https://www.aimspress.com/journal/mine

Mathematics in Engineering, 8(3): 402–455.
DOI:10.3934/mine.2026013
Received: 25 September 2025
Revised: 08 April 2026
Accepted: 11 June 2026
Published: 18 June 2026

Research article

Model reduction of parametric ordinary differential equations via autoencoders: representation properties and convergence analysis

Enrico Ballini[1,*], Marco Gambarini[2,3], Alessio Fumagalli[1], Luca Formaggia[1], Anna Scotti[1] and Paolo Zunino[1]

[1] MOX, Department of Mathematics, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy

[2] CMCC Foundation - Euro-Mediterranean Center on Climate Change, Via Marco Biagi 5, 73100 Lecce, Italy

[3] RFF-CMCC European Institute on Economics and the Environment, Via Solari 11, 20144 Milano, Italy

* **Correspondence:** Email: enrico.ballini@polimi.it.

Abstract: We propose a reduced-order modeling approach for nonlinear, parameter-dependent ordinary differential equations (ODE). Dimensionality reduction is achieved using nonlinear maps represented by autoencoders. The resulting low-dimensional ODE is then solved using standard integration in time schemes, and the high-dimensional solution is reconstructed from the low-dimensional one. We investigate the architecture of neural networks for constructing effective autoencoders that hold necessary properties to reconstruct the input manifold with exact representation capabilities. We study the convergence of the reduced-order model to the high-fidelity one. Numerical experiments show the robustness and accuracy of our approach in different scenarios, highlighting its effectiveness in highly complex and nonlinear settings without sacrificing accuracy. Moreover, we examine how the reduction influences the stability properties of the reconstructed high-dimensional solution.

Keywords: reduced order modeling; parametric differential equations; autoencoders; nonlinear dimensionality reduction; latent dynamics; manifold learning; architecture analysis; convergence analysis; stability analysis; scientific machine learning

1. Introduction

Ordinary differential equations (ODEs) and partial differential equations (PDEs) are ubiquitous in the modeling of time-dependent phenomena in a wide range of disciplines including, among

others, fluid dynamics [64], chemical kinetics [63], geological simulations [51] and solid continuum mechanics [48]. A typical approach to solving PDEs involves two steps: The first is to discretize the spatial variables, resulting in a high-dimensional system of ODEs, while the second step regards the integration in time. For simplicity, in this work we refer only to ODEs, with the understanding that the methodologies developed are equally applicable to PDEs in the sense described above.

The models that capture the phenomena of interest may depend on some parameters, denoted by μ. This variability may reflect uncertainty in physical properties that are difficult to measure, or may arise from the need to explore different scenarios, see, e.g., [2]. When the parameters vary, we obtain parametric ordinary differential equations. In many applications, such as uncertainty quantification, optimization, or inverse problems, it is often necessary to solve these ODEs several times for a wide range of parameter values. Direct computation can be prohibitively expensive, requiring the community to develop reduced-order modeling techniques that alleviate this computational burden.

In the past few years, scientific machine learning has increasingly incorporated neural networks, like autoencoders for data compression, into reduced-order modeling and methods for improving the efficiency of ODE solvers. Autoencoders have found widespread application in diverse fields and for multiple purposes. For example, in [17, 19, 24, 33, 39, 50], autoencoders have been employed to compress and subsequently reconstruct discrete solutions that approximate the governing physical equations on a computational grid, possibly using a linear interpolation of the low-dimensional solution [62].

A key strength of autoencoders lies in their ability to achieve maximal information compression, effectively addressing the challenge of slow Kolmogorov n-width decay [17]. The Kolmogorov n-width measures how well a nonlinear solution manifold can be captured by an n-dimensional linear subspace; a slow decay indicates that many linear modes are needed to represent the manifold accurately. In contrast, nonlinear autoencoders can approximate the underlying nonlinear manifold with the minimum number of parameters required, thus overcoming a fundamental limitation of the traditional proper orthogonal decomposition (POD) approach [17], where a linear combination of basis vectors is adopted to describe the manifold. This advantage comes from the ability of neural networks to learn nonlinear maps that describe high-dimensional structures using only a few latent variables. Moreover, autoencoders handle discontinuities effectively [39] and are robust to non-affine parameter dependencies, unlike the well-established POD.

Neural networks have emerged as a powerful tool for the solution of ODEs. A comprehensive survey of how machine learning techniques can improve ODE solvers is provided in [40]. Further insight in this area is provided in [47], which highlights the similarities between residual neural networks [31] and the discrete time-stepping of ODE solvers. Subsequently, the authors of [10] proposed the idea of learning the right-hand side of an ODE directly from the data. They also developed an efficient alternative to standard backpropagation for computing gradients of the loss function with respect to network weights using the adjoint sensitivity method. The Ph.D. thesis by Kidger [34] offers a detailed exploration of neural differential equations, and is a valuable resource not only for its own contributions but also for the extensive references it contains. In a related work, [61] demonstrated that recurrent neural networks can effectively model ODE systems even when training data are sampled at irregular time intervals. Furthermore, [9] proposed a framework for modeling discontinuous changes, called *events*, in a time-dependent system. More recently, [8] proposed a specific autoencoder architecture for time-dependent PDEs that learns a time-dependent vector field

associated with input spatial points without the need for a spatial mesh. The work in [44] uses a combination of a convolutional autoencoder to identify the reduced solution and an additional neural network that approximates a one-step time marching procedure in the reduced space. This idea has been tested on many different scenarios in which physical equations are solved.

In the context of reduced-order modeling, the authors in [26, 30] employ recurrent neural networks to advance in time the reduced solution obtained from an encoder. Rather than training the autoencoder to approximate the identity map, one can design it to predict the solution at the next time step based on previous states. This strategy is illustrated in [53], where the autoencoder takes the form of a UNet [60]. In [20] and similarly in [11, 36, 41], a feedforward network first computes the reduced representation, and then this state is propagated over a short time interval by a long-short-term memory (LSTM) network. More recently, the work of [22] has explored the use of self-attention neural networks to perform time evolution. The study in [14] employs an unknown low-dimensional ODE to approximate the solution of a related high-dimensional ODE. In this framework, the encoder is used solely to specify the initial condition of the low-dimensional system.

To give a comprehensive overview, it is worth mentioning other approaches based on neural networks that can be used to construct surrogate models, such as physics-informed neural networks [58], the use of graph neural networks to reproduce the solution in parameterized meshes [16], mesh-informed neural networks [18], and latent dynamic networks to directly estimate a quantity of interest from latent dynamics [59]. A particularly active area of research focuses on the approximation of the operators that define the governing differential equations. Several well-established architectures fall into this category, such as the DeepONet [46] and the Fourier neural operator [43], wavelet neural operator [65] and kernel operator learning [68].

We propose the *Dynamical Reduced Embedding* (DRE) method based on an autoencoder approximated by fully connected neural networks to define the dynamics of a low-dimensional ODE, which is subsequently learned from the data. Once the model is constructed, it provides an approximation of the solution to the high-dimensional ODE via the efficient solution of a low-dimensional ODE, followed by a reconstruction step to recover the solution in the high-dimensional space. Unlike architectures that rely on temporal dependencies and require sequential training (such as backpropagation through time), the use of an autoencoder allows independent processing of each input instance. This enables a straightforward parallelization of the training, as there are no temporal causality constraints to enforce [25].

Several recent approaches employ data-driven representations to approximate the solution manifold and its dynamics. On one hand, modal decomposition techniques such as dynamic mode decomposition (DMD) provide a spectral characterization of nonlinear dynamics by computing the eigendecomposition of an approximating linear operator and can be interpreted as finite-dimensional approximations of Koopman spectral analysis. While highly effective for extracting coherent dynamical structures from time-series data, these approaches rely fundamentally on linear representations and are not designed to approximate nonlinear solution manifolds or to provide convergence guarantees for reduced-order models. On the other hand, recent neural-network-based approaches construct nonlinear approximation manifolds or learn reduced dynamics directly from data. For instance, ANN-augmented projection-based reduced-order models aim at mitigating the Kolmogorov barrier by combining classical projection with nonlinear manifolds learned from data [5], while operator-inference frameworks, although generally related to projection and regression-driven

approaches, can be combined by learning low-dimensional dynamics through rollout-based training to improve robustness and stability properties [66]. Our work is aligned with these contributions in its use of nonlinear representations and data-driven reduced dynamics, and is closely related to the framework proposed by Farenga et al. [14], where a high-dimensional ODE is linked to a low-dimensional one.

In a broad sense, the main originality of the present work lies in the joint analysis of the representation error and dynamical approximation error and the characterization of the architectural conditions ensuring injectivity of the encoder. More precisely, we develop a mathematical framework for autoencoder-based model reduction that explicitly distinguishes representation and approximation errors, characterizes the topology of the solution manifold, and establishes convergence and stability results for the fully approximated reduced system. In particular, unlike approaches that treat the latent dynamics as a black-box model, our formulation introduces constrained reduced dynamics that enable rigorous analysis. The theoretical results are supported by representative numerical experiments inspired by applications.

