system modeling and simulation. Addressing these pitfalls, as outlined in Section 1.3, requires the development of hybrid methods that combine the best aspects of numerical analysis (such as stability and theoretical guarantees) with the flexibility and scalability of ML. This Thesis not only explores the fundamental questions outlined, but also contributes to a larger discussion on the integration of ML with traditional numerical techniques. In this context, the research questions aim to explore whether ML can mitigate the limitations of traditional methods, while also pushing the boundaries of ML to handle issues like stiffness, rare event prediction, and complex operator discovery from data. As ML continues to evolve, its capacity to tackle complex, high-dimensional problems will become increasingly relevant, and our findings lay the groundwork for future research in these domains.

## 1.5 Main Objective of the Thesis

The primary objectives of this Thesis, are connected to the solution of both the forward and the inverse problems, with the aid of ML and tailor-made numerical analysis to deal with the "curse of dimensionality". In particular, the focus is on the development of a multiscale ML numerical-analysis-assisted framework for the systematic extraction of coarse-scale observables from microscopic/fine-scale data, the construction and also the numerical solution of effective PDEs (or ROMs) using novel ML and numerical analysis algorithms, that can lead to significant computational savings in large-scale spatio-temporal simulations.

The work introduces novel fine-tuned random projection-based methods for both function approximation and operator learning, pioneering their application in inverse problem with reconstruction of bifurcation diagrams, and achieves state-of-the-art results in solving forward problems for ODEs and PDEs, supported by efficient computational implementations. A schematic overview of the main objectives of the thesis is depicted in Figure 1.1

To achieve this, the research will focus on the following key goals:

- *Reduction of Complexity*: Utilize manifold learning techniques to identify essential macroscopic observables that capture the key dynamics of complex systems.
- *Data-Driven Reduced Order Model Construction*: Develop ML frameworks, particularly based on FNNs and RPNNs, to learn the RHS of effective DEs based on the extracted observables, while handling uncertainty and nonlinearity.
- *Handling local and/or global Features*: Address challenges in capturing dominant spatial features by combining neural architectures with specialized feature selection techniques for more accurate model representation.
- *Assess Surrogate Models*: Investigate the trade-offs between low-dimensional surrogate models (e.g., ODEs/SDEs) and more complex infinite-dimensional PDEs, optimizing for both computational efficiency and model accuracy.

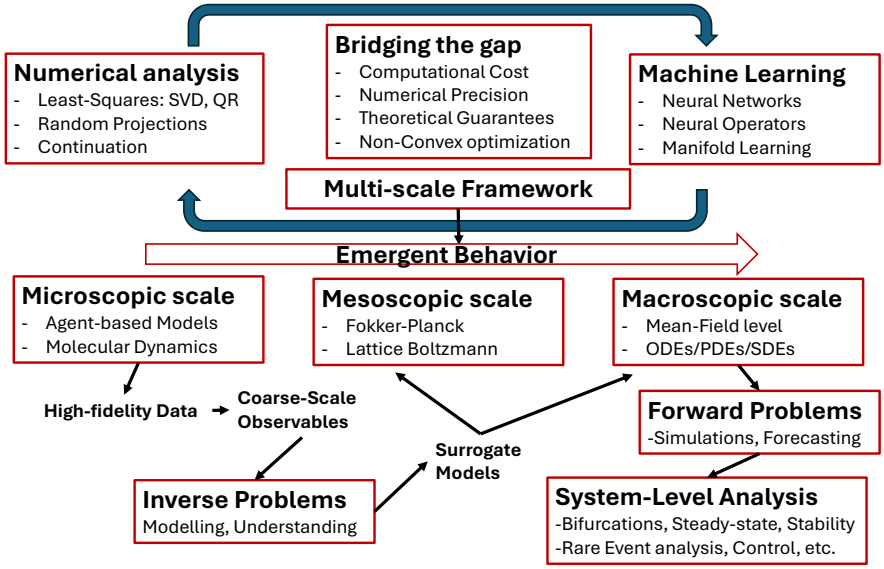



Figure 1.1: Schematic Overview diagram of the main objectives of the Thesis.

- *Analysis of the emergent dynamics*: studying the qualitative changes in system behavior by performing bifurcation analysis on the ML-assisted surrogates. The goal is to understand how macroscopic patterns and phenomena arise from underlying microscopic interactions.
- *Accelerate full-scale microscopic simulations*: by creating simplified models that retain key features of the dynamics, we can significantly reduce computation time, enabling faster simulations.
- *Rare Event Prediction*: Develop algorithms that accelerate rare event analysis, usually infeasible when performed at the level of full-scale microscopic simulations, improving the understanding of emergent behaviors and tipping points.
- *Overcoming Numerical Limitations*: particularly, the limitations of traditional numerical techniques, in high-dimensional spaces and stiff systems.
- *Hybrid Methods*: Design hybrid numerical-ML schemes that integrate the stability and theoretical guarantees of numerical analysis with the flexibility and scalability of ML to solve PDEs and ODEs efficiently.

Alongside the above, one additional goal is to advance the application of ML techniques, both theoretically and practically, tackling key challenges associated with their training, and making them suitable alternatives, if not the primary choices, when solving both forward and inverse problems in the context of complex systems.

These objectives guide the research questions and the methodological approaches developed throughout this thesis.

## 1.6 Contributions of this work

The research conducted during the 4-year PhD (2020-2024) has contributed to several key advancements in numerical analysis and PIML for complex systems. The work follows a three-tier approach: (a) developing new methodologies, (b) addressing inverse problems, and (c) solving and analyzing forward problems.

### 1.6.1 Methodologies and Theoretical Advances

In terms of methodologies and theoretical advances, notable contributions were made through the development and optimization of RPNNs. These networks have been tailored to effectively solve efficiently the forward problems for ODEs and PDEs in a physics-informed way [69, 95]. Additionally, they were successfully coupled and integrated with bifurcation analysis tools [69, 95]. We have also theoretically investigated the stability of such PINN-based methods, for the first time [152]. Besides, within the random feature framework, we have introduced the concept of RPNN of best approximation, establishing its existence, uniqueness, and fast convergence – both theoretically and experimentally – when approximating smooth functions [156]. Furthermore, we have introduced a novel architecture for the approximation of operators, termed RandONets, taking inspiration from DeepONets and the Chen and Chen universal approximation theorem of operators [157]. In such work, we have also extended the theoretical proof of universal approximation of operators to RandONets. In [157], this approach was tested on various linear and nonlinear PDE evolution operators, demonstrating RandONets' ability to effectively model and approximate these complex systems with minimal computational cost, outperforming their vanilla fully-trained DeepONets counterparts by several orders of magnitude.

In addition, we introduced a parsimonious approach for selecting the bounds of the uniform distribution of internal weights for boundary-value problems [69, 95]. This is a crucial step. Although, theoretically any random selection may suffice, in practice, it is advantageous to define appropriate ranges for biases and internal weights based on the chosen activation function. To achieve this, we have demonstrated that selecting and fixing internal weights and biases through random uniform sampling strategies can lead to more accurate approximations, surprisingly, than the conventional approach of training these parameters through optimization. This finding challenges the typical belief that optimizing weights via training is always necessary for improving model performance. By avoiding the computational cost and potential pitfalls of training (such as overfitting or getting stuck in local minima), we provide a simpler, more efficient method for constructing neural networks that are both robust and highly accurate. We show also how to extend the randomization of weights when dealing with high-dimensional inputs [45]. In [95], we conducted an extensive search to optimize the bounds of the uniform distribution from which the parameters are randomly sampled, rather than optimizing the parameters directly. This approach was applied to time-dependent problems, such as stiff ODEs and differential-algebraic equations (DAEs), demonstrating that carefully tuning the range of the random weights can lead to significant improvements in accuracy.

### 1.6.2 Solution of Forward Problems

Regarding the solution of the forward problem, we have established a framework for PINNs, taking advantages of random projections, that significantly improves both speed and accuracy compared to traditional PINNs trained with gradient-based methods, achieving performance improvements of up to 5 orders of magnitude, as demonstrated in [95]. From the theoretical point of view, we have established the convergence of the Physics-informed RPNN (PIRPNN) scheme when approximating ODEs and DAEs in the semi-implicit form [95]. Furthermore, we have coupled these ML solutions with numerical continuation techniques for bifurcation analysis, which is a key tool in understanding the nonlinear dynamics of complex systems, such as the identification of critical transitions and branching behaviors [69]. Additionally, we have emphasized the importance of hybrid methods that integrate concepts from numerical analysis to enhance both computational efficiency and accuracy [157, 95, 69]. This includes the use of continuation schemes to provide better initialization for solving ODEs in subsequent time intervals, and employing simple error control schemes to adapt time steps based on Newton's method iterations [95].

Most importantly, we have also demonstrated for the first time that RPNN-based ML methods can outperform established numerical solvers in both computational cost and numerical accuracy, such as FD and FEM for PDEs [69], and the adaptive backward differentiation formula of variable order 1 to 5, implemented in `MATLAB`'s `ode15s` function, for time dependent problems [95].

Such PIRPNN approaches have been tested against several benchmark problems, including the stationary PDE Liouville-Bratu-Gelfand PDE in 1d and 2d and Viscous Burgers' PDE with mixed boundary conditions [69]. As well, we have considered several benchmark problems for stiff ODEs and DAEs, including van der Pol ODEs, Robertson DAEs, Belushov-Zhabotinsky ODEs and the stiff Allen-Cahn phase field PDE discretized along space with FD [95].

### 1.6.3 Solution of Inverse Problems

Concerning the solution of inverse problems, we have demonstrated how parametric PDE models can be learned from data generated by lattice Boltzmann mesoscale models, successfully reconstructing the FitzHugh-Nagumo PDEs, which describe the *action potential propagation in unmyelinated neurons*, and its coarse-scale bifurcation diagram including correct localization of saddle-node and Andronov-Hopf bifurcations [45]. In addition, we investigated inverse problems starting from high-fidelity data generated by two microscopic ABMs: (a) a financial market ABM incorporating mimetic behavior among agents and (b) an ABM modeling the spread of an epidemic on an Erdős–Rényi social network [67]. For both models, we also learned one-dimensional surrogate SDE models near the tipping point [67] and conducted rare event analysis in its neighborhood. We have shown how the local targeted SDE accelerates the simulation, as well as the rare event analysis, of the full ABMs by several orders of magnitude. We have also utilized DMaps for the selection of coarse-scale variables, and applied ARD with GPR alongside parsimonious DMaps to identify the most relevant differential and integral features [67].

These features are explicitly recognized and utilized by a convolutional neural network with a fixed FD stencil, enhancing the model's ability to efficiently process and learn from spatio-temporal data. Toward the above, we established a general framework for inverse problems when neither physical insights nor macroscopic laws or observables are available.

In [67], we demonstrated that the choice of ML surrogate models should be dictated by the task at hand. We contrast global surrogates that operate across the entire parameter space, such as ML-learned IPDEs, with local, task-targeted surrogates like ML-learned SDEs. We demonstrate that global surrogates, though computationally intensive and data-demanding, are ideal for uncovering emergent behaviors, refining physical insights, and performing accurate numerical bifurcation analysis, especially for tipping point localization. They are particularly useful for correcting potential biases in macroscopic analytical models [67]. In contrast, low-dimensional ML-learned SDEs are more efficient, requiring less data and providing fast, scenario-specific insights. These models are well-suited for local control design and rare event analysis due to their ability to capture system volatility, whereas PDE-based approaches may require stochastic formulations (SPDEs), which are often more complex [67].

## 1.7 Relevance to risk and complexity

As has likely become evident, the core of our research addresses the inherent complexity of large-scale systems, particularly in determining and predicting their emergent behaviors, which are shaped by nonlinear dynamics and manifold structures [45, 67]. We focus on uncovering the intricate dependencies and behaviors within these systems, using ML to reveal hidden structures and relationships that standard analytical methods fail to capture. By tackling inverse problems in these contexts, we deepen our understanding of complexity itself, shedding light on how critical transitions and rare events emerge from the underlying microscopic chaos [67].

Furthermore, our research on solving inverse problems directly addresses the nexus of complexity and risk, central to the MERC PhD program, by focusing on tipping point analysis and rare event estimation at the emergent, macroscopic level of complex systems [67]. By developing innovative ML-based methodologies, we provide deeper insights into system vulnerabilities and the potential for catastrophic failures, such as those arising in financial markets [67], epidemic outbreaks [67], and climate change [10]. Our framework efficiently handles computationally demanding tasks, enabling the identification of critical thresholds – tipping and bifurcation points – and the quantification of associated risks, such as escape probabilities near these critical points.

While not explicitly focused on control theory, our work on recovering ROMs in the form of PDEs or low-dimensional SDEs lays the groundwork for advanced control strategies in complex systems. By generating models across different scales and by establishing mappings between fine-scale, microscopic observables (e.g., full agent configurations) and mesoscale or coarse-grained observables, either explicitly or through data-driven approaches, we can inform the design of control actions at the microscopic level. This could allow for the design of distributed control strategies in finite-dimensional environments

at a fine scale, guided by the emergent phenomena/consequences at larger scales.

This research is inherently interdisciplinary, drawing on ML, numerical analysis, bifurcation theory, and dynamical systems [69, 95, 67, 157]. By integrating ML and numerical methods, we push the boundaries of both fields to create more robust and accurate models for complex systems. The insights gained through this research are applicable to a broad range of fields, like neuroscience, finance, and epidemiology, offering valuable tools for risk mitigation and decision-making in uncertain, high-stakes environments.

## 1.8 Thesis structure and outline

The rest of this Thesis is structured into seven chapters, each addressing a different aspect of the methodologies, development, and applications of ML and RPNNs in solving complex systems problems.

Chapter 2 starts by introducing artificial neural networks (ANNs), covering fundamental concepts such as the universal approximation theorem and basic training algorithms, while also addressing key challenges related to fully-trained vanilla ANNs. Then provides an in-depth exploration of linear and nonlinear random projection techniques, beginning with the Johnson-Lindenstrauss (JL) Lemma and the work of Rahimi and Recht on Random Fourier Features (RFF) [158] and Random Kitchen Sinks (RKSN) for approximating kernel methods [159, 160]. The chapter transitions into the introduction of RPNNs of best approximation, a concept borrowed by polynomial of best approximation. Thus, we prove their existence, uniqueness and investigate their convergence properties. which achieve exponential convergence when approximating smooth functions. We demonstrate that there exists a choice of external weights, for any family of such RPNNs, with non-polynomial infinitely differentiable activation functions, that exhibit an exponential convergence rate when approximating any infinitely differentiable function. Additionally, this chapter presents various strategies for the random generation of weights in RPNNs. For illustration purposes, we test the proposed RPNN-based function approximation, with parsimoniously chosen basis functions, across three benchmark function approximation problems. Results show that RPNNs achieve comparable performance to established methods such as Legendre Polynomials, highlighting their potential for efficient and accurate function approximation.

In Chapter 3, inspired by DeepONets [86] and Chen and Chen (1995) Universal approximation theorem for operators [111], we introduce Random Projection-based Operator Networks (RandONets): shallow networks with random projections that learn linear and nonlinear operators.The implementation of RandONets involves: (a) incorporating random bases, thus enabling the use of shallow ANNs with a single hidden layer, where the only unknowns are the output weights of the network's weighted inner product; this reduces dramatically the dimensionality of the parameter space; and, based on this, (b) using established least-squares solvers (e.g., Tikhonov regularization and preconditioned QR decomposition) that offer superior numerical approximation properties compared to other optimization techniques used in deep-learning. In this chapter, we prove the universal approximation accuracy of RandONets for approximating nonlinear

operators and demonstrate their efficiency in approximating linear nonlinear evolution operators RHS, with a focus on PDEs. We show, that for this particular task, RandONets outperform, both in terms of numerical approximation accuracy and computational cost, the “vanilla" DeepONets.

Chapter 4 addresses a new numerical method based on RPNNs with both sigmoid and radial-basis functions, for the computation of steady-state solutions and the construction of (one-dimensional) bifurcation diagrams of nonlinear PDEs. For our illustrations, we considered two benchmark problems, namely (a) the one-dimensional viscous Burgers with both homogeneous (Dirichlet) and non-homogeneous boundary conditions, and, (b) the one- and two-dimensional Liouville–Bratu–Gelfand PDEs with homogeneous Dirichlet boundary conditions. For the one-dimensional Burgers and Bratu PDEs, exact analytical solutions are available and used for comparison purposes against the numerical derived solutions. Furthermore, the numerical efficiency (in terms of numerical accuracy, size of the grid and execution times) of the proposed numerical ML method is compared against central FD and Galerkin weighted-residuals FEM methods. We show that the proposed numerical ML method outperforms in terms of numerical accuracy both FD and FEM methods for medium to large sized grids, while provides equivalent results with the FEM for low to medium-sized grids. Furthermore, the computational times required with the proposed ML scheme were comparable and in particular slightly smaller than the ones required with FEM.

Chapter 5 continues the exploration of RPNNs for solving forward problems. We present a numerical method based on random projections with Gaussian kernels and physics-informed neural networks for the numerical solution of initial value problems (IVPs) of nonlinear stiff ODEs and index-1 DAEs, which may also arise from spatial discretization of PDEs. The unknown weights between the hidden and output layer are computed with Newton’s iterations using the Moore–Penrose pseudo-inverse for low to medium scale and sparse QR decomposition with $L^2$ regularization for medium- to large-scale systems. Building on previous works on random projections, we also prove its approximation accuracy. To deal with stiffness and sharp gradients, we propose an adaptive step-size scheme and address a continuation method for providing good initial guesses for Newton iterations. The “optimal" bounds of the uniform distribution from which the values of the shape parameters of the Gaussian kernels are sampled and the number of basis functions are “parsimoniously" chosen based on bias-variance trade-off decomposition. To assess the performance of the scheme in terms of both numerical approximation accuracy and computational cost, we used five benchmark problems (two index-1 DAEs problems, and three stiff ODEs problems including the Allen–Cahn phase-field PDE). The efficiency of the scheme was compared against two stiff ODEs/DAEs solvers, namely, `ode15s` and `ode23t` solvers of the `MATLAB` ODE suite as well as against deep learning as implemented in the `DeepXDE` library for the solution of the Lotka–Volterra ODEs included in the demos of the library.

Chapter 6 shifts the focus to the inverse problem. We present an ML framework bridging manifold learning, neural networks, Gaussian processes, and Equation-Free multiscale approach, for the construction of different types of effective ROMs from high-fidelity data produced by detailed microscopic simulators and the systematic multiscale

numerical analysis of their emergent dynamics. Key objectives include detecting tipping points and quantifying the uncertainty of rare events near these critical transitions. We address the discovery of underlying (global) IPDEs and (targeted) SDEs. We introduce the use of manifold learning to uncover reduced representations of high-dimensional data, specifically, DMaps is employed to reveal latent variables that capture the system's essential dynamics. Then, we focus on learning Mesoscopic IPDEs via ANNs presenting a Convolutional approach based on fixed FD stencil, to generate differential features candidates to learn the IPDE operators. Then we address ARD and the combination of DMaps with leave-one-out cross-validation to systematically identify the differential features that contribute the most to the representation of the dynamics. Then as an alternative to such endeavor, we propose an ML method, inspired by Euler-Maruyama scheme, to identify mean-field-level SDEs.

In Chapter 7, we demonstrate the application of this framework to three distinct case studies: (a) the one-dimensional FitzHugh-Nagumo (FHN) PDEs, which describe the action potential propagation in unmyelinated neurons, simulated at the mesoscopic scale using a D1Q3 Lattice Boltzmann method; (b) an event-driven, stochastic financial market ABM describing the mimetic behavior of traders; and (c) a compartmental stochastic epidemic ABM on an Erdös-Rényi social network. In each case, ML techniques are employed to recover ROMs, providing new insights into system dynamics and tipping points. Moreover, we contrast the pros and cons of the different types of surrogate models and the effort involved in learning them. Importantly, the proposed framework reveals that, around the tipping points, the emergent dynamics of both benchmark ABM examples can be effectively described by a one-dimensional SDE, thus revealing the intrinsic dimensionality of the normal form of the specific type of the tipping point. This allows a significant reduction in the computational cost of the tasks of interest.

Finally, Chapter 8 summarizes the key contributions of this work and discusses potential avenues for future research.

To facilitate a smooth presentation of the material presented in this Thesis, some basic definitions, preliminaries, and methodologies are included in Appendices A, B and C for a quick reference.

Specifically, in Appendix A, there is an overview of topology, metric spaces, elements of measure theory and Fourier transform.

In Appendix B, we briefly introduced the fundamental concepts of dynamical systems, manifolds, bifurcation theory and elements of multiscale modeling.

In Appendix C, to provide a robust framework for our analysis, key ML and data analysis techniques are briefly discussed.

# 2 Rethinking Neural Networks: a Random Projection perspective

This chapter explores an innovative approach to neural network architecture through the lens of random projections. Traditional neural networks face significant challenges during training, particularly due to the “curse of dimensionality" that arises from optimizing non-convex loss functions in high-dimensional parameter spaces. As architectures grow larger and more complex, they become overly parameterized, complicating the optimization process.

Random projections, alongside fixed basis functions, simplify this challenge by transforming the optimization problem into a convex one focused solely on the external weights. This allows for the resolution of linear least-squares problems, which may be highly ill-conditioned and underdetermined, using specialized numerical techniques such as Tikhonov regularization, Singular Value Decomposition (SVD), QR decomposition, and Complete Orthogonal Decomposition (COD) [161].

This perspective not only enhances computational efficiency but also opens new avenues for model interpretability and generalization.

In the next section, we will first offer a brief overview of ANNs and the challenges related to their training optimization. Following that, we will present the current state of the art and review the concept of random projection, highlighting its theoretical foundation rooted in low-distortion embeddings as outlined by the Johnson-Lindenstrauss (JL) Lemma. We will then present neural network approximations using random projections, focusing on RPNNs of best $L^p$ approximation. We will prove the existence and uniqueness of such optimal RPNNs and demonstrate their exponential convergence when approximating smooth functions. We conclude the chapter with some illustrative numerical example.

## 2.1 Artificial Neural Networks

In this section, we introduce the fundamental concepts of Artificial Neural Networks (ANNs). ANNs have gained significant popularity due to their versatility in addressing a wide range of ML tasks, such as classification, regression, clustering, and reinforcement learning. In particular, they have demonstrated outstanding performance in areas such as speech recognition, image synthesis, language modeling, and autonomous vehicles [162, 163, 164, 165, 166].

Composed of interconnected neurons, ANNs loosely simulate the structure and function of biological neural networks. Indeed, in the 1940s, W. McCulloch and W. Pitts [167] developed mathematical models that, in a highly simplified manner, replicated the functioning of a biological neuron, and it is to these models that the theory of neural networks can be traced back. In recent years, they have proven to be highly effective in improving the efficiency and accuracy of complex numerical and optimization problems across various fields, including finance, medical science, and engineering [168, 169, 66].

In mathematical and computational contexts, ANNs have become integral to function approximation for solving a range of numerical and mathematical problems, including DEs, system identification, and physics-informed solutions, as discussed earlier [113, 114, 116, 89, 65, 95, 66] They are also continually explored for advanced numerical analysis, artificial intelligence tasks, and emerging technologies. Considering the numerous numerical applications where ANNs have proven effective, it is important to explore the theoretical foundations that justify their broad use. Moving from practical applications to theoretical insights, it becomes clear that the success of ANNs across various fields is grounded in solid mathematical principles. Contrary to the view of neural networks as “magical" black boxes, a significant body of often underappreciated theoretical work supports their effectiveness. The universal approximation theorem [170, 171, 172, 173, 174, 175, 153] establishes that an ANN with a single (or multiple) hidden layer(s) can approximate any continuous function within a bounded domain to arbitrary accuracy, provided the number of hidden layer nodes is sufficiently large. This reflects the density property of ANN-based functions in $C(\mathbb{R}^d)$, with respect to the topology of uniform convergence on compact sets.

Recently, numerous results have expanded the approximation capabilities of ANNs to cover a broader range of function spaces. These advancements include approximation in Sobolev spaces [176], band-limited functions [177], Barron spaces [178], and Hölder spaces [179].

Despite significant empirical and theoretical progress, concerns persist that methods based on DNNs currently do not fully meet the traditional rigorous standards of stability, convergence, and efficiency expected in computational science and numerical analysis. While the theoretical foundations of DNNs highlight their ability to represent a broad class of functions, practical implementation remains challenging due to the non-convex nature of the optimization problem during network training.

The non-convex nature of the optimization landscape leads to challenges like saddle points, local minima, and flat regions, which can hinder the convergence of optimization algorithms. As a result, there remains a significant gap between the theoretical potential

of DNNs and their practical performance [147, 148, 149, 150, 151]

### 2.1.1 Architectures, interaction scheme and activation functions

An ANN consists of interconnected nodes, or neurons, which process input data to produce output through a series of functions. The process can be broken down into two main steps. First, there is an interaction scheme, combining multiple inputs from other neurons to produce a scalar output. The most common interaction scheme is the weighted sum, where each input is multiplied by a corresponding weight. However, the interaction scheme can vary based on the network type. For example, Radial Basis Function (RBF) networks rely on distance-based interactions [180, 95], Convolutional Neural Networks [112, 162] use convolutions, DeepONets employ weighted inner products [86, 157], Long Short-Term Memory (LSTM) networks utilize recurrent links [76, 74], and Transformers incorporate attention mechanisms [164, 166]. Then, after the interaction scheme, the output from the interaction scheme is fed into an activation function, which transforms it into the neuron's final output.

ANNs can adopt various architectures, such as deep feedforward networks (DFNN) [147, 149, 181], perceptrons [182], convolutional networks [112, 162], recurrent networks [183, 74], and autoencoders [72, 138, 139]. More advanced structures include Transformers [164, 166], Generative Adversarial Networks (GANs) [165], DeepONets [86, 157] and LSTM networks [76, 74], each designed to handle different types of tasks.

In the next section, we will focus specifically on FNNs.

### 2.1.2 Feedforward Neural Networks (FNNs)

FNNs are the simplest and most common architecture, particularly effective for supervised learning tasks such as regression, classification, forecasting, and model identification. These networks are characterized by their layered structure, where computing units (neurons) are arranged in multiple layers, each unidirectionally connected to the previous and next layers. In 1958, Frank Rosenblatt [184] expanded on the ideas initially proposed by W. McCulloch and W. Pitts with his invention of the *perceptron*, the first algorithm for training single-layer neural networks. Despite initial enthusiasm, the limitations of early neural networks, such as their inability to solve non-linear problems, were highlighted by M. Minsky and S. Papert in 1969 [185], significantly slowing progress in the field. It wasn't until the development of the *back-propagation* algorithm in 1986 [186] that neural networks, especially FNNs, experienced a resurgence.

In an FNN, neurons are usually distributed across $\mathcal{L}+2$ layers, indexed by $l = 0, 1, \ldots, \mathcal{L}, \mathcal{L}+1$. When $\mathcal{L} \geq 2$, the network is typically referred to as a *deep feedforward network* (DFFN). The first layer, $l = 0$, is known as the *input layer*, while the last layer, $l = \mathcal{L}+1$, is the *output layer*. The layers between the input and output, indexed by $l = 1, \ldots, \mathcal{L}$, are called *hidden layers*. Each $l$-th layer consists of $N_l$ neurons, which produce output values $\underline{x}^{(l)} = (x_1^{(l)}, \ldots, x_{N_l}^{(l)}) \in \mathbb{R}^{N_l}$ and are fully connected to the nodes of the previous layer ($(l-1)$-th layer) via weighted connections. The *weights* between the $(l-1)$-th and $l$-th layers are stored in a matrix $A^{(l)} \in \mathbb{R}^{N_l \times N_{l-1}}$. Additionally, a

*bias vector* $\underline{\beta}^{(l)} = (\beta_1^{(l)}, \ldots, \beta_{N_l}^{(l)}) \in \mathbb{R}^{N_l}$ is typically associated with each layer.

The role of hidden layers is often viewed as encoding the input data into $N_{\mathcal{L}}$ new features, thus acting as a filter that transforms the input. The final output layer linearly combines these $N_{\mathcal{L}}$ values to produce the network's final output. In this way, the network processes the input into new representations that, when combined with a fixed weight vector, determine the behavior of the underlying network function.

Let us now consider an DFNN with an $N_0$-dimensional input $\boldsymbol{y}^{(0)}$, and $L$ hidden layers, each containing $N_l$ neurons. The output $y_j^{(l)}$ of the $j$-th neuron ($j = 1, \ldots, N_l$) in the $l$-th layer ($l = 1, \ldots, L$) is obtained by applying the activation function $\psi : \mathbb{R} \to \mathbb{R}$ to a linear combination of the outputs from the neurons in the previous layer:

$$y_j^{(l)} = \psi\left(\sum_{i=1}^{N_{l-1}} w_{ji}^{(l)} y_i^{(l-1)} + b_j^{(l)}\right), \tag{2.1}$$

where $w_{ji}^{(l)}$ represents the weight of the connection between the $i$-th neuron of the $(l-1)$-th layer and the $j$-th neuron of the $l$-th layer, and $b_j^{(l)}$ denotes the bias term.

If we denote by $\Phi^{(l)} : \boldsymbol{y}^{(l-1)} \in \mathbb{R}^{N_{l-1}} \mapsto \boldsymbol{y}^{(l)} \in \mathbb{R}^{N_l}$ the mapping from the $(l-1)$-th layer to the $l$-th layer, the final output $\boldsymbol{y}^{(L+1)}$ of the network can be expressed as the composition of all these layer mappings:

$$\boldsymbol{y}^{(L+1)} = \Phi^{(L+1)} \circ \cdots \circ \Phi^{(1)}(\boldsymbol{y}^{(0)}). \tag{2.2}$$

The remarkable popularity of DFNNs is largely attributed to their ability to approximate any (piece-wise) continuous multivariate function (partially) with arbitrary precision, as guaranteed by the celebrated Universal Approximation Theorem [171, 172, 180, 174, 175, 111]. This theorem implies that any failure of a network to perform a task must arise from inadequate selection or calibration of weights and biases, or from an insufficient number of neurons in the hidden layers.

In the next section, we will dive into the details of such a theorem.

### 2.1.3 Universal approximation theorem

Numerous works have explored the approximation capabilities of neural networks for continuous functions of multiple variables. Originally inspired by Kolmogorov's superposition theorem (1957) [187], the late 1980s produced numerous significant results in neural network approximation theory. Wieland and Leighton (1987) [188] studied the ability of networks with one or two hidden layers, while Irie and Miyake (1988) [170] derived an integral representation formula with a predefined kernel, which could be realized using a three-layer neural network. Gallant (1988) [189] established that ANNs using the *cosine squasher* activation function possess the density property and can approximate any Fourier series. In 1989, several significant contributions emerged. Carroll and Dickinson [190] utilized the inverse Radon transform, while Cybenko [171] applied the Hahn-Banach and Riesz representation theorems to demonstrate uniform convergence on compact sets with continuous sigmoid functions. Funahashi (1989)

[173] approximated the integral representation of Miyake and Irie using a finite sum with a kernel expressible as the difference of two sigmoid functions. Hornik et al. [172] employed the Stone-Weierstrass theorem, using trigonometric functions to establish their results. Independently, Mhaskar and Micchelli (1992) [174] and Leshno et al. (1993) [175] provided a conclusive result, proving that the necessary and sufficient condition for a neural network to have universal approximation is that its activation function must be non-polynomial.

In the following, we will focus on the approach of Chen and Chen (1995) [111], where Tauber-Wiener functions and related techniques are used to derive a comprehensive universal approximation result.

**Definition 2.1.1** (Tauber-Wiener (TW) function)**.** A function $g : \mathbb{R} \to \mathbb{R}$ (continuous or discontinuous) is called a Tauber-Wiener (TW) function if all linear combinations of the form

$$\sum_{i=1}^{N} c_i g(\lambda_i x + \theta_i), \quad \lambda_i \in \mathbb{R},\ \theta_i \in \mathbb{R},\ c_i \in \mathbb{R},\ i = 1, 2, \ldots, N, \tag{2.3}$$

are dense in $C[a, b]$ for any interval $[a, b]$.

We will prove the following theorem:

**Theorem 2.1.1** (universal approximation)**.** Suppose that $g$ is a continuous function and $g \in \mathcal{S}'(\mathbb{R})$. Then $g \in (TW)$ if and only if $g$ is not a polynomial.

*Proof.* We will prove by contradiction. Suppose the set of all linear combinations $\sum_{i=1}^{N} c_i g(\lambda_i x + \theta_i)$ is not dense in $C[a, b]$. Then, by the Hahn-Banach extension theorem and the Riesz representation of continuous linear functionals, there exists a signed Borel measure $d\mu$ with $\text{supp}(d\mu) \subseteq [a, b]$, such that

$$\int_{\mathbb{R}} g(\lambda x + \theta) d\mu(x) = 0 \tag{2.4}$$

for all $\lambda \neq 0$ and $\theta \in \mathbb{R}$. Let $w \in S(\mathbb{R})$, a Schwartz rapidly decreasing function, then

$$\int_{\mathbb{R}} w(\theta) d\theta \int g(\lambda x + \theta) d\mu(x) = 0. \tag{2.5}$$

Let $\lambda x + \theta = u$, and change the order of integration. We obtain

$$\int_{\mathbb{R}} g(u) \int_{\mathbb{R}} w(\theta) d\mu \left( \frac{u - \theta}{\lambda} \right) = 0, \tag{2.6}$$

which is equivalent to

$$\hat{g}\big(\hat{w}(\cdot)\hat{d\mu}(\lambda \cdot)\big) = 0, \tag{2.7}$$

where $\hat{g}$ denotes the Fourier transform of $g$ in the sense of tempered distributions. To make sense of the left-hand side of the above equation, we need to show that $\hat{w}(t)\hat{d\mu}(\lambda t) \in \mathcal{S}(\mathbb{R})$.

Since $\text{supp}(d\mu) \subseteq [a,b]$, it is straightforward to show that $\hat{d\mu}(t) \in C^\infty(\mathbb{R})$. For each $k = 1, 2, \ldots$, there exists a constant $c_k$ such that

$$\left| \frac{\partial^k}{\partial t^k} \hat{d\mu}(t) \right| \leq c_k. \tag{2.8}$$

Consequently, $\hat{w}(t)\hat{d\mu}(t) \in \mathcal{S}(\mathbb{R})$. Since $d\mu \not\equiv 0$ and $\hat{d\mu}(t) \in C^\infty(\mathbb{R})$, there exists some $t_0 \neq 0$ with a neighborhood $(t_0 - \delta, t_0 + \delta)$ such that $\hat{d\mu}(t) \neq 0$ for all $t \in (t_0 - \delta, t_0 + \delta)$. Now, if $t_1 \neq 0$, let $\lambda = t_0/t_1$, then $\hat{d\mu}(\lambda t) \neq 0$ for all $t \in (t_1 - \delta/\lambda, t_1 + \delta/\lambda)$. Take any $\hat{w} \in C^\infty(t_0 - \delta/2\lambda, t_0 + \delta/2\lambda)$, then $\hat{w}(t)/\hat{d\mu}(\lambda t) \in \mathcal{S}(\mathbb{R})$, and by Eq. (2.7), we obtain

$$\hat{g}(\hat{w}(\cdot)) = \hat{g}\left( \frac{\hat{w}(\cdot)}{\hat{d\mu}(\lambda \cdot)} \hat{d\mu}(\lambda \cdot) \right) = 0, \tag{2.9}$$

This argument shows that for any fixed point $t^*$, there exists a neighborhood $[t^* - \eta, t^* + \eta]$ such that $\hat{g}(\hat{w}(\lambda)) = 0$ for all $\hat{w}$ with compact support in $[t^* - \eta, t^* + \eta]$, implying $\text{supp}(g) \subseteq \{0\}$. By the theory of distributions, $\hat{g}$ must be a linear combination of Dirac delta functions and their derivatives, which implies that $\hat{g}$ is a polynomial. □

For completeness, we also present the following theorem by Chen and Chen [111]:

**Theorem 2.1.2** (Universal approximation for functions[111])**.** Suppose $K$ is a compact set in $\mathbb{R}^d$, $U$ is a compact set in $C(K)$ and $\psi$ is a Tauber-Wiener function, then $\forall f \in U$ and any $\epsilon > 0$, there exist scaling factors $\{\xi_i\}_{i=1}^N$ and shifts $\{\theta_i\}_{i=1}^N$ both independent of $f$, and also coefficients $\{w_i[f]\}_{i=1}^N$ depending on $f$, such that

$$\left\| f(x) - \sum_{i=1}^{N} w_i[f] \psi(\xi_i x + \theta_i) \right\|_\infty < \epsilon. \tag{2.10}$$

Moreover, the coefficient $w_i[f]$ are continuous functionals on $U$.

### 2.1.4 Training process

Training an FNN involves finding an optimal (most of the times is just suboptimal) configuration of weights and biases that minimizes a predefined loss function, which quantifies the discrepancy between the predicted output of the network and the desired output. This process is framed as solving a high-dimensional, non-convex optimization problem. Various gradient-based methods are typically employed for this task, such as Stochastic Gradient Descent (SGD), momentum-based methods, Nesterov acceleration, and adaptive methods like ADAM (Adaptive Moment Estimation), RMSprop, and AdaGrad. Additionally, second-order methods, including Newton-like or quasi-Newton approaches like Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm and the Levenberg-Marquardt algorithm (LMA), are also applied. Methods like the scaled conjugate gradient are also widely used and suggest in practical software implementations, such as in `MATLAB`.

In regression problems, the loss function is often defined as the (Mean Squared Error) MSE between the network's output and the target output for a set of $M$ training samples. Let $\boldsymbol{y}_k^{(L+1)}$ represent the predicted output of the network for the $k$-th input sample $\boldsymbol{y}_k^{(0)}$, and let $\boldsymbol{d}_k$ be the corresponding desired output. The MSE loss function is expressed as:

$$E = \frac{1}{M}\sum_{k=1}^{M} ||\boldsymbol{d}_k - \boldsymbol{y}_k^{(L+1)}||_2^2. \tag{2.11}$$

Since one of the main task of a neural network is the generalization, Foresee and Hagan ([191]) showed that adding the $L^2$–regularization term $E_\omega = \sum_{j=1}^{N} \theta_j^2$, where $\theta_j$ is a trainable parameter of the network, to the cost function will maximize the posterior probability based on Bayes' rule. Hence, the total cost function is:

$$E_{total} = E + \lambda E_w, \tag{2.12}$$

where $\lambda$ is the regularization parameter that has to be tuned.

Regularization mitigates the risk of overfitting by constraining the complexity of the model. In particular, $L^2$–regularization discourages large weights, thus improving generalization. Conversely, $L^1$ regularization techniques, like the LASSO algorithm, can be employed to promote sparsity in the optimized weights or parameters. For example, [149] demonstrated the superior performance of $L^1$ regularization in simple regression tasks.

Solving the problem using the least-squares procedure involves finding the parameter values $p_j$ (including weights and biases) that minimize the total error. This is done through a method called *gradient descent*, which updates the weights proportionally to the negative of the partial derivative of the total error with respect to each parameter for the current input $\boldsymbol{y}_k^{(0)}$:

$$\Delta\theta_j = -\gamma \frac{\partial E^{(k)}}{\partial \theta_j}, \tag{2.13}$$

where $\gamma$ is a proportionality constant called the *learning rate*, such that $0 < \gamma < 1$. The computations of the partial derivatives of the loss function with respect to each weight and bias, even for deep architecture, can be straightforwardly unfolded by the use of the backpropagation algorithm [186], by applying the chain rule across the layers of the network.

**Examples of Gradient-based Optimization methods.** One of the simplest approaches for training an FNN is SGD. Unlike traditional gradient descent, which computes the gradient using the entire dataset, SGD uses only a single randomly selected data point (or a small batch of points) at each iteration. Given the large dimensionality of neural networks, first-order methods like SGD are popular due to their computational efficiency.

However, standard SGD can converge very slowly and may also get stuck in local minima. To mitigate this, several improved optimization techniques have been developed:

- Momentum: Adds a fraction ($0 < \mu < 1$) of the previous update $v_j^{(t)}$ to the current one $v_j^{(t+1)}$, helping to accelerate convergence by dampening oscillations in the gradient. Thus:

$$v_j^{(t+1)} = \mu v_j^{(t)} - \gamma \nabla_\theta E(\theta_j^{(t)}); \quad \theta_j^{(t+1)} = \theta_j^{(t)} + v_j^{(t+1)}. \tag{2.14}$$

- Nesterov Acceleration: A variant of momentum that looks ahead by computing gradients at an anticipated future point, leading to better convergence rates, thus computing $v_j^{(t+1)} = \mu v_j^{(t)} - \gamma \nabla_\theta E(\theta_j^{(t)} + \mu v_j^{(t)})$

- ADAM (Adaptive Moment Estimation): Combines the benefits of both momentum and adaptive learning rates. It adjusts the learning rate for each parameter individually based on estimates of the first and second moments (mean and variance) of the gradients.

The ADAM optimizer relies on the running estimates of the first and second moments of the gradient:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla E_t, \quad v_j^{(t)} = \beta_2 v_j^{(t-1)} + (1 - \beta_2)\nabla E_t^2, \tag{2.15}$$

where $\beta_1$ and $\beta_2$ are exponential decay rates (often set to 0.9 and 0.999, respectively). Then the bias-corrected moments are calculated as:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_j^{(t)}}{1 - \beta_2^t}. \tag{2.16}$$

Finally, the parameter update rule in ADAM is given by:

$$\theta_j^{(t)} = \theta_j^{(t-1)} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{2.17}$$

where $\alpha$ is the learning rate, and $\epsilon$ is a small constant to avoid division by zero.

ADAM is widely used due to its robust performance across a variety of tasks and the ability to handle sparse gradients.

**Quasi-Newton schemes.** Second-order methods, such as Newton-like or quasi-Newton methods, are more powerful in theory, as they consider the curvature of the loss function to make more informed updates. These methods, however, are computationally expensive due to the need to compute or approximate the Hessian matrix of second-order derivatives.

The Levenberg-Marquardt algorithm (LMA) is a widely used quasi-Newton method, particularly effective for training neural networks of moderate size. It combines elements of both the Gauss-Newton algorithm and gradient descent, allowing it to transition smoothly between the two methods. LMA is more robust than the Gauss-Newton method, making it more likely to find a solution even when starting far from the optimal minimum. However, for well-behaved functions and good initial guesses, LMA can be

slower than Gauss-Newton. It can also be interpreted as a Gauss-Newton method with a trust region approach.

LMA approximates the Hessian matrix $H_{\boldsymbol{\theta}}$ of the loss function using the Jacobian matrix $J_{\boldsymbol{\theta}}$, which contains partial derivatives of the network errors with respect to the parameters. Specifically, the Hessian is approximated as:

$$H_{\boldsymbol{\theta}} \approx J_{\boldsymbol{\theta}}^T J_{\boldsymbol{\theta}}. \tag{2.18}$$

This approximation leads to a simplified update rule for the parameters:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - (J_{\boldsymbol{\theta}}^T J_{\boldsymbol{\theta}} + \mu I)^{-1} J_{\boldsymbol{\theta}}^T \boldsymbol{e}, \tag{2.19}$$

where $\boldsymbol{e}$ is the vector of network errors, $\boldsymbol{\theta}$ is the vector of all trainable parameters (weights and biases), $\mu$ is a damping factor, and $I$ is the identity matrix. The damping factor $\mu$ controls the update strategy, interpolating between gradient descent and Gauss-Newton steps. The LMA is efficient for smaller networks but becomes computationally expensive for larger architectures due to the need to compute the Jacobian.

In `MATLAB`, the LMA is commonly used and is the default method for training networks that are not too large. It is particularly effective for function approximation tasks, where the training set is not overly large, and high precision is required.

### 2.1.5 Physics-informed Neural Networks (PINNs)

Physics-informed machine learning (PIML) algorithms are universal function approximators that incorporate governing physical laws directly into the learning process. By embedding this domain knowledge, they address the challenge of limited data availability often encountered in biological and engineering systems, where traditional ML models struggle to provide robust and reliable solutions.

In this section, and for the completeness of the presentation, we give a very brief introduction to the basic concept of PIML for the solution of DEs in the form of PDEs. Although, the concept of PIML can be extended to generic solution of nonlinear operators/functionals [66, 70, 125, 71].
Let's assume a set of $n_x$ points $\boldsymbol{x}_i \in \Omega \subset \mathbb{R}^d$ of the independent (spatial) variables which define the mesh in the domain $\Omega$, $n_{\partial\Omega}$ points along the boundary $\partial\Omega$ of the domain and $n_t$ points in the time interval, where the solution is sought. For our illustrations, let's consider a time-dependent PDE in the form of

$$\frac{\partial u}{\partial t} = L(\boldsymbol{x}, u, \nabla u, \nabla^2 u), \tag{2.20}$$

where $L$ is the partial differential operator acting on $u$ satisfying the boundary conditions $Bu = g$, in $\partial\Omega$, where $B$ is the boundary differential operator. Then, the solution with ML of the above PDE involves the solution of a minimization problem in the form:

$$\min_{\boldsymbol{P},\boldsymbol{Q}} E(\boldsymbol{P}, \boldsymbol{Q}) := \sum_{i=1}^{n_x} \sum_{j=1}^{n_t} \left\| \frac{\partial \Psi}{\partial t}(\cdot) - L(\boldsymbol{x}_i, \Psi(\cdot), \nabla\Psi(\cdot), \nabla^2\Psi(\cdot)) \right\|^2 \tag{2.21}$$

$$+\sum_{j=1}^{n_{\partial\Omega}} \|B\Psi(\cdot) - g\|^2 ,$$

where $\Psi(\cdot) := \Psi(\boldsymbol{x}_i, t_j, \mathcal{N}(\boldsymbol{x}_i, t_j, \boldsymbol{P}, \boldsymbol{Q}))$ represents an ML constructed function approximating the solution $u$ at $\boldsymbol{x}_i$ at time $t_j$ and $\mathcal{N}(\boldsymbol{x}_i, t_j, \boldsymbol{P}, \boldsymbol{Q})$ is an ML algorithm; $\boldsymbol{P}$ contains the parameters of the ML scheme (e.g., for a neural network the internal weights $\boldsymbol{W}$, the biases $\boldsymbol{B}$, the weights between the last hidden and the output layer $\boldsymbol{W}^o$), $\boldsymbol{Q}$ contains the hyperparameters (e.g., the parameters of the activation functions for a neural network, the learning rate, etc.). In order to solve the optimization problem (2.21), one usually needs quantities such as the derivatives of $\mathcal{N}(\boldsymbol{x}, \boldsymbol{P}, \boldsymbol{Q})$ with respect to $t, \boldsymbol{x}$ and the parameters of the ML scheme, such as the weights and biases. These can be obtained numerically using FD or other approximation schemes, or by symbolic or automatic differentiation [127, 93].

The above approach can be implemented also for solving systems of ODEs/DAEs as these may also arise by discretizing in space PDEs. For example, if we consider a 1D PDE and a grid of $n_x$ equally-spaced points $x_i$, $i = 1, \ldots, n_x$, with a space-step $\Delta x$, we can discretize the profile in space and approximate the spatial derivatives using e.g., central FD to get:

$$\frac{\partial u(t,x)}{\partial x} = \frac{u_{i+1}(t) - u_{i-1}(t)}{2\Delta x}, \quad \frac{\partial^2 u(t,x)}{\partial x^2} = \frac{u_{i+1}(t) - 2u_i(t) + u_{i-1}(t)}{\Delta x^2}. \tag{2.22}$$

Therefore, in this case, Eq. (2.20) can be reduced to a system of $n_x$ DAEs, given by:

$$\begin{aligned} &\frac{du_i(t)}{dt} = \tilde{L}(x_i, u_1(t), \ldots, u_{n_x}(t)), \quad i = 2, \ldots, (n_x - 1) \\ &\tilde{B}(u_1) = g(t, x_1), \quad \tilde{B}(u_{n_x} = g(t, x_{n_x}), \end{aligned} \tag{2.23}$$

where $\tilde{L}, \tilde{B}$ correspond to the discretization of the operators $L$ and $B$, respectively with FD. Then, the solution of the discretized system can be sought using $n_x$ (space-independent) ML constructed functions $\Psi_i(\cdot) := \Psi_i(t_j, \mathcal{N}_i(t, \boldsymbol{P}, \boldsymbol{Q}))$ approximating the solution $u_i$ at time $t_j$. Thus, the minimization problem given by Eq. (2.20) reduces to:

$$\begin{aligned} \min_{\boldsymbol{P},\boldsymbol{Q}} E(\boldsymbol{P},\boldsymbol{Q}) := &\sum_{i=1}^{n_x} \sum_{j=1}^{n_t} \left\| \frac{d\Psi_i}{dt}(\cdot) - \tilde{L}(\boldsymbol{x}_i, \Psi_1(\cdot), \ldots, \Psi_{n_x}(\cdot)) \right\|^2 + \\ &+ \left\| \tilde{B}\Psi_1(\cdot) - g(t_j, x_1) \right\|^2 + \left\| \tilde{B}\Psi_{n_x}(\cdot) - g(t_j, x_{n_x}) \right\|^2 . \end{aligned} \tag{2.24}$$

For the solution of the above optimization problem, one can compute the required numerical quantities such as the gradients with respect to the input and the unknown parameters; Yet, for deep learning schemes, but even for the simple case of single layer networks, when the number of hidden nodes is large, the solution of the resulting large-scale optimization problem is known to be difficult, often resulting in poor solutions as iterations stuck in local minima (for a detailed discussion about these problems, see, e.g., [147, 150, 151].)

The approach discussed above can be extended to a wide variety of functional problems beyond just PDEs, ODEs, and DAEs. This includes a variety of problems such

as integral equations, integro-differential equations, optimal control problems, transformations to normal forms [124], one-step observer problems [71], one-step feedback linearization control [70], and identifying low-dimensional slow-invariant manifolds [125].

This flexibility makes PIML applicable across a broad range of scientific computing disciplines, from classical physics and engineering to more complex fields like optimal control and bifurcation theory.

### 2.1.6 Challenges in Training Neural Networks

Training neural networks poses significant challenges, primarily due to the non-convex nature of the underlying optimization problem. The goal is to minimize a loss function (such as the MSE) by adjusting the network's weights and biases. However, this optimization problem is highly complex, as the solution space is filled with numerous local minima, especially in over-parameterized deep networks. Indeed, the number of local minima could be so high, that makes the probability of finding a globally optimal solution, among the many other local minima, close to zero in practice, particularly in deep architectures. Indeed, global optimization for non-convex problems is NP-hard, as shown by Murty and Kabadi (1985) [192], and training neural networks has been proven, according to Blum and Rivest (1988) [193], to be NP-complete, which is the most difficult class of NP-hard problem. This makes finding optimal solutions computationally prohibitive in practice and within a reasonable time-frame.

There is an inherent *paradox* in the foundations of neural network training: while the celebrated universal approximation theorem ensures that a neural network can approximate any continuous function to an arbitrary precision, the theorem lacks a constructive proof, meaning it does not provide a method to find the optimal weights. Additionally, achieving this optimal solution within polynomial time is not feasible due to the NP-completeness of the problem. In practice, hyperparameters such as the number of neurons $N$ number of layers $L$, learning rate $\gamma$, and batch size must be carefully tuned through extensive trial and error, which adds substantial complexity to the training process. Initialization of weights plays a crucial role, as poor initializations can lead to suboptimal solutions or significantly slow convergence. Therefore, training neural networks often requires multiple trials, each time exploring different configurations of network architectures and hyperparameters, further amplifying the computational cost.

Furthermore, the training of large-scale neural networks frequently requires extensive computational resources. Given the high-dimensional parameter space and the need for complex operations, modern neural networks often demand parallel hardware architectures, GPU-based systems, clusters of computers, or high-performance computing environments to train within a feasible time frame. Even with these resources, convergence remains slow, with training times often extending to hours or even days for large-scale networks, without a guarantee of reaching a truly optimal solution.

In the case of PINNs, the challenges of training become even more pronounced. PINNs integrate physical laws directly into the loss function, adding constraints derived from ODEs and PDEs. This results in a more complicated loss landscape, introducing additional stiffness, sharp gradients, and nonlinearities, which further complicate the op-

timization process. These factors exacerbate the non-convexity of the problem, hindering convergence even for relatively simple stiff low-dimensional ODEs.

Moreover, PINNs rely on penalties in the loss function to enforce initial and boundary conditions, which can skew the training process. For example, the network may overly focus on satisfying boundary or initial conditions at the expense of accurately solving the underlying PDE, or vice versa. Balancing these competing objectives is not trivial and further adds to the complexity of training. Efforts to bypass the penalty-based approach with unconstrained PINN-based methods (where the boundary conditions are encoded through transformed ansatz functions) have shown limitations in terms of accuracy, especially when handling complex geometries. These methods require the tuning of additional distance-based functions, which complicates the definition of the scheme itself.

The “curse of dimensionality" also impacts deep networks. While theoretically, neural networks should converge with complexity $O(N)$ in every dimension, as shown by Barron (1993) [153], the parameter space grows exponentially with the input dimension, making the search for optimal solutions even more difficult. In high-dimensional spaces, this exacerbates the computational demands, often requiring extensive computational resources such as GPU-based hardware or high-performance computing clusters to perform training in a reasonable timeframe. Even with these resources, convergence can be slow, often taking hours or even days for relatively large architectures, with no guarantee of achieving more than suboptimal solutions.

Furthermore, in comparison to classical numerical analysis methods, neural networks often fall short in interpolation tasks, where methods like orthogonal polynomials, Chebyshev, and Legendre polynomials can achieve exponential convergence rates and are computationally efficient when combined with techniques like the Fast Fourier Transform (FFT). When solving forward problems, neural networks also lack the numerical accuracy of traditional methods, often struggling to achieve better than $10^{-4}$ in terms of $L_2$ error, further emphasizing their limitations in both accuracy and computational efficiency [149].

In conclusion, while neural networks offer great flexibility and powerful learning capabilities, they are computationally demanding and prone to several optimization challenges. There is a growing need to develop new algorithms that combine the strengths of classical numerical analysis with ML techniques. The next sections will explore one such approach, where random projections and random features are leveraged to create efficient hybrid models that challenge the established flagships of numerical analysis solvers.

## 2.2 State-of-the-art of Random Projections

A more computationally efficient alternative to the above ML architectures is the Random Projection Neural Networks (RPNNs), which leverage fixed internal weights and biases to speed up calculations.

RPNNs are a subset of ANNs, which include Random Weights Neural Networks (RWNN) [194], Random Vector Functional Link Networks (RVFLN) [195, 196], Reservoir Computing (RC) [197], Extreme Learning Machines (ELM) [198], and Random Fourier Features Networks (RFFN) [158, 159]. The origins of this idea can be traced back to the *Gamba perceptron*, first introduced by Frank Rosenblatt [184] and reviewed by [185]. These methods share fundamental principles and have been used to mitigate the "curse of dimensionality" during training.

RPNNs have shown comparable or even superior accuracy in some cases, particularly for low-dimensional ODEs/DAEs, while significantly reducing training time compared to state-of-the-art numerical methods [69, 94, 91, 95, 67, 125, 181, 123].

An early example of random projection is found in the proof of the Johnson-Lindenstrauss (JL) theorem [199], which demonstrates that Euclidean distances can be preserved through a linear projection. More broadly, the hidden layers of RPNNs can be viewed as nonlinear random projections of input data. As shown in [158], these nonlinear projections are capable of preserving kernel distances, enabling the discovery of underlying nonlinear data structures. The core idea of RPNNs is to randomly predefine the weights between the input and hidden layers, along with the biases and activation function parameters. The weights connecting the last hidden layer to the output are then computed by solving a least-squares problem [198, 69, 123]. This least-squares solution constitutes the entire training process, eliminating the need for iterative training common in other ML approaches. Further motivation behind this idea, can be found in theoretical extensions of universal approximation to random fixed basis functions [153, 196, 198, 156, 157].

Within this context, in Fabiani et al. (2021) [69], the authors demonstrate for the first time that for low-dimensional nonlinear stationary PDEs, physics-informed random projection neural networks (PIRPNNs), trained using a regularized Gauss-Newton scheme, surpass FD and FEM in terms of convergence, accuracy, and computational efficiency. Dong and Li (2021) [91] introduced a method utilizing Extreme Learning Machines to solve time-dependent DEs through domain decomposition. The scheme was validated using the linear and nonlinear 1D Helmholtz equations, the 1D diffusion equation, and the 1D viscous Burger's equation. Additionally, a comparison was made with FEM regarding maximum error approximation and computational time. In Dong and Yang (2022) [94], the authors explore the use of ELM for PDEs. They propose a method to optimize the random initialization of weights in the hidden layer, improving ELM performance. Their approach outperforms traditional methods, such as the FEM, in solving both linear and nonlinear PDEs. In Fabiani et al. (2023) [95], the authors show that the integration of RPNNs with advanced numerical analysis and continuation methods outperforms conventional stiff time integrators for both ODEs and DAEs. In Bolager et al. (2024) [181], have proposed a gradient-based data-driven approach for

efficiently sampling weights of RPNNs in high-dimensional inputs, further extending a geometric approach proposed by Galaris et al. (2022) [45]. In Datar et al. (2024) [200], the authors propose using random sampling techniques for hidden weights and biases in neural networks to solve PDEs, outperforming traditional gradient-based optimization in both training time and accuracy. Their method excels in both time-dependent and static PDEs, leveraging neural basis functions for the spatial domain.

Overall, RPNNs represent a promising advancement in training ANNs, showing also promising performance in solving PDEs in both computational efficiency and accuracy over traditional methods. As ongoing research continues to refine weight sampling techniques and integration with advanced numerical strategies, RPNNs are poised to play a crucial role in future computational science and engineering applications.

## 2.3 Preliminaries on Linear Random Projection

A key result connecting the conceptually equivalent methods mentioned above is the well-known Johnson-Lindenstrauss (JL) Lemma [199]. This lemma asserts that there exists an approximate isometry map $\boldsymbol{F} : \mathbb{R}^d \to \mathbb{R}^k$ for input data $\boldsymbol{x} \in \mathbb{R}^n$, induced by a random matrix $R$, given by:

$$\boldsymbol{F}(\boldsymbol{x}) = \frac{1}{\sqrt{k}} R\boldsymbol{x}, \tag{2.25}$$

where $R = [R_{ij}] \in \mathbb{R}^{k\times d}$ is a random matrix with entries drawn i.i.d. from a normal distribution.

By projection, we refer to the case where, given a data point $\mathbf{x} \in \mathbb{R}^d$ and a set of $p$ vectors $\{\mathbf{u}_1, \ldots, \mathbf{u}_p\}$ with $p \ll d$, arranged as columns in the matrix $U \in \mathbb{R}^{d\times p}$, the projection of $\mathbf{x}$ onto the column space of $U$ is:

$$\hat{\mathbf{x}} = U\boldsymbol{\alpha}, \quad \text{where} \quad \boldsymbol{\alpha} = (U^\top U)^{-1} U^\top \mathbf{x}. \tag{2.26}$$

The coefficients $\boldsymbol{\alpha}$ are determined by ensuring that the reconstruction residual, $r = \mathbf{x} - \hat{\mathbf{x}}$, is orthogonal to the space spanned by $U$. This aligns with the formula for coefficients in linear regression. Furthermore, the projected point $\hat{\mathbf{x}}$ only coincides with the original point $\mathbf{x}$ if $\mathbf{x} \in \text{span}(U)$.

In random projection, data is projected using a matrix with randomly sampled entries, independent of the data itself. Unlike PCA, which constructs projections based on data characteristics, random projection does not necessitate learning from the data. Nevertheless, it effectively preserves geometric properties in an approximate manner, making it a valuable technique in dimensionality reduction and data analysis.

The JL lemma justifies why this random projection approach works.

**Theorem 2.3.1** (Johnson and Lindenstrauss)**.** Let $\mathcal{X}$ be a set of $n$ points in $\mathbb{R}^d$. Then, for any $\epsilon \in (0, 1)$ and $k \in \mathbb{N}$ such that $k \geq O\left(\frac{\ln n}{\epsilon^2}\right)$, there exists a mapping $\boldsymbol{F} : \mathbb{R}^d \to \mathbb{R}^k$ such that

$$(1-\epsilon)\|\boldsymbol{u}-\boldsymbol{v}\|^2 \leq \|\boldsymbol{F}(\boldsymbol{u}) - \boldsymbol{F}(\boldsymbol{v})\|^2 \leq (1+\epsilon)\|\boldsymbol{u}-\boldsymbol{v}\|^2 \quad \forall\, \boldsymbol{u}, \boldsymbol{v} \in \mathcal{X}. \tag{2.27}$$

The proof of this theorem, while deterministic, utilizes probabilistic methods alongside Kirszbraun's theorem, resulting in an extension mapping [199]. One such embedding is represented by a linear projection matrix with randomly generated elements, as formalized in the following lemma.

**Lemma 2.3.1.** Let $\mathcal{X}$ be a set of $n$ points in $\mathbb{R}^d$, $\epsilon \in (0,1)$, and let $\boldsymbol{F}(\boldsymbol{u})$ be the random projection defined by

$$\boldsymbol{F}(\boldsymbol{u}) = \frac{1}{\sqrt{k}} R\,\boldsymbol{u}, \quad \boldsymbol{u} \in \mathbb{R}^d,$$

where $R = [r_{ij}] \in \mathbb{R}^{k\times d}$ has entries that are i.i.d. random variables drawn from a normal distribution. Then, for all $\boldsymbol{u} \in \mathcal{X}$,

$$(1-\epsilon)\|\boldsymbol{u}\|^2 \leq \|\boldsymbol{F}(\boldsymbol{u})\|^2 \leq (1+\epsilon)\|\boldsymbol{u}\|^2$$

holds with probability $p \geq 1 - 2\exp\left(-(\epsilon^2 - \epsilon^3)\frac{k}{4}\right)$.

*Proof.* Here, we give only a sketch of the proof. Let be $R$ a matrix with rows $(\boldsymbol{R}_1, \ldots, \boldsymbol{R}_k)^T \in \mathbb{R}^{k\times d}$, with elements $r_{ij} \sim \mathcal{N}(0,1)$. Note that $\boldsymbol{R}_i$ are random vectors. Let $\boldsymbol{u} \in \mathbb{R}^d$. First, we consider the norm of the transformation, and we obtain:

$$\|\frac{1}{\sqrt{k}} R\boldsymbol{u}\|_2^2 = \frac{1}{k}\sum_{i=1}^{k} (< \boldsymbol{R}_i, \boldsymbol{u} >)^2. \tag{2.28}$$

Then we consider the mean value:

$$\mathbb{E}\|\frac{1}{\sqrt{k}} R\boldsymbol{u}\|_2^2 = \frac{1}{k}\sum_{i=1}^{k} \mathbb{E}[(< \boldsymbol{R}_i, \boldsymbol{u} >)^2] = \mathbb{E}[(< \boldsymbol{R}_i, \boldsymbol{u} >)^2]. \tag{2.29}$$

Now, let us recall that the sum of two Gaussian distribution is another Gaussian distribution, i.e., with an abuse of notation $\mathcal{N}(\mu_1, \sigma_1^2) + \mathcal{N}(\mu_2, \sigma_2^2) \sim \mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$. Therefore:

$$\begin{aligned} \langle \boldsymbol{R}_i, \boldsymbol{u} \rangle &= r_{i1}u_1 + r_{i2}u_2 + \ldots + r_{id}u_d \sim \\ &\sim \mathcal{N}(0, u_1^2) + \mathcal{N}(0, u_2^2) + \ldots + \mathcal{N}(0, u_d^2) \sim \mathcal{N}(0, \|\boldsymbol{u}\|_2^2) \end{aligned} \tag{2.30}$$

From the above, and combining with Eq. (2.29), we obtain that:

$$\mathbb{E}\left\|\frac{1}{\sqrt{k}}\boldsymbol{R}\boldsymbol{u}\right\|_2^2 = \mathbb{E}\left[\langle \boldsymbol{R}_i, \boldsymbol{u}\rangle^2\right] = \|\boldsymbol{u}\|_2^2. \tag{2.31}$$

In particular, Eq. (2.31) imply that the norm of the random transformation, as in Eq. (2.28), is a Chi-squared random variable $Z$ with k–degrees of freedom. Then the theorem holds following the next lemma about the fact that The chi-squared distribution exhibits strong concentration around its mean. □

**Lemma 2.3.2.** Let $Z$ be a Chi-squared random variable with $k$ degrees of freedom, then:

$$\mathbb{P}\left[|\mathbb{E}[Z] - Z| \geq \epsilon\, \mathbb{E}[Z]\right] \leq 2\exp\left(-\frac{k\epsilon^2}{8}\right). \tag{2.32}$$

Similar results have been demonstrated for random projections using distributions other than the normal distribution (e.g., see [201, 202]).

## 2.4 Nonlinear random projections

Linear dimensionality reduction techniques such as SVD and Principal Component Analysis (PCA) use a projection matrix to transform data into a lower-dimensional space, either to enhance data representation or to distinguish classes more effectively. Interestingly, random projection methods demonstrate that it is not always necessary to learn this projection matrix; instead, sampling its elements randomly can still preserve the geometric properties of the data, as justified by the Johnson-Lindenstrauss (JL) theorem.

Notwithstanding, constructing a feature space ($x \rightarrow F(x)$) that aims at preserving Euclidean distance may not consistently constitute the optimal approach when undertaking ML tasks. In this context, the effectiveness of nonlinear explicit random lifting operator, as realized by the hidden layer(s) of RPNNs, in preserving kernel distances has been investigated in detail in [158, 159]. Besides, as it has been shown (see, e.g., [153, 196, 203]), appropriately constructed nonlinear random projections may outperform such simple linear random projections.

However, nonlinear random projection involves more complex theoretical analysis. This type of projection can be viewed as a linear random projection followed by a nonlinear transformation. Two foundational studies on nonlinear random projection are the Random Fourier Features (RFF) method by Rahimi and Recht (2007) [158] and the Random Kitchen Sinks (RKS) approach by Rahimi and Recht (2008) [159].

When data naturally reside on non-linear structures or manifolds, applying conventional linear methods directly in the original space is often ineffective. This is because linear models cannot capture the underlying geometric or topological properties of the data.

In ML, the kernel trick is a common approach that is employed to implicitly map data into a high-dimensional, potentially infinite-dimensional space, where complex, non-linear relationships between data points can be captured by linear methods. In such spaces, every function behaves linearly, and thus the linear separation or regression of data points becomes feasible.

The *kernel trick* facilitates the generation of features for algorithms that rely solely on the inner product between pairs of input vectors, i.e., $\langle \phi(\mathbf{u}), \phi(\mathbf{v}) \rangle = K(\mathbf{u}, \mathbf{v})$, where $\phi$ represents an implicit feature mapping. However, the use of large training datasets results in significant computational and storage costs. The computation of kernels is computationally intensive due to the transformation of points into a potentially high-dimensional space, followed by the evaluation of their inner products within the Reproducing Kernel Hilbert Space (RKHS). To mitigate this, instead of using the implicit

mapping provided by the kernel trick, we seek an explicit transformation of the data into a low-dimensional Euclidean inner product space via a randomized feature map $\mathbf{z} : \mathbb{R}^d \to \mathbb{R}^D$. This approximation is given by:

$$K(\mathbf{u}, \mathbf{v}) = \langle \phi(\mathbf{u}), \phi(\mathbf{v}) \rangle \approx \mathbf{z}(\mathbf{u})^\top \mathbf{z}(\mathbf{v}). \tag{2.33}$$

RFFs are utilized to expedite kernel methods, as introduced by Rahimi and Recht (2007) [158]. RFF projects data into a low-dimensional space, contrasting with kernel methods that project data into a potentially high-dimensional space.

RFF are effective for positive definite kernels that are shift-invariant, also known as stationary kernels, i.e., $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{x} - \mathbf{y})$. Consider the inverse Fourier transform of the kernel function $k$:

$$k(\mathbf{x} - \mathbf{y}) = \int_{\mathbb{R}^d} \hat{k}(\mathbf{w}) e^{j\mathbf{w}^\top(\mathbf{x}-\mathbf{y})} \, d\mathbf{w} = \mathbb{E}_{\mathbf{w}}[\zeta_{\mathbf{w}}(\mathbf{x})\zeta_{\mathbf{w}}(\mathbf{y})^*] \tag{2.34}$$

where $\hat{k}(\mathbf{w})$ is the Fourier transformed of $k$, with $\mathbf{w}$ the frequency and $j$ the imaginary unit, and we select $\zeta_{\mathbf{w}} = e^{j\mathbf{w}^\top \mathbf{x}}$. Note that the Bochner's theorem guarantees that its Fourier transform is a proper probability distribution. Therefore, so $\zeta_{\mathbf{w}}(\mathbf{x})\zeta_{\mathbf{w}}(\mathbf{y})^*$ is an unbiased estimate of $k(x, y)$ when $\mathbf{w}$ is drawn from $\hat{k}$.

Let us also note that the kernel function $k$ and the transformed kernel $\hat{k}$ are real-valued, so the sine component of $e^{j\mathbf{w}^\top(\mathbf{x}-\mathbf{y})}$ can be ignored and replaced with $\cos(\mathbf{w}^\top(\mathbf{x}-\mathbf{y}))$. Hence, $\zeta_{\mathbf{w}}(\mathbf{x}) = \cos(\mathbf{w}^\top \mathbf{x})$.

Here, for the sake of completeness of the presentation, we restate the following theorem [158],

**Theorem 2.4.1** (Low-distortion of nonlinear kernel-embedding [158])**.** Let $K$ be a positive definite shift-invariant kernel $K(\boldsymbol{u}, \boldsymbol{v}) = K(\boldsymbol{u} - \boldsymbol{v})$. Consider the Fourier transform $p_{K,\boldsymbol{\alpha}} = \hat{\mathcal{F}}[K]$ of the kernel $K$, resulting a probability density function (pdf) $p_{K,\boldsymbol{\alpha}}$ in the frequency space $A$: $p_{K,\boldsymbol{\alpha}}(\alpha) = \frac{1}{2\pi} \int e^{j\alpha\Delta} K(\Delta) d\Delta$, and draw $N$ i.i.d. samples weights $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_N \in \mathbb{R}^d$ from $p_{K,\boldsymbol{\alpha}}$. Define

$$\boldsymbol{\phi}_N(\boldsymbol{u}; \boldsymbol{\alpha}) \equiv \sqrt{\frac{1}{N}} [\cos(\boldsymbol{\alpha}_1^T \boldsymbol{u}), \ldots, \cos(\boldsymbol{\alpha}_n^T \boldsymbol{u}), \sin(\boldsymbol{\alpha}_1^T \boldsymbol{u}), \ldots, \sin(\boldsymbol{\alpha}_n^T \boldsymbol{u})]. \tag{2.35}$$

Then,$\forall \epsilon > 0$

$$\begin{aligned} \mathbb{P}\Big[ (1-\epsilon) K(\boldsymbol{u}, \boldsymbol{v}) \leq \boldsymbol{\phi}_N(\boldsymbol{u})^T \boldsymbol{\phi}_N(\boldsymbol{v}) \leq (1+\epsilon) K(\boldsymbol{u}, \boldsymbol{v}) \Big] \geq \\ \geq 1 - O\left( \exp\left( -\frac{N\epsilon^2}{4(d+2)} \right) \right), \end{aligned} \tag{2.36}$$

where $\mathbb{P}$ stands for the probability function.

An equivalent result can be obtained by employing only cosine as the activation function and random biases in $[0, 2\pi]$ [158]. More generally, there is no constraint in considering trigonometric activation functions, as sigmoid and RBFs have equivalently shown remarkable results [69, 95, 156, 94, 91, 181, 125]. Here we restate, the following theorem [159, 160]:

**Theorem 2.4.2.** (cf. Theorem 3.1 and 3.2 in [160]) Consider the parametric set activation functions on $X \subseteq \mathbb{R}^d$, $\psi(\boldsymbol{x};\boldsymbol{\alpha}) : X \times A \to \mathbb{R}$ parametrized by random variables $\boldsymbol{\alpha}$ in $A$, that satisfy $\sup_{\boldsymbol{x},\boldsymbol{\alpha}} |\phi(\boldsymbol{x},\boldsymbol{\alpha})| \leq 1$. Let $p$ be a probability distribution on $A$ and $\mu$ be a probability measure on $X$ and the corresponding norm $\|f\|_{L^2(\mu)} = \int_X f(\boldsymbol{x})^2 \mu(d\boldsymbol{x})$. Define the set:

$$G_p \equiv \left\{ g(\boldsymbol{x}) = \int_A w(\boldsymbol{\alpha})\phi(\boldsymbol{x};\boldsymbol{\alpha})d\boldsymbol{\alpha} : \|g\|_{p(\boldsymbol{\alpha})} < \infty \right\}, \|g\|_p := \sup_{\boldsymbol{\alpha}\in A} \|w(\boldsymbol{\alpha})/p(\boldsymbol{\alpha})\|. \tag{2.37}$$

Fix a function $g^*$ in $G_p$. Then, for any $\delta > 0$, there exist $N \in \mathbb{N}$, and $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_N$ of $\boldsymbol{\alpha}$ drawn i.i.d. from $p$, and a function $\hat{g}$ in the random set of finite sums

$$\hat{G}_{\boldsymbol{\alpha}} \equiv \left\{ \hat{g} : \hat{g}(\boldsymbol{x}) = \sum_{j=1}^{N} w_j \phi(\boldsymbol{x};\boldsymbol{\alpha}_j) \right\} \tag{2.38}$$

such that

$$\sqrt{\int_X (g^*(\boldsymbol{x}) - \hat{g}(\boldsymbol{x}))^2 d\boldsymbol{\mu}(\boldsymbol{x})} \leq \frac{||g^*||_p}{\sqrt{N}} \left(1 + \sqrt{2\log\frac{1}{\delta}}\right), \tag{2.39}$$

holds with probability at least $1-\delta$. Moreover, if $\phi(\boldsymbol{x};\boldsymbol{\alpha}) = \varphi(\boldsymbol{\alpha}\cdot\boldsymbol{x})$, for a $L$-Lipschitz function $\psi$, the above approximation is uniform (i.e., in the supremum norm).

The above Theorem implies that the function class $G_p$ can be approximated to any accuracy when $N \to \infty$. Moreover (see [160]) this class of functions is dense in RKHS defined by $\phi$ and $p$. For a detailed discussion on the pros and cons of function approximation with such random bases, see [203].

## 2.5 Random projections in Neural Networks

Let us examine a single-output, single-hidden-layer FNN, represented by a function $f_N : \mathbb{R}^d \to \mathbb{R}$ with a *a priori* fixed matrix of *internal weights* $A \in \mathbb{R}^{N\times d}$, where $N$ rows are denoted by $\boldsymbol{\alpha}_j \in \mathbb{R}^{1\times d}$ and *biases* $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_N) \in \mathbb{R}^N$:

$$f_N(\boldsymbol{x};\boldsymbol{w},\beta^o,A,\boldsymbol{\beta}) = \sum_{j=1}^{N} w_j \psi(\boldsymbol{\alpha}_j\cdot\boldsymbol{x} + \beta_j) + \beta^o = \sum_{j=1}^{N} w_j \psi_j(\boldsymbol{x}) + \beta^o \tag{2.40}$$

Here, $N$ represents the number of neurons (nodes), $d$ is the dimension of the input $\boldsymbol{x} \in \mathbb{R}^{d\times 1}$, $\beta^o$ is a constant offset (referred to as the *output bias*), and $\psi : \mathbb{R} \to \mathbb{R}$ is the *activation* (transfer) function. For fixed parameters $\alpha_j$ and $\beta_j$, we denote this as a fixed *basis function* $\psi_j$. The weights $\boldsymbol{w} = (w_1, \ldots, w_N)^T \in \mathbb{R}^{N\times 1}$ are the *external (readout) weights* connecting the hidden layer to the output layer. In RPNNs, the only trainable parameters of the network are $\boldsymbol{w}$ and the offset bias $\beta^o$.

Activation functions, which are often inspired by the behavior of neuron spikes (e.g., sigmoid functions such as the logistic sigmoid and hyperbolic tangent), are critical in determining the expressive capacity of ANNs. However, as shown by Mhaskar and

Micchelli (1992) and Leshno et al. (1993) [174, 175], any non-polynomial function can achieve *universal approximation*, moving beyond biological motivations. Consequently, our focus in the subsequent theoretical sections will be on generic non-polynomial and infinitely differentiable activation functions. To illustrate their practical capabilities, our numerical experiments will employ the well-known logistic sigmoid function.

Furthermore, in the following sections, when no ambiguity arises, we will denote RPNNs with $N+1$ neurons, which depend solely on external weights $\boldsymbol{w}$, using the notation $f_N(\cdot;\boldsymbol{w})$, thus omitting explicit references to the internal fixed parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.

### 2.5.1 Linear independence of basis functions

In this section, we review the results presented in Ito (1996) [204] about the sufficient conditions to obtain a set of linear independent basis functions.

In particular in this section, we will show that to have a system of independent basis functions $\{\psi_1, \psi_2, \ldots, \psi_N\}$, in a neural network is sufficient to consider *slowly increasing* nonlinear *plane waves*, such that the support of their Fourier transform $\mathcal{F}[\psi]$ has an open subset, as basis function and make any pair of internal weights and biases $(\boldsymbol{\alpha}_j, \beta_j) \neq \pm(\boldsymbol{\alpha}_{j'}, \beta_{j'}), \forall j \neq j'$.

Note that, a plane wave function is a function $\phi : \mathbb{R}^d \to \mathbb{R}$ that can be represented as the composition of a univariate function with an affine transformation: $\phi(\mathbf{x}) = \psi(\mathbf{a} \cdot \mathbf{x} + \mathbf{b})$, where $\psi : \mathbb{R} \to \mathbb{R}$ is a univariate function, and $\mathbf{a}$ and $\mathbf{b}$ are vectors in $\mathbb{R}^d$.

**Remark 2.5.1.** As we will discuss later, the condition $(\boldsymbol{\alpha}_j, \beta_j) \neq \pm(\boldsymbol{\alpha}_{j'}, \beta_{j'}), \forall j \neq j'$ outlined in [204], does not universally guarantee that the basis functions are independent, as there exist certain counter-examples that can be identified [204]. However, such functions typically exhibit discontinuities and irregular shapes, and have not been employed in ANNs. In practical applications, widely used activation functions such as logistic sigmoid, hyperbolic tangent, Gaussian RBF, and many others, with weights following the aforementioned criterion, have been shown by the author to be effectively independent. Therefore, we will proceed under the assumption that the condition in [204] is sufficient.

Let us observe first that distributions supported at the origin can only be expressed as linear combinations of the Dirac delta function $\delta_0$ and its derivatives $\delta_0^{(k)}$. In the Fourier domain, the Fourier transform of $\delta_0$ is constant, whereas the Fourier transform of $\delta_0^{(k)}$ is $(i\xi)^k$, corresponding to polynomials of degree $k$ in $\xi$. Hence, the linear independence of such distributions reduces to the linear independence of the corresponding polynomials in the Fourier domain. Specifically, for a set of coefficients $a_k$, if

$$\sum_{k=0}^{n} a_k \delta_0^{(k)} = 0, \tag{2.41}$$

then, in the Fourier domain, this becomes

$$\sum_{k=0}^{n} a_k (i\xi)^k = 0. \tag{2.42}$$

The linear independence of the polynomials $(i\xi)^k$ ensures that all coefficients $a_k$ must be zero. Thus, checking the linear independence of the distributions $\delta_0^{(k)}$ is equivalent to verifying the linear independence of these polynomials.

Now, define the set

$$\mathcal{D}^0(\mathbb{R}^d) = \{\varphi \in \mathcal{D}(\mathbb{R}^d) \mid O \notin \operatorname{supp}(\varphi)\}, \tag{2.43}$$

where $O$ denotes the origin.

**Definition 2.5.1.** Let $T_i$, $i = 1, \ldots, n$, be distributions defined on $\mathbb{R}^d$. If the equation

$$\sum_{i=1}^{n} a_i \langle T_i, \varphi \rangle = 0 \quad \text{for all} \quad \varphi \in \mathcal{D}(\mathbb{R}^d) \tag{2.44}$$

implies that $a_i = 0$, $i = 1, \ldots, n$, then the distributions $T_i$ are called *strictly* linearly independent.

Strictly linearly independent distributions are also linearly independent. If the distributions $T_i$ are defined by

$$\langle T_i, \varphi \rangle = \int_{\mathbb{R}^d} f_i \varphi \, dx, \tag{2.45}$$

where the functions $f_i$ are locally integrable, then the two notions are equivalent. In this case, the condition

$$\sum_{i=1}^{n} a_i \langle T_i, \varphi \rangle = 0 \quad \text{for all} \quad \varphi \in \mathcal{D}^0(\mathbb{R}^d) \tag{2.46}$$

is equivalent to

$$\sum_{i=1}^{n} a_i f_i(x) = 0 \quad \text{for almost all} \quad x \in \mathbb{R}^d, \tag{2.47}$$

which, in turn, is equivalent to

$$\left\langle \sum_{i=1}^{n} a_i T_i, \varphi \right\rangle = 0 \quad \text{for all} \quad \varphi \in \mathcal{D}(\mathbb{R}^d). \tag{2.48}$$

Next, define a line $L_{\boldsymbol{w}}$ and a hyperplane $H_{\boldsymbol{w}}$ for a nonzero vector $\boldsymbol{w}$ as

$$L_{\boldsymbol{w}} = \left\{ \boldsymbol{x} \in \mathbb{R}^d ; \boldsymbol{x} = t\boldsymbol{w}, \, -\infty < t < \infty \right\}, \quad H_{\boldsymbol{w}} = \left\{ \boldsymbol{x} \in \mathbb{R}^d ; \boldsymbol{w} \cdot \boldsymbol{x} = 0 \right\}. \tag{2.49}$$

Then, $\mathbb{R}^d = L_{\boldsymbol{w}} \oplus H_{\boldsymbol{w}}$. We say that vectors $\boldsymbol{w}_i$ point in distinct directions if they are nonzero and the lines $L_{\boldsymbol{w}_i}$ intersect only at the origin.