The paper is organized as follows. In Section 2, we present the core of the reduced-order model strategy along with the main theoretical results. The remaining sections are dedicated to studying the methodology in detail. In particular, Section 3 investigates the conditions under which a solution manifold can be compressed by an autoencoder with null representation error. Then Section 4 expands this foundation by formalizing the solution manifold of a parametric ODE as a low-dimensional topological manifold and deriving the minimal latent dimension required for lossless compression. Time discretization is then addressed in Section 5, where a study of stability and convergence in time is presented. The fully approximated problem based on the use of neural networks is presented in Section 6, along with an analysis of the global error. Finally, numerical experiments are reported in Section 7 to validate the proposed approach.

2. Problem setup

We consider a parametric ODE of N components, called *N-ODE*, in the time interval $(0, T]$, with $T > 0$, in the form

$$\begin{aligned} \dot{u}_N &= F_N(u_N; \mu), \\ u_N(0) &= u_{N0}(\mu), \end{aligned} \tag{2.1}$$

where $u_N : [0, T] \to \mathbb{R}^N$ is the *N-solution*, $F_N : \mathbb{R}^N \to \mathbb{R}^N$ is a possibly nonlinear right-hand side, and u_{N0} is the given initial condition. The parameters μ vary in a compact set with non-empty interior $\mathcal{P} \subset \mathbb{R}^{a-1}$, with $a - 1 \geq 1$ being the number of parameters. Without loss of generality, we restrict our analysis to the autonomous case [37]. Unless it is important to remark it, we drop the dependence on μ in the notation and simply write the parametric ODE as

$$\begin{aligned} \dot{u}_N &= F_N(u_N), \\ u_N(0) &= u_{N0}. \end{aligned} \tag{2.2}$$

Since (2.2) can be computationally expensive due to N being large, for instance if the system results from a prior space discretization of a PDE, we aim to solve a smaller ODE, called *n-ODE*, whose size

is $n \ll N$

$$\begin{aligned} \dot{u}_n &= F_n(u_n), \\ u_n(0) &= \Psi'(u_{N0}), \end{aligned} \tag{2.3}$$

where Ψ' is a map that compresses the N-solution. It will be detailed in the following sections. Subsequently, from u_n, called the *n-solution*, we reconstruct a high-dimensional solution, possibly approximated, through a map Ψ, i.e., $u_N = \Psi(u_n)$. The map Ψ will be detailed in the following sections. We generally call *reconstructed solution* the N-solution obtained from the n-solution. It can be equal to the original N-solution of (2.2) or it can differ due to approximations introduced, for example, by the time discretization or the use of neural networks. It should be clear from the context whether it is the approximated solution or the exact one.

Equation (2.3) is solved discretizing the time axis with a constant timestep size, Δt, so $t_k = k\Delta t$, $k = 0, \dots, K$. The numerical solution is obtained through a discrete-time integration scheme, which is characterized by its difference operator, $\mathscr{L}_n$ [37]. For a linear multistep method with P steps (including also Forward Euler (FE) for $P = 1$), the difference operator is

$$\mathscr{L}_n(u_n^{k+1}, u_n^k, \dots, u_n^{k+1-P}, F_n, \Delta t) = -\sum_{p=0}^{P} \alpha_p u_n^{k+1-p} + \Delta t \sum_{p=0}^{P} \beta_p F_n(u_n^{k+1-p}), \quad k = P-1, P, \dots \tag{2.4}$$

where α_p and β_p are coefficients of the time integration scheme and u_n^k is the solution at time step k. The maps Ψ', Ψ, F_n will be approximated by neural networks $N_{\Psi'}$, N_Ψ, N_{F_n}, respectively. This leads to our reduced-order modeling strategy that consists of finding an approximation of (2.2) by solving the discrete problem $\mathfrak{P}(\mathfrak{u})$

$$\mathfrak{P}(\mathfrak{u}) = \begin{cases} \mathfrak{u}_N^{k+1} = N_\Psi(\mathfrak{u}_n^{k+1}), \\ \mathscr{L}_n(\mathfrak{u}_n^{k+1}, \mathfrak{u}_n^k, \dots, \mathfrak{u}_n^{k+1-P}, N_{F_n}, \Delta t) = 0, \quad k = P-1, P, \dots \\ \mathfrak{u}_n^0 = N_{\Psi'}(u_N^0), \end{cases} \tag{2.5}$$

where $\mathfrak{u} = (\mathfrak{u}_N, \mathfrak{u}_n)$ and $\mathfrak{u}_n$, $\mathfrak{u}_N$ represent, respectively, the discrete n-solution and the reconstructed one. We call the strategy of solving (2.2) through (2.5) DRE.

It is now important to study the properties of the proposed method. We are particularly interested in identifying the conditions under which $\mathfrak{u}_N^k \to u_N(t_k)$, in understanding how to construct and train $N_{\Psi'}$, N_Ψ, and N_{F_n} with the highest possible accuracy, and in analyzing the stability properties of $\mathfrak{P}$.

Detailed outline. For the reader's convenience, we summarize in this section what we consider to be the most relevant topics of this work. We do not focus here on precise notation or detailed assumptions and definitions of the framework; these will be introduced in the subsequent sections. Instead, given the large number of topics covered in this work, we want to informally provide the main outline and key issues addressed in the order of exposition in the text to facilitate the readability of this work.

- A linear–nonlinear autoencoder can reproduce with null representation error manifolds described by $f = f_V + f_{V^\perp}$, where $f_{V^\perp} \perp f_V$ (**Proposition 4**). This proposition justifies the use of a linear encoder.
  However, for a generic manifold, a nonlinear encoder is required and the reduction must be performed at the last layer of the encoder to ensure a null representation error (**Theorem 2**).

- **Theorem 3.** *a-manifold.* Under the assumptions of well-posedness of the N-ODE, continuity of F_N and u_{N0} with respect to μ, and local injectivity of the map $\mu \mapsto u_N$, the set $\{u_N(t;\mu)\}_{t\in[0,T],\ \mu\in\mathcal{P}}$ is an a-manifold. Moreover, the minimum latent dimension is $n \leq 2a + 1$ (**Theorem 4**).
- **Theorem 5.** *Well-posedness of the reduced problem.* Under proper continuity assumptions on the autoencoder, the reduced problem is well-posed.
- We present two training strategies for N_{F_n}: one offers the possibility to increase the accuracy of the reduced-order model by setting $\Delta t < \Delta t_{\text{train}}$, the other allows for a trivial parallelization in time of the training.
- **Theorem 6.** *Global convergence.* Assuming that enough training data are available, that the global minium of the loss function is reached, that convergent discretization schemes for both the N-ODE and the n-ODE are used, and that proper neural network architectures are used, we have the following bound

$$\max_{\mu} \|\underline{u}_N - \underline{\mathfrak{u}}_N\|_\infty \leq L_\Psi(\varepsilon_{\Psi'}(w) + \varepsilon_{F_n}(\Delta t_{\text{train}}, w)T)e^{L_{F_n}T} + L_\Psi C_1 \Delta t^P + \varepsilon_\Psi(w),$$

so, for $|w| \to \infty$ and $\Delta t, \Delta t_{\text{train}} \to 0$, we have $\max_\mu \|\underline{u}_N - \underline{\mathfrak{u}}_N\|_\infty \to 0$. Here, w is the total number of trainable weights, $\varepsilon_{\Psi'}$, ε_Ψ, ε_{F_n} are approximation errors, L_Ψ and L_{F_n} are Lipschitz constants, and C_1 a positive constant.

3. Autoencoder-based dimensionality reduction

In this section, we present the mathematical foundations for using autoencoders for nonlinear model reduction. Our focus is on understanding how the characteristics of the data manifold impose constraints on the design of the autoencoder architecture. To this end, we introduce the notions of representation error and approximation error, which allow us to distinguish between inaccuracies due to architectural limitations and those arising from the training of neural networks. Through concrete examples, we explore different encoder–decoder configurations and analyze the classes of manifolds they are capable of reconstructing exactly. These insights are essential to ensure that the autoencoder accurately captures the structure of the solution manifold, which is an important prerequisite for constructing a reliable reduced order method in the subsequent analysis.