Now we can state the following theorem (proof available in [204]):

**Theorem 2.5.1.** Let $T_{0j}$, $j = 1, \dots, n_0$, be linearly independent distributions on $\mathbb{R}^d$ supported at the origin. Let $\boldsymbol{w}_i \in \mathbb{R}^d$, $i = 1, \dots, n$, be vectors in distinct directions. For each $i$, let $T_{ij} \in \mathcal{D}'(\mathbb{R}^d)$, $j = 1, \dots, n_i$, be strictly linearly independent distributions supported by $L_{\boldsymbol{w}_i}$. Then, $T_{ij}$, $i = 0, \dots, n$, $j = 1, \dots, n_i$, are linearly independent.

**Remark 2.5.2.** Let $\boldsymbol{w}_i \in \mathbb{R}^d$, $i = 1, \dots, n$, be vectors in distinct directions, and let $T_i \in \mathcal{D}'(\mathbb{R}^d)$, $i = 1, \dots, n$, be distributions with $\text{supp}(T_i) \subseteq L_{\boldsymbol{w}_i}$. Suppose the support of each $T_i$ contains a point other than the origin. Then, the $T_i$ are strictly linearly independent.

The Fourier transform of elementary trigonometric functions $\sin(at)$ and $\cos(at)$ yields a distribution whose support is at two distinct points.

The following remark generalizes the linear independence of differently scaled trigonometric functions. Any number of strictly linearly independent distributions can be generated by scaling a distribution with compact support that includes a point other than the origin:

**Remark 2.5.3.** Let $T_i \in \mathcal{D}'(\mathbb{R})$, $i = 1, \dots, n$, be distributions such that the values of $\inf \text{ supp}(T_i)$ and $\sup \text{ supp}(T_i)$, for $i = 1, \dots, n$, are distinct. Then, the $T_i$'s are linearly independent. Moreover, at most one of these distributions can be supported at the origin. The remaining $T_i$'s are strictly linearly independent.

### *Linear independent scaled and rotated plane waves*

Tempered distributions $T_i$ are linearly independent if and only if their Fourier transforms $\mathcal{F}T_i$ are linearly independent.

When we need to represent explicitly the space where a distribution $T$ is defined, we denote the variable as $T(x)$. Let $R(x)$ and $T(y)$ be distributions defined on $\mathbb{R}^d$ and $\mathbb{R}^d$, respectively. Then, the tensor product $R(x) \otimes T(y)$ of the two distributions is well-defined on the product space $\mathbb{R}^d \times \mathbb{R}^d = \mathbb{R}^{2d}$. The Fourier transform of the tensor product is the tensor product $\mathcal{F}R \otimes \mathcal{F}T$ of the Fourier transforms.

The plane wave $g(w \cdot x - t)$ can be regarded as a tensor product of a function on $L_w$ and a constant 1 on $H_w$.

For a slowly increasing function $g$ on $\mathbb{R}$, set $G(x) = g(w \cdot x)$, where $w \in \mathbb{R}^d, w \neq 0$. Then, $G$ is also a slowly increasing function on $\mathbb{R}^d$. Hence, its Fourier transform is well-defined.

For convenience, we introduce a dual notion of strict linear independence.

**Definition 2.5.2.** We call functions on $\mathbb{R}^d$ *completely linearly independent* if they are slowly increasing and their Fourier transforms are strictly linearly independent.

We report a theorem by [204]:

**Theorem 2.5.2.** Let $p_i, i = 1, \dots, n_0$, be linearly independent polynomials on $\mathbb{R}^d$, and let $g_i, i = 1, \dots, n$, and $j = 1, \dots, n_i$, be slowly increasing functions on $\mathbb{R}$. Suppose that $g_{i,j}, j = 1, \dots, n_i$, are completely linearly independent for each $i$. Then, for any vectors $w_i \in \mathbb{R}^d, i = 1, \dots, n$, in distinct directions, the polynomials $p_i$ and the plane waves $g_{i,j}(w_i \cdot x)$ are linearly independent.

Hence, it is evident that differently scaled and distinctly rotated trigonometric plane waves are linearly independent.

A single distribution is strictly linearly independent if its support has a point other than the origin. The inverse Fourier transform of such a distribution is not a polynomial.

**Remark 2.5.4.** Let $p_i, i = 1, \ldots, n_0$, be linearly independent polynomials on $\mathbb{R}^d$ and let $g_i, i = 1, \ldots, n$, be slowly increasing non-polynomial functions on $\mathbb{R}$. Then, for vectors $w_i \in \mathbb{R}^d, i = 1, \ldots, n$, in distinct directions, the polynomials $p_i$ and the plane waves $g_i(w_i \cdot x)$ are linearly independent.

Thus, by rotation, we can create infinitely many linearly independent plane waves.

One of the meanings of Theorem 2.5.2 can be expressed in the following corollary:

**Corollary 2.5.1** (of Theorem 2.5.2)**.** Let $p_i, i = 1, \ldots, n_0$, be linearly independent polynomials on $\mathbb{R}^d$ and let $g_i, i = 1, \ldots, n$, and $j = 1, \ldots, n_i$, be slowly increasing non-polynomial functions on $\mathbb{R}$ such that their Fourier transforms $\mathcal{F}g_{i,j}$ are functions. Furthermore, let $g_{i,j}, j = 1, \ldots, n_i$, be linearly independent for each $i$. Then, for any vectors $w_i \in \mathbb{R}^d, i = 1, \ldots, n$, in distinct directions, the polynomials $p_i$ and the plane waves $g_{i,j}(w_i \cdot x)$ are linearly independent.

In many cases of neural networks, activation functions are differentiable functions and their derivatives are square integrable. Then, the Fourier transforms of the derivatives are functions.

### *Uniformly scaled and variously shifted plane waves*

In neural network theory, only a single type of activation function, say $g$, is often used. Accordingly, we have a particular interest in the case where the plane waves are of the form $g(w_i \cdot x - t_i)$. In this section, we treat the linear independence of polynomials and plane waves uniformly scaled direction-wise.

**Lemma 2.5.1.** Let $w$ be a nonzero vector in $\mathbb{R}^d$, let $t_i, i = 1, \ldots, n$, be distinct constants, and let $g$ be a slowly increasing function on $\mathbb{R}$. If the support of $\mathcal{F}g$ contains an open subset, then the plane waves $g(w \cdot x - t_i)$ are completely linearly independent, and their Fourier transforms are strictly linearly independent.

Note that, by assumption, the function $g$ in the lemma is non-polynomial.

**Remark 2.5.5.** The assumption that the support of $\mathcal{F}g$ contains an open subset is important. Although $\sin t$ is a slowly increasing non-polynomial function, $\sin t + \sin(t + \pi) = 0$ on $\mathbb{R}$. Hence, a sum of a plane wave defined by $\sin t$ and its shift can be zero in $\mathbb{R}^e$. Note that the support of the Fourier transform of $\sin t$ is nowhere dense.

**Theorem 2.5.3.** Let $P_i, i = 1, \ldots, n_0$, be linearly independent polynomials on $\mathbb{R}^d$, let $w_i, i = 1, \ldots, n$, be vectors in $\mathbb{R}^d$ in distinct directions, and let $g$ be a slowly increasing non-polynomial function on $\mathbb{R}$. Suppose that the support of $\mathcal{F}g$ has an open subset. Then, the polynomials $P_i, i = 1, \ldots, n_0$, and plane waves $g(w_i \cdot x - t_j), i = 1, \ldots, n, j = 1, \ldots, n_i$, are linearly independent if $t_j, j = 1, \ldots, n_i$, are distinct constants for each $i$.

### *Diversely scaled plane waves*

In this section, we treat the case where plane waves in the same direction are scaled diversely. Differently scaled shifts of a plane wave are not necessarily linearly independent. An example is the case of plane waves created from a ramp function $r(t) = \max(0, \min(t, 1))$:

$$2r\left(\frac{t}{2}\right) - r(t) - r(t-1) = 0. \tag{2.50}$$

However, in the general case where plane waves can be both scaled and shifted, a general condition ensuring that the constructed plane waves are linearly independent has not yet been established. Nevertheless, this has been investigated for a specific class of functions. Here, we briefly present such cases, classifying activation functions into three groups: (1) those which converge (or diverge) more rapidly than exponential functions, (2) those which converge (or diverge) in exponential order, and (3) those which converge (or diverge) in polynomial order.

The theorem and remark below, demonstrate linear independence of normal, cumulated normal, arctangent, symmetric exponential, hyperbolic tangent, and logistic sigmoid activation functions, and so on (see [204]). A plane wave and its reversion created from one of these (and a constant) are not linearly independent because they are symmetric ($g(-t) = g(t)$ or $g(-t) = -g(t)$). Ignoring such trivial linear dependence, we intend to prove that if $(w_i, t_i) \neq (w_j, t_j)$ for $i \neq j$, then $g(w_i x - t_i)$'s and linearly independent polynomials are linearly independent for the respective activation functions.

However, by Corollary 2.5.1, they are linearly independent if $\mathcal{F}g$ is a function. Hence, it is sufficient to prove that the $g(c_i t - t_j)$'s are linearly independent if $(c_i, t_i) \neq (c_j, t_j)$ for $i \neq j$.

Note that we may ignore the case $c_i = 0$ because $g(c_i t - t_j)$ is a constant.

Let us denote by $a_{ij}$ the constants for which holds:

$$\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij} g(c_i t - t_j) = 0. \tag{2.51}$$

Our purpose is, of course, to prove that $a_{ij} = 0$ for all $i$ and $j$. Let us define $g_{ij} = g(c_i t - t_j)$ and $h_{ij} = h(c_i t - t_j)$.

**Remark 2.5.6.** When the activation function is differentiable, Eq. (2.51) implies that

$$\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij} g'(c_i t - t_j) = 0. \tag{2.52}$$

Here, recall that $c_i \neq 0$ for all $i$. If Eq. (2.52) implies that $a_{ij} = 0$ for all $i$ and $j$, it also implies $a_0 = 0$ for all $i$ and $j$. Hence, linear independence of $g'(c_i t - t_j)$'s implies that $g(c_i t - t_j)$'s are linearly independent.

Let $g_k, k = 1, \ldots, n$, be nonzero functions on $\mathbb{R}$. Suppose that, for any $k < n$,

$$\left|\frac{g_{k+1}(t)}{g_k(t)}\right| \text{ converges to 0 as } t \to \infty \, (t \to -c_k)\,. \tag{2.53}$$

Then the $g_k$ are linearly independent. In fact, let $\sum_{i=1}^{n} a_k g_k(t) = 0$. Then,

$$\lim_{t\to\infty} \frac{a_k g_k(t)}{g_1(t)} = a_1 = 0. \tag{2.54}$$

Repeating this, we obtain that $a_1 = a_2 = \cdots = a_n = 0$. The condition on $g_k$ and $g_{k+1}$ in this remark defines an order. If a set of functions can be ordered in this sense, the functions are linearly independent.

Examples of functions that can be proven independent, by the above arguments, are the Gaussian RBF, $g(t) = \exp(-t^2/2)$, and its cumulative integral, the hyperbolic tangent and the logistic sigmoid.

### 2.5.2 Utilizing RPNNs for Function Approximation

When approximating a sufficiently smooth function $f : \Omega \subseteq \mathbb{R}^d \to \mathbb{R}$, for which we can evaluate $n+1$ points of its graph $(\boldsymbol{x}_0, y_0), (\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)$ such that $y_i = f(\boldsymbol{x}_i)$, the training of an RPNN reduce to the solution of a linear interpolation system of $n+1$ algebraic equation with $N+1$ unknowns $\tilde{\boldsymbol{w}} = (\beta^o, \boldsymbol{w})$:

$$R\tilde{\boldsymbol{w}} = \tilde{R}\boldsymbol{w} + \beta^o = \boldsymbol{y}, \qquad R_{ij} = \psi_j(\boldsymbol{x}_i) \tag{2.55}$$

where $\boldsymbol{y} = (y_0, y_1, \ldots, y_n)$ is the vector containing the desired outputs, and the random collocation matrix $R$ has elements $R_{ij}$.

Let us state the following theorem here:

**Theorem 2.5.4** (Exact interpolation of RPNNs)**.** Given an RPNN with $N$ hidden neurons as in Eq. (2.40) with an infinitely differentiable, non-polynomial and slowly-increasing activation function $\psi : \mathbb{R} \to \mathbb{R}$, such that the support of its Fourier transform is an open subset. Let internal weights $A$ and biases $\boldsymbol{\beta}$ be randomly chosen according to any continuous probability distribution. Then, for $N+1$ arbitrary input-output samples $(\boldsymbol{x}_i, y_i)$, with distinct inputs $\boldsymbol{x}_i \neq \boldsymbol{x}_{i'}$, $\forall i \neq i'$, there exist a unique choice of external weights $\boldsymbol{w}$ and offset $\beta^o$ that ensure with probability 1:

$$\mathbb{P}(|f_N(\boldsymbol{x}_i; \boldsymbol{w}, \beta^o, A, \boldsymbol{\beta}) - y_i| = 0) = 1, \qquad \forall i = 1, \ldots, n+1 \tag{2.56}$$

*Proof.* We will prove that, the matrix $R$ in Eq. (2.55) is invertible with probability 1. Consider the $j$-th column vector

$$\mathbf{v}(\beta_j) = [\psi_j(\boldsymbol{x}_0), \ldots, \psi_j(\boldsymbol{x}_N)]^T = [\psi(\boldsymbol{\alpha}_j \boldsymbol{x}_1 + \beta_j), \ldots, \psi(\boldsymbol{\alpha}_j \boldsymbol{x}_N + \beta_j)]^T \tag{2.57}$$

of a matrix $R$ in Euclidean space $\mathbb{R}^N$, where $\beta_j \in (\beta_{min}, \beta_{max}) \subset \mathbb{R}$ is the corresponding internal bias. We want to establish that the vector $\mathbf{v}$ does not belong to any subspace with dimension less than $N+1$. Since $\boldsymbol{x}_k \neq \boldsymbol{x}_{k'}$ are distinct, and since the weights $A$ are drawn from a continuous probability distribution, we can assume with probability 1 that also $\boldsymbol{\alpha}_j \boldsymbol{x}_k \neq \boldsymbol{\alpha}_j \boldsymbol{x}_{k'}$, for all $k \neq k'$. Also, we get with probability 1 that $(\boldsymbol{\alpha}_j, \beta_j) \neq \pm(\boldsymbol{\alpha}_{j'}, \beta_{j'})$, thus satisfying the condition of Ito [204] to obtain a system of independent (plane wave) basis functions. If by contradiction, we assume that

$\mathbf{v}$ belongs to a subspace of dimension $N$ (less than $N+1$). Then, there exists a vector $\mathbf{u} = (u_0, u_1, \ldots, u_N) \neq \mathbf{0}$ which is orthogonal to this subspace and such that:

$$\mathbf{u}\cdot(\mathbf{v}(\beta_j)-\mathbf{v}(\beta_{min})) = u_0\psi(\beta_j+z_0)+\ldots+u_N\psi(\beta_j+z_N)-\mathbf{u}\cdot\mathbf{v}(\beta_{min}) = 0, \quad (2.58)$$

where $z_k = \boldsymbol{\alpha}_j \boldsymbol{x}_k$ for $k = 0, \ldots, N$, for all $\beta_j \in (\beta_{min}, \beta_{max})$. Assuming, without loss of generality, $u_N \neq 0$, the above equation can be further expressed as:

$$\psi(\beta_j + z_N) = \sum_{l=0}^{N-1} \gamma_l \psi(\beta_j + z_l) + \frac{\mathbf{u}\cdot\mathbf{v}(\beta_{min})}{u_N}, \quad (2.59)$$

where $\gamma_l = \frac{u_l}{u_N}$ for $l = 0, \ldots, N-1$. Utilizing the fact that $\psi$ is infinitely differentiable and a non-polynomial, we have:

$$\psi^{(m)}(\beta_j + z_N) = \sum_{l=0}^{N-1} \gamma_l \psi^{(m)}(\beta_j + z_l), \qquad m = 1, 2, \ldots, N, N+1, \ldots \quad (2.60)$$

However, there are only $N$ free coefficients $\gamma_0, \gamma_1, \ldots, \gamma_{N-1}$, for the resulting (more than) $N+1$ linear equations in Eq. (2.60). Indeed, we can note that the matrix $W$ with entries $W_{m,l} = \psi^{(m)}(\beta_j + z_l)$ is a Wronskian matrix. Since we have a basis of linear independent functions with probability 1, the Wronskian matrix of size $N+1 \times N+1$ is invertible. But Eq. (2.60) implies that only $N$ columns are independent. This contradiction implies that the vector $\mathbf{v}$ does not belong to any subspace with a dimension less than $N+1$. □

However, even if the matrix $R$ in Eq. (2.55) is invertible under the condition of Theorem 2.5.4, for many random choices of internal weights $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and collocation points $\boldsymbol{x}$, in practice the random collocation matrix $R$ numerically tends to be ill-conditioned and close to singular. Therefore, in practice, it is often suggested to solve Eq. (2.55) via a Moore-Penrose pseudo-inverse, that can be computed by a truncated SVD (tSVD). This involves computing the SVD and the regularized pseudo-inverse $R^\dagger$, as follows:

$$R = U\Sigma V^T = [U_q \quad \tilde{U}] \begin{bmatrix} \Sigma_q & 0 \\ 0 & \tilde{\Sigma} \end{bmatrix} [V_q \quad \tilde{V}]^T, \qquad R^\dagger = V_q \Sigma_q^{-1} U_q^T, \quad (2.61)$$

where the matrices $U = [U_q \quad \tilde{U}] \in \mathbb{R}^{n+1\times n+1}$ and $V = [Vq \quad \tilde{V}] \in \mathbb{R}^{N+1\times n+1}$ are orthogonal and $\Sigma \in \mathbb{R}^{n+1\times N+1}$ is a diagonal matrix containing the singular values $\sigma_i = \Sigma_{(i,i)}$. Here, we select the $q$ largest singular values exceeding a specified tolerance $0 < \epsilon \ll 1$, i.e., $\sigma_1, \ldots, \sigma_q > \epsilon$, effectively filtering out insignificant contributions and improving numerical stability. Note that the solution of Eq. (2.55) with the Moore-Penrose pseudo-inverse correspond to the solution of the following regularized least-squares problem in $L^2$:

$$\arg\min_{\tilde{\boldsymbol{w}}\in\mathbb{R}^{N+1}} \|R\tilde{\boldsymbol{w}} - \boldsymbol{y}\|_2^2 + \epsilon\|\tilde{\boldsymbol{w}}\|_2^2. \quad (2.62)$$

On the other hand, the Tikhonov regularization approach offers an alternative way to deal with the ill-conditioning of $R$ by minimizing a similar regularized least-squares problem

$$\boldsymbol{w}_k = \arg\min \boldsymbol{w}_k \left\{ \|\boldsymbol{w}_k \Psi - Y_k\|_2^2 + \lambda \|\boldsymbol{w}_k L\|_2^2 \right\}, \tag{2.63}$$

where $Y_k$ is the vector of dimension $m$, containing the values (samples) of $f_k$ at $m$ sampling points $\boldsymbol{x}_i$, $\lambda > 0$ is the regularization parameter, and $L \in \mathbb{R}^{N \times N}$ is a regularization operator, often taken as the identity matrix $I$. The Tikhonov regularized solution can be expressed as:

$$\boldsymbol{w}_k = Y_k \Psi^T (\Psi \Psi^T + \lambda L L^T)^{-1}. \tag{2.64}$$

By setting $L = I$ and substituting the tSVD of $\Psi$, this solution can be further simplified as follows [205, 157]:

$$\boldsymbol{w}_k = \sum_{i=1}^{r} \frac{{\sigma_i}^2}{{\sigma_i}^2 + \lambda^2} \frac{{u_i}^T Y_k}{\sigma_i} v_i, \tag{2.65}$$

where $r$ is the rank of $\Psi$, $\sigma_i$ are the singular values of $\Psi$, and $u_i$, $v_i$ are the corresponding left and right singular vectors. This formulation highlights the impact of the regularization parameter $\lambda$ on the filtered contributions from each singular component.

Alternatively, a more robust approach that further enhances numerical stability, involves utilizing a rank-revealing QR decomposition with column-pivoting:

$$R\,P = [Q_1 \quad Q_2] \begin{bmatrix} T \\ 0 \end{bmatrix}, \tag{2.66}$$

where the matrix $Q = [Q_1 \quad Q_2] \in \mathbb{R}^{n+1 \times n+1}$ is orthogonal, $T \in \mathbb{R}^{N+1 \times N+1}$ is an upper triangular square matrix and the matrix $P \in \mathbb{R}^{n+1 \times n+1}$ is an orthogonal permutation of the columns. The key advantage of the column permutations lies in its ability to automatically identify and discard small values that contribute to instability. Indeed, in case of ill-conditioned (rank-deficient) system we have that effectively the column of the matrix $Q$ do not span the same space as the column of the matrix $R$. As a result, the matrix $T$ is not full upper triangular, but we have:

$$R = [Q_1 \quad Q_2] \begin{bmatrix} T_{11} & T_{12} \\ 0 & 0 \end{bmatrix} P^T, \tag{2.67}$$

where, if $rank(R) = r < N + 1$, the matrix $T_{11} \in \mathbb{R}^{r \times r}$ is effectively upper triangular and $T_{12} \in \mathbb{R}^{r \times (N+1-r)}$ are the remaining columns. Note that numerically, one selects a tolerance $0 < \epsilon << 1$ to estimate the rank $r$ of the matrix $R$ and set values of $T$ below the threshold to zero. Hence, based on (2.67), we can solve $R\tilde{\boldsymbol{w}} = \boldsymbol{y}$, by first setting $P^T \tilde{\boldsymbol{w}} = [\boldsymbol{z}_1\, \boldsymbol{z}_2]^T$ and solve for $\boldsymbol{z}_1 \in \mathbb{R}^r$ the following system:

$$T_{11} \boldsymbol{z}_1 = Q_1^T \boldsymbol{y} - T_{12} \boldsymbol{z}_2. \tag{2.68}$$

Note that casting the back-substitution algorithm for the matrix $T_{11}$ in Eq. (2.68), we can efficiently solve the system. In order to also obtain the *minimum-norm* solution, one

can additionally cast the COD [161], by also computing the QR decomposition of the transposed non-zero elements in $T$:

$$\begin{bmatrix} T_{11}^T \\ T_{22}^T \end{bmatrix} = V \begin{bmatrix} \tilde{T}_{11}^T \\ 0 \end{bmatrix}. \tag{2.69}$$

Finally, by setting $S = PV$, one obtains:

$$R = [Q_1 \quad Q_2] \begin{bmatrix} \tilde{T}_{11} & 0 \\ 0 & 0 \end{bmatrix} (S)^T, \tag{2.70}$$

where $\tilde{T}_{11}$ is a lower triangular matrix of size $r \times r$. Finally, the least-square minimum norm solution, as in Eq. (2.62). is given by:

$$\tilde{\boldsymbol{w}} = S \cdot (\tilde{T}_{11}^{-1}(Q_1^T \boldsymbol{y})) \tag{2.71}$$

where the inversion of $\tilde{T}_{11}$ is computed casting the forward substitution algorithm that is numerically stable.

## 2.6 Best Approximation with RPNNs

Let us consider the case in which the input is one-dimensional $x \in [a, b] \subset \mathbb{R}$. The Eq. (2.40) can be restated as:

$$f_N(x; \boldsymbol{w}, \beta^o, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_{j=1}^{N} w_j \psi(\alpha_j \cdot x + \beta_j) + \beta^o = \sum_{j=1}^{N} w_j \psi_j(x) + \beta^o, \tag{2.72}$$

where now we represent the internal weights by the column vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_N) \in \mathbb{R}^{N \times 1}$. Here, we define the concept of RPNN as the best approximation and provide proof about its existence and uniqueness. Let us first trivially note that if we consider an RPNN with independent fixed basis function, we have a linear space:

**Theorem 2.6.1.** Let $\psi$ be a slowly increasing (or infinitely differentiable) activation function. The set $\mathcal{M}_{(N,\boldsymbol{\alpha},\boldsymbol{\beta})}^{[a,b]} = \{f_N \in C[a,b] : f_N(x; \boldsymbol{w}, \beta^o, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_{j=1}^{N} w_j \psi(\alpha_j x + \beta_j) + \beta^o$, with $(\alpha_j, \beta_j) \neq \pm(\alpha_{j'}, \beta_{j'})$ and $\boldsymbol{w} \in \mathbb{R}^N\}$ of RPNN is a vector space of dimension $N + 1$ on $\mathbb{R}$ with usual vector sum and scalar vector product.

*Proof.* For $(\alpha_j, \beta_j) \neq \pm(\alpha_i, \beta_i)$ the set of basis functions is independent for [204]. Then clearly $f_N(\cdot\,; \tilde{\boldsymbol{w}}) \equiv 0$, with $\tilde{\boldsymbol{w}} = (\beta^o, \boldsymbol{w}) = \boldsymbol{0}$, is in $\mathcal{M}_{(N,\boldsymbol{\alpha},\boldsymbol{\beta})}^{[a,b]}$ and since if we have $f_N(\cdot\,; \boldsymbol{w}_1, \beta_1^o)$, $f_N(\cdot\,; \boldsymbol{w}_2, \beta_2^o) \in \mathcal{M}_{(N,\boldsymbol{\alpha},\boldsymbol{\beta})}^{[a,b]}$, based on (2.72), also we obtain $f_N(\cdot\,; \gamma_1 \boldsymbol{w}_1 + \gamma_2 \boldsymbol{w}_2, \gamma_1 \beta_1^o + \gamma_2 \beta_2^o) \in \mathcal{M}_{(N,\boldsymbol{\alpha},\boldsymbol{\beta})}^{[a,b]}$, $\forall \gamma_1, \gamma_2$. Finally, trivially, there exist an isomorphism between the set of external weights $\tilde{\boldsymbol{w}}$ and the usual vector space on $\mathbb{R}^{N+1}$. □

Let's also observe that the differentiability of the function of $\mathcal{M}^{[a,b]}_{(N,\boldsymbol{\alpha},\boldsymbol{\beta})}$, depends on the differentiability of the activation functions. Thus:

**Lemma 2.6.1.** If the activation function $\psi \in C^{\nu}(\mathbb{R})$, then the space $\mathcal{M}^{[a,b]}_{(N,\boldsymbol{\alpha},\boldsymbol{\beta})}$ is a vector subspace of $C^{\nu}([a,b])$.

Now, we proceed by considering a norm, such as the $L^p$-norm with $p = 1, 2, \infty$, denoted also as $\|\cdot\|_\infty$, $\|\cdot\|_2$, or $\|\cdot\|_1$. Since $C[a,b]$ equipped with $L^p$ norms is a Banach space, it follows that $\mathcal{M}^{[a,b]}_{!(N,\boldsymbol{\alpha},\boldsymbol{\beta})}$ is also a Banach space.

**Definition 2.6.1.** We define the RPNN of the best approximation in norm $L^p$ over $\mathcal{M}_{\boldsymbol{\alpha},\boldsymbol{\beta}}$ as the RPNN which solve

$$\|f - f_N\|_p = \min_{g_N \in \mathcal{M}^{[a,b]}_{(N,\boldsymbol{\alpha},\boldsymbol{\beta})}} \|f - g_N\|_p. \tag{2.73}$$

### 2.6.1 Existence

Since, $\mathcal{M}^{[a,b]}_{(N,\boldsymbol{\alpha},\boldsymbol{\beta})} \subset C([a,b])$ is a finite dimensional Banach space, and the function $\|f - \cdot\|_p$ is continuous, for the Stone–Weierstrass approximation theorem, there exists an ANN $f_N$ as defined above of the best $L^p$ approximation in $\mathcal{M}^{[a,b]}_{(N,\boldsymbol{\alpha},\boldsymbol{\beta})}$. Indeed, the zero function, described by the vector of parameters $\mathbf{0}$, is in $\mathcal{M}^{[a,b]}_{(N,\boldsymbol{\alpha},\boldsymbol{\beta})}$, which approximate $f$ with error $\|f\|_p$. So the best approximation should be in the closed ball of function $g_N \in \mathcal{M}^{[a,b]}_{(N,\boldsymbol{\alpha},\boldsymbol{\beta})}$ such that $\|f - g_N\|_p \leq \|f\|_p$, which is closed and limited, so it is a compact set. Therefore, a global minimum in $\mathcal{M}^{[a,b]}_{(N,\boldsymbol{\alpha},\boldsymbol{\beta})}$ exists.

### 2.6.2 Uniqueness

We have shown that $\mathcal{M}^{[a,b]}_{(N,\boldsymbol{\alpha},\boldsymbol{\beta})}$ forms a vectors space, implying it is also a convex set. For illustration, consider two vectors of weights $\tilde{\boldsymbol{w}}_0 = (\beta^o_0, \boldsymbol{w}_0)$ and $\tilde{\boldsymbol{w}}_1 = (\beta^o_1, \boldsymbol{w}_1)$ and a point $\tilde{\boldsymbol{w}}_\gamma$ on the segment, i.e.:

$$\tilde{\boldsymbol{w}}_\gamma = \gamma\tilde{\boldsymbol{w}}_1 + (1-\gamma)\tilde{\boldsymbol{w}}_0, \qquad \gamma \in [0,1]. \tag{2.74}$$

The vector $\tilde{\boldsymbol{w}}_\gamma$ still represents the external weights of an RPNN $f_N(\,\cdot\,;\tilde{\boldsymbol{w}}_\gamma)$ in $\mathcal{M}^{[a,b]}_{(N,\boldsymbol{\alpha},\boldsymbol{\beta})}$. Hence, we can say that, there exists either a unique best $L^p$ approximation within the space $\mathcal{M}^{[a,b]}_{(N,\boldsymbol{\alpha},\boldsymbol{\beta})}$ or infinitely many equally good approximations, all corresponding to the points along the segment. Let's assume $\tilde{\boldsymbol{w}}_0$ and $\tilde{\boldsymbol{w}}_1$ are two solutions of best $L^p$ approximation of a function $f$, thus $\|f - f_N(\,\cdot\,;\tilde{\boldsymbol{w}}_0)\|_p = \|f - f_N(\,\cdot\,;\tilde{\boldsymbol{w}}_1)\|_p = r$. Then for $\tilde{\boldsymbol{w}}_\gamma$ as in Eq. (2.74):

$$\begin{aligned} &\|f - f_N(\tilde{\boldsymbol{w}}_\gamma)\|_p = \|\gamma(f - f_N(\,\cdot\,;\tilde{\boldsymbol{w}}_1)) + (1-\gamma)(f - f_N(\,\cdot\,;\tilde{\boldsymbol{w}}_0))\|_p \leq \\ &\leq \gamma\|f - f_N(\,\cdot\,;\tilde{\boldsymbol{w}}_1)\|_p + (1-\gamma)\|f - f_N(\,\cdot\,;\tilde{\boldsymbol{w}}_0)\|_p = \gamma r + (1-\gamma)r = r \end{aligned} \tag{2.75}$$

Hence, to maintain consistency with the best approximations defined by $\tilde{\boldsymbol{w}}_0$ and $\tilde{\boldsymbol{w}}_1$, we require necessarily the equality $\|f - f_N(\,\cdot\,;\tilde{\boldsymbol{w}}_\gamma)\|_p = r$ to hold $\forall\gamma \in [0,1]$. This implies that all $\tilde{\boldsymbol{w}}_\gamma$ define alternative RPNNs with the same maximum error as the original best approximations. If we consider $L^p$ norms, with $1 < p < \infty$, the space is also strictly convex[1], which imply that necessarily $\tilde{\boldsymbol{w}}_1 = \tilde{\boldsymbol{w}}_2$. Otherwise, if they are distinct then, by strict convexity:

$$\|f - f_N(\,\cdot\,;\tilde{\boldsymbol{w}}_0) + f - f_N(\,\cdot\,;\tilde{\boldsymbol{w}}_1)\|_p < 2r \tag{2.76}$$

or, equivalently,

$$\Big\|f - \frac{f_N(\,\cdot\,;\tilde{\boldsymbol{w}}_0) + f_N(\,\cdot\,;\tilde{\boldsymbol{w}}_1)}{2}\Big\|_p < r \tag{2.77}$$

thus we obtain a better approximation with $\tilde{\boldsymbol{w}}_\gamma = \tilde{\boldsymbol{w}}_{1/2}$

$$\|f - f_N(\,\cdot\,;\tilde{\boldsymbol{w}}_{1/2})\|_p < r, \tag{2.78}$$

that is a contradiction. Therefore, we can state the following theorem.

**Theorem 2.6.2.** The best $L^p$ approximation problem with RPNNs in Eq. (2.73) has a unique solution in the space $\mathcal{M}^{[a,b]}_{(N,\boldsymbol{\alpha},\boldsymbol{\beta})}$ when $1 < p < \infty$.

## 2.7 Convergence of the RPNN of the best approximation

In this section, we will prove that by considering activation functions that are infinitely differentiable, then the RPNNs of the best $L^p$ approximation can exhibit exponential convergence towards approximating an infinitely differentiable function. Let us assume we want to approximate the function $f : \mathbb{R} \to \mathbb{R}$. Then we state the following theorem:

**Theorem 2.7.1.** The RPNN of the best $L^p$ approximation in $\mathcal{M}^{[a,b]}_{(N,\boldsymbol{\alpha},\boldsymbol{\beta})}$ equipped with an infinitely differentiable non-polynomial activation function converge exponentially fast when approximating an infinitely differentiable function $f : \mathbb{R} \to \mathbb{R}$.

Our goal is to demonstrate the existence of a choice of external weights and biases that exponentially enhance the convergence rate of an RPNN approximation. Subsequently, the unique RPNN configured for the best approximation is guaranteed to converge at least as rapidly. Specifically, we establish the following Lemma:

**Lemma 2.7.1.** There exist a choice of weights $\tilde{\boldsymbol{w}} = (\beta^o, \boldsymbol{w})$ that makes an RPNN in $\mathcal{M}^{[a,b]}_{(N,\alpha,\beta)}$ with infinitely differentiable non-polynomial activation functions, converging exponentially fast in $L^p$ when approximating an infinitely differentiable function $f : \mathbb{R} \to \mathbb{R}$.

*Proof.* Let's call $\mathcal{P}_n$ the polynomial of best $L^p$ approximation of degree $n$ of $f$ in the interval $[-1,1]$ w.r.t. the norm $\|\cdot\|_p$ with $1 < p < \infty$:

$$f(x) \simeq \mathcal{P}_n(x) = \sum_{k=0}^{n} a_k x^k. \tag{2.79}$$

[1] A normed linear space $(X, \|\cdot\|)$ is called strictly convex if for $u, v \in X$, $\|u\| \leq r\|$ and $\|v\| \leq r$, $\|u+v\| < 2r$ unless $u = v$.

Let's call $f_N(\,\cdot\,;\tilde{\boldsymbol{w}}) \in \mathcal{M}^{[a,b]}_{(N,\alpha,\beta)}$ a RPNN approximation of $f$ that we are seeking.

$$f(x) \simeq f_N(x;\tilde{\boldsymbol{w}}) = \sum_{j=1}^{n} w_j \psi(\alpha_j \cdot x + \beta_j) + \beta^o, \tag{2.80}$$

where $\psi$ is the activation function, which is a non-polynomial and infinitely differentiable. Let us also define $c_j$ the *centers* of the wave plane activation function such that $c_j = -\frac{\beta_j}{\alpha_j}$ (see, e.g., [69]), or equivalently, $\beta_j = -\alpha_j \cdot c_j$:

$$f(x) \simeq f_N(x;\tilde{\boldsymbol{w}}) = \sum_{j=1}^{N} w_j \psi(\alpha_j \cdot (x - c_j)) + \beta^o. \tag{2.81}$$

Let us consider also the best $L^p$ approximation polynomial $\mathcal{Q}_n$ of degree $n$ of the activation function $\psi$, in the interval $I_{\alpha,\beta} = [\min(\alpha_j \cdot x + \beta_j), \max(\alpha_j \cdot x + \beta_j)]$:

$$\psi(x) \simeq \mathcal{Q}_n(x) = \sum_{k=0}^{n} b_k x^k. \tag{2.82}$$

Note that since $\psi$ is a *non-polynomial* function, it cannot be *exactly* (with zero residual) represented by any polynomial $\mathcal{Q}_n$ of finite degree. Therefore, in what follows we can consider arbitrary large $n$, and a corresponding *non-zero* coefficient $b_n$. Also note that the interval $I_{\alpha,\beta}$ is considered, to maintain accurate approximation of $\psi_j$ by using the transformed polynomial $\mathcal{Q}_n(\alpha_j x + \beta_j)$.

We can then approximate the neural network $f_N$ with the polynomial $\mathcal{G}_{N,n}$, given by:

$$f_N(x;\boldsymbol{w}) \simeq \mathcal{G}_{N,n}(x) = \sum_{j=1}^{N} w_j \sum_{k=0}^{n} b_k \alpha_j^k (x - c_j)^k + \beta^o. \tag{2.83}$$

We can rewrite the polynomial $\mathcal{G}_{N,n}$ using Newton's binomial expansion:

$$\mathcal{G}_{N,n}(x) = \sum_{j=1}^{N} w_j \sum_{k=0}^{n} b_k \alpha_j^k \sum_{s=0}^{k} \binom{k}{s} (-c_j)^{n-s} x^s + \beta^o. \tag{2.84}$$

At this point, we want to prove that we can find a vector of coefficient $\tilde{\boldsymbol{w}}$ such that we can have $\mathcal{G}_{N,n} = \mathcal{P}_n$, thus we have to equate the coefficients of order $k$ in Eq. (2.84) and (2.79), thus resulting in a system of $n$ equations in $N$ unknowns:

$$a_k = \sum_{j=1}^{N} w_j (-c_j)^{n-k} \sum_{s=k}^{n} \binom{s}{k} b_s \alpha_j^s, \qquad k = 1, \ldots, n, \tag{2.85}$$

while trivially equate the two offsets $\beta^o = a_0$. Eq. (2.85) can be written in matrix form as

$$\boldsymbol{a} = \boldsymbol{w} \cdot M, \tag{2.86}$$

where $\boldsymbol{a}$ is the vector containing the coefficient $a_k$ and $M$ is the matrix with elements $M_{k,j}$ that reads:

$$M_{k,j} = (-c_j)^{n-k} \sum_{s=k}^{n} \binom{s}{k} b_s \alpha_j^s. \tag{2.87}$$

If we take $N = n$ and we prove that the matrix $M$ in Eq. (2.87) is invertible, with inverse $M^{-1}$, then there exist a unique choice of weights $\boldsymbol{w} = \boldsymbol{a} \cdot M^{-1}$ that make $\mathcal{G}_{N,N} = \mathcal{P}_N$. Note that, as proved in [204], the slowly increasing activation functions are independent if $(\alpha_j, \beta_j) \neq \pm(\alpha_{j'}, \beta_{j'})$. Thus, we may allow both $\alpha_j$ and $c_j$ to be different for $j \neq j'$. Let us consider a fixed index $k = 1, \dots, N$ in Eq. (2.87), then we have a polynomial in the two variables $\alpha, c$ of type:

$$M_{k,j} = \tau_N c_j^{N-k} \alpha_j^k + \tau_{N-1} c_j^{N-k} \alpha_j^{k+1} \dots + \tau_{N-k} c_j^{N-k} \alpha_j^N, \tag{2.88}$$

where the $\tau$s are the coefficients in Eq. (2.87). Different index $k' \neq k$ corresponds to polynomials of different order, which, as can be noted, do not have any monomials in common. Indeed $c_j^{N-k} \neq c_j^{N-k'}$, thus they are a set of independent polynomials. It follows that the rows of $M$ are independent and therefore, $M$ in Eq. (2.87) is invertible.

Finally, when approximating $f$ with an RPNN $f_N(\,\cdot\,; \tilde{\boldsymbol{w}})$, where the weights $\tilde{\boldsymbol{w}}$ make $f_N$ coincide with the polynomial $\mathcal{G}_{N,N}$ constructed as above, we can prove that $f_N(\,\cdot\,; \tilde{\boldsymbol{w}})$ exponentially convergence to $f$, since:

$$\begin{aligned} &\|f - f_N\|_p = \|f - \mathcal{P}_N + \mathcal{P}_N - \mathcal{G}_{N,N} + \mathcal{G}_{N,N} - f_N\|_p \leq \\ &\leq \|f - \mathcal{P}_N\|_p + \|\mathcal{P}_N - \mathcal{G}_{N,N}\|_p + \|\mathcal{G}_{N,N} - f_N\|_p. \end{aligned} \tag{2.89}$$

But since $\mathcal{P}_N$ is the best approximation polynomial, then $\|f - \mathcal{P}_N\|_p$ converges exponentially, as well as $\|\mathcal{G}_{N,N} - f_N\|_p$ converges exponentially because we have used the best approximation polynomial $\mathcal{Q}_N$ of the activation function $\psi$. Finally $\|\mathcal{P}_N - \mathcal{G}_{N,N}\|_p = 0$, because of the invertibility of the matrix $M$ in (2.87). □

Please note that the above constructive proof has shown that one has to invert a matrix $M$ in Eq. (2.87) which shares some similarities with the Vandermonde matrix, especially as $N$ becomes large or as the degrees of the polynomials increase, therefore it may be an ill-conditioned matrix.

In order to evaluate the upper bound of the convergence, as we have determined in Eq. (2.89), let us restate a classical result about the upper bounds of convergence of Legendre polynomials (see Chapter 2 in Gaier (1987) [206]).

**Theorem 2.7.2.** Let a function $f$ analytic in $[-1, 1]$, that is analytically continuable to the open Bernstein Ellipse $\mathcal{E}_\rho$, with $1 \leq \rho \leq \infty$, in the complex plane, given by:

$$\mathcal{E}_\rho := \left\{ z \in \mathbb{C} \,\middle|\, z = \frac{u + u^{-1}}{2}, u = \rho e^{i\theta}, 0 \leq \theta \leq 2\pi \right\} \tag{2.90}$$

and let $\mathcal{P}_N$ be the Legendre interpolants to $f$ in Legendre grid of $N + 1$ points $x_N \in [-1, 1]$. Then the error satisfies:

$$\lim_{N \to \infty} \|f - \mathcal{P}_N\|_p = O(\rho^{-N}). \tag{2.91}$$

A polynomial interpolant satisfying Eq. (2.91) is said to be *maximally convergent*. Other examples of such polynomials are interpolants in Chebyshev, or Gauss-Jacobi grids. The effective convergence rates of these polynomials may differ because of some algebraic factors. In particular. the value of $\rho$ depends on the smoothness of the function $f$ and on the domain interval $[a, b]$, that need to be rescaled to $[-1, 1]$, by considering the Bernstein Ellipse $\mathcal{E}_\rho$ associated with $\tilde{f}(x) = f\left(\frac{2x-a-b}{b-a}\right) : [-1, 1] \to \mathbb{R}$.

Now, we can note that in Eq. (2.89), $\|f - \mathcal{P}_N\|_p$ in the interval $[a, b]$ converges as $O(\rho(f, [a, b])^{-N})$, where $\rho(f, [a, b]) > 1$ is the constant associate to the maximum Bernstein ellipse $\mathcal{E}_\rho$ in which $\tilde{f}$ is analytically continuable. While the convergence of $\|\mathcal{G}_{N,N} - f_N\|_p$ is given by $N$ times the convergence of $\|\mathcal{Q}_N - \psi\|$, thus $\|\mathcal{G}_{N,N} - f_N\|_p$ is of order $O(N \cdot \rho(\psi, I_{\alpha,\beta})^{-N})$.

Finally, we emphasize that since the Theorem 2.7.1 hinges on the ability of RPNNs to accurately reproduce polynomials, as described in the upper bound in Eq. (2.89) when approximating less smooth functions, with a maximum order of differentiability $\nu \geq 0$, this sets a limit to the achievable convergence rate, reducing it to $O(N^{-\nu})$.

## 2.8 On the selection of weights, biases, and hyperparameters

In this section, we look into the *a priori* selection of the values of the internal weights and biases for the RPNN approximator. Despite the theoretical assurance that any random parameter selection should lead to exponential convergence, as proven in Theorem 2.7.1, practical considerations necessitate a judicious choice to mitigate the ill-conditioning of the matrix $R$ in Eq. (2.55).

Moving beyond a *naive random generation* [198] of internal parameter, we therefore explore three alternative strategies for selection, namely:

- *Parsimonious Function-Agnostic Selection:* Parameters dependent solely on the domain of data of the sought function.
- *Function-Informed Selection:* Addressing the specific shape of the function in consideration.
- *Geometric random sampling:* Addressing the case of high-dimensional inputs.

Note that other alternative approaches have also been investigated by other authors (see, e.g., [94, 181]).

### 2.8.1 Naive random generation

In a naive random generation approach for internal parameters and biases of an RPNN [198], a common practice involves the normalization of input $\boldsymbol{x} \in [a, b]$ and output data $\boldsymbol{y} \in [c, d]$ within the domain $[-1, 1]$. Thus, employing these transformations $x \to \tilde{x}$ and $y \to \tilde{y}$:

$$\tilde{x} = \frac{2x - a - b}{b - a} \in [-1, 1], \qquad \tilde{y} = \frac{2y - c - d}{d - c} \in [-1, 1]. \tag{2.92}$$

Subsequently, weights $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_N)$ are uniformly and randomly selected from the range $[-1, 1]$, and biases $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_N)$ are similarly chosen within the domain $[-1, 1]$. Despite its simplicity, this method has proven effective in certain scenarios, making it a straightforward yet powerful approach for initializing RPNNs [198]. However, as numerical results will demonstrate, this choice deviates considerably from optimal configurations.

### 2.8.2 Parsimonious function-agnostic selection

As proposed in previous works [95, 69], a fundamental strategy for obtaining effective basis functions involves ensuring that the activation functions $\psi_j$ have centers $c_j = -\beta_j/\alpha_j$ (as defined in Eq. (2.81)) within the domain $[a, b]$ of interest. This choice depends on both the activation function and the specific problem at hand. The ideal bounds of the distribution can vary slightly, whether it's a simple regression task, a problem involving spatial derivatives, or a time-evolving phenomenon with potentially hidden and challenging-to-determine stability conditions.

Here, for simplicity, we focus on common sigmoid-like functions, such as the logistic sigmoid, given by:

$$\psi_j(x) \equiv \sigma_j(x) = \frac{1}{1 + \exp(-\boldsymbol{\alpha_j} \cdot \boldsymbol{x} - \beta_j)}. \tag{2.93}$$

For this function, it is straightforward to compute the derivatives. In particular, the derivatives with respect to the $k$-th component of $\boldsymbol{x}$, $x_k$ are given:

$$\begin{aligned} \frac{\partial}{\partial x_k}\sigma_j(\boldsymbol{x}) &= \alpha_{j,k}\frac{\exp(z_j)}{(1 + \exp(z_j))^2}, \\ \frac{\partial^2}{\partial x_k^2}\sigma_j(\boldsymbol{x}) &= \alpha_{j,k}^2\frac{\exp(z_j) \cdot (\exp(z_j) - 1)}{(1 + \exp(z_j))^3}, \end{aligned} \tag{2.94}$$

where $z_j = \boldsymbol{\alpha}_j \cdot \boldsymbol{x} + \beta_j$.

A crucial point in the RPNN framework is how to fix the values of the internal weights and biases in a proper way. Indeed, despite the fact that theoretically any random choice should be good enough, in practice, it is convenient to define an appropriate range of values for the parameters $\alpha_{j,k}$ and $\beta_j$ that are strictly related to the selected activation

function. For the one-dimensional case ($d = 1$), the logistic sigmoid $\sigma_j$ is a monotonic function such that:

$$\begin{aligned} \alpha_{j,1} > 0 \Rightarrow \quad & \lim_{x\to+\infty} \sigma_j(x) = 1, \qquad \lim_{x\to-\infty} \sigma_j(x) = 0 \\ \alpha_{j,1} < 0 \Rightarrow \quad & \lim_{x\to+\infty} \sigma_j(x) = 0, \qquad \lim_{x\to-\infty} \sigma_j(x) = 1. \end{aligned}$$

This function has an inflection point, that we call *center* $c_j$ defined by the following property:

$$\sigma_j(\alpha_{j,1}c_j + \beta_j) = \frac{1}{2}. \tag{2.95}$$

Now since $\sigma(0) = 1/2$, the following relation between parameters holds:

$$c_j = -\frac{\beta_j}{\alpha_{j,1}}.$$

Finally, $\sigma_j$ has a steep transition that is governed by the amplitude of $\alpha_{j,1}$: if $|\alpha_{j,1}| \to +\infty$, then $\sigma_j$ approximates the Heaviside function, while if $|\alpha_{j,1}| \to 0$, then $\sigma_j$ becomes a constant function. Now, since in the RPNN framework, these parameters are fixed a priori, what one needs to avoid is to have some function that can be "useless" in the domain, say $I = [a, b]$.
Therefore, for the one-dimensional case, our suggestion is to chose $\alpha_{j,1}$ uniformly distributed as:

$$\alpha_{j,1} \sim \mathcal{U}\left(-\frac{N-55}{10|I|}, \frac{N+35}{10|I|}\right), \tag{2.96}$$

where $N$ is the number of neurons in the hidden layer and $|I| = b - a$ is the domain length. Moreover, we also suggest avoiding too small in module coefficients $a_j$ by setting:

$$|\alpha_{j,1}| > \frac{1}{2|I|}.$$

Then, for the centers $c_j$, we select equally-spaced points in the domain $I$, that are given by imposing the $\beta_j$s to be:

$$\beta_j = -\alpha_{j,1} \cdot c_j.$$

In the two-dimensional case ($d = 2$), we denote as $\boldsymbol{x} = (x, y)^T \in \mathbb{R}^2$ the input and $A \in \mathbb{R}^{N\times 2}$ the matrix with rows $\boldsymbol{\alpha}_j = (\alpha_{j,1}, \alpha_{j,2})$. Then, the condition (2.95) becomes:

$$\sigma_j(x, y) = \sigma(\alpha_{j,1}x + \alpha_{j,2}y + \beta_j) = \frac{1}{2}.$$

So, now we have:

$$s \equiv y = -\frac{\alpha_{j,1}}{\alpha_{j,2}}x - \frac{\beta_j}{\alpha_{j,2}},$$

where $s$ is a straight line of inflection points that we call *central direction*. As the direction parallel to the central direction $\sigma_j$ is constant, while the orthogonal direction to $s$, the sigmoid $\sigma_j$ is exactly the one-dimensional logistic sigmoid. So considering

one point $\boldsymbol{c}_j = (c_{j,1}, c_{j,2})$ of the straight line $s$, we get the following relation between parameters:

$$\beta_j = -\alpha_{j,1} \cdot c_{j,1} - \alpha_{j,2} \cdot c_{j,2}.$$

Note that, the bias parameter $\beta_j$ influences the position of the center $c_j$ of the activation function. One can either decide these centers beforehand (e.g., equally spaced) or randomly sample them within the domain of interest and subsequently determine the corresponding bias values using the following relation $\beta_j = -\frac{c_j}{\alpha_j}$.

Note that the above bounds have been optimized heuristically for application on the solution of stationary nonlinear PDEs. In other works, such as [156], where we investigate simple regression task in one dimension, we choose $\boldsymbol{\alpha}$ to be a vector of i.i.d random uniformly distributed values in a range that depends on the number of neurons $N$; if the domain of interest is $[a, b]$, we propose to set:

$$\alpha_j \sim \mathcal{U}\left[-\frac{(400+9N)}{10(b-a)}, \frac{(400+9N)}{10(b-a)}\right], \qquad j = 1, \ldots, N \tag{2.97}$$

where $\mathcal{U}$ denotes the uniform distribution.

Now, the difference with the one-dimensional case is the fact that in a domain $I^2 = [a, b]^2$ discretized by a grid of $n \times n$ points, the number of neurons $N = n^2$ grows quadratically, while the distance between two adjacent points decreases linearly, i.e., is given by $|I|/(n-1)$. Thus, for the two-dimensional case, we take $\alpha_{j,k}$ uniformly distributed as:

$$\alpha_{j,k} \sim \mathcal{U}\left(-\frac{\sqrt{N}-60}{20|I|}, \frac{\sqrt{N}+40}{20|I|}\right), \qquad k = 1, 2 \tag{2.98}$$

where $N$ is the number of neuron in the network and $|I| = b - a$. Such above selection will be used later in 2d Nonlinear PDE solution.

It is worth to note that this function-agnostic approach is especially effective for physics-informed problems, i.e., the solution of nonlinear PDEs, where the desired output is not directly available. Indeed, the solution needs to be derived solely from physical laws. This approach offers greater flexibility and adaptability, making it well-suited for diverse applications, such as resolving sharp gradients in the time evolution of PDEs or computing bifurcation diagrams [69, 95]. This is also particularly beneficial when varying underlying parameters, which may result in abrupt changes in the solution [69, 95].

### 2.8.3 Function-informed selection

In contrast to the just described *function-agnostic* selection, here we propose a *function-informed* selection for internal parameters and biases of an RPNN, that involves a more systematic approach. Here, we employ equally spaced centers in the input domain and leverage a centered FD scheme to efficiently compute the first derivative of the input function. Utilizing this derivative information, we try to set accordingly the $\alpha_j$ parameters for each neuron. This method ensures a more tailored initialization based on

the specific characteristics of the input function. Here below, an example for the specific case of a logistic sigmoid in Eq. (2.93), which have analytical first derivative:

$$\frac{d\psi_j(x)}{dx} = \frac{\alpha_j \exp(\alpha_j x + \beta_j)}{(1+\exp(\alpha_j x + \beta_j)^2} = \frac{\alpha_j \exp\left(\alpha_j (x - c_j)\right)}{\left(1+\exp(\alpha_j (x - c_j)\right)^2}, \tag{2.99}$$

where $c_j$ are the centers defined as $c_j = -\alpha_j \beta_j$. First (i) as for the vanilla generation, we normalize the input and the output data as in Eq. (2.92). Hence, we consider the rescaled function $\tilde{f}$. Then we set the centers $\tilde{c}_j$ equally spaced in $[-1, 1]$. Then, (ii) we compute the approximated derivatives of $\tilde{f}'$ of the function $\tilde{f}$ at the centers $\tilde{c}_j$ with a centered FD scheme. By observing that the derivative of the logistic sigmoid in Eq. (2.99) evaluated at the center $\tilde{c}_j = -\beta_j/\alpha_j$ is:

$$\frac{d\psi_j(c_j)}{dx} = \frac{\alpha_j}{4}, \tag{2.100}$$

Finally, (iii) we locally assume that:

$$\frac{d\psi_j(c_j)}{dx} = \tilde{f}'(c_j). \tag{2.101}$$

The above assumption is not generally correct as the contribution for the derivative of the RPNN $f_N(\cdot, \boldsymbol{w})$ is not only given by the single activation function $\psi_j$, but is also influenced by the corresponding weight $w_j$ and possibly by all the other activations functions. However, in the case of a sharp gradient, a sharp sigmoid is suitable and the main variation (the highest derivative value) is localized at $c_j$, while far from $c_j$ the sigmoid becomes rather constant. Therefore, we set ultimately and heuristically:

$$\alpha_j = \gamma \frac{f(c_{j+1}) - f(c_{j-1})}{\Delta x} + \varepsilon_j, \tag{2.102}$$

where $\Delta x$ is the distance between two consecutive centers, the proportionality constant is set $\gamma = 3/2$ and $\varepsilon_j$ is a random variable uniformly distributed, accounting for the inexactness of the assumption in (2.101), as:

$$\varepsilon_j \sim \mathcal{U}\left[-\frac{(400+9N)}{100(b-a)}), \frac{(400+9N)}{100(b-a)}\right], \qquad j = 1, \ldots, N. \tag{2.103}$$

This function-informed approach demonstrates greater efficiency over the function-agnostic method specifically for regression tasks, as it adapts the basis functions to the known output [156]. However, further research is required to effectively integrate basis function adaptation with physics-informed problem-solving using RPNNs. Currently, the function-agnostic approach is simpler, more versatile, and generally requires less optimization, potentially leading to lower computational costs.

### 2.8.4 Geometric Random Sampling Procedure for high-dimensional input

For the construction of an appropriate set of random basis functions when dealing with high-dimensional inputs, e.g., for the solution of the inverse problem (i.e., that of learning the effective PDEs from data) with multiple features, we suggest a different random sampling procedure [45], than the one usually implemented in RPNNs and in particular in Extreme Learning Machines [69, 91, 94] for the solution of the forward problem, i.e., that of the numerical solution of PDEs. Indeed, the above-mentioned scheme of sampling, using mesh of centers are prone to the "curse of dimensionality" in generating grid, and it may be already infeasible when the dimension is greater than 3. Since in the inverse problem, we aim at solving a high-dimensional over-determined system, it is important to parsimoniously select the underlying basis functions $\psi_j^{(i)}$, i.e., to seek for appropriate internal weights $W^{(i)}$ and biases $\boldsymbol{b}^{(i)}$ that lead to non-trivial functions.

In general, the weights $w_j^{(i)}$ and biases $b_j^{(i)}$ are uniformly random sampled from a subset of the input/feature space, e.g., $w_j^{(i)}$, $b_j^{(i)} \sim \mathcal{U}([-1,1]^{\gamma(i)}$, where a high dimension ($d$) of the input/feature space leads to the phenomenon of "curse of dimensionality". Indeed, it is necessary to use many function ($N \propto 10^d$) to correctly "explore" the input space and give a good basis function.

Hence, our goal is to construct $w_j^{(i)}$ and $b_j^{(i)}$ with a simple data-driven manifold learning in order to have a basis of functions $\psi_j^{(i)}$ that well describe the manifold $\mathcal{M}^{(i)}$ where the data $\boldsymbol{z}^{(i)}(\boldsymbol{x}_q, t_s) \in \mathcal{M}^{(i)}, \forall q, \forall s$ are embedded. It is well-known that the output of a neuron is given by a ridge function $f : \mathbb{R}^N \to \mathbb{R}$, where $N$ is the number of neurons, such that $f(z_1, \ldots, z_n) = g(\boldsymbol{a}^T \cdot \boldsymbol{z})$, where $g : \mathbb{R} \to \mathbb{R}$ and $\boldsymbol{a} \in \mathbb{R}^n$. The inflection point of the logistic sigmoid is at ($y = 0$, $\psi(y) = 1/2$). The points that satisfy the following relation [69]:

$$y = \boldsymbol{w}_j^{(i)} \cdot \boldsymbol{z}^{(i)}(\boldsymbol{x}_q, t_s) + b_j^{(i)} = 0 \tag{2.104}$$

form a hyperplane $\mathcal{H}_j^{(i)}$ of $\mathbb{R}^M$ (M dimension of $\boldsymbol{z}$) defined by the direction of $\boldsymbol{w}_j^{(i)}$. Along $\mathcal{H}_j$, $\psi_j^{(i)}$ is constantly $1/2$. We call the points $c_j^{(i)} \in \mathcal{H}_j^{(i)}$ the *centers* of the ridge function $\psi_j^{(i)}$. Here the goal is to select $N$ centers $c_j^{(i)}$ that are on the manifold $\mathcal{M}^{(i)}$ (note that this is not achieved by random weights and biases) and find directions $w_j^{(i)}$ that make $\psi_j^{(i)}$ non-constant/non-trivial along $\mathcal{M}^{(i)}$ (note that for ridge functions there are many directions for which this does not happen).

Thus, being $N << M$, we suggest to uniformly random sample $N$ points $c_j^{(i)}$ from $\boldsymbol{z}(\boldsymbol{x}_q, t_s)$ to be the centers of the functions $\psi_j^{(i)}$: in this way the inflection points of $\psi_j^{(i)}$ are on the manifold $\mathcal{M}$. Also, we independently randomly sample other $N$ points $\tilde{c}_j^{(i)}$ from the inputs $\boldsymbol{z}(\boldsymbol{x}_q, t_s)$. Then, we construct the hidden weights as:

$$\boldsymbol{w}_j^{(i)} = \tilde{c}_j^{(i)} - c_j^{(i)}, \tag{2.105}$$

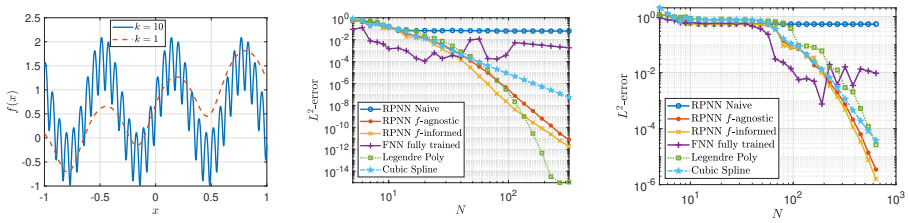
in order to set the direction $\boldsymbol{w}_j^{(i)}$ of the hyperplane $H_j^{(i)}$ parallel to the one connecting $\tilde{c}_j^{(i)}$ and $c_j^{(i)}$. By doing so, the ridge function will be constant on a direction orthogonal to the connection between two points in the manifold $\mathcal{M}^{(i)}$ and along this line will change in value, so it will be able to discriminate between the points lying on this direction. Thus, the biases $b_j^{(i)}$ are set as:

$$b_j^{(i)} = -\boldsymbol{w}_j^{(i)} \cdot c_j^{(i)}. \tag{2.106}$$

Eq. (2.104) ensures that $c_j^{(i)} \in \mathcal{H}_j^{(i)}$ is a center of the ridge function.

## 2.9 Numerical Examples for RPNN Convergence

In this section, we conduct numerical tests of the proposed RPNN-based best $L^2$ approximation method as in Eq. (2.73), illustratively we show 2 benchmark functions.



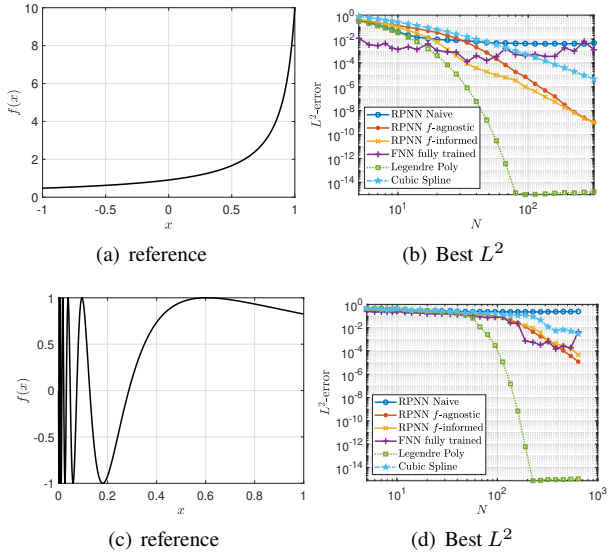

(a) references (b) results $k = 1$ (c) results $k = 10$

Figure 2.1: function $f_1$ in Eq. (2.107), with $k = 1$ and $k = 10$, presenting high-oscillations. (a) reference functions; (b)-(c) Convergence diagrams in $L^2$-norm, for $k = 1$ and $k = 100$, respectively. We compare Legendre Polynomials, RPNNs of best $L^2$ approximation, standard FNN and Cubic Spline. For the RPNNs we compare 3 different selection (naive, function-agnostic and function-informed) of the internal parameters as explained in Section 2.8. For the RPNNs, we report the mean accuracy out of 100 different Monte-Carlo selections of the internal parameters. For the FNN, we report the best network out of 10 runs with different initialization of the weights.

The training of the RPNNs is performed by employing COD as explained in Eq. (2.70), where we set the tolerance $\epsilon$ for the rank estimation to: $\epsilon = n \cdot \texttt{eps}(\|R\|_2)/1000$, where $n$ is the number of data points, $R$ is the matrix in (2.55) and eps is the built-in MATLAB function returning the floating-point relative accuracy (e.g., eps(1.0)=2.2204E−16). For a comparison of the use of the SVD-based and the COD-based least-square minimum norm solutions for different values of the tolerance $\epsilon$, see [156].

We compare RPNNs with classical numerical analysis methods, including the Legendre Polynomials, implemented through a Lagrange barycentric interpolation formula, and the Cubic spline, implemented with the `spline` built-in function of `MATLAB`. For

the implementation of different strategies for the random selections of the internal parameters of the RPNNs, see Section 2.8. In particular in the following for all the tests, we have compared the use of (a) naive random generated RPNNs; (b) function-agnostic RPNNs and (c) function-informed RPNNs, without employing any further training of the basis functions. Additionally, for comparison purposes we also compared with a standard single hidden layer FNN, *fully trainable*, with $N$ neurons in the hidden layer and logistic sigmoid activation functions, as implemented by the `MATLAB` deep learning toolbox (e.g., the function `net=feedforwardnet(N)`).




(a) reference (b) Best $L^2$

(c) reference (d) Best $L^2$

Figure 2.2: Benchmark examples, function $f_2$ in Eq. (2.108) and $f_3$ in Eq. (2.109) approaching discontinuity. (a)-(c) reference functions; (b)-(d) Convergence diagrams $L^2$-norm comparing Legendre Polynomial, RPNNs of best $L^2$ approximation, standard FNN and Cubic Spline. For the RPNNs we compare 3 different selection (naive, function-agnostic and function-informed) of the internal parameters as explained in Section 2.8. For the RPNNs, we report the mean accuracy out of 100 different Monte-Carlo selections of the internal parameters. For the FNN, we report the best network out of 10 runs with different initialization of the weights.

For the training process of the FNNs, we utilized the LMA, as implemented in `MATLAB` by the option `net.trainFcn='trainlm'`, with 1000 as maximum number of epochs and $1e-10$ as minimum gradient of the loss function. In the following case studies, `trainlm` was able to obtain lower MSEs than any of the other algorithms we tested. Moreover, the training process is influenced by the first random initialization of the weights, therefore, we repeated 10 times the training starting from different random initializations, and we report the best performing FNN from these runs. For each number of neurons $N$, both FNN and RPNNs were trained on $n = 5N$ equally spaced data points

within the domain of interest (training set).

### 2.9.1 Case study 1: functions with high-oscillations

Here, we consider the approximation of the function[149]:

$$f_1(x) := \log(\sin(10kx) + 2) + \sin(kx), \qquad x \in [-1, 1], \tag{2.107}$$

for $k = 1$ and $k = 10$. The reference functions and the results are depicted in Figure 2.1. As can be noted, for $k = 10$ the reference function presents high frequency oscillations. We report the $L^2$-error computed on a dense grid of $10,000$ equally spaced points in the interval $[-1, 1]$. Figure 2.1, highlights the high accuracy of both function-agnostic and function-informed RPNNs, matching the exponential convergence of the Legendre polynomials up to 8 digits of accuracy. However, beyond 8 digits of accuracy, RPNNs seems to transition towards a less efficient high-order algebraic convergence (greater than cubic splines), suggesting potential ill-conditioning issues during training. In this case, for high-oscillations, naive random selection RPNNs fails and plateaus around only 1 digit of accuracy. Meanwhile, the performance of fully trained FNNs exhibit decent performance in adapting its basis to the high-oscillations, yet struggling to reach more than 4 digits of accuracy.

### 2.9.2 Case study 2: Numerical challenges in approximating functions near singularities

We consider the approximation of two functions in the vicinity of a discontinuity.

The first diverges approaching the discontinuity at $x = -1 - \epsilon$:

$$f_2(x) := \frac{1}{1 + \epsilon - x}, \qquad x \in [-1, 1] \tag{2.108}$$

and the second infinitely increases its oscillations approaching the discontinuity at $x = -\epsilon$ :

$$f_3(x) := \sin\left(\frac{1}{x + \epsilon}\right), \qquad x \in [0, 1], \qquad \epsilon = \frac{1}{10\pi}, \tag{2.109}$$

where for the selected the parameter $\epsilon = \frac{1}{10\pi}$, the function $f_3$ has 10 zeros and 5 oscillations in the domain $[0, 1]$ [123]. The reference functions $f_2$ and $f_3$ and the corresponding numerical results are depicted in Figure 2.2. For both $f_2$ and $f_3$ examples, we report the $L^2$-error computed on a dense grid of $10,000$ equally spaced points in the intervals $[-1, 1$ and $[0, 1]$, respectively. The two functions are infinitely differentiable in the considered domain, thus, as can be seen in Figure 2.2, the Legendre polynomials converge exponentially. However, the vicinity of the singularities reduces the convergence rates of the non-naive RPNNs. Nevertheless, RPNNs ultimately reach 8 digits of accuracy for $f_2$ and 4 digits of accuracy for $f_3$, outperforming the cubic spline interpolations. While FNNs encounter greater difficulty with the diverging function $f_2$, they accommodate the high oscillations of $f_3$.

# 3 Learning nonlinear operators through Random Projection operator networks

## 3.1 A new paradigm: State-of-the-art of Operator Learning

In recent years, several advanced ML-based methods for approximating nonlinear operators, particularly focused on partial differential operators, have emerged as prominent approaches: Deep Operator Networks (DeepONet) [86], Fourier Neural Operators (FNOs) [96], and Graph-based Neural Operators [97, 207] are among the most notable. DeepONet extends the operator approximation framework from the 1990s by Chen and Chen [111], utilizing branch and trunk networks to handle input functions and spatial variables or parameters separately. This approach offers a flexible and robust framework for learning operators in dynamical systems, with various network architectures available for either or both branches, enabling solutions to complex dynamical system problems [86, 78]. Such DeepONet operates at a higher level of abstraction by addressing the identification/approximation of functional operators, i.e., maps between infinite dimensional functional spaces.

DeepONets has been generalized into the broader class of Neural Operators. FNOs [96] take advantage of the Fourier transform to capture global patterns and dependencies within data, using convolutional layers in the frequency domain and applying the inverse Fourier transform to return to the original domain. This enables efficient learning of complex, high-dimensional inputs and outputs with long-range correlations, making it particularly useful for large spatial domain problems.

The family of graph-based neural Operators [207, 97] represent nonlinear operators as graphs, where nodes correspond to spatial locations of the output function, learning the network kernel to approximate the PDE. Each layer defines a map between infinite-dimensional spaces with finite-dimensional parametric dependencies. The authors also demonstrate a universal approximation result, proving that this framework can approximate any continuous nonlinear operator. For a comprehensive review on the applications of neural operators, the interested reader can refer to the recent work in [208].

DeepONets and Neural Operators, while powerful, have notable limitations. Training

these models often requires multiple iterations over large datasets to update parameters in high-dimensional spaces, resulting in significant computational time and memory consumption. The complexity of nonlinear operators and the size of the problem domain further exacerbate the computational load. In addition, hyperparameter tuning, regularization, and model selection introduce further computational overhead. As a result, training Neural Operators typically demands considerable resources, such as high-performance computing clusters or GPUs.

These computational demands can also affect convergence and numerical accuracy. The high-dimensional parameter space makes it difficult to reach a near-global optimum, and the optimization algorithm may become trapped in local minima or plateaus, limiting the network's ability to accurately approximate underlying operators. Addressing these challenges requires a careful balance between optimization strategies, regularization methods, and dataset size to achieve the desired level of accuracy without excessive computational cost (see critical discussions and approaches to deal with this cost-accuracy tradeoff in [128, 209, 210, 211]).

## 3.2 Description of the problem

In this section, we focus on the challenging task of learning linear and nonlinear functional operators $\mathcal{F} : U \to V$ which constitute maps between two infinite-dimensional function spaces $U$ and $V$. Here, for simplicity, we consider both $U$ and $V$ to be subsets of the set $C(\mathbb{R}^d)$ of continuous functions on $\mathbb{R}^d$. The elements of the set $U$ are functions $u : X \subseteq \mathbb{R}^d \to \mathbb{R}$ that are transformed to other functions $v = \mathcal{F}[u] : Y \subseteq \mathbb{R}^d \in \mathbb{R}$ through the application of the operator $\mathcal{F}$. We use the following notation for an operator evaluated at a location $\boldsymbol{y} \in Y \subseteq \mathbb{R}^d$

$$v(\boldsymbol{y}) = \mathcal{F}[u](\boldsymbol{y}). \tag{3.1}$$

These operators play a pivotal role in various scientific and engineering applications, particularly in the context of PDEs. By effectively learning (discovering from data) such nonlinear operators, we seek to enhance our understanding and predictive capabilities in diverse fields, ranging from fluid dynamics and materials science to financial and biological systems and beyond [19, 86, 66, 128, 78, 45, 67]. One prominent example is the right-hand side (RHS) evolution operators $\mathcal{L}$ associated with PDEs, which govern the temporal evolution of the associated system dynamics. We denote these *evolution* operators in the following way:

$$v(\boldsymbol{x}, t) = \frac{\partial u(\boldsymbol{x}, t)}{\partial t} = \mathcal{L}[u](\boldsymbol{x}, t), \qquad \boldsymbol{x} \in \Omega, \quad t \in [0, T], \tag{3.2}$$

where $u : \Omega \times [0, T] \subseteq \mathbb{R}^d \to \mathbb{R}$ is the unknown solution of the PDE. Given a state profile $u(\cdot, t) : \Omega \to \mathbb{R}$ at each time $t$, e.g., the initial condition $u_0$ of the system at time $t = 0$, the *evolution operator* (the RHS of DEs) $\mathcal{L}$ provides the corresponding time derivative (the output $v(\cdot, t)$) of the system at that time $t$. Again, a method for learning the RHS of PDEs with a different ANN architecture than DeepONet was proposed back in the '90s in [112]. There, the RHS was estimated in terms of spatial derivatives.

One can also learn the corresponding *solution operators* $\mathcal{S}_t$, which embody both the time integration and the satisfaction of boundary conditions, of the underlying physical phenomena. Given the initial condition $u_0$ at time $t = 0$, the solution operator outputs the state profile $u(\cdot, t) : \Omega \to \mathbb{R}$ after a certain amount of time $t$:

$$v(\boldsymbol{x}) = u(\boldsymbol{x}, t) = \mathcal{S}_t[u_0](\boldsymbol{x}). \tag{3.3}$$

We will deal with this problem in the part II that will follow.

Although our objective is to learn functional operators from data, which take functions $(u)$ as input, we must discretize them to effectively represent them and be able to apply network approximations. One practical approach, as implemented in the DeepONet framework, is to use the function values $(u(\boldsymbol{x}_j))$ at a sufficient, but finite, number of locations $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m$, where $\boldsymbol{x}_j \in X \subseteq \mathbb{R}^d$; these locations are referred to as "sensors". Other methods to represent functions in functional spaces include the use of Fourier coefficients [96], wavelets [212], spectral Chebychev basis [213] and graph neural operators [207]. Regarding the availability of data for the output function, we encounter two scenarios. In the first scenario, the functions in the output are known at the same fixed grid $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n$, where $y_i \in Y$; this case is termed as "aligned" data. Conversely, there are cases where the output grid may vary randomly for each input function, known as "unaligned" data. If this grid is uniformly sampled and dense enough, interpolation can be used to approximate the output function at fixed locations. Thus, this leads us back to the aligned data case. However, if the output is only available at sparse locations, interpolation becomes impractical. As we will see later in the text, despite this challenge, our approach can address this scenario, albeit with a higher computational cost for training the ML model (since, in such cases, the fixed structure of the data cannot be fully leveraged).

## 3.3 Preliminaries on DeepONets

Deep Operator Networks (DeepONet) [93] represent a novel neural network architecture specifically designed to learn functional operators. Unlike traditional neural networks focused on mapping input vectors to output vectors (e.g., regression, classification), DeepONet operates at a higher level of abstraction. It aims to approximate functionals and operators, which are mathematical objects that map functions from one infinite-dimensional space to another. This capability is theoretically grounded in the universal approximation theorem for operators, proven by Chen & Chen in 1995 [111]. This theorem guarantees the existence of neural networks that can approximate any continuous functional operator to an arbitrary degree of accuracy. Therefore, as universal approximators, FNNs have the capability to approximate continuous functions effectively [171, 172, 153, 175]. However, a lesser-known theorem by Chen & Chen (1995) [111], which gained prominence with the advent of DeepONet by Lu et al. (2021) [86] and Fourier Neural Operator (FNO) by Li et al. (2020) [96], asserts the existence of a neural network architecture capable of approximating any continuous nonlinear operator to an arbitrary degree of accuracy.

Based on the theorem 2.1.2, one can also prove the following one on operators:

**Theorem 3.3.1** (Universal approximation for operators [111])**.** Suppose that $\psi$ is a Tauber-Wiener function, $X$ is a Banach space, and $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are two corresponding compact sets. Let $U$ be a compact set in $C(K_1)$, and let $\mathcal{F} : U \to C(K_2)$ be a nonlinear continuous operator. Then, for any $\epsilon > 0$, there are $N, M, m \in \mathbb{N}$, and network parameters $w_{ki}$, $\xi_{kij}$, $\theta_{ki}$, $\beta_k \in \mathbb{R}$, $\boldsymbol{c}_k \in \mathbb{R}^d$, $\boldsymbol{x}_j \in K_1$, with $k = 1, \dots, N$, $i = 1, \dots, M$, $j = 1, \dots, m$, such that $\forall u \in U, \boldsymbol{y} \in K_2$:

$$\left| \mathcal{F}(u)(\boldsymbol{y}) - \sum_{k=1}^{N} \sum_{i=1}^{M} w_{ki} \psi\big( \sum_{j=1}^{m} \xi_{kij} u(\boldsymbol{x}_j) + \theta_{ki} \big) \cdot \psi(\boldsymbol{c}_k \cdot \boldsymbol{y} + \beta_k) \right| < \epsilon. \tag{3.4}$$

To briefly describe how the above Theorem in the original paper of Chen & Chen in [111] works, let us assume that our goal is to approximate an operator $\mathcal{F}$, acting on the set of functions $u \in U$. These functions $u$ (which are inputs to the DeepONet) are assumed to be known and sampled at $m$ fixed locations $x_j$ in the domain $K_1$. The vector $\boldsymbol{U} = (u(\boldsymbol{x}_1), u(\boldsymbol{x}_2), \dots, u(\boldsymbol{x}_m)) \in \mathbb{R}^{m \times 1}$ (a column vector) is the input of a single-hidden layer FNN with $M$ neurons, the so-called *branch network*, that process the function values space. At the same time there is a second single-hidden layer FNN with $N$ neurons, the so-called *trunk network*, that process the new location $\boldsymbol{y} \in K_2 \subset \mathbb{R}^{1 \times d}$ (for convenience let us assume it as a row vector) in which we have to evaluate the transformed function $\mathcal{F}[u]$. For convenience, let us define the vector $\boldsymbol{B} = (B_1, B_2, \dots, B_M) \in \mathbb{R}^{M \times 1}$ (column vector) of hidden layers value of the branch network:

$$B_i(\boldsymbol{U}) = \psi\big( \sum_{j=1}^{m} \xi_{kij} u(\boldsymbol{x}_j) + \theta_{ki} \big), i = 1, 2, \dots, M, \tag{3.5}$$

and let us define the vector $\boldsymbol{T} = (T_1, T_2, \dots, T_N) \in \mathbb{R}^{1 \times N}$ (row vector):

$$T_k(\boldsymbol{y}) = \psi({\boldsymbol{c}_k}^T \cdot \boldsymbol{y} + \beta_k), k = 1, 2, \dots, N. \tag{3.6}$$

Then, the output of the network as in Eq. (3.4) can be written as:

$$\mathcal{F}[u](\boldsymbol{y}) \simeq \sum_{k=1}^{N} \sum_{i=1}^{M} w_{ki} B_i(\boldsymbol{U}) T_k(\boldsymbol{y}) \Leftrightarrow \mathcal{F}[u](y) = \boldsymbol{T} W \boldsymbol{B} = \langle \boldsymbol{T}, \boldsymbol{B} \rangle_W, \tag{3.7}$$

where the matrix $W \in \mathbb{R}^{N \times M}$ has elements $w_{ki}$. As can be seen, the output of the scheme is a weighted inner product $\langle \cdot, \cdot \rangle_W$ of the trunk and branch networks. In the next section, we will take advantage of this formulation for an efficient and accurate training of the network through the use of random bases.

We note, that the original theorem 3.3.1 considers only two shallow FNNs with a single hidden layer. On the other hand, DeepONet uses deep networks instead, but also can incorporate any other type of networks such as CNNs. An extension of the theorem 3.3.1, given by Lu et al. [86], states that the branch network and the trunk network can be chosen by diverse classes of ANNs, which satisfy the classical universal approximation theorem. Also, while the Chen and Chen architecture in (3.4) does not include an

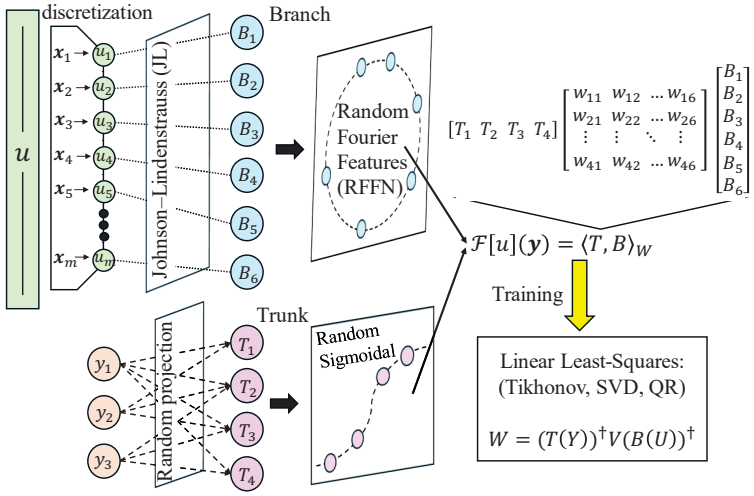



Figure 3.1: Schematic of the Random Projection-based Operator Network (RandONet). The RandONets first discretizes the input function ($u$) over a fixed grid of spatial points. Then it separately embeds the space of the spatial locations ($\boldsymbol{y}$) into a random hidden layer (e.g., with sigmoid activations functions) and the space of the discretized functions into low-distortion kernel-embedding (e.g., with Johnson-Lindenstrauss random projections [199] or Rahimi and Recht RFFs [158]). Finally, the output is composed of a weighted ($W$) inner product of the branch ($B$) and trunk ($T$) features. The training can be performed through linear least-squares techniques (e.g., Tikhonov regularization, SVD and QR decomposition).

output bias, the DeepONet usually utilize biases to improve generalization performance [86]. More broadly, DeepONets can be considered conditional models, where $\mathcal{F}[u](\boldsymbol{y})$ represents a function of $\boldsymbol{y}$ given $u$. These two independent inputs, $u$ and $\boldsymbol{y}$, are given as inputs to the trunk and branch networks, respectively. At the end, the embeddings of $u$ and $\boldsymbol{y}$ are combined through an inner product operation. However, the challenge remains in finding efficient approaches to train these networks. It is also worth noting that, as it happens for shallow FNNs, while the universal approximation theorem for operators guarantees the existence of a successful approximation DeepONet, it does not offer a numerical method for constructing the specific weights and biases of the networks. Furthermore, deep learning networks used in DeepONet do not come without limitations. While they enhance the models' ability to capture complex relationships, they introduce challenges in the optimization process. Specifically, determining the values of the networks *parameters and hyperparameters* requires significant computational resources, entailing complexity that can lead to moderate generalization ability and/or numerical approximation accuracy. Hence, it is nearly implicit that training such DeepONet heavily relies on parallel computing and GPU-based computations.

Here we present a computationally efficient method for approximating nonlinear

operators, based on shallow networks with a single hidden layer, as in the paper of Chen and Chen in [111], coupled with random projections, that relaxes the "curse of dimensionality" in the training process.

## 3.4 Random Projection-based Operator Networks (RandONets)

In this section, we present Random Projection-based Operator Networks (RandONets) [157] for approximating nonlinear operators. Building on previous works, we first demonstrate that the proposed shallow – single hidden layer – RPNNs are universal approximators of non-linear operators. Then we discuss how RandONets can be used in both the aligned and unaligned data cases.

### 3.4.1 RandONets as universal approximators of nonlinear operators

In this section, we prove that RandONets are universal approximators of nonlinear operators. Following the methodological thread in [111], we first state the following proposition:

**Proposition 3.4.1.** Let $K \subset \mathbb{R}^d$ compact and $U \subset C(K)$ compact and consider a parametric family of random activation functions $\{\psi(\boldsymbol{x};\boldsymbol{\alpha}) \ : \boldsymbol{x} \in \mathbb{R}^d, \ \boldsymbol{\alpha} \in A\}$, where $\boldsymbol{\alpha} \in A$ is a vector of randomly chosen (hyper) parameters, and assume that $\psi$ are uniformly bounded in $\mathbb{R}^d \times A$. Let $p$ be a probability distribution on $A$. Given any $\epsilon$, there exists a $N \in \mathbb{N}$ and i.i.d. sample $\boldsymbol{\alpha}_1, \cdots, \boldsymbol{\alpha}_N$ from $p$, chosen independently of $f$, such that for every $f \in U$ the random approximation

$$f_\epsilon(\boldsymbol{x}) = \sum_{j=1}^{N} c_j[f]\psi(\boldsymbol{x};\boldsymbol{\alpha}_j), \tag{3.8}$$

approximates $f$ in the sense that with high probability

$$\|f - f_\epsilon\|_{L^2(\mu)} < \epsilon, \tag{3.9}$$

for a suitable probability measure $\mu$ over $K$. Moreover, if $\psi(\boldsymbol{x};\boldsymbol{\alpha}) = \varphi(\boldsymbol{\alpha} \cdot \boldsymbol{x})$, for a $L$-Lipschitz function $\psi$, the above approximation is uniform (i.e., in the supremum norm).

*Proof.* We only show the uniform approximation result. The $L^2(\mu)$ approximation follows similar arguments.

We apply Theorem 3 in [111]. Given a continuous sigmoid (non-polynomial) function $\sigma$, for every $\epsilon > 0$, there exist $N \in \mathbb{N}$, $(\hat{\theta}_i, \boldsymbol{\omega}_i) \in \mathbb{R} \times \mathbb{R}^d$, $i = 1, \cdots, N$ such that for every $f \in U \subset C(K)$, there exist a linear continuous functional on $U$, $f \mapsto c_i[f]$, with the property that

$$|f(\boldsymbol{x}) - \sum_{i=1}^{N} c_i[f]\sigma(\boldsymbol{\omega}_i \cdot \boldsymbol{x} + \hat{\theta}_i)| < \epsilon, \quad \forall \boldsymbol{x} \in K. \tag{3.10}$$

This approximation is deterministic. Note that even though according to Theorem 3 op. cit. this approximation holds for any Tauber-Wiener function (i.e., even for non-continuous $\sigma$) here we must insist on the continuity of $\sigma$.

In the above, we obtain a uniform approximation $f_\epsilon$ to $f$ in terms of

$$f_\epsilon(\boldsymbol{x}) = \sum_{i=1}^{N} c_i[f]\sigma(\boldsymbol{\omega}_i \cdot \boldsymbol{x} + \hat{\theta}_i) \tag{3.11}$$

We emphasize the $\sigma$ is chosen to be a continuous and bounded sigmoid function. Note that the choice of $N$, $(\theta_i, \omega_i)$, are independent of $f$, while the coefficients $c_i[f]$ depend on $f$ and in fact $f \mapsto c_i[f]$ is a linear continuous functional on $U$.

To connect with the random features approach of Rahimi and Rechts we first employ an expansion of each sigmoid in (3.10) in terms of an RBF (which is eligible to a random features expansion a la Rahimi and Rechts) and then follow with the expansion of the RBFs in random features. We follow the subsequent steps:

(a) For each $i = 1, \cdots, N$, we consider the function $\phi_i$, defined by $K \ni \boldsymbol{x} \mapsto \phi_i(\boldsymbol{x}; \boldsymbol{\omega}_i, \hat{\theta}_i) := \sigma(\boldsymbol{\omega}_i \cdot \boldsymbol{x} + \hat{\theta}_i)$. By the properties of $\sigma$, the functions $\phi_i \in C(K)$. Hence, we may apply the approximation of each $\phi_i$ in terms of an RBF neural network. Using standard results (e.g., Theorem 3 [214]) we have that if $g \in C(\mathbb{R}^d)$ is a bounded RBF $\mathcal{S} := \text{span}\{g(a\boldsymbol{x} + \boldsymbol{b}) \ : \ a \in \mathbb{R}, \ \boldsymbol{b} \in \mathbb{R}^d\}$ is dense in $C(K)$. Note that without loss of generality we may impose the extra assumption that $g$ can be expressed in terms of the inverse Fourier transform of some function (i.e., an element of a function space on which the Fourier transform is surjective, for example, $g$ belongs in the Schwartz space). This assumption also allows us to invoke the standard results of [180], [215] leading to the same density result). Note that this step does not affect the generality of our results, as it is only used in the intermediate step (a) which re-expands the general sigmoids used in (3.10) into a more convenient basis on which the step (b) is applicable. Moreover, the choice of $g$ is not unique.

Using the above result, we can approximate each $\phi_i$ in terms of the functions

$$\phi_{i,\epsilon}(\boldsymbol{x}) = \sum_{j=1}^{M_i} w_{ij} g(a_{ij}\boldsymbol{x} + \boldsymbol{b}_{ij}), \tag{3.12}$$

where, importantly, the $(w_{ij}, a_{ij}, \boldsymbol{b}_{ij}) \in \mathbb{R} \times \mathbb{R} \times \mathbb{R}^d$ are independent of the choice of the function $f$ (as they only depend on $(\boldsymbol{\omega}_i, \hat{\theta}_i) \in \mathbb{R}^d \times \mathbb{R}$, which are independent of $f$ – see (3.10)). The function $\phi_{i,\epsilon}$ satisfies the property $\|\phi_i - \psi_{i,\epsilon}\| < \frac{\epsilon}{N}$, in the uniform norm $\forall \epsilon > 0$. Combining (3.10) (and (3.11)) with (3.12), we obtain an approximation $f_\epsilon^{RBF}$ to $f_\epsilon$ in terms of

$$f_\epsilon^{RBF}(\boldsymbol{x}) = \sum_{i=1}^{N} \sum_{j=1}^{M_i} c_i[f] w_{ij} g(a_{ij}\boldsymbol{x} + \boldsymbol{b}_{ij}), \tag{3.13}$$

such that $\|f_\epsilon - f_\epsilon^{RBF}\| < \epsilon$, hence satisfying by (3.10) that $\|f - f_\epsilon^{RBF}\| < 2\epsilon$ (in the uniform norm).

(b) Now, by appropriate choice of the RBF function $g$ we apply Rahimi and Recht for a further expansion of each term $g(a_{ij}\boldsymbol{x} + \boldsymbol{b}_{ij})$, using the random features RPNN. By the choice of $g$ as above, this is now possible, and the results of Rahimi and Recht [160](theorem 2.4.2), are now applicable to $g$. This holds, since RBFs can be elements of the RKHS that the Rahimi and Recht framework applies to, i.e., they belong to the space of functions $\mathcal{G}_p$, defined in (2.37). For such a choice, Theorem 3.1 in [160] can be directly applied to each of the RBF $g$ in the function (3.13). Under the extra assumption that $\phi(\boldsymbol{x};\boldsymbol{\alpha}) = \varphi(\boldsymbol{\alpha}\cdot\boldsymbol{x})$, for $\varphi$ L-Lipschitz (which without loss of generality can be shifted so that $\varphi(0) = 0$ and scaled so that $\sup|\varphi| \leq 1$), we can also apply Theorem 3.2 op. cit for a corresponding uniform approximation. We only present the second case, the first one being similar. There are two equivalent ways to proceed.

b1) Using Theorem 3.2 op cit, for an $L$-Lipschitz $\varphi$ as defined above, for any $\delta > 0$ there exists a random function $g_\delta$ of the form

$$g_\delta(\boldsymbol{x}) = \sum_{k=1}^{K} \hat{c}_k \varphi(\boldsymbol{\alpha}_k \cdot \boldsymbol{x}), \tag{3.14}$$

where $\boldsymbol{\alpha}_k$ is i.i.d. randomly sampled from a chosen distribution $p$, which approximates $\hat{\epsilon}(\delta)$ close $g$ with probability at least $1-\delta$.

Using (3.14) into (3.13) we obtain

$$f_{\epsilon,\delta}(\boldsymbol{x}) = \sum_{i=1}^{N}\sum_{j=1}^{M_i}\sum_{k=1}^{K} c_i[f] w_{ij} \hat{c}_k \varphi(\boldsymbol{\alpha}_k \cdot (a_{ij}\boldsymbol{x} + \boldsymbol{b}_{ij})), \tag{3.15}$$

which, if $\delta$ is chosen such that $\hat{\epsilon}(\delta) < \frac{\epsilon}{NM}$, satisfies $\|f^{RBF} - f_{\epsilon,\delta}\| < \epsilon$, hence, $\|f - f_{\epsilon,\delta}\| < 3\epsilon$.

Using a resummation of (3.15) in terms of a single summation index $\ell$, we end up with an approximation $f_{\epsilon,\delta}$ for $f$ in the form

$$f_{\epsilon,\delta}(\boldsymbol{x}) = \sum_{\ell=1}^{\hat{N}} \hat{w}_\ell[f] \varphi(\boldsymbol{\alpha}_\ell \cdot (a_\ell \boldsymbol{x} + \boldsymbol{b}_\ell)), \tag{3.16}$$

where $\hat{w}_\ell[f] = c_i[f] w_{ij} \hat{c}_k$ are continuous functionals on $U$. Hence, we obtain an approximation in terms of shifted and re-scaled $L$-Lipschitz random feature functions.

b2) One possible drawback of this expansion is that – see (3.15) it depends both on the $a_j$, $\boldsymbol{b}_j$ and the $\boldsymbol{\alpha}_k$ and not on the $\boldsymbol{\alpha}_k$ only. An alternative could be to expand each one of the $g_j(\boldsymbol{x}) := g(\alpha_j \boldsymbol{x} + \boldsymbol{b}_j)$ separately. If it holds that $g_j \in \mathcal{G}$ for each $j$, then

$$g_j(\boldsymbol{x}) = \sum_{k=1}^{K} \hat{c}_{jk} \varphi(\boldsymbol{\alpha}_k^{(j)} \cdot \boldsymbol{x}), \;\; j = 1, \cdots, M, \tag{3.17}$$

where $\boldsymbol{\alpha}^{(j)} := \{\boldsymbol{\alpha}_k^{(j)}, \; : \; k = 1, \cdots, K\} \sim_{i.i.d} p$ for each $j = 1, \cdots, M$ and with the $\hat{\boldsymbol{\alpha}}^{(j)}$ for different $j$ being independent. When using this approach, we get the expansion (3.16) with $\alpha_k$ i.i.d. from our initial distribution $p$. Upon resummation the stated result follows. □

Based on the proposition 3.4.1, we can now prove the following proposition for universal approximation of functional $\mathcal{F} : U \to \mathbb{R}$ in terms of the RPNN:

**Proposition 3.4.2** (Random Projection Neural Networks (RPNNs) for functionals)**.** Adopting the framework from Proposition 3.4.1, and additionally, let $U$ be a compact subset of $C(K)$ and $\mathcal{F}$ be a continuous functional in $U$. Let us define the compact set $U_m \subseteq \mathbb{R}^d$ of vectors, whose elements consist of the values of the function $u \in U$ on a finite set of $m$ grid points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m \in \mathbb{R}^d$ and denote the vector $\boldsymbol{u} := [u(\boldsymbol{x}_1), \ldots, u(\boldsymbol{x}_m)] \in \mathbb{R}^m$.

Then, with high probability, w.r.t. $p$, for any $\epsilon > 0$, there exist $M, m \in \mathbb{N}$, $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_M \in A$, i.i.d distributed from $p$, such that:

$$\left\| \mathcal{F}(u) - \sum_{i=1}^{M} w_i \varphi\left(\boldsymbol{\alpha}_i \cdot \boldsymbol{u}(\boldsymbol{x})\right) \right\|_{\infty} < \epsilon, \quad \forall u \in U. \tag{3.18}$$

*Proof.* The representation of the function $u \in U$ through a finite set of $m$ evaluations $u(\boldsymbol{x}_j)$ is possible by the Tietze Extension Theorem for functionals from the set $U_m$ to $U$ (see [111] for more details). Then the proof comes directly from Proposition 3.4.1. □

Finally, using the above ideas and results and possibly allowing for different random embeddings for the branch and trunk networks, we can prove the following theorem:

**Theorem 3.4.1** (RandONets universal approximation for Operators)**.** Adopting the framework of Propositions 3.4.1 and the notation of Theorem 3.2, and additionally, let: $X$ be a Banach Space, and $K_1 \subset X$, $K_2 \subset \mathbb{R}^d, U \subset C(K_1)$ be compact sets, and $\mathcal{F} : U \to C(K_2)$ be a continuous (in the general case nonlinear) operator. Then, with high probability w.r.t. $p$, for any $\epsilon > 0$, there exist positive integers $M, N, m \in \mathbb{N}$, and network (hyper)parameters $\boldsymbol{\alpha}_1^{br,tr}, \ldots, \boldsymbol{\alpha}_N^{br,tr} \in A^{br,tr}$, i.i.d distributed from $p_{\boldsymbol{\alpha}}$ such that:

$$\left\| \mathcal{F}(u)(\boldsymbol{y}) - \sum_{k=1}^{N} \sum_{i=1}^{M} w_{ki} \varphi^{br}\left(\boldsymbol{\alpha}_i^{br} \cdot \boldsymbol{u}(\boldsymbol{x})\right) \varphi^{tr}(\boldsymbol{\alpha}_k^{tr} \cdot \boldsymbol{y}) \right\|_{\infty} < \epsilon, \quad \forall u \in U, \boldsymbol{y} \in K_2, \tag{3.19}$$

where the superscripts $br, tr$ correspond to branch and trunk networks and can be chosen in generally independently.

*Proof.* From the Proposition 3.4.1, we have that with high probability, for any $\epsilon_1 > 0$, there are $N \in \mathbb{N}$, $\tilde{w}_k[\mathcal{F}[u]]$ and $\boldsymbol{\alpha}_k^{tr} \in A^{tr}$, such that

$$\left\| \mathcal{F}(u)(\boldsymbol{y}) - \sum_{k=1}^{N} \tilde{w}_k[\mathcal{F}[u]] \varphi^{tr}(\boldsymbol{\alpha}^{tr} \cdot \boldsymbol{y}) \right\|_{\infty} < \epsilon_1. \tag{3.20}$$

Moreover, from Proposition 3.4.1, we have that for any $k = 1, \ldots, N$, $\tilde{w}_k[\mathcal{F}[u]]$ is a continuous functional on $U$. We can therefore repeatedly apply Proposition 3.4.2, for

each $k$, and obtain approximations of each functional $\tilde{w}_k[\mathcal{F}[u]]$ on $U_m$. Thus, with high probability, for any $\epsilon_2 > 0$ there exist $m, M \in \mathbb{N}$, $w_{ki}$, $\boldsymbol{\alpha}_i^{br} \in A^{br}$, such that:

$$\left\| \tilde{w}_k[\mathcal{F}[u]] - \sum_{i=1}^{M} w_{ik} \varphi^{br} \left( \boldsymbol{\alpha}_i^{br} \cdot \boldsymbol{u}(\boldsymbol{x}) \right) \right\|_{\infty} < \epsilon_2. \tag{3.21}$$

Combining (3.20) and (3.21) we obtain Eq. (3.19). This completes the proof. □

### 3.4.2 Implementation of RandONets

In this section, we present the architecture of RandONets, depicted in Figure 3.1. As in Theorem 3.3.1, we use two single-hidden-layer FNNs with appropriate random bases as activation functions. We employ (nonlinear) random based projections for embedding, in the two separate hidden layer features, both the (high-dimensional) space of the discretized function ($u$) and the domain (low-dimensional) of spatial locations ($\boldsymbol{y}$) of the transformed output ($v(\boldsymbol{y}) = \mathcal{F}[u](\boldsymbol{y})$).

Specifically, we propose leveraging nonlinear random projections to embed the space of spatial locations efficiently, employing parsimoniously chosen random bases. Thus, the random projected-based trunk feature vector $\boldsymbol{T} = (T_1, \ldots, T_N)$, as denoted in Eq. (3.6), can be re-written as:

$$\boldsymbol{T} = \boldsymbol{\varphi}_n^{tr}(\boldsymbol{y}; \boldsymbol{\alpha}^{tr}) = [\varphi(\boldsymbol{y} \cdot \boldsymbol{\alpha}_1^{tr} + b_1), \ldots, \varphi(\boldsymbol{y} \cdot \boldsymbol{\alpha}_N^{tr} + b_N)], \tag{3.22}$$

where $\boldsymbol{\alpha}_k^{tr} \in \mathbb{R}^d$, $b_k$, $j = 1, \ldots, N$ are i.i.d. randomly sampled from a continuous probability distribution function.

Here, when the domain of $\mathcal{F}[u]$ is a one-dimensional interval $[a, b] \subseteq \mathbb{R}$, we select the activation function $\varphi$ of the trunk network to be the hyperbolic tangent, and we utilize a parsimonious function-agnostic randomization of the weights as explained in [156, 69, 95]. In particular, the weights $\boldsymbol{\alpha}_j^{tr}$ are uniformly distributed as $\mathcal{U}[-a_U, a_U]$. The bounds $a_U$, of the uniform distributions, have been optimized in [156, 69, 95].

For the branch network, we have implemented two types of embeddings:

- Linear random Johnson-Lindenstrauss (JL) embeddings [199], in which case, we denote the branch feature vector $\boldsymbol{B} = (B_1, \ldots, B_M)$ as:

$$\boldsymbol{B} = \boldsymbol{\phi}_M^{br}(\boldsymbol{U}) = \boldsymbol{\phi}_M^{JL}(\boldsymbol{U}) = \frac{1}{\sqrt{M}} R \boldsymbol{U}, \tag{3.23}$$

  where $R$ is a matrix with elements that are sampled from a standard Gaussian distribution and $\boldsymbol{U}$ is the vector of function evaluation in the input grid.

- Nonlinear random embeddings [160]. Here, for our illustrations, we use a random Fourier feature network (RFFN) [158], as embedding of the functional space:

$$\begin{aligned} \boldsymbol{B} &= \boldsymbol{\varphi}_M^{br}(\boldsymbol{U}; \boldsymbol{\alpha}^{br}) = \boldsymbol{\varphi}_M^{RFFN}(\boldsymbol{U}; \boldsymbol{\alpha}^{br}, \boldsymbol{b}^{br}) = \\ &= \frac{1}{m} \sqrt{\frac{2}{M}} [\cos(\boldsymbol{\alpha}_1^{br} \cdot \boldsymbol{U} + b_1^{br}), \ldots, \cos(\boldsymbol{\alpha}_M^{br} \cdot \boldsymbol{U} + b_M^{br})], \end{aligned} \tag{3.24}$$

where we have two vectors of random variables $\boldsymbol{\alpha}^{br}$ and $\boldsymbol{b}^{br}$, from which we sample $M$ realizations. The weights $\boldsymbol{\alpha}_i^{br}$ are i.i.d. sampled from a standard Gaussian distribution, and the biases $\boldsymbol{b}_i^{br}$ are uniformly distributed in $\mathcal{U}[0, 2\pi]$. This explicit random lifting has a low distortion for a Gaussian shift-invariant kernel distance.

The training of RandONets reduces to the solution of a linear least-squares problem in the unknowns $W$, i.e., the external weights of the weighted inner product as in Eq. (3.7). In what follows, before presenting the numerical implementation, we first present the treatment of aligned and unaligned data. The aligned data case refers to datasets where the training pairs are consistently organized, say in a grid, facilitating the learning process. In such scenarios, the network can more effectively learn the underlying nonlinear operator mappings due to the structured form of the data. On the other hand, training for unaligned data presents challenges compared to the aligned data case. In such scenarios, where the input and output pairs are not consistently organized, the network must learn to identify and map the complex relationships between disjoint datasets. This lack of alignment can make it more difficult for the network to capture the underlying nonlinear operator mappings accurately. In general, achieving high accuracy in this context often demands greater computational resources and more extensive hyperparameter tuning to ensure the network converges to an optimal solution.

**RandONets for aligned data.** Let us assume that the training dataset consists of $s$ sampled input functions at $m$ collocation/grid points. Thus, the input is included in a matrix $U \in \mathbb{R}^{m\times s}$. Let us also assume that the output function can be evaluated on a fixed grid of $n$ points $\boldsymbol{y}_k \in \mathbb{R}^d$, which are stored in a matrix $Y \in \mathbb{R}^{n\times d}$ (row-vector); $d$ is the dimension of the domain. In this case, we assume that for each input function $u$, we have function evaluations $v$ at the grid $Y$ stored in the matrix $V = \mathcal{F}[U] \in \mathbb{R}^{n\times s}$.

While this assumption may appear restrictive at a first glance (as for example some values in the matrix $V$ could be missing, or $Y$ can be nonuniform, and may change in time), nonetheless, for many problems in dynamical systems and numerical analysis, such as the numerical solution of PDEs, entails employing a fixed grid where the solution is sought. This is clearly the case in methods like FD or FEM-based numerical schemes without mesh adaptation. Additionally, even in cases where the grid is random or adaptive, there is still the opportunity to construct a "regular" output matrix $V$ through "routine" numerical interpolation of outputs on a fixed regular grid. Now, given that the data are aligned, following Eq. (3.7), we can solve the following linear system (double-sided) of $n \times s$ algebraic equations in $N \times M$ unknowns:

$$V = \mathcal{F}[U] = \boldsymbol{\varphi}_n^{tr}(Y; \boldsymbol{\alpha}^{tr})\, W\, \boldsymbol{\varphi}_m^{br}(U; \boldsymbol{\alpha}^{br}) = T(Y)\, W\, B(U). \tag{3.25}$$

Let us observe that –differently from a classical system of equations (e.g., $Ax = b$), here we have two matrices from the trunk and the branch features, that multiply the readout weights $W$ on both sides.

Although the number of unknowns and equations appears large due to the product, the convenient alignment of the data allows for effective operations that involve separate

and independent (pseudo-) inversion of the trunk/branch matrices $T(Y) \in \mathbb{R}^{n\times N}$ and $B(U) \in \mathbb{R}^{M\times s}$. Thus, the solution weights of Eq. (3.25), can be found by employing methods such as the Tikhonov regularization [216], tSVD, QR/LQ decomposition and COD[161] of the two matrices, as we will detail later, obtaining:

$$W = \big(T(Y)\big)^{\dagger} V \big(B(U)\big)^{\dagger}. \tag{3.26}$$

As one might expect, the trunk matrix typically features smaller dimensions compared to the branch matrix. This is because the branch matrix may involve numerous samples $s$ of functions (usually exceeding the number $n$ of points in the output grid), along with a higher number of neurons $M$ required to represent the high-dimensional function input, as opposed to the $N$ neurons of the RP-FNN trunk which embeds the input space. At the end, the computational cost associated with the training (i.e., the solution of the linear least-squares problem) of RandONets is of the order $O(M^2 s + s^2 M)$. Here, we use the COD algorithm [161] for the inversion of both $T$ and $B$ matrices (for a comparison of truncated SVD and COD algorithms for RP-FNN training, see in [156]).

**RandONets for unaligned data.** In contrast to the aligned data, the output $V$ cannot be usually stored in a matrix, but we have to consider a (long) vector. To address learning with unaligned data, it is sufficient to assume that for each input function $u$, the output $v(\boldsymbol{y})$ is available at a *single* random location in the output domain. This encompasses scenarios with a sparse random grid, where each output in the grid is treated separately, yet necessitating the introduction of multiple copies of the function $u$. Thus, let us assume we have stored the input functions in a matrix $U \in \mathbb{R}^{m\times S}$ and a vector of outputs $V \in \mathbb{R}^{1\times S}$.

Here, $S \in \mathbb{N}$ denotes both the total number of output functions and the total number of input functions. Unlike the aligned case, we store the random points for each input in the matrix $Y \in \mathbb{R}^{d\times S}$, where $d$ now represents the columns instead of the rows, and $S$ reflect the total number of (single) random locations where the individual outputs are sought.

Now, returning to Eq. (3.7), we notice that with the current format of inputs (both column-wise), we can express the output using the Hadamard (Shur) product ($\otimes$):

$$V = \sum_{k=1}^{N}\sum_{i=1}^{M} w_{ki} T_k(Y) B_i(U) = \sum_{k=1}^{n} T_k(Y) \otimes \boldsymbol{w}_k\, B(U), \tag{3.27}$$

where $\boldsymbol{w}_k$ are the rows of the matrix $W$. This corresponds to the original formulation of the DeepONet by Lu et al. (2021) [86], where instead of considering the merging of the branch and trunk networks as a weighted inner product, the focus is on the individual output at a single location, rather than treating the output as an entire transformed function. In this scenario, to solve the linear least-squares problem in terms of the $N \times M$ unknown weights $W$, we need to reshape the matrix $W$ into a row vector $\boldsymbol{\omega}$, where the elements $\omega_q = w_{ki}$, with $q = k + (i-1)n$. Then we construct the full collocation matrix $Z \in \mathbb{R}^{NM\times S}$, such that the rows $\boldsymbol{z}_q$ are obtained as:

$$\boldsymbol{z}_q = T_k \otimes B_i, \qquad q = k + (i-1)N. \tag{3.28}$$

Note that the Hadamard product $T_k \otimes B_i$ is possible as both lie in $\mathbb{R}^S$.

At this point, the weights $\boldsymbol{\omega}$ can be computed, through a (pseudo-) inversion of the matrix $Z$, in analogy to what was detailed in section 2.3, resulting in:

$$\boldsymbol{\omega} = Y\, Z^{\dagger}. \tag{3.29}$$

We note that the total number $S$ of single outputs can be viewed as proportional to the product of $s$ (the number of different input functions) and $N$ (the number of points in the output grid), as explained in the aligned case. In particular, employing an unaligned training algorithm for aligned data (by augmenting the input function with copies and reshaping the output) will result exactly in $S = Ns$. Now, the pseudo-inversion of the matrix $Z$ will result in a computational cost of the order $O\big((Ns)^2MN + (MN)^2Ns\big)$, which is significantly higher. For instance, in the case the values of $M$, $s$ are similar/proportional to $N$, we obtain a transition, in terms of computational complexity, from an order $O(N^3)$ for the aligned case to an order $O(N^6)$ for the unaligned case.

To this end, we argue that the unaligned approach described here should only be considered if the output data display substantial sparsity, suggesting that the random output grid does not adequately represent the output function. Conversely, we advocate for prioritizing the aligned approach in other scenarios. Even if it entails performing interpolation on a fixed grid to generate new aligned output data.

**Numerical implementation of the training of RandONets.** Below, we provide more details on the training process of RandONets. From a numerical point of view, the resulting random trunk $T(Y)$ and branch $B(U)$ matrices, tend to be ill-conditioned. Therefore, in practice, we suggest solving Eq. (3.25) as described in 2.3 via a truncated SVD (tSVD), Tikhonov regularization, QR decomposition or regularized COD [161, 156]. Here, we describe the procedure for the branch network. For the trunk matrix, the procedure is similar.

The regularized pseudo-inverse $(B(U))^{\dagger}$, for the solution of the problem in Eq. (3.26) is computed as:

$$B(U) = U\Sigma V^T = [U_r \quad \tilde{U}] \begin{bmatrix} \Sigma_k & 0 \\ 0 & \tilde{\Sigma} \end{bmatrix} [V_k \quad \tilde{V}]^T, \qquad (B(U))^{\dagger} = V_k \Sigma_k^{-1} U_r^T, \tag{3.30}$$

where the matrices $U = [U_k \quad \tilde{U}] \in \mathbb{R}^{k\times n} \oplus \mathbb{R}^{(n-k)\times n}$ and $V = [V_k \quad \tilde{V}] \in \mathbb{R}^{k\times s} \oplus \mathbb{R}^{(s-k)\times s}$ are orthogonal and $\Sigma \in \mathbb{R}^{n\times s}$ is a diagonal matrix containing the singular values $\sigma_i = \Sigma_{(i,i)}$. Here, we select the $k$ largest singular values exceeding a specified tolerance $0 < \epsilon \ll 1$, i.e., $\sigma_1, \ldots, \sigma_k > \epsilon$, effectively filtering out insignificant contributions and improving numerical stability.

## 3.5 Some numerical Examples of the RandONets

In this section, we present numerical results focusing exclusively on the aligned case within the framework of RandONets. This choice stems from (a) the observed superior computational cost of the proposed aligned approach; and (b) the aligned case fits well with the nature of the considered PDEs problems in dynamical systems and numerical analysis approaches. Here we report the Antiderivative problem, a benchmark problem originally addressed in the paper introducing DeepONet by Lu et al. (2021) [86]. Additionally, we focus on one other benchmark problem concerning the evolution operator (right-hand-side (RHS)) of PDEs, the viscous Burgers' PDE.

In [86], there is a discussion on how to generate the dataset of functions: they compare Gaussian Random Fields and other random parametrized orthogonal polynomial sets. Here, we decided to generate the input-output data functions without using precomputed datasets. We consider a random parametrized RP-FNN with 200 neurons and Gaussian RBFs combined with a few additional random polynomial terms.

$$u(t) = \boldsymbol{w} \exp\left(\boldsymbol{s}(t - \boldsymbol{c})^2\right) + a_0 + a_1\, t + a_2\, t^2, \tag{3.31}$$

where the parameters $\boldsymbol{w}, \boldsymbol{s}, \boldsymbol{c} \in \mathbb{R}^{200}$, $a_0, a_1, a_2 \in \mathbb{R}$ are all randomized to generate the dataset of input functions.

In all numerical examples, we select many different realizations of these functions. In some of these, in selecting the training datasets, we distinguish the case in which we utilize *limited-data* for training. In particular, we utilize $15\%$ of the data for the training set and $85\%$ for the test set. In the case in which we assume that we have available *extensive-data*, we utilize $80\%$ of the data for the training set and $20\%$ for the test set.

The range of the values of the parameters $(\boldsymbol{w}, \boldsymbol{s}, \boldsymbol{c}, a_0, a_1, a_2)$ is detailed for each case study. When possible, the output function is computed analytically. Otherwise, a well-established numerical method (with sufficiently small tolerances) is used to compute accurate solutions as the "ground truth". To represent both the input functions $u$ and output functions $v$, we use an equally spaced grid of 100 points in the domains $K_1$ and $K_2$ of interest. In particular, in all examples considered here, we take as input, one-dimensional domains (intervals in $[a, b]$).

Regarding the architecture of the RandONets, as also detailed in Section 3.4.2, we investigate and compare the performance of two different RandONets architectures, with two well-established embedding techniques, respectively: linear random Johnson-Lindenstrauss (JL) embeddings denoted by $\phi_M^{JL}$ (as presented in Eq. (3.23)) and RFFN embeddings denoted by $\phi_M^{RFFN}$ (as presented in Eq. (3.24)). These architectures will be subsequently referred to as RandONets-JL and RandONets-RFFN, respectively. We will explore the impact of varying the number of neurons ($M$) within the single hidden layer of the branch, effectively controlling the dimension of the branch embedding. For both RandONets-JL and RandONets-RFFN, the trunk embedding leverages a non-linear RP-FNN architecture denoted by $\phi_N^{tr}$ (as presented in (3.22)), which utilizes hyperbolic tangent activation functions $\psi$ and parsimoniously function-agnostic randomization of the internal weights (as described in [156, 95, 69]). Throughout the experiments, we will maintain a consistent number of neurons ($N = 200$) within the trunk's hidden layer,

thus ensuring a fixed size for the trunk embedding. It is important to note that the RandONets-JL architecture incorporates a combination of linear and non-linear embedding techniques, whereas the RandONets-RFFN architecture is entirely non-linear.

Given the big difference in the computational cost for the inversion of the branch matrix compared to the trunk matrix, we decided to fix the number of neurons $N = 200$ in the trunk RP-FNN embedding for both the RandONets-JL and RandONets-RFFN. The inversion of the corresponding trunk matrix $T \in \mathbb{R}^{100\times 200}$ thus it is, for any practical purposes, relatively negligible. Here, we investigate the performance of the scheme for $M = 10, 20, 40, 80, 100, 150, 300, 500, 1000, 2000$ neurons in the branch embedding of both RandONets-JL and RandONets-RFFN and the corresponding increment in computational cost.

**Metrics.** To assess the performance of the RandONets, we utilize both the MSE for the entire test set, as well the $L^2$–error for each output-function in the test set. In particular, we report the median $L^2$ and the percentiles $5\% - 95\%$. Importantly, we report the execution time in seconds of the scheme, thus indicating when the computations are performed with GPU or CPU.

**Remark on the DeepONets architectures used.** Given the high-computational cost associated when training DeepONets with the Adaptive Moment Estimation (Adam) algorithm, (even if we employ a GPU hardware), we do not focus now on performing a convergence diagram of the scheme or finding the best architecture. For our illustrations, we just selected a few configurations. In particular, we selected 2 hidden layers for both trunk and branch networks, each layer with a prescribed number $N = \{5, 10, 20, 40\}$ of neurons. Also, we employ hyperbolic tangent as activation functions for both branch and trunk networks. We will refer to the performance of these vanilla DeepONets in Tables with the notation $[N, N]$. We remark that the number of free trainable parameters $\zeta$ of such DeepONets configurations is $m \times N + 3N \times N + 5N$, (e.g., $N = 40, m = 100$, then $\zeta = 9000$) which is not higher than the biggest considered RandONets. Indeed, RandONets have a number of free trainable parameters equal to $N \times M$ (e.g., in the biggest case considered here, $N = 200$, $M = 2000$, it corresponds to $\zeta = 400,000$ parameters). However, as we will show, despite the high number of parameters, such RandONets can be trained in around one second. Finally, we also remark that when employing a gradient-descent based algorithm, like the Adam one, there is no guarantee of convergence, and the generalization of the network can be moderate. We anyway decided to train DeepONet for a fixed number of iterations equal to $20,000$ for ODE benchmarks and to $50,000$ for the PDEs.

**Remark on the hardware and software used.** In our experiments, we utilized the DeepONet framework implemented in Python with the DeepXde library [93], leveraging TensorFlow as the back-end for computations. These computations are executed on a GPU NVIDIA GeForce RTX 2060, harnessing its parallel processing capabilities to expedite training and evaluation. Additionally, we ran the Python code on `Google Colab` using a Tesla K80 GPU, resulting in computational times approximately 6 to 7 times slower than those reported in the main text. While we do not include these execution times in the main text, we mention them here as a reference. In contrast, the RandONets framework is implemented in MATLAB 2024a and executed on a single CPU of an Intel® Core™ i7-10750H CPU @ 2.60GHz, with 16 GB of RAM. The code is available at https://github.com/GianlucaFabiani/RandONets.

### 3.5.1 A simple (pedagogical) ODE benchmark problem

We start with a very simple benchmark problems involving non-autonomous ODEs subject to a time dependent source term $u(t)$, in the form:

$$\frac{dv(t)}{dt} = f(t,v) + u(t), \qquad v(0) = v_0, \quad t \in [0,T], \tag{3.32}$$

with some forcing time-dependent input function $u(t)$. The solution function $v(t)$ depends directly on the forcing term, the function $u(t)$. Thus, there exists an operator that maps $u(t)$ into $v(t)$. The task here, different from the one for the PDEs, is to learn the "solution operator" for one initial condition. Of course, learning the full solution operator would need a set of different initial conditions as in [86] but as mentioned these serve purely for pedagogical purposes.

#### *Case study 1: Antiderivative operator*

The first benchmark problem that we consider is the antiderivative operator[217], thus the solution $v(t)$ given the function $u(t)$ of the following (phase-independent, $f(t,v) \equiv 0$) ODE problem:

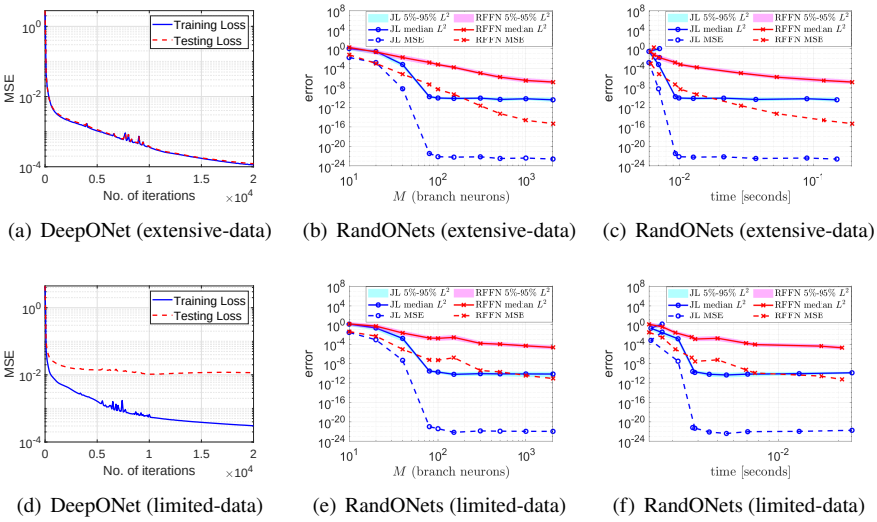
$$\frac{dv(t)}{dt} = u(t), \qquad v(0) = v_0, \quad t \in [0,1], \tag{3.33}$$

thus learning the linear operator $\mathcal{F}[u](t) = v(t)$. The corresponding analytical antiderivatives are:

$$v(t) = \boldsymbol{w}\left(-\sqrt{\pi}\,\mathrm{erfi}\big(\sqrt{\boldsymbol{s}}(\boldsymbol{c}-t)\big)/(2\sqrt{\boldsymbol{s}})\right) + a_0 t + a_1 t^2/2 + a_2 t^3/3 + C, \tag{3.34}$$

where erfi is the error function and $C$ is a constant that has to be fit by the initial condition $v(0) = v_0$. We select $v_0 = 0$ and we set $\tilde{v}(t) = v(t) - v(0)$ as the output function.

The values of the parameters $\boldsymbol{w}$, $a_0$, $a_1$, $a_2 \sim \mathcal{U}[-1,1]$, $\boldsymbol{s} \sim \mathcal{U}[0,500]$ and $\boldsymbol{c} \sim \mathcal{U}[0,1]$, of the RP-FNN based function dataset, as in Eq. (3.31), are (element-wise) sampled from the corresponding aforementioned uniform distributions. To generate the data, we used 1000 random realizations. We considered two different sizes of training




(a) DeepONet (extensive-data) (b) RandONets (extensive-data) (c) RandONets (extensive-data)

(d) DeepONet (limited-data) (e) RandONets (limited-data) (f) RandONets (limited-data)

Figure 3.2: Case study 1: Antiderivative Operator, in Eq. (3.33). (First row) extensive-data case, 800 training input functions; (Second row) limited-data case, 150 training input functions. (a), (d) MSE for the training and test sets, with DeepONets with 2 hidden layers (indicatively) with 40 neurons each, for both the branch and trunk networks. (b), (c), (e), (f) MSE and $L^2$ error, $5\% - 95\%$ range and median, of the RandONets, for different size $M$ of the branch embedding. The errors are computed w.r.t only the output functions in the test dataset. Comparison of Johnson-Lindenstrauss (JL) branch embedding with RFFN embeddings. We set the size of the Trunk network to $N = 200$ and the grid of input points to $m = 100$. Numerical approximation accuracy vs. (b)-(e) the number of neurons $M$ in the hidden layer of the branch network; and (c)-(f) vs. computational times in seconds.

sets. In particular, we used $15\%$ for the training and $85\%$ for the test set (we remind the reader that we call this limited-data case). As described above, for the extensive data-case, we used $80\%$ for the training and $20\%$ for the test set.

In Figure 3.2, we depict the numerical approximation accuracy for the test set in terms of the MSE and percentiles median, $5\% - 95\%$ of $L^2$–errors. As shown, the training of all RandONets takes approximately less than 1 second and is performed without iterations. In Table 3.1, we summarize the comparison results with the vanilla DeepONet in terms of the best accuracy and best computational times. For the RandONets, we used 100 neurons for the JL embedding, and 2000 neurons for the RFFN embedding for the branch network. As shown, the JL-based RandONets gets an astonishing almost machine-precision accuracy of $MSE \simeq 1E\text{-}23$, $L^2 \simeq 1E\text{-}11$ with just 40 neurons in the branch with a computational time of $\simeq 0.01$ seconds. Such "perfect" results are due to the simplicity of the problem and its linearity. The nonlinear RFFN embedding result in a lower performance with respect to the JL RandONets, for this linear problem, obtaining

Table 3.1: Case study 1: Antiderivative Operator in Eq. (3.33). We report Mean Squared Error (MSE), percentiles (median, $5\%$, $95\%$ of $L^2$ error across the test set. The extensive-data case comprises $800$ training functions, while the limited-data case uses $150$ functions as training. We employed vanilla DeepONets with 2 hidden layers, denoted as $[N, N]$ neurons, for both the branch and the trunk. We set $N = 5, 10, 20, 40$. DeepONets are trained with $20,000$ Adam iterations (with learning rate $0.001$ and then $0.0001$). We report the RandONets encompassing Johnson-Lindenstrauss (JL) Featured branch network (with $M = 100$ neurons) as well as the RFFN branch (with $M = 2000$ neurons).

| data | ML-model | MSE | $5\%$ $L^2$ | median–$L^2$ | $95\%$ $L^2$ | comp. time |
|---|---|---|---|---|---|---|
| 80% | DeepONet $[5, 5]$ | 9.83E−01 | 2.31E+00 | 6.90E+00 | 1.80E+01 | 4.75E+02 (GPU) |
| | DeepONet $[10, 10]$ | 2.28E−03 | 2.43E−01 | 4.37E−01 | 7.46E−01 | 4.62E+02 (GPU) |
| | DeepONet $[20, 20]$ | 4.39E−04 | 1.26E−01 | 2.00E−01 | 2.97E−01 | 4.79E+02 (GPU) |
| | DeepONet $[40, 40]$ | 1.22E−04 | 7.04E−02 | 1.03E−01 | 1.60E−01 | 5.23E+02 (GPU) |
| | RandONets–JL (100) | 9.43E−23 | 4.33E−11 | 8.01E−11 | 1.68E−10 | 1.02E−02 (CPU) |
| | RandONets–RFFN (2000) | 8.09E−16 | 6.81E−08 | 1.73E−07 | 5.99E−07 | 1.96E−01 (CPU) |
| 15% | DeepONet $[5, 5]$ | 8.88E−02 | 1.08E+00 | 2.48E+00 | 5.15E+00 | 1.05E+02 (GPU) |
| | DeepONet $[10, 10]$ | 2.99E−03 | 2.73E−01 | 4.91E−01 | 8.51E−01 | 9.78E+01 (GPU) |
| | DeepONet $[20, 20]$ | 7.48E−04 | 1.51E−01 | 2.54E−01 | 4.07E−01 | 1.13E+02 (GPU) |
| | DeepONet $[40, 40]$ | 1.16E−02 | 2.57E−01 | 7.36E−01 | 2.12E+00 | 1.24E+02 (GPU) |
| | RandONets–JL (100) | 1.66E−21 | 2.22E−10 | 3.74E−10 | 6.11E−10 | 3.60E−03 (CPU) |
| | RandONets–RFFN (2000) | 8.12E−12 | 8.14E−06 | 2.03E−05 | 5.25E−05 | 1.88E−02 (CPU) |

an $MSE \simeq 1E$-16 and a median $L^2 \simeq 1E$-08, using 2000 neurons in the branch, with a computational time of less of the order $0.1$ seconds. We employ vanilla DeepONets with two hidden layers, denoted as $[N, N]$ neurons, in both Trunk and Branch. We observe a rather slow convergence in accuracy by increasing the size of the Vanilla DeepONets. However, the vanilla DeepONets need many iterations to reach an adequate accuracy. After $20,000$ iterations, the accuracy in the extensive-data case, for $N = 40$, is around $1E$-04 in terms of MSE, but the $L^2$ error is on the order of $1E$-01. In the limited-data case, the vanilla DeepONet, with $N = 40$, gives a rather poor performance: the MSE on test data is stuck at $1E$-02 thus, overfitting. The corresponding $L^2$ error is on the order of 1. Indeed, the DeepONet with $N = 20$ performs better. We can explain such failure due to difficult dataset considered, that needs sufficient input-output functions to be well represented.

Our results for the two hidden layers vanilla DeepONet are in line with the ones presented in [86]. Also, for investigations on different architectures, one can refer to the same paper. In particular, there, for the vanilla DeepONet with $4$ hidden layers and $[2560, 2560, 2560, 2560]$ neurons in both trunk and branch, they report an MSE of around $1E$-08 after $50,000$ iterations.

As a matter of fact, for this case study, the execution times (training times) for RandONet, utilizing both linear JL and nonlinear RP-FNN random embeddings, are 10,000 times faster, while achieving $L^2$ accuracy that is 6 to 10 orders of magnitude higher.

### 3.5.2 Approximation of Evolution Operators (RHS) of time-dependent PDEs

Here, we consider a benchmark problem relative to the identification of the evolution operator, i.e., the right-hand-side (RHS) of time-dependent PDEs:

$$v(x) = \frac{\partial u(x,t)}{\partial t} = \mathcal{L}(u)(x,t). \tag{3.35}$$

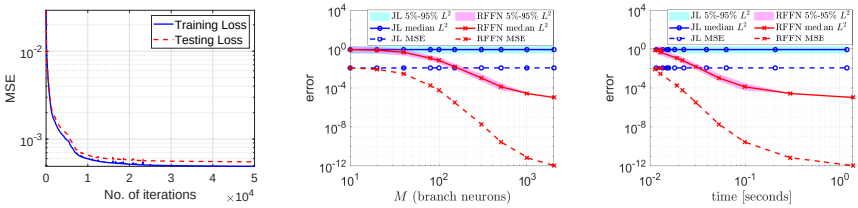
The output function, the time derivative, (i.e., the right-hand-side of the evolutionary PDE) depends on the current state profile $u(x,t)$ at a certain time $t$. In the following examples, we do not consider the limited-data case.

#### *Case study 2: 1D viscous Burgers PDE*

We consider the nonlinear evolution operator of the Burgers' equation, given by:

$$v = \frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2} - u \frac{\partial u}{\partial x}, \tag{3.36}$$

where $\nu = 0.01$.




(a) DeepONet (b) RandONets (c) RandONets

Figure 3.3: Case study 2: 1D nonlinear Burgers' PDE (Eq. (3.36)). We used 1600 training input functions: (a) MSE when using a vanilla DeepONet with 2 hidden layers with (indicatively) 40 neurons each, for both branch and trunk networks. (b), (c), MSE and $L^2$ error percentiles (median, $5\% - 95\%$), of the RandONets for different size $M$ of the branch embedding. Comparison of Johnson-Lindenstrauss random embeddings, as in Eq. (3.23), with RFFN embeddings, as in Eq. (3.24). We have set the size of the trunk network to $N = 200$ and the grid of input points to $m = 100$. Numerical approximation accuracy vs. (b) number of neurons $M$ in the hidden layer of the branch network; and (c) vs. computational time in seconds.

The output function can be computed analytically/symbolically based on (3.31). The parameters $\boldsymbol{w}$, $a_0$, $a_1$, $a_2 \sim \mathcal{U}[-0.05, 0.05]$, $\boldsymbol{s} \sim \mathcal{U}[0, 50]$ and $\boldsymbol{c} \sim \mathcal{U}[-1, 1]$, of the RP-FNN based function dataset, as in Eq. (3.31), to represent the functional space are (element-wise) uniformly distributed. Here we select $w$ in a smaller range, as higher values may correspond to high second derivatives approaching singularity.

Table 3.2: Case study 2: Burgers' Nonlinear PDE in Eq. (3.36). We report Mean Squared Error (MSE) and percentiles (median, $5\% - 95\%$) of $L^2$ approximation errors, for the test set. We use $1600$ training functions. We employ DeepONets with $2$ hidden layers with $[N, N]$ neurons in both the branch and trunk. We set $N = 5, 10, 20, 40$. DeepONets are trained with $50,000$ Adam iterations (with learning rate $0.001$ and then $0.0001$). We report the RandONets results encompassing Johnson-Lindenstrauss (JL) Featured branch network (with $M = 40$ neurons) and the RFFN branch network (with $M = 2000$ neurons).

| ML-model | MSE | $5\%$ $L^2$ | median–$L^2$ | $95\%$ $L^2$ | comp. time |
|---|---|---|---|---|---|
| DeepONet $[5, 5]$ | 9.00E−03 | 3.70E−01 | 7.60E−01 | 1.66E+00 | 2.16E+03 (GPU) |
| DeepONet $[10, 10]$ | 4.75E−03 | 3.43E−01 | 5.59E−01 | 1.20E+00 | 2.01E+03 (GPU) |
| DeepONet $[20, 20]$ | 1.51E−03 | 2.24E−01 | 3.28E−01 | 6.16E−01 | 2.40E+03 (GPU) |
| DeepONet $[40, 40]$ | 5.50E−04 | 1.30E−01 | 2.03E−01 | 3.82E−01 | 2.34E+03 (GPU) |
| RandONets–JL (40) | 1.09E−02 | 3.32E−01 | 8.11E−01 | 1.91E+00 | 1.29E−02 (CPU) |
| RandONets–RFFN (2000) | 1.12E−12 | 1.01E−05 | 1.04E−05 | 1.19E−05 | 1.51E+00 (CPU) |

To generate the data, we used $2000$ random realizations of the parameters. We set $80\%$ for training and $20\%$ for testing. In Figure 3.3, we report the accuracy w.r.t. the test set in terms of the MSE and the median (and percentiles $5\% - 95\%$) of $L^2$–errors. As shown, the training of all RandONets takes approximately less or around one second. In Table 3.2, we also report the comparison results in terms of the numerical approximation accuracy and computational times.

Due to the inherent non-linearity of this example, linear JL random embeddings exhibit limitations in efficiently approximating the non-linear operator. This observation aligns with the theoretical understanding of JL embeddings being most effective in capturing linear relationships. Unlike case study 2, the non-linearity within only the trunk architecture appears insufficient for this specific problem. Therefore, incorporating non-linearity also in the branch embedding becomes crucial for achieving optimal performance.

Interestingly, the performance of JL embeddings approaches that of fully trained, entirely non-linear vanilla DeepONets. While DeepONets can achieve a minimum Mean Squared Error (MSE) on the order of 1E−04 and an $L^2$ error on the order of 1E−01, their performance is not significantly better than the JL approach. In contrast, the RandONets-RFFN architecture emerges as the clear leader in this specific case study. It achieves a remarkably low MSE on the order of 1E−12, demonstrating its superior capability in handling the non-linearities present in this example.

Also for this case study, RandONets, utilizing both JL and RP-FFN random embeddings, demonstrates execution times (training times) that are $3$ to $5$ order times faster, while achieving $L^2$ accuracy that is $4$ orders of magnitude higher in the case of RandONets-RFFN, and of a similar level of accuracy in the case of JL random embeddings.

## 3.6 Discussion

The results presented in this chapter represent significant advances in both RPNNs and RandONets, with important implications for scientific ML, especially in the context of numerical analysis and complex systems modeling.

**RPNNs and Best Approximation.** Our work on RPNNs has established a strong theoretical foundation by proving exponential convergence for smooth functions, inspired by the best approximation polynomials in $L^p$ spaces. This convergence suggests that RPNNs are a promising alternative to traditional methods for function approximation, offering robust and efficient solutions without the iterative training required by fully connected neural networks (FNNs). While classical FNNs can theoretically match or exceed the performance of RPNNs, they are often constrained by the limitations of current optimization algorithms, which struggle to exceed four-digit precision [149].

The practical results further demonstrate that RPNNs can effectively approximate a wide variety of functions, including those with steep gradients, high oscillations, and near-discontinuities. However, certain limitations were observed, especially when dealing with functions that exhibit rapid oscillatory or divergent behavior. Addressing these edge cases through more sophisticated basis function selection and enhanced regularization techniques presents a clear direction for future research.

These findings open the way to new avenues in ML, where RPNNs could serve as a key component in solving forward and inverse problems. Their rapid convergence and computational efficiency make them especially suited for applications where computational cost is a concern, such as real-time modeling or large-scale simulations in complex systems. Additionally, their success in approximating low-dimensional functions suggests that extensions to higher-dimensional problems, though currently challenging, could yield fruitful results. Investigating function-informed basis selection strategies for such cases would be a logical next step.

**RandONets for Operator Learning.** The introduction of RandONets as a tool for operator learning further extends the utility of random projection-based methods. Our work builds on three keystones: (a) random embeddings [199, 160, 158, 203] for nonlinear random embeddings, (b) one-hidden layer (shallow) RPNNs whose "birth" can be traced back to early 90s [195, 194, 153], and, (c) tailor-made numerical analysis techniques for the solution of a linear ill-posed problem. First, based on the above, we prove the universal approximation property of RandONets. By leveraging nonlinear low-distortion random embeddings, we have shown that RandONets can achieve remarkable accuracy and efficiency when approximating both linear and nonlinear operators, such as the solution operators of ODEs and PDEs. In particular, RandONets outperform vanilla DeepONets by several orders of magnitude in both computational speed and accuracy for the benchmark problems considered. This is especially notable for linear operators, where RandONets with JL embeddings nearly achieve machine-precision accuracy.

The performance boost observed with nonlinear operators, such as in the Burgers's PDEs, highlights the potential for RandONets to address more challenging problems

in operator learning, where traditional deep learning methods like DeepONets are less efficient. RandONets' superior performance can be attributed to their ability to efficiently leverage random embeddings, which offer a compact and effective representation of the underlying operator without the need for deep architectures or costly training procedures.

**Comparison with Related Work and Relevance of the Presented Results.** To the best of our knowledge, the earliest approach that utilized random sampling and fixed weights for training DeepONet is by Bolager et al. [181]. Their method draws inspiration from geometric random sampling techniques, as also implemented in [45], a part of this thesis.

In particular in [181], the authors investigated random sampling for various deep learning frameworks, including a POD-based DeepONet where only the branch network is sampled, while the trunk is not considered. In contrast, our work is the first to propose a *double randomization* strategy for both the branch and trunk networks. This is achieved through a single-shot approach with straightforward random embeddings, which has proven to be computationally highly efficient. Specifically, as said, our method achieves up to five orders of magnitude faster training than vanilla fully-trained DeepONet, enabling training in under one second while achieving up to 10 orders of magnitude improvement in $L^2$-norm accuracy (and 20 orders of magnitude in mean squared error, MSE). Such big improvement sets the state-of-the-art in training DeepONets.

Subsequently, Lee and Shin [218] independently explored Least-Squares training for the branch and trunk layers. However, their approach relies on inefficient sampling methods that require additional optimization through iterative line searches and iterative two-steps alternate least-squares to refine the embedding. Specifically, their technique employs a prescribed trunk QR-based pseudo-inverse and iterative QR-Least-Squares optimization for the branch layer. While their method demonstrates potential for improving suboptimal random embeddings, it incurs significant computational overhead, as each training epoch becomes more resource-intensive. The combination of efficient random projections/samplings [181, 157, 160, 158, 199] and their algorithm has not yet been investigated in literature.

Another relevant contribution is the methodology proposed by Nelsen and Stuart [219], which adopts a spectral approach leveraging Random Feature Maps. While innovative, their framework has practical limitations. For example, in their work, the random feature map for the Darcy equation is constructed with full knowledge of the underlying physics. Consequently, this leads to a neural operator whose naive evaluation can be as computationally expensive as solving the original PDE.

Nelsen and Stuart argue that RFMs enable resolution invariance when paired with techniques like the Fast Fourier Transform (FFT). However, this resolution invariance is contingent upon highly refined discretizations, often requiring dense grids to maintain accuracy. For sparse grids or domains with complex geometries, their method becomes less effective, as its performance heavily depends on grid resolution and Fourier coefficient fitting.

In contrast, DeepONets and RandONets utilize are built on a straightforward discretization grid, supported by universal approximation theorems. For grids that differ from the desired configuration, interpolation techniques allow for adaptation, achieving

effective resolution invariance. While interpolation introduces minor errors, such errors are present in all methods involving mismatched discretizations. Unlike the approach of Nelsen and Stuart, our framework avoids dependencies on Fourier basis fitting across resolutions and the associated challenges with non-equally spaced grids.

**Broader Implications and Future Directions.** The broader implication of this work is that RandONets can serve as versatile building blocks in the ML toolkit for both forward and inverse problems. Their theoretical guarantees, combined with the demonstrated practical efficiency, suggest that these architectures can be highly impactful in complex systems modeling. For example, RandONets could be employed in learning the right-hand sides of macroscopic laws in ABMs or in discovering coarse-grained PDE representations from microscopic simulations.

Moreover, the rapid convergence and computational simplicity of these methods offer an appealing alternative to more complex neural network architectures, such as deep learning or convolutional neural networks, in scenarios where interpretability, speed, and sustainability are critical. RandONets, in particular, present an exciting opportunity to extend the frontier of operator learning, making it more accessible and scalable, especially in high-dimensional or computationally expensive problems.

Looking ahead, the potential to further refine these methods lies in the exploration of function- and operator-informed basis function selection strategies, improved regularization techniques, and the development of algorithms capable of handling high-dimensional problems. In addition, a comprehensive comparison with other state-of-the-art methods, including advanced DeepONets and Fourier Neural Operators (FNOs), will provide a clearer picture of the relative strengths and weaknesses of each approach. By continuing to develop these techniques, we aim to unlock new capabilities in ML for scientific applications, ultimately improving our ability to model and understand complex systems.

We believe that our work will trigger further advances in the field, paving the way for further exploration of how numerical analysis and concepts from the numerical bifurcation theory, can enhance the applicability and interpretability of shallow neural networks in several ways, potentially allowing them to outperform DNNs both in accuracy and sustainability for specific tasks of scientific ML.

# 4 Solution of the Forward Problems I: bifurcations of nonlinear stationary PDEs

This chapter and the next one examine the use of RPNNs to address various stationary and time-dependent PDEs, as well as stiff ODEs and DAEs. We will explore the integration of classical numerical techniques with RPNNs, emphasizing the benefits of hybrid approaches in improving stability and convergence. A central theme of this chapter is the importance of exploiting concepts from numerical analysis and dynamical systems' theory when applying ML techniques to complex problems. To paraphrase a well-known saying, while ML can be effective on its own, its potential is greatly enhanced when combined with established numerical methods.

Additionally, we provide a comprehensive analysis of constructing bifurcation diagrams for PDE solutions using RPNNs, which offers new insights into their application in complex dynamical systems.

## 4.1 Overview of the forward problem in complex systems

The solution of forward problems in the context of DEs – both PDEs and ODEs – poses several significant challenges, particularly when addressing issues such as stiffness, complex geometries, and high-dimensionality. These challenges are particularly pronounced in applications where rapid solutions are crucial, as delays can lead to suboptimal decision-making. For instance, in control systems, timely computations are essential for real-time adjustments. Similarly, in fields like climate modeling or financial forecasting, the need for immediate insights can be critical for effective responses to rapidly changing conditions.

While Physics-Informed Neural Networks (PINNs) have shown potential in solving DEs by embedding physical laws into neural networks, their training phase is computationally demanding and slow, making them impractical for real-time or large-scale applications. This performance bottleneck limits their usability in scenarios that require quick and efficient solutions. Here, we aim to improve these approaches, challenging the

already established flagship solvers of numerical analysis, to make them more viable for practical, time-sensitive tasks.

Forward problems are inherently connected to complex system simulations, where understanding the macroscopic emergent behavior often necessitates thorough analysis of both stationary solutions and bifurcation diagrams. This analysis is vital for making predictions and studying rare events, which can be critical in applications such as epidemiology or disaster management. Accurate solutions are paramount; poor solvers can yield inaccurate results that distort our understanding of the underlying models. For example, if a system is characterized by stiffness and we employ inadequate numerical methods, we may misinterpret the behavior we have learned through ML techniques.

In high-dimensional, multiscale complex systems, such as those modeled by ABM frameworks, the computational demands can escalate dramatically. Simulating millions of ODEs, often tackled with simple methods like Euler's, can become untenable if stiffness is present. In such cases, the inability to solve these equations accurately and efficiently can hinder our ability to draw meaningful conclusions from our models, emphasizing the necessity for robust, fast, and precise solvers. Thus, advancing the methodologies for solving forward problems in DEs is critical not only for theoretical advancements but also for practical applications across various domains.

## 4.2 Numerical solution and bifurcation analysis of Nonlinear Partial Differential Equations

In this section, we introduce the general setting for the numerical solution and bifurcation analysis of the general class of nonlinear PDEs with RPNNs based on basic numerical analysis concepts and tools (see, e.g., [220, 221, 222, 106, 223]). Let's start from a nonlinear PDE of the general form:

$$Lu = f(u, \lambda) \text{ in } \Omega, \tag{4.1}$$

with boundary conditions:

$$B_l u = g_l, \text{ in } \partial\Omega_l \,, \; l = 1, 2, \ldots, m \,, \tag{4.2}$$

where $L$ is the partial differential operator acting on $u$, $f(u, \lambda)$ is a nonlinear function of $u$ and $\lambda \in \mathbb{R}^p$ is the vector of model parameters, and $\{\partial\Omega_l\}_l$ denotes a partition of the boundary.

A numerical solution $\tilde{u} = \tilde{u}(\lambda)$ to the above problem at particular values of the parameters $\lambda$ is typically found iteratively by applying e.g., Newton-Raphson or matrix-free Krylov-subspace methods (Newton-GMRES) (see, e.g., [224]) on a finite system of $M$ nonlinear algebraic equations. In general, these equations reflect some zero residual condition, or exactness equation, and thus the numerical solution that is sought is the optimal solution w.r.t. the condition in the finite dimensional space. Assuming that $\tilde{u}$ is fixed via the degrees of freedom $\boldsymbol{w} \in \mathbb{R}^N$ – we use the notation $\tilde{u} = \tilde{u}(\boldsymbol{w})$ — then these degrees of freedom are sought by solving:

$$F_k(w_1, w_2, \ldots w_j \ldots w_N; \lambda) = 0 \,, \; k = 1, 2, \ldots M \,. \tag{4.3}$$

Many methods for the numerical solution of Eq. (4.1), (4.2) are written in the above form after the application of an approximation and discretization technique such as FD, FEM and (Pseudo-) Spectral Expansion, as we detail next.
The system of $M$ algebraic equations (4.3) is solved iteratively (e.g., by Newton's method), that is by solving until a convergence criterion is satisfied, the following linearized system:

$$\nabla_{\boldsymbol{w}} \boldsymbol{F}(\boldsymbol{w}^{(n)}, \lambda) \cdot d\boldsymbol{w}^{(n)} = -\boldsymbol{F}(\boldsymbol{w}^{(n)}, \lambda), \quad \boldsymbol{w}^{(n+1)} = \boldsymbol{w}^{(n)} + d\boldsymbol{w}^{(n)}. \tag{4.4}$$

$\nabla_{\boldsymbol{w}} \boldsymbol{F}$ is the Jacobian matrix:

$$\nabla_{\boldsymbol{w}} \boldsymbol{F}(\boldsymbol{w}^{(n)}, \lambda) = \left[\frac{\partial F_k}{\partial w_j}\right]_{|(\boldsymbol{w}^{(n)}, \lambda)} = \begin{bmatrix} \frac{\partial F_1}{\partial w_1} & \frac{\partial F_1}{\partial w_2} & \cdots & \frac{\partial F_1}{\partial w_j} & \cdots & \partial \frac{F_1}{\partial w_N} \\ \frac{\partial F_2}{\partial w_1} & \frac{\partial F_2}{\partial w_2} & \cdots & \frac{\partial F_2}{\partial w_j} & \cdots & \frac{\partial F_2}{\partial w_N} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{\partial F_k}{\partial w_1} & \frac{\partial F_k}{\partial w_2} & \cdots & \frac{\partial F_k}{\partial w_j} & \cdots & \frac{\partial F_k}{\partial w_N} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{\partial F_M}{\partial w_1} & \frac{\partial F_M}{\partial w_2} & \cdots & \frac{\partial F_M}{\partial w_j} & \cdots & \frac{\partial F_M}{\partial w_N} \end{bmatrix} \tag{4.5}$$

If the system is not square (i.e., when $M \neq N$), then at each iteration, one would perform, e.g., QR-factorization of the Jacobian matrix

$$\nabla_{\boldsymbol{w}} \boldsymbol{F}(\boldsymbol{w}^{(n)}, \lambda) = R^T Q^T = \begin{bmatrix} R_1^T & 0 \end{bmatrix} \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix}, \tag{4.6}$$

where $Q \in \mathbb{R}^{N \times N}$ is an orthogonal matrix and $R \in \mathbb{R}^{N \times M}$ is an upper triangular matrix. Then, the solution of Eq.(4.4) is given by:

$$d\boldsymbol{w}^{(n)} = -Q_1 R_1^{-1} \cdot \boldsymbol{F}(\boldsymbol{w}^{(n)}, \lambda).$$

Branches of solutions in the parameter space past critical points on which the Jacobian matrix $\nabla F$ with elements $\dfrac{\partial F_k}{\partial w_j}$ becomes singular can be traced with the aid of numerical bifurcation analysis theory (see, e.g., [225, 223, 226]). For example, solution branches past saddle-node bifurcations (limit/turning points) can be traced by applying the so called "pseudo" arc-length continuation method [221]. This involves the parametrization of both $\tilde{u}(\boldsymbol{w})$ and $\lambda$ by the arc-length $s$ on the solution branch. The solution is sought in terms of both $\tilde{u}(\boldsymbol{w}; s)$ and $\lambda(s)$ in an iterative manner, by solving until convergence the following augmented system:

$$\begin{bmatrix} \nabla_{\boldsymbol{w}} \boldsymbol{F} & \nabla_{\lambda} \boldsymbol{F} \\ \nabla_{\boldsymbol{w}} N & \nabla_{\lambda} N \end{bmatrix} \cdot \begin{bmatrix} d\boldsymbol{w}^{(n)}(s) \\ d\lambda^{(n)}(s) \end{bmatrix} = - \begin{bmatrix} \boldsymbol{F}(\boldsymbol{w}^{(n)}(s), \lambda(s)) \\ N(\tilde{u}(\boldsymbol{w}^{(n)}; s), \lambda^{(n)}(s)) \end{bmatrix}, \tag{4.7}$$

where

$$\nabla_{\lambda} \boldsymbol{F} = \begin{bmatrix} \frac{\partial F_1}{\partial \lambda} & \frac{\partial F_2}{\partial \lambda} & \cdots & \frac{F_M}{\partial \lambda} \end{bmatrix}^T, \tag{4.8}$$

and

$$N(\tilde{u}(\boldsymbol{w}^{(n)};s),\lambda^{(n)}(s)) = (\tilde{u}(\boldsymbol{w}^{(n)};s) - \tilde{u}(\boldsymbol{w};s)_{-2})^T \cdot \frac{(\tilde{u}(\boldsymbol{w})_{-2} - \tilde{u}(\boldsymbol{w})_{-1})}{ds} + (\lambda^{(n)}(s) - \lambda_{-2}) \cdot \frac{(\lambda_{-2} - \lambda_{-1})}{ds} - ds, \tag{4.9}$$

is one of the choices for the so-called "pseudo" arc-length condition (for more details see, e.g., [221, 222, 225, 223, 226]); $\tilde{u}(\boldsymbol{w})_{-2}$ and $\tilde{u}(\boldsymbol{w})_{-1}$ are two already found consequent solutions for $\lambda_{-2}$ and $\lambda_{-1}$, respectively and $ds$ is the arc-length step for which a new solution around the previous solution $(\tilde{u}(\boldsymbol{w})_{-2}, \lambda_{-2})$ along the arc-length of the solution branch is being sought. Additional information on numerical continuation can be found in Appendix B.3.1.

### 4.2.1 Finite Differences and Finite Elements cases: the application of Newton's method

In FD methods, one aims to find the values of the solution per se (i.e., $u_j = w_j$) at a finite number of nodes within the domain. The operator in the differential problem (4.1) and the boundary conditions (4.2) are approximated by means of some FD operator: $L^h \approx L$; $B_l^h \approx B_l$: the finite operator reveals in some linear combination of the function evaluations for the differential part, while keeping non-linear requirement to be satisfied due to the presence of nonlinearities. Then, approximated equations are collocated in internal and boundary points $\boldsymbol{x}_k$, giving equations that can be written as residual equations (4.3).

With FEM and SE methods, the aim is to find the coefficients of a properly chosen basis function expansion of the solution within the domain such that the boundary conditions are satisfied precisely. In the Galerkin-FEM with Lagrangian basis (see, e.g., [102, 106]), the discrete counterpart seeks for a solution of Eq. (4.1)-(4.2) in $N$ points $x_j$ of the domain $\Omega$ according to:

$$u = \sum_{j=1}^{N} w_j \phi_j, \tag{4.10}$$

where the basis functions $\phi_j$ are defined so that they satisfy the completeness requirement and are such that $\phi_j(x_k) = \delta_{jk}$. This, again with the choice of nodal variables to be the function approximation at the points, gives that $u(x_j) = w_j$ are exactly the degrees of freedom for the method. Then, the numerical approximation of the solution is achieved by setting zero the weighted residuals $R_k, k = 1, 2, \ldots N$ defined as:

$$R_k = \int_\Omega (Lu - f(u,\lambda))\phi_k \, d\Omega + \sum_{l=1}^{m} \int_{\partial\Omega_k} (B_k u - g_l)\phi_l \, d\sigma, \tag{4.11}$$

where the weighting functions $\phi_i$ are the same basis functions used in Eq. (4.10) for the approximation of $u$. The above constitutes a nonlinear system of $N$ algebraic equations

that for a given set of values for $\lambda$ are solved by Newton-Raphson, thus solving until convergence the following linearized system seen in Eq. (4.4), where $R_k$ plays the role of $F_k$.

Note that the border rows and columns of the Jacobian matrix (4.5) are appropriately changed so that Eq. (4.4) satisfy the boundary conditions. Due to the construction of the basis functions, the Jacobian matrix is sparse, thus allowing the significant reduction of the computation cost for the solution of (4.4) at each Newton's iteration.

### 4.2.2 RPNNs Collocation: the application of Newton's method

In an analogous manner to FE methods, a physics-indormed RPNN-based approach aim at solving the problem (4.1)-(4.2), using an approximation $\tilde{u}_N$ of $u$ with $N$ neurons as an ansatz. The difference is that, similarly to FD methods, the equations are constructed by collocating the solution on $M_\Omega$ points $\boldsymbol{x}_i \in \Omega$ and $M_l$ points $\boldsymbol{x}_k \in \partial\Omega_l$, where $\Omega_l$ are the parts of the boundary where boundary conditions are posed, see, e.g., [106]:

$$\begin{aligned} L\tilde{u}_N(\boldsymbol{x}_i;\boldsymbol{w}) &= f(\tilde{u}_N(\boldsymbol{x}_i;\boldsymbol{w}),\lambda), \quad i = 1,\dots,M_\Omega \\ B_l\tilde{u}_N(\boldsymbol{x}_k;\boldsymbol{w}) &= g_l(\boldsymbol{x}_k), \quad k = 1,\dots,M_l, \quad l = 1,\dots,m. \end{aligned}$$

Then, if we denote $M = M_\Omega + \sum_{l=1}^{m} M_l$, we have a system of $M$ nonlinear equations with $N$ unknowns that can be rewritten in a compact way as:

$$F_k(\boldsymbol{w},\lambda) = 0, \quad k = 1,\dots,M,$$

where for $k = 1,\dots,M_\Omega$, we have:

$$F_k(\boldsymbol{w},\lambda) = L\left(\sum_{i=1}^{N} w_j\psi(\boldsymbol{\alpha}_j\cdot\boldsymbol{x}_i+\beta_j)\right) - f\left(\sum_{i=1}^{N} w_j\psi(\boldsymbol{\alpha}_j\cdot\boldsymbol{x}_i+\beta_j)\right) = 0,$$

while for the $l$-th boundary condition, for $k = 1,\dots,M_l$ we have:

$$F_k(\boldsymbol{w},\lambda) = B_l\left(\sum_{i=1}^{N} w_j\psi(\boldsymbol{\alpha}_j\cdot\boldsymbol{x}_i+\beta_j)\right) - g\left(\sum_{i=1}^{N} w_j\psi(\boldsymbol{\alpha}_j\cdot\boldsymbol{x}_i+\beta_j)\right) = 0.$$

At this system of non-linear algebraic equations, here we apply Newton's method (4.4). Notice that the application of the method requires the explicit knowledge of the derivatives of the functions $\psi$; in the RPNN case as described, we have explicit formulae for these (see Eq. (2.94), (4.15)).

**Remark 4.2.1.** In our case, Newton's method is applied to non-squared systems. When the rank of the Jacobian is small, here we have chosen to solve the problem with the use of Moore–Penrose pseudo inverse of $\nabla_{\boldsymbol{w}}F$ computed by the SVD decomposition; as discussed above, another choice would be $QR$-decomposition (4.6). This means that we cut off all the eigenvectors correlated to small eigenvalues[1], so:

$$\nabla_{\boldsymbol{w}}\boldsymbol{F} = U\Sigma V^T, \qquad (\nabla_{\boldsymbol{w}}\boldsymbol{F})^+ = V\Sigma^+U^T,$$

[1]The usual algorithm implemented in Matlab is that any singular value less than a tolerance is treated as zero: by default, this tolerance is set to max(size($A$)) * eps(norm($A$))

where $U \in \mathbb{R}^{M\times M}$ and $V \in \mathbb{R}^{N\times N}$ are the unitary matrices of left and right eigenvectors respectively, and $\Sigma \in \mathbb{R}^{M\times N}$ is the diagonal matrix of singular values. Finally, we can select $q \leq rank(\nabla F)$ to get:

$$\nabla_{\boldsymbol{w}} \boldsymbol{F} = U_q \Sigma_q V_q^T, \qquad (\nabla_{\boldsymbol{w}} \boldsymbol{F})^+ = V_q \Sigma_q^+ U_q^T, \tag{4.12}$$

where $U_q \in \mathbb{R}^{M\times q}$ and $V \in \mathbb{R}^{N\times q}$ and $\Sigma_q \in \mathbb{R}^{q\times q}$. Thus, the solution of Eq.(4.4) is given by:

$$d\boldsymbol{w}^{(n)} = -V_q \Sigma_q^+ U_q^T \cdot \boldsymbol{F}(\boldsymbol{w}^{(n)}, \lambda).$$

Branches of solutions past turning points can be traced by solving the augmented, with the pseudo-arc-length condition, problem given by Eq.(4.7). In particular in (4.7), for the RPNN framework (2.40), the term $\nabla_{\boldsymbol{w}} N$ becomes:

$$\nabla_{\boldsymbol{w}} N = \boldsymbol{R}^T \frac{(\tilde{u}(\boldsymbol{w})_{-2} - \tilde{u}(\boldsymbol{w})_{-1})}{ds},$$

where $\boldsymbol{R}$ is the collocation matrix defined in Eq. (2.55).

**Remark 4.2.2.** The three numerical methods (FD, FEM and RPNN) are compared w.r.t. the dimension of the Jacobian matrix $J$, that in the case of FD and FEM is square and related to the number $N$ of nodes, i.e., $J \in \mathbb{R}^{N\times N}$, and in the case RPNN is rectangular and related to both the number $M$ of collocation nodes and the number $N$ of neurons, i.e., $J \in \mathbb{R}^{M\times N}$. Actually, $N$ is the parameter related to the computational cost, i.e., the inversion of the $J$ for FD and FEM is done by LU factorization that has a computational cost of $O(N^3)$. For the RPNN case, which leads to an under-determined system, the computational cost is related to the inversion of the matrix $J^T J \in \mathbb{R}^{N\times N}$, that therefore has the same leading computational cost $O(N^3)$. Moreover, if the solution is computed with the Moore–Penrose pseudo–inverse, the computational cost is based on SVD, which has a computational cost of $O(MN^2 + M^2N)$. Finally, we make explicit that in all the rest of the paper, we use RPNN networks with a number of $M$ of collocation points that is half the number $N$ of neurons in the hidden layer. In general, we pinpoint that by increasing the number $M$ to be $\frac{2N}{3}, \frac{3N}{4}, etc..$[2] one gets even better results.

Thus, in the case of collocation methods, such as FD and the proposed ML scheme (RPNNs), most of the computational cost is related to the solution of the linear system. In the case of isoparametric Galerkin methods, there is an extra computational cost related with the computation of the quantities of interest, such as the derivatives of the shape functions and the computation of the integrals.

[2]The case $M = N$ can be solved only by the use of a (Moore-Penrose) pseudo-inverse (4.12), because the invertibility of the Jacobian of the nonlinear PDE operator cannot be guaranteed in advance.

**RPNN with sigmoid functions.** Here, we selected two different activation functions, sigmoid functions (SF) and radial basis functions (RBF). For the SF case, we select the logistic sigmoid, that is defined by

$$\psi_j(\boldsymbol{x}) \equiv \sigma_j(\boldsymbol{x}) = \frac{1}{1+\exp(-\boldsymbol{\alpha}_j \cdot \boldsymbol{x} - \beta_j)}. \tag{4.13}$$

the selection of the relative selection of the internal weights and biases have been already introduced in Eq. (2.96) for the 1d case and (2.98) for the 2d case.

In the next section, we repeat the analysis, specifically also for the RBF approach.