3.1. Neural networks

The main goal of this section is to fix the notation. Taking into account a layer $\ell \in 1, \dots, L$, we denote by n_ℓ its size, by $y^{(\ell)} \in \mathbb{R}^{n_\ell}$ its output, by $W^{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ and $b^{(\ell)} \in \mathbb{R}^{n_\ell}$ its trainable weights and biases, and by $\sigma^{(\ell)}$ its nonlinear activation function, except for the last one, which is set to be the identity, $\sigma^{(L)} = \text{Id}$. The activation functions are possibly parameterized by $w_\sigma^{(\ell)}$. Throughout the paper, we assume the use of monotonic activation functions. Each layer is constituted by an affine transformation followed by a nonlinear activation:

$$y^{(\ell)} = \sigma^{(\ell)}\left(W^{(\ell)} y^{(\ell-1)} + b^{(\ell)}\right). \tag{3.1}$$

To simplify notation, we define the layer-wise transformation $z^{(\ell)} : \mathbb{R}^{n_{\ell-1}} \to \mathbb{R}^{n_\ell}$ by:

$$z^{(\ell)}(y^{(\ell-1)}) := \sigma^{(\ell)}\left(W^{(\ell)} y^{(\ell-1)} + b^{(\ell)}\right), \tag{3.2}$$

so that Eq (3.1) becomes $y^{(\ell)} = z^{(\ell)}(y^{(\ell-1)})$. The feed-forward fully connected neural network is a composition of L layers from $\mathbb{R}^{n_{\text{in}}}$ to $\mathbb{R}^{n_{\text{out}}}$, where $n_{\text{in}} = n_0$ and $n_{\text{out}} = n_L$:

$$y^{(L)} = z^{(L)} \circ z^{(L-1)} \circ \cdots \circ z^{(1)}(y^{(0)}), \tag{3.3}$$

where $y^{(0)} \in \mathbb{R}^{n_{\text{in}}}$ is the input to the network. To simplify the notation, we define $w = \cup_\ell (W^{(\ell)}, b^{(\ell)}, w_\sigma^{(\ell)})$ as all trainable weights of a neural network, and we denote by $|w|$ the total number of trainable weights.

3.2. Dimensionality reduction

We consider a m-manifold, $\mathcal{M}$, of topological dimension m, in the ambient space $\mathbb{R}^N$. We now aim to relate the properties of $\mathcal{M}$ to the properties of the autoencoder. We call n the latent size, $\Psi' : \mathbb{R}^N \to \mathbb{R}^n$, $m \leq n < N$, the encoder, and $\Psi : \mathbb{R}^n \to \mathbb{R}^N$ the decoder. The autoencoder is defined as the composition $\Psi \circ \Psi' : \mathbb{R}^N \to \mathbb{R}^N$, such that $\Psi \circ \Psi' = \text{Id}_{\mathcal{M}}$, where $\text{Id}_{\mathcal{M}}$ represents the identity map on $\mathcal{M}$. This requires that $\Psi \circ \Psi'$ is bijective and Ψ' injective on $\mathcal{M}$, i.e., $\Psi' : \mathcal{M} \to \mathbb{R}^n$ is injective.

The maps Ψ' and Ψ will be approximated by neural networks (see Section 6), denoted by $N_{\Psi'}$ and N_Ψ, and still referred to as the encoder and decoder, respectively. The distinction between Ψ', Ψ and their neural network approximations $N_{\Psi'}$, N_Ψ should be clear from the context. $N_{\Psi'}$ and N_Ψ are defined as compositions of affine and nonlinear functions, as detailed in Section 3.1. We argue that such compositions may compromise the properties that the encoder and the decoder are required to satisfy: the former should be injective, while the latter should properly reproduce the possible nonlinearities of $\mathcal{M}$. Accordingly, we quantify the extent of the violation of these requirements by means of the following definitions:

Definition 1 (Encoder inexactness index, $\boldsymbol{\mathcal{E}_e}$)**.** Let $V = \{v \text{ s.t. } v = N_{\Psi'}(x_{N,1}) = N_{\Psi'}(x_{N,2}),\ x_{N,1}, x_{N,2} \in \mathcal{M},\ x_{N,1} \neq x_{N,2}\}$. Then, $\boldsymbol{\mathcal{E}_e} := |V|$, where $|V|$ is the cardinality of V.

Definition 2 (Decoder inexactness index, $\boldsymbol{\mathcal{E}_d}$)**.** Let $W = \{w \text{ s.t. } w = N_\Psi(x_n),\ x_n \in \mathbb{R}^n\}$. Then, $\boldsymbol{\mathcal{E}_d} := N - \dim(W)$.

We focus on the cases where $\boldsymbol{\mathcal{E}_e} = 0$ and $\boldsymbol{\mathcal{E}_d} = 0$, corresponding to cases in which the encoder and decoder fully satisfy their intended requirements. We note that, according to the definitions above, $\boldsymbol{\mathcal{E}_e}$ could become unbounded, however, in this work we are interested in the cases where it is null.

For practical convenience, we introduce the following definition to combine the two aforementioned requirements:

Definition 3 (Representation inexactness index, $\boldsymbol{\mathcal{E}_1}$)**.** We call the representation inexactness index the following quantity: $\boldsymbol{\mathcal{E}_1} = \boldsymbol{\mathcal{E}_e} + \boldsymbol{\mathcal{E}_d}$. This quantity represents an error which is intrinsically introduced by an improper choice of n_ℓ. This requirement is related to the topological properties of $\mathcal{M}$ and to the reduction/expansion at each step in the composition (3.3). We are particularly interested in the latent size n since, throughout the paper, we aim to achieve the maximum compression of the data x_N, i.e., to have n as close as possible to m.

The representation inexactness index is not the only source of error affecting the approximation of the identity, namely $N_\Psi \circ N_{\Psi'} \approx \Psi \circ \Psi' = \text{Id}_{\mathcal{M}}$. Accordingly, we introduce a second source of error. Let $\mathcal{A}$ be the search set for the trainable weights of the autoencoder, here considered as a single

neural network. The layers in $\mathcal{A}$ have size $(n_\ell^{\mathcal{A}})_{\ell=1}^L$. The training of the autoencoder can be cast as the following optimization problem

$$w_{\text{opt}} = \arg\min_{W \in \mathcal{A}} \mathscr{L}(w),$$

where $\mathscr{L}$ is the loss function. We denote by $\overline{N}_{\Psi'}$ the network associated with w_{opt} obtained from the optimization over $\overline{\mathcal{A}}$, which is constructed such that the layers have sizes $(\underbrace{n_\ell^{\mathcal{A}}, \dots, n_\ell^{\mathcal{A}}}_{M \text{ times}})_{\ell=1}^L$ in the limit of $M \to \infty$.

Definition 4 (Approximation error, ε_2)**.** This source of error is specifically related to the use of neural networks and their associated training procedure. We define it as $\varepsilon_2 \coloneqq \|\overline{N}_\Psi \circ \overline{N}_{\Psi'} - N_\Psi \circ N_{\Psi'}\|_{L^\infty(\mathcal{M})}$. It arises because $\mathcal{A}$ may not be sufficiently large, for instance due to a finite number of layers and neurons, and because the optimization procedure may not reach the global minimum, so that $w_{\text{train}} \neq w_{\text{opt}}$, where w_{train} denotes the weights obtained through training.

Note that $\overline{N}_\Psi \circ \overline{N}_{\Psi'}$ may not represent the identity on the manifold due to the error associated to a possible $\varepsilon_1 > 0$. Therefore, $\varepsilon_2 = 0 \nRightarrow N_\Psi \circ N_{\Psi'} = \text{Id}_{\mathcal{M}}$. It follows that it is relevant to study the effects of the functional composition in (3.3), i.e., to study ε_1, even before considering the approximation error, ε_2. Note that this study is independent of the specific activation function used. In particular, we aim to have $\varepsilon_1 = 0$, so that the only source of error in the approximation of $\text{Id}_{\mathcal{M}}$ is due to the practical and unavoidable limitations arising from the use of neural networks, represented by ε_2. In the following, after presenting two preliminary lemmas, we analyze three distinct scenarios with respect to types of manifolds and autoencoders. For clarity of exposition, we first make assumptions about the autoencoder that we subsequently use and then, from these assumptions, we then derive the properties of the underlying manifold for which, for the given autoencoder, we have $\Psi \circ \Psi' = \text{Id}_{\mathcal{M}}$.

Proposition 1 (Image of a linear decoder)**.** Let $\Psi : \mathbb{R}^n \to \mathbb{R}^N$ be a linear decoder with $n < N$, and let $J_\Psi \in \mathbb{R}^{N \times n}$ denote its Jacobian. Then, $\Psi(\mathbb{R}^n) = \text{col}(J_\Psi)$, where $\text{col}(J_\Psi) \subset \mathbb{R}^N$ denotes the column space of J_Ψ. In particular, for any subset $\mathcal{S} \subset \mathbb{R}^n$, the reconstructed set $\Psi(\mathcal{S})$ is contained in the linear subspace $\text{col}(J_\Psi)$ of $\mathbb{R}^N$.

Note that, to avoid any ambiguity in the notation, whenever a linear decoder is considered, its application to a vector x_n is expressed in terms of its Jacobian matrix. Similarly for the encoder. In the following, let $V \subset \mathbb{R}^N$ with $\dim(V) = n$ be a linear subspace.

Proposition 2 (Injectivity of linear encoder)**.** A linear encoder can only be injective on manifolds for which there exist both a linear subspace V and a linear transformation $T : \mathcal{M} \to V$ such that T is injective. Indeed, by the arbitrariness of the encoder, we can always define an encoder that is injective on a linear subspace of $\mathbb{R}^N$, and in particular V. It follows that, given the existence of a linear injection to V, the encoder is also injective to $\mathcal{M}$.

3.2.1. Linear encoder – Linear decoder

We assume here to have a linear encoder and a linear decoder and we analyze, also with a numerical example, the type of manifold that is possible to reconstruct fulfilling the requirement $\varepsilon_1 = 0$.

Proposition 3 (Exactness of autoencoders with a linear encoder and/or a linear decoder)**.** An autoencoder with a linear encoder and/or a linear decoder can reproduce with $\mathcal{E}_1 = 0$ only linear manifolds.

Proposition 3 is a direct consequence of Propositions 1 and 2. In fact, the limiting condition is that the decoder is only capable of reproducing linear manifolds, $\mathcal{M} \in V$, while the encoder can be applied to the linear $\mathcal{M}$ preserving the injectivity.

Note that the previous propositions also hold for affine encoders, but, for the sake of simplicity, we treat only the linear case. We present now different cases to provide constructive insight into the conditions under which an autoencoder can represent a manifold without loss, that is, when $\mathcal{E}_1$ vanishes. By analyzing specific configurations of the encoder and decoder, we clarify the limit of the expressivity of various architectural choices. See Figure 1 for an example. In this section, we intentionally show simple examples such as Figures 1–3, to enable a graphical interpretation of the topic.

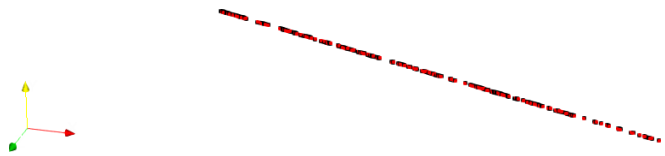


Figure 1. Example of a 1-manifold in $\mathbb{R}^3$ that can be reproduced exactly with linear transformations. Black point (overlapped by red points): original manifold. Red points: output of autoencoder. Here, $n = 1$.

3.2.2. Linear encoder – Nonlinear decoder

Here, we consider a linear encoder and a nonlinear decoder, with the same objective as before: having $\mathcal{E}_1 = 0$. Let $f : \mathbb{R}^n \to \mathcal{M}$ be the map that describes the manifold. Given a linear space $V \subset \mathcal{M}$, we denote by f_V and $f_{V^\perp}$ the decomposition of f into its components in V and in its orthogonal complement, respectively.

Proposition 4 (Exactness of autoencoders with linear encoder and nonlinear decoder)**.** A linear–nonlinear autoencoder can reproduce a manifold $\mathcal{M}$ with $\mathcal{E}_1 = 0$ if there exist a linear space V and a map f_V such that the manifold is described by $f = f_V + f_{V^\perp}$ with f_V injective.

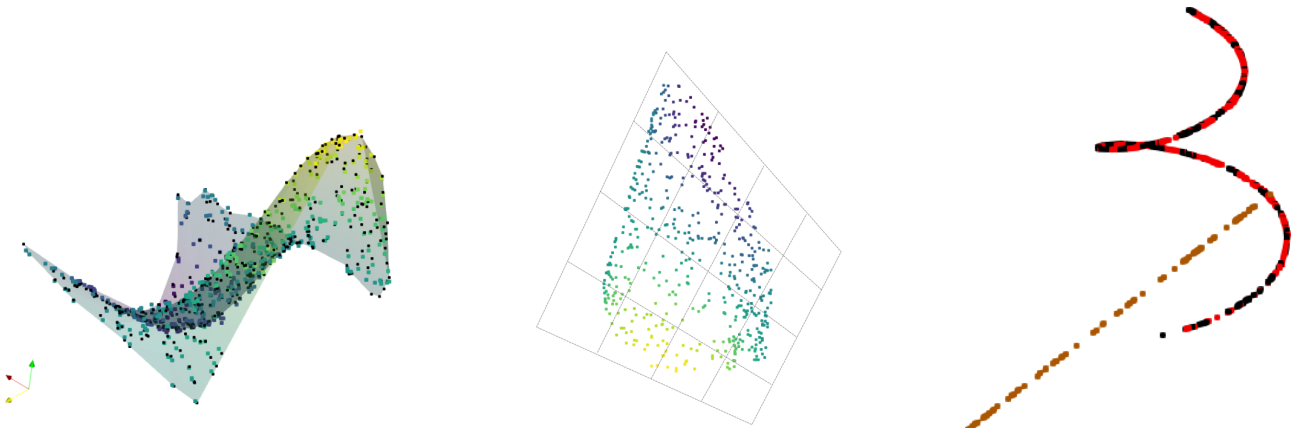
This is a direct consequence of Proposition 2: The linear encoder performs an injective projection onto a linear subspace, while the nonlinearity of the decoder is a necessary condition to reconstruct the lost orthogonal contribution, $f_{V^\perp}$.

In the following examples, we choose f_V to be a linear map and thus consider manifolds of the form $x_N = Ax_n + g(x_n)$, with $A \in \mathbb{R}^{N\times n}$ full rank and $g(x_n) \perp Ax_n$. Despite its apparent simplicity, the latter expression can describe non-trivial geometries, see Figure 2

Examples are represented by the surface of a function or a parametric curve, Figure 2. In the latter, the line is a contracting coil described by $(x, y, z) = (r\cos(\theta), r\sin(\theta), 0.2\theta)$, where $r = \frac{2\pi}{2\pi+\theta}$, for which a possible representation is $A = (0, 0, 0.2)^T$ and $g = (r\cos(\theta), r\sin(\theta), 0)^T$. For practical simplicity, in these numerical examples, we set $\Psi' = W^{(1)}$ (see (3.1)) and Ψ is approximated by neural networks with

a finite and relatively small number of layers, trained using the common gradient-based method (as will also see in Section 6.1) resulting in nonzero, but small, values of ε_2. Consequently, in Figure 2, the data points and the autoencoder output do not exactly coincide.

Incidentally, by constructing the decoder as $\Psi = \psi \circ \phi$, $\phi : \mathbb{R}^n \to \mathbb{R}^N$ a linear function, $\psi : \mathbb{R}^N \to \mathbb{R}^N$ a nonlinear function, we can flatten the curved manifold, obtaining $\mathcal{M}_{\text{flat}} = \phi(\Psi'(\mathcal{M}))$, see Figure 2.



(a) Graph of function. $m = 2$, $n = 2$.

(b) Coil. $m = 1$, $n = 1$.

Figure 2. **(a)** Manifold defined as graph of a function. Left panel, black dots: encoder inputs; colored dots: decoder output. The shaded surface is reconstructed with a Delaunay triangulation. The color scale is related to the z-coordinate and it is used just to ease the graphical interpretation. Right panel: flat surface obtained from $\phi \circ \Psi'$. **(b)** Coil geometry. Black dots: encoder input. Red dots: decoder output. Ochre line: output of $\phi \circ \Psi'$.

3.2.3. Nonlinear encoder – Nonlinear decoder

We consider here a nonlinear encoder combined with a nonlinear decoder. The nonlinearities in the encoder allow us to represent more complex manifolds. As already mentioned, in our approach, the encoder will be implemented as a neural network that combines activation functions with affine transformations.

By the universal approximation theorem, see, e.g., [27, 29, 42, 45], we see that, under appropriate continuity hypotheses, fully-connected neural networks are capable of approximating functions from $\mathbb{R}^N$ to $\mathbb{R}^n$, such as Ψ'.

For the reader's convenience, we summarize here the main result from [29] (Theorem 1), which serves both as a proof of the universal approximation property and as an upper bound on the width size:

Theorem 1 (Upper bound for width [29])**.** *Let $w_{\min}(n_{in}, n_{out})$ be the minimal value of the width w such that for every continuous function $f\colon [0,1]^{n_{in}} \to \mathbb{R}^{n_{out}}$ and every $\varepsilon > 0$ there is a ReLU neural network N_f with input dimension n_{in}, hidden layer widths at most w, and output dimension n_{out} that ε-approximates f:*

$$\sup_{x \in [0,1]^{n_{in}}} \|f(x) - N_f(x)\| \leq \varepsilon.$$

Then, for every $n_{in}, n_{out} \geq 1$, we have $n_{in} + 1 \leq w_{\min}(n_{in}, n_{out}) \leq n_{in} + n_{out}$, proving that $w_{\min}(n_{in}, n_{out}) \leq n_{in} + n_{out}$.

This theorem leads to the following proposition about fully nonlinear autoencoders.