**RPNN with radial basis functions.** Here, for the RBF case, we select the Gaussian kernel, that is defined as follows:

$$\psi_j(\boldsymbol{x}) \equiv \varphi_j(\boldsymbol{x}) = \exp(-\varepsilon_j^2||\boldsymbol{x}-\boldsymbol{c}_j||_2^2) = \exp\left(-\varepsilon_j^2 \sum_{k=1}^{d} (\boldsymbol{x}(k) - c_{j,k})^2\right), \tag{4.14}$$

where $\boldsymbol{c}_j \in \mathbb{R}^d$ is the center point and $\varepsilon_j \in \mathbb{R}$ is the inverse of the standard deviation. For such functions, we have:

$$\begin{aligned} &\frac{\partial}{\partial \boldsymbol{x}(k)}\varphi_j(\boldsymbol{x}) = -2\varepsilon_j^2 \exp(-\varepsilon_j^2 r_j^2)(\boldsymbol{x}(k) - c_{j,k}), \\ &\frac{\partial^2}{\partial \boldsymbol{x}(k)^2}\varphi_j(\boldsymbol{x}) = \exp(-\varepsilon_j^2 r_j^2)\big(-2\varepsilon_j^2 + 4\varepsilon_j^4(\boldsymbol{x}(k) - c_{j,k})^2\big), \end{aligned} \tag{4.15}$$

where $r_j = ||\boldsymbol{x} - \boldsymbol{c}_j||_2$. In all the directions, the Gaussian kernel is a classical bell function such that:

$$\lim_{\|\boldsymbol{x}-\boldsymbol{c}_j\|\to+\infty} \phi_j(\boldsymbol{x}) = 0, \qquad \phi_j(\boldsymbol{c}_j) = 1.$$

Moreover, the parameter $\varepsilon_j^2$ controls the steepness of the amplitude of the bell function: if $\varepsilon_j \to +\infty$, then $\phi_j$ approximates the Dirac function, while if $\varepsilon \to 0$, $\phi_j$ approximates a constant function. Thus, in the case of RBFs, one can relate the role of $\varepsilon_j$ to the role of $\alpha_{j,k}$ for the case of SF. For RBFs, it is well known that the center has to be chosen as a point internal to the domain and also more preferable to be exactly a grid point, while the steepness parameter $\varepsilon$ is usually chosen to be the same for each function. Here, since we are embedding RBFs in the RPNN framework, we take randomly the steepness parameter $\varepsilon_j$ in order to have more variability in the functional space, while for the centers $\boldsymbol{c}_j$ we select equally-spaced points in the domain. Thus, as for the SF case, we set the parameters $\varepsilon_j^2$ random uniformly distributed as:

$$\varepsilon_j^2 \sim \mathcal{U}\left(\frac{1}{|I|}, \frac{N+65}{15|I|}\right),$$

where $N$ denotes the number of neurons in the hidden layer and $|I| = b - a$ is the domain length. Besides, note that for the RBF case, it is trivial to extend the above

into the multidimensional case, since $\varphi_j$ is already expressed w.r.t. the center. For the two-dimensional case, we do the same reasoning as for the SF taking:

$$\varepsilon_j^2 \sim \mathcal{U}\left(\frac{1}{2|I|}, \frac{\sqrt{N}+50}{30|I|}\right).$$

## 4.3 Numerical Analysis Results: the Case Studies

The efficiency of the proposed numerical scheme is demonstrated through two benchmark nonlinear PDEs, namely (a) the one dimensional nonlinear Burgers' equation with Dirichlet boundary conditions and also mixed boundary conditions, and, (b) the one- and two-dimensional Liouville–Bratu–Gelfand problem. These problems have been widely studied as have been used to model and analyze the behavior of many physical and chemical systems (see, e.g., [227, 228, 221, 222, 229, 230]).
All computations were made with a CPU Intel® Core™ i7-10750H CPU @2.60GHz, RAM 16.0 GB using MATLAB R2020b.
In this section, we present some known properties of the proposed problems and provide details on their numerical solution with FD, FEM and the proposed ML scheme (RPNN) with both logistic and Gaussian RBF transfer functions.

### 4.3.1 The Nonlinear Viscous Burgers Equation

Here, we consider the one-dimensional steady state viscous Burgers problem:

$$\nu \frac{\partial^2 u}{\partial x^2} - u \frac{\partial u}{\partial x} = 0 \tag{4.16}$$

in the unit interval $[0, 1]$, where $\nu > 0$ denotes the viscosity. For our analysis, we considered two different sets of boundary conditions:

- Dirichlet boundary conditions

$$u(0) = \gamma\,,\ u(1) = 0\,,\ \gamma > 0\,; \tag{4.17}$$

- Mixed boundary conditions: Neumann condition on the left boundary and zero Dirichlet on the right boundary:

$$\frac{\partial u}{\partial x}(0) = -\vartheta\,,\ u(1) = 0\,,\ \vartheta > 0\,. \tag{4.18}$$

The two sets of boundary conditions result in different behaviors (see [227, 231]). We summarize in the next two lemmas some main results.

**Lemma 4.3.1** (Dirichlet case)**.** Consider Eq. (4.16) with boundary conditions given by (4.17). Moreover, take (notice that $\gamma \xrightarrow[\nu\to 0]{} 1$):

$$\gamma = \frac{2}{1 + \exp(\frac{-1}{\nu})} - 1.$$

Then, the problem (4.16)-(4.17) has a unique solution given by:

$$u(x) = \frac{2}{1 + \exp(\frac{x-1}{\nu})} - 1\,. \tag{4.19}$$

We will use this test problem because the solution has a boundary layer.

**Lemma 4.3.2** (Mixed case)**.** Consider Eq.(4.16) with boundary conditions given by (4.18). The solution of the problem can be written as [227] :

$$u(x) = \sqrt{2c}\tanh\left(\frac{\sqrt{2c}}{2\nu}(1-x)\right), \tag{4.20}$$

where $c$ is a constant value which can be determined by the imposed Neumann condition. Then, for $\vartheta$ sufficiently small, the viscous Burgers' problem with mixed boundary conditions admits two solutions:

(a) a stable lower solution such that $\forall x \in (0,1)$:

$$u(x) \xrightarrow[\vartheta\to 0]{} 0, \quad \frac{\partial u(x)}{\partial x} \xrightarrow[\vartheta\to 0]{} 0,$$

(b) an unstable upper solution $u(x) > 0\, \forall x \in (0,1)$ such that:

$$\frac{\partial u(0)}{\partial x} \xrightarrow[\vartheta\to 0]{} 0, \quad \frac{\partial u(1)}{\partial x} \xrightarrow[\vartheta\to 0]{} -\infty\,,$$

and

$$\forall x \in (0,1)\,, \quad u(x) \xrightarrow[\vartheta\to 0]{} \infty\,.$$

*Proof.* The spatial derivative of (4.20) is given by:

$$\frac{\partial u(x)}{\partial x} = -\frac{c}{\nu}\mathrm{sech}^2\left(\frac{\sqrt{2c}}{2\nu}(1-x)\right). \tag{4.21}$$

(a) When $c \to 0$ then from Eq.(4.20), we get asymptotically the zero solution, i.e., $u(x) \to 0$, $\forall x \in (0,1)$ and from Eq.(4.21), we get $\frac{\partial u(x)}{\partial x} \to 0$, $\forall x \in (0,1)$. At $x = 1$, the Dirichlet boundary condition $u(1) = 0$ is satisfied exactly (see Eq.(4.20)), while at the left boundary $x = 0$ the Neumann boundary condition is also satisfied as due to Eq.(4.21) and our assumption ($\vartheta \to 0$): $\frac{\partial u(0)}{\partial x} = -\vartheta \to 0$, when $c \to 0$.

(b) When $\frac{\partial u(1)}{\partial x} \to -\infty$, then (4.21) is satisfied $\forall x \in (0,1)$ when $c \to \infty$. In that case, at $x = 0$, the Neumann boundary condition is satisfied as due to Eq.(4.21) is easy to prove that $\frac{\partial u(0)}{\partial x} \to 0$.

Indeed, from Eq.(4.21):

$$\lim_{c\to\infty} \frac{\partial u(x)}{\partial x} = -\lim_{c\to\infty} \frac{\nu}{\exp\frac{\sqrt{2c}}{\nu}} = 0. \tag{4.22}$$

Finally Eq.(4.20) gives $u(x) \to \infty$, $\forall x \in (0,1)$. □

To better understand the behavior of the unstable solution w.r.t. the left boundary condition, we can prove the following.

**Corollary 4.3.1.** Consider Eq.(4.16) with boundary conditions given by (4.18). For the non-zero solution, when $\vartheta = \epsilon \to 0$ the solution at $x = 0$ goes to infinity with values:

$$u(0) = \nu \log\left(\frac{\nu}{\epsilon}\right) \tanh\left(\frac{1}{2}\log\left(\frac{\nu}{\epsilon}\right)\right). \tag{4.23}$$

*Proof.* By setting the value of $\vartheta$ in the Neumann boundary condition to be a very small number, i.e., $\vartheta = \epsilon \ll 1$, then from Eq.(4.22), we get that the slope of the analytical solution given by Eq.(4.21) is equal to $\epsilon$, when

$$c = \frac{1}{2}\nu^2 \log^2\left(\frac{\nu}{\epsilon}\right). \tag{4.24}$$

Plugging the above into the analytical solution, given by Eq.(4.20), we get Eq.(4.23). □

The above findings imply also the existence of a limit point bifurcation w.r.t. $\vartheta$ that depends also on the viscosity. For example, as shown in [227], for $\vartheta > 0$ and $\nu = 1/10$, there are two equilibria arising due to a turning point at $\vartheta^* = 0.087845767978$.

### *Numerical Solution of the Burgers' equation with Finite Differences and Finite Elements*

The discretization of the one-dimensional viscous Burgers' problem in $N$ points with second-order central FD in the unit interval $0 \leq x \leq 1$ leads to the following system of $N-2$ algebraic equations $\forall x_j = (j-1)h, j = 2, \ldots N-1, h = \frac{1}{N-1}$:

$$F_j(\boldsymbol{u}) = \frac{\nu}{h^2}(u_{j+1} - 2u_j + u_{j-1}) - u_j \frac{u_{j+1} - u_{j-1}}{2h} = 0\,.$$

At the boundaries $x_1 = 0, x_N = 1$, we have $u_1 = \gamma$, $u_N = 0$, respectively for the Dirichlet boundary conditions (4.17) and $u_1 = (2h\vartheta + 4u_2 - u_3)/3$, $u_N = 0$, respectively for the mixed boundary conditions (4.18).

The above $N-2$ nonlinear algebraic equations are the residual equations (4.3) that are solved iteratively using Newton's method (4.4). The Jacobian (4.5) is now tridiagonal: at each $i$-th iteration, the non-null elements are given by:

$$\frac{\partial F_j}{\partial u_{j-1}} = \frac{\nu}{h^2} + \frac{u_j}{2h}\,;\; \frac{\partial F_j}{\partial u_j} = -\nu\frac{2}{h^2} - \frac{u_{j+1} - u_{j-1}}{2h}\,;\; \frac{\partial F_j}{\partial u_{j+1}} = \frac{\nu}{h^2} - \frac{u_j}{2}\,.$$

The Galerkin residuals (4.11) in the case of the one-dimensional Burgers equation read:

$$R_k = \int_0^1 \left(\nu\frac{\partial^2 u(x)}{\partial x^2} - u\frac{\partial u(x)}{\partial x}\right)\phi_k(x)dx. \tag{4.25}$$

By inserting the numerical solution (4.10) into Eq.(4.25) and by applying the Green's formula for integration, we get:

$$\begin{aligned} R_k =& \nu\phi_k(x)\frac{du}{dx}\Big|_0^1 - \nu\sum_{j=1}^{N} u_j \int_0^1 \frac{d\phi_j(x)}{dx}\frac{d\phi_k(x)}{dx}dx \\ &- \int_0^1 \sum_{j=1}^{N} u_j\phi_j(x) \sum_{j=1}^{N} u_j \frac{d\phi_j(x)}{dx}\phi_k(x)dx. \end{aligned} \tag{4.26}$$

At the above residuals, we have to impose the boundary conditions. If Dirichlet boundary conditions (4.17) are imposed, Eq. (4.26) becomes:

$$\begin{aligned} R_k =& -\nu\sum_{j=1}^{N} u_j \int_0^1 \frac{d\phi_j(x)}{dx}\frac{d\phi_k(x)}{dx}dx \\ &- \int_0^1 \sum_{j=1}^{N} u_j\phi_j(x) \sum_{j=1}^{N} u_j \frac{d\phi_j(x)}{dx}\phi_k(x)dx. \end{aligned} \tag{4.27}$$

In the case of the mixed boundary conditions (4.18), Eq.(4.26) becomes:

$$\begin{aligned} R_k =& \nu\vartheta\phi_k(0) - \nu\sum_{j=1}^{N} u_j \int_0^1 \frac{d\phi_j(x)}{dx}\frac{d\phi_k(x)}{dx}dx \\ &- \int_0^1 \sum_{j=1}^{N} u_j\phi_j(x) \sum_{j=1}^{N} u_j \frac{d\phi_j(x)}{dx}\phi_k(x)dx. \end{aligned} \tag{4.28}$$

In this paper, we use a $P^2$ FEM space, thus quadratic basis functions using an affine element mapping in the interval $[0,1]^d$. For the computation of the integrals, we used the Gauss quadrature numerical scheme: for the one-dimensional case, we used the three-points Gaussian rule:

$$\left\{ \left(\frac{1}{2} - \sqrt{\frac{3}{20}}, \frac{5}{18}\right), \left(0.5, \frac{8}{18}\right), \left(\frac{1}{2} + \sqrt{\frac{3}{20}}, \frac{5}{18}\right) \right\}.$$

When writing Newton's method (4.4), the elements of the Jacobian matrix for both (4.27) and (4.28) are given by:

$$\frac{\partial R_k}{\partial u_j} = -\nu\int_0^1 \frac{d\phi_j(x)}{dx}\frac{d\phi_k(x)}{dx}dx - 2\int_0^1 \sum_{j=1}^{N} u_j\phi_j(x)\frac{d\phi_j(x)}{dx}\phi_k(x)dx. \tag{4.29}$$

Finally, with all the above, the Newton's method (4.4) involves the iterative solution of a

linear system. For the Dirichlet problem this becomes:

$$\begin{bmatrix} 1 & 0 & \dots & 0 & \dots & 0 \\ \frac{\partial R_2}{\partial u_1} & \frac{\partial R_2}{\partial u_2} & \dots & \frac{\partial R_2}{\partial u_j} & \dots & \frac{\partial R_2}{\partial u_N} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{\partial R_k}{\partial u_1} & \frac{\partial R_k}{\partial u_2} & \dots & \frac{\partial R_k}{\partial u_j} & \dots & \frac{\partial R_k}{\partial u_N} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \dots & 1 \end{bmatrix}_{|u^{(n)}} \cdot \begin{bmatrix} du_1^{(n)} \\ du_2^{(n)} \\ \vdots \\ du_j^{(n)} \\ \vdots \\ du_N^{(n)} \end{bmatrix} = - \begin{bmatrix} 0 \\ R_2 \\ \vdots \\ R_k \\ \vdots \\ 0 \end{bmatrix}_{|u^{(n)}} , \tag{4.30}$$

while for the problem with the mixed boundary conditions, at each iteration, we need to solve the following system:

$$\begin{bmatrix} \frac{\partial R_1}{\partial u_1} & \frac{\partial R_1}{\partial u_2} & \dots & \frac{\partial R_1}{\partial u_j} & \dots & \frac{\partial R_1}{\partial u_N} \\ \frac{\partial R_2}{\partial u_1} & \frac{\partial R_2}{\partial u_2} & \dots & \frac{\partial R_2}{\partial u_j} & \dots & \frac{\partial R_2}{\partial u_N} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{\partial R_k}{\partial u_1} & \frac{\partial R_k}{\partial u_2} & \dots & \frac{\partial R_k}{\partial u_j} & \dots & \frac{\partial R_k}{\partial u_N} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \dots & 1 \end{bmatrix}_{|u^{(n)}} \cdot \begin{bmatrix} du_1^{(n)} \\ du_2^{(n)} \\ \vdots \\ du_j^{(n)} \\ \vdots \\ du_N^{(n)} \end{bmatrix} = - \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_k \\ \vdots \\ 0 \end{bmatrix}_{|u^{(n)}} . \tag{4.31}$$

### *Numerical Solution of the Burgers' equation with RPNN Collocation*

Collocating the RPNN network function for the one-dimensional Burgers equation leads to the following nonlinear algebraic system for $i = 2, \dots, M-1$:

$$F_i(\boldsymbol{w}, \nu) = \nu \sum_{j=1}^{N} w_j \frac{d^2\psi_j(x_i)}{dx^2} - \left(\sum_{j=1}^{N} w_j \psi_j(x_i)\right)\left(\sum_{j=1}^{N} w_j \frac{d\psi_j(x_i)}{dx}\right) = 0 . \tag{4.32}$$

Then, the imposition of the boundary conditions (4.17) gives:

$$F_1(\boldsymbol{w}, \nu) = \sum_{j=1}^{N} w_j \psi_j(0) - \gamma = 0, \qquad F_M(\boldsymbol{w}, \nu) = \sum_{j=1}^{N} w_j \psi_j(1) = 0 , \tag{4.33}$$

while boundary conditions (4.18) lead to:

$$F_1(\boldsymbol{w}, \nu) = \sum_{j=1}^{N} w_j \frac{d\psi_j(0)}{dx} + \vartheta = 0, \qquad F_M(\boldsymbol{w}, \nu) = \sum_{j=1}^{N} w_j \psi_j(1) = 0 . \tag{4.34}$$

These equations are the residual equations (4.3) that we solve by Newton's method (4.4). The elements of the Jacobian matrix $\nabla_{\boldsymbol{w}} \boldsymbol{F}$ are given by:

$$\frac{\partial F_i}{\partial w_j} = \nu \frac{d^2\psi_j(x_i)}{dx^2} - \psi_j(x_i)\left(\sum_{j=1}^{N} w_j \frac{d\psi_j(x_i)}{dx}\right) - \left(\sum_{j=1}^{N} w_j \psi_j(x_i)\right)\frac{d\psi_j(x_i)}{dx}.$$

For $i = 2, \ldots, M-1$ and due to the Dirichlet boundary conditions (4.33), we have:

$$\frac{\partial F_1}{\partial w_j}(\boldsymbol{w}, \lambda) = \psi_j(0) \qquad \frac{\partial F_M}{\partial w_j}(\boldsymbol{w}, \lambda) = \psi_j(1).$$

On the other hand, due to the mixed boundary conditions given by (4.34), we get:

$$\frac{\partial F_1}{\partial w_j}(\boldsymbol{w}, \lambda) = \frac{d\psi_j(0)}{dx} \qquad \frac{\partial F_M}{\partial w_j}(\boldsymbol{w}, \lambda) = \psi_j(1).$$

At this point, the application of Newton's method (4.4) using the exact computation of the derivatives of the basis functions is straightforward (see (2.94) and (4.15)).

### *Numerical Results*

In all the computations with FD, FEM and the proposed ML scheme (RPNN), the convergence criterion for Newton's iterations was the $L_2$ norm[3] of the relative error between the solutions resulting from successive iterations; the convergence tolerance was set to $10^{-6}$. In fact, for all methods, Newton's method converged quadratically also up to the order of $10^{-10}$, when the bifurcation parameter was not close to zero where the solution of both Burgers with mixed boundary conditions and Bratu problems goes asymptotically to infinity. The exact solutions that are available for the one-dimensional Burgers and Bratu problems are derived using Newton's method, with a convergence tolerance of $10^{-12}$.

| | RPNN SF | | | RPNN RBF | | |
|---|---|---|---|---|---|---|
| N | 5% | mean | 95% | 5% | mean | 95% |
| 80 | 2.73E−03 | 4.70E−03 | 4.38E−03 | 2.31E−03 | 2.67E−03 | 3.16E−03 |
| 160 | 8.46E−03 | 9.97E−03 | 1.12E−02 | 7.16E−03 | 8.20E−03 | 9.08E−03 |
| 320 | 3.72E−02 | 4.23E−02 | 4.60E−02 | 3.52E−02 | 3.89E−02 | 4.28E−02 |
| 640 | 1.60E−01 | 1.67E−01 | 1.75E−01 | 1.56E−01 | 1.69E−01 | 1.97E−01 |
| | FD | | | FEM | | |
| N | 5% | mean | 95% | 5% | mean | 95% |
| 80 | 1.72E−04 | 3.37E−04 | 3.48E−04 | 2.33E−02 | 2.49E−02 | 2.76E−02 |
| 160 | 4.31E−04 | 4.55E−04 | 5.26E−04 | 5.68E−02 | 6.39E−02 | 6.95E−02 |
| 320 | 1.29E−03 | 1.33E−03 | 1.44E−03 | 1.22E−01 | 1.24E−01 | 1.31E−01 |
| 640 | 1.05E−02 | 1.10E−02 | 1.16E−02 | 3.34E−01 | 3.40E−01 | 3.55E−01 |

Table 4.1: Execution times (s) for the Burgers Eq. (4.16) with Dirichlet boundary conditions (4.17) and $\nu = 0.1$.

First, we present the numerical results for the Burgers' Eq. (4.16) with Dirichlet boundary conditions (4.17). Recall that for this case, the exact solution is available (see Eq. (4.19)).

[3]The relative error is the $L_2$–norm of the difference between two successive solutions $||u(\boldsymbol{w})_{-2} - u(\boldsymbol{w})_{-1}||_2$. In particular, for the RPNN framework, it is given by $||R^T \cdot (\boldsymbol{w}_{-2} - \boldsymbol{w}_{-1})||_2$, where $R$ is the collocation matrix defined in Eq. (2.55).

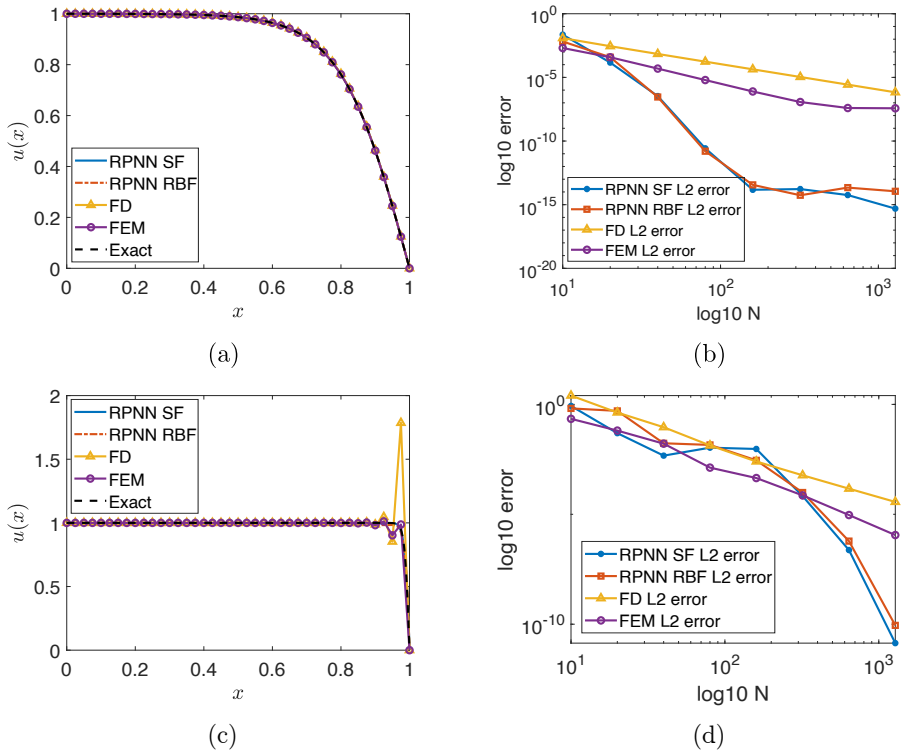



Figure 4.1: Numerical solution and accuracy of the FD, FEM and the proposed ML scheme (RPNN) for the one-dimensional viscous Burgers' problem with Dirichlet boundary conditions (4.16), (4.17), (a,b) with viscosity $\nu = 0.1$: (a) Solutions for a fixed problem size $N = 40$; (b) $L_2$–norm of differences w.r.t. the exact solution (4.19) for various problem sizes. (c,d) with viscosity $\nu = 0.007$: (c) Solutions for a fixed problem size $N = 40$; (d) $L_2$–norm errors w.r.t. the exact solution for various problem sizes.

For our illustrations, we have selected two different values for the viscosity, namely $\nu = 0.1$ and $\nu = 0.007$. Results were obtained with Newton's iterations starting from an initial guess that is a linear segment that satisfies the boundary conditions.

Figure 4.1 shows the corresponding computed solutions for a fixed size $N = 40$, as well as the relative errors w.r.t. the exact solution. As it is shown, the proposed ML scheme outperforms both the FD and FEM schemes for medium to large sizes of the grid; from low to medium sizes of the grid, all methods perform equivalently.

However, as shown in Figure 4.1(c), for $\nu = 0.007$, and the particular choice of the size ($N = 40$), the FD scheme fails to approximate sufficiently the steep-gradient appearing at the right boundary. Table 4.1, summarizes the execution times of the four methods (RPNN SF, RPNN RBF, FD and FEM), when applied for the solution of the Burgers' Eq. (4.16) with Dirichlet conditions (4.17) and $\nu = 0.1$ for various sizes of the

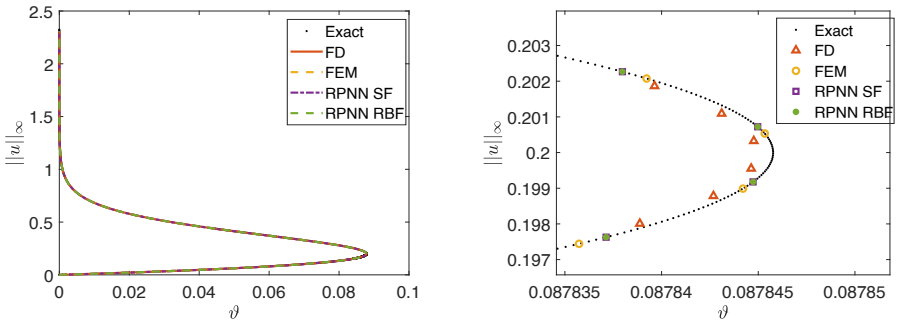



Figure 4.2: (a) One-dimensional Burgers equation (4.16) with mixed boundary conditions (4.18). Bifurcation diagram w.r.t. the Neumann boundary value $\theta$ as obtained for $\nu = 1/10$, with FD, FEM and RPNN schemes with a fixed problem size $N = 400$; (b) Zoom near the turning point.

| N | FD | FEM | RPNN SF | RPNN RBF |
|---|---|---|---|---|
| 20 | -3.32E−04 | -4.86E−09 | 2.75E−08 | -4.37E−06 |
| 50 | -5.35E−05 | -7.67E−09 | -2.06E−09 | -2.14E−09 |
| 100 | -1.34E−05 | -2.16E−09 | -9.84E−09 | -9.85E−09 |
| 200 | -3.34E−06 | -5.93E−09 | -9.62E−09 | -9.61E−09 |
| 400 | -8.35E−07 | 4.15E−09 | 9.38E−10 | 9.33E−10 |

Table 4.2: One-dimensional Burgers equation (4.16) with mixed boundary conditions (4.18). Comparative results w.r.t. the error between the estimated value of the turning point as obtained with FD, FEM and proposed ML scheme (RPNN) and the exact value of the turning point at $\vartheta^* = 0.087845767978$ for $\nu = 1/10$. The value of the turning point was estimated by fitting a parabola around the four points with the largest $\lambda$ values, as obtained by the arc-length continuation.

grid.

The computations are performed 100 times, and we also provide the 5% and 95% percentiles. For all practical means, the execution times obtained with RPNNs are comparable with the ones obtained with FD and FEM (with the ones obtained with the proposed ML scheme to be slightly faster that the ones obtained with FEM), while, as seen, numerical approximation accuracy is better in the RPNNs case; the execution times when using the FD are smaller but as shown the FD scheme fails to approximate solutions with steep gradients while its numerical accuracy is generally lower when compared with FEM and the proposed ML scheme.

Then, we considered the case of the non-homogeneous Neumann condition on the left boundary (4.16)- (4.18); here, we have set $\nu = 1/10$. In this case, the solution is not unique and the resulting bifurcation diagram obtained with FD, FEM and the proposed

| N | FD | FEM | RPNN SF | RPNN RBF |
|---|---|---|---|---|
| 20 | -1.81E−01 | 2.05E−02 | -6.55E−01 | -6.14E−01 |
| 50 | -2.66E−02 | 7.67E−04 | -5.83E−01 | -6.09E−01 |
| 100 | -6.52E−03 | 1.58E−04 | -2.00E−01 | -1.05E−01 |
| 200 | -1.61E−03 | 8.99E−05 | -2.50E−06 | -5.05E−06 |
| 400 | -4.00E−04 | 6.28E−05 | -3.47E−06 | -9.52E−06 |

Table 4.3: One-dimensional Burgers equation (4.16) with mixed boundary conditions (4.18). Comparative results w.r.t. the error between the computed solution (at $x = 0$) with FD, FEM and proposed ML scheme (RPNN) (with both sigmoid and RBF) and the exact solution $u(0) = 1.798516682636303$ (see Eq. (4.20)) for $\vartheta = 1e - 6$ (the value of the Neumann condition at the left boundary).

ML scheme (RPNN) is depicted in Fig.(4.2). In Table 4.2, we report the error between the value of the bifurcation point as computed with FD, FEM and the proposed ML scheme (RPNN) for various problem sizes $N$, w.r.t. the exact value of the bifurcation point (occurring for the particular choice of viscosity at $\vartheta^* = 0.087845767978$). The location of the bifurcation point for all numerical methods was estimated by fitting a parabola around the four points (two on the lower and two on the upper branch) of the largest values of $\lambda$, as obtained by the pseudo-arc-length continuation. As shown, the proposed RPNN scheme performs equivalently to FEM for low to medium sizes of the grid, thus outperforming FEM for medium to large grid sizes; both methods FEM and the proposed ML scheme (RPNN) outperform FD for all sizes of the grid.

In this case, steep gradients arise at the right boundary related to the presence of the upper unstable solution, as discussed in Lemma 4.3.2 and Corollary 4.3.1. In Table 3, we report the error between the numerically computed and the exact analytically obtained value (see Eq. (4.20)) at $x = 0$ when the value of boundary condition $\vartheta$ at the left boundary is $\vartheta = 10^{-6}$. Again, as shown, near the left boundary, the proposed RPNN scheme outperforms both FEM and FD for medium to larger sizes of the grid.

### 4.3.2 The one- and two-dimensional Liouville–Bratu–Gelfand Problem

The Liouville–Bratu–Gelfand model arises in many physical and chemical systems. It is an elliptic PDE which in its general form is given by [228]:

$$\Delta u(\boldsymbol{x}) + \lambda e^{u(\boldsymbol{x})} = 0 \ \ \boldsymbol{x} \in \Omega, \tag{4.35}$$

with homogeneous Dirichlet conditions,

$$u(\boldsymbol{x}) = 0\,, \ \boldsymbol{x} \in \partial\Omega. \tag{4.36}$$

The domain that we consider here is the $\Omega = [0,1]^d$ in $R^d$ , $d = 1, 2$.

The one-dimensional problem admits an analytical solution given by [232]:

$$u(x) = 2\ln\frac{\cosh\theta}{\cosh\theta(1-2x)}, \text{ where } \theta \text{ is such that } \cosh\theta = \frac{4\theta}{\sqrt{2\lambda}}. \tag{4.37}$$

It can be shown that when $0 < \lambda < \lambda_c$, the problem admits two branches of solutions that meet at $\lambda_c \sim 3.513830719$, a limit point (saddle-node bifurcation) that marks the onset of two branches of solutions with different stability, while beyond that point no solutions exist.

For the two-dimensional problem, to the best of our knowledge, no such (as in the one-dimensional case) exact analytical solution exist that is verified by the numerical results that have been reported in the literature (e.g., [221, 229]), in which the authors report the value of the turning at $\lambda_c \sim 6.808124$.

### *Numerical Solution with Finite Differences and Finite Elements*

The discretization of the one-dimensional problem in $N$ points with central FD at the unit interval $0 \leq x \leq 1$ leads to the following system of $N-2$ algebraic equations $\forall x_j = (j-1)h, j = 2, \dots N-1, h = \frac{1}{N-1}$:

$$F_j(u) = \frac{1}{h^2}(u_{j+1} - 2u_j + u_{j-1}) + \lambda e^{u_j} = 0,$$

where, at the boundaries $x_1 = 0, x_N = 1$, we have $u_1 = u_N = 0$.

The solution of the above $N-2$ nonlinear algebraic equations is obtained iteratively using the Newton-Raphson method. The Jacobian is now tridiagonal; at each $n$-th iteration, the elements at the main diagonal are given by $\frac{\partial F_j}{\partial u_j}^{(n)} = -\frac{2}{h^2} + \lambda e^{u_j^{(n)}}$ and the elements of the first diagonal above and the first diagonal below are given by $\frac{\partial F_{j+1}}{\partial u_j}^{(n)} = \frac{\partial F_j}{\partial u_{j+1}}^{(n)} = \frac{1}{h^2}$, respectively.

The discretization of the two-dimensional Bratu problem in $N \times N$ points with central FD on the square grid $0 \leq x, y \leq 1$ with zero boundary conditions leads to the following system of $(N-2) \times (N-2)$ algebraic equations $\forall (x_i = (i-1)h, y_j = (j-1)h), i, j = 2, \dots N-1, h = \frac{1}{N-1}$:

$$F_{i,j}(u) = \frac{1}{h^2}(u_{i+1,j} + u_{i,j+1} - 4u_{i,j} + u_{i,j-1} + u_{i-1,j}) + \lambda e^{u_{i,j}} = 0.$$

The Jacobian is now a $(N-2)^2 \times (N-2)^2$ block diagonal matrix of the form:

$$\nabla F = \frac{1}{h^2}\begin{bmatrix} T_2 & I & 0 & 0 & \dots & \dots & 0 \\ I & T_3 & I & 0 & \dots & \dots & 0 \\ 0 & I & T_4 & I & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & \dots & \dots & I & T_{N-1} \end{bmatrix},$$

where $I$ is the $(N-2) \times (N-2)$ identity matrix and $T_i$ is the $(N-2) \times (N-2)$ tridiagonal matrix with non-null elements on the $j$-th row:

$$1\,, \quad -4 + h^2 \lambda e^{u_{i+j,i+j}}\,, \quad 1$$

Regarding the FEM solution, for the one-dimensional Bratu problem, Eq. (4.11) gives:

$$R_k = \int_\Omega \left( \frac{\partial^2 u}{\partial x^2} + \lambda e^{u(x)} \right) \phi_k(x) dx. \tag{4.38}$$

By inserting Eq.(4.10) into Eq.(4.38) and by applying the Green's formula for integration, we get:

$$R_k = \phi_k(x) \frac{du}{dx}\Big|_0^1 - \sum_{j=1}^{N} u_j \int_0^1 \frac{d\phi_j(x)}{dx} \frac{d\phi_k(x)}{dx} dx + \lambda \int_0^1 e^{\sum_{j=1}^{N} u_j \phi_j(x)} \phi_k(x) dx \tag{4.39}$$

and because of the zero Dirichlet boundary conditions, Eq.(4.39) becomes:

$$R_k = -\sum_{j=1}^{N} u_j \int_0^1 \frac{d\phi_j(x)}{dx} \frac{d\phi_k(x)}{dx} dx + \lambda \int_0^1 e^{\sum_{j=1}^{N} u_j \phi_j(x)} \phi_k(x) dx.$$

The elements of the Jacobian matrix are given by:

$$\frac{\partial R_k}{\partial u_j} = -\int_0^1 \frac{d\phi_j(x)}{dx} \frac{d\phi_k(x)}{dx} dx + \lambda \int_0^1 e^{\sum_{j=1}^{N} u_j \phi_j(x)} \phi_j(x) \phi_k(x) dx \tag{4.40}$$

Due to the Dirichlet boundary conditions, Eq.(4.40) becomes:

$$\begin{bmatrix} 1 & 0 & \dots & 0 & \dots & 0 \\ \frac{\partial R_2}{\partial u_1} & \frac{\partial R_2}{\partial u_2} & \dots & \frac{R_2}{\partial u_j} & \dots & \frac{R_2}{\partial u_N} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{\partial R_k}{\partial u_1} & \frac{\partial R_k}{\partial u_2} & \dots & \frac{R_k}{\partial u_j} & \dots & \frac{R_k}{\partial u_N} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \dots & 1 \end{bmatrix}_{|u^{(n)}} \cdot \begin{bmatrix} du_1^{(n)} \\ du_2^{(n)} \\ \vdots \\ du_j^{(n)} \\ \vdots \\ du_N^{(n)} \end{bmatrix} = - \begin{bmatrix} 0 \\ R_2 \\ \vdots \\ R_k \\ \vdots \\ 0 \end{bmatrix}_{|u^{(n)}} . \tag{4.41}$$

For the two-dimensional Bratu problem, the residuals are given by:

$$R_k = \iint_\Omega \left( \frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} + \lambda e^{u(x,y)} \right) \phi_k(x,y) dx dy.$$

By applying the Green's formula for integration, we get:

$$R_k = \oint_{\partial\Omega} \nabla u(x,y) d\ell - \iint_\Omega \nabla u(x,y) \nabla \phi_k(x,y) dx dy + \iint_\Omega \lambda e^{u(x,y)} \phi_k(x,y) dx dy.$$

By inserting Eq.(4.10) and the zero Dirichlet boundary conditions, we get:

$$R_k = -\sum_{j=1}^{N} u_j \iint_\Omega \nabla \phi_j(x,y) \nabla \phi_k(x,y) dx dy + \iint_\Omega \lambda e^{\sum_{j=1}^{N} u_j \phi_j(x,y)} \phi_k(x,y) dx dy.$$

Thus, the elements of the Jacobian matrix for the two-dimensional Bratu problem are given by:

$$\frac{\partial R_k}{\partial u_j} = -\iint_\Omega \nabla\phi_j(x,y)\nabla\phi_k(x,y)dxdy + \iint_\Omega \lambda e^{\sum_{j=1}^N u_j\phi_j(x,y)}\phi_j(x,y)\phi_k(x,y)dxdy.$$

As before, for our computations we have used quadratic basis functions using an affine element mapping in the domain $[0,1]^2$.

### *Numerical Solution with RPNN Collocation*

Collocating the RPNN network function (2.40) in the 1D Bratu problem (4.35) leads to the following system:

$$F_i(\boldsymbol{w},\lambda) = \sum_{j=1}^N w_j \frac{d^2\psi_j(x_i)}{dx^2} + \lambda \exp\left(\sum_{j=1}^N w_j\psi_j(x_i)\right) = 0, \quad i = 2,\ldots,M-1,$$

with boundary conditions:

$$F_1(\boldsymbol{w},\lambda) = \sum_{j=1}^N w_j\psi_j(0) = 0, \qquad F_M(\boldsymbol{w},\lambda) = \sum_{j=1}^N w_j\psi_j(1) = 0.$$

Thus, the elements of the Jacobian matrix $\nabla_{\boldsymbol{w}}\boldsymbol{F}$ are given by:

$$\frac{\partial F_i}{\partial w_j} = \frac{d^2\psi_j(x_i)}{dx^2} + \lambda\psi_j(x_i)\exp\left(\sum_{j=1}^N w_j\psi_j(x_i)\right), \qquad i = 2,\ldots,M-1,$$

and

$$\frac{\partial F_1}{\partial w_j}(\boldsymbol{w},\lambda) = \psi_j(0) \qquad \frac{\partial F_M}{\partial w_j}(\boldsymbol{w},\lambda) = \psi_j(1).$$

The application of Newton's method (4.4) is straightforward, using the exact computation of derivatives of the basis functions (see (2.94) and (4.15)).

For the two-dimensional Bratu problem (4.35), we have:

$$F_i(\boldsymbol{w},\lambda) = \sum_{j=1}^N w_j \frac{\partial^2\psi_j(x_i,y_i)}{\partial x^2} + \sum_{j=1}^N w_j \frac{\partial^2\psi_j(x_i,y_i)}{\partial y^2} + \lambda\exp\left(\sum_{j=1}^N w_j\psi_j(x_i,y_i)\right) = 0, \quad i = 1,\ldots,M_\Omega$$

with boundary conditions:

$$\frac{\partial F_i}{\partial w_j} = \frac{\partial^2\psi_j(x_i,y_i)}{\partial x^2} + \frac{\partial^2\psi_j(x_i,y_i)}{\partial y^2} + \lambda\psi_j(x_i,y_i)\exp\left(\sum_{j=1}^N w_j\psi_j(x_i,y_i)\right), \qquad i = 1,\ldots,M_\Omega$$

and

$$\frac{\partial F_k}{\partial w_j}(\boldsymbol{w}, \lambda) = \psi_j(x_k, y_k) = 0, \quad k = 1, \ldots, M_1 \, .$$

Also in this case, with the above computations, the application of Newton's method (4.4) is straightforward.

### *Numerical results for the one-dimensional problem*

First, we show the numerical results for the one-dimensional Liouville–Bratu–Gelfand equation (4.35) with homogeneous Dirichlet boundary conditions (4.36).

Recall that an exact solution, although in implicit form, is available in this case (see Eq. (4.37)); thus, as discussed, the exact solutions are derived using Newton's method with a convergence tolerance of $10^{-12}$. Figure 4.3 depicts the comparative results between the exact, FD, FEM and RPNN solutions on the upper-branch as obtained by applying Newton's iterations, for two values of the parameter $\lambda$ and a fixed $N = 40$, namely for $\lambda = 3$ close to the turning point (occurring at $\lambda_c \sim 3.513830719$) and for $\lambda = 0.2$. For our illustrations, we have set as initial guess $u_0(x)$ a parabola that satisfies the homogeneous boundary conditions, namely:

$$u_0(x) = 4l_0(x - x^2),$$

with a fixed $L_\infty$–norm $||u||_\infty = l_0$ close to the one obtained from the exact solution.

In particular, for $\lambda = 3$, we used as initial guess a parabola with $l_0 = 2.2$; in all cases Newton's iterations converge to the correct unstable upper-branch solution. For $\lambda = 0.2$, we used as initial guess a parabola with $l_0 = 6.4$ (the exact solution has $l_0 \sim 6.5$); again in all cases, Newton's iterations converged to the correct unstable upper-branch solution. In Table 4, we compare the execution times of the four methods when applied to the solution of the Bratu Eq. (4.35) with Dirichlet boundary condition (4.36) and $\lambda = 1$. Computations are performed 100 time, and we also provide the 5% and 95% percentiles. Again, for all practical means, the execution times obtained with the proposed ML scheme are comparable with the ones obtained with FD and FEM (note that the execution times obtained with the proposed ML scheme are slightly smaller that the ones obtained with FEM), while, as seen, numerical approximation accuracy is better in the RPNNs case; the execution times when using the FD are slightly smaller but as shown the FD scheme results to a lower numerical accuracy compared with FEM and the proposed ML scheme (RPNN).

### *Bifurcation diagram and numerical accuracy*

In this section, we report the numerical results obtained by the numerical bifurcation analysis of the one-dimensional Bratu problem (4.35).

Figure 4.4 shows the constructed bifurcation diagram w.r.t. the parameter $\lambda$ and in Table 5, we report the accuracy of the computed value as obtained with FD, FEM and the proposed ML scheme (RPNN), versus the exact value of the turning point.

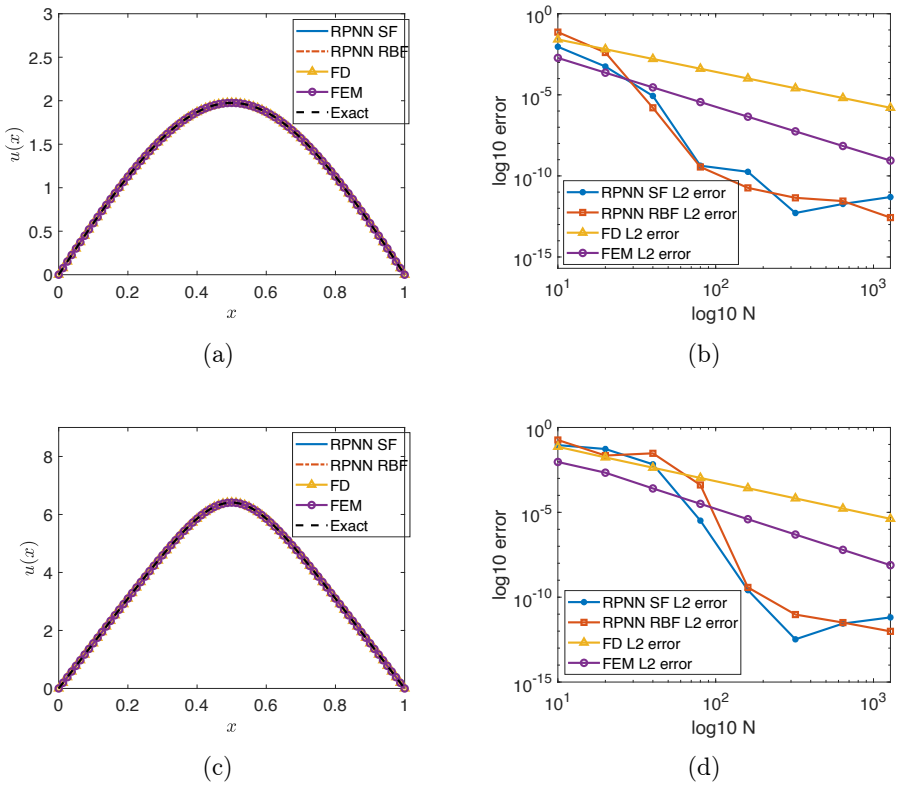



Figure 4.3: Numerical solutions and accuracy of the FD, FEM and the proposed ML scheme (RPNN) for the one-dimensional Bratu problem (4.35). (a) Computed solutions at the upper-branch unstable solution at $\lambda = 3$ for a fixed problem size $N = 40$. (b) $L_2$–norm of differences w.r.t. the exact unstable solution (4.37) at $\lambda = 3$ for various values of $N$. (c) Computed solutions at the upper-branch unstable solution at $\lambda = 0.2$ with a fixed problem size $N = 40$. (d) $L_2$–norm of differences w.r.t. the exact unstable solution (4.37) at $\lambda = 0.2$ for various values of $N$. The initial guess of the solutions was a parabola satisfying the homogeneous boundary conditions with a fixed $L_\infty$–norm $||u||_\infty = l_0$ close to the one resulting from the exact solution.

As shown, our proposed ML scheme provides a bigger numerical accuracy for the value of the turning point for medium to large sizes of the grid, and equivalent results (RPNN with SF) to FEM, both outperforming the FD scheme.

In Figures 4.5 and 4.6, we depict the contour plots of the $L_\infty$–norms of the differences between the computed solutions by FD, FEM and the proposed ML scheme (RPNN) and the exact solutions for the lower- (4.5) and upper-branch (4.6), respectively w.r.t. $N$ and $\lambda$.

As it is shown, the proposed ML schemes outperform both FD and FEM methods

| | RPNN SF | | | RPNN RBF | | |
|---|---|---|---|---|---|---|
| N | 5% | mean | 95% | 5% | mean | 95% |
| 80 | 7.16E−03 | 8.14E−03 | 9.27E−03 | 2.07E−03 | 2.31E−03 | 2.61E−03 |
| 160 | 3.81E−02 | 4.23E−02 | 4.93E−02 | 3.53E−03 | 4.31E−03 | 4.96E−03 |
| 320 | 1.09E−02 | 1.16E−02 | 1.30E−02 | 7.12E−03 | 7.96E−03 | 8.57E−03 |
| 640 | 3.19E−02 | 3.38E−02 | 3.58E−02 | 2.81E−02 | 3.01E−02 | 3.17E−02 |
| | FD | | | FEM | | |
| N | 5% | mean | 95% | 5% | mean | 95% |
| 80 | 1.93E−04 | 2.60E−04 | 2.65E−04 | 2.58E−03 | 2.88E−03 | 3.03E−03 |
| 160 | 5.72E−04 | 7.01E−04 | 8.24E−04 | 5.76E−03 | 6.32E−03 | 6.89E−03 |
| 320 | 1.59E−03 | 1.86E−03 | 2.08E−03 | 1.15E−02 | 1.17E−02 | 1.20E−02 |
| 640 | 8.77E−03 | 9.07E−03 | 9.49E−03 | 3.00E−02 | 3.11E−02 | 3.21E−02 |

Table 4.4: Execution times (s) for the Bratu Eq. (4.35) with Dirichlet boundary condition (4.36) and $\lambda = 1$.

| N | FD | FEM | RPNN SF | RPNN RBF |
|---|---|---|---|---|
| 20 | -4.57E−03 | 3.44E−05 | 8.76E−05 | 3.00E−02 |
| 50 | -7.31E−04 | 8.44E−07 | 2.98E−07 | 6.61E−05 |
| 100 | -1.83E−04 | 5.06E−08 | -3.71E−08 | 6.13E−08 |
| 200 | -4.57E−05 | 2.36E−08 | -4.55E−09 | -2.68E−09 |
| 400 | -1.14E−05 | 1.36E−08 | 2.02E−09 | 2.03E−09 |

Table 4.5: One-dimensional Bratu problem (4.35). Accuracy of FD, FEM and the proposed ML scheme (RPNN) in the approximation of the value of the turning point w.r.t. the exact value $\lambda = 3.513830719125162$. Values express the difference with the computed turning point and the exact one. The value of the turning point was estimated by fitting a parabola around the four points with the largest $\lambda$ values as obtained with arc-length continuation.

for medium to large problem sizes $N$, and provide equivalent results with FEM for low to medium problem sizes, thus both (FEM and the proposed ML scheme (RPNN)) outperforming the FD scheme.

### *Numerical results for the two-dimensional problem*

For the two-dimensional problem (4.35)-(4.36), no exact analytical solution is available.

Thus, for comparing the numerical accuracy of the FD, FEM and the proposed ML scheme (RPNN), we considered the value of the bifurcation point that has been reported in key works as discussed in Section 4.3.2. Figure 4.7 depicts the computed bifurcation diagram as computed via pseudo-arc-length continuation (see section 4.2). Table 6, summarizes the computed values of the turning point as estimated with the FD, FEM and RPNN schemes for various sizes $N$ of the grid.

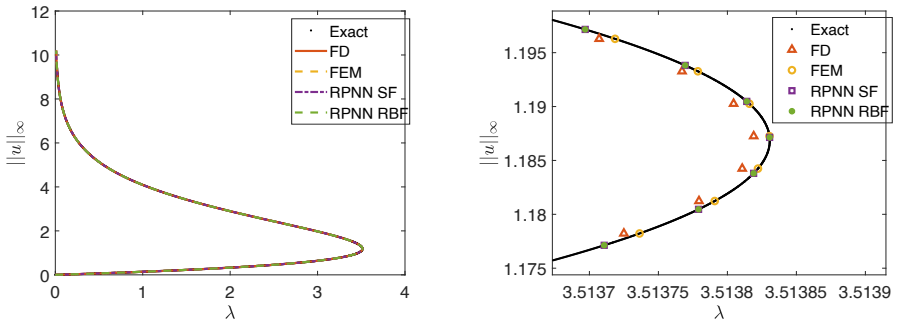



Figure 4.4: (a) Bifurcation diagram for the one-dimensional Bratu problem (4.35), with a fixed problem size $N = 400$. (b) Zoom near the turning point.

| N | Grid | FD | FEM | RPNN SF | RPNN RBF |
|---|---|---|---|---|---|
| 64 | 8×8 | 6.783434 | 7.083742 | 6.845015 | 7.207203 |
| 100 | 10×10 | 6.792626 | 6.984260 | 6.723902 | 6.930798 |
| 196 | 14×14 | 6.800361 | 6.900313 | 6.855055 | 6.882435 |
| 400 | 20×20 | 6.804392 | 6.856401 | 6.799440 | 6.829754 |
| 784 | 28×28 | 6.806235 | 6.835771 | 6.801689 | 6.806149 |
| 1600 | 40×40 | 6.807220 | 6.824770 | 6.806899 | 6.804600 |

Table 4.6: Turning point estimation of the two-dimensional Bratu problem. The value that has been reported in the literature in key works (see, e.g., [221]) is $\lambda^* = 6.808124$. The value of the turning point was estimated by fitting a parabola around the four points with the largest $\lambda$ values as obtained by the arc-length continuation.

**Remark 4.3.1** (The Gelfand-Bratu model)**.** The Liouville–Bratu–Gelfand Eq. (4.35) in a unitary ball $B \subset \mathbb{R}^d$ with homogeneous Dirichlet boundary conditions is usually refereed as Gelfand-Bratu model.

Such equation posses radial solutions $u(r)$ of the one-dimensional non-linear boundary-value problem [233]:

$$\begin{cases} u''(r) + \dfrac{d-1}{r} u'(r) + \lambda e^{u(r)} = 0 \quad 0 < r < 1 \\ u(1) = u'(0) = 0 \end{cases} \tag{4.42}$$

In the case $d = 2$, this equation gives multiple solutions if $\lambda < \lambda_c = 2$. For example, in [230], the authors have used `Mathematica` software to give analytical solutions at

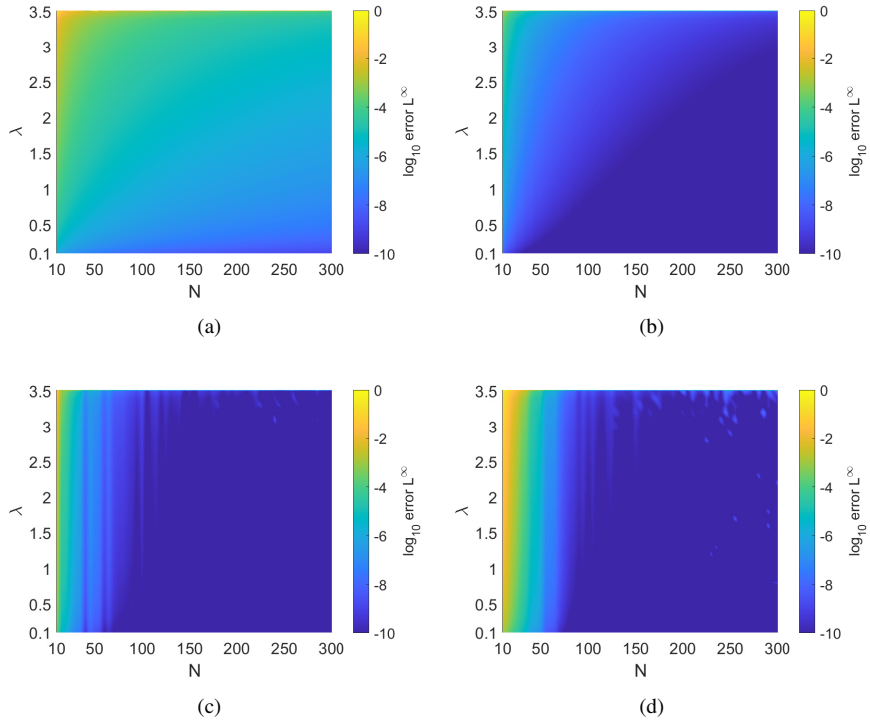



Figure 4.5: One-dimensional Bratu problem (4.35). Contour plots of the $L_\infty$–norms of the differences between the computed and exact (4.37) solutions for the lower stable branch: (a) FD, (b) FEM, (c) RPNN with logistic SF (4.13), (d) RPNN with Gaussian RBF (4.14).

various values of $\lambda$; for our tests we consider:

$$\begin{aligned} \lambda = \tfrac{1}{2} \;\rightarrow\quad & u(r) = \log\left(\frac{16\left(7+4\sqrt{3}\right)}{\left(7+4\sqrt{3}+r^2\right)^2}\right) \\ \lambda = 1 \;\rightarrow\quad & u(r) = \log\left(\frac{8\left(3+2\sqrt{2}\right)}{\left(3+2\sqrt{2}+r^2\right)^2}\right) . \end{aligned} \tag{4.43}$$

Figure 4.8 depicts the numerical accuracy of the proposed ML (RPNN) collocation scheme w.r.t. the exact solutions for two values of $\lambda$, namely for $\lambda = 1/2$ and for $\lambda = 1$. Because no meshing procedure is involved, and because the collocation equation seeks no other point, the implementation of the Newton's method is straightforward when changing the geometry of the domain.

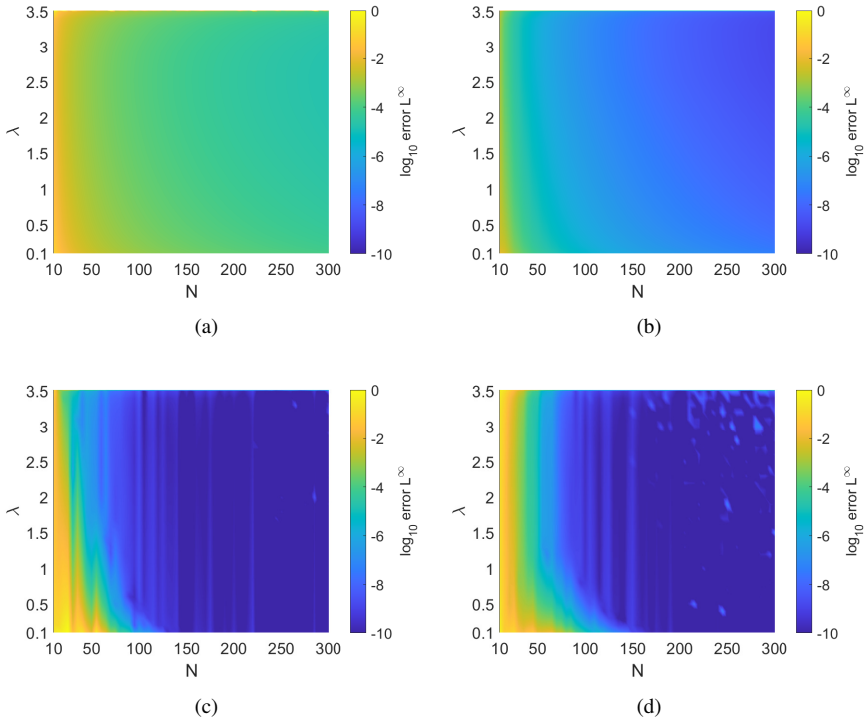



Figure 4.6: One-dimensional Bratu problem (4.35). Contour plots of the $L_\infty$–norms of the differences between the computed and exact (4.37) solutions for the upper unstable branch: (a) FD, (b) FEM, (c) RPNN with logistic SF (4.13), (d) RPNN with Gaussian RBF (4.14).

## 4.4 Discussion

We proposed an ML numerical method based on RPNNs and collocation for the approximation of steady-state solutions of non-linear PDEs. The proposed numerical scheme takes advantage of the property of the RPNNs as universal function approximators, bypassing the need of the computational very expensive - and most-of-the times without any guarantee for convergence - of the training phase of other types of ML approaches, such as single or multilayer ANNs and Deep-learning networks. The base of the approximation subspace on which a solution of the PDE is sought are the randomized transfer functions of the hidden layer which are weighted by the only unknown parameters, that is the weights of the hidden to output layer. Here, we address a new numerical scheme based on RPNNs that can be used to solve steady state problems of non-linear PDEs, and by bridging them with numerical continuation methods, we show how one can exploit the arsenal of numerical bifurcation theory to trace branches of solutions past

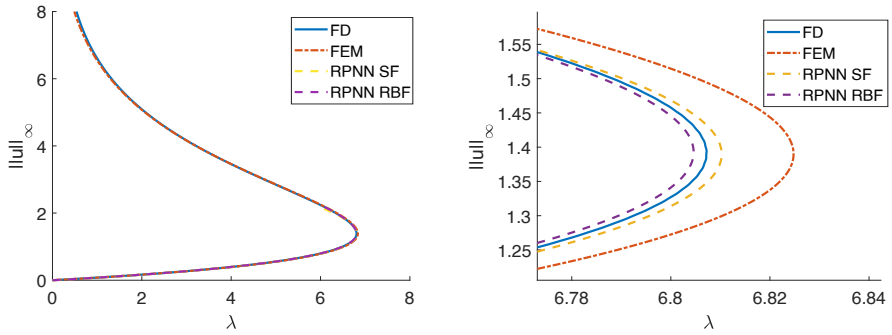



Figure 4.7: (a) Computed bifurcation diagram for the two-dimensional Bratu problem (4.35), with a grid of $40 \times 40$ points. b) Zoom near the turning point.

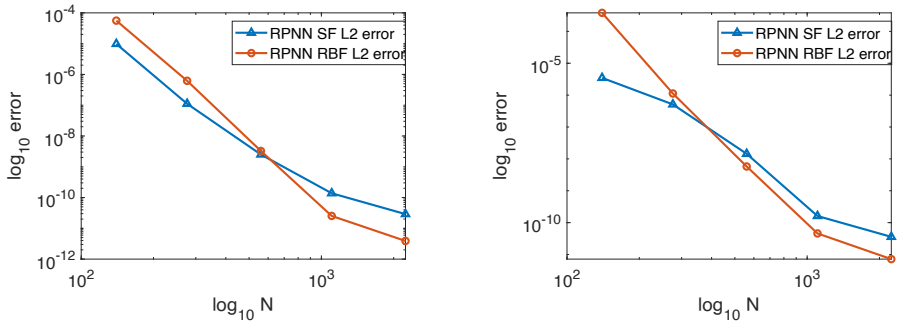



Figure 4.8: Numerical accuracy of RPNNs for the radial two-dimensional Gelfand-Bratu problem (4.42). $L_2$–norm of differences of the analytical solutions (4.43) w.r.t. the number of neurons $N$ in RPNNs with both logistic SF (4.13) and Gaussian RBF (4.14): (a) $\lambda = 1/2$, (b) $\lambda = 1$.

turning points. For our demonstrations, we considered two celebrated classes of nonlinear PDEs whose solutions bifurcate as parameter values change: the one-dimensional viscous Burgers equation (a fundamental representative of advection-diffusion PDEs) and the one- and two-dimensional Liouville–Bratu–Gelfand equation (a fundamental representative of reaction-diffusion PDEs). By coupling the proposed numerical scheme with Newton-Raphson iterations and the "pseudo" arc-length continuation method, we constructed the corresponding bifurcation diagrams past turning points. The efficiency of the proposed numerical ML collocation method was compared against two of the most established numerical solution methods, namely central FD and Galerkin FEM. By doing so, we showed that (for the same problem size) the proposed ML approach

outperforms FD and FEM schemes for relatively medium to large sizes of the grid, both w.r.t. the accuracy of the computed solutions for a wide range of the bifurcation parameter values and the approximation accuracy of the turning points. Thus, we show that the computational times of the proposed ML method are comparable with the ones obtained with the other two schemes (actually the computational times obtained with the proposed method are slightly smaller than the ones obtained with FEM; the FD scheme provides slightly smaller times but fails to approximate solutions with steep gradients and in general results to poorer numerical approximations). Hence, the proposed method arises as an alternative and powerful new numerical technique for the approximation of steady-state solutions of non-linear PDEs. Furthermore, its implementation is far simpler than the implementation of FEM, thus providing equivalent or even better numerical accuracy, and in all cases is shown to outperform the simple FD scheme, which fails to approximate steep gradients as here arise near the boundaries. Of course there are many open problems linked to the implementation of the proposed numerical method that ask for further and deeper investigation, such as the theoretical investigation of the impact of the type of transfer functions and the probability distribution of their parameter values functions to the approximation of the solutions. Further directions could be towards the extension of the method for the solution of time-dependent non-linear PDEs as well as the solution of inverse-problems in PDEs.

# 5 Solution of the Forward Problems II: stiff ODEs and index-1 DAEs

In this chapter, we address the solution of time-dependent forward problems, focusing on the challenges posed by stiff ODEs and index-1 DAEs. We introduce a novel PINN approach using RPNNs with RBF activations. Our method incorporates elements from numerical analysis, including a time-adaptive scheme and continuation-based weight initialization across intervals, and optimizes the sampling bounds of weight distributions. Six benchmark problems are presented to illustrate the effectiveness of this approach in handling stiffness and high-dimensional dynamics in complex systems.

## 5.1 Description of the problem

Here, we consider initial-value problems (IVPs) of ODEs and index-1 DAEs that may also arise from the spatial discretization of PDEs using for example FD, FEM and spectral methods. In particular, we consider IVPs in the linear implicit form of:

$$\boldsymbol{M}\frac{d\boldsymbol{u}(t)}{dt} = \boldsymbol{f}(t,\boldsymbol{u}(t)), \qquad \boldsymbol{u}(0) = \boldsymbol{z}. \tag{5.1}$$

$\boldsymbol{u} \in \mathbb{R}^m$ denotes the set of the states $\{u_1, u_2, \ldots, u_i, \ldots, u_m\}$, $\boldsymbol{M} \in \mathbb{R}^{m\times m}$ is the so-called mass matrix with elements $M_{ij}$, $\boldsymbol{f} : D \subseteq \mathbb{R}\times\mathbb{R}^m \to \mathbb{R}^m$ denotes a Lipschitz continuous multivariate function, with components $f_i(t, u_1, u_2, \ldots, u_m)$ defined in a closed domain $D$, and $\boldsymbol{z} \in \mathbb{R}^m$ are the initial conditions. When $\boldsymbol{M} = \boldsymbol{I}$, the system reduces to the canonical form. The above formulation includes problems of DAEs when $\boldsymbol{M}$ is a singular matrix, including semi-explicit DAEs in the form [104]:

$$\begin{aligned} &\frac{d\boldsymbol{u}(t)}{dt} = \boldsymbol{f}(t,\boldsymbol{u}(t),\boldsymbol{v}(t)), \quad \boldsymbol{u}(0) = \boldsymbol{z}, \\ &\boldsymbol{0} = \boldsymbol{g}(t,\boldsymbol{u}(t),\boldsymbol{v}(t)), \end{aligned} \tag{5.2}$$

where now $\boldsymbol{f} : \mathbb{R}\times\mathbb{R}^{m-l}\times\mathbb{R}^l \to \mathbb{R}^{m-l}$), $\boldsymbol{g} : \mathbb{R}\times\mathbb{R}^{m-l}\times\mathbb{R}^l \to \mathbb{R}^l$ and we assume that the Jacobian $\nabla_{\boldsymbol{v}}\boldsymbol{g}$ is non-singular. In this work, we use PIRPNNs for the

numerical solution of the above type of IVPs which solutions are characterized both by sharp gradients and stiffness [99, 104]. We review the concept of stiffness in the next paragraph.

**Stiffness.** Stiffness, a concept of utmost importance in time integration, is particularly relevant in complex system simulations. Due to their intricate interactions and diverse components, these systems often exhibit a wide range of time scales, leading to stiff ODEs/PDEs. There is not a general definition of stiffness, yet as Shampine and Gear observed, "stiff problems are the ones which integration *with a code that aims at nonstiff problems* proves conspicuously inefficient for no obvious reason" [99]. This inefficiency can arise from several factors, including: (a) the rapid decay of certain solution components can impose stringent limitations on the time step size. To maintain numerical stability, the time step must be sufficiently small to accurately capture the fastest-decaying components; (b) some components of the solution, which may have significantly different concentrations or magnitudes, despite having very large reaction rates, decay much more rapidly than others, as often occurs in chemical reactions. (c) discretizing PDEs can often introduce stiffness into the resulting system of ODEs. This is particularly true when dealing with complex geometries or non-linear terms. Finally, at this point, it is worthy to emphasize that stiffness is not connected to the presence of steep gradients. For example, at the regimes where the relaxation oscillations of the van der Pol model exhibit very sharp changes resembling discontinuities, the equations are not stiff [99].

## 5.2 Classical Numerical Analysis Approaches for ODEs

Numerical methods for solving Ordinary Differential Equations (ODEs) are designed to approximate the solution of IVPs of the form:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0, \tag{5.3}$$

where $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ is a continuous function. The exact solution $y(t)$ is generally not available in closed form, necessitating the use of numerical approximations. While Differential-Algebraic Equations (DAEs) often some more sophisticated solvers, this section will focus on classical numerical methods for simple ODEs.

### 5.2.1 Runge-Kutta Methods

Runge-Kutta (RK) methods are a family of iterative techniques for approximating the solution of ODEs. Developed by Carl Runge and Wilhelm Kutta in the early 20th century, these methods are used extensively for temporal discretization in various applications.

The core idea behind RK methods is to approximate the integral in the Picard-Lindelöf theorem, which expresses the solution of an ODE as an integral equation:

$$y(t) = y_0 + \int_{t_0}^{t} f(s, y(s))\, ds. \tag{5.4}$$

The RK methods replace this integral with a weighted sum of function evaluations at specific points within the interval $[t_n, t_{n+1}]$. In particular, polynomial interpolation is used to approximate the vector field $f$ and integrate the polynomial with quadrature rules.

The general form of an explicit $s$-stage RK method is given by:

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i, \quad k_i \quad = f\left(t_n + c_i h, y_n + h\sum_{j=1}^{i-1} a_{ij} k_j\right), \tag{5.5}$$

where $h$ denotes the step size, and $b_i$, $c_i$, and $a_{ij}$ are coefficients specific to each method, defining the weightings and the intermediate time points. The $k_i$ terms represent the stage derivatives calculated during each step.

### *Euler's Method*

Euler's method is the simplest form of a RK method, characterized by a single stage ($s = 1$). Its update formula is:

$$y_{n+1} = y_n + hf(t_n, y_n). \tag{5.6}$$

This method uses a linear approximation of the solution and corresponds to the left endpoint rule for approximating the integral. It has a local truncation error of $\mathcal{O}(h^2)$ and is only first-order accurate globally.

### *Second-Order Runge-Kutta (RK2)*

The second-order RK2 method, commonly known as the midpoint method, employs two stages ($s = 2$) to achieve higher accuracy. It is defined by:

$$\begin{aligned} k_1 &= f(t_n, y_n), \\ k_2 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right), \\ y_{n+1} &= y_n + hk_2. \end{aligned} \tag{5.7}$$

This method approximates the slope at the midpoint of the interval and has a local truncation error of $\mathcal{O}(h^3)$.

### *Fourth-Order Runge-Kutta (RK4)*

The fourth-order explicit RK4 is one of the most widely used methods due to its balance between accuracy and computational cost. It is defined as follows:

$$\begin{aligned} k_1 &= f(t_n, y_n), \\ k_2 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right), \\ k_3 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right), \\ k_4 &= f(t_n + h, y_n + hk_3), \\ y_{n+1} &= y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4). \end{aligned} \tag{5.8}$$

Here, $k_1$ represents the slope at the beginning of the interval, $k_2$ and $k_3$ are the slopes at the midpoint, and $k_4$ is the slope at the endpoint. This method achieves a local truncation error of $\mathcal{O}(h^5)$ and a global error of $\mathcal{O}(h^4)$. Note that, in averaging the four slopes, greater weight is given to the slopes at the midpoint. If $f$ is independent of $y$, so that the DE is equivalent to a simple integral, then RK4 correspond to the Simpson's quadrature rule.

**Adaptive Step Size Control.** In practical applications, maintaining a fixed step size may not be efficient or accurate enough. Therefore, adaptive methods are designed to produce an estimate of the local truncation error of a single Runge–Kutta step. This is done by having two methods, one with order $p$ and one with order $p-1$. These methods have common intermediate steps. Thanks to this, estimating the error has little or negligible computational cost compared to a step with the higher-order method.

Adaptive methods, such as the Dormand-Prince method (DOPRI45), are designed to dynamically adjust the step size based on an error estimate. The Dormand-Prince method is a fifth-order method with an embedded fourth-order error estimate, commonly implemented in MATLAB as `ode45`.

The adaptive step size strategy aims to ensure that the local error $e_n$ satisfies:

$$\|e_n\| \approx \epsilon, \tag{5.9}$$

where $\epsilon$ is a user-defined tolerance. If the estimated error is too large, the step size $h$ is decreased; if it is too small, $h$ is increased, thereby optimizing the computational effort while maintaining the desired accuracy.

Adaptive methods use two sets of coefficients $b_i$ and $b_i^*$ to compute two estimates of $y_{n+1}$ as follows:

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i, \quad y_{n+1}^* = y_n + h\sum_{i=1}^{s} b_i^* k_i, \tag{5.10}$$

where $k_i$ are the same stage in both the two methods. The error is then estimated as:

$$e_n = y_{n+1} - y^*_{n+1} = h \sum_{i=1}^{s} (b_i - b^*_i) k_i. \tag{5.11}$$

This error estimate, which is $O(h^p)$, is used to adaptively control the step size, ensuring that the integration process remains efficient and accurate.

**Implicit Runge-Kutta schemes.** All RK methods mentioned up to now are explicit methods. Explicit RK methods are generally unsuitable for the solution of stiff equations because their region of absolute stability is small; in particular, it is bounded. This issue is especially important in the solution of PDEs. A generic $s$-stages implicit-RK scheme can be written as:

$$\begin{aligned} k_1 &= f(t_0 + c_1 h, y_0 + h \sum_{\ell=1}^{s} a_{1\ell} k_\ell) \\ &\vdots \\ k_s &= f(t_0 + c_s h, y_0 + h \sum_{\ell=1}^{s} a_{s\ell} k_\ell) \end{aligned} \tag{5.12}$$

where $k_1, \ldots, k_s$ satisfy the above set of nonlinear equations. The updated solution $y_1$ is then computed as follows:

$$y_1 = y_0 + h \sum_{i=1}^{s} b_i k_i. \tag{5.13}$$

The coefficients defining an implicit RK method can be organized compactly in a Butcher tableau:

$$\begin{array}{c|c} \boldsymbol{c} & A \\ \hline & \boldsymbol{b}^T \end{array} = \begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ c_2 & a_{21} & \cdots & a_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array} \tag{5.14}$$

Note that in contrast to explicit methods, the matrix $A$ is a general matrix not required to be strictly lower triangular.

The consequence of this difference is that at every step, a system of algebraic equations has to be solved. This increases the computational cost considerably. If a method with s stages is used to solve a DE with m components, then the system of algebraic equations has ms components. This can be contrasted with implicit linear multistep methods (the other big family of methods for ODEs): an implicit s-step linear multistep method needs to solve a system of algebraic equations with only m components, so the size of the system does not increase as the number of steps increases.

The three simplest examples of implicit RK methods are the implicit Euler, midpoint and trapezoidal methods, that have the following tableaux:

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}, \qquad \begin{array}{c|c} 1/2 & 1/2 \\ \hline & 1 \end{array}, \qquad \begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1/2 & 1/2 \\ \hline & 1/2 & 1/2 \end{array}, \tag{5.15}$$

respectively. For more accurate higher order scheme, one can combine collocation with numerical quadrature. For example, using Gauss or Radau quadrature scheme. Here we consider the 2-stages Gauss, 4th order, A-stable Gauss method with the following array of butcher:

$$\begin{array}{c|cc} \frac{1}{2}-\frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{1}{4}-\frac{\sqrt{3}}{6} \\ \frac{1}{2}+\frac{\sqrt{3}}{6} & \frac{1}{4}+\frac{\sqrt{3}}{6} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \tag{5.16}$$

We also consider in our comparison, the A-stable, 2-stages (IIA) and 3-stages (IA) Radau schemes that have the following Butcher tableaux:

$$\begin{array}{c|cc} \frac{1}{3} & \frac{5}{12} & -\frac{1}{12} \\ 1 & \frac{3}{4} & \frac{1}{4} \\ \hline & \frac{3}{4} & \frac{1}{4} \end{array}, \qquad \begin{array}{c|ccc} 0 & \frac{1}{9} & \frac{-1-\sqrt{6}}{18} & \frac{-1+\sqrt{6}}{18} \\ \frac{6-\sqrt{6}}{10} & \frac{1}{9} & \frac{88+7\sqrt{6}}{360} & \frac{88-43\sqrt{6}}{360} \\ \frac{6+\sqrt{6}}{10} & \frac{1}{9} & \frac{88+43\sqrt{6}}{360} & \frac{88-7\sqrt{6}}{360} \\ \hline & \frac{1}{9} & \frac{16+\sqrt{6}}{36} & \frac{16-\sqrt{6}}{36} \end{array}. \tag{5.17}$$

**Stability of Runge–Kutta Methods.** Implicit RK methods are preferred over explicit methods for their superior stability, particularly when solving stiff equations. To illustrate this, consider the linear test equation:

$$y' = \lambda y. \tag{5.18}$$

Applying a RK method to this equation results in the iterative formula:

$$y_{n+1} = r(h\lambda)y_n, \tag{5.19}$$

where the stability function $r(z)$ is defined as:

$$r(z) = 1 + zb^T(I - zA)^{-1}e = \frac{\det(I - zA + zeb^T)}{\det(I - zA)}, \tag{5.20}$$

with $z = h\lambda$ and $e$ representing the vector of ones. Here, $A$, $b$, and $c$ are the coefficients that characterize the specific RK method. The stability function $r(z)$ determines the behavior of the numerical solution in response to the linear test equation.

For a method with $s$ stages, $r(z)$ is the ratio of two polynomials of degree $s$. In the case of explicit methods, the matrix $A$ is strictly lower triangular, which simplifies the determinant $\det(I - zA)$ to 1. Consequently, the stability function $r(z)$ for explicit methods is always a polynomial.

The numerical solution $y_n$ decays to zero if $|r(z)| < 1$, with $z = h\lambda$. The set of all $z$ values satisfying this condition is known as the domain of absolute stability. A method is termed *absolutely stable* if all $z$ with $\mathrm{Re}(z) < 0$ are within this domain. Since the stability function of explicit RK methods is a polynomial, these methods can never be $A$-stable, meaning they cannot maintain stability for all $\mathrm{Re}(z) < 0$.

In contrast, implicit RK methods have a more complex stability function, allowing them to achieve $A$-stability and thus handle stiff problems more effectively.

### 5.2.2 Linear multi-step methods

Linear multistep methods leverage information from the previous $s$ steps to approximate the solution at the current step. These methods are defined as:

$$\sum_{j=0}^{s} a_j y_{n+j} = h \sum_{j=0}^{s} b_j f(t_{n+j}, y_{n+j}), \tag{5.21}$$

where $a_j$ and $b_j$ are coefficients, $h$ is the step size, and $s$ is the number of previous steps used in the method. The values of $y_n$ and $f(t_n, y_n)$ are linearly combined to compute $y_{n+s}$. The method is termed explicit if $b_s = 0$ and implicit if $b_s \neq 0$. Implicit methods typically require solving a nonlinear equation at each step, often using iterative techniques like Newton's method.

Three prominent families of linear multistep methods are the Adams–Bashforth methods, the Adams–Moulton methods, and the BDF.

**Adams–Bashforth Methods.** The Adams–Bashforth methods are a class of explicit linear multistep methods. They are defined by the coefficients $a_{s-1} = -1$ and $a_j = 0$ for $j < s - 1$. The coefficients $b_j$ are selected to ensure that the method achieves an order of accuracy $s$. This uniquely determines the coefficients of the method.

**Adams–Moulton Methods.** The Adams–Moulton methods share similarities with the Adams–Bashforth methods in terms of the coefficients $a_j$, where $a_{s-1} = -1$ and $a_j = 0$ for $j < s - 1$. However, these methods are implicit, allowing the coefficient $b_s$ to be non-zero. This flexibility enables an $s$-step Adams–Moulton method to achieve an order of $s + 1$, whereas an $s$-step Adams–Bashforth method only attains an order of $s$.

**Backward Differentiation Formulas.** The BDF methods are implicit linear multistep methods designed for stiff DEs. In these methods, $b_j = 0$ for all $j < s$, and the remaining coefficients are chosen to achieve the maximum possible order $s$. BDF methods are particularly useful for solving stiff problems due to their favorable stability properties.

The MATLAB solver 'ode15s' is a popular implementation of a variable-order BDF method. It adjusts the order and step size dynamically to maintain a balance between computational efficiency and accuracy. The solver uses orders ranging from 1 to 5, automatically selecting the most appropriate order and step size based on error estimates.

**Dahlquist Barriers.** The convergence properties of linear multistep methods are constrained by the Dahlquist barriers:

1. First Dahlquist Barrier: A zero-stable $q$-step linear multistep method cannot exceed an order of $q+1$ if $q$ is odd and $q+2$ if $q$ is even. For explicit methods, the maximum attainable order is $q$.

2. Second Dahlquist Barrier: No explicit linear multistep method is $A$-stable. Moreover, the maximum order for an implicit $A$-stable linear multistep method is 2. Among these, the trapezoidal rule has the lowest error constant for methods of order 2.

These barriers underscore the limitations in designing high-order, stable multistep methods, particularly for stiff problems.

## 5.3 The Proposed Physics-Informed Random Projection Neural Network for the solution of ODEs and index-1 DAEs.

Here, we propose a PIML scheme based on the concept of random projections, and particularly based on Theorem 1 (see also [203, 159, 160]) for the solution of IVPs of systems given by Eq.(5.1)-(5.2) in $n$ collocation points in an interval, say $[t_0 \quad t_f]$. Based on the above, the output of the random projection network is spanned by the range $\mathcal{R}(\boldsymbol{\Phi})$, i.e., the column vectors of $\boldsymbol{\Phi}_N$, say $\boldsymbol{\phi}_i \in \mathbb{R}^n$. Hence, the output of the network can be written as:

$$\boldsymbol{Y}_N = \sum_{i=1}^{N} w_i^o \boldsymbol{\phi}_i \tag{5.22}$$

For an IVP of $m$ variables, we construct $m$ such networks.

Let's denote by $\boldsymbol{\Psi}(t, \boldsymbol{W}, \boldsymbol{W}^o, \boldsymbol{P})$ the set of functions $\Psi_{Ni}(t, \boldsymbol{w}_i^o, \boldsymbol{p}_i), i = 1, 2, \ldots, m$ that approximate the solution profile $u_i$ at time $t$, defined as:

$$\Psi_{Ni}(t, \boldsymbol{w}_i, \boldsymbol{w}_i^o, \boldsymbol{p}_i) = z_i + (t - t_0)\boldsymbol{w}_i^{oT} \boldsymbol{\Phi}_{Ni}(t, \boldsymbol{w}_i, \boldsymbol{p}_i), \tag{5.23}$$

where $\boldsymbol{\Phi}_{Ni}(t, \boldsymbol{w}_i, \boldsymbol{p}_i) \in \mathbb{R}^N$ is the column vector containing the values of the $N$ basis functions at time $t$ as shaped by $\boldsymbol{w}_i$ and $\boldsymbol{p}_i$ containing the values of the parameters of the $N$ basis functions and $\boldsymbol{w}_i^o = [w_{1i}^o \; w_{2i}^o \; \ldots \; w_{Ni}^o]^T \in \mathbb{R}^N$ is the vector containing the

values of the output weights of the $i$-th network. Note that the above set of functions are continuous functions of $t$ and satisfy explicitly the initial conditions.

For index-1 DAEs, with say $M_{ij} = 0, \forall i \geq l,\ j = 1, 2, \ldots, m$, or in the semi-explicit form of (5.2), there are no explicit initial conditions $z_i$ for the variables $u_i$, $i = l, l+1, \ldots, m$, or the variables $\boldsymbol{v}$ in (5.2): these values have to satisfy the constraints $f_i(t, \boldsymbol{u}) = 0,\ i \geq l$, (equivalently $\mathbf{0} = \boldsymbol{g}(t, \boldsymbol{u}, \boldsymbol{v})$) $\forall t$, starting with consistent initial conditions. Assuming that the corresponding Jacobian matrix of the $f_i(t, \boldsymbol{u}) = 0$, $i = l, l+1, \ldots, m$ w.r.t. $u_i$, and for the semi-explicit form (5.2), $\nabla_{\boldsymbol{v}} \boldsymbol{g}$, is not singular, one has to solve initially at $t = 0$, using for example Newton-Raphson iterations, the above nonlinear system of $m - l$ algebraic equations in order to find a consistent set of initial values. Then, one can write the approximation functions of the $u_i$, $i = l, l+1, \ldots, m$ (or $\boldsymbol{v}$ in the case of semi-explicit form (5.2)) as in Eq. (5.23).

With $n$ collocation points in $[t_0 \quad t_f]$, by fixing the values of the interval weights $\boldsymbol{w}_i$ and the shape parameters $\boldsymbol{p}_i$, the loss function that we seek to minimize w.r.t. the unknown coefficients $\boldsymbol{w}_i^o$ is given by:

$$\mathcal{L}(\boldsymbol{W}^o) = \sum_{k=1}^{n} \sum_{i=1}^{m} \sum_{j=1}^{m} \left( M_{ij} \frac{d\Psi_{Ni}}{dt}(t_k, \boldsymbol{w}_i, \boldsymbol{w}_i^o, \boldsymbol{p}_i) - f_i\big(t_k, \boldsymbol{\Psi}(t_k, \boldsymbol{W}, \boldsymbol{W}^o, \boldsymbol{P})\big) \right)^2. \tag{5.24}$$

When the system of ODEs/DAEs results from the spatial discretization of PDEs, we assume that the corresponding boundary conditions have been appropriately incorporated into the resulting algebraic equations explicitly or otherwise can be added in the loss function as algebraic constraints. Here, for each $\mathcal{N}_i$, and for $j = 1, \ldots, N$, $i = 1, \ldots, m$, we take $N$ Gaussian kernels given by:

$$g_{ji}(t, w_{ji}, b_{ji}, \alpha_{ji}, c_j) = e^{-\alpha_{ji}(w_{ji}t + b_{ji} - c_j)^2}. \tag{5.25}$$

The values of the (hyper) parameters, namely $w_{ji}$, $b_{ji}$, and $c_j$ are set as:

$$w_{ji} = 1, \quad b_{ij} = 0, \quad c_j = t_j = t_0 + (j-1)\frac{t_f - t_0}{N-1}, \tag{5.26}$$

while the values of the shape parameters $\alpha_{ji} > 0$ are sampled from an appropriately chosen uniform distribution (see below). The time derivative of $\Psi_{Ni}$ is given by:

$$\frac{d\Psi_{Ni}}{dt} = \sum_{j=1}^{N} w_{ji}^o e^{-\alpha_{ji}(t-t_j)^2} - 2(t - t_0) \sum_{j=1}^{N} \alpha_{ji} w_{ji}^o (t - t_j) e^{-\alpha_{ji}(t-t_j)^2}. \tag{5.27}$$

### 5.3.1 Approximation with the PIRPNN

Here, we show that the PIRPNN given by Eq.(5.23) is a universal approximator of the solution $\boldsymbol{u}$ of the ODEs in canonical form or of the index-1 DAEs in the semi-explicit form (5.2) [95].

**Theorem 5.3.1.** For the IVP problem (5.1) in the canonical form or in the semi-explicit form (5.2) for which the Picard-Lindelöf Theorem [234] holds true, the PIRPNN solution $\Psi_{Ni}$ given by Eq.(5.23) with $N$ Gaussian basis functions defined by Eq.(5.25) whose shape parameters $\alpha_{ji}$ are drawn i.i.d. from a uniform distribution across the sample space, converges uniformly to the actual solution profile $\boldsymbol{u}(t)$ in a closed time interval $[t_0 \quad t_f]$ with an upper bound of the order of $O(\frac{1}{\sqrt{N}})$ with a probability $1-\delta$ for any small $\delta > 0$.

*Proof.* Assuming that the system in Eq. (5.1) can be written in the canonical form and the Picard-Lindelöf Theorem [234] holds true, then it exists a unique continuously differentiable function defined on a closed time interval $[t_0 \quad t_f]$ given by:

$$u_i(t) = z_i + \int_{t_0}^{t} f_i(s, \boldsymbol{u}(s))ds, \quad i = 1, 2, \dots m. \tag{5.28}$$

From Eq.(5.23), we have:

$$\Psi_{Ni}(t) = z_i + (t - t_0) \sum_{j=1}^{N} w_j^o e^{-\alpha_j (t-t_j)^2}. \tag{5.29}$$

By the change of variables, $\tau = \dfrac{s - t_0}{t - t_0}$, the integral in Eq.(5.28) becomes

$$\int_{t_0}^{t} f_i(s, \boldsymbol{u}(s))ds = (t - t_0) \int_{0}^{1} f_i(\tau(t - t_0) + t_0, \boldsymbol{u}(\tau(t - t_0) + t_0))d\tau. \tag{5.30}$$

Hence, by Eqs.(5.28),(5.29),(5.30), we have:

$$I_n(t) \equiv \int_{0}^{1} f_i(\tau(t - t_0) + t_0, \boldsymbol{u}(\tau(t - t_0) + t_0))d\tau \approx \sum_{j=1}^{N} w_j^o e^{-\alpha_j (t-t_j)^2}. \tag{5.31}$$

Thus, in fact, upon convergence, the PIRPNN provides an approximation of the normalized integral. By Theorem 2.1, we have that in the interval $[t_0 \quad t_f]$, the PIRPNN with the shape parameter of the Gaussian kernel drawn i.i.d. from a uniform distribution provides, a uniform approximation of the integral in Eq.(5.28) in terms of a Monte Carlo integration method as also described in [196]. Hence, as the initial conditions are explicitly satisfied by $\Psi_{Ni}(t)$, we have from Eq.(2.39) an upper bound for the uniform approximation of the solution profile $u_i$.

For index-1 DAEs in the semi-explicit form of (5.2), by the implicit function theorem, we have that the DAE system is *in principle* equivalent with the ODE system in the canonical form:

$$\frac{d\boldsymbol{u}(t)}{dt} = \boldsymbol{f}(t, \boldsymbol{u}(t), \mathcal{H}(t, \boldsymbol{u})), \tag{5.32}$$

where $\boldsymbol{v}(t) = \mathcal{H}(t, \boldsymbol{u}(t))$ is the unique solution of $\mathbf{0} = \boldsymbol{g}(t, \boldsymbol{u}(t), \boldsymbol{v}(t))$. Hence, in that case, the proof of convergence reduces to the one above for the ODE system in the canonical form. □

### *Computation of the unknown weights*

For $n$ collocation points, the outputs of each network $\mathcal{N}_i \equiv \mathcal{N}_i(t_1, t_2, \dots t_n, \boldsymbol{w}_i^o, \boldsymbol{p}_i) \in \mathbb{R}^n$, $i = 1, 2, \dots m$ read:

$$\mathcal{N}_i = \boldsymbol{R}_i \boldsymbol{w}_i^o, \quad \boldsymbol{R}_i \equiv \boldsymbol{R}_i(t_1, \dots, t_n, \boldsymbol{p}_i) = \begin{bmatrix} g_{1i}(t_1) & \cdots & g_{Ni}(t_1) \\ \vdots & \vdots & \vdots \\ g_{1i}(t_n) & \cdots & g_{Ni}(t_n) \end{bmatrix}. \tag{5.33}$$

The minimization of the loss function (5.24) is performed over the $nm$ nonlinear residuals $F_q$:

$$F_q(\boldsymbol{W}^o) = \sum_{j=1}^{m} M_{ij} \frac{d\Psi_{Nj}}{dt_l}(t_l, \boldsymbol{w}_j^o) - f_i(t_l, \Psi_{N1}(t_l, \boldsymbol{w}_1^o), \dots, \Psi_{Nm}(t_l, \boldsymbol{w}_m^o)), \tag{5.34}$$

where $q = l + (i-1)n$, $i = 1, 2, \dots m$, $l = 1, 2, \dots n$, $\boldsymbol{W}^0 \in \mathbb{R}^{mN}$ is the column vector obtained by collecting the values of all $m$ vectors $\boldsymbol{w}_i^o \in \mathbb{R}^N$, $\boldsymbol{W}^o = [W_k^o] = [\boldsymbol{w}_1^o, \boldsymbol{w}_2^o \dots, \boldsymbol{w}_m^o]^T$, $k = 1, 2, \dots, mN$. Thus, the solution to the above non-linear least squares problem can be obtained, e.g., with Newton-type iterations such as Newton-Raphson, quasi-Newton and Gauss-Newton methods (see, e.g., [235]). For example, by setting $\boldsymbol{F}(\boldsymbol{W}^o) = [F_1(\boldsymbol{W}^o) \cdots F_q(\boldsymbol{W}^o) \cdots F_{(nm)}(\boldsymbol{W}^o)]^T$, the update $d\boldsymbol{W}^{o(\nu)}$ at the $(\nu)$-th Gauss-Newton iteration is computed by the solution of the linearized system:

$$\nabla_{\boldsymbol{W}^{o(\nu)}} \boldsymbol{F} d\boldsymbol{W}^{o(\nu)} = -\boldsymbol{F}(\boldsymbol{W}^{o(\nu)}), \tag{5.35}$$

where $\nabla_{\boldsymbol{W}^{o(\nu)}} \boldsymbol{F} \in \mathbb{R}^{nm \times mN}$ is the Jacobian matrix of $\boldsymbol{F}$ w.r.t. $\boldsymbol{W}^{o(\nu)}$. Note that the residuals depend on the derivatives $\frac{\partial \Psi_{Ni}(\cdot)}{\partial t_l}$ and the approximation functions $\Psi_{Ni}(\cdot)$, while the elements of the Jacobian matrix depend on the derivatives of $\frac{\partial \Psi_{Ni}(\cdot)}{\partial w_{ji}^o}$ as well as on the mixed derivatives $\frac{\partial^2 \Psi_{Ni}(\cdot)}{\partial t_l \partial w_{ji}^o}$. Based on (5.27), the latter are given by

$$\frac{\partial^2 \Psi_{Ni}}{\partial t_l \partial w_{ji}^o} = \frac{\partial \mathcal{N}_i(t_l, \boldsymbol{w}_i^o, \boldsymbol{p}_i)}{\partial w_{ji}^o} - 2(t_l - t_0)\alpha_{ji}(t_l + b_{ji} - c_j) e^{\left(-\alpha_{ji}(t_l + b_{ji} - c_j)^2\right)}. \tag{5.36}$$

Thus, the elements of $\nabla_{\boldsymbol{W}^{o(\nu)}} \boldsymbol{F}$, for $q = l + (i-1)n$, $p = j + (k-1)N$, are given by:

$$\frac{\partial F_q}{\partial W_p^o} = \frac{\sum_{j=1}^{m} M_{ij} \partial^2 \Psi_{Ni}(\cdot)}{\partial t_l \partial w_{jk}^o} - \frac{\partial f_i(t_l)}{\partial w_{jk}^o}. \tag{5.37}$$

However, even when $N \geq n$, the Jacobian matrix is expected to be rank deficient, or nearly rank deficient, since some rows due to the random construction of the basis

functions can be nearly linear dependent [203]. Thus, the solution of the corresponding system, and depending on the size of the problem, can be solved using for example tSVD or QR factorization with regularization. The truncated SVD decomposition scheme leads to the Moore-Penrose pseudo-inverse and the updates $\boldsymbol{dW}^{o(\nu)}$ are given by:

$$\boldsymbol{dW}^{o(\nu)} = -(\nabla_{\boldsymbol{W}^{o(\nu)}}\boldsymbol{F})^{\dagger}\boldsymbol{F}(\boldsymbol{W}^{o(\nu)}), \quad (\nabla_{\boldsymbol{W}^{o(\nu)}}\boldsymbol{F})^{\dagger} = \boldsymbol{V}_{\epsilon}\boldsymbol{\Sigma}_{\epsilon}^{\dagger}\boldsymbol{U}_{\epsilon}^{T}, \tag{5.38}$$

where $\boldsymbol{\Sigma}_{\epsilon}^{\dagger}$ is the inverse of the diagonal matrix with singular values of $\nabla_{\boldsymbol{W}^{o}}\boldsymbol{F}$ above a certain threshold $\epsilon$, and $\boldsymbol{U}_{\epsilon}$, $\boldsymbol{V}_{\epsilon}$ are the matrices with columns the corresponding left and right eigenvectors, respectively. At this point, in order to decrease the computational cost, one can implement a Quasi-Newton scheme, thus using the same pseudo-inverse of the Jacobian for the next iterations until convergence.

For large-scale sparse Jacobian matrices, as those arising for example from the discretization of PDEs, one can solve the regularization problem using other methods such as sparse QR factorization. Here, to account for the ill-posed Jacobian, we have used a sparse QR factorization with regularization as implemented by `SuiteSparseQR`, a multithreaded sparse QR factorization package [236].

To summarize, the above scheme provides a *numerical analysis-assisted* PINN in a form that approximates the integral solution of the Picard–Lindelöf Theorem based on random projections, thus providing analytically the Jacobian matrix for the Newton's iterations.

## 5.4 Parsimonious construction of the PIRPNN

### 5.4.1 The adaptive step-size scheme

In order to deal with the presence of stiffness and sharp changes that resemble singularities at the time interval of interest, we propose an adaptive step-size scheme for adjusting the interval of integration as follows. The full-time interval of integration $[t_0 \quad t_f]$ is divided into sub-intervals, i.e., $[t_0 \quad t_f] = [t_0 \quad t_1] \cup [t_1 \quad t_2] \cup \ldots, \cup [t_k \quad t_{k+1}] \cup \cdots \cup [t_{end-1} \quad t_f]$, where $t_1, t_2, \ldots, t_k, \ldots, t_{end-1}$ are determined in an adaptive way. This decomposition of the interval leads to the solution of consecutive IVPs. Let's assume that the problem has been solved up to $[t_{k-1} \quad t_k]$, hence we have found $u_i^{(k-1)}$ and we are seeking to advance $u_i^{(k)}$ to the next interval, say, $[t_k \quad t_{k+1}]$ with a step size of $\Delta t_k = t_{k+1} - t_k$.

At each Newton's iteration, say $\nu$ (here $\nu \leq \nu_{max} = 5$), we compute the *normalized residuals (precision to tolerance ratio)* $res_q^{(\nu)}$, for each component $F_q(\boldsymbol{W}^{o(\nu)})$ of $\boldsymbol{F}(\boldsymbol{W}^{o(\nu)})$, as: [98, 237]

$$res_q^{(\nu)} = \frac{F_q(\boldsymbol{W}^{o(\nu)})}{AbsTol + RelTol \cdot \dfrac{d\Psi_{Ni}}{dt_l}(t_l, \boldsymbol{w}_i^{o(\nu)})} \tag{5.39}$$

where $AbsTol$ is the absolute threshold tolerance, $RelTol$ is the tolerance relative to the size of each derivative component at time $t_l$ and, as in Eq. 5.34, $q = l + (i-1)n$, $i =$

$1, 2, \dots m$, $l = 1, 2, \dots n$. Thus, we compute the *approximation error*, say $err^{(\nu)}$, as the $l^2$-norm of the vector of the normalized residuals $\boldsymbol{res}^{(\nu)} = [res_1^{(\nu)}, res_2^{(\nu)}, \dots, res_{nm}^{(\nu)}]$:

$$err^{(\nu)} = \left\| \boldsymbol{res}^{(\nu)} \right\|_{l^2}. \tag{5.40}$$

Now, if at the $\nu$-th iteration, for one $\nu \leq \nu_{max}$, $err^{(\nu)} < 1$ the numerical solution is accepted, otherwise (if up to the last iteration $\nu_{max}$, $err^{(\nu_{max})} \geq 1$) the numerical solution is rejected.

In both cases, the size of the time interval is updated according to the elementary local error control algorithm [237]:

$$\Delta t_k^* = 0.8\gamma \cdot \Delta t_k, \qquad \text{with} \qquad \gamma = \left( \frac{1}{err} \right)^{\frac{1}{\nu+1}}, \tag{5.41}$$

where $\gamma$ is the *scaling factor* and $0.8$ is a safe/conservative factor; $\Delta t_k^*$ is not allowed to increase or decrease a lot, so $\gamma$ is not higher than a $\gamma_{max}$ (here set to $4$) and smaller than a $\gamma_{min}$ (here set to $0.1$). Thus, if the Quasi-Newton scheme does not converge to the desired tolerance within a number of iterations, say $\nu_{max}$, then the step size is decreased, thus redefining a new guess $t_{k+1}^* = t_k + \Delta t_k^*$ for $t_{k+1}$ and the Quasi-Newton scheme is repeated in the interval $[t_k \quad t_{k+1}^*]$. Finally, we note that in the above scheme, the choice of the first sub-interval $[t_0 \quad t_1]$ was estimated using an automatic detection code for selecting the starting step as described in [238].

### 5.4.2 A continuation method for Newton's iterations

For Newton-type schemes, the speed of the convergence to the solution depends on the choice of the initial guess, here, for the unknown weights. Here, to facilitate the convergence of Newton's iterations, we address a numerical natural continuation method for providing "good" initial guesses for the weights of the PIRPNN.

Suppose that the algorithm has already converged to the solution in the interval $[t_{k-1} \quad t_k]$; we want to provide for the next time interval $[t_k \quad t_{k+1}]$, as computed from the proposed adaptation scheme described above, a good initial guess for the weights of the PIRPNN. We state the following proposition [95].

**Proposition 5.4.1.** Let $\boldsymbol{\Psi}(t_k) \in \mathbb{R}^m$ be the solution found with PIRPNN at the end of the time interval $[t_{k-1} \quad t_k]$. Then, an initial guess for the weights of the PIRPNN for the time interval $[t_k \quad t_{k+1}]$ is given by:

$$\hat{\boldsymbol{W}}^o = \frac{d\boldsymbol{\Psi}(t_k)}{dt} \frac{\boldsymbol{\Phi}^T}{||\boldsymbol{\Phi}||_{l_2}^2}, \tag{5.42}$$

where $\hat{\boldsymbol{W}}^o \in \mathbb{R}^{m \times N}$ is the matrix with the initial guess of the output weights of the $m$ PIRPNNs and $\boldsymbol{\Phi} \in \mathbb{R}^N$ is the vector containing the values of the random basis functions in the interval $[t_k \quad t_{k+1}]$.

*Proof.* At time $t_k$, a first-order estimation of the solution, $\mathbf{\Psi}(t_{k+1}) \in \mathbb{R}^m$ reads:

$$\hat{\mathbf{\Psi}}(t_{k+1}) = \mathbf{\Psi}(t_k) + \frac{d\mathbf{\Psi}(t_k)}{dt}(t_{k+1} - t_k), \tag{5.43}$$

where $\frac{d\mathbf{\Psi}(t_k)}{dt}$ is known. For the next time interval $[t_k \quad t_{k+1}]$, the approximation of the solution with the PIRPNNs is given by:

$$\mathbf{\Psi}(t_{k+1}) = \mathbf{\Psi}(t_k) + (t_{k+1} - t_k)\boldsymbol{W}^o\mathbf{\Phi}. \tag{5.44}$$

By Eqs.(5.43), (5.44), we get:

$$\hat{\boldsymbol{W}}^o\mathbf{\Phi} = \frac{d\mathbf{\Psi}(t_k)}{dt}. \tag{5.45}$$

It can be easily seen, that the truncated SVD of $\mathbf{\Phi}$ is given by:

$$\mathbf{\Phi}_{N\times 1} = \boldsymbol{U}_{N\times 1}\sigma_1, \quad \boldsymbol{U}_{N\times 1} = \frac{\mathbf{\Phi}_{N\times 1}}{||\mathbf{\Phi}||_{l_2}}, \sigma_1 = ||\mathbf{\Phi}||_{l_2}. \tag{5.46}$$

Thus, the pseudo-inverse of $\mathbf{\Phi}$, is $\mathbf{\Phi}^\dagger = \dfrac{\mathbf{\Phi}^T}{||\mathbf{\Phi}||^2_{l_2}}$. Hence, by Eq.(5.45), an initial guess for the weights for the time interval $[t_k \quad t_{k+1}]$ is given by:

$$\hat{\boldsymbol{W}}^o = \frac{d\mathbf{\Psi}(t_k)}{dt}\mathbf{\Phi}^\dagger = \frac{d\mathbf{\Psi}(t_k)}{dt}\frac{\mathbf{\Phi}^T}{||\mathbf{\Phi}||^2_{l_2}}. \tag{5.47}$$

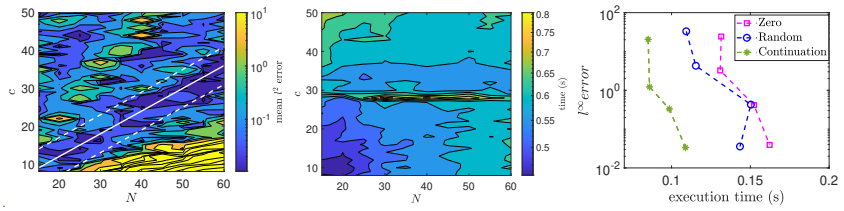
□

### 5.4.3 Estimation of the interval of the uniform distribution based on the variance/bias trade-off decomposition

Based on Eqs.(2.40), (5.22) one has to choose the number $N$ of the basis functions, and the interval, say $\mathcal{U} = [0 \quad \alpha_u]$, from which the values of the shape parameters $\alpha_i$ are drawn based on a probability distribution $p$. The theorems of uniform convergence 2.4.2 and 5.3.1 consider the problem from the function approximation point of view.

Here, we construct $N$ random vectors by parsimoniously sampling the values of the shape parameter from an appropriately bounded uniform interval for minimizing the two sources of error approximation, i.e., the bias and the variance in order to get good generalization properties. In our scheme, these, over all possible values of the shape parameter $\alpha$ are given by (see Eq. (5.29)):

$$\begin{aligned} e(t) &= \mathbb{E}\left[\sum_{j=1}^{N} w_j^o e^{-\alpha_j(t-t_j)^2}\right] - I_n(t), \\ \sigma^2(t) &= \mathbb{E}\left[(\sum_{j=1}^{N} w_j^o e^{-\alpha_j(t-t_j)^2})^2\right] - \mathbb{E}^2\left[\sum_{j=1}^{N} w_j^o e^{-\alpha_j(t-t_j)^2}\right] \end{aligned} \tag{5.48}$$




(a) (b) (c)

Figure 5.1: Numerical solution of the van der Pol ODEs (5.53) with $\mu = 100$ using the PIRPNN in the interval $[0 \quad 3\mu]$ w.r.t. $c$ and $N$ for $n = 20$; $RelTol$ and $AbsTol$ were set to 1e−06. (a) Bias-Variance trade-off loss (5.54) w.r.t. the reference solution as obtained with `ode15s` with $AbsTol$ and $RelTol$ set to 1E−14. (b) Computational times (s). (c) Numerical approximation accuracy (indicatively $l^\infty$ error for $u_2$) vs. execution times with the proposed continuation method (green) and without continuation (thus initializing all weights to zero (magenta) or randomly (blue)).

where $\mathbb{E}$ denotes the expectation operator. In the above, overfitting which is connected with a high variance occurs for large values of $\alpha$ and underfitting, which is connected with a high bias approximation error) occurs for small values of $\alpha$.
The expected value of the kernel $\phi(t - t_j; \alpha) = e^{-a(t-t_j)^2}, t \neq t_j$ w.r.t. the probability density function of the uniform distribution of the random variable $\alpha$ reads:

$$\mathbb{E}[\phi(t - t_j; \alpha)] = \int_0^{\alpha_u} f_\alpha(\alpha) e^{-\alpha(t-t_j)^2} d\alpha = \frac{1 - e^{-\alpha_u (t-t_j)^2}}{\alpha_u (t - t_j)^2}. \tag{5.49}$$

Similarly, the variance is given by:

$$\begin{aligned} \sigma^2[\phi(t - t_j; \alpha)] = \int_{e^{-\alpha_u (t-t_j)^2}}^{1} \phi^2 \frac{1}{\alpha_u (t - t_j)^2} \frac{1}{\phi} d\phi - \mathbb{E}[\phi]^2 = \\ = \frac{1 - e^{-2\alpha_u (t-t_j)^2}}{2\alpha_u (t - t_j)^2} - \mathbb{E}[\phi]^2. \end{aligned} \tag{5.50}$$

At the limits of $t - t_j = dt = \frac{t_f - t_0}{N}$, from Eqs.(5.49), (5.50), we get:

$$\begin{aligned} \mathbb{E}[\phi(dt; \alpha)] &= \frac{N^2}{(t_f - t_0)^2} \frac{1 - e^{-\alpha_u \frac{(t_f - t_0)^2}{N^2}}}{\alpha_u}, \\ \sigma^2[\phi(dt; \alpha)] &= \frac{N^2}{(t_f - t_0)^2} \frac{1 - e^{-2\alpha_u \frac{(t_f - t_0)^2}{N^2}}}{2\alpha_u} - \mathbb{E}[\phi(dt; \alpha)]^2. \end{aligned} \tag{5.51}$$

The above expressions suggest that $\alpha_u = \dfrac{N^2}{c^2(N)}\dfrac{1}{(t_f - t_0)^2}$, $c(N) > 0$.

Indeed, our choice of such expression for $\alpha_u$, transform (5.51), taking rid of the dependence to the time-step $(t - t_0)$ analogously to a re-normalization of the input, lead to the following *time-step independent* mean and variance:

$$\begin{aligned}\mathbb{E}[\phi(dt;\alpha)] &= c^2(N)\Big(1 - \exp\big(-\frac{1}{c^2(N)}\big)\Big),\\ \sigma^2[\phi(dt;\alpha)] &= c^2(N)\frac{1 - \exp\big(-\frac{2}{c^2(N)}\big)}{2} - \mathbb{E}[\phi(dt;\alpha)]^2.\end{aligned} \tag{5.52}$$

This leaves us with only one parameter $c = c(N)$ to be determined for the "optimal" estimation of the upper bound of $\mathcal{U}$. Here, the value of $c(N)$ is found based on a reference solution, say $\boldsymbol{u}_{ref}$ resulting from the integration of a stiff problem, whose solution profiles contain also sharp gradients.

Thus, in order to calibrate the hyperparameters of the scheme, *once and for all*, we have chosen as a reference solution the one resulting from the van der Pol (vdP) ODEs given by:

$$\frac{du_1}{dt} = u_2, \quad \frac{du_2}{dt} = \mu(1 - {u_1}^2)u_2 - u_1, \tag{5.53}$$

for $\mu = 100$ and $u_1(0) = 2$, $u_2(0) = 0$ as initial conditions; the time interval was set to $[0 \quad 3\mu]$, i.e., approximately three times the period of the relaxation oscillations, which for $\mu \gg 1$, is $T \approx \mu(3 - 2\ln 2)$. The particular choice of $\mu = 100$ results to a stiff problem, containing also very sharp gradients resembling approximately a discontinuity in the solution profile within the integration interval. The reference solution was obtained using the `ode15s` with absolute and relative error tolerances set to 1E−14. In order to estimate the optimal values $(c, N)$ (while we fixed $n = 20$), we computed the bias (B)-variance(V) trade-off loss $\mathcal{L}_{BV}$ function for $u_2$ (whose amplitude for the particular setting is about 75 times bigger than the amplitude of $u_1$), using $600,000$ equidistant points $t_k$ in $[0 \quad 3\mu]$ and running each PIRPNN configuration 100 times. Thus, the B-V trade-off loss is given by:

$$\begin{aligned}\mathcal{L}_{BV} &= (B(\Psi_{N2}))^2 + V(\Psi_{N2}) = \mathbb{E}_\alpha((\Psi_{N2} - u_{ref,2})^2)\\ B(\Psi_{N2}) &= \mathbb{E}_\alpha\Big(\sum_{k=1}^{600,000}(\Psi_{N2}(t_k,\alpha,\boldsymbol{w}_2^o) - u_{ref,2}(t_k))\Big)\\ V(\Psi_{N2}) &= \mathbb{E}_\alpha\Big(\sum_{k=1}^{600,000}(\Psi_{N2}(t_k,\alpha,\boldsymbol{w}_2^o) - u_{ref,2}(t_k))^2\Big),\end{aligned} \tag{5.54}$$

where the expectation is estimated over the 100 runs. Based on the above, the parsimonious selection of the values of the variables $N$ and $c$ giving the best numerical accuracy and minimum computational cost are $N = 20$, $c = 12$ (see Figs. 5.1(a),(b). We note that the above parsimonious optimal values are fixed once and for all, for all benchmark problems considered here.

Finally, in order to demonstrate the efficiency of the proposed continuation approach, in Figure 5.1(c), we illustrate the $l^\infty$ numerical approximation accuracy (indicatively for $u_2$) w.r.t. the reference solution vs. the required execution times with and without (thus setting all weights to zero or randomly as initial guesses for the Newton iterations) the proposed continuation method. As shown, the implementation of the proposed continuation scheme results in significantly better performances.

## 5.5 Numerical Implementation and Results

We implemented the proposed numerical scheme in MATLAB 2020b. Numerical results were obtained running on an Intel Xeon Gold 6240R CPU @2.40GHz frequency and 35.75 MB of cache. Each execution is a single-thread, thus parallel encoding is not used. The `For-loop` for the formation of the Jacobian matrix $\nabla_{\boldsymbol{w}^o}\boldsymbol{F}$ was implemented via a MEX file calling a C function and the Moore-Penrose pseudo-inverse was computed with the MATLAB built-in function `pinv`, with the default tolerance. In all our computations, we have used a fixed number of collocation points $n = 20$ and number of basis functions $N = 20$, with $c = 12$ as discussed above. We note that a different choice of $n$ would result in different values of $c$, thus affecting the step-size adaptation.

For assessing the performance of the proposed scheme, here we considered six benchmark problems. In particular, we considered (a) two index-1 DAEs: the Robertson [239, 104] model of chemical kinetics and a non-autonomous power discharge control model [104]; (b) two stiff systems of ODEs: the Prothero-Robinson [240] and the van der Pol model [99]; and (c) the Allen-Chan metastable PDE phase-field model [58, 105] discretized in space with central FD. The performance of the proposed scheme was compared against two adaptive step-size solvers of the MATLAB ODE suite [103], namely `ode15s` and `ode23t`, appropriate for stiff ODEs and index-1 DAEs, thus using the analytical Jacobian. Moreover, we compared the performance of the scheme with a deep learning PINN as implemented in the DeepXDE library for scientific ML and physics-informed learning [93] for the solution of the Lotka-Volterra ODEs included in the demos of the library.

In order to estimate the numerical approximation error, we used as reference solution the one computed with `ode15s` setting the relative and absolute tolerances to 1E−14 and 1E−16, respectively. To this aim, we computed the $l^2$ and $l^\infty$ norms of the approximation errors, between the computed solutions using the various schemes, and the reference solutions, using grids of say, $M$ equidistant points in the time intervals of interest. In the following, we report the aforementioned error metrics, $||\boldsymbol{\epsilon}||_{l^2} = \sqrt{\sum_{j=1}^{M} {\epsilon_j}^2}$, $||\boldsymbol{\epsilon}||_{l^\infty} = \max_{j=1}^{M} |\epsilon_j|$, for the component of the solution for which the absolute error was maximum. Finally, we ran each solver for a wide range of relative tolerances $reltol$, thus setting the absolute tolerances $abstol = reltol \cdot 10^{-3}$. For each case, we ran the ODE solvers for 30 times and computed the median, maximum and minimum computational times. We note that a direct comparison of the `ode15s` and `ode23t` solvers and our scheme, based only on the relative and absolute tolerances (that is fixing them and check which one results in the best numerical approximation accuracy) cannot be done as these tolerances/convergence errors for `ode15s` and `ode23t` are *at the level of the solution*,

while for our PIRPNN scheme are *at the level of the differential operator/first derivative*.

Finally, we underline that the proposed PIRPNN scheme provides an approximate solution in the form of an analytical function that can be evaluated explicitly at any point in the interval, while with the `odesuite` solvers, for the evaluation of the solution at any point in the interval, one needs to resort to the computationally expensive interpolation (as implemented by the function `deval`[241] in particular). Thus, if one needs to evaluate the solution in a *dense grid* of equidistant points in order to perform tasks such as the analysis of chaotic and quasi-periodic dynamics using for example FFT, the computational cost can be considerably high. Therefore, the comparison between solvers was made both on the grid of points resulting from the corresponding adaptive step-size methods and on dense grids of equidistant points.

### 5.5.1 Case Study 1: Prothero-Robinson problem

Our first problem is the Prothero-Robinson stiff ODE benchmark problem [240, 101] given by

$$\frac{du}{dt} = \lambda(u - \phi(t)) + \phi'(t), \quad \lambda < 0. \tag{5.55}$$

Its analytical solution is $u(t) = \phi(t)$. The problem becomes stiff for $\lambda \ll -1$. For our numerical simulations, we chose $\phi(t) = sin(t)$, $u(0) = \phi(0) = sin(0) = 0$, and $[0, 2\pi]$ as the time interval where the solution is sought, while the parameter $\lambda$ controlling the stiffness is set equal to $-1$E+05.

Please note that, although the analytical solution is simple and smooth, i.e., does not exhibit any steep gradient, the problem is very difficult to solve with a non-stiff solver (for example using ode45 Matlab solver employing the adaptive Dormand-Prince scheme).

Figures 5.2(a)-(f) depict the $l^2$, $l^\infty$ numerical approximation accuracy w.r.t. the analytical solution vs. the required computational times. Figures 5.2(a)-(b) depict the computational times of the corresponding adaptive step-size solution procedure. Figure 5.2(c)-(d) depict the computational times of the various solvers when the solution is sought in a grid of 10,000 equidistant points in $[0 \quad 2\pi]$. Finally, Figures 5.3(a),(b) depict the $l^2$ numerical approximation accuracy (indicatively for $u_2$) w.r.t. the reference solution vs. the number of adaptive steps (Figure 5.3(a)), the number of function evaluations (Figure 5.3(b)).

As it is shown, the PIRPNN outperforms `ode23t` in all metrics, while for all practical purposes its performance is equivalent to the `ode15s` w.r.t. the computational times resulting from the corresponding adaptive step-size solution procedure. Besides, when the solution is sought in the dense grid of equidistant points, the computational times resulting from the implementation of the `odesuite` solvers are much larger than the ones resulting from the implementation of the proposed PIRPNN scheme. As it is also shown in Figure 5.3, our scheme, compared to both `ode15s` and `ode23t`, is more efficient in terms of the number of adaptive steps needed to compute the solution, while it needs more function evaluations than `ode15s` and less than `ode23t`.

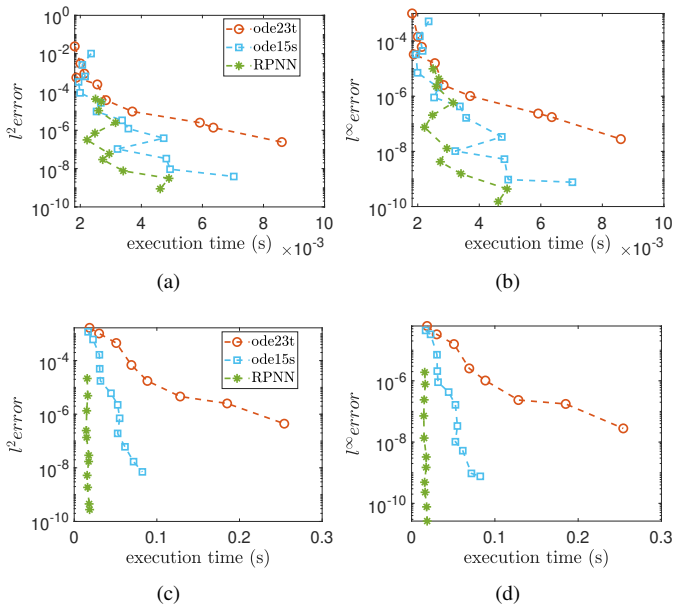



Figure 5.2: The Prothero-Robinson[240] benchmark stiff ODE problem with $\lambda = -1\text{E}{+}05$, see Eq. (5.55). (a)-(b) $l^2$, $l^\infty$ numerical approximation errors w.r.t. the analytical solution $u(t) = sin(t)$ vs. execution times (s) of the various schemes, using the adaptation in time step. (c)-(d) $l^2$, $l^\infty$ numerical approximation errors w.r.t. the analytical solution vs. execution times (s) when the solution is sought *in a grid of 10,000 equidistant points* in $[0 \quad 2\pi]$ times (s).

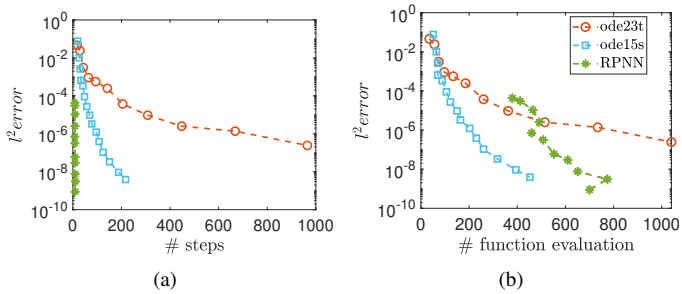



Figure 5.3: The Prothero-Robinson[240] stiff ODE with $\lambda = -1\text{E}{+}05$, see Eq. (5.55). $l^2$ numerical approximation error vs. (a) the number of adaptive steps, (b) the number of function evaluations.

### 5.5.2 Case Study 2: The van der Pol model

Our second benchmark problem is the stiff van der Pol system of ODEs (5.53) introduced in section 5.4. Figures 5.4(a),(b) depict the reference solution profiles for $u_1$, $u_2$, with

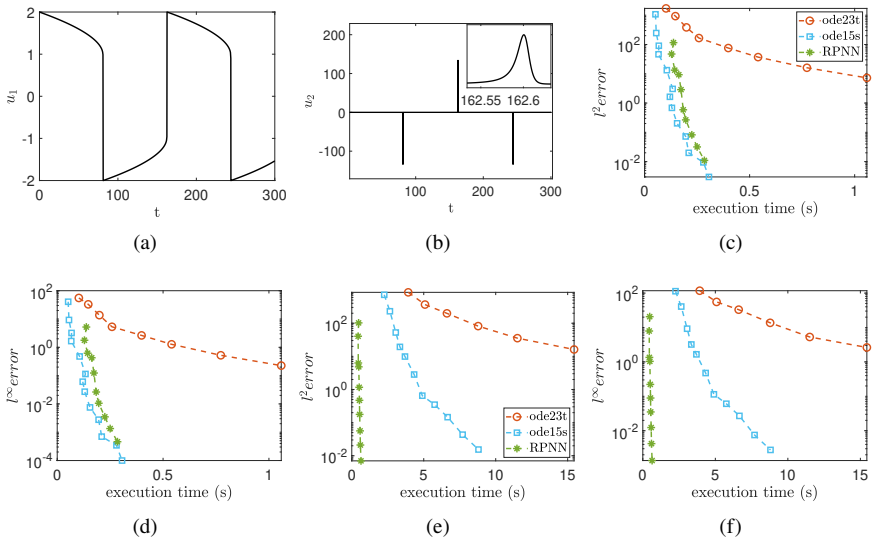



Figure 5.4: The vdP[99] ODEs with $\mu = 100$, see Eq. (5.53). The numerical reference solution is obtained in the time interval $[0 \quad 3\mu]$ with `ode15s` with relative and absolute tolerances set to 1E−14 and 1E−16, respectively. (a) Reference solution for $u_1$, (b) reference solution for $u_2$ with a zoom close to a steep gradient. (c)-(d) $l^2$, $l^\infty$ numerical approximation errors (indicatively for $u_2$) w.r.t. the reference solution vs. execution times (s) of the corresponding *adaptive step-size solution procedure*. (e)-(f) $l^2$, $l^\infty$ numerical approximation errors (indicatively for $u_2$) w.r.t. the reference solution vs. execution times (s) when the solution is sought *in a grid of 600'000 equidistant points* in $[0 \quad 3\mu]$ times (s).

$\mu = 100$ in the time interval $[0 \quad 3\mu]$ as obtained with `ode15s` with the relative and absolute tolerances set to 1E−14 and 1E−16, respectively. The relaxation oscillations of the vdP model exhibit both very sharp gradients resembling discontinuities, and stiffness [99]. Figures 5.4(c)-(f) depict the $l^2$, $l^\infty$ numerical approximation accuracy (indicatively for $u_2$) w.r.t. the reference solution vs. the required computational times. Figures 5.4(c)-(d) depict the computational times of the corresponding adaptive step-size solution procedure; upon convergence, the approximation errors are computed based on a uniform grid of 600'000 points in $[0 \quad 3\mu]$. Figure 5.4(e)-(f) depict the computational times of the various solvers when the solution is sought in a grid of 600,000 equidistant points (such a number of points is required in order to appropriately trace uniformly the solution in the interval of interest due to the presence of very steep gradients). Finally, Figures 5.5(a),(b) depict the $l^2$ numerical approximation accuracy (indicatively for $u_2$)

w.r.t. the reference solution vs. the number of adaptive steps (Figure 5.5(a)), the number of function evaluations (Figure 5.5(b)).

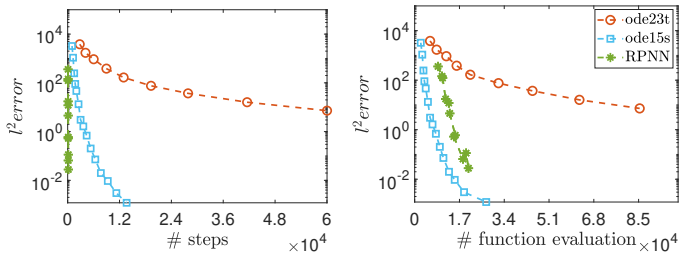



Figure 5.5: The vdP [99] ODEs with $\mu = 100$, see Eq. (5.53). $l^2$ numerical approximation error (indicatively for $u_2$) vs. (a) the number of adaptive steps, (b) the number of function evaluations.

As it is shown, the PIRPNN outperforms `ode23t` in all metrics, while for all practical purposes its performance is equivalent to the `ode15s` w.r.t. the computational times resulting from the corresponding adaptive step-size solution procedure. Besides, when the solution is sought in the dense grid of equidistant points, the computational times resulting from the implementation of the `odesuite` solvers are much larger than the ones resulting from the implementation of the proposed PIRPNN scheme.

As it is shown, our scheme, compared to both `ode15s` and `ode23t`, is more efficient in terms of number of adaptive steps needed to compute the solution, while it needs a comparable number of function evaluations with `ode15s` and significantly less than `ode23t`.

### 5.5.3 Case Study 3: The Robertson index-1 DAEs

The Robertson model describes the kinetics of an autocatalytic reaction [239]. This system of three DAEs is part of the benchmark problems considered in [104]. The set of the reactions reads:

$$A \xrightarrow{k_1} B, \quad B + C \xrightarrow{k_2} A + C, \quad 2B \xrightarrow{k_3} B + C, \tag{5.56}$$

where $A$, $B$, $C$ are chemical species and $k_1 = 0.04$, $k_2 = 10^4$ and $k_3 = 3 \times 10^7$ are reaction rate constants. Assuming that the total mass of the system is conserved, we have the following system of index-1 DAEs:

$$\begin{aligned} &\frac{dA}{dt} = -k_1 A + k_2 BC, \quad \frac{dB}{dt} = +k_1 A - k_2 BC - k_3 B^2, \\ &A + B + C = 1, \end{aligned} \tag{5.57}$$

where $A$, $B$ and $C$ denote the concentrations of $[A]$, $[B]$ and $[C]$, respectively. In our simulations, we set $A(0) = 1$, $B(0) = 0$ as initial conditions of the concentrations, and we solve in the time interval $[0 \quad 4 \cdot 10^6]$ as in [242].

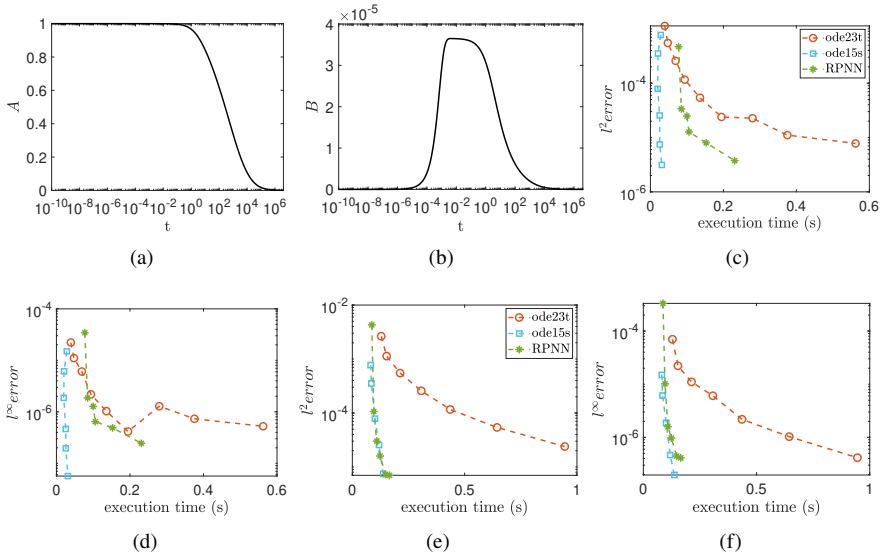



Figure 5.6: The Robertson[104] index-1 DAEs, see Eq. (5.57). The reference solution is obtained in the time interval $[0 \quad 4 \cdot 10^6]$ with `ode15s` with relative and absolute tolerances set to 1E−14 and 1E−16, respectively. (a)-(b) Reference solution profiles (indicatively for $A$, $B$). (c)-(d) $l^2$, $l^\infty$ numerical approximation errors (indicatively for $A$) w.r.t. the reference solution vs. execution times (s) of the corresponding *adaptive step-size solution procedure*. (e)-(f) $l^2$, $l^\infty$ numerical approximation errors (indicatively for $A$) w.r.t. the reference solution vs. execution times (s) when the solution is sought *in a grid of 40,000 logarithmically-equidistant points* in $[0 \quad 4 \cdot 10^6]$.

Figures 5.6(a)-(b) show the solution profiles of $A$, $B$ and $C$ as obtained with `ode15s` with relative and absolute tolerances set to 1E−14 and 1E−16, respectively. Figures 5.6(c)-(f) depict the $l^2$, $l^\infty$ numerical approximation errors (indicatively for $A$) upon convergence of the corresponding adaptive step-size procedure, w.r.t. the reference solution using $40,000$ logarithmically-equidistant points in the interval $[0 \quad 4 \cdot 10^6]$. Figures 5.6(c)-(d) depict the computational times of the corresponding adaptive step-size solution procedure, while Figures 5.6(e)-(f) depict the computational times required when the solution is sought in a grid of $40,000$ logarithmically-equidistant points in the interval $[0 \quad 4 \cdot 10^6]$.

As it is shown, the proposed PIRPNN scheme outperforms `ode23t` in all metrics, but compared with the `ode15s`, the computational times resulting from the adaptive step-size solution procedure are larger. However, when the solution is sought in the denser grid of points, the performance of the PIRPNN scheme is equivalent to the one of the `ode15s` solver. As it is shown, our scheme is comparable to `ode15s` and significantly more efficient than `ode23t` in terms of number of adaptive steps needed to compute the solution, while it needs a bigger number of function evaluations than `ode15s` and

significantly less than `ode23t`.

### 5.5.4 Case Study 4: Power discharge control index-1 DAEs problem

This is a non-autonomous model of six index-1 DAEs, and it is part of the benchmark problems considered in [104]. The governing equations are:

$$\begin{aligned}
\frac{du_1}{dt} &= \frac{(u_2-u_1)}{20}, \quad \frac{du_2}{dt} = -\frac{(u_4-99.1)}{75}, \\
\frac{du_3}{dt} &= \mu - u_6, \quad 0 = 20u_5 - u_3 \\
0 &= (3.35 - 0.075u_6 + 0.001u_6^2) - \frac{u_4}{u_5}, \\
0 &= \frac{u_3}{400}\frac{du_3}{dt} + \frac{\mu\mu_p}{(1.2u_1)^2} - \frac{du_1}{dt}\frac{\mu^2}{(1.44u_1)^3}, \\
\mu &= 15 + 5tanh(t-10), \quad \mu_p = \frac{5}{cosh^2(t-10)}.
\end{aligned} \tag{5.58}$$

The initial conditions are $u_1(0) = u_2(0) = 0.25$, $u_3(0) = 734$, and consistent initial conditions ($u_4(0) = 99.08999492002$, $u_5(0) = 36.7$ and $u_6(0) = 10.00000251671$) were found with Newton-Raphson with a tolerance of 1E$-$16.

Figures 5.7(a)-(b) depict the reference solution profiles for, indicatively, $u_2$, $u_3$ as obtained with `ode15s` with both relative and absolute tolerances set to 1E$-$14 and 1E$-$16, respectively. Similarly to the Robertson model, the solution profiles do not exhibit very steep gradients. Figures 5.7(c)-(f) depict the $l^2$, $l^\infty$ approximation errors (indicatively for $u_3$), upon convergence of the corresponding adaptive step-size procedure, w.r.t. the reference solution on the basis of $40,000$ equidistant points in the interval $[0 \quad 40]$. Figures 5.7(c)-(d) depict the computational times of the corresponding adaptive step-size procedure, while Figures 5.7(e)-(f) depict the required computational times when the solution is sought in a grid of 40,000 equidistant points.

As it is shown, the proposed PIRPNN scheme outperforms the `ode23t` solver in all metrics for higher numerical approximation accuracy, while the best performance w.r.t. the corresponding adaptive step-size procedure is the one obtained with the `ode15s` solver. However, when the solution is sought in the grid of 40,000 equidistant points, the PIRPNN outperforms `ode15s`. Finally, our scheme is comparable with `ode15s` and significantly more efficient than `ode23t` in terms of number of adaptive steps needed to compute the solution, and is comparable with `ode15s` and significantly more efficient than `ode23t` regarding the number of function evaluations.

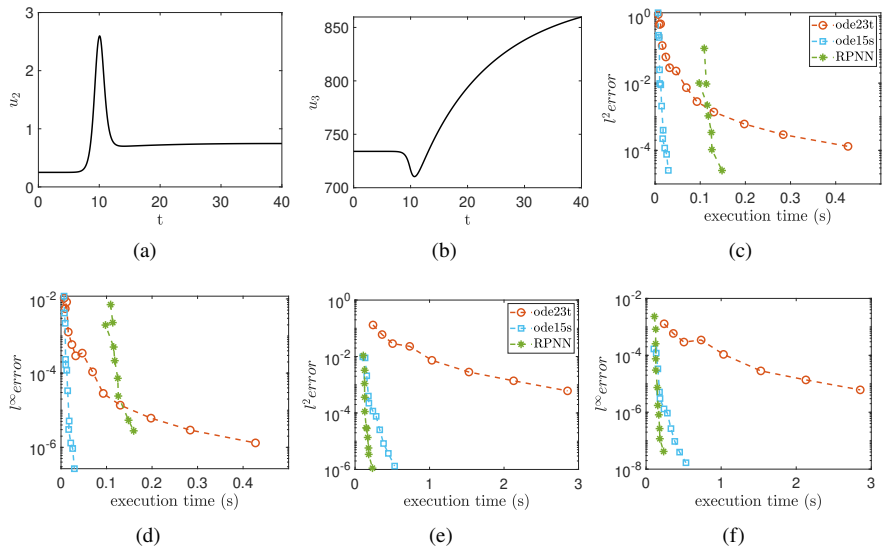



Figure 5.7: Power discharge control non-autonomous index-1 DAEs problem [104], see Eq. (5.58). The reference solution is obtained in the interval $[0 \quad 40]$ with `ode15s` with relative and absolute tolerances set to 1E−14 and 1E−16, respectively. (a)-(b) Reference profiles for, indicatively, $u_2, u_3$. (c)-(d) $l^2$, $l^\infty$ numerical approximation errors (indicatively for $u_3$) w.r.t. the reference solution vs. execution times (s) of the corresponding *adaptive step-size solution procedure*. (e)-(f) $l^2$, $l^\infty$ numerical approximation errors (indicatively for $u_3$) w.r.t. the reference solution vs. execution times (s) when the solution is sought *in a grid of 40,000 equidistant points*.

### 5.5.5 Case Study 5: The Allen-Cahn PDE phase-field model

The Allen-Cahn equation is a famous reaction-diffusion PDE that was proposed in [58] as a phase-field model for describing the dynamics of the mean curvature flow. Here, for our illustrations, we considered a one-dimensional formulation given by [105]:

$$\begin{aligned} &\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2} + u - u^3, \qquad x \in [-1 \quad 1], \\ &u(-1,t) = -1, \quad u(1,t) = 1, \end{aligned} \tag{5.59}$$

with initial condition $u(x,0) = 0.53\,x + 0.47\,\sin(-1.5\,\pi\,x)$. Here, we integrate until $t = 70$. For $\nu = 0.01$, the solution is stiff [105], thus exhibiting a metastable behavior with an initial two-hill configuration that disappears close to $t = 40$ with a fast transition to a one-hill stable solution, as depicted in Figure 5.8(a). For our illustrations, we used an equally spaced grid of 102 points in space and second order centered FD. Hence, (5.59)

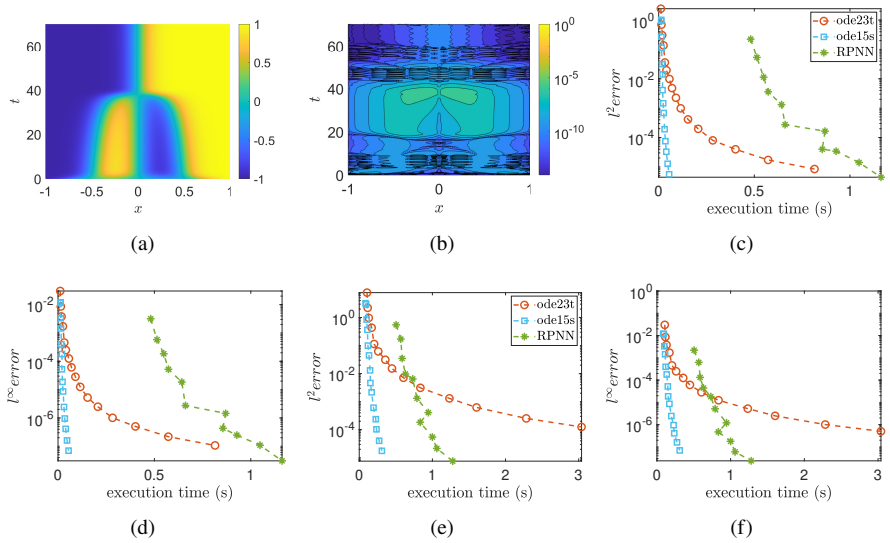



Figure 5.8: Allen-Cahn PDE [105] discretized in space with central FD (see Eq. (5.60)). The reference solution is obtained for $\nu = 0.01$ in the time interval $[0 \quad 70]$ with the`ode15s` solver, with relative and absolute tolerances set to 1E−14 and 1E−16, respectively. (a) Contour plot of the reference solution. (b) Contour plot of absolute approximation error computed with the PIRPNN using 1e−03 for the relative tolerance and 1e−06 for the absolute tolerance. (c)-(d) $l^2$, $l^\infty$ approximation errors w.r.t. the reference solution vs. execution times (s) of the corresponding *adaptive step-size procedure*. (e)-(f) $l^2$, $l^\infty$

becomes a system of 100 ODEs:

$$\begin{aligned} &\frac{\partial u_i}{\partial t} = \nu \frac{(u_{i+1} - 2u_i + u_{i-1})}{dx^2} + u_i - u_i^3, \\ &u_0 = -1, \quad u_{101} = 1. \end{aligned} \tag{5.60}$$

Here, for the implementation of the PIRPNN, we have used a sparse QR decomposition as implemented in the `SuiteSparseQR` [236]. Figures 5.8(b) depict the absolute numerical approximation error when using the PIRPNN for relative and absolute tolerances set to 1e−03 and 1e−06, respectively. Figures 5.8(c)-(f) depict the $l^2$, $l^\infty$ numerical approximation errors, upon convergence of the corresponding adaptive step-size procedure, w.r.t. the reference solution in $70\,000 \times 102$ equidistant points in the time $[0 \quad 70]$ and in the space interval $[-1 \quad 1]$, respectively. Figures 5.8(c)-(d) depict the computational times of the corresponding adaptive step-size procedure, while Figures 5.8(e)-(f) depict the computational times required when the solution is sought in the uniform spatio-temporal grid of $102 \times 70,000$ points.

As shown, the proposed PIRPNN scheme is less efficient than both `ode23t` and `ode15s` when considering the computational times resulting from the corresponding

adaptive step-size procedure. However, when considering the computational times required when the solution is sought in the mesh of 70,000 equidistant points in time, the PIRPNN outperforms `ode23t` but still is less efficient than `ode15s`. However, our proposed scheme, compared to both `ode15s` and `ode23t` is more efficient in terms of number of adaptive steps needed to compute the solution, while it needs more number of function evaluations with `ode15s` and significantly less than `ode23t`. The relatively higher computational cost is due to the considerable bigger size of the Jacobian required by the proposed scheme at each Newton iteration (here of size $2000 \times 2000$) compared to the Jacobian processed by `ode15s/ode23t` (here of size $100 \times 100$). Thus, to speed up the computations in a subsequent work, we aim in a future work, at exploiting the arsenal of matrix-free methods in the Krylov subspace [224] such as Newton-GMRES for the solution of such large-scale problems.

### 5.5.6 Comparison with the DeepXDE library: Lotka-Volterra ODEs

In this section, we compare the performance of the proposed scheme with a deep learning PINN, as implemented in the DeepXDE library [93]. In particular, we consider a demo of the DeepXDE library for the solution of the Lotka-Volterra ODEs reading:

$$\begin{aligned} \frac{dr}{dt} &= \frac{R}{U}(2Ur - 0.04U^2rp) \\ \frac{dp}{dt} &= \frac{R}{U}(0.02U^2rp - 1.06Up) \\ r(0) &= \frac{100}{U}; \qquad p(0) = \frac{15}{U}, \end{aligned} \tag{5.61}$$

where the parameters are set $U = 200, R = 20$ and the solution is sought in the time interval [0, 1].

The reference solution were obtained with the `ode15s` solver with relative and absolute tolerances set to 1E−14 and 1E−16, respectively.

For the considered demos, DeepXDE uses 3000 training residual points inside the domain and 3000 points for testing the ODE residual, given by:

$$\begin{aligned} \frac{dr}{dt} - \frac{R}{U}(2Ur - 0.04U^2rp) &= 0 \\ \frac{dp}{dt} - \frac{R}{U}(0.02U^2rp - 1.06Up) &= 0. \end{aligned} \tag{5.62}$$

The neural network architecture employed in the DeepXDE demo is a deep feed-forward one with 6 hidden layers with 64 neurons each, and hyperbolic tangent as activation function. Furthermore, in order to enforce the prediction to be periodic and thus more accurate, in the demo, the time input is first projected in a 7-dimensional feature layer, given by:

$$t \to \begin{bmatrix} t & sin(t) & sin(2t) & sin(3t) & sin(4t) & sin(5t) & sin(6t) \end{bmatrix} \tag{5.63}$$

Finally, to hard constrain the DeepXDE to satisfy the initial conditions, the 2-dim output $y = (y_1, y_2)$ of the neural network is transformed as:

$$\hat{r} = \frac{100}{U} + y_1 tanh(t), \quad \hat{p} = \frac{15}{U} + y_2 tanh(t). \tag{5.64}$$

The default optimization procedure implements the Adam algorithm with a learning rate 0.001 and $50,000$ iterations and then the optimization continues with the L-BFGS algorithm in order to achieve a higher accuracy. Please note that the DeepXDE approach is not adaptive and the solution is sought directly in the entire interval (i.e., without any step-size adaptation).

Given the above, from a computational point of view, it is clear that it is much more efficient to proceed in an adaptive-step employing a single-hidden layer RPNN with only 20 neurons (i.e., the only unknown weights are the ones that connect the hidden layer to the output) that can be computed using a newton-scheme with pseudo-inverse of the Jacobian vs. a DeepXDE neural network with $(7 \times 64 + 64^2 \times 5 + 64 \times 2 + 64 \times 6 + 2) = 21,442$ unknowns (i.e., all the weights and biases need to be learned) that is trained by many iterations of the Adam+L-BFGS algorithms.

Table 5.1: Lotka-Volterra[93] ODEs in the interval $[0, 1]$, see Eq. (5.61). Mean computational time in seconds (s) and approximation errors ($l^2$-norm, $l^\infty$-norm and MAE) for (indicatively) the $r$ component w.r.t. the reference solution computed with `ode15s` with relative and absolute tolerances set to 1E$-$14 and 1E$-$16, respectively. The PIRPNN solutions are computed with relative tolerances ranging from 1e$-$03 to 1e$-$06, and the DeepXDE PINN solutions with $3, 4, 5, 6$ hidden layers with $8, 16, 32, 64$ neurons, respectively.

| | | TIME (s) | $l^2$-error | $l^\infty$-error | MAE |
|---|---|---|---|---|---|
| RPNN | tol=1E$-$03 | 6.75E$-$02 | 2.11E$-$02 | 8.72E$-$04 | 1.22E$-$04 |
| | tol=1E$-$04 | 7.93E$-$02 | 2.33E$-$03 | 9.75E$-$05 | 1.38E$-$05 |
| | tol=1E$-$05 | 1.19E$-$01 | 1.50E$-$04 | 6.27E$-$06 | 8.92E$-$07 |
| | tol=1E$-$06 | 1.24E$-$01 | 1.00E$-$05 | 4.14E$-$07 | 6.29E$-$08 |
| DeepXDE | $3 \times 8$ | 6.39E+02 | 2.31E+01 | 6.50E$-$01 | 1.71E$-$01 |
| | $4 \times 16$ | 4.04E+02 | 2.97E+00 | 1.24E$-$01 | 1.70E$-$02 |
| | $5 \times 32$ | 1.20E+03 | 3.95E$-$01 | 1.63E$-$02 | 2.32E$-$03 |
| | $6 \times 64$ | 1.73E+03 | 8.03E$-$03 | 2.99E$-$04 | 5.04E$-$05 |

In Table 5.1 we compare the performance of the two PIML schemes in terms of mean computational time in seconds and $l^2$, $l^\infty$ and MAE for (indicatively) for the $r$ component w.r.t. the reference solution computed with `ode15s` setting relative and absolute tolerances to 1E$-$14 and 1E$-$16, respectively. In particular, for the proposed scheme, we have selected a range of 4 relative tolerances, ranging from 1E$-$03 to 1E$-$06 for the PIRPNN, and we have varied the number of hidden layer and neurons of the DeepXDE, using (a) $3 \times (8)$; (b) $4 \times 16$; (c) $5 \times 32$ and (d) $6 \times 64$ as Deep network architectures. The best result, in terms of numerical approximation accuracy, among the different DeepXDE architectures is taken with the $6 \times 64$ structure. This corresponds

to a comparable accuracy resulting from the proposed PIRPNN when relatively large tolerances (reltol=1E−-03 or 1E−04). But in terms of computational times, the proposed PIRPNN can obtain this approximation in just 7.93E−02 (s) versus the 1.73E+03 times needed by the DeepXDE.

## 5.6 Discussion

We presented a PIRPNN to solve the forward problem of ODEs and index-1 DAEs. The proposed scheme is a “numerical analysis-assisted" one, in the sense that the approach is not only “Physics-informed", but the PIRPNN is sought to approximate the Picard-Lindelöf integral, thus it is also an integration scheme with similar inspiration/sharing similar concepts to RK methods. Furthermore, we have borrowed important specialized techniques of numerical analysis, incorporating an adaptive step-size as in the traditional stiff solvers, and a continuation method (a concept borrowed from the numerical bifurcation analysis theory) for providing good initial guesses to facilitate the convergence of Newton iterations. Furthermore, in the case of sparse systems we exploit, state-of-the-art numerical analysis methods such as the sparse QR decomposition. The numerical results on six benchmark problems show that the scheme arises as a promising alternative to well established ODE solvers for stiff problems, and appears to be much more efficient in terms of the computational cost than deep-learning ML schemes for the solution of ODEs.

Future work will be focused on the further development and application of the scheme for solving large-scale systems of stiff ODEs, DAEs as well as PDEs. For this task, we aim at integrating ideas from other numerical methods such as multistep methods, aided by slow inertial manifolds such as CSP [125] and matrix-free methods in the Krylov-subspace [224] in order to speed up computations.

# 6 Solution of the Inverse Problem for Complex Systems I: The ML Framework

The solution of inverse problems is critical in complex systems as they involve deducing underlying rules or models from observed data. Unlike forward problems, where the system dynamics are known and predictions are made, inverse problems aim to recover the governing equations, parameters, or structures of the system from observed outputs. These challenges are compounded in high-dimensional and noisy environments, necessitating robust methodologies to ensure accurate model identification and prediction.

In this chapter, we introduce the ML framework for the reconstruction of underlying macroscopic laws, from microscopic high-fidelity data, and the analysis of tipping points and rare events.

## 6.1 ML framework for the inverse problem

Given high dimensional spatio-temporal trajectory data acquired through microscopic simulator, such as ABM simulations, the main steps of the framework are summarized as follows (see also Figure 6.1, for a schematic):

a. Discover low-dimensional latent spaces, on which the emergent dynamics can be described at the mesoscopic or the macroscopic scale.

b. Identify, via ML, black-box mesoscopic IPDEs, ODEs, or (after further dimensional reduction), macroscopic mean-field SDEs.

c. Locate tipping points by exploiting numerical bifurcation analysis of the different surrogate models.

d. Use the identified (NN-based) surrogate mean-field SDEs to perform rare-event analysis (uncertainty quantification) for the catastrophic transitions. This is done here in two ways: (i) performing repeated brute-force simulations around the tipping points, (ii) for this effectively 1D problem, using explicit statistical mechanical (Feynman-Kac) formulas for escape time distributions.

In what follows, we present the elements of the methodology.

### 6.1.1 Challenges in solving the inverse problem

Addressing inverse problems in complex systems requires overcoming several challenges, particularly in identifying appropriate coarse-scale variables, ensuring model robustness and generalization, and quantifying uncertainty in predictions. Extracting meaningful macroscopic variables from high-dimensional microscopic data is a key difficulty. ML surrogates, like RPNNs, are instrumental in this process, as they effectively reduce dimensionality and highlight the system's essential features.

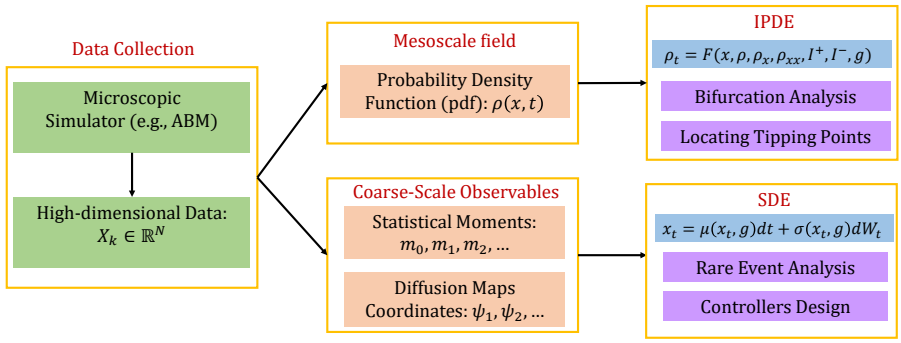



Figure 6.1: Schematic of the ML-based approach for the multiscale modelling and analysis of tipping points. At the first step, and depending on the scale of interest, we discover via Diffusion Maps latent spaces using, mesoscopic fields (probability density functions (pdf) and corresponding spatial derivatives) with the aid of Automatic Relevance Determination (ARD); or macroscopic mean-field quantities, such as statistical moments of the probability density function. At the second step, on the constructed latent spaces, we solve the inverse problem of identifying the evolutionary laws, as IPDEs for the mesosopic field scale, or mean-field SDEs for the macroscopic scale. Finally, at the third step, based on the constructed surrogate models, we perform system level analysis, such as numerical integration at a lower computational cost, numerical bifurcation analysis for the detection and characterization of tipping points, and rare event analysis (uncertainty quantification) for the catastrophic transitions occurring in the neighborhood of the tipping points.

Once the relevant variables are identified, accurately recovering the underlying dynamics and ensuring that the learned model generalizes beyond the training data become critical tasks. This is particularly important for predicting rare events or behaviors near tipping points, where small changes in parameters can lead to significant shifts in system dynamics.

Training models on high-dimensional and often sparse data can lead to overfitting, where the model captures noise rather than the true signal. To mitigate this, regularization techniques such as dropout in neural networks or ARD in Gaussian processes

are employed. These methods help stabilize learning by preventing the model from becoming overly complex and ensuring it remains robust when exposed to new data.

Moreover, inverse problems are frequently ill-posed, meaning that different models may explain the same data with similar accuracy. Quantifying the uncertainty in model predictions is therefore essential for reliable interpretation and decision-making. Techniques such as Bayesian inference and bootstrapping are used to assess the confidence in the reconstructed models, providing a measure of the range and reliability of the predictions made by the ML framework. This comprehensive approach ensures that the solutions are not only accurate but also trustworthy and interpretable.

## 6.2 Discovering low-dimensional latent spaces.

The computational modeling of complex systems featuring a multitude of interacting agents poses a significant challenge due to the enormous number of potential states that such systems can have. Thus, a fundamental step, for the development of ROMs that are capable of effectively capturing the collective behavior of ensembles of agents is the discovery of an embedded, in the high-dimensional space, low-dimensional manifold and an appropriate set of variables that can usefully parametrize it.

Let's denote, by $\mathbf{X}_k \in \mathbb{R}^D$, $k = 1, 2, \ldots$ the high-dimensional state of the ABM at time $t$. The goal is then to project/map the high-dimensional data onto lower-dimensional latent manifolds $\mathcal{M} \subset \mathbb{R}^D$, that can be defined by a set of coarse-scale variables. The hypothesis of the existence of this manifold is related to the existence of useful ROMs and vice versa.

Here, to discover such a set of coarse-grained coordinates for the latent space, we used DMaps [129, 130] (see Section 6.4.2 for a brief description of the DMaps algorithm).

For both ABMs, we have some *a priori* physical insight for the mesoscopic description. For the financial ABM, one can for example use the pdf $\rho(X)dx = \mathbb{P}(X(t) \in [x, x + dx])$ across the possible states $X_k$ in space. Thus, the continuum pdf constitutes a spatially dependent *mesoscopic* field that can be modelled by an FP IPDE as explained above. For the epidemic ABM, there is a physical insight on the macroscopic mean-field description, which is the well-known mean-field SIRS model. Multiscale macroscopic descriptions can also be constructed, including higher-order closures [12]. Alternatively, one can also collect "enough" statistical moments of the underlying distribution such as the expected value, variance, skewness, kurtosis, etc. Nevertheless, the collected statistics may not automatically provide insight into their relevance in the effective dynamics and a further feature selection/sensitivity analysis may be needed.

Focusing on a reduced set of coarse-scale variable is particularly relevant when there exists a significant separation of time scales in the system's dynamics. By selecting only a few dominant statistics, one can effectively summarize the behavior of the system at a coarser level.

The choice of the scale and details of coarse-grained description, leads to different modelling approaches. For example, focusing at the *mesoscale* for the population density dynamics, we aim at constructing a FP-level IPDE for the financial ABM, and a mean-field SIR surrogate for the epidemic ABM. At an even coarser scale, e.g., for the first moment

of the distribution, and taking into account the underlying stochasticity, a natural first choice is the construction of a mean-field macroscopic SDE. Here, we construct surrogate models via ML at both these distinct coarse-grained scales.

### 6.2.1 Diffusion Maps: a Dimension Reduction Approach

Diffusion Maps (DMaps), introduced by Coifman and Lafon in 2006 [129, 130], is a manifold learning technique designed to reveal both linear and non-linear structures in high-dimensional data. It provides an intrinsic embedding of the low-dimensional manifold on which the data are assumed to lie.

DMaps employs a random walk over the data points, each considered as a node of a graph, to uncover the geometric structure of the data by approximating the Laplace-Beltrami operator on the manifold. In this graph, the edges between nodes represent the transition probabilities from one data point to another. Given a data matrix $X \in \mathbb{R}^{m\times d}$, where $m$ is the number of data points and $d$ is the dimension of each point $x_i$, the DMaps algorithm constructs an affinity (kernel) matrix $W$, where each entry $w_{ij}$ is computed as:

$$w_{ij} = \exp\left(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\epsilon}\right), \tag{6.1}$$

with $\epsilon$ being a scale parameter. Here, $||\cdot||$ represents the $l_2$ norm.

To make the kernel matrix invariant to the sampling density and to approximate the Laplace-Beltrami operator numerically, the following normalization is applied:

$$\tilde{W} = P^{-1}WP^{-1}, \quad P_{ii} = \sum_{j=1}^{m} W_{ij}. \tag{6.2}$$

Next, the matrix $D \in \mathbb{R}^{m\times m}$ is computed, with $D_{ii} = \sum_{j=1}^{m} \tilde{W}_{ij}$, and a second normalization is applied to obtain the row-stochastic matrix $A$:

$$A = D^{-1}\tilde{W}. \tag{6.3}$$

We then perform the eigen-decomposition of $A$:

$$A\phi_i = \lambda_i\phi_i, \tag{6.4}$$

where the eigenvectors $\phi_i$ are ordered according to their corresponding eigenvalues $\lambda_i$.

It is important to select eigenvectors that span linearly independent directions, often referred to as non-harmonic modes. While spectral gaps typically indicate the data's multiscale structure, identifying the optimal "minimal" set of eigenfunctions is not always straightforward. The non-harmonic eigenvectors can be selected using the local linear regression algorithm proposed by Dsilva et al. [146]. This algorithm identifies the non-harmonic eigenvectors by fitting eigenvectors $\psi_i$ (for $i > 1$) as local linear functions of preceding eigenvectors.

The details of this parsimonious algorithm are explained in the following section.

## 6.3 Learning mesoscopic IPDEs via neural networks.

The identification of evolution operators of spatio-temporal dynamics using ML tools, including deep learning and Gaussian processes, represents a well-established field of research. The main assumption here is that the emergent dynamics of the complex system under study on a domain $\Omega \times [t_0, t_{end}] \subseteq \mathbb{R}^d \times \mathbb{R}$ can be modelled by a system of say $m$ IPDEs in the form of:

$$\begin{aligned} \frac{\partial u^{(i)}(\mathbf{x},t)}{\partial t} \equiv u_t^{(i)} = F^{(i)}(\mathbf{x}, \mathbf{u}, \mathcal{D}\mathbf{u}, \mathcal{D}^{\mathbf{2}}\mathbf{u}, \dots, \\ \dots, \mathcal{D}^{\boldsymbol{\nu}}\mathbf{u}, I_1^{(i)}(\mathbf{u}), I_2^{(i)}(\mathbf{u}), \dots, \boldsymbol{\varepsilon}), \\ (\mathbf{x},t) \in \Omega \times [t_0, t_{end}], \qquad i = 1, 2, \dots, m, \end{aligned} \tag{6.5}$$

where $F^{(i)}, i = 1, 2, \dots m$ are $m$ non-linear IPDE operators; $\mathbf{u}(\mathbf{x},t) = [u^{(1)}(\mathbf{x},t), \dots$ $\dots, u^{(m)}(\mathbf{x},t)]$ is the vector containing the spatio-temporal fields, $\mathcal{D}^{\boldsymbol{\nu}}\mathbf{u}(\mathbf{x},t)$ is the generic multi-index $\boldsymbol{\nu}$-th order spatial derivative at time $t$:

$$\begin{aligned} \mathcal{D}^{\boldsymbol{\nu}}\mathbf{u}(\mathbf{x},t) := \left\{ \frac{\partial^{|\boldsymbol{\nu}|}\mathbf{u}(\mathbf{x},t)}{\partial x_1^{\nu_1} \cdots \partial x_d^{\nu_d}}, \nu_1, \dots, \nu_d \geq 0 \right\}, \\ \text{where } |\boldsymbol{\nu}| = \nu_1 + \nu_2 + \cdots + \nu_d, \end{aligned} \tag{6.6}$$

$I_1^{(i)}, I_2^{(i)}, \dots$ are a collection of integral features on subdomains $\Omega_1^{(i)}, \Omega_2^{(i)}, \dots \subseteq \Omega$:

$$I_j^{(i)}(\mathbf{u}) = \int_{\Omega_j^{(i)}} K_j^{(i)}(\mathbf{x}, \mathbf{u}(\mathbf{x},t)) d\Omega, \qquad j = 1, 2, \dots; \tag{6.7}$$

$K_j^{(i)} : \mathbb{R}^d \times \mathbb{R}^m \mapsto \mathbb{R}^d$ are nonlinear maps and $\boldsymbol{\varepsilon} \in \mathbb{R}^p$ denotes the (bifurcation) parameters of the system. The right-hand-side of the $i$-th IPDE depends on say, a number of $\gamma^{(i)}$ variables and on bifurcation parameters from the set of features:

$$\mathcal{S}^{(i)} = \{\mathbf{x}, \mathbf{u}(\mathbf{x},t), \mathcal{D}\mathbf{u}(\mathbf{x},t), \mathcal{D}^{\mathbf{2}}\mathbf{u}(\mathbf{x},t), \dots, \mathcal{D}^{\boldsymbol{\nu}}\mathbf{u}(\mathbf{x},t), I_1^{(i)}, I_2^{(i)}, \dots, \boldsymbol{\varepsilon}\}. \tag{6.8}$$

At each spatial point $\mathbf{x}_q, q = 1, 2, \dots, M$ and time instant $t_s, s = 1, 2, \dots, N$, a single sample point (an observation) in the set $\mathcal{S}^{(i)}$ for the $i$-th IPDE can be described by a vector $Z_j^{(i)} \equiv Z_{(q,s)}^{(i)} \in \mathbb{R}^{\gamma^{(i)}}$, with $j = q + (s-1)M$. Here, we assume that such mesoscopic IPDEs in principle exist, but they are not available in closed-form. Henceforth, we aim to learn the macroscopic laws by employing an FNN, in which the effective input layer is constructed by a finite stencil (sliding over the computational domain), mimicking convolutional operations where the applied "filter" involves values of our field variable(s) $u^{(i)}$ on the stencil, and returns features $Z_j^{(i)} \in \mathcal{S}^{(i)}$ of these variables at the stencil center-point, i.e., spatial derivatives as well as (local or global) integrals (see Figure 6.2(a) for a schematic). The FNN is fed with the sample points $Z_j^{(i)} \in \mathbb{R}^{\gamma^{(i)}}$ and its output is an approximation of the time derivative $\mathbf{u}_t$.

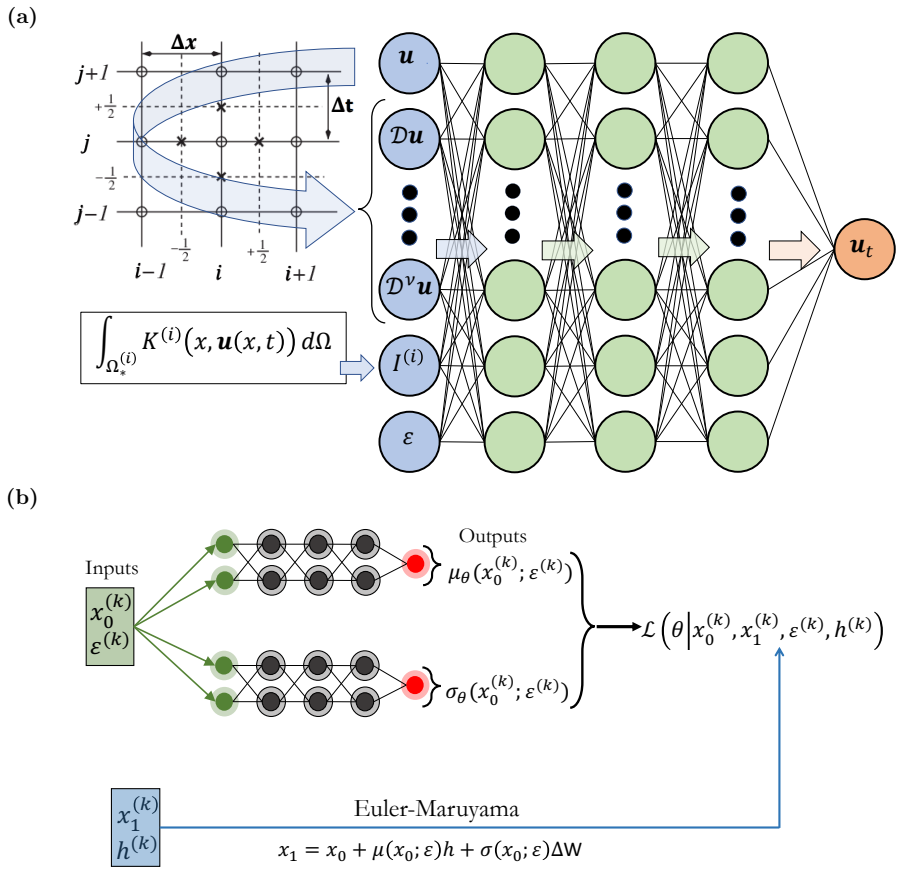



Figure 6.2: A Schematic representation of the neural networks used for constructive ML assisted surrogates. (a) Feedforward Neural Network (FNN). the input is constructed by convolution operations, i.e., a combination of sliding FD stencils, and, integral operators, for learning mesoscopic models in the form of IPDEs (Eq. (6.5)); the inputs to the RHS of the IPDE are the features in Eq. (6.8). (b) A schematic representation of the neural network architecture, inspired by numerical stochastic integrators, used to construct macroscopic models in the form of mean-field SDEs.