Proposition 5. A nonlinear–nonlinear autoencoder can reproduce with $\mathcal{E}_1 = 0$ generic manifolds**.**

However, dimensionality reduction is performed layer by layer via a sequence of affine maps, which leads to the following theorem:

Theorem 2 (Sufficient condition for $\mathcal{E}_1 = 0$)**.** *We assume $\mathcal{M}$ to be a compact m-manifold, and n and L to be sufficiently large, with $m \leq n < N$. A sufficient condition to ensure the existence of an injective encoder is that the width of the encoder layers satisfies $n_{in} \leq n_\ell \leq n_{in} + n_{out}$, for $\ell = 1, \ldots, L-1$.*

Proof. To ensure the existence of an injective encoder approximated by a neural network, we need to ensure the existence of injective layers, condition which can only be satisfied in general when $n_\ell \geq n_{\ell-1}$ for all hidden layers, so $n_\ell \geq n_{\text{in}}$ for $\ell = 1, \ldots, L-1$. By Theorem 1, we have that $n_\ell = n_{\text{in}} + n_{\text{out}}$ is sufficient to satisfy $\mathcal{E}_1 = 0$. □

Additionally, under the above conditions, in particular $n_{\text{in}} \leq n_\ell \leq n_{\text{in}} + n_{\text{out}}$, for $\ell = 1, \ldots, L-1$, and assuming ideal training procedure, we have that $\lim_{|W| \to \infty} \|\Psi' - N_{\Psi'}\| = 0$. This result also provides practical insights on the construction of the encoder.

Therefore, the encoder can be written as: $\Psi' = \phi' \circ \psi'$, where $\phi' : \mathbb{R}^M \to \mathbb{R}^n$ is a linear function, and $\psi' : \mathbb{R}^N \to \mathbb{R}^M$, $M \geq N$, a nonlinear function. As a result, the role of ψ' is that of reshaping the manifold to ensure that the linear reduction in the last layer is injective. Similar conclusions can be drawn for the decoder: due to representation error and accuracy considerations, the expansion should not be performed solely in the last layer, as this approach would limit the network to representing an affine space to $\mathbb{R}^{n_{L-1}}$.

Remark 1. The message of Theorem 2 is that, without knowledge of the specific problem at hand, it is generally inappropriate to design a fully connected encoder with layers whose sizes reduce progressively (similarly for the decoder). Indeed, a reduction in dimension at an intermediate layer may compromise the injectivity of the encoder.

In the following examples and in the numerical cases in Section 7, we use either PReLU or ELU activation functions. Empirically, we notice that $M = N$ is sufficient to have $\mathcal{E}_1 = 0$ for the considered cases.

Figure 3 shows a perturbed coil with an additional oscillation along its length. It is an example of a manifold for which a nonlinear encoder is necessary. In the same figure, the output of ψ' is represented in green. We see that the coil has been unwound, and can now be projected injectively onto a straight line, since in this case $n = 1$.

Remark 2. Dimensionality reduction and similarity with POD-DL-ROM [21]: The reduction procedure presented in [21] relies on a first linear reduction based on the singular value decomposition of the snapshot matrix, followed by a non linear reduction made with an encoder. Similarly for the reconstruction step. The overall compression map can be interpreted as a single $N_{\Psi'}$, where $\sigma^{(1)} = \text{Id}$, the identity map, $b^{(1)} = 0$ and $W^{(1)}$, a rectangular matrix, computed from the singular value decomposition. For practical simplicity, in our method we just use neural networks in the conventional sense where $W^{(1)}$ (as all the other weights) is computed by an optimization procedure as it will be explained in Section 6.1.

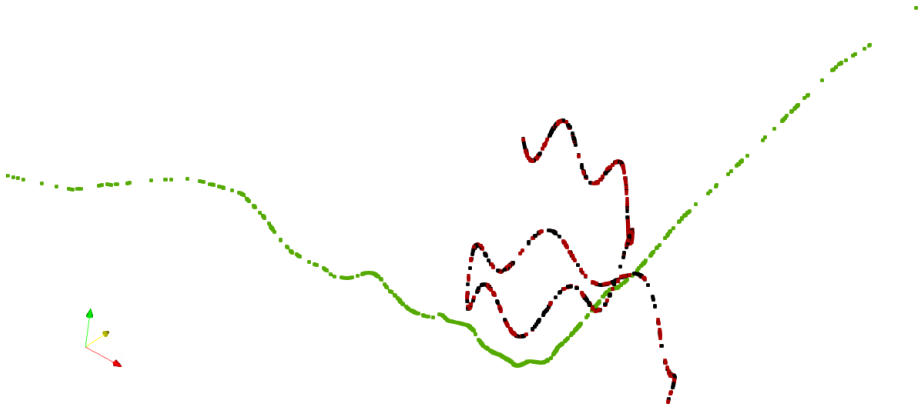


Figure 3. Noisy coil. A fully nonlinear autoencoder in necessary to reproduce $\mathrm{Id}_{\mathcal{M}}$ with $\varepsilon_1 = 0$. Black point: original manifold, red points: its reconstruction with $\Psi \circ \Psi'$, green points: output of ψ'. Here, $m = 1$ and $n = 1$.

4. ODE dimensionality reduction

In the following, our aim is to characterize the set of solutions of the N-ODE, and the possibility of describing it with a limited number of coordinates. We define U as the subset of the solution set in a neighborhood of given μ^* and t^*: $U = \{u_N(t;\mu)\}_{B_r((t^*,\mu^*))}$, where $B_r(\cdot)$ is the closed ball of radius r.

Theorem 3 (a-manifold)**.** *We assume that F_N and u_{N0} depend continuously on μ and* (2.1) *is well-posed for each $\mu \in \mathcal{P}$, and the map $B_r((t^*,\mu^*)) \ni (t,\mu) \to u_N(t;\mu)$ is injective $\forall (t^*,\mu^*) \in [0,T] \times \mathcal{P}$, with r small enough. Then, the set $\{u_N(t;\mu)\}_{t\in[0,T],\mu\in\mathcal{P}}$ is an a-manifold.*

Proof. First, we note that the set $\{u_N(t;\mu)\}_{t\in[0,T],\mu\in\mathcal{P}}$ is a subset of a second countable Hausdorff space. Then, by the well-posedness of (2.1) and by the continuity of F_N and u_{N0} in μ, we see that $(t,\mu) \to u_N(t;\mu)$ is continuous. By the continuity and by the compactness of $\mathcal{P}$, we have that U is compact and $B_r((t,\mu)) \to U$ has a continuous inverse, thus $B_r((t,\mu)) \to U$ is a local homeomorphism. We conclude that U is locally Euclidean, so $\{u_N(t;\mu)\}_{t\in[0,T],\mu\in\mathcal{P}}$ is an a-manifold, that we denote by $\mathcal{M}$. □

It follows that $\mathcal{M}$ can be described by a small number of coordinates, $n < N$, as presented in Section 3.2. With the following theorem, we aim to characterize the minimum value of n.

Theorem 4 (Latent dimension)**.** *Under the hypothesis of Theorem 3, the minimum latent dimension is $n \le 2a + 1$.*

Proof. According to Menger-Nöbeling embedding theorem (see, e.g., [49]), any compact topological space of topological dimension a admits a topological embedding in $\mathbb{R}^{2a+1}$. Hence, there exists a continuous injective map $\Psi' : \mathcal{M} \to \mathbb{R}^{2a+1}$. □

This result provides an upper bound on the minimal latent dimension required to represent the manifold $\mathcal{M}$ without loss of information, and justifies the use of autoencoder architectures with a reduced dimension, n, not greater than $2a+1$ for the exact encoding of $\mathcal{M}$. We note that similar results, obtained under a different setting, can be found in [17].

Observation 1. Throughout the paper, our aim is to describe the map $\mathcal{M} \to \mathbb{R}^n$ using a *single* function Ψ', and similarly for the inverse mapping $\mathbb{R}^n \to \mathcal{M}$. Although an atlas-based approach is possible, see [15], the use of single global maps offers the advantage of requiring the approximation of only one encoder and one decoder. This comes with the negligible drawback of a potentially larger reduced dimension, at most equal to $2a + 1$.

Observation 2. Global injectivity can be obtained by augmenting the solution set with parameters and time, i.e., considering the set $\{(u_N(t;\mu), t, \mu)\}_{t\in[0,T],\,\mu\in\mathcal{P}}$; see [17] and [38, Example 3.5].

By the injectivity and continuity of $(t,\mu) \to (u_N(t;\mu), t, \mu)$, and by the compactness of $\mathcal{P}$, follows that there exist a global homeomorphism to $\mathbb{R}^a$. The minimum number of coordinates required to describe $\mathcal{M}$ is the same as that of $[0,T] \times \mathcal{P}$ (see [17]*), which is equal to a. Thus, we can achieve the lowest latent size $n = a$.

Remark 3. In this paper, we assume a setting in which the number of parameters is much smaller than the size of N-ODE (which may arise from the spatial discretization of a PDE, so N is expected to be very large), i.e., $a \ll N$. The upper bound for n, namely $n \leq 2a + 1$, also satisfies the property of being much smaller than N, resulting in an effective reduction.

4.1. ODE of size n

We now aim to utilize the information originating from N-ODE to better describe $\mathcal{M}$. To this end, we recall the definition of the n-solution $u_n = \Psi'(u_N)$, with $\Psi' \in C^1$ as an encoder. Its dynamic is determined by F_N and the autoencoder:

$$\begin{aligned} \dot{u}_n &= \frac{\partial \Psi'(u_N)}{\partial t} = \frac{\partial \Psi'}{\partial u_N}\frac{\partial u_N}{\partial t} = J_{\Psi'}\frac{\partial u_N}{\partial t} \\ &= J_{\Psi'}(u_N)F_N(u_N) = J_{\Psi'}(\Psi(u_n))F_N(\Psi(u_n)). \end{aligned} \tag{4.1}$$

We call $F_n(u_n) \coloneqq J_{\Psi'}(\Psi(u_n))F_N(\Psi(u_n)) : \mathbb{R}^n \to \mathbb{R}^n$, and $u_{n0} \coloneqq \Psi'(u_{N0})$. Therefore, we have specified the right hand side, F_n, of the n-ODE (2.3), that we report here for the reader's convenience

$$\begin{aligned} \dot{u}_n &= F_n(u_n), \\ u_{n0} &= \Psi'(u_{N0}). \end{aligned}$$

At this stage, we have linked the original N-ODE to the smaller n-ODE. This becomes useful in the context of multi-query scenarios where it is necessary to find $u_N(t)$ for many different values of μ. Instead of solving the N-ODE, where N may be very large and it may derive from a discretization of a PDE, it may be convenient to learn a parameterization of $\mathcal{M}$, solve the n-ODE, where n can be much smaller than N, and then retrieve the N-solution as $u_N = \Psi(u_n)$. We will see in Section 6.1 how to efficiently achieve this task from a practical standpoint. We now aim to better characterize u_N in terms of u_n. However, before proceeding with the analysis, it is necessary to compute some useful preliminary quantities.

*Note that in [17, Theorem 1], Hilbert spaces are considered, although the result also holds for more general settings.

4.2. Preliminary properties

In this section, we collect a set of auxiliary results that will be instrumental in the theoretical analysis of the proposed reduction method and its numerical discretization. These properties concern both geometric and functional aspects of the encoding and decoding maps, and provide the foundation for the error estimates and approximation results presented in the subsequent sections. For clarity and completeness, we state them independently before incorporating them into the main convergence analysis.

Proposition 6 (Lipschitz constant of F_n)**.** We assume that Ψ, Ψ' and F_N are Lipschitz continuous with Lipschitz constants L_Ψ, $L_{\Psi'}$ and L_{F_N}, respectively. Furthermore, we assume that the encoder is differentiable and that its Jacobian, denoted by $J_{\Psi'}$, has a Lipschitz constant $L_{J_{\Psi'}}$ and is bounded with $M_{J_{\Psi'}} := \sup_{u_N \in \mathcal{M}} \|J_{\Psi'}(u_N)\|$. Additionally, we assume that F_N is bounded, with $M_{F_N} := \sup_{u_N \in \mathcal{M}} \|F_N(u_N)\|$. From the definition of F_n, we compute an upper bound, $\overline{L}_{F_n,\mu}$, for its Lipschitz constant L_{F_n} (see Appendix A for the detailed steps):

$$\begin{aligned}\|F_n(u_{n1}) - F_n(u_{n2})\| &= \|J_{\Psi'}(\Psi(u_{n1}))F_N(\Psi(u_{n1})) - J_{\Psi'}(\Psi(u_{n2}))F_N(\Psi(u_{n2}))\| \\ &\leq \underbrace{(M_{J_{\Psi'}} L_{F_N} + M_{F_N} L_{J_{\Psi'}})L_\Psi}_{=\overline{L}_{F_n,\mu}} \|u_{n1} - u_{n2}\|. \end{aligned} \tag{4.2}$$

Note that $\overline{L}_{F_n,\mu}$ depends on μ since the quantities related to F_N do. Whenever necessary, we take the maximum over the parameters: $\overline{L}_{F_n} = \max_\mu \overline{L}_{F_n,\mu}$, and we have $L_{F_n} \leq \overline{L}_{F_n}$. Lipschitz continuity ensures that the solution of the n-ODE exists and is unique [6, 57].

We anticipate here that Ψ, Ψ', and F_N will be approximated by neural networks. Similar conclusions to those of the above proposition can be drawn for neural networks. In this case, it is also possible to relate upper bounds of the Lipschitz constants to the architecture of the network, see [55].

Proposition 7 (Equivalence of autoencoders)**.** Let $g : \mathbb{R}^n \to \mathbb{R}^n$ be an invertible function. The composition $\Psi \circ g^{-1} \circ g \circ \Psi'$ describes the same autoencoder $\Psi \circ \Psi'$ but allows us to consider different n-solutions. Indeed, let $u_n^g = g(u_n)$, $\Psi'^g = g \circ \Psi'$, and $\Psi^g = \Psi \circ g$, we have that $u_n \neq u_n^g$ while $\Psi \circ \Psi'$ and $\Psi^g \circ \Psi'^g$ represents the same autoencoder in the sense that they both coincide with $\mathrm{Id}_\mathcal{M}$.

Proposition 8 (Lipschitz constants of scaled problem)**.** Let us consider the scaled mapping $g(u_n) = K_s u_n$ with $K_s \in \mathbb{R}^+$, which represents a linear scaling of the n-solution. We now proceed to compute the Lipschitz constants of the scaled encoder and decoder, defined respectively as $\tilde{\Psi}' = g \circ \Psi'$ and $\tilde{\Psi} = \Psi \circ g^{-1}$.

$$\begin{aligned}\|\tilde{\Psi}(u_{n1}) - \tilde{\Psi}(u_{n2})\| &= \|\Psi(g^{-1}(u_{n1})) - \Psi(g^{-1}(u_{n2}))\| \leq L_\Psi \|g^{-1}(u_{n1}) - g^{-1}(u_{n2})\| = 1/K_s L_\Psi \|u_{n1} - u_{n2}\|, \\ \|\tilde{\Psi}'(u_{N1}) - \tilde{\Psi}'(u_{N2})\| &= \|g(\Psi'(u_{N1})) - g(\Psi'(u_{N2}))\| = K_s \|\Psi'(u_{N1}) - \Psi'(u_{N2})\| \leq K_s L_{\Psi'} \|u_{N1} - u_{N2}\|,\end{aligned}$$

so $L_{\tilde{\Psi}'} = K_s L_{\Psi'}$ and $L_{\tilde{\Psi}} = 1/K_s L_\Psi$. For the right-hand side F_n, we have that the Lipschitz constant remains invariant under the scaling, i.e., $L_{\tilde{F}_n} = L_{F_n}$ (see Appendix A for the computation). In order to modify L_{F_n}, a nonlinear g is required.

Proposition 9 (Initial conditions for u_n)**.** Let $g_\mu(u_n) = u_n - u_{n,0}$ be a μ-dependent function. The use of g_μ allow us to modify the initial conditions. In particular, defining $\overline{u}_n := g_\mu(u_n) = u_n - u_{n,0}$, we obtain

the following n-ODE:

$$\dot{\overline{u}}_n = F_n(\overline{u}_n + u_{n,0}) =: \overline{F}_n(\overline{u}_n),$$
$$\overline{u}_n(0) = 0.$$

Then, u_N is the retrieved by re-adding the initial condition: $u_N = \Psi(\overline{u}_n + u_{n,0})$. It is therefore possible to set the initial condition to zero for any value of the parameters, assuming the knowledge of (or the possibility to approximate) g_μ. Otherwise, the initial conditions are computed with the encoder: $u_{n,0} = \Psi'(u_{N,0})$.

4.3. Stability after exact reduction

We aim to establish the Lyapunov stability, i.e., an estimate for the discrepancy between the exact solution and that derived from a perturbed ODE. To emphasize the impacts of the reduction, we perturb both N-ODE and n-ODE. Subsequently, we calculate the error between the exact solution and the reconstructed u_N. Additionally, establishing stability in the Lyapunov sense, paired with Lipschitz continuity, ensures the well-posedness of the n-ODE [57]. Moreover, this study will be instrumental, in Section 6, in studying the perturbations associated with the approximation errors of neural networks.

Let us consider a single trajectory. Given ε_0, ε, γ_0, $\gamma > 0$, we define the perturbations as follows:

- N-ODE: $\Delta_0 \in \mathbb{R}^n$, and $\Delta(t) : (0, T] \to \mathbb{R}^n$, with $\|\Delta_0\| < \varepsilon_0$, and $\|\Delta(t)\| < \varepsilon, \forall t$;
- n-ODE: $\delta_0 \in \mathbb{R}^n$, and $\delta(t) : (0, T] \to \mathbb{R}^n$, with $\|\delta_0\| < \gamma_0$, and $\|\delta(t)\| < \gamma, \forall t$.