**Remark on learning mean-field ODEs.** The proposed framework can be also applied for the simpler task of learning a system of $m$ ODEs in terms of the $m$ variables $\mathbf{u} = (u^{(1)}, u^{(2)}, \ldots, u^{(m)})$, thus learning $i = 1, \ldots, m$ functions:

$$\frac{du^{(i)}(t)}{dt} = F^{(i)}(\mathbf{u}, \mathbf{p}),\ t \in [t_0, t_{end}], \tag{6.9}$$

where $\mathbf{p} \in \mathbb{R}^k$ denotes the parameter vector.

## 6.4 Feature selection

### 6.4.1 Feature selection with Automatic Relevance Determination.

Training the FNN with the "full" set of inputs $\mathcal{S}^{(i)} \subseteq \mathbb{R}^{\gamma^{(i)}}$, described in Eq. (6.8), consisting of all local mean field values as well as all their coarse-scale spatial derivatives (up to some order $\nu$) is prohibitive due to the "curse of dimensionality". Therefore, one important task for the training of the FNN is to extract a few "relevant"/dominant variable combinations. Towards this aim, we used ARD in the framework of GPR [40].The approach assumes that the collection of all observations $\mathbf{Z}^{(i)} = (Z_1^{(i)}, Z_2^{(i)}, \ldots, Z_{MN}^{(i)})$, of the features $z_l \in \mathcal{S}^{(i)}$, are a set of random variables whose finite collections have a multivariate Gaussian distribution with an unknown mean (usually set to zero) and an unknown covariance matrix $K$. This covariance matrix is commonly formulated by a Euclidean distance-based kernel function $k$ in the input space, whose hyperparameters are optimized based on the training data. Here, we employ a RBF, which is the default kernel function in Gaussian process regression, with ARD:

$$K_{jh}^{(i)} = k(Z_j^{(i)}, Z_h^{(i)}, \boldsymbol{\theta}^{(i)}) = \theta_0^{(i)} \exp\left(-\frac{1}{2}\sum_{l=1}^{\gamma^{(i)}} \frac{z_{l,j} - z_{l,h}}{\theta_l^{(i)}}\right); \tag{6.10}$$

$\boldsymbol{\theta}^{(i)} = [\theta_0^{(i)}, \theta_1^{(i)}, \ldots, \theta_{\gamma^{(i)}}^{(i)}]$ are a $(\gamma^{(i)} + 1)$–dimensional vector of hyperparameters. Specifically, the optimal hyperparameter set $\tilde{\boldsymbol{\theta}}^{(i)}$ can be obtained by minimizing a negative log marginal likelihood over the training data set $(\mathbf{Z}^{(i)}, \mathbf{Y}^{(i)})$, with inputs the observation $\mathbf{Z}^{(i)}$ of the set $\mathcal{S}^{(i)}$ and corresponding desired output given by the observation $\mathbf{Y}^{(i)}$ of the time derivative $u_t^{(i)}$.

$$\tilde{\boldsymbol{\theta}}^{(i)} = \arg\min_{\boldsymbol{\theta}^{(i)}} -\log p(\mathbf{Y}^{(i)}|\mathbf{Z}^{(i)}, \boldsymbol{\theta}^{(i)}). \tag{6.11}$$

As can be seen in equation (6.10), a large value of $\theta_l$ nullifies the difference between target function values along the $l$-th dimension, allowing us to designate the corresponding $z_l$ feature as "insignificant". Practically, in order to build a reduced input data domain, we define the normalized effective *relevance weights* $W_r^{(i)}(\cdot)$ of each feature input $z_l \in \mathcal{S}^{(i)}$, by taking:

$$\bar{W}_r^{(i)}(z_l) = exp(-\tilde{\theta}_l^{(i)}), \qquad W_r^{(i)}(z_l) = \frac{\bar{W}_r^{(i)}(z_l)}{\sum_l \bar{W}_r^{(i)}(z_l)}. \tag{6.12}$$

Thus, we define a small tolerance $tol$ in order to disregard the components such that $W_r^{(i)}(z_l) < tol$. The remaining selected features ($W_r^{(i)}(z_l) \geq tol$) can still successfully (for all practical purposes) parametrize the approximation of the right-hand-side of the underlying IPDE.

### 6.4.2 Feature selection using Diffusion Maps with leave-one-out cross-validation

For the selection of the lowest-dimensional embedding set of eigenfunction, a normalized leave-one-out error, denoted as $r_k$, is used for this selection and quantifies gradually which eigenvectors are independent (non-harmonic) and which are not (harmonic). If the number of non-harmonic eigenvectors is smaller than $d$ the dimensionality reduction is achieved. Furthermore, this algorithm can be applied for feature selection purpose.

For this purpose, given a set of $\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \ldots, \boldsymbol{\phi}_{k-1} \in \mathbb{R}^N$ Diffusion Maps eigenvectors, for each element $i = 1, 2 \ldots, N$ of $\boldsymbol{\phi}_k$, we use a local linear regression model:

$$\phi_{k,i} \approx \alpha_{k,i} + \beta_{k,i}^T \Phi_{k-1,i}, \quad i = 1, 2, \ldots, N \tag{6.13}$$

to investigate if $\phi_k$ is a dependent eigen-direction; $\Phi_{k-1,i} = [\phi_{1,i}, \phi_{2,i}, \ldots, \phi_{k-1,i}^T]$, $\alpha_{k,i} \in \mathbb{R}$ and $\beta_{k,i} \in \mathbb{R}^{k-1}$. The values of parameters $\alpha_{k,i}$ and $\beta_{k,i}$ are found solving an optimization problem of the form:

$$\hat{\alpha}_{k,i}, \hat{\beta}_{k,i} = \underset{\alpha,\beta}{\operatorname{argmin}} \sum_{j \neq i} K(\Phi_{k-1,i}, \Phi_{k-1,j})(\phi_{k,j} - (\alpha + \beta^T \Phi_{k-1,j}))^2, \tag{6.14}$$

where $K$ is a kernel weighted function, usually the Gaussian kernel:

$$K(\Phi_{k-1,i}, \Phi_{k-1,j}) = \exp\left(-\frac{||\Phi_{k-1,i} - \Phi_{k-1,j}||}{\sigma^2}\right), \tag{6.15}$$

where $\sigma$ is the shape parameter. The final normalized leave-one-out cross-validation error for this local linear fit is defined as:

$$r_k = \sqrt{\frac{\sum_{i=1}^N (\phi_{k,i} - (\hat{\alpha}_{k,i} + \hat{\beta}_{k,i}^T \Phi_{k-1,i})^2}{\sum_{i=1}^{\mu} (\phi_{k,i})^2}}. \tag{6.16}$$

For small values of $r_k$, $\boldsymbol{\phi}_k$ is considered to be dependent of the other eigenvectors and hence as a harmonic or repeated eigen-direction, while large values of $r_k$, suggest that $\boldsymbol{\phi}_k$ can serve as a new independent eigen-direction.

In practice, for feature selection, one can provide as inputs to the DMaps algorithm the combined input-output domain. With this, we can seek the subsets of variables of the input space that minimally parametrize the intrinsic embedding by quantifying it with a total regression loss $L_T$ based on an MSE:

$$L_T = (\sum_{k=1}^{\mu} L_{\boldsymbol{\phi}_k}^2)^{\frac{1}{2}}. \tag{6.17}$$

Here, as $L_{\phi_j}$, we define the regression loss for representing the intrinsic coordinate $\phi_j$ when using $s$ out of $n$ selected input features:

$$L_{\boldsymbol{\phi}_k} = \frac{1}{N} \sum_{i=1}^{N} (\phi_{k,i} - g(\cdot))^2, \tag{6.18}$$

where $g(\cdot)$ is the output of the regressors with inputs the values of the features in the ambient space and target values the eigenvectors $\phi_k$.

## 6.5 Macroscopic mean-field SDEs via neural networks.

Here, we present our approach for the construction of embedded surrogate models in the form of mean-field SDEs. Under the assumption that we are close (in phase- and parameter space) to a previously located tipping point, we can reasonably assume that the effective dimensionality of the dynamics can be reduced to the corresponding normal form. We already have some qualitative insight on the type of the tipping point, based for example on the numerical bifurcation calculations that located it (e.g., for the financial ABM from the analytical FP IPDE, from our surrogate IPDE, or from the EF analysis [28, 131]), while for the epidemic ABM from the EF analysis in [12]. For both these two particular problem, we have found that the tipping point corresponds to a saddle-node bifurcation.

Given the nature of the bifurcation (and the single variable corresponding normal form) we identify a one-dimensional SDE, driven by a Wiener process, from data. We note that learning higher-order such SDEs, or SDEs based on the more general Lévy process and the Ornstein–Uhlenbeck process [243], is straightforward.

For a diffusion process with drift, say $X_t = \{x_t, t > 0\}$, the drift, $\mu(x_t)$ and diffusivity $\sigma^2(x_t)$ coefficients over an infinitesimally small-time interval $dt$, are given by:

$$\begin{aligned} \mu(x_t) &= \lim_{\delta t \to 0} \frac{1}{\delta t} \mathbb{E}(\delta x_t | X_t = x_t), \\ \sigma^2(x_t) &= \lim_{\delta t \to 0} \frac{1}{\delta t} \mathbb{E}(\delta x_t^2 | X_t = x_t), \end{aligned} \tag{6.19}$$

where, $\delta x_t = x_{t+\delta t} - x_t$. The 1D SDE driven by a Wiener process $W_t$ reads:

$$dx_t = \mu(x_t; \varepsilon)dt + \sigma(x_t; \varepsilon)dW_t. \tag{6.20}$$

Here, for simplicity, we assume that the one-dimensional parameter $\varepsilon$, enters into the dynamics, via the drift and diffusivity coefficients. Our goal is to identify the functional form of the drift $\mu(x, \varepsilon)$ and the diffusivity $\sigma(x, \varepsilon)$ given noisy data close to the tipping point via ML. For the training, the data might be collected from either long-time trajectories or short bursts initialized at scattered snapshots, as in the EF framework. These trajectories form our data set of input-output pairs of discrete-time maps. A data point in the collected data set can be written as $(x_0^{(k)}, h^{(k)}, x_1^{(k)} \varepsilon^{(k)})$, where $x_0^{(k)}$ and $x_1^{(k)}$ measures two consecutive states at $t_0^{(k)}$ and $t_1^{(k)}$ with (small enough) time step $h^{(k)} = t_1^{(k)} - t_0^{(k)}$ and $\varepsilon^{(k)}$ is the parameter value for this pair. Based on the above formulation, going to $x_1^{(k)}$ from $x_0^{(k)}$ by:

$$x_1^{(k)} = x_0^{(k)} + \int_{t_0^{(k)}}^{t_1^{(k)}} \mu(x_t; \varepsilon^{(k)})dt + \int_{t_0^{(k)}}^{t_1^{(k)}} \sigma(x_t; \varepsilon^{(k)})dW_t. \tag{6.21}$$

Here, for the numerical integration of the above equation to get stochastic realizations of $x_1^{(k)}$, we assume that the Euler-Maruyama numerical scheme can be used, reading:

$$x_1^{(k)} \approx x_0^{(k)} + h^{(k)}\mu(x_0^{(k)};\varepsilon^{(k)}) + \sigma(x_0^{(k)};\varepsilon^{(k)})\Delta W^{(k)}, \tag{6.22}$$

where $\Delta W^{(k)} = W_{t_1^{(k)}} - W_{t_0^{(k)}} \in \mathbb{R}$ is a one-dimensional random variable, normally distributed with expected value zero and variance $h^{(k)}$.

Considering the point $x_1^{(k)}$ as a realization of a random variable $X_1$, conditioned on $x_0^{(k)}$ and $h^{(k)}$, drawn by a Gaussian distribution of the form:

$$\begin{aligned} p_{X_1}(x_1^{(k)}) &= \mathbb{P}\left(X_1 = x_1^{(k)} \mid X_0 = x_0^{(k)}, h^{(k)}\right) \sim \\ &\sim \mathcal{N}\left(x_0^{(k)} + h^{(k)}\mu(x_0^{(k)};\varepsilon^{(k)}), h^{(k)}\sigma(x_0^{(k)};\varepsilon^{(k)})^2\right), \end{aligned} \tag{6.23}$$

we approximate the drift $\mu(x_0^{(k)};\varepsilon^{(k)})$ and diffusivity $\sigma(x_0^{(k)};\varepsilon^{(k)})$ functions by simultaneously training two neural networks, denoted as $\mu_\theta$ and $\sigma_\theta$, respectively. This training process involves minimizing the loss function:

$$\begin{aligned} \mathcal{L}(\theta|x_0^{(k)}, x_1^{(k)}, h^{(k)}) := \sum_k \frac{(x_1^{(k)} - x_0^{(k)} - h^{(k)}\mu_\theta(x_0^{(k)};\varepsilon^{(k)}))^2}{h^{(k)}\sigma_\theta(x_0^{(k)};\varepsilon^{(k)})^2} + \\ + \log|h^{(k)}\sigma_\theta(x_0^{(k)};\varepsilon^{(k)})^2|. \end{aligned} \tag{6.24}$$

which is derived in order to maximize the log-likelihood of the data and where $\theta$ denotes the trainable parameters (e.g., weights and biases of the neural networks $\mu_\theta$ and $\sigma_\theta$). A schematic representation of the Neural Network, based on Euler-Maruyama, is shown in Figure 6.2(b).

## 6.6 Rare event analysis and Tipping points

### 6.6.1 Locating tipping points via our surrogate models.

In order to locate the tipping point, based on either the mesoscopic (Integro-) PDE or the embedded mean-field 1D SDE model, we construct the corresponding bifurcation diagram in its neighborhood, using pseudo-arc-length continuation as implemented in numerical bifurcation packages (see Appendix Section B.3). For the identified SDE, we used its deterministic part, i.e., the drift term, to perform continuation. The required Jacobian of the activation functions of the neural network is computed by symbolic differentiation. Note that, for the SDE, this is just a validation step (we already know the location and nature of the tipping point).

### 6.6.2 Rare-event analysis/UQ of catastrophic shifts.

Given a sample space $\Omega$, an index set of times $T = \{0, 1, 2, \dots\}$ and a state space $S$, the *first passage time*, also known as *mean exit time* or *mean escape time*, of a stochastic process $x_t : \Omega \times T \mapsto S$ *on a measurable subset* $A \subseteq S$ is a random variable which can be defined as

$$\tau(\omega) := \inf\{t \in T \mid x_t(\omega) \in S \setminus A\}, \tag{6.25}$$

where $\omega$ is a sample out of the space $\Omega$. One can define the mean escape time from $A$, which works as the expectation of $\tau(\omega)$:

$$\langle \tau \rangle := \mathbb{E}[\tau(\omega)]. \tag{6.26}$$

For a $n$-dimensional stochastic process, as it is the ABM under study, $S$ is typically set to be $\mathbb{R}^n$, and $A$ is usually a bounded subset of $\mathbb{R}^n$. In the case of our local 1D SDE model, the subset $A$ reduces to an open interval $(a, b)$, with the initial condition of this stochastic process $x_0$ also chosen in this interval.

We discuss two ways for quantifying the uncertainty of the occurrence of those rare-events. The first, presented in the main text, involves direct computational "cheap" temporal simulations of the 1D SDE, where one gets an empirical probability distribution; the second, is a closed-form expression, based on statistical mechanics, for the mean escape time (assuming an exponential distribution of escape times).

# 7 Solution of the Inverse Problem for Complex Systems II: Case studies

In this chapter, we present three complex systems where inverse problems are addressed using ML:

- FitzHugh-Nagumo PDE Model [35, 40]: This model is a simplification of the Hodgkin-Huxley model for neuronal activity and is typically used to describe excitable media. Here, this can describe the action potential propagation in unmyelinated neurons. In our work, we use the Lattice Boltzmann Method (LBM) at a mesoscale level to simulate the FitzHugh-Nagumo PDE. The goal is to recover the underlying PDE from data generated by the LBM simulation and to construct a coarse-scale bifurcation diagram.
- Event-Driven Financial Agent-Based Model [3, 28]: Here, we model a financial market where agents interact based on mimesis, imitating the behavior of other investors. This model captures the emergence of collective behavior, such as market bubbles and crashes. The inverse problem involves reconstructing the stochastic dynamics of the system and identifying coarse-grained variables that describe large-scale market behavior. Additionally, we aim to predict rare events, such as market crashes, by learning the probability of escaping from a stable state near tipping points.
- Epidemic Spreading over Erdos-Renyi social Network [11, 12]: This model addresses the spread of infectious diseases over a random network. The challenge is to predict the dynamics near critical transitions, such as the outbreak threshold. We also investigate the probability of rare events, such as explosive outbreaks, using a combination of ML-driven coarse-scale ODEs, correcting inaccurate mean-field SIR model and 1d ML-assisted surrogate in the form of an SDE.

## 7.1 Case study 1: A mesoscopic model of action potential in unmyelinated neurons

As first case study, used for assessing the performance of the proposed scheme, we selected the celebrated, well studied FitzHugh-Nagumo (FHN) model first introduced in [244] to simplify the Hodgkin-Huxley model into a two-dimensional system of ODEs to describe the dynamics of the voltage across a nerve cell. In particular, we consider the FHN equations, which add a spatial diffusion term to describe the propagation of an action potential as a traveling wave.

It is important to note that this is a toy model, as the PDE system is already well-established. However, we use it to test the framework's capability of reconstructing the model when a reference is available for comparison. The mesoscopic Lattice Boltzmann Method (LBM) serves as a mesoscopic simulator in this context.

The bifurcation diagram of the one-dimensional set of PDEs is known to have a turning point and two supercritical Andronov-Hopf bifurcation points. In what follows, we describe the model along with the initial and boundary conditions, and then we present the $D1Q3$ Lattice Boltzmann model.

### 7.1.1 The Macroscale model: the FitzHugh-Nagumo Partial Differential Equations

The evolution of activation $u : [x_0, x_{end}] \times [t_0, t_{end}] \to \mathbb{R}$ and inhibition $v : [x_0, x_{end}] \times [t_0, t_{end}] \to \mathbb{R}$ dynamics are described by the following two coupled nonlinear parabolic PDEs:

$$\begin{aligned} \frac{\partial u(x,t)}{\partial t} &= D^u \frac{\partial^2 u(x,t)}{\partial x^2} + u(x,t) - u(x,t)^3 - v(x,t), \\ \frac{\partial v(x,t)}{\partial t} &= D^v \frac{\partial^2 v(x,t)}{\partial x^2} + \varepsilon(u(x,t) - \alpha_1 v(x,t) - \alpha_0), \end{aligned} \tag{7.1}$$

with homogeneous von Neumann Boundary conditions:

$$\frac{du(x_{end},t)}{dx} = 0, \quad \frac{dv(x_0,t)}{dx} = 0. \qquad \frac{du(x_{end},t)}{dx} = 0, \quad \frac{dv(x_0,t)}{dx} = 0. \tag{7.2}$$

$\alpha_0$ and $\alpha_1$ are parameters, $\varepsilon$ is the kinetic bifurcation parameter.

For our simulations, we have set $x_0 = 0$, $x_{end} = 20$, $\alpha_1 = 2, \alpha_0 = -0.03, D^u = 1, D^v = 4$ and varied the bifurcation parameter $\varepsilon$ in the interval $[0.005, 0.955]$ [35]. For our simulations, in order to explore the dynamic behavior, we considered various initial conditions $u_0(x) = u(x, 0)$ and $v_0(x) = v(x, 0)$ selected randomly as follows:

$$\begin{aligned} u_0(x) &= w \tanh\big(\alpha(x-c)\big) + \beta & v_0(x) &= 0.12 \cdot u_0(x), \\ w &\sim \mathcal{U}(0.8, 1.2), \quad \alpha \sim \mathcal{U}(0.5, 1) & c &\sim \mathcal{U}(2, 18), \quad \beta \sim \mathcal{U}(-0.4, 0), \end{aligned} \tag{7.3}$$

where $\mathcal{U}(a, b)$ denotes the uniform distribution in the interval $[a, b]$.

### 7.1.2 The D1Q3 Lattice Boltzmann model

The Lattice Boltzmann model serves as our fine-scale simulator. The statistical description of the system at a mesoscopic level uses the concept of distribution function $f(\vec{r},\vec{c},t)$, i.e., $f(\vec{r},\vec{c},t)d\vec{r}d\vec{c}dt$ is the infinitesimal probability of having particles at location $\vec{r}$ with velocities $\vec{c}$ at a given time $t$, for reducing the high-number of equations and unknowns. Then, at this level, a system without an external force is governed by the Boltzmann Transport equation [245]:

$$\frac{\partial f}{\partial t} + \vec{c}\cdot\nabla f = \mathcal{R}(f), \tag{7.4}$$

where the term $\mathcal{R}(f)$ describes the rate of collisions between particles. In 1954, Bhatnagar, Gross and Krook (BGK) [245] introduced an approximation model for the collision operator:

$$\mathcal{R}(f) = \frac{1}{\tau}(f^{eq} - f), \tag{7.5}$$

where $\tau$ is the so-called relaxing time coefficient and $f^{eq}$ denote the local equilibrium distribution function.

In the LBM, Eq. (7.4)-(7.5) is collocated (assumed valid) along specific directions $\vec{c}_i$ on a lattice:

$$\frac{\partial f_i}{\partial t} + \vec{c}_i\cdot\nabla f_i = \frac{1}{\tau}(f_i^{eq} - f_i) \tag{7.6}$$

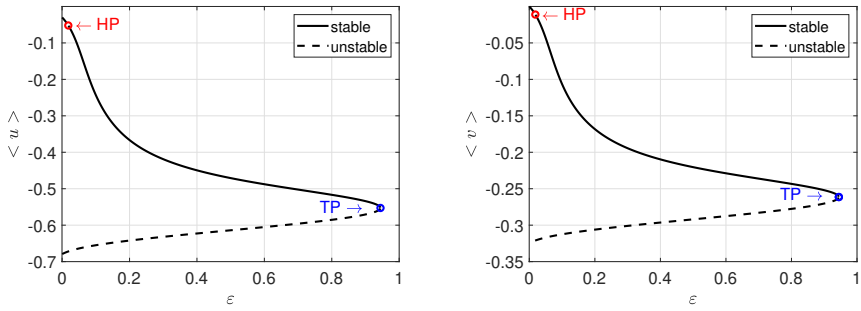
and then Eq. (7.6) is discretized with a time step $\Delta t$ as follows:

$$f_i(\vec{r}+\vec{c}_i\Delta t, t+\Delta t) = f_i(\vec{r},t) + \frac{\Delta t}{\tau}(f_i^{eq} - f_i). \tag{7.7}$$

One common interpretation of Eq. (7.7) is to think about the distribution functions as fictitious particles that stream and collide along specified linkages of the lattice. Lattices are usually denoted by the notation $DnQm$, where $n$ is the spatial dimension of the problem and $m$ refer to the number of connections of each node in the lattice. The node in the lattices coincide with the points of a spatial grid with a spatial step $\Delta x$.

Here, in order to estimate the coarse-scale observables $u$ and $v$ of the FHN dynamics, we considered the $D1Q3$ implementation, i.e., we used the one-dimensional lattice with three velocities $c_i$: particles can stream to the right ($c_1 = \frac{\Delta x}{\Delta t}$), to the left ($c_{-1} = -\frac{\Delta x}{\Delta t}$) or staying still on the node ($c_0 = 0$). Also, we assume the coexistence of two different distribution functions for describing the distribution of the activator particles $f_i^u$ and the distribution of the inhibitor particles $f_i^v$, where the subscript $i$ refer to the associated direction. Therefore, one can figure that at each instant there are six fictitious particles on each node of the lattice: two resting on the node (with distribution $f_0^u$ and $f_0^v$), two moving on the left (with distribution $f_{-1}^u$ and $f_{-1}^v$) and two moving on the right (with distribution $f_1^u$ and $f_1^v$). The relation between the above distributions and the coarse-scale density $u$ and $v$ is given by the zeroth moment (across the velocity directions) of the overall distribution function:

$$u(x_j,t_k) = \sum_{i=-1}^{1} f_i^u(x_j,t_k), \qquad v(x_j,t_k) = \sum_{i=-1}^{1} f_i^v(x_j,t_k). \tag{7.8}$$




(a) (b)

Figure 7.1: Reference bifurcation diagram of the FHN PDEs with respect to $\varepsilon$ as computed with FD and $N = 200$ points. (a) Mean values $< u >$ for stable and unstable branches, (b) Mean values $< v >$ for stable and unstable branches. Andronov-Hopf Point: $HP_\varepsilon$=0.01827931. Turning Point: $TP_\varepsilon$=0.94457768.

The coexistence of multiple distributions renders necessary to introduce weights $\omega_i$ for the connections in the lattice that should satisfy the following properties:

(a) Normalization $\omega_0 + \omega_1 + \omega_{-1} = 1$

(b) Symmetry $\omega_1 - \omega_{-1} = 0$

(c) Isotropy:

(c.1) $\omega_0 c_0^2 + \omega_1 c_1^2 + \omega_{-1} c_{-1}^2 = c_s^2$
(c.2) $\omega_0 c_0^3 + \omega_1 c_1^3 + \omega_{-1} c_{-1}^3 = 0$
(c.3) $\omega_0 c_0^4 + \omega_1 c_1^4 + \omega_{-1} c_{-1}^4 = 3c_s^4$,

where $c_s$ is the speed of sound in the lattice. Thus, the weights are equal to $\omega_{\pm 1} = 1/6$ for the moving particles and $\omega_0 = 4/6$ for the resting particle. The resulting speed of sound in the lattice is $c_s = \frac{\sqrt{3}\Delta x}{3\Delta t}$.

As the BGK operator (7.5) suggests, one key step in applying LBM for solving reaction-advection-diffusion PDEs is to determine the local equilibrium distribution function $f^{eq}$ associated to a given model. For particles with macroscopic density $\rho$ that move in a medium macroscopic velocity $\vec{u}_m$, the Maxwell distribution is:

$$\begin{aligned} f^{eq}(\vec{c}) &= \frac{\rho}{(2\pi RT)^{d/2}} \exp\left(-\frac{(\vec{c}-\vec{u}_m)^2}{2RT}\right) = \\ &= \frac{\rho}{(2\pi RT)^{d/2}} \exp\left(-\frac{\vec{c}\cdot\vec{c}}{2RT}\right) \exp\left(-\frac{-2\vec{c}\cdot\vec{u}_m + \vec{u}_m\cdot\vec{u}_m}{2RT}\right), \end{aligned} \tag{7.9}$$

where $d$ is the spatial dimension of the problem, $T$ is the temperature and $R$ is the universal gas constant. The exponential in Eq. (7.9) can be expanded using Taylor series,

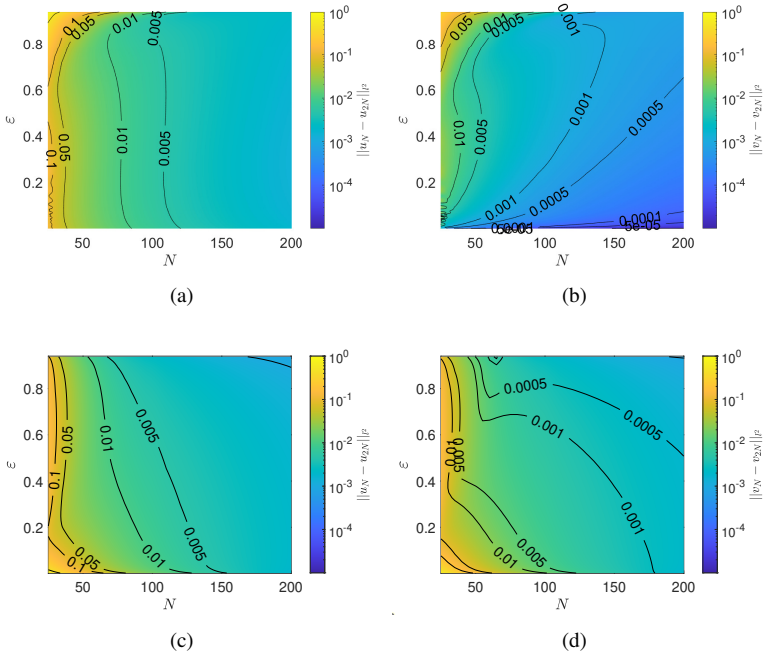



Figure 7.2: Contour plot of the $l^2$ norms of the convergence of the solutions as computed with FD with respect to the size of the grid $N$ computed as $||u_N - u_{2N}||_{l^2}$, $||v_N - v_{2N}||_{l^2}$. The convergence error was computed on 1001 grid points, using linear piecewise interpolation. (a) upper branch for $u$, (b) upper branch for $v$, (c) lower branch for $u$, (d) lower branch for $v$.

ignoring terms of order $O(u^3)$ and higher, thus obtaining:

$$f^{eq}(\vec{c}) = \rho\omega(\vec{c})\left[1 + \frac{2\vec{c}\cdot\vec{u}_m - \vec{u}_m\cdot\vec{u}_m}{2c_s^2} + \frac{(\vec{c}\cdot\vec{u}_m)^2}{2c_s^4}\right], \tag{7.10}$$

with $\omega(\vec{c}) = (2\pi RT)^{-d/2}\exp\left(-\frac{\vec{c}\cdot\vec{c}}{2RT}\right)$ and $RT = c_s^2$, with $c_s$ speed of the sound.

Now, since the FHN PDEs are only diffusive, i.e., there are no advection terms, the medium is stationary ($\vec{u}_m = 0$) and the equilibrium distribution function, discretized on the lattice direction $c_i$, is simplified in:

$$\begin{aligned} f_i^{u,eq}(x_j,t_k) &= \omega_i u(x_j,t_k), \quad i = -1,0,1 \\ f_i^{v,eq}(x_j,t_k) &= \omega_i v(x_j,t_k). \end{aligned} \tag{7.11}$$

Now, in the FHN model, we need to consider also reaction terms $R_i^l$ and so finally, the time evolution of the microscopic simulator associated to the FHN on a given $D1Q3$

lattice is:

$$f_i^l(x_{j+i}, t_{k+1}) = f_i^l(x_j, t_k) + \frac{\Delta t}{\tau^l}(f_i^{l,eq}(x_j, t_k) - f_i^l(x_j, t_k)) + \Delta t R_i^l(x_j, t_k), l \in \{u, v\} \tag{7.12}$$

where the superscript $l$ denotes the activator $u$ and the inhibitor $v$ and the reaction terms $R_i^l$ are directly derived by:

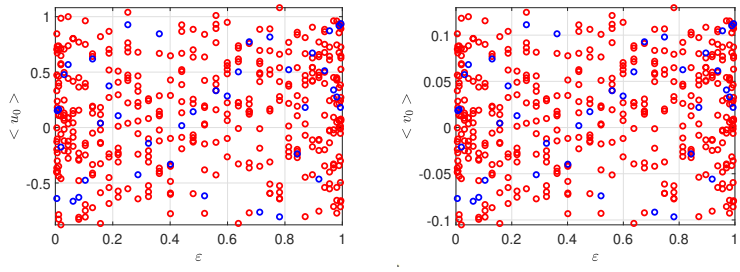
$$\begin{aligned} R_i^u(x_j, t_k) &= \omega_i(u(x_j, t_k) - u^3(x_j, t_k) - v(x_j, t_k)), \\ R_i^v(x_j, t_k) &= \omega_i\,\varepsilon(u(x_j, t_k) - \alpha_1 v(x_j, t_k) - \alpha_0). \end{aligned} \tag{7.13}$$

Finally, the relaxation coefficient $\dfrac{\Delta t}{\tau^l}$ is related to the macroscopic kinematic viscosity $D^l$ of the FHN model and in general depends on the speed of the sound $c_s$ associated to the lattice [246]:

$$\frac{\Delta t}{\tau^l} = \frac{2}{1 + \frac{2}{c_s^2 \Delta t} D^l} = \frac{2}{1 + 6 D^l \frac{\Delta t}{\Delta x^2}}. \tag{7.14}$$

### 7.1.3 Numerical bifurcation analysis of the FHN PDEs

For comparison purposes, we first constructed the bifurcation diagram of the FHN PDEs using central FD.




(a) (b)

Figure 7.3: Coarse initial conditions for (a) $u$ and (b) $v$ for the training. Every dot denotes a point whose $\varepsilon$ and mean $u$ (or $v$) were used for input data. Red dots are training points, blue points are test points. The grid is spanned with Chebychev-Gauss-Lobatto points for epsilons in the interval $[0.005, 0.995]$ and the initial condition are randomly selected as in Eq. (7.3)

The discretization of the one-dimensional PDEs in $M$ points with second-order central FD in the unit interval $0 \leq x \leq 20$ leads to the following system of $2(M-2)$

non-linear algebraic equations $\forall x_j = (j-1)h, j = 2, \dots M-1, h = \frac{1}{M-1}$:

$$F_j^u(u,v) = \frac{D^u}{h^2}(u_{j+1} - 2u_j + u_{j-1}) + u_j - u_j^3 - v_j = 0$$
$$F_j^v(u,v) = \frac{D^v}{h^2}(v_{j+1} - 2v_j + v_{j-1}) + \varepsilon(u_j - \alpha_1 v_j - \alpha_0) = 0.$$

At the boundaries, we imposed homogeneous von Neumann boundary conditions.

The above $2(M-2)$ set of non-linear algebraic equations is solved iteratively using Newton's method. The non-null elements of the Jacobian matrix are given by:

$$\frac{\partial F_j^u}{\partial u_{j-1}} = \frac{D^u}{h^2}; \frac{\partial F_j^u}{\partial u_j} = -D^u\frac{2}{h^2} - 3u_j^2; \frac{\partial F_j^u}{\partial u_{j+1}} = \frac{D^u}{h^2}; \frac{\partial F_j^u}{\partial v_j} = -1$$
$$\frac{\partial F_j^v}{\partial v_{j-1}} = \frac{D^v}{h^2}; \frac{\partial F_j^v}{\partial v_j} = -D^v\frac{2}{h^2} - \varepsilon\alpha_1 v_j; \frac{\partial F_j^v}{\partial v_{j+1}} = \frac{D^u}{h^2}; \frac{\partial F_j^v}{\partial u_j} = \varepsilon.$$

To trace the solution branch along the critical points, we used the pseudo arc-length-continuation method ([221, 222, 225]). This involves the parametrization of $u(x)$, $v(x)$ and $\varepsilon(x)$ by the arc-length $s$ on the solution branch. The solution is sought in terms of $\tilde{u}(x,s)$, $\tilde{v}(x,s)$ and $\tilde{\varepsilon}(s)$ in an iterative manner, by solving until convergence the following augmented system:

$$\begin{bmatrix} \nabla_u \boldsymbol{F^u} & \nabla_v \boldsymbol{F^u} & \nabla_\varepsilon \boldsymbol{F^u} \\ \nabla_u \boldsymbol{F^v} & \nabla_v \boldsymbol{F^v} & \nabla_\varepsilon \boldsymbol{F^v} \\ \nabla_u \boldsymbol{N} & \nabla_v \boldsymbol{N} & \nabla_\varepsilon \boldsymbol{N} \end{bmatrix} \begin{bmatrix} du^{(n)}(x,s) \\ dv^{(n)}(x,s) \\ d\varepsilon^{(n)}(s) \end{bmatrix} = - \begin{bmatrix} \boldsymbol{F^u}(u^{(n)}(x,s), v^{(n)}(x,s), \varepsilon^{(n)}(s)) \\ \boldsymbol{F^v}(u^{(n)}(x,s), v^{(n)}(x,s), \varepsilon^{(n)}(s)) \\ \boldsymbol{N}(u^{(n)}(x,s), v^{(n)}(x,s), \varepsilon^{(n)}(s)) \end{bmatrix}, \tag{7.15}$$

where

$$\nabla_\varepsilon \boldsymbol{F^u} = \begin{bmatrix} \frac{\partial F_1^u}{\partial \varepsilon} & \frac{\partial F_2^u}{\partial \varepsilon} & \dots & \frac{F_N^u}{\partial \varepsilon} \end{bmatrix}^T, \nabla_\varepsilon \boldsymbol{F^v} = \begin{bmatrix} \frac{\partial F_1^v}{\partial \varepsilon} & \frac{\partial F_2^v}{\partial \varepsilon} & \dots & \frac{F_M^v}{\partial \varepsilon} \end{bmatrix}^T,$$

and

$$\begin{aligned} \boldsymbol{N}(u^{(n)}(x,s), v^{(n)}(x,s), \varepsilon^{(n)}(s)) = \\ (u^{(n)}(x,s) - \tilde{u}(x,s)_{-2})^T \cdot \frac{(\tilde{u}(x)_{-2} - \tilde{u}(x)_{-1})}{ds} + \\ (v^{(n)}(x,s) - \tilde{v}(x,s)_{-2})^T \cdot \frac{(\tilde{v}(x)_{-2} - \tilde{v}(x)_{-1})}{ds} + \\ (\varepsilon^{(n)}(s) - \tilde{\varepsilon}_{-2}) \cdot \frac{(\tilde{\varepsilon}_{-2} - \tilde{\varepsilon}_{-1})}{ds} - ds, \end{aligned}$$

where $(\tilde{u}(x)_{-2}, \tilde{v}(x)_{-2})$ and $(\tilde{u}(x)_{-1}, \tilde{v}(x)_{-1})$ are two already found consequent solutions for $\tilde{\varepsilon}_{-2}$ and $\tilde{\varepsilon}_{-1}$, respectively and $ds$ is the arc-length step for which a new solution around the previous solution $(\tilde{u}(x)_{-2}, \tilde{v}(x)_{-2}, \tilde{\varepsilon}_{-2})$ along the arc-length of the solution branch is being sought. The corresponding reference bifurcation diagram is shown in Figure 7.1. In this range of values, there is an Andronov-Hopf bifurcation at $\varepsilon \approx 0.018497$ and a fold point at $\varepsilon \approx 0.95874$.

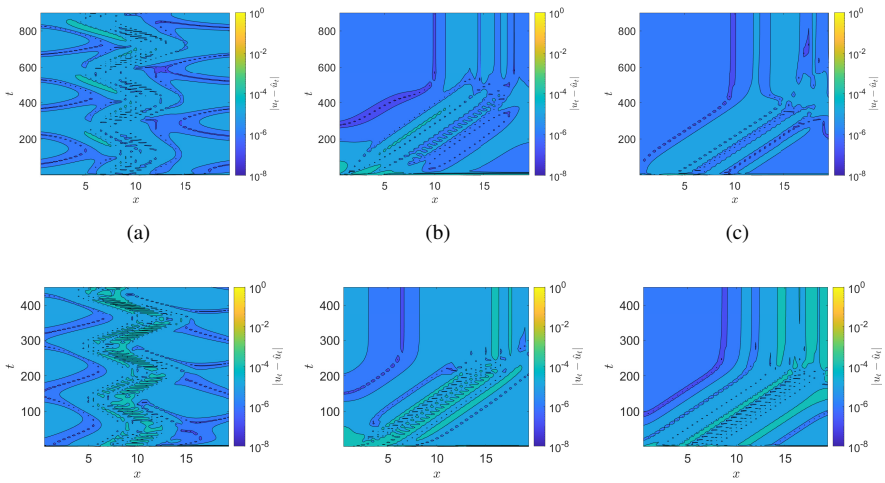



Figure 7.4: Approximation accuracy in the test data without feature selection, as obtained with FNNs (a)-(c) and RPNNs (d)-(f). Contour plot of the absolute values of differences in space and time, of, indicatively $|u_t(x,t) - \hat{u}_t(x,t)|$ for characteristic values of $\varepsilon$: (a) and (b) $\varepsilon = 0.0114$ near the Andronov-Hopf point, (c), (d) $\varepsilon = 0.4$, (e) and (f) $\varepsilon = 0.9383$ near the turning point.

### 7.1.4 Numerical bifurcation analysis from mesoscopic LBM simulations

We collected transients of $u(x,t)$ and $v(x,t)$ with a sampling rate of 1s, from 10 different random sampled initial conditions for 40 different values for the bifurcation parameter $\varepsilon$. In particular, we created a grid of 40 different $\varepsilon$ in [0005, 0.955] using Gauss-Chebychev-Lobatto points, while the 10 initial conditions are sampled according to Eq.(7.3).

Figure 7.3 depicts the total of 400 training initial conditions. Thus, we end up with a dataset consisting of 40 (values of $\varepsilon$) $\times$ 10 (initial conditions) $\times$ 448 (time points ignoring the first 2s of the transient) $\times$ 40 (space points) $\simeq$ 7.168.000 data points.
For learning the coarse-grained dynamics and construct the corresponding bifurcation diagram, we trained two FNNs and two single-layer RPNNs (one for each one of the variables $u$ and $v$). The FNNs were constructed using two hidden layers, with 12 units in each layer. Hidden units were employed with the hyperbolic tangent sigmoid activation function, while the regularization parameter was tuned and set $\lambda = 0.01$. For the training of the FNNs, we used the Deep Learning toolbox of MATLAB 2021a on an Intel Core i5-8265U with up to 3.9 GHz frequency with a memory of 8 GB.

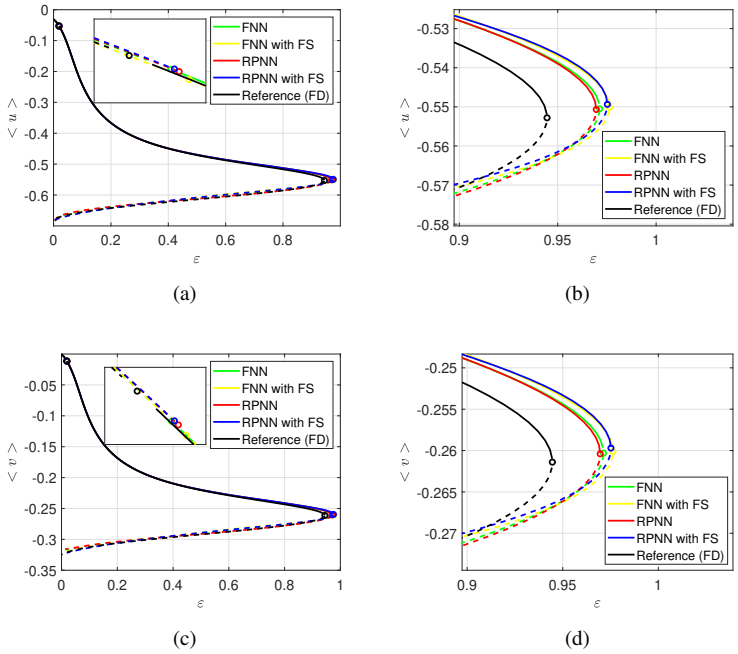



Figure 7.5: Reconstructed bifurcation diagrams from the Lattice Boltzmann simulations of the FHN dynamics with respect to $\varepsilon$ with FNNs and RPNNs with and without feature selection. (a) Mean values $< u >$ for stable and unstable branches; the inset zooms near the Andronov-Hopf bifurcation point (b) zoom near the turning Point for $< u >$, (c) Mean values $< v >$ for stable and unstable branches; the inset zooms near the Andronov-Hopf bifurcation point, (d) zoom near the turning Point for $< v >$.

### *Numerical bifurcation analysis without feature selection*

Table 7.1 summarizes the performance of the two schemes on the test data set. As

| | test set | | | |
|---|---|---|---|---|
| | MSE $(u)$ | $l^\infty$ $(u)$ | MSE $(v)$ | $l^\infty$ $(v)$ |
| FNN | 7.90E−09 | 2.26E−02 | 1.56E−09 | 6.63E−03 |
| FNN(FS) | 5.39E−08 | 2.93E−02 | 1.16E−08 | 7.65E−03 |
| RPNN | 2.91E−08 | 2.98E−02 | 4.50E−10 | 2.22E−03 |
| RPNN(FS) | 7.10E−08 | 3.07E−02 | 1.73E−08 | 1.60E−02 |

Table 7.1: Mean-square error (MSE) and $l^\infty$ errors between the predicted $\hat{u}_t$ and $\hat{v}_t$ from the FNNs and RPNNs and the actual time derivatives $u_t$ and $v_t$ without and with feature selection (FS).

it is shown, for any practical purposes, both schemes resulted to equivalent numerical

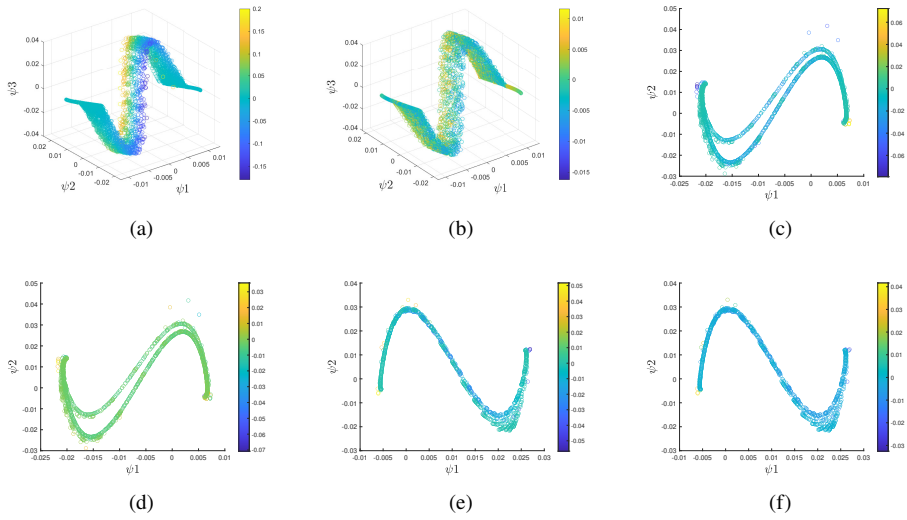


Figure 7.6: (a) and (b): The three parsimonious DMaps coordinates for $\varepsilon = 0.01114$ near the Andronov-Hopf point, respectively. (c) and (d): the two parsimonious DMaps coordinates for $\varepsilon = 0.4010$. (e) and (f): the two parsimonious DMaps coordinates for $\varepsilon = 0.9383$ near the turning point. Colors represent $u_t$ ((a), (c), (e)) and $v_t$ ((b), (d), (f)).

accuracy for all metrics. For the FNNs, the training phase (using the deep-learning toolbox in Matlab R2020b) required $\sim$ 1000 epochs and $\sim$ 4 hours, with the minimum tolerance set to $1E$-07.

Differences between the predicted $\hat{u}_t(x,t)$ and the actual values of the time derivatives $u_t(x,t)$ for three different values of $\varepsilon$ are shown in Figure 7.4(a)-(c) when using FNNs and in Figure 7.4(d)-(f) when using RPNNs. Note $u_t$ is reported indicatively, $\hat{v}_t(x,t) - v_t$ obtains similar results.

Instead, for the proposed RPNN scheme, the training phase, i.e., the solution of the least-squares problem with regularization, required around 8 minutes, thus resulting in a training phase of at least 20 times faster than that of the FNNs.
After training, we used the FNNs and RPNNs to compute with FD the quantities required for performing the bifurcation analysis (see Eq.(7.15)), i.e.:

$$\frac{\partial \hat{F}^u}{\partial u_j} = \frac{\hat{F}^u(u_j, v_j, \varepsilon) - \hat{F}^u(u_j + \delta, v_j, \varepsilon)}{2\delta}; \frac{\partial \hat{F}^u}{\partial v_j} = \frac{\hat{F}^u(u_j, v_j, \varepsilon) - \hat{F}^u(u_j, v_j + \delta, \varepsilon)}{2\delta}$$
$$\frac{\partial \hat{F}^v}{\partial u_j} = \frac{\hat{F}^v(u_j, v_j, \varepsilon) - \hat{F}^v(u_j + \delta, v_j, \varepsilon)}{2\delta}; \frac{\partial \hat{F}^v}{\partial v_j} = \frac{\hat{F}^v(u_j, v_j, \varepsilon) - \hat{F}^v(u_j, v_j + \delta, \varepsilon)}{2\delta}$$
$$\frac{\partial \hat{F}^u}{\partial \varepsilon} = \frac{\hat{F}^u(u_j, v_j, \varepsilon) - \hat{F}^u(u_j, v_j, \varepsilon + \delta)}{2\delta}; \frac{\partial \hat{F}^v}{\partial \varepsilon} = \frac{\hat{F}^v(u_j, v_j, \varepsilon) - \hat{F}^v(u_j, v_j, \varepsilon + \delta)}{2\delta},$$

with $\delta = 1E$-06. The reconstructed bifurcation diagrams are shown in Figure 7.5.

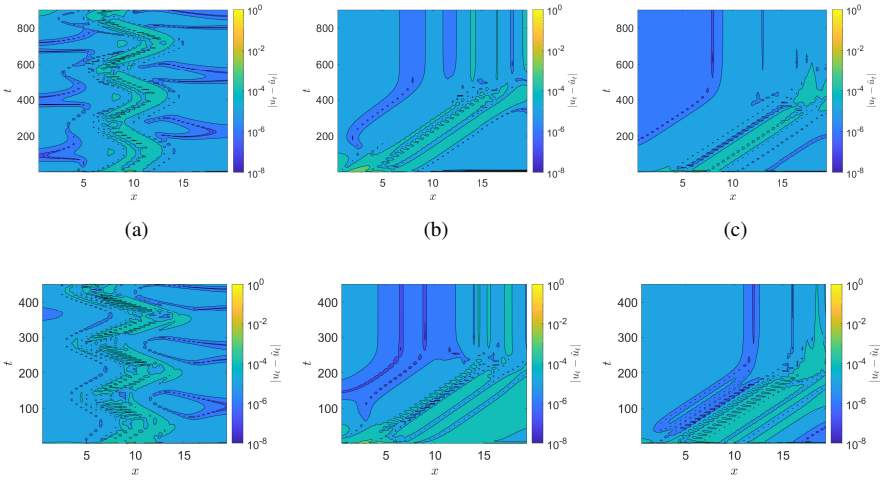



Figure 7.7: Approximation accuracy in the test data with feature selection, as obtained with the FNNs and RPNNs. Contour plot of the absolute values of differences in space and time, of, indicatively, $|u_t(x,t) - \hat{u}_t(x,t)|$ for characteristic values of $\varepsilon$: (a) and (b) $\varepsilon = 0.0114$ near the Andronov-Hopf point, (c), (d) $\varepsilon = 0.4$, (e) and (f) $\varepsilon = 0.9383$ near the turning point.

Using the FNNs, we estimated the Andronov-Hopf point at $\varepsilon \approx 0.0191$ and the turning point at $\varepsilon \approx 0.9713$; using the RPNNs, we estimated the Andronov-Hopf point at $\varepsilon \approx 0.0193$ and the turning point at $\varepsilon \approx 0.9696$. We approximated the same points using the FD scheme in the previous section at $\varepsilon \approx 0.0183$ for the Andronov-Hopf point and at $\varepsilon \approx 0.9446$ for the turning point. Hence, compared to the FNNs, the RPNNs approximated slightly better the reference turning point.

### *Numerical bifurcation analysis with feature selection*

We used Diffusion Maps (setting the width parameter of the Gaussian kernel to $\sigma = 10$) to identify the three parsimonious leading eigenvectors as described in section 6.4.2. We denote them as $\phi_1, \phi_2, \phi_3$. The three parsimonious DMaps coordinates for different values of the parameter $\varepsilon$ are shown in Figure 7.6. For $\varepsilon = 0.114$ that is close to the Andronov-Hopf point, the embedded space is a two-dimensional "carpet" in the three-dimensional space. The oscillatory behavior leads to different values of the time derivative which can be effectively parametrized as shown by the coloring of the manifold (Figures 7.6(a), 7.6(b)). For $\varepsilon = 0.4010$ and $\varepsilon = 0.9383$, the embedded space is a one dimensional line, since time derivatives converges rapidly to zero (Figures 7.6(c),7.6(e),7.6(d) and 7.6(f)). Based on the feature selection methodology, the "good" subsets of the input data domain are presented in Table 7.2. As expected, the best candidate features are the $(u, v, u_{xx})$ for $u_t$ and $(u, v, v_{xx})$ for $v_t$, which are the only features that indeed appear in the closed form of the FHN PDEs.

| | $u_t = (\phi_1^u, \phi_2^u, \phi_3^u)$ | | $v_t = (\phi_1^v, \phi_2^v, \phi_3^v)$ | |
|---|---|---|---|---|
| | **Features** | **Total Loss** | **Features** | **Total Loss** |
| **1d** | $(u)$ | 4.3E-03 | $(u)$ | 7.6E-03 |
| **2d** | $(u, v)$ | 6.37E-06 | $(u, v)$ | 1.91E-05 |
| **3d** | $(u, v, u_{xx})$ | 2.77E-07 | $(u, v, v_{xx})$ | 6.29E-07 |
| **4d** | $(u, v, u_x, u_{xx})$ | 1.03E-07 | $(u, v, v_x, v_{xx})$ | 1.34E-07 |

Table 7.2: The "best" set of variables that effectively parametrize the intrinsic coordinates ($(\phi_1^u, \phi_2^2, \phi_3^u)$ and $(\phi_1^v, \phi_2^v, \phi_3^v)$) and the corresponding sums of total losses across all the values of the bifurcation parameter $\varepsilon$.

Finally, we repeated the same steps, but now using as inputs in the FNNs and RPNNs the reduced input domain as obtained from the feature selection process. Table 7.1 summarizes the performance of the schemes on the test set. Figures 7.7(a)-(c) and 7.7(d)-(f) illustrate the norms of the differences between the predicted from the FNNs and RPNNs and the actual time derivatives of, indicatively, $u$.

Hence, as it is shown, the proposed feature selection approach based on the parsimonious Diffusion Maps revealed correctly the structure of the embedded PDEs in the form of:

$$\begin{aligned} \frac{\partial u(x,t)}{\partial t} &= \hat{F}^u(u(x,t), v(x,t), u_{xx}(x,t), \varepsilon), \\ \frac{\partial v(x,t)}{\partial t} &= \hat{F}^v(u(x,t), v(x,t), v_{xx}(x,t), \varepsilon) \end{aligned} \tag{7.16}$$

where $\hat{F}^u$ and $\hat{F}^v$ are the outputs of the FNNs (or the RPNNs). The constructed bifurcation diagram with feature selection is shown in Figure 7.5. Using the FNNs, we estimated the Andronov-Hopf point at $\varepsilon \approx 0.0195$ and the turning point at $\varepsilon \approx 0.9762$. Using the RPNNs, we estimated the Andronov-Hopf point at $\varepsilon \approx 0.0192$ and the turning point at $\varepsilon \approx 0.9752$.

## 7.2 Case study 2: Tipping points in a financial market with mimesis.

ABMs enable the creation of digital twins for financial markets, thus offering a valuable tool in our arsenal for explaining out-of-equilibrium phenomena such as “bubbles" and crashes [5] that emerge mainly due to positive feedback mechanisms of imitation and herding of investors that lead to an escalating increase of the demand. While the practical application of ABMs for providing predictions about real-world financial instabilities remains an ongoing area of research, they can be used to shed light on the mechanisms that lead to such crises [5].

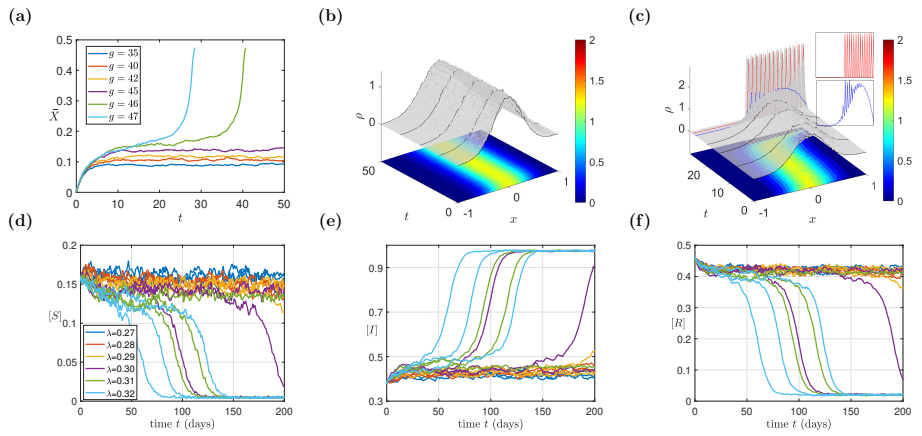



Figure 7.8: Stochastic Agent-based model simulations of the two case studies. (a)-(c) traders in a simple financial market. (a) Trajectories for different values of the parameter $g$. (b) Probability density function (pdf) evolution for $g = 45$; (c) pdf evolution for $g = 47$ (past the tipping point); Insets show the blow up of the pdf; the blue curve depicts the pdf just a few time steps before the explosion and the red curve depicts the pdf at the financial “bubble”. (d)-(f) Stochastic simulations of the epidemic ABM. Trajectories of the densities [S] in (d), [I] in (e) and [R] in (f), for different values of the parameter $\lambda$.

Towards to this aim, our first illustrative example is an event-driven ABM approximating the dynamics of a simple financial market with mimesis proposed by Omurtag and Sirovich [3]. The ABM describes the interactions of a large population of, say $N$, financial traders. Each agent is described by a real-valued state variable $X_i(t) \in (-1, 1)$

associated to their tendency to buy (positive values) or sell (negative values) stocks in the financial market according to constantly updated financial news, as well as to their interactions with the other traders [3]. The $i$–th agent acts, i.e., buys or sells, only when its state $X_i$ crosses one of the decision boundaries/thresholds $X = \pm 1$. As soon as an agent $i$ buys or sells, the agent's state is forthwith reset to zero.

In the absence of any incoming good news $I_i^+$ or bad news $I_i^-$, the preference state exponentially decays to zero with a constant rate $\gamma$. Thus, each agent is governed by the following SDE:

$$dX_i(t) = -\gamma X_i(t)dt + dI_i^+(t) + dI_i^-(t), \qquad |X_i| < 1. \tag{7.17}$$

The effect of information arrivals $I_i^{\pm}(t)$ is represented by a series of instantaneous positive/negative "discrete jumps" of size $\epsilon^{\pm}$, arriving randomly at Poisson distributed times $t_{k^+}$, $k^+ = 1, 2, \ldots$ and $t_{k^-}$, $k^- = 1, 2, \ldots$, with average rates of arrival $\nu^+(t)$ and $\nu^-(t)$, respectively. Furthermore, the dynamics of each agent are driven by arrivals of two types of information: *exogenous* (ex) (e.g., publicly available financial news), as well as an *endogenous* (en) stream of information arising from the social connections of the agents, so that

$$\nu^{\pm} = \nu_{ex}^{\pm} + \nu_{en}^{\pm}. \tag{7.18}$$

A tunable parameter $g$ embodies *the strength of mimesis*: the extent to which arriving information affects the willingness or apprehension of the agent to buy or sell. For this model, the term $\nu_{en}^{\pm}$ is set to be the same for all agents and is influenced by the perceived overall buying $R^+(t)$ and selling $R^-(t)$ rates:

$$\nu^{\pm}(t) = \nu_{ex}^{\pm} + gR^{\pm}(t), \tag{7.19}$$

where $R^{\pm}(t)$ are defined as the fraction of agents buying or selling per unit of time $\Delta t$:

$$\begin{aligned} R^{\pm}(t) &= \frac{\text{number of agents buying/selling}}{\Delta t \cdot \text{total number of agents}} = \\ &= \frac{1}{N\Delta t}\int_t^{t+\Delta t} \delta(s - T_i^{\pm})ds, \end{aligned} \tag{7.20}$$

where $T_i^{\pm}$ are the instants at which the $i$-th agent crosses the decision boundary $\pm 1$.

In Figure 7.8(a), we depict the mean preference state for $N = 50,000$ agents for values of the mimesis strength $g = 35, 40, 42, 45, 46, 47$. In Figures 1(b)-1(c), we depict representative trajectories of the time evolution of the agent probability density distribution (pdf) for $g = 45$ and $g = 47$, respectively. We see that the simulations exhibit a tipping point that arises at a parameter value $g \approx 45.5$. At the neighborhood of this tipping point, due to the inherent stochasticity of the mimetic trading process, emanate "financial bubbles", where all agents hurry to buy assets (see Figure 7.8(a)). The ABM model also predicts financial crashes in regimes of the phase-space where the mean value of the mesoscopic density field is negative, and the agents rush to sell (for more details see [28]).

A concise analytical mesoscopic description of the population dynamics was derived by Omurtag and Sirovich in [3]. The model, reported here, is a Fokker-Planck-type (FP) IPDE for the agent pdf $\rho(x,t)$, given by:

$$\frac{\partial \rho(t,x)}{\partial t} = \frac{1}{2}\sigma^2(t)\frac{\partial^2 \rho(t,x)}{\partial x^2} + \frac{\partial(\mu(t,x)\rho(t,x))}{\partial x} + \\ + (J^+ + J^-)\delta(x). \tag{7.21}$$

where $\mu$ and $\sigma$ are drift and diffusivity time-dependent parameters, respectively, $\delta$ is the Dirac delta and $J^{\pm}$ are integral operators accounting for the agents crossing the decision boundaries.

Further details about the derivation of the FP equation in Eq. (7.21) are presented in the next section.

### 7.2.1 Mesoscopic dynamics: the analytically derived Fokker-Planck-type IPDE

In order to obtain a concise description at the level of population dynamics, Omurtag and Sirovich derived a mesoscopic Fokker-Plank-type (FP) IPDE [3] (a continuity equation) for the agent pdf $\rho(t,x)$, where the spatial variable corresponds to the preference state of the agents, i.e., $x \equiv X$. For the particular problem, at the limit of infinitely many agents and averaged along many possible trajectories, the continuity equation in terms of a probability flux $J(t,x)$ and a source $Q(t,x)$ reads:

$$\begin{aligned} &\frac{\partial \rho(t,x)}{\partial t} = -\frac{\partial J(t,x)}{\partial x} + Q(t,x), \\ &\text{with } J(t,x) = -\frac{1}{2}\sigma^2\frac{\partial \rho(t,x)}{\partial x} - \mu\rho(t,x), \end{aligned} \tag{7.22}$$

where, if we denote the fluxes at the boundaries $J(t,\pm 1) = J^{\pm}$, the source $Q(t,x)$ can be set to be $Q(t,x) = (J^+ + J^-)\delta(x)$ to compensate for the creation/disappearance of density at the boundaries.

The above equation can be written as:

$$\frac{\partial \rho(t,x)}{\partial t} = \frac{1}{2}\sigma^2(t)\frac{\partial^2 \rho(t,x)}{\partial x^2} + \frac{\partial(\mu(t,x)\rho(t,x))}{\partial x} + \\ + (J^+ + J^-)\delta(x). \tag{7.23}$$

In the above, $\sigma^2(t)$ is the time-dependent diffusivity coefficient given by:

$$\sigma^2(t) = \nu^+(\epsilon^+)^2 + \nu^-(\epsilon^-)^2, \tag{7.24}$$

$\mu$ is the time-dependent, space-dependent drift coefficient, given by:

$$\mu(t,x) = \gamma x - \nu^+\epsilon^+ - \nu^-\epsilon^-. \tag{7.25}$$

$J^{\pm}$ denote the *inflow* and *outflow* through the boundaries that are restored/re-injected at the origin through a Dirac $\delta(x)$ in Eq. (7.23), in order to maintain agent conservation. Indeed,

since $\rho$ is a probability distribution, the equation has also to satisfy the normalization property:

$$\int_{-1}^{1} \rho(t,x)dx = 1, \quad \forall t; \tag{7.26}$$

it is convenient to consider homogeneous Dirichlet boundary conditions $\rho(\pm 1, t) = 0$ and the agents there instantaneously reset back to $X = 0$. In addition, the fluxes at the boundaries $J(t, \pm 1) = J^{\pm}$ are given by:

$$J^{\pm}(t,x) = \mp \frac{1}{2}\sigma^2 \frac{\partial \rho(t,x)}{\partial x}\Big|_{x=\pm 1}, \tag{7.27}$$

reflecting the resetting process of agents that cross the boundaries.

Besides, in order to solve/integrate the Eq. (7.23), one has to find algebraic closures for the time-evolving diffusivity $\sigma$ and drift $\mu$ coefficients. In [3], a mean field approximation of the buying/selling rates was proposed:

$$R^{\pm} = \pm \nu^{\pm} \int_{\pm 1 \mp \epsilon^{\pm}}^{\pm 1} \rho(x,t)dx. \tag{7.28}$$

Finally, based on Eq. (7.19): and Eq. (7.28), one obtains:

$$\nu^{\pm} = \frac{\nu_{ex}^{\pm}}{1 - g\int_{\pm 1 \mp \epsilon^{\pm}}^{\pm 1} \rho(x,t)dx}, \tag{7.29}$$

from which one can retrieve at each time instance $t$, the coefficients $\mu(t)$ and $\sigma^2(t)$ of the FP model, as defined in Eqs. (7.24)-(7.25).

The designation “Fokker-Planck" notwithstanding, it is important to restate that the above approximation is an IPDE with space-dependent coefficients.

### 7.2.2 ML mesoscopic IPDE surrogate for the financial ABM.

In [28], it was shown that the analytical ROM IPDE in Eq. (7.21) non-trivially underestimates the location of the tipping point with respect to the parameter $g$, defined in Eq. (7.19).

Here, we show how one can achieve a better approximation through data-driven black-box surrogates. For learning the right-hand-side operator of the IPDE, we have considered the relevant features that we found with ARD (see Sections 6.4.1 and 7.2.3). We used the following (black-box) mesoscopic model for the dynamic evolution of the density $\rho$:

$$\frac{\partial \rho(x,t)}{\partial t} = F(x, \rho(x,t), I^+, I^-, \frac{\partial \rho(x,t)}{\partial x}, \frac{\partial^2 \rho(x,t)}{\partial x^2}; g) \tag{7.30}$$

where $I^+, I^-$ are integrals in a small neighborhood of the boundaries (see Section 7.2.3).Here, for learning the RHS of the black-box IPDE (7.30), we implemented two different structures, namely (a) an FNN; and (b) a RPNN in the form of RF [158].

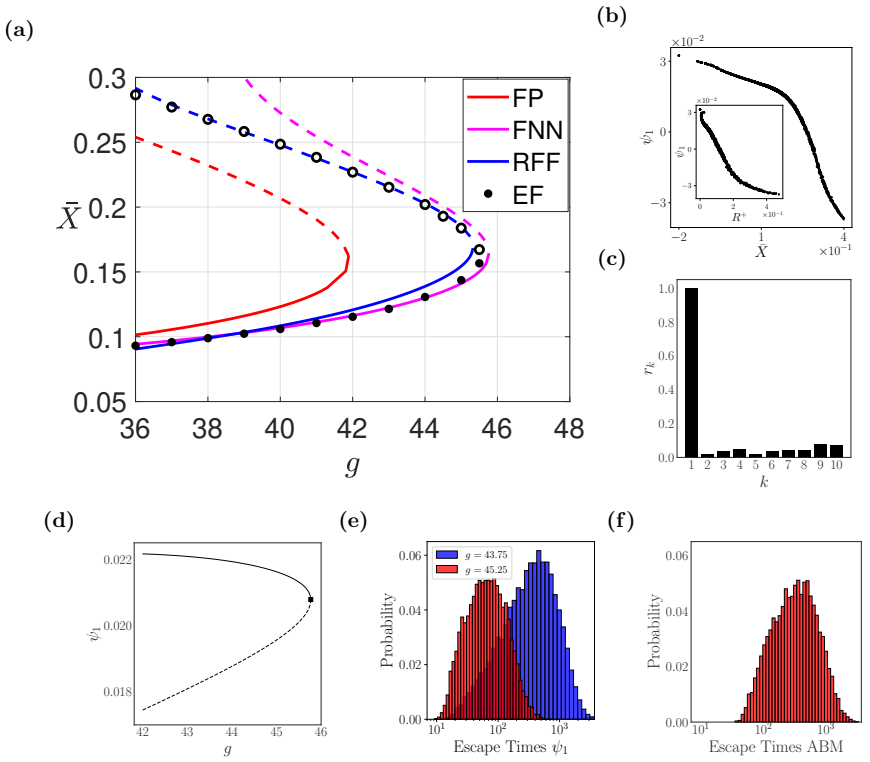



Figure 7.9: Numerical results for the financial ABM. (a) Reconstructed bifurcation diagram w.r.t. $g$ obtained with the mesoscopic IPDE surrogate FNN and RFF models; the one computed from the analytical Fokker-Planck (FP) IPDE, see Eq. (7.21), and the one constructed with the Equation-free (EF) approach are also given[28, 131]. Dashed lines (open circles) represent the unstable branches. (b)-(c) ABM-based-simulation- and Diffusion Maps- (DMaps) driven observables; (b) The first DMaps coordinate $\psi_1$ is plotted against the mean preference state $\bar{X}$. In the inset, the buying rate $R^+$ plotted against the mean preference state ($\bar{X}$). (c) The estimated residual $r_k$ based on the local linear regression algorithm [146]. (d) The effective bifurcation diagram based on the drift component of the identified mean-field macroscopic SDE in $\psi_1$. (e-f) Histograms of escape times obtained with simulations of 10,000 stochastic trajectories for (e) the SDE model for $g = 45.25$ (blue histogram) and $g = 43.75$ (red histogram) (f) the full ABM at $g = 45.25$.

The two alternative ML schemes, on the test set, obtain similar performance in terms of accuracy. In terms of the Mean Absolute Error (MAE) the FNN got 1.10E−04 and the RFF got 1.09E−04. For MSE, the FNN got 2.60E−08 and the RFF got 2.53E−08. For the Regression Pearson correlation $R$, FNN got 0.9866 and RFF 0.9873. The main notable difference between the two schemes is in the computational time needed to

perform the training, since remarkably, the training of the RFF, which required 26.06 (s), turned out to be at least 50 times faster than the one required for the deep-learning scheme, which required 1488.33 (s).

**Details on Learning the Integro- Partial Differential Equation for the finance ABM.** ABM simulations were performed using $N = 50,000$ agents, with $\nu_{ex}^{+} = \nu_{ex}^{-} = 20$, $\gamma = 1$, $\epsilon^{-} = -0.072$, $\epsilon^{+} = 0.075$. The mimetic strength $g$ is our bifurcation parameter. For the data set, we selected 41 equally-spaced points in the range $g \in [30, 50]$ and for each of the values of $g$, we randomly generated 1000 different initial profiles $\rho_0$, as Gaussian distributions $\rho_0 \sim \mathcal{N}(\tilde{m}_0, \tilde{s}_0^2)$ with varying mean $\tilde{m}_0$ and variance $\tilde{s}_0^2$. The initial $\tilde{m}_0$ and $\tilde{s}_0$ were uniformly randomly sampled as $\tilde{m}_0 \sim \mathcal{U}([-0.3, 0.3])$, $\tilde{s}_0 \sim \mathcal{U}([0.3, 0.5])$. The initial state of the agents is sampled from the initial distribution $\rho_0$, creating a consistent microscopic realization. At each time step, as the agents dynamically evolve, to estimate the corresponding coarse-grained density profile, we used 81 equally-spaced bins, with equally-spaced centers $x_i \in [-1, 1]$. Since we are dealing with a stochastic model, in order to generate smooth enough profiles for the spatial derivatives, for each fixed initial condition, we ran 100 random stochastic realizations, and we averaged along the generated copies of the density field. Then to further smooth out the computed densities, we applied a weighted moving average smoothing $\rho_i$ as $\rho_i^{\star} = \frac{2\rho_i + \rho_{i-1} + \rho_{i+1}}{4}$. ($i$ denotes spatial mesh points).

The ABM simulations were run for a time interval $t \in [0, 15]$ and we collected points with a time step of $\Delta t = 0.25$. When the mean value $\bar{X}$ crossed, $\pm 0.4$ we stopped the simulations, because the pdf profile blows up, very fast after that. Furthermore, we also ignored the first two time points, to exclude the initial "healing" transients due to the way we initialize (see the discussion for such healing periods in [19]). We thus end up with a data set consisting of 40 (values of $g$) $\times 100$ (initial conditions) $\times 58$ (maximum time points ignoring the first 2 steps of the transient) $\times 81$ (space points) $\approx 15 \times 10^6$ data points. Since the amount of data is practically too large, for the training set we have randomly downsampled to $10^6$ data and used the remaining data as our test set.

### 7.2.3 Feature selection for the mesoscopic IPDE finance model

For dealing with the "curse of dimensionality" in training the FNN, learning our IPDE model, we used ARD as implemented in `Matlab` by the function `fitrgp` for feature selection. Here, we *a priori* selected as candidate features, the space $x$ *per se*, the field $\rho$, the first $\rho_x$, second $\rho_{xx}$ and third $\rho_{xxx}$ spatial derivatives estimated with central FD, as well as $I^{\pm}$ defined as the integrals of $\rho$ in a small region close to the boundaries:

$$I^{\pm}(t) = \pm \int_{\pm 1 \mp \epsilon}^{\pm 1} \rho(x, t) dx, \tag{7.31}$$

where $\epsilon = 0.05$ corresponds to the size of the last two bins of the grid. We note that the latter two candidate mesoscopic variables $I^{\pm}$ as defined above are related to the buying and selling rates (see Eq.7.28), yet they do not depend on the frequencies $v^{\pm}$ as the true buying and selling rates do. These "internal"/hidden variables $v^{\pm}$ are considered

unknown. We also consider unknown the "quantum" jump sizes $\epsilon^{\pm}$. The target variable to learn is the time derivative, at each collocation point $x$, estimated with forward FD as:

$$\rho_t(x,t) = \frac{\partial \rho(x,t)}{\partial t} = \frac{\rho(x,t+dt) - \rho(x,t)}{dt}. \tag{7.32}$$

The effective relevance weights $W_r(\cdot)$ of the features, as obtained by using ARD, are $W_r(\rho) = 0.25$, $W_r(\rho_x) = 0.22$, $W_r(\rho_{xx}) = 0.14$, $W_r(\rho_{xxx}) = 0.04$, $W_r(I^+) = 0.18$, $W_r(I^-) = 0.12$, $W_r(x) = 0.27$. As can be noted, the third derivative is the least important feature, and we thus decided to disregard it; the most important feature is the space $x$ highlighting that the IPDE is not translational invariant. The dependency on $x$ implicitly captures the location of the source term $Q$ representing the resetting of the state of the agents at the origin, as in the FP IPDE Eq. (7.23). Note that the integral features $I^{\pm}$ are also important. This is in line with the theoretical results regarding the FP IPDE Eq. (7.23).

### 7.2.4 Bifurcation analysis of the mesoscopic IPDE for the financial ABM.

To locate the tipping point, we have performed bifurcation analysis, using both ML-identified IPDE surrogates. Furthermore, we compared the derived bifurcation diagram(s) and tipping point(s) with what was obtained in [28, 131] using the EF approach (see in the section B.5.2 for a very brief description of the EF approach). As shown in Fig. 2(a) the two ML schemes approximate visually accurately the location of the tipping point in parameter space. However, the FNN scheme fails to trace accurately the actual coarse-scale unstable branch, near which simulations blow up extremely fast. More precisely, the analytical FP predicts the tipping point at $g^* = 41.90$ with corresponding steady-state $\bar{X}^* = 0.1607$ and the EF at $g^* = 45.60$ and $\bar{X}^* = 0.1627$; our FNN predictions are at $g^* = 45.77$ and $\bar{X}^* = 0.1644$, the RFF ones at $g^* = 45.34$ and $\bar{X}^* = 0.1684$.

### 7.2.5 Macroscopic physical observables and latent data-driven observables via DMaps.

An immediate physically meaningful candidate observable is the first moment $\bar{X}$ of the agent distribution function (as also shown in [52]).

As simulations of the ABM show (see the inset in Figure 7.9(b), the mean preference state $\bar{X}$, is one-to-one with another physically meaningful observable, the buying rate $R^+$. We also used the DMaps algorithm, to discover *data-driven* macroscopic observables. In our case, DMaps applied to collected data, discovers a 1D latent variable $\psi_1$ that is itself one-to-one with $\bar{X}$, see Figure 7.9(b). The local-linear regression algorithm proposed in [146] was applied to make sure that all the higher eigenvectors can be expressed as local-linear combinations of $\psi_1$ and thus they do not span independent directions. Figure 7.9(c) illustrates that the normalized leave-one-out error, denoted as $r_k$, is small for $\psi_2, \ldots, \psi_{10}$ suggesting they are all dependent/harmonics of $\psi_1$.

Therefore, any of the three macroscopic observables (two physical and one data-driven) can be interchangeably used to study the collective behavior of the model.

### 7.2.6 Learning the mean-field SDE and performing bifurcation analysis for the financial ABM.

Here, for our illustrations, we learned parameter−dependent SDEs for all the three coarse variables we mentioned, namely the physically meaningful variables $\bar{X}$, $R^+$, and the DMaps coordinate $\psi_1$. Here, we report the results for the identified SDEs in terms of the DMaps coordinate $\psi_1$, while later in section 7.2.9 we will report the additional results for the identified SDEs with respect to $\bar{X}$ and $R^+$ (see Figure 7.10).

We used three neural networks to identify, respectively, three alternative different one-dimensional mean-field SDEs based on the DMaps coordinate $\psi_1$, mean preference state $\bar{X}$ and buying rate $R^+$. Each network had 5 hidden layers with 32 neurons per layer. The activation function for the drift network was `tanh`, and for the diffusivity network, it was `softplus`. The data were divided into a 90|10 training|validation split. Given this trained macroscopic SDE surrogate, the drift term (deterministic component) of the identified dynamics was used to construct the bifurcation diagram with AUTO [226] (see Figure 7.9(d). A saddle-node bifurcation was identified for $g^* = 45.77$ where $\psi_1^\star = 0.021$. The estimated critical parameter value from the SDE is in agreement with our previous work ($g \approx 45.60$ [28]).

**Details on Data collection and preprocessing Learning the SDE of the financial ABM.** In this section we describe how we collected data, specifically targeted to the neighborhood of the tipping point, for the purpose of learning a parametric mean-field SDE. ABM simulations were performed using $N = 50,000$ agents, with $\nu^{\pm} = 20$, $\epsilon^+ = 0.075$, $\epsilon^- = -0.072$, $\gamma = 1$. We selected 11 equally-spaced values of the mimetic strength $g$, which is used as bifurcation parameter, in the range $g \in [42, 47]$. We gathered at each time stamp the state of every agent ($X_i$), as well as the overall buying/selling rates ($R^+$ and $R^-$). For convenience, we use the vector $\boldsymbol{s} = (X_1, X_2, \cdots, X_n, R^+, R^-)^T$, and denote the mean preference value of the agent state as:

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i.$$

To select initial conditions that populate the state space (in terms of $\bar{X}$, across multiple values of the parameter $g$) the following protocol was used: We start by running one trajectory of the full ABM for $g = 47$, which ultimately leads to an explosion. This trajectory was initialized by sampling the agents from a triangular distribution $p(X)$, as implemented in python, with lower limit $-1$, upper limit 1, and mode $-0.6$, corresponding to a value $\bar{X} \approx -0.2$. This trajectory was stopped using the termination condition $\bar{X} \leq 0.4$, indicative of incipient explosion. We selected 25 distinct instances of the state of agents along the stochastic trajectory, corresponding to 25 different values of $\bar{X}$ in the range $[-0.02, 0.32]$. Then, for each value of the parameter $g$, we simulated a total of 50 new stochastic trajectories, two for each distinct initial condition.

As previously done in [52], in order to find the data-based one-dimensional coarse-scale observable $\psi_1$, the DMaps algorithm is carried out on 39 intermediate coarse variables. Thus, we set up 37 percentile points $p_1, p_2, \cdots, p_{37}$ referring to our discretization of the cumulative distribution function (cdf) of the agents' preference state. For each $p_i$, we computed its quantile function value $Q(p_i)$, where the quantile function $Q(\cdot)$ is defined as the inverse function of cdf $F_X$ of the random variable $X$. The first 19 percentile values $p_1, p_2, \cdots, p_{18}, p_{19}$ are set as 0.0005, 0.001, 0.002, 0.003, 0.004, 0.005, 0.0075, 0.01, 0.02, 0.03, 0.04, 0.05, 0.075, 0.1, 0.15, 0.2, 0.3, 0.4, 0.5. The last 18 probability values are set to be symmetric with the first 18, i.e., $p_i = 1 - p_{38-i}, \quad \forall i = 20, 21, \cdots, 37$; The remaining two coarse variable are the overall buying and selling rates. Therefore, the full state $\boldsymbol{s}$ has a coarse 39-dimensional representation $\boldsymbol{s'} = (Q(p_1), Q(p_2), \cdots, Q(p_{37}), R^+, R^-)^T$.

### 7.2.7 Rare-event analysis/UQ of catastrophic shifts (financial "bubbles") via the identified mean-field SDE.

Given the identified steady states at a fixed value $g$, we performed escape time computations. For $g = 45.25$, we estimated the average escape time needed for a trajectory initiated at the stable steady state to reach $\bar{X} = 0.3$, i.e., sufficiently above the unstable branch. As shown in Figure 7.9(b), $\psi_1$ and $R^+$ are effectively one-to-one with $\bar{X}$, and we can easily find the corresponding critical values for $\psi_1 = -0.01$ (flipped) and $R^+ = 0.16$. We now report a comparison between the escape times of an SDE identified based on the DMaps coordinate $\psi_1$ and those of the full ABM. In section 7.2.9 we also report the escape times of the SDE for $\bar{X}$ and $R^+$ observables. To estimate these escape times, we sampled a large number ($10,000$ in our case) of trajectories.