We call z_N the solution of the perturbed N-ODE:

$$\dot{z}_N = F_N(z_N) + \Delta,$$
$$z_N(0) = u_{N0} + \Delta_0.$$

In addition, we perturb n-ODE and we call w_n the solution of the perturbed n-ODE, obtained from the reduction of the perturbed N-ODE:

$$\dot{w}_n = F_n(w_n) + J_{\Psi'}\Delta + \delta,$$
$$w_n(0) = \Psi'(u_{N0} + \Delta_0) + \delta_0.$$

We now compute the discrepancy between the reconstructed perturbed solution $w_N = \Psi(w_n)$ and u_N. We obtain the following result, see Appendix A for the computations:

$$\|w_n(t) - u_n(t)\| \leq \Big(L_{\Psi'}\|\Delta_0\| + \|\delta_0\| + (M_{\Psi'}\|\Delta\| + \|\delta\|)t\Big)e^{L_{F_n,\mu}t}, \tag{4.3}$$

for the discrepancy of n-solutions, and:

$$\|w_N(t) - u_N(t)\| \leq L_\Psi\Big(L_{\Psi'}\|\Delta_0\| + \|\delta_0\| + (M_{\Psi'}\|\Delta\| + \|\delta\|)t\Big)e^{L_{F_n,\mu}t}, \tag{4.4}$$

for the discrepancy between the reconstructed and the original solution. Note that these bounds depend on μ because $L_{F_n,\mu}$ does. For a global bound, we can replace $L_{F_n,\mu}$ with L_{F_n}.

Theorem 5 (Well-posedness of reduced problem)**.** *Under the assumption of Proposition 6, the reduced problem*

$$\begin{cases} u_N = \Psi(u_n), \\ \dot{u}_n = F_n(u_n), \\ u_n(0) = \Psi'(u_{N0}), \end{cases}$$

is well-posed.

Proof. The proof follows from Proposition 6 and Lyapunov stability (4.4), which are sufficient requirements for well-posedness [57]. □

Scaling reduced problem for improved stability. We can exploit the scaling to reduce the effects of the perturbation of the n-ODE propagated to the reconstructed u_N. Rescaling the autoencoder with a factor K_s: $\tilde{\Psi}' := K_s\Psi'$, and $\tilde{\Psi}(u_n) := \Psi(1/K_s u_n)$, and as a consequence $\tilde{F}_n(u_n) = J_{\tilde{\Psi}'}F_N(\tilde{\Psi}(u_n))$, we easily obtain the following relations: $J_{\tilde{\Psi}'} = K_s J_{\Psi'}$, $M_{\tilde{\Psi}'} = K_s M_{\Psi'}$, $L_{\tilde{\Psi}'} = K_s L_{\Psi'}$, and $L_{\tilde{\Psi}} = 1/K_s L_{\Psi}$. It follows that

$$\|w_N(t) - u_N(t)\| \leq L_\Psi\Big(L_{\Psi'}\|\Delta_0\| + 1/K_s\|\delta_0\| + \big(M_{\Psi'}\|\Delta\| + 1/K_s\|\delta\|\big)t\Big)e^{L_{\tilde{F}_{n,\mu}}t}. \tag{4.5}$$

We first observe that the bound for the perturbation of the n-ODE is influenced by the decoder, as reflected in the terms $L_\Psi/K_s\|\delta_0\|$ and $L_\Psi/K_s\|\delta\|$. In general $L_{\tilde{F}_{n,\mu}} \neq L_{F_{n,\mu}}$ although it is possible to check that they have the same upper bound according to the computations in Appendix A; the exact values of the Lipschitz constants depend on the specific problem at hand and on the nonlinearities of F_N. Since $L_\Psi \geq 1$, the bound is increased by the presence of the decoder, and we can expect that the decoder amplifies the perturbation of the n-ODE. However, due to the presence of the scaling factor K_s, we conclude the following:

Proposition 10. Supposing that the maximum norm of the perturbation of the n-ODE is independent of the scaling K_s, and that F_n and $\tilde{F}_n$ share the same Lipschitz constant, the bound for stability decreases as K_s increases, see (4.5).

Example 1. SIR. We present now a numerical example where the scaling is practically effective. We consider here the SIR (susceptible, infected, recovered) model of infectious diseases:

$$\begin{cases} \dot{S} = -\frac{\beta}{N}IS, \\ \dot{I} = \frac{\beta}{N}IS - \gamma I, \\ \dot{R} = \gamma I, \end{cases} \tag{4.6}$$

where S represents the susceptible population, I infected people and R recovered people, N is the total number of people. The initial conditions are: $S(0) = 90$, $I(0) = 10$, $R(0) = 0$. The recovery rate, γ, and the infection rate, β, are considered uncertain and randomly sampled 200 times in the range $[0.5, 2.5]$ to create $\mathcal{M}$. The autoencoder is made of fully connected layers and is trained until the mean squared error loss reaches a low values of about 10^{-5}. The latter is computed on the test dataset, made of 50 samples. F_n is computed from (4.1), after the training of the autoencoder. The perturbations

are set to zero except for δ, which is a random value in time with a uniform distribution $\mathcal{U}(-0.2, 2)$. Figure 4 shows the relative ℓ_2-error in time, $\|w_N - u_N\|_2/\|u_{N0}\|_2$, for two solutions of the SIR, w_N and u_N, obtained with 2 samples of β in the test dataset. The solutions are obtained numerically with the Forward Euler scheme with a time step size small enough to make the error introduced by the numerical integration negligible compared to δ. As predicted by Proposition 10, we see that a scaling factor of $K_s = 2$ reduces the upper bounds and also the error in most of the computed time steps (yellow line). Note that this occurs because the perturbation is independent of the solution itself. Therefore, rescaling alters the ratio between the magnitude of the external perturbation and the magnitude of u_n. However, this condition may not hold in practical scenarios when, for instance, the perturbation arises from floating-point truncation errors, which are dependent on the magnitude of the solution. Additionally, we remark that the error bound is derived using Grönwall's lemma (see Appendix A), which is known to be non-sharp. As a consequence, the numerical value of the bound itself may not provide a reliable reference for practical use while it remains useful for highlighting the role of the different quantities in the stability analysis.

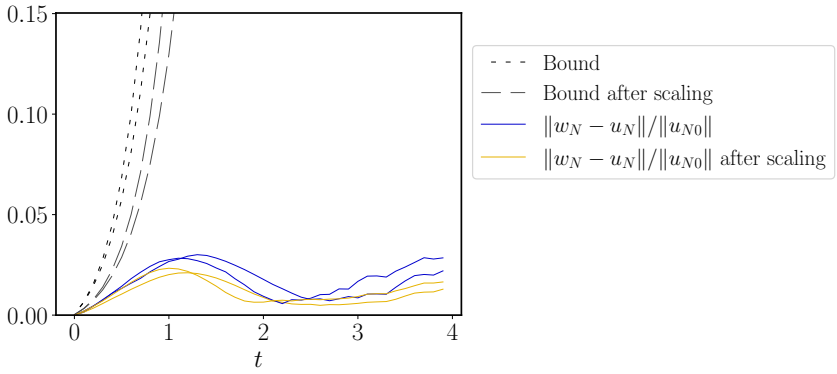


Figure 4. Lyapunov stability. Bounds (black lines) for the given μ and errors $\|w_N - u_N\|/\|u_{N0}\|$ (blue and yellow lines) in time.

5. Analysis of the time discretization error and stability

This section focuses on the time discretization, assuming to have access to a perfect autoencoder, i.e., one that is not subject to approximation errors introduced by neural networks. This latter source of approximation error will be addressed separately in Section 6. The most important topic is the convergence of the numerical scheme, which ensures the reliability of the computed solutions. To complement this analysis, we also investigate the consistency and stability of the numerical solution. For simplicity, we adopt the FE method as the reference scheme throughout most of the discussion.

5.1. Convergence in time for exact autoencoder

We consider a uniform time discretization, $t \in \{k\Delta t\}_{k=0}^{K}$, where K is the total number of timesteps, including the initial condition. We call $\mathfrak{u}_n$ the discrete-in-time solution of the n-ODE, and $\mathfrak{u}_N = \Psi(\mathfrak{u}_n)$ the corresponding reconstructed solution. We are interested in the error on the solution of N-ODE

reconstructed from the reduced one, i.e., $\|u_N - \mathfrak{u}_N\|$. The discrete-in-time problem, P, is:

$$\mathrm{P}(\mathfrak{u}) = \begin{cases} \mathfrak{u}_N^{k+1} = \Psi(\mathfrak{u}_n^{k+1}), \\ \mathscr{L}_n(\mathfrak{u}_n^{k+1}, \mathfrak{u}_n^k, \dots, \mathfrak{u}_n^{k+1-P}, F_n, \Delta t) = 0, \quad k = P-1, P, \dots \\ \mathfrak{u}_n^0 = \Psi(\mathfrak{u}_{N0}), \end{cases}$$

where $\mathfrak{u} = (\mathfrak{u}_N, \mathfrak{u}_n)$, and $\mathscr{L}_n$ is the difference operator (2.4). Our goal is to ensure that the error between the N-solutions vanishes as the time step tends to zero, i.e., $\|u_N - \mathfrak{u}_N\| \to 0$, as $\Delta t \to 0$. By the well-posedness of the n-ODE (Section 4.3) and assuming to use a convergent time integration scheme to solve the n-ODE, the convergence of the N-ODE is simply ensured by the continuity of the decoder: $\|u_N - \mathfrak{u}_N\| \le L_\Psi \|u_n - \mathfrak{u}_n\| \to 0$ as $\Delta t \to 0$ which holds for all the trajectories in $\mathcal{M}$.