In Figure 7.9(e) the histograms of the escape times for the identified SDE trained on $\psi_1$ for $g = 45.25$ and $g = 43.75$ are shown. In Figure 7.9(f), we also illustrate the empirical histogram of escape times of the full ABM for $g = 45.25$. The estimated values for the mean and standard deviation, as computed with temporal simulations from the SDE trained on the DMaps variable $\psi_1$ for $g = 45.25$, $g = 43.75$ and the full ABM at $g = 45.25$ are here reported in Table 7.3.

Table 7.3: Escape time computations for the financial ABM. Means and Standard deviations as computed with temporal simulations from the SDE trained on the DMaps variable $\psi_1$ for $g = 45.25$, $g = 43.75$ and the ABM at $g = 45.25$ respectively.

| Models | SDE at $g = 45.25$ | SDE at $g = 43.75$ | ABM |
|---|---|---|---|
| Mean Escape Time | 84.07 | 480.92 | 434.00 |
| Escape Time Standard deviation | 68.91 | 454.83 | 363.64 |

As shown, the mean escape time of the full ABM is a factor of five larger than that estimated by the simplified SDE model in $\psi_1$ for $g = 45.25$ (still within an order of magnitude!). The SDE model for $g = 43.75$ gives an escape time comparable to the one of the ABM for $g = 45.25$. Given that the escape times change exponentially with respect to the parameter distance from the actual tipping point, a small error in the

identified tipping point easily leads to large (exponential) discrepancies in the estimated escape times.

### 7.2.8 Computational cost for the financial ABM.

We compared the computational cost required to estimate escape times with many stochastic temporal simulations, through the full ABM and the identified mean-field SDE. To fairly compare the computational costs, we computed the escape times with the ABM for $g = 45.25$, and that of the SDE for $g = 43.75$, since the two distributions of the escape times are more comparable. The estimation in both cases was conducted on *Rockfish* (a community-shared cluster at Johns Hopkins University) by using a single core with 4 GB RAM. For the 10,000 sampled stochastic trajectories, the total computational for the identified coarse SDE in $\psi_1$ was 33.56min and the average time per trajectory, $3.36 \times 10^{-3}$min. The mean time per function evaluation was approximated as the ratio of mean time per trajectory over mean number of iterations.

For the ABM, the total computational time needed was 18.56 days and the mean time per trajectory was 2.67min. Therefore, the total computational time for computing the escape time with the SDE model in $\psi_1$ was around 800 times faster than the ABM. This highlights the computational benefits of using the reduced surrogate models in lieu of the full ABM for escape time computations.

### 7.2.9 Additional Results: Two physical based alternative mean-field SDEs

In this section, we illustrate results obtained for two alternative one-dimensional mean-field SDE surrogates, using as their effective state variable $\bar{X}$ and $R^+$, respectively; these were omitted in the main text for brevity.

The constructed bifurcation diagrams from the identified deterministic drift of the two SDE surrogates trained on $\bar{X}$ and $R^+$ are shown in Figure 7.10(a)-7.10(b). The bifurcation point, in both cases, occurs for $g^* \approx 45.5$. Indeed, we obtained a tipping point at $g^* = 45.90$ with a corresponding $\bar{X}^* = 0.1751$ for the SDE trained on $\bar{X}$; and at $g^* = 45.76$, with a corresponding $R^{+*} = 0.003586$ for the SDE trained on $R^+$. The identified bifurcation point (our tipping point) for all three identified SDE models is consistent with the value reported in Liu P. et al. [28, 52, 131].

For each of the identified SDEs, in terms of $\bar{X}$ and $R^+$ respectively, we estimated the escape time distribution by computing $10,000$ stochastic simulations. Histograms of the obtained escape times for $\bar{X}$ and $R^+$ are shown in Figure 7.10(c)-7.10(d), respectively. Means of escape times distribution and their standard deviations for the identified SDE in terms of $\bar{X}$ and $R^+$ here. We computed for the SDE on $\bar{X}$ a mean of 75627.73 and standard deviation 75198.20, for $g = 45.25$. While for $g = 45.61$ we got mean 468.06 and standard deviation of 456.71. For the SDE on $R^+$ at $g = 43.25$ we got mean 73.61 and standard deviation 63.95. While at $g = 43.15$ we got mean 436.59 and standard deviation 420.82. Similarly to the computations reported in the main text, for the SDE models in $\bar{X}$ and $R^+$ we estimated the mean escape times for $g = 45.25$; We also found

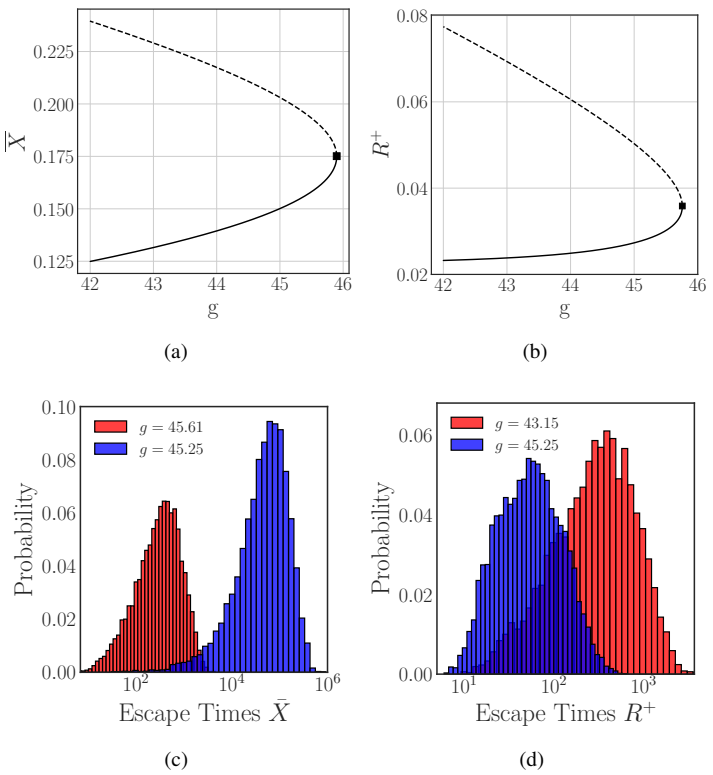



Figure 7.10: Additional results for two alternative SDEs surrogates. (a)-(b) The constructed bifurcation diagrams are based on the drift component of the identified effective SDEs in terms of (a) $\bar{X}$, and of (b) $R^+$. The bifurcation point is marked with a black square. (c)-(d) Histograms of escape times obtained with simulations of 10,000 stochastic trajectories for the SDE models trained on (c) $\bar{X}$, (d) $R^+$.

parameter values of $g$ that provide comparable mean escape time with that of the full ABM for $g = 45.25$.

The estimated escape times of the identified SDE in $\bar{X}$ for $g = 45.25$ is much larger than any other model (including the ABM). This might suggest that this surrogate model might be unreliable. The surrogate SDE model constructed in $R^+$ has an escape time similar to the model trained on $\psi_1$.

## 7.3 Case study 3: Tipping points in a compartmental epidemic model on a complex network

Compartmental models serve as structured population models, where the population is categorized based on their roles in the epidemiological process. We consider a stochastic version of a susceptible-infected-recovered-susceptible (SIRS) in discrete time. The rules that drive the model are presented in the main text. Here we give more details about the structure of the network on which the dynamics evolves. This network is constructed in a straightforward manner: within a population of $N$ nodes, it is assumed that each node can potentially be connected to any of the other $N-1$ nodes with a probability $p$. This implies that a node has an equal probability, denoted as $p$, of forming a connection with every other node in the network. Therefore, the degree distribution of Erdös-Rényi network follows the binomial law:

$$P(k) = \binom{N-1}{k} p^k (1-p)^{N-1-k}, \tag{7.33}$$

where $k$ denote a degree value. The mean degree of the network is $\mathbb{E}(k) = \bar{k} = pN$. This distribution for $p = \frac{1}{2}$ is exactly symmetric, while for other values of $p < 1/2$ is almost symmetric but with a "long tail", i.e., there is a very low probability for the occurrence of degrees $k > 2pN$. In our computation we selected $p = 0.0008$ and $N = 10,000$ which gives $\bar{k} = 8$. In particular along all computation we have used a predetermined Erdös-Rényi network, with the maximum degree of a node being $k_{max} = 21$.

- Rule 1 ($S \to I$): Susceptible individuals may become infected upon contact with infected individuals, with probability $P_{S\to I} = \lambda$. This tunable parameter is "tracked" for studying the outcomes of abrupt changes in the macroscopic behavior.

- Rule 2 ($I \to R$): The transition between $I$ and $R$ happens with a probability $P_{I\to R} = \mu([I])$. The probability of recovery depends, at each time step, on the overall density of infected individuals $[I]$, according to the function [12]):

$$\mu([I]) = 0.3\left(1 - \frac{1}{1+\exp(-9([I]-0.5))}\right). \tag{7.34}$$

  Such a non-linear function for the probability of recovery has also been used in other works to express the heterogeneity in the "environment" around each individual (see also the discussion in [12]).

- Rule 3 ($I \to R$): A recovered individual ($R$) loses its immunity and becomes susceptible ($S$) with a fixed probability $P_{R\to S} = \epsilon = \frac{1}{5}$. This condition expresses the case of temporal immunity.

The above rules establish a complex stochastic microscopic model that change the state of each individual over time. In order to describe the model at a macroscopic (emergent) level, let us represent the overall density of susceptible, infected, and recovered

individuals as $[S]$, $[I]$, and $[R]$, respectively. In Figure 7.8(d)-(f) we depict the stochastic trajectories of the overall densities $[S]$, $[I]$, $[R]$ of the $N = 10,000$ agents for values of the rate of infection $\lambda = 0.27, 0.28, 0.29, 0.30, 0.31, 0.32$. For such macroscopic observables, a simple analytical closure can be found assuming that a *uniform and homogeneous network* is a good approximation. This means assuming that (a) the degree of each node practically coincides with the mean degree of the network, denoted as $z = \mathbb{E}(k)$; (b) that the probabilities of two connected nodes being in a susceptible and infected state, respectively, are independent of each other. The resulting mean field model reads [12]:

$$\begin{aligned} \frac{d[S]}{dt} &= -\lambda z[S][I] + \epsilon[R], \\ \frac{d[I]}{dt} &= \lambda z[S][I] - \mu([I])[I], \\ [S] + [I] + [R] &= 1. \end{aligned} \tag{7.35}$$

For other higher-order analytical macroscopic pairwise closures, such as the Bethe Ansatz, the Kirkwood approximation and Ursell expansion, the interested reader can consult [12].

### 7.3.1 ML macroscopic mean-field surrogates for the epidemic ABM.

It is well known that for dynamics evolving on complex networks, a closed-form, analytically derived mean-field approximation in Eq. (7.35) is usually not accurate [12]. Here, we show how one can achieve a better approximation through the construction of effective mean field-level ML surrogate models. We will start with the identification, from data, of a mean field-level effective SIR model. Then, following the proposed approach, we identify -again from data- an effective one-dimensional SDE to model the stochastic dynamics close to the tipping point and quantify the probability of occurrence of an outbreak where all the population becomes infected.

Given the high-fidelity data collected from the epidemic ABM, to learn the ML mean field SIR surrogate we used two coupled FNN, labeled $F_S$ and $F_I$, each with two hidden layers with 10 neurons for each layer, for learning a black-box evolution for the effective dynamics of the two macroscopic densities $[S]$ and $[I]$, that reads:

$$\begin{aligned} \frac{d[S]}{dt} &= F_S([S], [I], [R]; \lambda), \\ \frac{d[I]}{dt} &= F_I([S], [I], [R]; \lambda), \end{aligned} \tag{7.36}$$

with the constraint $[S] + [I] + [R] = 1$. We remind the reader that the parameter$\lambda$, representing the probability of a susceptible individual to get infected, is tracked for bifurcation analysis purposes. The training process results in an MAE of $7.27E$-04 and an MSE of $1.27E$-06 on the test set. The regression error for the two networks was $R(F_S) = 0.9996$ and $R(F_I) = 0.9992$.

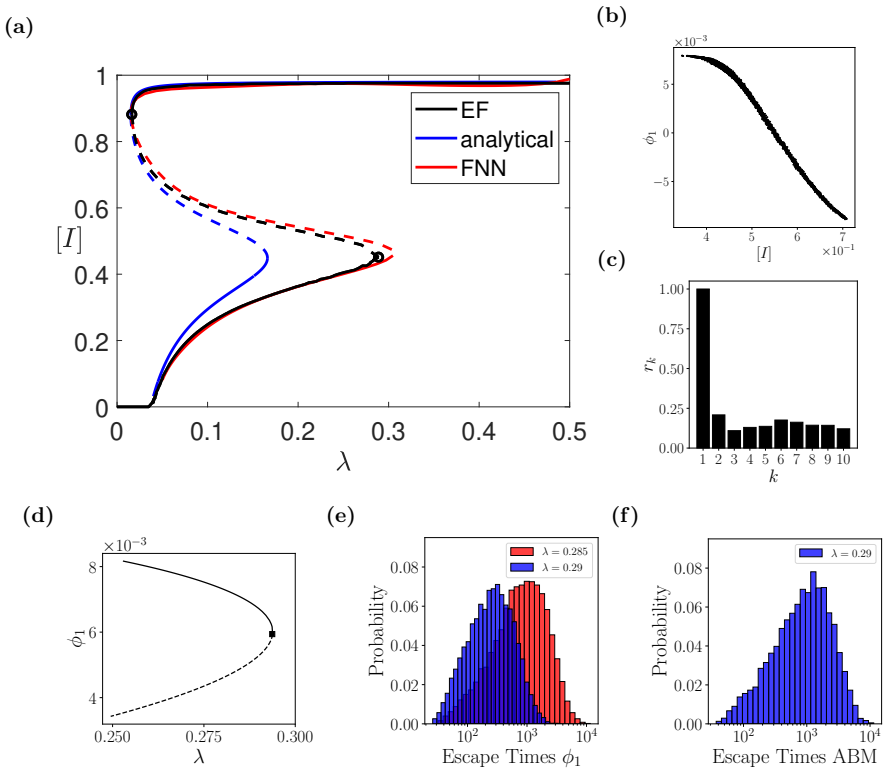



Figure 7.11: Numerical results for the epidemic ABM. (a) Reconstructed bifurcation diagram w.r.t. the probability of infection, $\lambda$, with the ML mean-field surrogate model; the one computed from the analytical mean-field Eq. (7.35), and the one constructed with the Equation-free (EF) approach are also given [12]. Dashed lines represent the unstable branches. (b)-(c) ABM-based simulations-and Diffusion Maps- (DMaps) driven observables. (b) the density of infected $[I]$ vs. the first DMaps coordinate $\phi_1$. (c) The estimated residual $r_k$ based on the local linear regression algorithm [146]. (d) The effective bifurcation diagram based on the drift component of the identified mean-field macroscopic SDE based on $\phi_1$. (e-f) Histograms of escape times obtained with simulations of $10,000$ stochastic trajectories using: (e) the constructed SDE for $\lambda = 0.29$ (blue histogram), $\lambda = 0.285$ (red histogram), and (f) the full ABM at $\lambda = 0.29$.

**Data collection for learning the ML mean-field-level SIR** ABM simulations were performed using $N = 10,000$ agents on an Erdös Rényi distributed network (see Section 7.3). The probability $\lambda$ that susceptible individuals become infected is our bifurcation parameter. For the data set, we selected $51$ equally spaced points in the range $\lambda \in (0, 0.5]$. For each of the values of $\lambda$, we created a grid of $146$ points in the triangle with corners $[(0,0),(0,1),(1,0)]$. The grid points are concentrated close to the edges $[(0,0)-(0,1)]$

and $[(0,0)-(1,0)]$ and also close to the effective unstable steady-states, obtained by the EF approach in [12]. These values correspond to the initial values at time $t=0$ of the densities $[S_0]$ and $[I_0]$ that are constrained by $[S]+[I]\leq 1$. These initial states are then lifted into the agents' space, i.e., randomly picking a fraction of $[S]$ nodes and a fraction of $[I]$ nodes in the network and assigning the corresponding state $S$ and $I$. The remaining nodes are then set as $R$. At each time step $t$ of the ABM, we collect the densities $[S](t)$ and $[I](t)$. Since we are dealing with a stochastic model, for each initial condition, we ran 20 stochastic simulations of the ABM, and we averaged along the generated copies (realizations) of the density variables.

The ABM simulations were performed for a time interval $t\in[0,12]$ with a constant time step $\Delta t=1$ corresponding to 1 day in unit of time. We ignored the first two time points, to avoid the initial "healing" transients. We thus collected a data set consisting 51 (values of $\lambda$) $\times$146 (initial conditions) $\times$10 (time points) $\times$20 (copies) $\approx 1,500,000$ data points. The estimation of the time derivative, for learning the system of ML mean-field-level ODEs, was made with a three-point centered FD stencil.

### 7.3.2 Bifurcation analysis of the ML mean-field SIR surrogates.

To locate the tipping point, we have performed bifurcation analysis using the ML mean-field SIR surrogates. For our illustration, we also compare the derived bifurcation diagram(s) and tipping point(s) with those obtained in [12] using the EF approach and the analytically derived mean-field SIR model given by Eq. (7.35). As shown in Figure 7.11(a), the ML SIRS surrogate approximates adequately the location of the tipping point in parameter space, as well as the entire bifurcation diagram as constructed with the EF approach. Additionally, it outperforms the statistical-mechanics-derived mean-field approximation given by Eq. (7.35). More precisely, the statistical-mechanics-derived mean-field SIRS model predicts the tipping point at $\lambda^*\approx 0.166$ with corresponding steady-state $([S]^*,[I]^*)=(0.138,0.449)$ and the EF at $\lambda^*=0.289$ with corresponding steady-states $([S]^*,[I]^*)=(0.138,0.451)$; our ML mean-field SIR surrogate predictions are at $\lambda^*=0.304$ with corresponding steady-states $([S]^*,[I]^*)=(0.135,0.456)$.

### 7.3.3 Macroscopic physical observables and data-driven observables via DMaps for the epidemic ABM.

Two immediate physically meaningful candidate observables are the densities $[S]$, $[I]$. However, close to the saddle node bifurcation the system is effectively one-dimensional and $[S]$ and $[I]$ are effectively one-to-one with each other in the long term dynamics, eventually taking place on the slow eigenvector of the stable steady state.

We also demonstrate this via the DMaps algorithm. In our case, DMaps applied to the collected data discovers a one-dimensional latent variable parameterized by $\phi_1$ that is also one-to-one with $[I]$ (see Figure 7.11(b)). The local-linear regression algorithm proposed in [146] was also applied to confirm that the remaining eigenvectors are harmonics of $\phi_1$ and thus they do not span independent directions, see Figure 7.11(c). This confirms that the emergent ABM dynamics close to the tipping point lie on one-dimensional manifold.

### 7.3.4 Learning the effective mean-field-level SDE and performing bifurcation analysis for the epidemic ABM.

Here, for our illustration, we learned a one-dimensional parameter–dependent SDE for the DMaps coordinate $\phi_1$.

We used a neural network to identify, a one-dimensional mean-field SDE based on the DMaps coordinate $\phi_1$. The network had 5 hidden layers with 32 neurons per layer. The activation function for the drift network was `tanh`, and for the diffusivity network, it was `softplus`. The data were divided into a 90|10 training|validation split.

Given this trained macroscopic SDE surrogate, the drift term (deterministic component) of the identified dynamics was used to construct the bifurcation diagram with AUTO (see Figure 7.11(d). A saddle-node bifurcation was identified for $\lambda^* = 0.294$ where $\phi_1^\star = 0.006$. The estimated critical parameter value from the SDE is in agreement with our previous work ($\lambda \approx 0.289$ [12]).

**Data collection for learning the ML SDE surrogate targeted in the neighborhood of the tipping point.** We have, in this case, focused our attention on a smaller region in parameter space, targeted to the tipping point, where the dynamics is effectively one-dimensional. In particular, we have selected 21 parameter values in $\lambda \in [0.2, 0.4]$ and for each a fixed grid of 63 initial conditions in the box $([S_0], [I_0]) \in [0.05, 0.3] \times [0.25, 0.7]$. We have also run the ABM for 20 time instants, collecting 5 different stochastic realization (we do not average in this case, as we are interested in learning also the stochastic nature of the dynamics). We ignored the first five time points, to avoid the initial "healing" transients and also to keep only the long-term dynamics that live on the one-dimensional manifold. Therefore, we obtained a data set consisting of 21 (values of $\lambda$) $\times 63$ (initial conditions) $\times 15$ (time points) $\times 5$ (different realizations) $\approx 100,000$ data points. However, for the ML SDE surrogate, the data were randomly subsampled to just $25,000$ data points.

### 7.3.5 Rare-event analysis/UQ of catastrophic shifts via the identified epidemic SDE.

Given the identified steady states at a fixed value of $\lambda$, we performed escape time computations. For $\lambda = 0.29$, we estimated the average escape time needed for a trajectory initiated at the stable steady state to reach $[I] = 0.662$, a value sufficiently above the unstable branch. The corresponding value for the DMaps coordinate was $\phi_1 = -0.007$. We now report a comparison between the escape times of the SDE identified based on the DMaps coordinate $\phi_1$ and those of the epidemic ABM. For this model, we also sampled $10,000$ stochastic trajectories.

In Figure 7.11(e) the histograms of the escape times for the SDE trained on $\phi_1$ are shown for $\lambda = 0.29$ and $\lambda = 0.285$ . For the full epidemic ABM in Figure 7.11(f) we depict the histogram of escape times for $\lambda = 0.29$.

The estimated values for the mean and standard deviation, as computed with temporal simulations from the SDE trained on the Diffusion Map variable $\phi_1$ for $\lambda = 0.29$, are reported in Table 7.4.

Table 7.4: Escape time computations for the epidemic ABM. Means and Standard deviations as computed with temporal simulations from the SDE trained on the Diffusion Map variable $\phi_1$ for $\lambda = 0.29$, $\lambda = 0.285$ and the ABM at $\lambda = 0.29$, respectively.

| Models | SDE at $\lambda = 0.29$ | SDE at $\lambda = 0.285$ | ABM |
|---|---|---|---|
| Mean Escape Time | 348.24 | 1164.65 | 1308.65 |
| Escape Time Standard deviation | 310.16 | 1117.56 | 1247.70 |

As shown, the mean escape time of the full ABM is four times larger than that estimated by the simplified SDE model for $\lambda = 0.29$ (still within an order of magnitude). The SDE model for $\lambda = 0.285$ gives an escape time comparable to the one of the ABM for $\lambda = 0.29$.

As we mentioned earlier, the escape times change exponentially depending on the parameter value. Therefore, a small error in the estimated location of the tipping point can lead to large discrepancies in the estimated escape times.

### 7.3.6 Computational cost for the epidemic ABM.

We compared the computational time required to estimate escape times with the full ABM and the identified mean-field SDE. To compare the computational times, we computed the escape times with the ABM for $\lambda = 0.29$, and that of the SDE for $\lambda = 0.285$, since the mean escape times there are more similar. The estimation of the computational cost for both models (SDE and ABM) was conducted in Matlab. For the 10,000 stochastic trajectories, the total computational time for the identified coarse SDE was $\sim 1$ minute and the one for the full epidemic ABM was $\sim 16$ hours.

## 7.4 Discussion

Designing control policies to prevent catastrophic shifts and performing uncertainty quantification for such events are significant challenges today [28, 131], particularly given their potential to impact areas such as climate change, ecosystem collapse, pandemics, and economic crises. These shifts often occur due to a combination of systematic changes and stochastic perturbations, especially near tipping points, where a system can abruptly transition to a different, potentially catastrophic regime. Tipping points are frequently tied to underlying bifurcations, making it crucial to systematically identify the bifurcation mechanisms governing these shifts and quantify their likelihood [18, 21].

To tackle these challenges, mathematical models and large-scale ABMs offer powerful tools to develop “digital twins" for analysis and prediction, as real-world experiments are often infeasible or unethical. However, the inherent complexity and high dimensionality of such models make the computational task demanding.

In this work, we proposed a machine-learning (ML)-based framework to infer tipping points from high-fidelity spatio-temporal data generated by large-scale simulators. Specifically, we focused on the identification of bifurcation points, their types, and the

probability distributions of catastrophic shift occurrences. Our analysis was demonstrated on three models: (i) a FitzHugh-Nagumo PDE model for action potential propagation in unmyelinated neurons; (ii) a mimetic financial market ABM; and (iii) an epidemic ABM on a complex social network. In particular in the last two, tipping points, such as financial bubbles or epidemic outbreaks, emerged in the dynamics.

Analytical surrogates, while useful, can introduce biases when applied to such complex models, particularly for inverse problems involving IPDEs. Alternatively, manifold learning can identify low-dimensional latent spaces for the emergent dynamics, followed by learning surrogate SDEs in these spaces. This approach reduces computational complexity while maintaining accurate approximations of tipping points. However, it may lead to macroscopic observables that lack direct physical interpretability.

Different system-level tasks benefit from distinct coarse-scale surrogates, highlighting the importance of selecting appropriate ML models for specific objectives. Extensions of this work may involve learning more general stochastic dynamics (e.g., those based on Lévy processes) or moving towards effective SPDEs or fractional evolution operators, as these can yield more informative surrogate models.

While the focus here is not on real-world early warning systems, some collective variables identified through diffusion maps (DMaps) could serve as candidate coordinates for such systems. The primary goal, however, has been to demonstrate how ROMs can be systematically constructed through ML to analyze the emergence of tipping points and quantify the probability of rare events, particularly in systems with high-dimensional ABM simulations.

# 8 Conclusions and future work

This thesis addressed significant advancements in ML numerical analysis-assisted methodologies tailored to address complex systems modeling and analysis, offering both theoretical and practical contributions across several key domains. The integration of Random Projection Neural Networks (RPNNs) and RandONets with classical numerical analysis frameworks has led to innovative solutions for both forward and inverse problems, significantly improving our ability to model, simulate, and predict the behavior of complex systems, thus relaxing the curse of dimensionality. The exploration of RPNNs and RandOnets has shown their potential as efficient, scalable alternatives to traditional machine (deep) learning methods for function and operator approximation. By demonstrating exponential convergence and computational efficiency, RPNNs have been positioned as a strong contender in solving both forward and inverse problems, especially in terms of computational speed and accuracy, with particular advantages in scenarios requiring real-time or large-scale simulations. Their success in approximating nonlinear operators, such as those found in ODEs and PDEs, suggests broad applicability in complex systems, where high-dimensional and computationally expensive problems are prevalent.

Regarding, the forward problems, particularly for nonlinear stationary PDEs and stiff ODEs/DAEs, RPNNs have been effectively employed to develop new numerical methods that outperform traditional schemes like Finite Difference (FD) and Finite Element Methods (FEM), as well as professional code built-in in `MATLAB` as `ode15s`. These results underscore the method's potential in handling complex phenomena such as timescale separations in stiff systems as well as bifurcations and tipping points, which are crucial in understanding the dynamics of complex systems. The future integration of techniques like slow inertial manifolds and matrix-free methods promises to extend the applicability of these schemes to even larger and more complex systems.

The work on inverse problems for the identification of coarse-scale parametrized PDEs and SDEs, as well as their bifurcation analysis and the study of the associated rare events, brings ML into the domain of uncertainty quantification and the prediction of catastrophic events in complex systems. Through machine-learning-based frameworks, tipping points have been identified in high-dimensional ABMs, providing insights into the mechanisms behind phenomena like financial bubbles or epidemic outbreaks.

The thesis demonstrates how manifold learning techniques can identify low- dimen-

sional latent spaces in which surrogate models, such as SDEs, can be developed. This approach enables the analysis of emergent behavior and tipping points while reducing computational cost. However, it also introduces challenges, particularly in ensuring the physical interpretability of macroscopic observables.

**Broader Implications for Complex Systems.** The advancements and findings of this Thesis have profound implications for modeling, simulation, and analysis of the behavior of complex systems. In particular, the advancements in RPNNs and RandONets offer a pathway to more efficient and scalable models that can handle the inherent complexity of real-world systems. These methods provide robust alternatives to deep learning architectures, offering better computational efficiency and interpretability, which are critical for scientific applications.

One of the key takeaways from this work is the importance of integrating ML with tailor-made numerical analysis methods. This hybrid approach allows for the development of interpretable SciML schemes that are not only efficient but also grounded in established theoretical frameworks.

## 8.1 Future Directions

While this Thesis has made significant strides in addressing some key challenges in complex systems modeling, several open problems remain. Future research could focus on:

- Extending RPNNs and RandONets to handle high-dimensional problems and time-dependent PDEs.
- Developing improved basis function selection strategies and regularization techniques for RPNNs.
- Exploring the use of multistep methods and slow inertial manifolds to enhance the performance of machine-learning-based ODE and DAE solvers.
- Refining surrogate models for rare event analysis and tipping point detection, particularly in systems governed by stochastic dynamics.
- Further integrating ML with state-of-the-art numerical methods, such as Krylov-subspace techniques, to tackle large-scale simulations in complex systems.

By addressing these challenges, the methods developed in this thesis could be extended to even more complex scenarios, providing new tools for scientists and engineers to better understand and predict emergent behaviors in systems ranging from climate dynamics to financial markets. Moreover, the insights gained from this work are not only applicable to the specific case studies presented, but also offer a broader framework for tackling a wide range of scientific and engineering challenges.

## 8.2 List of publications

Here are listed the publication of Gianluca Fabiani, under the affiliation of the SSM, while enrolled in PhD program in MERC:

- **G. Fabiani**, F. Calabrò, L. Russo, C. Siettos, "Numerical solution and bifurcation analysis of nonlinear partial differential equations with extreme learning machines", Journal of Scientific Computing, 89, 1-35. 2021.
- F. Calabrò, **G. Fabiani**, C. Siettos, "Extreme learning machine collocation for the numerical solution of elliptic PDEs with sharp gradients". Computer Methods in Applied Mechanics and Engineering, 387, 114188, 2021.
- E. Galaris, **G. Fabiani**, I. Gallos, I.G. Kevrekidis, 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
- **G. Fabiani**, E. Galaris, L. Russo, C. Siettos, "Parsimonious Random Projection Neural Networks for the Numerical Solution of Initial-Value Problems of ODEs and index-1 DAEs", Chaos 33, 043128, 2023.
- H. Vargas Alvarez, **G. Fabiani**, N. Kazantzis, C. Siettos, I. G. Kevrekidis, "Discrete-Time Nonlinear Feedback Linearization via Physics-Informed Machine Learning", Journal of Computational Physics, 492, 112408, 2023
- **G. Fabiani**, N. Evangelou, T. Cui, J. M. Bello-Rivas, C. P. Martin-Linares, C. Siettos, I. G. Kevrekidis, "Task-Oriented Machine Learning Assisted Surrogates for Tipping Points of agent-based models", Nature Communications 15, n. 4117, pp. 1-13, 2024
- D. G. Patsatzis, **G. Fabiani**, L. Russo, and C. Siettos. "Slow invariant manifolds of singularly perturbed systems via physics-informed machine learning." SIAM Journal of Scientific Computing 46 (4), C297-C322, 2024
- H. Vargas Alvarez, **G. Fabiani**, I. G. Kevrekidis, N. Kazantzis, C. Siettos, "Nonlinear Discrete-Time Observers with Physics-Informed Neural Networks", Chaos Solitons and Fractals 186, 2024
- A. Gnanadesikan, **G. Fabiani**, J. Liu, R. Gelderloos, G.J. Brett, Y. Kevrekidis, T. Haine, M. Pradal, C. Siettos, J. Sleeman, "Tipping points in overturning circulation mediated by ocean mixing and the configuration and magnitude of the hydrological cycle: A simple model", Journal of Physical Oceanography, 2024
- F. Auricchio, M. R. Belardo, F. Calabrò, **G. Fabiani**, A. F. Pascaner, "On the accuracy of interpolation based on single-layer artificial neural networks with a focus on defeating the Runge phenomenon", Soft Computing, 2024

- **G. Fabiani**, I. G. Kevrekidis, C. Siettos, A. N. Yannacopoulos, "RandONets: Shallow-Networks with Random Projections for learning linear and nonlinear operators", Journal of Computational Physics, 520, 113433,2025.

Preprints under Review:

- **G. Fabiani**, "Random Projection Neural Networks of Best Approximation: Convergence theory and practical applications", arXiv preprint arXiv:2402.11397, 2024
- **G. Fabiani**, E. Bollt, C. Siettos, A. N. Yannacopoulos, "Stability Analysis of Physics-Informed Neural Networks for Stiff Linear Differential Equations", arXiv preprint arXiv:2408.15393, 2024

Appendix A

# Preliminaries of Functional Analysis and measure theory for Data Mining

In data analysis, measuring the difference, similarity, or “closeness" between data points is fundamental to many algorithms and models. These concepts are formalized through mathematical structures such as norms, distances, and metrics, which are crucial for evaluating and quantifying differences in high-dimensional spaces – particularly in the context of complex systems. In this appendix chapter, we review key concepts from functional analysis, measure theory, and $L^p$-spaces.

## A.1 Topologies and metrics

### A.1.1 Topology

Next, we define a *topology* on a set $X$, which provides a framework for discussing and exploring concepts such as convergence, continuity, compactness, and connectedness.

**Definition A.1.1.** A topology $\mathcal{T}$ on $X$ is a collection of subsets of $X$ (called *open sets*) satisfying: (i) $X \in \mathcal{T}$ and $\emptyset \in \mathcal{T}$; (ii) Any union of sets in $\mathcal{T}$ is in $\mathcal{T}$; (iii) Any finite intersection of sets in $\mathcal{T}$ is in $\mathcal{T}$.

Within a topological space, one can define continuity and compactness, both of which are important in many problems in analysis and optimization.

**Definition A.1.2.** A function $f : X \to Y$ between topological spaces $(X, \mathcal{T}_X)$ and $(Y, \mathcal{T}_Y)$ is said to be *continuous* if, for every open set $V \in \mathcal{T}_Y$, the preimage $f^{-1}(V) \in \mathcal{T}_X$.

**Definition A.1.3.** A subset $K \subset X$ of a topological space $X$ is said to be *compact* if every open cover of $K$ has a finite *subcover*. That is, if $K \subset \bigcup_{\alpha \in A} U_\alpha$ where each $U_\alpha$ is open, then there exist finitely many indices $\alpha_1, \alpha_2, \ldots, \alpha_n \in A$ such that

$$K \subset \bigcup_{i=1}^{n} U_{\alpha_i}. \tag{A.1}$$

Connectedness in topology refers to the idea that a topological space cannot be split into two disjoint non-empty open sets. Here's a formal definition:

**Definition A.1.4.** A topological space $X$ is said to be *connected* if there do not exist two open sets $U$ and $V$ in $\mathcal{T}$ such that: (i) $X = U \cup V$; (ii) $U \cap V = \emptyset$; and (iii) $U \neq \emptyset$ and $V \neq \emptyset$.

### A.1.2 Distance Functions and Metric Spaces

In ML, data mining, and complex systems, a distance or metric in vector space quantifies the similarity or dissimilarity between data points, guiding clustering, classification, and pattern recognition tasks.

**Definition A.1.5.** A distance function (or metric) is a function $d : V \times V \to [0, \infty)$ that satisfies the following for all $x, y, z \in V$: (i) $d(x, y) \geq 0$, with $d(x, y) = 0 \iff x = y$; (ii) $d(x, y) = d(y, x)$ (symmetry); (iii) $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality).

**Definition A.1.6.** A metric space $(V, d)$ is a set $V$ together with a distance function $d$. The topology induced by a metric is defined by the open sets (balls) $B(x, r) = \{y \in V : d(x, y) < r\}$, where $x \in V$ and $r > 0$.

A crucial concept that can be defined within a metric space is that of convergence:

**Definition A.1.7.** Let $(X, d)$ be a metric space with metric $d$, and let $\{x_n\}$ be a sequence of points in $X$. The sequence $\{x_n\}$ is said to *converge* to a point $x \in X$ if for every $\epsilon > 0$, there exists an integer $N$ such that for all $n \geq N$, $d(x_n, x) < \epsilon$. In this case, we write $x_n \to x$ or $\lim_{n\to\infty} x_n = x$.

This definition generalizes the concept of convergence in $\mathbb{R}^n$, where $d(x_n, x)$ would typically represent the Euclidean distance between $x_n$ and $x$.

Here, we provide examples of commonly used distances in data analysis. For two finite-dimensional vectors $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, $\mathbf{y} = (y_1, y_2, \ldots, y_n) \in \mathbb{R}^n$, the Minkowski distance is defined as:

$$d_p(\boldsymbol{x}, \boldsymbol{y}) = \left( \sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}, \quad 1 \leq p < \infty. \tag{A.2}$$

In particular, we have: (i) for $p = 1$ the Manhattan distance; (ii) for $p = 2$ the Euclidean distance; and (iii) for $p = \infty$ the Maximum distance, $d_\infty(\mathbf{x}, \mathbf{y}) = \max_i |x_i - y_i|$.

The Euclidean distance is particularly significant because it arises from the scalar product $\langle \cdot, \cdot \rangle$.

Indeed, the Euclidean norm of a vector $\mathbf{x}$ can be defined as $\|\mathbf{x}\|_2 = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$. Then, the Euclidean distance $d_2(\mathbf{x}, \mathbf{y})$ between two vectors $\mathbf{x}$ and $\mathbf{y}$ is then defined as the norm of their difference: $d_2(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2$ .

Moreover, the scalar product allows us to derive important geometric concepts such as the angle $\theta$ between two vectors, defined through the relationship: $\cos(\theta) = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\| \|\mathbf{y}\|}$,.

### A.1.3 Hausdorff spaces

Hausdorff spaces guarantee that points can be “separated" by neighborhoods, which is essential for discussing limits and uniqueness of limits in data analysis. More formally,

**Definition A.1.8.** A topological space $(X, \mathcal{T})$ is called a *Hausdorff space* if for any two distinct points $x, y \in X$, there exist disjoint open sets $U, V \in \mathcal{T}$ such that $x \in U$ and $y \in V$. Formally:

$$\forall x, y \in X,\ x \neq y \implies \exists U, V \in \mathcal{T},\ x \in U,\ y \in V,\ U \cap V = \emptyset. \tag{A.3}$$

Every metric space, with topology induced by the metric, is a Hausdorff space. In a metric space, the distance function allows us to separate any two distinct points by open balls of sufficiently small radii.

The separability of the space is fundamental for the following properties of Hausdorff spaces:

1. Limits of sequences are unique. Specifically, if $x_n \to x$ and $x_n \to y$, then $x = y$.
2. A set $A \subset X$ is closed if and only if, for every sequence $\{x_n\} \subset A$ that converges to some $x \in X$, we have $x \in A$. This means that convergence sequences stay within closed sets.
3. Compact sets are closed. That is, if $K \subset X$ is compact, then $K$ is also a closed subset.

### A.1.4 Norms and Banach Spaces

A *norm* on a vector space $V$ is a function that assigns a non-negative scalar to each vector, representing its “length" or “magnitude". Formally, we provide the following definition:

**Definition A.1.9** (Norms)**.** A *norm* on a $\mathbb{C}$-vector space $V$ is a function, denoted by $\|\cdot\| : V \to [0, \infty[$, that satisfies the following properties for all $x, y \in V$ and $\alpha \in \mathbb{C}$:
1. *Non-negativity*: $\|x\| \geq 0$ with $\|x\| = 0 \iff x = 0$ 2. *Homogeneity*: $\|\alpha x\| = |\alpha| \|x\|$ 3. *Triangle inequality*: $\|x + y\| \leq \|x\| + \|y\|$

The pair $(V, \|\cdot\|)$ is said a normed vector space. We can define, based on the norm, the function $d$ induced by the norm, $d : V \times V \to [0, \infty[$. The property of the norm show easily that $d$ is a distance function and that $V$ is also a metric space. If the space $V$ is also embedded with the topology induced by the metric $d$, then it is also a Hausdorff space with countable basis. Specifically, given a sequence of vector $x_n \in V$ converging

to $x$, i.e., $(x_n \to x)$, we have that $(x_n \to x) \iff \|x_n - x\| \to 0$. Also, easily from triangular inequality we get that $\|x-y\| \geq \|x\| - \|y\|$ and thus the following proposition holds:

**Proposition A.1.1.** $\|\cdot\|$ is continuous.

*Proof.* Let $x_n \to x$, then $|\|x_n\| - \|x\|| \leq \|x_n - x\| \to 0$. Hence, $\|x_n\| \to \|x\|$. □

Now, let $(V, d)$ be a metric space, and let $x_n$ be a sequence in $V$. The sequence $x_n$ is called a *Cauchy sequence* if and only if:

$$\forall \epsilon > 0, \exists N \in \mathbb{N}, \forall n, m > N, \quad d(x_n, x_m) < \epsilon. \tag{A.4}$$

It is straightforward to show that convergent sequences are Cauchy, and Cauchy sequences are bounded. A metric space $(V, d)$ is said to be *complete* if every Cauchy sequence converges. A *complete normed space* $(V, \|\cdot\|)$ is called a *Banach space*.

## A.2 Measurable Spaces and Positive Measures

In data analysis and ML, the concept of a measure plays a crucial role in defining probabilities, volumes, and densities over spaces. A measure $\mu$ is a function that assigns a non-negative value (often interpreted as “size" or “weight") to subsets of a given set, formalized within the framework of measure theory.

Let us introduce some definitions.

**Definition A.2.1.** A $\sigma$-algebra $\mathcal{A}$ on a set $X$ is a collection of subsets of $X$ that is closed under complements and countable unions. That is, $\mathcal{A}$ satisfies: (i) $X \in \mathcal{A}$; (ii) if $A \in \mathcal{A}$, then $X \setminus A \in \mathcal{A}$; and (iii) if $\{A_n\}_{n=1}^{\infty} \subset \mathcal{A}$, then $\bigcup_{n=1}^{\infty} A_n \in \mathcal{A}$.

As an example, the *Borel* $\sigma$-algebra is the $\sigma$-algebra generated by the open sets of a topological space, i.e., all sets that can be formed from them using countable unions and intersections.

A pair $(X, \mathcal{A})$ is called a measurable space, where $X$ is a set and $\mathcal{A}$ is a $\sigma$-algebra on $X$.

**Definition A.2.2.** A measure $\mu$ on $(X, \mathcal{A})$ is a function $\mu : \mathcal{A} \to [0, \infty]$ that satisfies: (i) $\mu(\emptyset) = 0$; (ii) $\mu$ is countably additive: if $\{A_n\}_{n=1}^{\infty} \subset \mathcal{A}$ is a disjoint collection, then

$$\mu\left(\bigcup_{n=1}^{\infty} A_n\right) = \sum_{n=1}^{\infty} \mu(A_n); \tag{A.5}$$

A positive measure is a measure where $\mu(A) \geq 0$ for all $A \in \mathcal{A}$. Also, since $\mu \equiv +\infty$ is a measure, we explicitly require that for a positive measure, there exist an $A \in \mathcal{A}$ such that $\mu(A) < +\infty$. Basic simple properties of positive measures are monotonicity and sub-additivity.

**Definition A.2.3.** A function $f : X \to \mathbb{R}$ is called *measurable* with respect to the measure $\mu$ if for every Borel set $B \subset \mathbb{R}$, the pre-image $f^{-1}(B) \in \mathcal{A}$, where $\mathcal{A}$ is the sigma-algebra associated with $\mu$. In simpler terms, a function is measurable if the sets where it takes particular values can be measured using the measure $\mu$.

In the following, we introduce different types of measures, in order to present the fundamental intuitions and examples that can be associated to the formal definition:

### A.2.1 Some example of Measure

**Probability measure.** A probability measure is a measure $\mu$ defined on a measurable space $(X, \Sigma)$ where $\mu(X) = 1$. This corresponds to the probability of the entire space being 1, meaning that the total probability of all events in $X$ sums up to 1. Probability measures are used to describe the distribution of random variables over a feature space $X$. Usually $\mu$ is defined with the notation $\mathbb{P}$.

**Dirac measure.** The Dirac measure $\delta_x$ is the simplest example of a measure, concentrated entirely at a single point $x \in X$. It is defined as:

$$\delta_x(S) = \begin{cases} 1 & \text{if } x \in S, \\ 0 & \text{otherwise.} \end{cases} \tag{A.6}$$

This measure assigns a weight of 1 to any set $S$ that contains $x$, and 0 otherwise. From a probabilistic viewpoint, the Dirac measure represents a deterministic scenario: if $X \sim \delta_x$, then $X = x$ almost surely, meaning $\mathbb{P}(X = x) = 1$.

**Lebesgue Measure.** The Lebesgue measure $\lambda_{\mathbb{R}^D}$ is a fundamental example in the theory of integration, defining a way to measure "volume" in $\mathbb{R}^D$. For a measurable set $S \subset \mathbb{R}^D$, the Lebesgue measure is intuitively described as:

$$\lambda_{\mathbb{R}^D}(S) = \int_S dx. \tag{A.7}$$

This measure generalizes the concepts of length, area, and volume in one, two, and higher dimensions, respectively. The Lebesgue measure is defined rigorously using the Borel $\sigma$-algebra $\mathcal{B}(\mathbb{R}^D)$, which consists of all Borel measurable sets in $\mathbb{R}^D$.

In probabilistic terms, a uniform distribution over the interval $[0, 1]$ is an example of a random variable $X$ that follows the law of the Lebesgue measure restricted to the unit interval:

$$\lambda_{[0,1]}(S) = \int_{S \cap [0,1]} dx, \quad S \subset \mathbb{R}. \tag{A.8}$$

**Gaussian measure.** The Gaussian measure is one of the most important probability distributions in statistics and data analysis. In $\mathbb{R}^d$, the Gaussian measure is defined by its density function:

$$N_{\mathbb{R}^d}(S) = \frac{1}{(2\pi)^{D/2}} \int_S \exp\left(-\frac{\|x\|^2}{2}\right) dx, \tag{A.9}$$

where $\|x\|$ denotes the Euclidean norm in $\mathbb{R}^d$.

The Gaussian distribution is a cornerstone in probability theory, particularly because of its appearance in the Central Limit Theorem, which states that under certain conditions, the sum of independent random variables converges to a Gaussian distribution.

### A.2.2 Random Variables

In the context of complex systems and ML, a random variable $X$ is a mathematical tool that models uncertainty and variability within a system. Random variables allow us to quantify and analyze randomness in processes where deterministic modeling is infeasible or incomplete.

Formally, in a *probability space* $(\Omega, \mathcal{F}, \mathbb{P})$, where $\mathcal{F}$ is a $\sigma$-algebra over $\Omega$, a random variable is a measurable function $X : \Omega \to \mathbb{R}$. $\Omega$ represents the set of all possible system states, and $X$ associate to each element/state $x \in \Omega$ a real value. For example, in a complex system like a climate network or a social network, $X$ could represent variables such as temperature fluctuations or the number of connections between individuals, each influenced by randomness and external factors.

There are two main types of random variables: (a) discrete random variables, which take on a countable set of values, and, (b) continuous random variables, which take values in an uncountable set, typically an interval in $\mathbb{R}$. The *distribution* of a random variable $X$ describes how the probability is spread over the possible values that $X$ can take. The distribution can be characterized in several ways, depending on whether the random variable is discrete or continuous.

If $X$ is a discrete random variable, the distribution is described by the probability mass function (pmf) $p_X(x)$. The pmf gives the probability that $X$ takes a specific value $x$:

$$p_X(x) = \mathbb{P}(X = x), \quad x \in \Omega.$$

If $X$ is a continuous random variable, its distribution is described by the probability density function (pdf) $f_X(x)$. The pdf gives the relative likelihood that $X$ is near a specific value $x$. However, the probability that $X$ takes any exact value is 0; instead, probabilities are computed over intervals:

$$\mathbb{P}(a \leq X \leq b) = \int_a^b f_X(x)\, dx.$$

The pdf satisfies the following properties:
(i) $f_X(x) \geq 0$ for all $x$, and,
(ii) $\int_{-\infty}^{\infty} f_X(x)\, dx = 1$.

### A.2.3 Lebesgue integral

The Lebesgue integral is a generalization of the Riemann integral, particularly useful for integrating functions that may be difficult or impossible to handle with the Riemann approach, especially in higher dimensions or for more complex sets. Unlike the Riemann integral, which partitions the domain of the function, the Lebesgue integral partitions the *codomain* (the range of the function values). This allows it to extend the notion of the integral to functions that are not necessarily continuous. By measuring the "size" of the sets where the function takes particular values, the Lebesgue integral provides a more flexible framework for integrating a broader class of functions, including those with discontinuities.

Let $(X, \mathcal{A}, \mu)$ be a measure space, where $\mu$ is a measure on the measurable space $X$ with $\sigma$-algebra $\mathcal{A}$. For a non-negative measurable function $f : X \to [0, \infty]$, the Lebesgue integral of $f$ with respect to $\mu$ is defined as the supremum of integrals over simple functions $\phi$, i.e., functions that can be written as a finite linear combination of indicator functions $\phi = \sum_i \alpha_i \chi_{Ai}$, that approximate $f$ from below:

$$\int_X f \, d\mu = \sup \left\{ \int_X \phi \, d\mu : 0 \leq \phi \leq f, \, \phi \text{ simple} \right\}. \tag{A.10}$$

If $f$ is integrable, this generalizes naturally to real-valued functions by decomposing them into positive and negative parts, $f = f^+ - f^-$.

### A.2.4 $L^p$-Spaces

Given a measurable space $(X, \mathcal{A}, \mu)$ and a real-valued function $f : X \to \mathbb{R}$, we define the $L^p$-norm for $1 \leq p < \infty$ as:

$$\|f\|_p = \left( \int_X |f(x)|^p \, d\mu(x) \right)^{1/p}. \tag{A.11}$$

The space of $\mu$-measurable functions for which $\|f\|_p < \infty$ is called the $L^p$-space:

$$L^p(X, \mu) = \{ f : X \to \mathbb{R} \mid \|f\|_p < \infty \}. \tag{A.12}$$

For $p = \infty$, the corresponding norm is defined as:

$$\|f\|_\infty = \operatorname{ess\,sup}_{x \in X} |f(x)|, \tag{A.13}$$

where $\operatorname{ess\,sup}$ refers to the essential supremum, which is the smallest number $M$ such that $|f(x)| \leq M$ almost everywhere with respect to $\mu$. The space of functions with finite $\|f\|_\infty$ is denoted $L^\infty(X, \mu)$.

The $L^p$-spaces are Banach spaces for all $1 \leq p \leq \infty$, meaning they are complete with respect to the $L^p$-norm. That is, every Cauchy sequence of functions in $L^p$ converges to a function in $L^p$.

A fundamental inequality in $L^p$-spaces is Hölder's inequality, which states that for any $f \in L^p(X, \mu)$ and $g \in L^q(X, \mu)$, where $1/p + 1/q = 1$, the following inequality holds:

$$\int_X |f(x)g(x)|\, d\mu(x) \leq \|f\|_p \|g\|_q. \tag{A.14}$$

$L^p$-spaces are fundamental in many areas of analysis and applied mathematics, including solving forward and inverse problems in complex systems. In data analysis, they provide a framework for studying the behavior of functions with respect to various norms, ensuring robust convergence properties. $L^p$-spaces are also essential in numerical analysis, where approximations and solutions are often studied in terms of their behavior in these spaces. Moreover, in ML, these spaces offer a rigorous setting for analyzing function spaces and error metrics, particularly in tasks such as regression, signal processing, and functional approximation.

## A.3 Fourier Transforms and Schwartz Spaces in Functional Analysis

In the context of functional analysis, the understanding of function spaces is critical for advancing theoretical frameworks and applications in data analysis. Hence, for convenience, we include some definition and concept that will be used later.

We first define the space of test functions $\mathcal{D}(\mathbb{R}^d)$, which consists of infinitely differentiable functions with compact support. This means that each function $\phi \in \mathcal{D}(\mathbb{R}^d)$ vanishes outside a bounded region in $\mathbb{R}^d$, ensuring that it can be integrated over $\mathbb{R}^d$ without concerns regarding divergence.

The dual space, denoted $\mathcal{D}'(\mathbb{R}^d)$, contains distributions, which are continuous linear functionals that act on test functions. Distributions generalize classical functions and allow us to manipulate and analyze objects that may not be well-defined in the classical sense, such as Dirac's delta function.

A particularly important subclass of functions is the Schwartz space $\mathcal{S}(\mathbb{R}^d)$, which comprises rapidly decreasing functions. A function $g \in \mathcal{S}(\mathbb{R}^d)$ satisfies the condition that, for all multi-indices $\alpha$ and $\beta$, the derivatives $x^\alpha \partial^\beta g(x)$ decay faster than any polynomial as $|x| \to \infty$. The dual of this space, $\mathcal{S}'(\mathbb{R}^d)$, consists of tempered distributions, which provide a suitable framework for Fourier analysis, particularly when dealing with non-analytic functions.

The Fourier transform is a pivotal operator in this setting, transforming functions between the spatial and frequency domains. For $f \in L^1(\mathbb{R})$, the Fourier transform of $f$ is defined as:

$$\widehat{f}(t) = F(f)(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} f(x) e^{-ixt}\, dx \quad \forall t \in \mathbb{R}. \tag{A.15}$$

We can introduce the measure $dm(x) = \frac{1}{\sqrt{2\pi}} dx$ to express this as:

$$F(f)(t) = \int_{-\infty}^{+\infty} f(x) e^{-ixt}\, dm(x). \tag{A.16}$$

This definition can be extended to Schwarz space. For $g \in \mathcal{S}(\mathbb{R}^d)$, the Fourier transform is defined as:

$$\widehat{g}(\xi) = \int_{\mathbb{R}^d} e^{-2\pi i \xi \cdot x} g(x)\, dx. \tag{A.17}$$

This transformation not only provides insights into the frequency components of $g$ but also preserves the smoothness and decay properties inherent to functions in the Schwartz space.

It is immediately evident that $F(f)(t)$ is well-defined and continuous. First, note that:

$$\int_{-\infty}^{+\infty} |f(x)e^{-ixt}|\, dx = \int_{-\infty}^{+\infty} |f(x)|\, dx < +\infty, \tag{A.18}$$

because $f \in L^1(\mathbb{R})$, thus $F(f)$ is well-defined. Additionally, it can be shown that we can dominate $|f(x)e^{-ixt}|$ with $g(x) = |f(x)|$, which is an integrable function. Although $f$ may not be continuous, the parameter $t$ is only part of the continuous $e^{-ixt}$, therefore $F(f)(t)$ is continuous. Furthermore, we also have:

$$|F(f)(t)| \leq \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} |f(x)e^{-ixt}|\, dx = \frac{1}{\sqrt{2\pi}} \|f\|_{L^1(\mathbb{R})}. \tag{A.19}$$

Thus, we can define the operator $F : L^1(\mathbb{R}) \to C(\mathbb{R})$ as the Fourier transform. It is trivial to observe that $F$ is linear.

Both the Fourier transform $\mathcal{F}$ and its inverse $\mathcal{F}^{-1}$ can be extended to apply to tempered distributions, allowing for the analysis of a wider class of functions, including those that arise in the context of partial differential equations and signal processing. The formulation of the Fourier transform for tempered distributions is given by:

$$\mathcal{F}(T)(\xi) = \widehat{T}(\xi) = \int_{\mathbb{R}^d} e^{-2\pi i \xi \cdot x} T(x)\, dx, \tag{A.20}$$

for suitable test functions $T$ in $\mathcal{S}'(\mathbb{R}^d)$.

We refer to functions as slowly increasing if they are tempered distributions, which allows us to work within a framework where the Fourier transform is well-defined and can provide meaningful results. The interplay between the properties of Schwartz spaces, tempered distributions, and the Fourier transform is fundamental in various applications, including data analysis, where frequency domain techniques can be employed for filtering, reconstruction, and the analysis of time-series data.

### A.3.1 Properties of the Fourier Transform

We now present some fundamental properties of the Fourier transform that can simplify the calculation of certain transforms.

Let $f \in L^1(\mathbb{R})$. We list several properties of the Fourier transform:

**Proposition A.3.1.** (i) $F(e^{i\theta} f(x))(t) = \widehat{f}(\theta t) \quad \forall \theta \in \mathbb{R}$

(ii) $F(f_y(x))(t) = \widehat{f}(t)e^{-iyt} \quad \forall y \in \mathbb{R}$

(iii) $F(f(\lambda x))(t) = \frac{1}{|\lambda|}\widehat{f}\left(\frac{t}{\lambda}\right) \quad \forall \lambda \neq 0$

Finally, if $f$ is even, then $\widehat{f}$ is also even. If $f$ is real and even, then $\widehat{f}$ is real and even. If $f$ is odd, then $\widehat{f}$ is also odd. If $f$ is real and odd, then $\widehat{f}$ is odd and purely imaginary.

## Appendix B

# An introduction to Manifolds, differential equations, bifurcation theory and multiscale modeling

This appendix chapter introduces key mathematical and theoretical tools necessary for analyzing the emergent behavior in complex systems. To model, analyze, and predict such emergent behavior, we will explore both forward and inverse problems in the context of dynamical systems, which typically involve ODEs, PDEs, and SDEs.

We begin by introducing the foundational concepts of dynamical systems and manifolds, which form the mathematical setting for many complex systems. Manifolds, equipped with local Euclidean structures, are essential for understanding systems with high-dimensional, nonlinear behavior, where Diffusion Maps and the Laplace-Beltrami operator can be used to approximate the underlying geometry. Additionally, we delve into ODE and PDE theory, which provide the governing equations for many real-world systems, and discuss boundary conditions, weak solutions, and stochastic processes, which are critical for modeling uncertainty and noise in these systems.

Bifurcation theory, which studies sudden qualitative changes in system dynamics, will be introduced in the context of ODEs and extended to PDEs. We also cover continuation theory to address how solutions evolve as parameters vary. Operator theory will also be discussed briefly.

This chapter sets the mathematical groundwork for solving both forward problems (predicting system behavior) and inverse problems (inferring underlying system properties from observed data) in complex systems.

## B.1 Manifolds and differential equations

A manifold is a topological space that is locally similar to a Euclidean space. Informally, a smooth $d$-dimensional manifold is a space that looks flat in small regions, resembling $\mathbb{R}^d$. On such spaces, we can define velocities or directions at each point due to the smoothness of the manifold, allowing the traversal of the surface by following these directions. This forms the foundation for defining dynamical systems, which describe how points evolve on manifolds over time.

To formalize this, we first need to define continuous deformations between spaces, known as *homeomorphisms*. A homeomorphism is a bijective and continuous function between topological spaces that has a continuous inverse function.

**Definition B.1.1.** A homeomorphism $\phi : A \to B$ between two topological spaces $A$ and $B$ is a continuous, bijective function such that its inverse $\phi^{-1}$ is also continuous.

Intuitively, homeomorphisms can be understood as continuous transformations (stretching and bending) of space without tearing. Equipped with the concept of homeomorphisms, we can now define a manifold.

**Definition B.1.2.** A $d$-dimensional *manifold* $\mathcal{M}$ is a topological Hausdorff space such that every point has a neighborhood homeomorphic to an open subset of $\mathbb{R}^d$. A coordinate *chart* $(U, \phi)$ is an open set $U \subset \mathcal{M}$ along with a homeomorphism $\phi : U \to V$, where $V \subset \mathbb{R}^d$. An *atlas* $A$ is a collection of charts

$$A = \{(U_\alpha, \phi_\alpha)\}_{\alpha \in I}$$

where $I$ is an index set, and $\mathcal{M} = \bigcup_{\alpha \in I} U_\alpha$. If, for all $\alpha, \beta \in I$, the transition map

$$\phi_\alpha \circ \phi_\beta^{-1} : \phi_\beta(U_\alpha \cap U_\beta) \to \mathbb{R}^d$$

is continuously differentiable any number of times, the atlas is called a smooth atlas or $C^1$-atlas. A smooth manifold is a manifold with a smooth atlas.

Next, we define a *diffeomorphism*, which extends homeomorphisms by requiring differentiability.

**Definition B.1.3.** Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two smooth manifolds. A $C^k$-diffeomorphism $\psi : \mathcal{M}_1 \to \mathcal{M}_2$ is a homeomorphism such that both $\psi$ and $\psi^{-1}$ are $k$-times continuously differentiable. In this case, $\mathcal{M}_1$ and $\mathcal{M}_2$ are said to be diffeomorphic.

While it is often desirable to represent a manifold by a global coordinate chart, many manifolds cannot be globally represented in this way. A classic example is the 2-sphere, which cannot be mapped onto $\mathbb{R}^2$ without singularities. However, the concept of embedding allows us to represent a manifold in a higher-dimensional Euclidean space.

**Definition B.1.4.** Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be smooth manifolds, and let $H : \mathcal{M}_1 \to \mathcal{M}_2$ be a smooth map. The Jacobian $dH_p$ of $H$ at a point $p \in \mathcal{M}_1$ defines a linear map between the tangent spaces $T_p\mathcal{M}_1$ and $T_{H(p)}\mathcal{M}_2$. If $H$ is injective and $\operatorname{rank}(dH_p) = \dim(\mathcal{M}_1)$ for all $p \in \mathcal{M}_1$, then $H$ is called an immersion. If $H$ is a homeomorphism onto its image, then $H$ is called an *embedding* of $\mathcal{M}_1$ into $\mathcal{M}_2$.

The tangent space provides a local linear approximation to a manifold at any point.

**Definition B.1.5.** Let $\mathcal{M}$ be a $d$-dimensional smooth manifold, and let $p \in \mathcal{M}$. The *tangent space* $T_p\mathcal{M}$ at $p$ is the vector space of all possible directions in which one can tangentially pass through $p$. Formally, it is the set of equivalence classes of curves passing through $p$, where two curves are equivalent if their derivatives at $p$ are the same.

The concept of embedding is particularly important in the Takens/Whitney Embedding Theorem, which guarantees that any $d$-dimensional smooth manifold can be embedded in $\mathbb{R}^{2d}$. This embedding allows us to study manifolds using the familiar tools of Euclidean space.

With these definitions in place, we can now introduce dynamical systems, which describe the evolution of points on a manifold over time.

**Definition B.1.6.** A *dynamical system* is a manifold $\mathcal{M}$, called the *state space*, together with a diffeomorphism $\phi : T \times \mathcal{M} \to \mathcal{M}$, where $T$ is the *time*. If $T = \mathbb{R}$, the system is called continuous, and $\phi$ is called a *flow*. If $T = \mathbb{N}_0$, the system is called *discrete*, and $\phi$ is called a discrete-time map. The flow $\phi_t(x)$ represents the state of the system at time $t$, starting from $x \in M$.

For continuous dynamical systems, differential equations govern the time evolution. Consider a smooth vector field $f$ on $\mathcal{M}$. The flow $\phi_t$ of the system satisfies the differential equation:

$$\frac{d}{dt}\phi_t(x) = f(\phi_t(x)), \quad \phi_0(x) = x.$$

This is an ODE that describes how the state $x$ evolves over time. Writing $x(t) := \phi_t(x)$, the ODE becomes:

$$\frac{dx}{dt} = f(x),$$

where $f(x)$ is the vector field describing the velocity at each point. This formalism lays the groundwork for describing both ODEs and PDEs in dynamical systems. The set $\{x(t) \mid t \in \mathbb{R}^+, x(0) = x_0 \in \mathcal{M}\}$ is referred to as the *trajectory* of the dynamical system, starting at $x(0) = x_0$. If the function $f$ in the differential equation depends only on $x(t) = \phi_t(x)$, the system is called an ODE. If $f$ also depends on the derivatives of $\phi$ with respect to $x$, it is called a PDE.

## B.2 Initial Value and Boundary Value Problems

In mathematical modeling and numerical analysis, the study of differential equations plays a critical role in describing the behavior of dynamical systems. The focus of this section is to introduce initial value problems (IVPs) for ODEs and boundary value problems (BVPs) for PDEs.

As we saw in the previous section, an ODE is a relationship between a function and its derivatives. Where $x(t) \in \mathbb{R}^n$ is the state vector, and $f(x, t)$ describes the system's evolution over time. When we refer to the solution of a forward problem, most of the time, we actually address an initial value problem (IVP).

**Definition B.2.1.** An IVP for an ODE consists of an equation of the form $\frac{dx}{dt} = f(x,t)$ together with an initial condition $x(0) = x_0$.

The Picard-Lindelöf Theorem guarantees the existence and uniqueness of a solution to an IVP under certain conditions.

**Theorem B.2.1** (Picard-Lindelöf Theorem)**.** Consider the IVP:

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

If $f$ is Lipschitz continuous with respect to $x$ and continuous in $t$, then there exists a unique solution $x(t)$ in some neighborhood of $t = 0$.

In the context of inverse problem, Takens' Theorem and the Whitney Embedding Theorem are essential for the reconstruction of dynamical systems from time-series data.

**Theorem B.2.2** (Takens' Theorem)**.** Let $\mathcal{M}$ be a compact $d$-dimensional manifold. For a smooth dynamical system on $\mathcal{M}$ with generic properties, the map defined by time-delay coordinates $x(t), x(t-\tau), \ldots, x(t-(k-1)\tau)$, where $k > 2d$, is an *embedding* of $\mathcal{M}$ into $\mathbb{R}^k$.

### B.2.1 Basic theory of PDEs

Unlike ODEs, which typically depend on a single variable (e.g., time), a PDE is an equation that involves an unknown function $u\colon \Omega \subset \mathbb{R}^n \to \mathbb{R}$ of two or more independent variables ($n > 1$) and certain of its partial derivatives. In many cases, the variables can represent time ($t \in \mathbb{R}$) and space ($\boldsymbol{x} \in \mathbb{R}^d$), that in physical application is usually one, two, or three dimensions. This introduces a broader range of problems, including BVPs, where the solution is determined by conditions specified along the boundaries of the spatial domain. However, PDEs, as a mathematical object, are not restricted to such specific physical variable choices. In general, we can denote with the variable $\boldsymbol{x} \in \mathbb{R}^d$, the independent variable of the function $u$, either including or not time as one of the dimensions. In many cases, when the PDE is time-dependent, it also involves an IVP, alongside a BVP, where an initial profile, $u_0(\boldsymbol{x})$, is specified at time $t = 0$.

Occasionally, a PDE in one single coordinate $x \in \mathbb{R}$, while not being properly a PDE as it do not involve *partial* derivatives, can be thought of as a stationary PDE representing stable (or unstable) solution(s) of the associated time evolving problem. That is, the corresponding long-term behavior (i.e., $t \to \infty$) of the system. This scenario is especially interesting in bifurcation theory.

Although it is common to classify PDEs based on two independent variables, such as $x, y$ or $t, x$, this classification scheme does not generalize neatly to higher dimensions, which can create a misconception that there exists a universal classification method for all PDEs. While many classical PDEs are linear, much of the theoretical interest lies in nonlinear PDEs, which govern complex physical and mathematical phenomena.

To represent a PDE mathematically, consider a vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_d)$, with $\alpha_i \in \mathbb{N}_0$, that we call a *multi-index* of order $|\boldsymbol{\alpha}| = \alpha_1 + \ldots + \alpha_d$. Then a *multi-index*

*derivative* of order $k \in \mathbb{N}$ is given by :

$$\mathcal{D}^k u(\mathbf{x}) := \left\{ \frac{\partial^{|\boldsymbol{\alpha}|}\mathbf{u}(\mathbf{x})}{\partial x_1^{\alpha_1} \cdots \partial x_d^{\alpha_d}} : \forall \boldsymbol{\alpha}, \text{such that } |\boldsymbol{\alpha}| = k \right\}. \tag{B.1}$$

**Definition B.2.2.** A general $k$-th order PDE is an equation of the form:

$$F(\mathcal{D}^k u, \mathcal{D}^{k-1} u, \ldots, \mathcal{D}u, u, \boldsymbol{x}) = 0,$$

where $F$ is a given function and $u \colon \Omega \subset \mathbb{R}^n \to \mathbb{R}$ is the unknown solution.

Here, $\mathcal{D}^k u$ denotes the highest-order derivative of $u$. Solving a PDE means finding a function $u(\boldsymbol{x})$ such that $F(u) = 0$.

In some cases, the PDE can be expressed in the form of an operator equation:

$$Lu = f(\boldsymbol{x}) \quad \text{or} \quad Lu = 0,$$

where $L$ is a differential operator acting on $u$, and $f$ is a known function. When $f = 0$, the PDE is called homogeneous.

A linear PDE is one where the operator $L$ is linear, meaning that the coefficients of the derivatives do not depend on the unknown function $u$. More formally, a linear PDE has the form:

$$\sum_{k \leq |\alpha|} a_\alpha(x) D^k u(\boldsymbol{x}) = f(\boldsymbol{x}),$$

where $a_\alpha(x)$ are the coefficients, which depend on the independent variables but not on $u$. If $f(x) = 0$, the equation is called homogeneous.

A PDE is called semilinear if it is linear in the highest-order derivatives, but may be nonlinear in lower-order terms. Otherwise, it is classified as nonlinear. A system of PDEs consists of a collection of equations for a vector-valued unknown function $\mathbf{u} \colon \Omega \to \mathbb{R}^m$

**Boundary Conditions.** In addition to the PDE, problems are typically posed within a domain $\Omega \subset \mathbb{R}^n$, with conditions specified on the boundary of the domain, $\partial\Omega$. A general form for a boundary condition is:

$$\mathcal{B}u = g \quad \text{on} \quad \partial\Omega, \tag{B.2}$$

where $\mathcal{B}$ represents an operator acting on $u$ at the boundary, and $g$ is a given function.

**Definition B.2.3.** A boundary value problem (BVP) for a PDE involves solving the equation within a domain $\Omega$ subject to boundary conditions (BCs) on $\partial\Omega$.

Common types of BCs include: (a) Dirichlet boundary conditions: Specify the value of the function on the boundary, i.e., $u|_{\partial\Omega} = g$; (b) Neumann boundary conditions: Specify the normal derivative of the function on the boundary, i.e., $\frac{\partial u}{\partial n}\big|_{\partial\Omega} = h$; and (c) Robin boundary conditions: A combination of Dirichlet and Neumann conditions, $\alpha u + \beta \frac{\partial u}{\partial n} = g$.

While Dirichlet, Neumann, and Robin boundary conditions are the most well-known types, many nonlinear extensions exist, often tailored to specific applications such as fluid dynamics, heat transfer, biology, and materials science. Free-boundary problems, in contrast, involve boundaries that evolve with the solution itself, making the boundary position part of the problem. These arise in fluid mechanics, phase transitions, and biological growth, where the boundary is not fixed and must be determined as part of the solution process.

### B.2.2 Well-Posedness

A well-posed PDE problem satisfies three key criteria: (i) existence of a solution; (ii) uniqueness of the solution; and (iii) continuous dependence of the solution on the given data.

While uniqueness (ii) may seem desirable, in complex systems modeling, we often encounter phenomena such as multistability and bifurcations, where multiple solutions are expected. For many nonlinear stationary PDEs, multiple solutions can arise, leading to bifurcations in equilibrium states. In contrast, time-dependent PDEs generally evolve towards a unique solution based on the initial conditions. Existence (i) is equally challenging, as there is no universal theory guaranteeing a solution for all PDEs, and existence must be established on a case-by-case basis. Furthermore, how we define a solution is crucial; the next sections will distinguish between classical and weak solutions.

**Classical vs. Weak Solutions.** When we refer to a “solution" of a PDE, it is important to consider the regularity of the function $u$. Is it reasonable to require $u$ being infinitely differentiable or analytic? In many cases, for a PDE of order $k$, we ask that $u$ be at least $k$-times differentiable. Such a solution is called a *classical solution*.

However, for some equations, classical solutions may not exist or may not capture important physical phenomena, such as shock waves. For instance, shock waves are solutions to scalar conservation laws of the form:

$$u_t + F(u)_x = 0.$$

In such cases, weak solutions provide a more general framework by requiring $u$ to satisfy the PDE in an averaged or integral sense, particularly useful in problems involving discontinuities or sharp interfaces.

Mathematically, weak solutions are formulated within the framework of Sobolev spaces. A Sobolev space, denoted $W^{k,p}(\Omega)$, is a function space that generalizes the concept of differentiability, allowing for functions that possess weak (distributional) derivatives up to order $k$, integrable in the $L^p$-sense over a domain $\Omega$. Specifically, $W^{1,2}(\Omega)$, often referred to as $H^1(\Omega)$, is the most common Sobolev space used in PDEs, and consists of functions $u \in L^2(\Omega)$ whose first weak derivatives also belong to $L^2(\Omega)$.

A weak derivative of a function $u$ in $L^2(\Omega)$ is a generalization of the classical derivative. We say that $u$ has a weak derivative $v$ if for every smooth test function

$\varphi \in C_0^\infty(\Omega)$ (compactly supported smooth functions), the following holds:

$$\int_\Omega u(x)\frac{d}{dx}\varphi(x)\,dx = -\int_\Omega v(x)\varphi(x)\,dx.$$

In this context, $v$ is the weak derivative of $u$ and $u$ is differentiable in the "weak sense", meaning that it satisfies the PDE almost everywhere (a.e.) in $\Omega$, not necessarily pointwise.

**Weak Solutions of PDEs.** While for ODEs there is a clear criterion for existence of solutions, given by the Picard-Lindelöf theorem B.2.1, for some PDEs, classical solutions may not exist, and one usually resort to weak solutions. This concept is particularly important for variational problems and finite elements methods:

**Definition B.2.4.** A weak solution to a PDE is a function $u \in H^1(\Omega)$ that satisfies an integral form of the PDE for all test functions $v \in H^1(\Omega)$, where $H^1(\Omega)$ is the Sobolev space of functions with square-integrable first derivatives.

For example, the weak form of the Laplace equation $\Delta u = f$ is:

$$\int_\Omega \nabla u \cdot \nabla v\,dx = \int_\Omega fv\,dx \quad \text{for all } v \in H^1(\Omega).$$

## B.3 Introduction to Bifurcation Theory and Continuation Methods

Bifurcation theory is a mathematical framework that studies the changes in the qualitative behavior of dynamical systems as parameters are varied. In the context of ODEs, bifurcations occur when a small change in a parameter value causes a sudden change in the structure of the solution set.

Bifurcations can be classified into two primary categories: (a) *local bifurcations*, which are analyzed through changes in the local stability properties of equilibria, periodic orbits, or other invariant sets as parameters cross critical thresholds; and (b) *global bifurcations*, which typically occur when larger invariant sets of the system collide with one another or with the system's equilibria. These bifurcations cannot be identified solely by conducting a stability analysis of the equilibria.

Local bifurcations encompass phenomena such as period-halving bifurcations, which lead to order, and period-doubling bifurcations that can induce chaos. A local bifurcation arises when a change in a parameter affects the stability of an equilibrium. In continuous systems, this corresponds to the real part of an eigenvalue of an equilibrium crossing zero. In discrete-time systems, it corresponds to a fixed point possessing a Floquet multiplier with modulus equal to one.

Formally, consider the continuous dynamical system represented by the ordinary differential equation (ODE):

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \lambda), \quad f : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n.$$

A local bifurcation occurs at $(\boldsymbol{x}_0, \lambda_0)$ when the Jacobian matrix $\nabla_{\boldsymbol{x}_0,\lambda_0} f$ has an eigenvalue with a zero real part. If this eigenvalue equals zero, the bifurcation is classified as a steady-state bifurcation; if it is non-zero but purely imaginary, the bifurcation is identified as a Hopf bifurcation.

Among the types of local bifurcations, the *saddle-node bifurcation* is particularly significant. It occurs when two fixed points – one stable and the other unstable –collide and annihilate each other as a parameter crosses a critical value. This phenomenon can be expressed mathematically by its one-dimensional *normal form* ODE, as:

$$\frac{du}{dt} = r - u^2,$$

where $r$ is a parameter. At $r = 0$, the fixed points merge and disappear, resulting in a qualitative change in the system's dynamics.

Another important type is the *Andronov-Hopf bifurcation*, which occurs when a pair of complex conjugate eigenvalues of the linearized system crosses the imaginary axis. This bifurcation can be represented by its two-dimensional *normal form* ODEs, in polar coordinate, as:

$$\frac{dr}{dt} = \pm(\mu - r^2)r; \qquad \frac{d\theta}{dt} = \omega,$$

where $\mu$ is the bifurcation parameter and $\omega$ is the natural angular frequency. For $\mu > 0$, a limit cycle arises; with $+$ sign, the limit cycle is stable, and the bifurcation is called supercritical; while with $-$ sign, the limit cycle is unstable and the bifurcation is called subcritical.

### B.3.1 Numerical Continuation

Continuation algorithms for numerical bifurcation analysis play a crucial role in identifying tipping points. Specifically, these methods help in analyzing the behavior of the complex system at hand as it undergoes a "hard" bifurcation, such as a saddle-node, a limit point, or a *subcritical* Hopf bifurcation. The present discussion focuses on continuation past limit points (saddle-node bifurcations), without aiming to provide a comprehensive guide to all bifurcation scenarios, for which one can refer to a number of published studies [223, 226, 69]. Consider a parameter-dependent dynamical system, described by a system of *autonomous* ODEs (that can also result from the discretization of a system of PDEs, by employing, e.g., finite differences):

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(\boldsymbol{y}; \lambda), \qquad f : \mathbb{R}^{n+1} \to \mathbb{R}^n \tag{B.3}$$

where $\boldsymbol{y} \in \mathbb{R}^n$ is the $n$-dimensional state variable vector, $\lambda \in \mathbb{R}$ is a scalar parameter and the function $\boldsymbol{f}$ is time-independent and sufficiently smooth. The goal is to construct a *solution curve* $\Gamma$ for the system of nonlinear algebraic equations:

$$\Gamma := \{(\boldsymbol{y}; \lambda) \in \mathbb{R}^{n+1} \text{ such that } \boldsymbol{f}(\boldsymbol{y}, \lambda) = \boldsymbol{0}\}, \tag{B.4}$$

corresponding to the equilibria of the system (B.3) for various values of the parameter $\lambda$. The main concept underlying numerical continuation methods is to generate a sequence of pairs $(\boldsymbol{y}_i, \lambda_i)$, $i = 1, 2, \ldots$ that approximate a specific branch of steady-states, satisfying a chosen tolerance criterion ($||\boldsymbol{f}(\boldsymbol{y}_i; \lambda_i)|| \leq tol$ for some small $tol > 0$) and involves a *predictor-corrector* process. We start from a known point on the curve, $(\boldsymbol{y}_i; \lambda_i) \in \Gamma$, and the tangent vector $\boldsymbol{v}_i$ to the curve there, computed through the implicit function theorem. To compute a new point $(\boldsymbol{y}_{i+1}; \lambda_{i+1})$, we need two steps: (a) finding an initial guess for $(\boldsymbol{y}_{i+1}, \lambda_{i+1})$ and (b) iteratively refining the guess to converge towards a point on the curve $\Gamma$ (B.4). We denote the initial guess for $\boldsymbol{x}_{i+1} \equiv (\boldsymbol{y}_{i+1}, \lambda_{i+1})$ as $X^{(0)}_{i+1}$, given by:

$$X^{(0)}_{i+1} = \boldsymbol{x}_i + h\boldsymbol{v}_i, \tag{B.5}$$

where $h$ is a chosen step size. For simplicity, one can use a finite difference approximation for the vector $\boldsymbol{v}_i$, so obtaining the so-called *natural continuation*:

$$\boldsymbol{v}_i = \frac{\boldsymbol{x}_i - \boldsymbol{x}_{i-1}}{h}, \qquad X^{(0)} = 2\boldsymbol{x}_i - \boldsymbol{x}_{i-1}. \tag{B.6}$$

For a small enough $h$ the prediction $X^{(0)}_{i+1}$ is close to the solution curve and can be corrected via e.g. a Newton-like scheme. Indeed, by fixing an increment $d\lambda$ for the value of the parameter $\lambda_{i+1} = \lambda_i + d\lambda$ the system (B.4) has $n$ equations and $n$ unknowns $y_{i+1}$. Let's assume we have reached an approximation point $X^{(k)}$ approximating $x_{i+1}$ at the $k$-th Newton's iteration, in order to find the correction $dX$, we need to compute the Jacobian matrix $\nabla_{\boldsymbol{y}}\boldsymbol{f}(X_k)$ and solve the following system:

$$\nabla_{\boldsymbol{y}}\boldsymbol{f}(X_k)dX = -\boldsymbol{f}(X_k). \tag{B.7}$$

Thus, we can compute the new approximation $X_{k+1} = X^{(k)} + dX$ of $x_{i+1}$. The problem is that this approach can work only far from a bifurcation/critical point, for which the Jacobian matrix becomes singular and the system (B.7) cannot be solved. Beyond critical points, where the Jacobian matrix becomes singular, solution branches can be traced with the aid of numerical bifurcation theory. For example, solution branches past saddle-node bifurcations (limit points) can be traced by applying the so-called pseudo arc-length continuation method. This involves the parametrization of both $\boldsymbol{y}$ and $\lambda$ by the arc-length $s$ on the solution curve. The solution is sought in terms of both $\boldsymbol{y}(s)$ and $\lambda(s)$ in an iterative manner, by solving until convergence an augmented system, involving Eq. (B.4) and the following pseudo arc-length condition:

$$N(X^{(k)}_{i+1}) = (X^{(k)}_{i+1}(s) - X^{(0)}_{i+1})^T \cdot \boldsymbol{v}_i = 0. \tag{B.8}$$

The tangent vector $v_{i+1}$ to the curve at the new point is then computed. The direction along the curve must be preserved, i.e. $\boldsymbol{v}_i^T \boldsymbol{v}_{i+1} = 1$, and $\boldsymbol{v}_{i+1}$ must be normalized.

## B.4 Stochastic processes and SDEs

A stochastic process is a collection of random variables indexed by time or space, representing the evolution of a system over time under uncertainty. Let $(\Omega, \mathcal{A}, P)$ be a probability space, and let $I$ be an index set with a total order $\leq$ (often $\mathbb{N}$, $\mathbb{R}^+$). For every $i \in I$, let $\mathcal{F}_i$ be a sub-$\sigma$-algebra of $\mathcal{A}$. Then $\mathbb{F} := (\mathcal{F}_i)_{i \in I}$ is called a *filtration*, if $\mathcal{F}_k \subseteq \mathcal{F}_\ell$ for all $k \leq \ell$. Filtrations are families of $\sigma$-algebras that are ordered non-decreasingly. If $\mathbb{F}$ is a filtration, then $(\Omega, \mathcal{A}, \mathbb{F}, P)$ is called a filtered probability space. A stochastic process $\{X(t) : t \in T\}$ is defined on this filtered probability space. Stochastic processes can be categorized into various types, including discrete-time processes, continuous-time processes, Markov processes, and Gaussian processes.

### B.4.1 Stochastic Differential Equations (SDEs)

An SDE is a differential equation in which one or more of the terms is a stochastic process. In the context of complex systems, SDEs are employed to model the effects of noise and intrinsic volatility within a system. Noise can arise from external perturbations, measurement errors, or environmental fluctuations, leading to a more accurate representation of real-world phenomena. A prototypical SDE, is the Brownian motion, that can typically be expressed in the following form:

$$dX(t) = a(X(t), t)dt + b(X(t), t)dW(t), \tag{B.9}$$

where $X(t)$ represents the state of the system at time $t$, $a(X(t), t)$ is the drift term that models deterministic behavior, $b(X(t), t)$ is the diffusion term representing the intensity of randomness, and $W(t)$ is a standard Brownian motion (or Wiener process).

Brownian motion, denoted as $W(t)$, is a fundamental example of a stochastic process that serves as a model for random movement. It has several key properties: (i) The increments $W(t) - W(s)$ are normally distributed with mean 0 and variance $t - s$ for $t > s$. (ii) It has independent increments, meaning that the future movements of the process are independent of past movements. (iii) The paths of $W(t)$ are continuous almost surely but nowhere differentiable.

**Stopping time.** A stopping time is a random variable $T$ that signifies the moment at which a specific condition is met. It can be defined with respect to a filtration. Mathematically, $T$ is considered a stopping time if, for any $t$, the event $\{T \leq t\}$ is measurable with respect to the $\sigma$-algebra at that time. Later, we will specifically refer to stopping times in the context of escaping time, where the stopping time is defined by the condition of leaving a particular region. This is particularly relevant for dynamical systems that are in a stable steady state, as we explore the probabilities associated with crossing unstable steady branches and qualitatively undergoing critical transitions to alternate steady states of the system.

## B.5 Complex multiscale systems

Low-dimensional dynamical systems, often described by a few variables, provide insights into the fundamental behavior of systems through ODEs and PDEs. However, many real-world phenomena involve interactions among a large number of components, resulting in high-dimensional dynamics that are significantly more complex. These high-dimensional systems give rise to emergent behaviors, making prediction and analysis more challenging.

Complex systems are defined by their high-dimensional interactions among many components, which lead to emergent behaviors that cannot be easily inferred from the behavior of individual elements. They can be modeled using complex networks, where nodes represent the individual components and edges illustrate the interactions between them. The dynamics of each agent are typically governed by nonlinear equations, giving rise to a wide range of behaviors. In this section, we will outline some of these modeling approaches, focusing on complex networks and ABMs.

**Complex Networks.** Many social, biological, and technological networks exhibit significant and non-trivial topological characteristics, with connection patterns among their elements that are neither completely regular nor entirely random. These characteristics include a heavy-tailed degree distribution, a high clustering coefficient, community structures, and hierarchical arrangements. Two prominent and extensively studied categories of complex networks are scale-free networks and small-world networks, both of which serve as canonical case studies in the field. Scale-free networks are defined by their power-law degree distributions, while small-world networks are characterized by short path lengths and high clustering coefficients.

A complex network can be represented using an adjacency matrix $A$, where the element $a_{ij}$ signifies the presence (and potentially the strength or weight) of the connection between nodes $i$ and $j$. The dynamics of the network can be modeled using a system of ODEs (or SDEs if noise is incorporated):

$$\frac{dx_i}{dt} = f_i(x_i) + \sum_{j=1}^{N} a_{ij} g(x_i, x_j),$$

where $x_i$ denotes the state of the node $i$, $f_i$ is a nonlinear function that describes the internal dynamics of the node $i$, and the subscript $i$ indicates potential heterogeneity among the nodes in the network. The term $g(x_i, x_j)$ captures the influence of connected nodes on a node $i$.

In complex systems, the nonlinear interactions between agents can result in behaviors such as synchronization, oscillations, and even chaotic dynamics.

**Agent-based modeling.** Agent-based modeling is a powerful method for studying complex systems through computer simulations. In ABM, individual agents operate according to agent-agent and environment-agent interactions, as well as to event-driven, stochastic rules. Each agent is defined by attributes and programmed to behave according to specific rules, which include decision-making heuristics that guide their actions based on perceived interests such as reproduction or social status. These agents may also exhibit behaviors such as learning, adaptation, and reproduction. These agents may interact with one another and their environment, with interactions influenced by factors such as spatial distance, changes in their states, environmental changes and decision-making processes.

ABMs generate rich, emergent phenomena that are often difficult to predict from the behavior of individual agents alone. However, the flexibility of these models can lead to over-parameterization, as modelers may attempt to include numerous details to achieve a desired yet unforeseen output, highlighting the intricate nature of complex systems.

### B.5.1 Multiscale dynamics

The study of systems with multiple scales encompasses a wide range of fields. However, modeling and analysis at a single scale have historically been more prevalent. This section focuses on systems characterized by multiple scales.

A key distinction exists between identifying a dynamical system at a coarser scale and simply reducing the dimension of a state space. Coarsening a system may actually require a higher-dimensional state space. For instance, the transition from a particle system to its probability distribution as the number of particles approaches infinity involves a finite-dimensional state space for the particles, described by ODEs, while the probability distribution represents an infinite-dimensional space characterized by PDEs. Thus, the coarser system can exhibit infinite-dimensional characteristics despite originating from a finite-dimensional microscopic model.

If coarsening is not feasible, one can often achieve temporal advancement by reducing the dimensionality of the state space through model order reduction, leading to reduced-order models. This approach maintains the same observational scale, although it may sacrifice some accuracy for a more significant reduction in complexity.

Multiscale systems can often be illustrated through slow-fast systems, which consist of ODEs with fast and slow variables $x$ and $y$. The dynamics are governed by functions $f$ and $g$, separated by a small scale parameter $\epsilon$ (where $0 < \epsilon \ll 1$):

$$\begin{aligned} \epsilon \dot{x} &= f(x, y, \epsilon), \\ \dot{y} &= g(x, y, \epsilon). \end{aligned}$$

In PDE frameworks, the scale parameter frequently appears as a coefficient for the highest-order derivatives, as seen in the Navier-Stokes equations for incompressible flows, where the Reynolds number $Re$ serves as the scale parameter:

$$\begin{aligned} &\rho \left( \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} \right) + \nabla p = \frac{1}{Re} \nabla^2 \mathbf{u}, \\ &\nabla \cdot \mathbf{u} = 0. \end{aligned}$$

Slow-fast ODE and PDE systems can be analyzed using singular perturbation theory, which is grounded in Fenichel theory. A dynamical system with a small scale parameter $\epsilon$ generates trajectories on a manifold $M_\epsilon$. As $\epsilon \to 0$, the system simplifies to trajectories on a manifold $M_0$, often making analysis easier by eliminating the fast components.

Slow-invariant manifolds [125] are crucial in understanding systems with disparate time scales, allowing for the separation of fast and slow dynamics. This approach facilitates the development of reduced-order models that effectively capture the essential behavior of complex systems.

While many properties of multiscale systems can be understood through two-scale systems, some systems may possess finite, countably infinite, or even continuous scales, making differentiation challenging. For example, turbulent flow systems involve a continuum of scales that interact, complicating numerical solutions. In these cases, closure relations, such as assuming a linear relationship between viscous stress and local strain in Newtonian fluids, are often employed to approximate the effects of unaccounted scales.

**Surrogate Models.** Surrogate models aim to replace complex models with simpler, more computationally efficient alternatives. Every model of a natural process can be viewed as a surrogate for the actual process. However, surrogate models specifically substitute one model for another, not a real-world process. When only observational data is available, surrogate models can help elucidate the dynamics and internal structures of the original model, which aligns with the objectives of ML.

### B.5.2 Equation-Free approach: an example of multiscale modeling

The Equation-Free (EF) framework, proposed in [19, 37], operates on the key assumption that for a given microscopic simulator there exists a fundamental coarse-scale description: as the distributions evolve, higher-order moments quickly become dependent on lower-order ones, ultimately converging towards a slow invariant manifold. This concept embodies the singularly perturbed system paradigm, where the interconnected nonlinear ODEs governing the moments of the agent distribution rapidly approach a low-dimensional slow manifold. The main tool employed in the EF approach are the so-called *coarse time-steppers*. Coarse time-steppers establish a link between microscopic simulators, such as ABM, and traditional continuum numerical algorithms. Such methods encompass a series of essential stages, as outlined below:

- Assume we start from a coarse-scale initial condition, corresponding to the appropriate coarse-scale variables $\boldsymbol{x}(t)$ (e.g., diffusion map coordinates) of the evolving ensemble of agents at time $t$;

- Map the macroscopic description, $\boldsymbol{x}(t)$, through a *lifting operator*, $\mathcal{L}$, to an ensemble of consistent microscopic realizations:

$$X(t) = \mathcal{L}(\boldsymbol{x}(t)); \tag{B.10}$$

- For an (appropriately chosen) short macroscopic time $\Delta T$, evolve these realizations using the black-box ABM simulator to obtain the value(s):

$$X(t + \Delta T) = \Phi_{\Delta T}(X(t)) \equiv \Phi_{\Delta T}(\mathcal{L}\boldsymbol{x}(t)); \tag{B.11}$$

- Map the ensemble of agents back to the macroscopic description through the *restriction operator* $\mathcal{M}$:

$$\boldsymbol{x}(t + \Delta T) = \mathcal{M}X(t + \Delta T). \tag{B.12}$$

The entire procedure, i.e., the coarse time-stepper, can be thought of as a "black box":

$$\boldsymbol{x}(t + \Delta T) = \phi_{\Delta T}[\boldsymbol{x}(t)] \equiv \mathcal{M}\Phi_{\Delta T}(\mathcal{L}[\boldsymbol{x}(t)]). \tag{B.13}$$

Such an approach, allows one to accelerate simulations and also to perform bifurcation analysis, exploiting the Newton-GMRES [224] algorithm for the computation of equilibria of (B.13). To find a stationary state $\boldsymbol{x}$ of Eq. (B.13), if there exists, we seek a solution of the following equation:

$$\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{x} - \phi_{\Delta T}[\boldsymbol{x}] = 0, \tag{B.14}$$

wrapping around it the Newton-GMRES method.

## Appendix C

# Fundamentals of Machine Learning and Data Analysis

Unlike the traditional scientific paradigm, where data are collected to verify or refute a hypothesis under study, data now become the central focus of inquiry, especially in the context of complex systems [19, 64, 66]. *Data Science* distinguishes itself from classical science by emphasizing the use of existing data for new purposes, thus reversing the cause-effect relationship between data and information. Hypothesis generation now emerges from data analysis activities, using techniques specifically developed for this purpose. The efforts in creating these techniques have led to the emergence of new research fields, known as *Data Mining*, *Data Analysis* and *Machine Learning*.

The term *Data Mining* refers to the process of uncovering valuable knowledge from *databases* or unstructured data by analyzing the underlying *patterns*. In data science, data, patterns, and processes form a closely linked triad. Data, which consists of records or observations of a phenomenon, serves as the foundational material. Patterns, expressed mathematically, reveal the hidden structures within the data and thus emergent behavior in complex systems. Processes guide the transformation from raw data to meaningful insights. This iterative process involves data preparation, pattern discovery, extraction, and evaluation, culminating in valuable knowledge.

Extracting knowledge from data requires a systematic approach that includes data selection, pre-processing, transformation, mining, and interpretation. A key consideration in this process is evaluating the discovered patterns using criteria such as validity, usefulness, generalization, and comprehensibility. Patterns must remain valid when applied to new data in future iterations of the model. However, data mining is only one step within the broader knowledge discovery process. Data selection identifies relevant subsets, while pre-processing ensures the quality and consistency of the data. Transformation enhances its utility by reducing noise and inconsistencies. Mining applies algorithms to uncover patterns, and interpretation translates these patterns into practical insights for decision-making. Thus, the data-driven process for knowledge discovery is not a specific technique or single algorithm but a heterogeneous discipline that extends

beyond traditional statistical methods into a multidisciplinary domain, particularly when dealing with complex dynamical systems.

## C.1 Data treatment and pre-processing

Before applying data mining and ML algorithms to analyze complex systems, it is essential to conduct a thorough exploration phase to prepare the data for analysis. This involves addressing errors, removing redundancies, and selecting representative samples from the dataset to make accurate inferences about the entire population. Sampling techniques play a crucial role in ensuring that a subset of data accurately reflects the broader dataset, especially when dealing with complex systems characterized by their high dimensionality, nonlinear dynamics, and emergent properties.

During data collection, missing or inconsistent information is a common challenge, making data pre-processing a critical step. This process improves the quality of the data and assists in identifying which variables are most useful for analysis. Pre-processing includes several techniques: data cleaning, which handles missing values, outliers and unrealistic data; data integration, which resolves inconsistencies by combining data from different sources; data transformation, which prepares data for specific analysis algorithms, such as normalization or scaling, to ensure compatibility and comparability, especially when dealing with complex systems characterized by different scales and units; and data reduction, which decreases data volume without compromising the validity of the analysis. This is particularly important for complex systems, which can generate massive amounts of data that can be difficult to analyze. Specifically, common approaches to data transformation and reduction includes: *normalization*, scaling data to a specific range, typically between 0 and 1, to standardize values and improve algorithm performance.; *discretization*, converting continuous variables into discrete intervals. *dimensionality reduction*, finding and extracting new low-dimensional parametrization of the data, using few significant observables, while maintaining its integrity; *subsampling*, which select a subset of data that adequately represents the full dataset; and *feature selection*, identifying and selecting the most relevant variables for the analysis to avoid overfitting and improve model performance.

While these preprocessing steps are essential to ensure that the data is of high quality and properly prepared for analysis, it is important to note that the purpose of this thesis is not to explore the data preprocessing phase in depth. Instead, the focus will be on the dimensionality reduction, feature selection and ML algorithms and their performance, assuming that the data has already been well-sampled, normalized, and preprocessed effectively.

## C.2 Statistical Inference

In statistical inference, the goal is to draw meaningful conclusions about a population based on a representative subset, known as a *sample*. For a set of independent and identically distributed random variables $X_1, X_2, \ldots, X_n$, this constitutes a *random sample*. Each observation $X_k$ in the sample is termed an *independent observation*, and the actual values $x_1, x_2, \ldots, x_n$ are referred to as *realizations* of the sample. When analyzing complex systems, which often involve high-dimensional and nonlinear interactions, traditional inference methods like sampling become more challenging. Problems include estimating system parameters and/or governing macroscopic laws from partial data, handling uncertainty, and dealing with the chaotic, unpredictable or non-linear nature of such systems.

Rarely is a sample studied for its own sake. Instead, the ultimate goal is to make inferences about the population from which the sample was drawn. Since studying an entire population is often too costly, methods must be used that allow generalizing the results obtained from analyzing the sample to the reference population.

This involves two key methods: *parameter estimation* and *hypothesis testing*. Parameter estimation seeks to determine unknown population parameters, such as the mean and variance, using estimators like the sample mean and sample variance. Hypothesis testing is used to assess whether a sample could have originated from a hypothesized population.

*Statistics* are functions of sample data that do not rely on unknown parameters. Notable statistics include:

**Definition C.2.1.** *Sample mean*, denoted as $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$,

**Definition C.2.2.** *Sample variance*, given by $S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$,

**Definition C.2.3.** *$k$-th order sample moment*, defined as $\bar{X}^{(k)} = \frac{1}{n}\sum_{i=1}^{n} X_i^k$.

These basic statistics are instrumental in making inferences about the population from which the sample was drawn. However, in the context of high-dimensional complex systems, they often fail to capture the intricate dependencies and structure of the data. This is particularly true due to the curse of dimensionality, where traditional statistics may become less informative as the dimensionality of the data increases. In such cases, more sophisticated methods are required to reveal the underlying low-dimensional manifold on which the data resides, as is often the case in complex dynamical systems. Techniques like Dimensionality Reduction, including Diffusion Maps or Singular Value Decomposition (SVD), become essential to extract meaningful insights beyond what classical statistics can offer.

## C.3 Estimators

In statistical inference, a key task is estimating unknown parameters of a population from a random sample.

**Definition C.3.1.** Let $(X_1, \ldots, X_n)$ be a random sample from a population with an unknown parameter $\theta$. An *estimator* of $\theta$ is a statistic $\hat{\Theta} = g(X_1, \ldots, X_n)$ used to estimate $\theta$. The value of the estimator for a realization $(x_1, \ldots, x_n)$ is called the *point estimate* of $\theta$.

The quality of an estimator is evaluated by its sampling distribution, with preference given to those closely centered around $\theta$. An estimator $\hat{\Theta}$ is *unbiased* if $\mathbb{E}(\hat{\Theta}) = \theta$.

An estimator $\hat{\theta}_n$, where $n$ represents the number of realizations (sample size), is called *consistent* if it converges in probability to the true parameter $\theta$ as $n \to \infty$. More formally, $\hat{\theta}_n$ is consistent for $\theta$ if for every $\epsilon > 0$:

$$\lim_{n\to\infty} \mathbb{P}\left(|\hat{\theta}_n - \theta| \geq \epsilon\right) = 0. \tag{C.1}$$

Additionally, under certain conditions, consistency can be strengthened to *almost sure convergence*, that is:

$$\mathbb{P}\left(\lim_{n\to\infty} \hat{\theta}_n = \theta\right) = 1. \tag{C.2}$$

**Theorem C.3.1.** The mean is the optimal predictor for minimizing mean squared error (MSE) in a prediction.

*Proof.* For a random variable $X$ with mean $\mu$, the squared error is $(X - c)^2$. The expected value is:

$$\mathbb{E}[(X-c)^2] = \mathbb{E}[(X-\mu+\mu-c)^2] = \mathbb{E}[(X-\mu)^2] + (\mu-c)^2 \geq \mathbb{E}[(X-\mu)^2]. \tag{C.3}$$

Thus, MSE is minimized when $c = \mu$. □

An estimator's quality also depends on its dispersion around $\theta$. The MSE of $\hat{\Theta}$ is defined as: $\text{mse}(\hat{\Theta}) = \mathbb{E}[(\hat{\Theta} - \theta)^2]$.

For unbiased estimators, MSE equals the variance $D^2(\hat{\Theta})$: $\text{mse}(\hat{\Theta}) = D^2(\hat{\Theta}) + [\mathbb{E}(\hat{\Theta}) - \theta]^2$.Thus, an estimator is unbiased if its MSE equals its variance.

**Efficiency of Estimators.** For unbiased estimators, the one with the smallest variance is considered *efficient* because it reduces the uncertainty of the estimation. More formally, an estimator $\hat{\Theta}_1$ is said to outperform an estimator $\hat{\Theta}_2$ if the MSE of $\hat{\Theta}_1$ is lower than that of $\hat{\Theta}_2$, i.e.,

$$\text{MSE}(\hat{\Theta}_1) < \text{MSE}(\hat{\Theta}_2) \tag{C.4}$$

In a specific case where both $\hat{\Theta}_1$ and $\hat{\Theta}_2$ are unbiased estimators of the same parameter $\theta$, their variances can be compared directly. In this context, $\hat{\Theta}_2$ is more efficient than $\hat{\Theta}_1$ if the variance of $\hat{\Theta}_2$ is less than that of $\hat{\Theta}_1$, that is,

$$\text{Var}(\hat{\Theta}_1) > \text{Var}(\hat{\Theta}_2) \tag{C.5}$$

for all possible values of $\theta$.

## C.4 Bayesian Inference

In data analysis, Bayesian statistics plays a crucial role. Bayesian probability is an interpretation of probability where, instead of frequency or propensity, probability is interpreted as rational expectation. In the Bayesian view, a probability is assigned to a hypothesis, whereas in the frequentist approach to inference, a hypothesis is typically tested without assigning it a probability. To evaluate the probability of a hypothesis, the Bayesian must specify a prior probability, which is then updated to a posterior probability in light of new relevant data.

A fundamental part of Bayesian statistics involves the concepts of joint, marginal, and conditional probability. Consider a vector $\boldsymbol{X}$ of $n$ random variables $\boldsymbol{X} = X_1, \ldots, X_n$.

**Definition C.4.1.** The function $F_{\boldsymbol{X}} = F_{X_1,\ldots,X_n} : \mathbb{R}^n \to [0, 1]$ defined as:

$$F_{\boldsymbol{x}} = \mathbb{P}(X_1 \leq x_1, \ldots, X_n \leq x_n) \tag{C.6}$$

is called the *joint distribution function* of the vector $\boldsymbol{X}$ of random variables.

Now, divide the random variables into two groups, $X_A$ and $X_B$, where $A$ and $B$ form a partition of the index set.

**Definition C.4.2.** The *marginal probability* of $X_A$ is defined as:

$$\mathbb{P}(X_A) = \int \mathbb{P}(\boldsymbol{X}) dX_B, \tag{C.7}$$

where the integral is replaced by a sum if the random variables are discrete. Note that if $A$ contains more than one variable, the marginal probability is also a joint probability. Furthermore, if the joint probability distribution equals the product of the marginal probabilities, the variables are said to be *independent*.

**Definition C.4.3.** The *conditional probability* is defined as:

$$\mathbb{P}(X_A|X_B) = \frac{\mathbb{P}(X_A, X_B)}{\mathbb{P}(X_B)}. \tag{C.8}$$

If $X_A$ and $X_B$ are independent, then the marginal probability $\mathbb{P}(X_A) = \mathbb{P}(X_A|X_B)$.
Using the definition of conditional probabilities, we obtain Bayes' Theorem:

**Theorem C.4.1 (Bayes' Theorem).** For two events $X_A$ and $X_B$, the following expression for the conditional probability holds:

$$\mathbb{P}(X_A|X_B) = \frac{\mathbb{P}(X_A)\mathbb{P}(X_B|X_A)}{\mathbb{P}(X_B)}. \tag{C.9}$$

The interpretation given to this theorem, which is widely used in Bayesian inference, is as follows:

$$\text{posterior probability} = \frac{\text{prior probability} \times \text{likelihood}}{\text{evidence}}. \tag{C.10}$$

Bayesian inference applies this framework to various models, continuously updating beliefs in light of new data.

## C.5 Machine Learning

*Machine Learning* (ML) is a branch of *computer science* that focuses on using existing data to predict future outcomes and trends. It involves identifying patterns and models from data through *data mining* techniques. ML applications include tasks like facial recognition and spam filtering, where machines learn autonomously from data without explicit programming. The main goal of ML is to create methods that enable learning for specific tasks. This process can be compared to how a child learns to associate objects with words, forming a classification function $f$ that maps an input $x$ to an output $y$. Outputs in ML can be discrete classes in classification or clustering, or continuous values in regression.

**Types of Machine Learning.** ML is typically divided into two main branches: *Supervised Learning*, where the system is provided with a labeled *training set* containing the desired output, forming the basis for a classification function; and *Unsupervised Learning*, where the system receives an unlabeled *training set* and automatically extracts information from the dataset. In supervised learning, common applications include classification and regression, while unsupervised learning is often used for clustering, grouping similar objects into clusters. In this case, the goal is not to find a mapping function $f$, but to analyze how the points are organized in the *input* space.

In this section, after introducing some basic definitions, we will focus on regression problem, including the Gaussian Processes regression (GPR).

**Supervised Learning.** Supervised learning is an ML technique in which a model is trained to approximate a classification map or a continuous function $f : X \to Y$ to predict outcomes based on input data. This training is achieved by providing the model with labeled examples, enabling it to learn patterns and relationships. Once trained, the model can predict outcomes for new, unseen data.

To evaluate a model's performance, data are typically divided into training, testing, and validation sets. The training set is used to teach the model, the testing set assesses its accuracy, and the validation set fine-tunes it.

These datasets must be distinct and independent. The ultimate goal of ML is to apply the model trained on known data to gain insights from new data, correctly labeling unseen patterns while acquiring predictive and generalization capabilities.

### C.5.1 Regression

Regression is a data mining technique used to identify the functional relationship between specific datasets of continuous values. It is employed for predictive and descriptive analysis of a phenomenon, establishing a connection between input values of interest and the output value. Given a random sample $(X_1, \ldots, X_n)$ and a random variable $Y$, the goal is to find a function $f$ such that

$$Y = f(X_1, \ldots, X_n) + \varepsilon, \tag{C.11}$$

where, since a deterministic relationship is rarely plausible, an error term $\varepsilon$ is added.

Here, $\varepsilon$ is a random variable that represents our uncertainty regarding the relationship $f$ between $X$ and $Y$, and is thus called the error variable.

Consider a set of $n$ observations $(x_1, x_2, \ldots, x_n)$ corresponding to $n$ output values $y_i$. The simple regression model is specified by the relationship

$$y_i = f(x_i; \boldsymbol{\beta}) + \varepsilon_i \tag{C.12}$$

If the function $f$ is linear, it is expressed as:

$$y_i = b_0 + b_1 x_i + \varepsilon_i \tag{C.13}$$

Where $b_0$ is the intercept, and $b_1$ is the slope of the linear model. The error variable $\varepsilon_i$ is assumed to be a zero-mean Gaussian random variable with variance $\sigma^2$. This implies that the random variable $Y_i$ is also a Gaussian random with variance $\sigma^2$.

Let us use the maximum likelihood method, to find point estimators of the parameters. The likelihood function is:

$$L(b_0, b_1, \sigma^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2}\left(\frac{y_i - b_0 - b_1 x_i}{\sigma}\right)^2\right] \tag{C.14}$$

By taking the logarithm, we obtain:

$$\log L(b_0, b_1, \sigma^2) = -\frac{n}{2}(\log 2\pi + \log \sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - b_0 - b_1 x_i)^2 \tag{C.15}$$

At this point, by partially differentiating with respect to $b_0$, $b_1$, and $\sigma^2$, and setting the derivatives to zero, three equations are obtained, whose solutions are the estimators $\hat{b}_0$, $\hat{b}_1$, and $\hat{\sigma}^2$:

$$\hat{b}_1 = \frac{\sum(y_i - \bar{y})(x_i - \bar{x})}{\sum(x_i - \bar{x})^2} \tag{C.16}$$

$$\hat{b}_0 = \bar{y} - \hat{b}_1 \bar{x} \tag{C.17}$$

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{b}_0 - \hat{b}_1 x_i)^2 \tag{C.18}$$

The estimators found above have the desirable property of being unbiased. Note that these estimators are consistent with those obtained by minimizing the loss function using the least squares method.

Let $\hat{y}_i = \hat{b}_0 + \hat{b}_1 x_i$ represent the estimated value of $Y$. The difference between the observed value $y_i$ and the estimated value $\hat{y}_i$ is called the $i$-th residual. The least squares estimates possess an important property known as the decomposition of total variance:

$$\begin{aligned} \sum_{i=1}^{n}(y_i - \bar{y})^2 &= \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2 + \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \\ SST \quad &= \quad SSR \quad + \quad SSE \end{aligned} \tag{C.19}$$

The sum of squares total (SST) is equal to the sum of squares regression (SSR) plus the sum of squares errors (SSE).

**Definition C.5.1.** The coefficient of determination $R^2$ is defined as:

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST} \tag{C.20}$$

The coefficient of determination is a ratio that measures the proportion of data variability explained by the statistical model. It indicates how well the regression captures the variance in the dependent variable. $R^2$ ranges from 0 to 1: 0 means the model does not explain the data at all, while 1 indicates the model explains the data perfectly.

## C.6 Gaussian Process Regression

In this section, we introduce Gaussian Processes (GPs), an exceptionally useful tool for modeling and exploring unknown functions. The fundamental philosophy of this method is based on empirical Bayesian learning of hyperparameters by maximizing a marginal likelihood.

There are various interpretations of a GP. One way is to view it as a distribution over functions with inference performed directly in the function space, while another equivalent approach is to consider the parameter space. Let us consider the training set, $\{(x_i, y_i) \mid i = 1, \dots, \}$, where $\boldsymbol{x}_i$ is a vector of dimension $D$ and $y_i$ is a scalar. Let us collect the *inputs* into a matrix $X$ of dimension $D \times n$ and the *target* values into a vector $\boldsymbol{y}$.

### C.6.1 Gaussian Processes as a Prior over Functions

Gaussian Processes provide a general framework for defining priors over functions. A Gaussian Process is a distribution over functions where any finite collection of function values follows a joint Gaussian distribution. Formally, a Gaussian Process is defined by its mean function $\mu(x)$ and covariance function $k(x, x')$:

$$f(x) \sim \mathcal{GP}(\mu(x), k(x, x')) \tag{C.21}$$

Here, $f(x)$ represents the function drawn from the Gaussian Process. The mean function $\mu(x)$ is usually set to zero for simplicity, while the covariance function $k(x, x')$ determines the function's smoothness and behavior. For example, the squared exponential kernel is commonly used, defined by:

$$k(x, x') = \sigma^2 \exp\left(-\frac{\|x - x'\|^2}{2\ell^2}\right) \tag{C.22}$$

where $\sigma^2$ controls the function's variance and $\ell$ determines the length scale of the function's variations. With this kernel, the covariance between two function values depends on their distance, allowing the model to learn smooth functions and adapt to different scales of variation.

The covariance function is a fundamental element for Gaussian Processes, as it describes our assumptions about the function we aim to learn. From another perspective, the covariance function also defines a similarity between nearby inputs that should yield similar target values.

**Definition C.6.1.** A covariance function $f(\boldsymbol{x}, \boldsymbol{x}')$ is said to be *stationary* if it depends only on the difference $\boldsymbol{x} - \boldsymbol{x}'$. Thus, it is invariant under translations in space.

For example, the squared exponential *kernel* function is stationary. Moreover, if the function depends on the norm of the difference $|\boldsymbol{x} - \boldsymbol{x}'|$, it is called *isotropic*. Thus, it is invariant under all rigid transformations. Given $r = |\boldsymbol{x} - \boldsymbol{x}'|$, the function $k$ depends only on the radius of the sphere centered at $\boldsymbol{x}$ and is therefore known as a radial basis function (RBF).

**Definition C.6.2.** Given a set of inputs $\{\boldsymbol{x}_i \mid i = 1, \ldots, n\}$, we can compute the *Gram matrix* $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. If $k$ is a covariance function, the matrix $K$ is referred to as the *covariance matrix*.

The Gaussian Covariance Function is an example of a radial basis *kernel*,

$$k(r) = \exp\left(-\frac{r^2}{2l^2}\right), \tag{C.23}$$

where the parameter $l$ defines the characteristic scale of the function. This function is infinitely differentiable, implying that the Gaussian Process with this particular covariance function has derivatives of the mean square of every order and is therefore very smooth.

### C.6.2 Bayesian Model Selection and Hyperparameter Tuning

The goal of Gaussian regression is to be a practical tool in applications, which necessitates decisions regarding the model's details. The process of selecting hyperparameters for a covariance function is called *training* the Gaussian Process.

Covariance functions, such as the Gaussian one, can be parameterized in terms of hyperparameters, for example

$$k(\boldsymbol{x}_p, \boldsymbol{x}_q) = \sigma_f^2 \exp\left(-\frac{1}{2}(\boldsymbol{x}_p - \boldsymbol{x}_q)^T M (\boldsymbol{x}_p - \boldsymbol{x}_q)\right) + \sigma_n^2 \delta_{pq}, \tag{C.24}$$

where $\boldsymbol{\theta} = (\{M\}, \sigma_f^2, \sigma_n^2)^T$ is a vector containing the hyperparameters, and $\{M\}$ denotes the parameters in the symmetric matrix $M$. For the Gaussian covariance function, $M = \text{diag}(l)^{-2}$ represents the distance measure, where the hyperparameters $l_1, \ldots, l_D$ define the scale characteristic, i.e., how much distance between two inputs makes them uncorrelated. Such a covariance function uses a method called *automatic relevance determination* (ARD). The idea is that if a parameter $l$ becomes very large, the covariance becomes almost independent of that input, effectively removing that point from inference. The ARD method is thus used to remove irrelevant inputs, as we will present later in feature selection section.

Bayesian principles provide a consistent framework for inference, and particularly when the noise in the data is Gaussian, the integrals over parameters are analytically tractable, making the model very flexible. Since Gaussian Processes are a non-parametric model, it may not be obvious what the model parameters are. Generally, we might consider the values $\boldsymbol{f}$ at the training set inputs as the parameters, meaning more inputs imply more parameters. Applying Bayesian inference, one can obtain the following marginal likelihood:

$$\log \mathbb{P}(\boldsymbol{y} \mid X, \boldsymbol{\theta}) = -\frac{1}{2}\boldsymbol{y}^T K_{\boldsymbol{y}}^{-1}\boldsymbol{y} - \frac{1}{2}\log|K_{\boldsymbol{y}}| - \frac{n}{2}\log(2\pi), \tag{C.25}$$

where $K_{\boldsymbol{y}} = K_f + \sigma_n^2 I$ is the covariance matrix for the noisy target values $\boldsymbol{y}$, and $K_f$ is the covariance matrix for the noise-free values $\boldsymbol{f}$. Note that the marginal likelihood is also conditioned on the hyperparameters $\boldsymbol{\theta}$. To find the hyperparameters, we maximize the marginal likelihood by calculating the partial derivatives with respect to the hyperparameters, yielding

$$\begin{aligned} \frac{\partial}{\partial\theta_j}\log \mathbb{P}(\boldsymbol{y} \mid X, \boldsymbol{\theta}) &= \frac{1}{2}\boldsymbol{y}^T K^{-1}\frac{\partial K}{\partial\theta_j}K^{-1}\boldsymbol{y} - \frac{1}{2}\text{tr}\left(K^{-1}\frac{\partial K}{\partial\theta_j}\right) \\ &= \frac{1}{2}\text{tr}\left(\left(\boldsymbol{\alpha}\boldsymbol{\alpha}^T - K^{-1}\right)\frac{\partial K}{\partial\theta_j}\right), \qquad \text{where } \boldsymbol{\alpha} = K^{-1}\boldsymbol{y} \end{aligned} \tag{C.26}$$

The computational complexity of calculating the marginal likelihood is dominated by the need to compute the inverse of $K$. Traditional methods for inverting a symmetric positive definite matrix require $\mathcal{O}(n^3)$ time. Once the inverse matrix is computed, the derivative calculations take $\mathcal{O}(n^2)$ time for each hyperparameter. Therefore, gradient-based optimization is advantageous.

# Bibliography


[1] E. Bullmore and O. Sporns, "Complex brain networks: graph theoretical analysis of structural and functional systems," *Nature reviews neuroscience*, vol. 10, no. 3, pp. 186–198, 2009.

[2] D. R. Chialvo, "Emergent complex neural dynamics," *Nature physics*, vol. 6, no. 10, pp. 744–750, 2010.

[3] A. Omurtag and L. Sirovich, "Modeling a large population of traders: Mimesis and stability," *Journal of Economic Behavior & Organization*, vol. 61, no. 4, pp. 562–576, 2006.

[4] B. LeBaron, "Agent-based computational finance," *Handbook of computational economics*, vol. 2, pp. 1187–1233, 2006.

[5] J. D. Farmer and D. Foley, "The economy needs agent-based modelling," *Nature*, vol. 460, no. 7256, pp. 685–686, 2009.

[6] R. L. Axtell and J. D. Farmer, "Agent-based modeling in economics and finance: Past, present, and future," *Journal of Economic Literature*, 2022.

[7] W. Steffen, J. Rockström, K. Richardson, T. M. Lenton, C. Folke, D. Liverman, C. P. Summerhayes, A. D. Barnosky, S. E. Cornell, M. Crucifix *et al.*, "Trajectories of the earth system in the anthropocene," *Proceedings of the National Academy of Sciences*, vol. 115, no. 33, pp. 8252–8259, 2018.

[8] V. Dakos, B. Matthews, A. P. Hendry, J. Levine, N. Loeuille, J. Norberg, P. Nosil, M. Scheffer, and L. De Meester, "Ecosystem tipping points in an evolving world," *Nature ecology & evolution*, vol. 3, no. 3, pp. 355–362, 2019.

[9] D. I. Armstrong McKay, A. Staal, J. F. Abrams, R. Winkelmann, B. Sakschewski, S. Loriani, I. Fetzer, S. E. Cornell, J. Rockström, and T. M. Lenton, "Exceeding 1.5 c global warming could trigger multiple climate tipping points," *Science*, vol. 377, no. 6611, p. eabn7950, 2022.

[10] A. Gnanadesikan, G. Fabiani, J. Liu, R. Gelderloos, G. J. Brett, Y. Kevrekidis, T. Haine, M.-A. Pradal, C. Siettos, and J. Sleeman, "Tipping points in overturning

circulation mediated by ocean mixing and the configuration and magnitude of the hydrological cycle: A simple model," *Journal of Physical Oceanography*, vol. 54, no. 7, pp. 1389–1409, 2024.

[11] A. I. Reppas, K. G. Spiliotis, and C. I. Siettos, "Epidemionics: from the host-host interactions to the systematic analysis of the emergent macroscopic dynamics of epidemic networks," *Virulence*, vol. 1, no. 4, pp. 338–349, 2010.

[12] A. Reppas, Y. De Decker, and C. Siettos, "On the efficiency of the equation-free closure of statistical moments: dynamical properties of a stochastic epidemic model on erdős–rényi networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2012, no. 08, p. P08020, 2012.

[13] C. Siettos, C. Anastassopoulou, L. Russo, C. Grigoras, and E. Mylonakis, "Modeling the 2014 ebola virus epidemic–agent-based simulations, temporal analysis and future predictions for liberia and sierra leone," *PLoS currents*, vol. 7, 2015.

[14] S. A. Kauffman, *The origins of order: Self-organization and selection in evolution*. Oxford University Press, 1993.

[15] G. Parisi, "Complex systems: a physicist's viewpoint," *Physica A: Statistical Mechanics and its Applications*, vol. 263, no. 1, pp. 557–564, 1999, proceedings of the 20th IUPAP International Conference on Statistical Physics.

[16] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[17] P. S. Craig, M. Goldstein, J. C. Rougier, and A. H. Seheult, "Bayesian forecasting for complex systems using computer simulators," *Journal of the American Statistical Association*, vol. 96, no. 454, pp. 717–729, 2001.

[18] M. Scheffer and S. R. Carpenter, "Catastrophic regime shifts in ecosystems: linking theory to observation," *Trends in ecology & evolution*, vol. 18, no. 12, pp. 648–656, 2003.

[19] I. G. Kevrekidis, C. W. Gear, and G. Hummer, "Equation-free: The computer-aided analysis of complex multiscale systems," *AIChE Journal*, vol. 50, no. 7, pp. 1346–1355, 2004.

[20] C. E. Hmelo-Silver and R. Azevedo, "Understanding complex systems: Some core challenges," *The Journal of the learning sciences*, vol. 15, no. 1, pp. 53–61, 2006.

[21] M. Scheffer, "Foreseeing tipping points," *Nature*, vol. 467, no. 7314, pp. 411–412, 2010.

[22] C. Giardina, J. Kurchan, V. Lecomte, and J. Tailleur, "Simulating rare events in dynamical processes," *Journal of statistical physics*, vol. 145, pp. 787–811, 2011.

[23] N. Malik and U. Ozturk, "Rare events in complex systems: Understanding and prediction," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 30, no. 9, 2020.

[24] E. Schrödinger, *What is life?: The physical aspect of the living cell*. Cambridge university press Cambridge, 1944, vol. 1.

[25] D. Kondepudi, T. Petrosky, and J. A. Pojman, "Dissipative structures and irreversibility in nature: Celebrating 100th birth anniversary of ilya prigogine (1917–2003)," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 27, no. 10, 2017.

[26] C. Colon, D. Claessen, and M. Ghil, "Bifurcation analysis of an agent-based model for predator–prey interactions," *Ecological Modelling*, vol. 317, pp. 93–106, 2015.

[27] C. I. Siettos, M. D. Graham, and I. G. Kevrekidis, "Coarse brownian dynamics for nematic liquid crystals: Bifurcation, projective integration, and control via stochastic simulation," *The Journal of chemical physics*, vol. 118, no. 22, pp. 10 149–10 156, 2003.

[28] C. Siettos, C. W. Gear, and I. G. Kevrekidis, "An equation-free approach to agent-based computation: Bifurcation analysis and control of stationary states," *EPL (Europhysics Letters)*, vol. 99, no. 4, p. 48007, 2012.

[29] D. Lebiedz, V. Reinhardt, and J. Siehr, "Minimal curvature trajectories: Riemannian geometry concepts for slow manifold computation in chemical kinetics," *Journal of Computational Physics*, vol. 229, no. 18, pp. 6512–6533, 2010.

[30] L. Helfmann, N. Djurdjevac Conrad, A. Djurdjevac, S. Winkelmann, and C. Schütte, "From interacting agents to density-based modeling with stochastic pdes," *Communications in Applied Mathematics and Computational Science*, vol. 16, no. 1, pp. 1–32, 2021.

[31] C. Siettos and L. Russo, "A numerical method for the approximation of stable and unstable manifolds of microscopic simulators," *Numerical Algorithms*, vol. 89, no. 3, pp. 1335–1368, 2022.

[32] N. Zagli, G. A. Pavliotis, V. Lucarini, and A. Alecio, "Dimension reduction of noisy interacting systems," *Physical Review Research*, vol. 5, no. 1, p. 013078, 2023.

[33] C. Fefferman, S. Mitter, and H. Narayanan, "Testing the manifold hypothesis," *Journal of the American Mathematical Society*, vol. 29, no. 4, pp. 983–1049, 2016.

[34] I. G. Kevrekidis, C. W. Gear, J. M. Hyman, P. G. Kevrekidis, O. Runborg, and C. Theodoropoulos, "Equation-free, coarse-grained multiscale computation: Enabling microscopic simulators to perform system-level analysis," *Communications in Mathematical Sciences*, vol. 1, no. 4, pp. 715–762, 2003.

[35] C. Theodoropoulos, Y.-H. Qian, and I. G. Kevrekidis, ""coarse" stability and bifurcation analysis using time-steppers: A reaction-diffusion example," *Proceedings of the National Academy of Sciences*, vol. 97, no. 18, pp. 9840–9843, 2000.

[36] G. Hummer and I. G. Kevrekidis, "Coarse molecular dynamics of a peptide fragment: Free energy, kinetics, and long-time dynamics computations," *The Journal of chemical physics*, vol. 118, no. 23, pp. 10 762–10 773, 2003.

[37] R. Erban, T. A. Frewen, X. Wang, T. C. Elston, R. Coifman, B. Nadler, and I. G. Kevrekidis, "Variable-free exploration of stochastic models: a gene regulatory network example," *The Journal of chemical physics*, vol. 126, no. 15, p. 04B618, 2007.

[38] C. Siettos, "Coarse-grained computational stability analysis and acceleration of the collective dynamics of a monte carlo simulation of bacterial locomotion," *Applied Mathematics and Computation*, vol. 232, pp. 836–847, 2014.

[39] S. Fresca, L. Dede, and A. Manzoni, "A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized pdes," *Journal of Scientific Computing*, vol. 87, no. 2, pp. 1–36, 2021.

[40] S. Lee, M. Kooshkbaghi, K. Spiliotis, C. I. Siettos, and I. G. Kevrekidis, "Coarse-scale pdes from fine-scale observations via machine learning," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 30, no. 1, p. 013141, 2020.

[41] G. K. Batchelor, *An introduction to fluid dynamics*. Cambridge university press, 2000.

[42] E. N. Lorenz, "Deterministic nonperiodic flow," *Journal of atmospheric sciences*, vol. 20, no. 2, pp. 130–141, 1963.

[43] J. M. Yeomans, "The lattice boltzmann equation for fluid dynamics and beyond," *Physics Today*, vol. 55, no. 12, pp. 58–, 60, 2002.

[44] L. Russo, C. I. Siettos, and I. G. Kevrekidis, "Reduced computations for nematic-liquid crystals: A timestepper approach for systems with continuous symmetries," *Journal of non-newtonian fluid mechanics*, vol. 146, no. 1-3, pp. 51–58, 2007.

[45] 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*, vol. 92, no. 2, pp. 1–30, 2022.

[46] M. Doi, "Molecular dynamics and rheological properties of concentrated solutions of rodlike polymers in isotropic and liquid crystalline phases," *Journal of Polymer Science: Polymer Physics Edition*, vol. 19, no. 2, pp. 229–243, 1981.

[47] L. Russo, P. Russo, D. Vakalis, and C. Siettos, "Detecting weak points of wildland fire spread: a cellular automata model risk assessment simulation approach," *Chemical Engineering Transactions*, vol. 36, pp. 253–258, 2014.

[48] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," *Physical review E*, vol. 51, no. 5, p. 4282, 1995.

[49] N. Bellomo, J. Liao, A. Quaini, L. Russo, and C. Siettos, "Human behavioral crowds review, critical analysis and research perspectives," *Mathematical Models and Methods in Applied Sciences*, vol. 33, no. 08, pp. 1611–1659, 2023.

[50] J. L. Casti, *Would-be worlds: How simulation is changing the frontiers of science*. John Wiley & Sons, Inc., 1996.

[51] J. M. Bello-Rivas and R. Elber, "Simulations of thermodynamics and kinetics on rough energy landscapes with milestoning," *Journal of Computational Chemistry*, vol. 37, no. 6, pp. 602–613, 2015.

[52] P. Liu, C. Siettos, C. W. Gear, and I. Kevrekidis, "Equation-free model reduction in agent-based computations: Coarse-grained bifurcation and variable-free rare event analysis," *Mathematical Modelling of Natural Phenomena*, vol. 10, no. 3, pp. 71–90, 2015.

[53] D. Reguera, J. Rubi, and J. Vilar, "The mesoscopic dynamics of thermodynamic systems," pp. 21 502–21 515, 2005.

[54] H. Lei, B. Caswell, and G. E. Karniadakis, "Direct construction of mesoscopic models from microscopic simulations," *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, vol. 81, no. 2, p. 026704, 2010.

[55] R. D. Groot and P. B. Warren, "Dissipative particle dynamics: Bridging the gap between atomistic and mesoscopic simulation," *The Journal of chemical physics*, vol. 107, no. 11, pp. 4423–4435, 1997.

[56] H. Adeli and S. Ghosh-Dastidar, "Mesoscopic-wavelet freeway work zone flow and congestion feature extraction model," *Journal of Transportation Engineering*, vol. 130, no. 1, pp. 94–103, 2004.

[57] A. M. Turing, "The chemical basis of morphogenesis," *Bulletin of mathematical biology*, vol. 52, pp. 153–197, 1990.

[58] S. M. Allen and J. W. Cahn, "A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening," *Acta metallurgica*, vol. 27, no. 6, pp. 1085–1095, 1979.

[59] B. Pena and C. Perez-Garcia, "Stability of turing patterns in the brusselator model," *Physical review E*, vol. 64, no. 5, p. 056213, 2001.

[60] E. F. Keller and L. A. Segel, "Model for chemotaxis," *Journal of theoretical biology*, vol. 30, no. 2, pp. 225–234, 1971.

[61] V. Mayer-Schönberger and K. Cukier, *Big data: A revolution that will transform how we live, work, and think*. Houghton Mifflin Harcourt, 2013.

[62] J. N. Kutz, *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

[63] X. Jin, B. W. Wah, X. Cheng, and Y. Wang, "Significance and challenges of big data research," *Big data research*, vol. 2, no. 2, pp. 59–64, 2015.

[64] A. Karpatne, G. Atluri, J. H. Faghmous, M. Steinbach, A. Banerjee, A. Ganguly, S. Shekhar, N. Samatova, and V. Kumar, "Theory-guided data science: A new paradigm for scientific discovery from data," *IEEE Transactions on knowledge and data engineering*, vol. 29, no. 10, pp. 2318–2331, 2017.

[65] 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*, vol. 378, pp. 686–707, 2019.

[66] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," *Nature Reviews Physics*, vol. 3, no. 6, pp. 422–440, 2021.

[67] G. Fabiani, N. Evangelou, T. Cui, J. M. Bello-Rivas, C. P. Martin-Linares, C. Siettos, and I. G. Kevrekidis, "Task-oriented machine learning assisted surrogates for tipping points of agent-based models," *Nature Communications volume*, vol. 15, no. 4117, pp. 1–13, 2024.

[68] T. C. Silva and L. Zhao, *Machine learning in complex networks*. Springer, 2016.

[69] 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*, vol. 89, no. 2, pp. 1–35, 2021.

[70] H. Vargas Alvarez, G. Fabiani, N. Kazantzis, C. Siettos, and I. G. Kevrekidis, "Discrete-time nonlinear feedback linearization via physics-informed machine learning," *Journal of Computational Physics*, vol. 492, p. 112408, 2023.

[71] H. V. Alvarez, G. Fabiani, N. Kazantzis, I. G. Kevrekidis, and C. Siettos, "Nonlinear discrete-time observers with physics-informed neural networks," *Chaos, Solitons & Fractals*, vol. 186, p. 115215, 2024.

[72] T. Bertalan, F. Dietrich, I. Mezić, and I. G. Kevrekidis, "On learning hamiltonian systems from data," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 29, no. 12, p. 121107, 2019.

[73] J. Pathak, Z. Lu, B. R. Hunt, M. Girvan, and E. Ott, "Using machine learning to replicate chaotic attractors and calculate lyapunov exponents from data," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 27, no. 12, 2017.

[74] P. R. Vlachas, J. Pathak, B. R. Hunt, T. P. Sapsis, M. Girvan, E. Ott, and P. Koumoutsakos, "Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics," *Neural Networks*, vol. 126, pp. 191–217, 2020.

[75] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[76] S. Hochreiter, "Long short-term memory," *Neural Computation MIT-Press*, 1997.

[77] J. Hlinka, D. Hartman, M. Vejmelka, J. Runge, N. Marwan, J. Kurths, and M. Paluš, "Reliability of inference of directed climate networks using conditional mutual information," *Entropy*, vol. 15, no. 6, pp. 2023–2045, 2013.

[78] S. Goswami, M. Yin, Y. Yu, and G. E. Karniadakis, "A physics-informed variational deeponet for predicting crack path in quasi-brittle materials," *Computer Methods in Applied Mechanics and Engineering*, vol. 391, p. 114587, 2022.

[79] Y. Itoh, S. Uenohara, M. Adachi, T. Morie, and K. Aihara, "Reconstructing bifurcation diagrams only from time-series data generated by electronic circuits in discrete-time dynamical systems," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 30, p. 013128, 2020.

[80] L. Boninsegna, F. Nüske, and C. Clementi, "Sparse learning of stochastic dynamical equations," *The Journal of chemical physics*, vol. 148, no. 24, p. 241723, 2018.

[81] T. Qin, Z. Chen, J. D. Jakeman, and D. Xiu, "Deep learning of parameterized equations with applications to uncertainty quantification," *International Journal for Uncertainty Quantification*, vol. 11, no. 2, 2021.

[82] V. Isakov, *Inverse problems for partial differential equations*. Springer, 2006, vol. 127.

[83] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Numerical gaussian processes for time-dependent and nonlinear partial differential equations," *SIAM Journal on Scientific Computing*, vol. 40, no. 1, pp. A172–A198, 2018.

[84] H. Arbabi, J. E. Bunder, G. Samaey, A. J. Roberts, and I. G. Kevrekidis, "Linking machine learning with multiscale numerics: Data-driven discovery of homogenized equations," *Jom*, vol. 72, no. 12, pp. 4444–4457, 2020.

[85] W. Chen, Q. Wang, J. S. Hesthaven, and C. Zhang, "Physics-informed machine learning for reduced-order modeling of nonlinear problems," *Journal of Computational Physics*, vol. 446, p. 110666, 2021.

[86] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, "Learning nonlinear operators via deeponet based on the universal approximation theorem of operators," *Nature Machine Intelligence*, vol. 3, no. 3, pp. 218–229, 2021.

[87] S. Kim, W. Ji, S. Deng, Y. Ma, and C. Rackauckas, "Stiff neural ordinary differential equations," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 31, no. 9, p. 093122, 2021.

[88] Q. Wei, Y. Jiang, and J. Z. Chen, "Machine-learning solver for modified diffusion equations," *Physical Review E*, vol. 98, no. 5, p. 053304, 2018.

[89] J. Han, A. Jentzen, and E. Weinan, "Solving high-dimensional partial differential equations using deep learning," *Proceedings of the National Academy of Sciences*, vol. 115, no. 34, pp. 8505–8510, 2018.

[90] Y. Chen, B. Hosseini, H. Owhadi, and A. M. Stuart, "Solving and learning nonlinear pdes with gaussian processes," *Journal of Computational Physics*, vol. 447, p. 110668, 2021.

[91] S. Dong and Z. Li, "Local extreme learning machines and domain decomposition for solving linear and nonlinear partial differential equations," *Computer Methods in Applied Mechanics and Engineering*, vol. 387, p. 114129, 2021.

[92] W. Ji, W. Qiu, Z. Shi, S. Pan, and S. Deng, "Stiff-pinn: Physics-informed neural network for stiff chemical kinetics," *The Journal of Physical Chemistry A*, vol. 125, no. 36, pp. 8098–8106, 2021.

[93] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "DeepXDE: a deep learning library for solving differential equations," *SIAM Review*, vol. 63, no. 1, pp. 208–228, 2021.

[94] S. Dong and J. Yang, "On computing the hyperparameter of extreme learning machines: Algorithm and application to computational pdes, and comparison with classical and high-order finite elements," *Journal of Computational Physics*, vol. 463, p. 111290, 2022.

[95] G. Fabiani, E. Galaris, L. Russo, and C. 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*, vol. 33, no. 4, 2023.

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

[97] N. B. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. M. Stuart, and A. Anandkumar, "Neural operator: Learning maps between function spaces with applications to pdes." *J. Mach. Learn. Res.*, vol. 24, no. 89, pp. 1–97, 2023.

[98] C. W. Gear, "Numerical initial value problems in ordinary differential equations," *Prentice-Hall series in automatic computation*, 1971.

[99] L. F. Shampine and C. W. Gear, "A user's view of solving stiff ordinary differential equations," *SIAM review*, vol. 21, no. 1, pp. 1–17, 1979.

[100] C. Gear, "Numerical solution of ordinary differential equations: Is there anything left to do?" *SIAM review*, vol. 23, no. 1, pp. 10–24, 1981.

[101] ——, "An introduction to numerical methods for odes and daes," in *Real-time integration methods for mechanical system simulation*. Springer, 1990, pp. 115–126.

[102] L. G. Olson, G. C. Georgiou, and W. W. Schultz, "An efficient finite element method for treating singularities in laplace's equation," *Journal of Computational Physics*, vol. 96, no. 2, pp. 391–410, 1991.

[103] L. F. Shampine and M. W. Reichelt, "The MATLAB ODE suite," *SIAM Journal on Scientific Computing*, vol. 18, no. 1, pp. 1–22, 1997.

[104] L. F. Shampine, M. W. Reichelt, and J. A. Kierzenka, "Solving index-1 daes in matlab and simulink," *SIAM review*, vol. 41, no. 3, pp. 538–552, 1999.

[105] L. N. Trefethen, *Spectral methods in MATLAB*. SIAM, 2000.

[106] A. Quarteroni and A. Valli, *Numerical approximation of partial differential equations*. Springer Science & Business Media, 2008, vol. 23.

[107] J. Hudson, M. Kube, R. Adomaitis, I. G. Kevrekidis, A. Lapedes, and R. Farber, "Nonlinear signal processing and system identification: applications to time series from electrochemical reactions," *Chemical Engineering Science*, vol. 45, no. 8, pp. 2075–2081, 1990.

[108] R. Rico-Martinez, K. Krischer, I. G. Kevrekidis, M. C. Kube, and J. L. Hudson, "Discrete-vs. continuous-time nonlinear signal processing of cu electrodissolution data," *Chemical Engineering Communications*, vol. 118, no. 1, pp. 25–48, 1992.

[109] K. Krischer, R. Rico-Martinez, I. G. Kevrekidis, H. Rotermund, G. Ertl, and J. Hudson, "Model identification of a spatiotemporally varying catalytic reaction," *AIChE J*, vol. 39, no. 1, pp. 89–98, 1993.

[110] S. F. Masri, A. G. Chassiakos, and T. K. Caughey, "Identification of nonlinear dynamic systems using neural networks," *Journal of Applied Mechanics*, vol. 60, no. 1, pp. 123–133, 1993.

[111] T. Chen and H. Chen, "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems," *IEEE Transactions on Neural Networks*, vol. 6, no. 4, pp. 911–917, 1995.

[112] R. González-García, R. Rico-Martìnez, and I. G. Kevrekidis, "Identification of distributed parameter systems: A neural net based approach," *Computers & chemical engineering*, vol. 22, pp. S965–S968, 1998.

[113] H. Lee and I. S. Kang, "Neural algorithm for solving differential equations," *Journal of Computational Physics*, vol. 91, no. 1, pp. 110–131, 1990.

[114] A. J. Meade Jr and A. A. Fernandez, "The numerical solution of linear ordinary differential equations by feedforward neural networks," *Mathematical and Computer Modelling*, vol. 19, no. 12, pp. 1–25, 1994.

[115] R. Gerstberger and P. Rentrop, "Feedforward neural nets as discretization schemes for ODEs and DAEs," *Journal of Computational and Applied Mathematics*, vol. 82, no. 1-2, pp. 117–128, 1997.

[116] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE transactions on neural networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[117] A. Yeşildirek and F. L. Lewis, "Feedback linearization using neural networks," *Automatica*, vol. 31, no. 11, pp. 1659–1664, 1995.

[118] J. Anderson, I. G. Kevrekidis, and R. Rico-Martinez, "A comparison of recurrent training algorithms for time series analysis and system identification," *Computers & Chemical Engineering*, vol. 20, pp. S751–S756, 1996, european Symposium on Computer Aided Process Engineering-6.

[119] S. Y. Shvartsman, C. Theodoropoulos, R. Rico-Martınez, I. Kevrekidis, E. S. Titi, and T. Mountziaris, "Order reduction for nonlinear dynamic models of distributed reacting systems," *Journal of Process Control*, vol. 10, no. 2-3, pp. 177–184, 2000.

[120] J. Berg and K. Nyström, "A unified deep artificial neural network approach to partial differential equations in complex geometries," *Neurocomputing*, vol. 317, pp. 28–41, 2018.

[121] E. Samaniego, C. Anitescu, S. Goswami, V. M. Nguyen-Thanh, H. Guo, K. 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*, vol. 362, p. 112790, 2020.

[122] H. Tang, L. Li, M. Grossberg, Y. Liu, Y. Jia, S. Li, and W. Dong, "An exploratory study on machine learning to couple numerical solutions of partial differential equations," *Communications in Nonlinear Science and Numerical Simulation*, vol. 97, p. 105729, 2021.

[123] F. Calabrò, G. Fabiani, and C. Siettos, "Extreme learning machine collocation for the numerical solution of elliptic pdes with sharp gradients," *Computer Methods in Applied Mechanics and Engineering*, vol. 387, p. 114188, 2021.

[124] M. Kalia, S. L. Brunton, H. G. Meijer, C. Brune, and J. N. Kutz, "Learning normal form autoencoders for data-driven discovery of universal, parameter-dependent governing equations," *arXiv preprint arXiv:2106.05102*, 2021.

[125] D. Patsatzis, G. Fabiani, L. Russo, and C. Siettos, "Slow invariant manifolds of singularly perturbed systems via physics-informed machine learning," *SIAM Journal on Scientific Computing*, vol. 46, no. 4, pp. C297–C322, 2024.

[126] F. Regazzoni, L. Dede, and A. Quarteroni, "Machine learning for fast and reliable solution of time-dependent differential equations," *Journal of Computational physics*, vol. 397, p. 108852, 2019.

[127] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," *Journal of machine learning research*, vol. 18, 2018.

[128] S. Wang, H. Wang, and P. Perdikaris, "Learning the solution operator of parametric partial differential equations with physics-informed deeponets," *Science Advances*, vol. 7, no. 40, p. eabi8605, 2021.

[129] R. R. Coifman, S. Lafon, A. B. Lee, M. Maggioni, B. Nadler, F. Warner, and S. W. Zucker, "Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps," *Proceedings of the national academy of sciences*, vol. 102, no. 21, pp. 7426–7431, 2005.

[130] R. R. Coifman and S. Lafon, "Diffusion maps," *Applied and Computational Harmonic Analysis*, vol. 21, no. 1, pp. 5–30, 2006, special Issue: Diffusion Maps and Wavelets.

[131] D. G. Patsatzis, L. Russo, I. G. Kevrekidis, and C. Siettos, "Data-driven control of agent-based models: An equation/variable-free machine learning approach," *Journal of Computational Physics*, vol. 478, p. 111953, 2023.

[132] M. Balasubramanian and E. L. Schwartz, "The isomap algorithm and topological stability," *Science*, vol. 295, no. 5552, pp. 7–7, 2002.

[133] E. Bollt, "Attractor modeling and empirical nonlinear model reduction of dissipative dynamical systems," *International Journal of Bifurcation and Chaos*, vol. 17, no. 04, pp. 1199–1219, 2007.

[134] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[135] P. G. Papaioannou, R. Talmon, I. G. Kevrekidis, and C. Siettos, "Time-series forecasting using manifold learning, radial basis function interpolation, and geometric harmonics," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 32, no. 8, p. 083113, 2022.

[136] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE journal*, vol. 37, no. 2, pp. 233–243, 1991.

[137] W. Chen and A. L. Ferguson, "Molecular enhanced sampling with autoencoders: On-the-fly collective variable discovery and accelerated free energy landscape exploration," *Journal of computational chemistry*, vol. 39, no. 25, pp. 2079–2102, 2018.

[138] P. R. Vlachas, G. Arampatzis, C. Uhler, and P. Koumoutsakos, "Multiscale simulations of complex systems by learning their effective dynamics," *Nature Machine Intelligence*, vol. 4, no. 4, pp. 359–366, 2022.

[139] D. Floryan and M. D. Graham, "Data-driven discovery of intrinsic dynamics," *Nature Machine Intelligence*, vol. 4, no. 12, pp. 1113–1120, 2022.

[140] M. Raissi, "Deep hidden physics models: Deep learning of nonlinear partial differential equations," *The Journal of Machine Learning Research*, vol. 19, no. 1, pp. 932–955, 2018.

[141] F. Dietrich, A. Makeev, G. Kevrekidis, N. Evangelou, T. Bertalan, S. Reich, and I. G. Kevrekidis, "Learning effective stochastic differential equations from microscopic simulations: Linking stochastic numerics to deep learning," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 33, no. 2, p. 023121, 2023.

[142] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the national academy of sciences*, vol. 113, no. 15, pp. 3932–3937, 2016.

[143] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," *Advances in neural information processing systems*, vol. 31, 2018.

[144] G. Samaey, W. Vanroose, D. Roose, and I. G. Kevrekidis, "Newton–krylov solvers for the equation-free computation of coarse traveling waves," *Computer Methods in Applied Mechanics and Engineering*, vol. 197, no. 43-44, pp. 3480–3491, 2008.

[145] C. Vandekerckhove, I. Kevrekidis, and D. Roose, "An efficient newton-krylov implementation of the constrained runs scheme for initializing on a slow manifold," *Journal of Scientific Computing*, vol. 39, no. 2, pp. 167–188, 2009.

[146] C. J. Dsilva, R. Talmon, R. R. Coifman, and I. G. Kevrekidis, "Parsimonious representation of nonlinear dynamical systems through manifold learning: A chemotaxis case study," *Applied and Computational Harmonic Analysis*, vol. 44, no. 3, pp. 759–773, 2018.

[147] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring strategies for training deep neural networks." *Journal of machine learning research*, vol. 10, no. 1, 2009.

[148] J. Almira, P. Lopez-de Teruel, D. Romero-Lopez, and F. Voigtlaender, "Negative results for approximation using single layer and multilayer feedforward neural networks," *Journal of mathematical analysis and applications*, vol. 494, no. 1, p. 124584, 2021.

[149] B. Adcock and N. Dexter, "The gap between theory and practice in function approximation with deep neural networks," *SIAM Journal on Mathematics of Data Science*, vol. 3, no. 2, pp. 624–655, 2021.

[150] S. Wang, Y. Teng, and P. Perdikaris, "Understanding and mitigating gradient flow pathologies in physics-informed neural networks," *SIAM Journal on Scientific Computing*, vol. 43, no. 5, pp. A3055–A3081, 2021.

[151] S. Wang, X. Yu, and P. Perdikaris, "When and why pinns fail to train: A neural tangent kernel perspective," *Journal of Computational Physics*, vol. 449, p. 110768, 2022.

[152] G. Fabiani, E. Bollt, C. Siettos, and A. N. Yannacopoulos, "Stability analysis of physics-informed neural networks for stiff linear differential equations," *arXiv preprint arXiv:2408.15393*, 2024.

[153] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information theory*, vol. 39, no. 3, pp. 930–945, 1993.

[154] F. P. Kemeth, T. Bertalan, T. Thiem, F. Dietrich, S. J. Moon, C. R. Laing, and I. G. Kevrekidis, "Learning emergent partial differential equations in a learned emergent space," *Nature Communications*, vol. 13, no. 1, p. 3318, 2022.

[155] N. Evangelou, N. J. Wichrowski, G. A. Kevrekidis, F. Dietrich, M. Kooshkbaghi, S. McFann, and I. G. Kevrekidis, "On the parameter combinations that matter and on those that do not: data-driven studies of parameter (non) identifiability," *PNAS Nexus*, vol. 1, no. 4, p. pgac154, 2022.

[156] G. Fabiani, "Random projection neural networks of best approximation: Convergence theory and practical applications," *arXiv preprint arXiv:2402.11397*, 2024.

[157] G. Fabiani, I. G. Kevrekidis, C. Siettos, and A. N. Yannacopoulos, "Randonets: Shallow networks with random projections for learning linear and nonlinear operators," *Journal of Computational Physics*, vol. 520, p. 113433, 2025.

[158] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," *Advances in neural information processing systems*, vol. 20, 2007.

[159] ——, "Weighted sums of random kitchen sinks: replacing minimization with randomization in learning." in *Nips*. Citeseer, 2008, pp. 1313–1320.

[160] ——, "Uniform approximation of functions with random bases," in *2008 46th annual allerton conference on communication, control, and computing*. IEEE, 2008, pp. 555–561.

[161] P. D. Hough and S. A. Vavasis, "Complete orthogonal decomposition for weighted least squares," *SIAM Journal on Matrix Analysis and Applications*, vol. 18, no. 2, pp. 369–392, 1997.

[162] S. S. Yadav and S. M. Jadhav, "Deep convolutional neural network based medical image classification for disease diagnosis," *Journal of Big data*, vol. 6, no. 1, pp. 1–18, 2019.

[163] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan, "Speech recognition using deep neural networks: A systematic review," *IEEE access*, vol. 7, pp. 19 143–19 165, 2019.

[164] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 2020, pp. 38–45.

[165] J. Gui, Z. Sun, Y. Wen, D. Tao, and J. Ye, "A review on generative adversarial networks: Algorithms, theory, and applications," *IEEE transactions on knowledge and data engineering*, vol. 35, no. 4, pp. 3313–3332, 2021.

[166] K. Han, Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, Y. Tang, A. Xiao, C. Xu, Y. Xu *et al.*, "A survey on vision transformer," *IEEE transactions on pattern analysis and machine intelligence*, vol. 45, no. 1, pp. 87–110, 2022.

[167] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[168] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, 2018.

[169] R. Ferdiana *et al.*, "A systematic literature review of intrusion detection system for network security: Research trends, datasets and methods," in *2020 4th International Conference on Informatics and Computational Sciences (ICICoS)*. IEEE, 2020, pp. 1–6.

[170] B. Irie and S. Miyake, "Capabilities of three-layered perceptrons," in *IEEE 1988 International Conference on Neural Networks*, 1988, pp. 641–648 vol.1.

[171] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[172] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[173] K.-I. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural networks*, vol. 2, no. 3, pp. 183–192, 1989.

[174] H. N. Mhaskar and C. A. Micchelli, "Approximation by superposition of sigmoidal and radial basis functions," *Advances in Applied mathematics*, vol. 13, no. 3, pp. 350–373, 1992.

[175] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural networks*, vol. 6, no. 6, pp. 861–867, 1993.

[176] I. Gühring, G. Kutyniok, and P. Petersen, "Error bounds for approximations with deep relu neural networks in w s, p norms," *Analysis and Applications*, vol. 18, no. 05, pp. 803–859, 2020.

[177] H. Montanelli, HadrienYang and Q. Du, "Deep relu networks overcome the curse of dimensionality for generalized bandlimited functions," *Journal of Computational Mathematics*, vol. 39, no. 6, pp. 801–815, 2021.

[178] E. Weinan, C. Ma, and L. Wu, "Barron spaces and the compositional function spaces for neural network models," *arXiv preprint arXiv:1906.08039*, 2019.

[179] H. Shen, ZuoweiYang and S. Zhang, "Deep network approximation characterized by number of neurons," *Communications in Computational Physics*, vol. 28, no. 5, pp. 1768–1811, 2020.

[180] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Computation*, vol. 3, no. 2, pp. 246–257, 1991.

[181] E. L. Bolager, I. Burak, C. Datar, Q. Sun, and F. Dietrich, "Sampling weights of deep neural networks," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[182] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[183] J. B. Butcher, D. Verstraeten, B. Schrauwen, C. R. Day, and P. W. Haycock, "Reservoir computing and extreme learning machines for non-linear time-series data analysis," *Neural Networks*, vol. 38, pp. 76–89, 2013.

[184] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, 1961.

[185] M. Minsky and S. Papert, "An introduction to computational geometry," *Cambridge tiass., HIT*, vol. 479, no. 480, p. 104, 1969.

[186] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[187] A. N. Kolmogorov, "On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition," in *Doklady Akademii Nauk*, vol. 114(5). Russian Academy of Sciences, 1957, pp. 953–956.

[188] A. Wieland and R. Leighton, "Geometric analysis of neural network capabilities," in *Proceedings of the 1st IEEE International Conference on Neural Networks*, San Diego, CA, USA, 1987, pp. 385–392.

[189] Gallant, "There exists a neural network that does not make avoidable mistakes," in *IEEE 1988 International Conference on Neural Networks*. IEEE, 1988, pp. 657–664.

[190] Carroll and Dickinson, "Construction of neural nets using the radon transform," in *International 1989 Joint Conference on Neural Networks*. IEEE, 1989, pp. 607–611.

[191] F. Dan Foresee and M. Hagan, "Gauss-newton approximation to bayesian learning," in *Proceedings of International Conference on Neural Networks (ICNN'97)*, vol. 3, 1997, pp. 1930–1935 vol.3.

[192] K. G. Murty and S. N. Kabadi, "Some np-complete problems in quadratic and nonlinear programming," *Mathematical Programming*, vol. 39, pp. 117–129, 1987.

[193] A. Blum and R. Rivest, "Training a 3-node neural network is np-complete," *Advances in neural information processing systems*, vol. 1, 1988.

[194] W. F. Schmidt, M. A. Kraaijveld, R. P. Duin *et al.*, "Feed forward neural networks with random weights," in *International Conference on Pattern Recognition*. IEEE Computer Society Press, 1992, pp. 1–1.

[195] Y.-H. Pao and Y. Takefuji, "Functional-link net computing: theory, system architecture, and functionalities," *Computer*, vol. 25, no. 5, pp. 76–79, 1992.

[196] B. Igelnik and Y.-H. Pao, "Stochastic choice of basis functions in adaptive function approximation and the functional-link net," *IEEE Transactions on Neural Networks*, vol. 6, no. 6, pp. 1320–1329, 1995.

[197] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note," *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, no. 34, p. 13, 2001.

[198] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1-3, pp. 489–501, 2006.

[199] W. B. Johnson, "Extensions of lipschitz mappings into a hilbert space," *Contemp. Math.*, vol. 26, pp. 189–206, 1984.

[200] C. Datar, T. Kapoor, A. Chandra, Q. Sun, I. Burak, E. L. Bolager, A. Veselovska, M. Fornasier, and F. Dietrich, "Solving partial differential equations with sampled neural networks," *arXiv preprint arXiv:2405.20836*, 2024.

[201] S. Dasgupta and A. Gupta, "An elementary proof of a theorem of Johnson and Lindenstrauss," *Random Structures & Algorithms*, vol. 22, no. 1, pp. 60–65, 2003.

[202] S. S. Vempala, *The random projection method*. American Mathematical Soc., 2005, vol. 65.

[203] A. N. Gorban, I. Y. Tyukin, D. V. Prokhorov, and K. I. Sofeikov, "Approximation with random bases: Pro et contra," *Information Sciences*, vol. 364, pp. 129–145, 2016.

[204] Y. Ito, "Nonlinearity creates linear independence," *Advances in Computational Mathematics*, vol. 5, pp. 189–203, 1996.

[205] R. D. Fierro, G. H. Golub, P. C. Hansen, and D. P. O'Leary, "Regularization by truncated total least squares," *SIAM Journal on Scientific Computing*, vol. 18, no. 4, pp. 1223–1241, 1997.

[206] D. Gaier, *Lectures on complex approximation*. Springer, 1987, vol. 188.

[207] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, A. Stuart, K. Bhattacharya, and A. Anandkumar, "Multipole graph neural operator for parametric partial differential equations," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6755–6766, 2020.

[208] K. Azizzadenesheli, N. Kovachki, Z. Li, M. Liu-Schiaffini, J. Kossaifi, and A. Anandkumar, "Neural operators for accelerating scientific simulations and design," *Nature Reviews Physics*, pp. 1–9, 2024.

[209] M. V. de Hoop, D. Z. Huang, E. Qian, and A. M. Stuart, "The cost-accuracy trade-off in operator learning with neural networks," *arXiv preprint arXiv:2203.13181*, 2022.

[210] S. Venturi and T. Casey, "Svd perspectives for augmenting deeponet flexibility and interpretability," *Computer Methods in Applied Mechanics and Engineering*, vol. 403, p. 115718, 2023.

[211] S. Goswami, A. Bora, Y. Yu, and G. E. Karniadakis, "Physics-informed deep neural operator networks," in *Machine Learning in Modeling and Simulation: Methods and Applications*. Springer, 2023, pp. 219–254.

[212] T. Tripura and S. Chakraborty, "Wavelet neural operator for solving parametric partial differential equations in computational mechanics problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 404, p. 115783, 2023.

[213] V. Fanaskov and I. V. Oseledets, "Spectral neural operators," in *Doklady Mathematics*, vol. 108, Suppl 2. Springer, 2023, pp. S226–S232.

[214] Y. Liao, S.-C. Fang, and H. L. Nuttle, "Relaxed conditions for radial-basis function networks to be universal approximators," *Neural Networks*, vol. 16, no. 7, pp. 1019–1028, 2003.

[215] J. Park and I. W. Sandberg, "Approximation and radial-basis-function networks," *Neural computation*, vol. 5, no. 2, pp. 305–316, 1993.

[216] G. H. Golub, P. C. Hansen, and D. P. O'Leary, "Tikhonov regularization and total least squares," *SIAM journal on matrix analysis and applications*, vol. 21, no. 1, pp. 185–194, 1999.

[217] Y. Lu, R. Maulik, T. Gao, F. Dietrich, I. G. Kevrekidis, and J. Duan, "Learning the temporal evolution of multivariate densities via normalizing flows," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 32, no. 3, p. 033121, 2022.

[218] S. Lee and Y. Shin, "On the training and generalization of deep operator networks," *SIAM Journal on Scientific Computing*, vol. 46, no. 4, pp. C273–C296, 2024.

[219] N. H. Nelsen and A. M. Stuart, "Operator learning using random features: A tool for scientific computing," *SIAM Review*, vol. 66, no. 3, pp. 535–571, 2024.

[220] F. Brezzi, J. Rappaz, and P.-A. Raviart, "Finite dimensional approximation of nonlinear problems: Part iii: Simple bifurcation points," *Numerische Mathematik*, vol. 38, no. 1, pp. 1–30, 1982.

[221] T. F. C. Chan and H. B. Keller, "Arc-length continuation and multigrid techniques for nonlinear elliptic eigenvalue problems," *SIAM Journal on Scientific and Statistical Computing*, vol. 3, no. 2, pp. 173–194, 1982.

[222] R. Glowinski, H. B. Keller, and L. Reinhart, "Continuation-conjugate gradient methods for the least squares solution of nonlinear boundary value problems," *Siam Journal on Scientific and Statistical Computing*, vol. 6, pp. 793–832, 1985.

[223] A. Dhooge, W. Govaerts, Y. A. Kuznetsov, H. G. E. Meijer, and B. Sautois, "New features of the software matcont for bifurcation analysis of dynamical systems," *Mathematical and Computer Modelling of Dynamical Systems*, vol. 14, no. 2, pp. 147–175, 2008.

[224] C. T. Kelley, *Iterative methods for optimization*. SIAM, 1999.

[225] W. J. Govaerts, *Numerical methods for bifurcations of dynamical equilibria*. SIAM, 2000.

[226] E. Doedel and L. S. Tuckerman, *Numerical methods for bifurcation problems and large-scale dynamical systems*. Springer Science & Business Media, 2012, vol. 119.

[227] E. J. Allen, J. A. Burns, and D. S. Gilliam, "Numerical approximations of the dynamical system generated by burgers' equation with neumann-dirichlet boundary conditions," *ESAIM: Mathematical Modelling and Numerical Analysis-Modélisation Mathématique et Analyse Numérique*, vol. 47, no. 5, pp. 1465–1492, 2013.

[228] J. P. Boyd, "An analytical and numerical study of the two-dimensional bratu equation," *Journal of Scientific Computing*, vol. 1, pp. 183–206, 1986.

[229] M. Hajipour, A. Jajarmi, and D. Baleanu, "On the accurate discretization of a highly nonlinear boundary value problem," *Numerical Algorithms*, vol. 79, pp. 679–695, 2018.

[230] M. A. Z. Raja, S.-u.-I. Ahmad, and R. Samar, "Neural network optimized with evolutionary computing technique for solving the 2-dimensional bratu problem," *Neural Computing and Applications*, vol. 23, pp. 2199–2210, 2013.

[231] E. R. Benton and G. W. Platzman, "A table of solutions of the one-dimensional burgers equation," *Quarterly of Applied Mathematics*, vol. 30, no. 2, pp. 195–212, 1972.

[232] A. Mohsen, "A simple solution of the bratu problem," *Computers & Mathematics with Applications*, vol. 67, no. 1, pp. 26–33, 2014.

[233] M. I. Syam, "The modified broyden-variational method for solving nonlinear elliptic differential equations," *Chaos, Solitons & Fractals*, vol. 32, no. 2, pp. 392–404, 2007.

[234] P. Collins and O. U. K. M. Inst;, *Differential and Integral Equations: Part II*. University of Oxford Mathematical Institute, 1988.

[235] N. De Villiers and D. Glasser, "A continuation method for nonlinear regression," *SIAM Journal on Numerical Analysis*, vol. 18, no. 6, pp. 1139–1154, 1981.

[236] T. A. Davis, "Algorithm 915, suitesparseqr: Multifrontal multithreaded rank-revealing sparse qr factorization," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, pp. 1–22, 2011.

[237] G. Söderlind, "Automatic control and adaptive time-stepping," *Numerical Algorithms*, vol. 31, no. 1, pp. 281–310, 2002.

[238] I. Gladwell, L. Shampine, and R. Brankin, "Automatic selection of the initial step size for an ode solver," *Journal of computational and applied mathematics*, vol. 18, no. 2, pp. 175–192, 1987.

[239] H. Robertson, "The solution of a set of reaction rate equations," *Numerical Analysis: an Introduction*, vol. 178182, 1966.

[240] A. Prothero and A. Robinson, "On the stability and accuracy of one-step methods for solving stiff systems of ordinary differential equations," *Mathematics of Computation*, vol. 28, no. 125, pp. 145–162, 1974.

[241] W. H. Enright, K. R. Jackson, S. P. Nørsett, and P. G. Thomsen, "Interpolants for runge-kutta formulas," *ACM Transactions on Mathematical Software (TOMS)*, vol. 12, no. 3, pp. 193–218, 1986.

[242] L. Métivier and P. Montarnal, "Strategies for solving index one dae with non-negative constraints: Application to liquid–liquid extraction," *Journal of Computational Physics*, vol. 231, no. 7, pp. 2945–2962, 2012.

[243] I. Karatzas, I. Karatzas, S. Shreve, and S. E. Shreve, *Brownian motion and stochastic calculus*. Springer Science & Business Media, 1991, vol. 113.

[244] R. FitzHugh, "Impulses and physiological states in theoretical models of nerve membrane," *Biophysical journal*, vol. 1, no. 6, pp. 445–466, 1961.

[245] P. L. Bhatnagar, E. P. Gross, and M. Krook, "A model for collision processes in gases. i. small amplitude processes in charged and neutral one-component systems," *Physical Review*, vol. 94, pp. 511–525, May 1954.

[246] Y. Qian and S. Orszag, "Scalings in diffusion-driven reaction $a+b \rightarrow c$: Numerical simulations by lattice bgk models," *J Stat Phys*, vol. 81, p. 237–253, 1995.