5.2. Consistency

Although convergence has already been established, it remains interesting to investigate how the reduction process influences consistency to better understand the role of Ψ' and Ψ. A method is said to be consistent if, upon substituting the exact solution of the N-ODE into the discrete problem, denoted by P, the residual decays sufficiently rapidly. Precisely, the method is consistent if $\lim_{\Delta t \to 0} \max_k |\tau_N^k| = 0$ [37, 57], where τ_N^k is the local truncation error at timestep k for the N-ODE. In our setting, the consistency requirement can be reformulated in terms of the n-ODE: substituting $u = (u_N, u_n)$ into P reads

$$\mathrm{P}(u) = \begin{cases} u_N^{k+1} = \Psi(u_n^{k+1}), & \text{(5.1a)} \\ \mathscr{L}_n(u_n^{k+1}, u_n^k, \dots, u_n^{k+1-P}, F_n, \Delta t) - \Delta t\, \tau_n^k = 0, \quad k = P-1, P, \dots & \text{(5.1b)} \\ u_n^0 = \Psi'(u_{N0}). & \text{(5.1c)} \end{cases}$$

Here, to simplify the notation, we added a superscript to denote the timestep: $u_N^k = u_N(t_k)$. Equation (5.1a),(5.1c) is satisfied by construction. In (5.1b), $\Delta t \tau_n^k$ is the residual written in terms of the local truncation error of the n-ODE, τ_n. Equation (5.1) indicates that the consistency of the reconstructed solution depends on the consistency of the n-ODE. Consequently, using a consistent method for solving the n-ODE ensures the consistency of the N-ODE.

To have a better insight into how the reduction process influences consistency, we now examine a one-step method, such as the FE scheme. All such methods can be written with the following difference operator

$$\mathscr{L}_n(\mathfrak{u}_n^{k+1}, \mathfrak{u}_n^k, \dots, \mathfrak{u}_n^{k+1-P}, F_n, \Delta t) = \mathfrak{u}_n^{k+1} - \Delta t \Phi(\mathfrak{u}_n^k, F_n),$$

where $\Phi(\mathfrak{u}_n^k, F_n)$ is called *incremental function*. We proceed by substituting $u = (u_N, u_n)$ into P with $\mathscr{L}_n$ as previously defined:

$$\mathrm{P}(u) = \begin{cases} u_N^{k+1} = \Psi(u_n^{k+1}), \\ u_n^{k+1} - u_n^k - \Delta t \Phi(u_n^k) - \Delta t \tau_n^k = 0, \\ u_n^0 = \Psi'(u_{N0}). \end{cases} \tag{5.2}$$

Then, we compute a Taylor expansion centered in u_n^k to express the variation of u_n in a timestep: $u_n^{k+1} = u_n^k + \dot{u}_n \Delta t + \ddot{u}_n \Delta t^2/2 + O(\Delta t^3)$, where $\dot{u}_n = F_n$ and, assuming an encoder smooth enough,

$$\ddot{u}_{n,i} = F_{N,m} \frac{\partial^2 \Psi_i'(x)}{\partial x_m \partial x_j} F_{N,j} + J_{\Psi',ij} J_{F_N,jk} J_{\Psi,km} J_{\Psi',mz} F_{N,z},$$

where $F_{N,m} = F_N(\Psi(u_n))_m$ and similarly for $J_{\Psi',ij}$, $J_{\Psi,ij}$, $J_{F_N,ij}$. Replacing it into (5.2):

$$u_n^k + \dot{u}_n \Delta t + \ddot{u}_n \Delta t^2/2 + O(\Delta t^3) = u_n^k + \Delta t \Phi(u_n^k) + \Delta t \tau_n^k, \tag{5.3}$$

from which we obtain the local truncation error for the one-step method:

$$\tau_n^k = \dot{u}_n - \Phi(u_n^k; \Delta t) + \ddot{u}_n \Delta t/2 + O(\Delta t^2). \tag{5.4}$$

By the fact the $\lim_{\Delta t \to 0} \Phi(u_n; \Delta t) = F_n(u_n)$, we have that $\lim_{\Delta t \to 0} \max_k \tau_n^k = 0$, so consistency is confirmed. We write explicitly τ_n^k for FE:

$$\tau_n^k = \frac{1}{2} F_{N,m} \frac{\partial^2 \Psi_i'(x)}{\partial x_m \partial x_j} F_{N,j} \Delta t + \frac{1}{2} J_{\Psi',ij} J_{F_N,jk} J_{\Psi,km} J_{\Psi',mz} F_{N,z} \Delta t + O(\Delta t^2),$$

where we note that the encoder affects τ_n with both its second and first derivatives, while the decoder appears in terms of its Jacobian, as argument in F_N.

5.3. Zero stability

We aim to understand how the reduction, and therefore the autoencoder, impacts the zero stability of the time-marching method, focusing on one-step schemes, such as the FE scheme. Similarly to Section 4.3, we consider a perturbed problem with perturbed solution, w_n^k, at timestep k and bounded perturbations Δ, Δ_0, δ, δ_0:

$$\begin{aligned} \mathrm{w}_n^{k+1} &= \mathrm{w}_n^k + \Delta t \Phi(\mathrm{w}_n^k) + \Delta t J_{\Psi'} \Delta + \Delta t \delta, \\ \mathrm{w}_n^0 &= \Psi'(\mathfrak{u}_{N0} + \Delta_0) + \delta_0. \end{aligned}$$

For simplicity, we sum the perturbations contribution, so we group them in a unique $\overline{\delta}_0 = \Psi'(u_{N0} + \Delta_0) + \delta_0 - \Psi'(u_{N0})$, and $\overline{\delta} = J_{\Psi'}\Delta + \delta$, bounded by $\|\overline{\delta}_0\| < \varepsilon_0$, with $\varepsilon_0 > 0$, and $\|\overline{\delta}\| < \varepsilon$, with $\varepsilon > 0$. Thus, from the previous equation we obtain

$$\begin{aligned} \mathrm{w}_n^{k+1} &= \mathrm{w}_n^k + \Delta t \Phi(\mathrm{w}_n^k) + \Delta t \overline{\delta}, \\ \mathrm{w}_n^0 &= \Psi'(\mathfrak{u}_{N0}) + \overline{\delta}_0. \end{aligned}$$

Writing the solution at timestep $k+1$ in terms of the initial conditions, we have: $\mathfrak{u}_n^{k+1} = \mathfrak{u}_n^0 + \Delta t \sum_{i=0}^k \Phi(\mathfrak{u}_n^i)$ and $\mathrm{w}_n^{k+1} = \mathrm{w}_n^0 + \Delta t \sum_{i=0}^k \Phi(\mathrm{w}_n^i) + \Delta t \sum_{i=0}^k \overline{\delta}^i$ for the perturbed one. Using the triangular inequality and the discrete Grönwall's lemma (see e.g., [57]), the error for the n-ODE can be bounded as:

$$\|\mathrm{w}_n^{k+1} - \mathfrak{u}_n^{k+1}\| \leq \|\overline{\delta}_0\| + \Delta t \sum_{i=0}^k \|\Phi(\mathrm{w}_n^i) - \Phi(\mathfrak{u}_n^i)\| + \Delta t \sum_{i=0}^k \|\overline{\delta}^i\| \leq (\varepsilon_0 + K \Delta t \varepsilon)\, e^{(k+1)\Delta t L_\Phi},$$

where L_Φ is the Lipshitz constant of Φ. Using the Lipschitz continuity of Ψ, we have:

$$\|\mathrm{w}_N^{k+1} - \mathfrak{u}_N^{k+1}\| \leq L_\Psi\, (\varepsilon_0 + (k+1)\Delta t \varepsilon)\, e^{(k+1)\Delta t L_\Phi}.$$

The latter inequality says that the perturbation on the reconstructed solution is bounded. Moreover, the decoder appears in both L_Ψ and in L_Φ, while the encoder affects only L_Φ. Indeed, adopting for example FE, we have $L_\Phi = L_{F_{n,\mu}}$, where $L_{F_{n,\mu}}$ depends on both the encoder and decoder, see Eq (4.2).

5.4. Linear stability

We now turn to the study of whether the regions of absolute stability are preserved after reduction. We consider the following linear N-ODE, called *model problem*

$$\begin{aligned} \dot{u}_N &= \Lambda u_N, \quad t \in (0, \infty), \\ u_N(0) &= u_{N0}, \end{aligned} \tag{5.5}$$

where Λ is a square matrix whose eigenvalues have negative real part, thus, $u_N \to 0$ as $t \to \infty$. The goal is to determine under which conditions the numerical reconstructed solution of (5.5) goes to zero as well. After brief observations on the exact n-ODE, we study the properties of the time-integration method in relation to the reduction.

Step 1: reduction of model problem. The n-ODE associated to (5.5) is

$$\begin{aligned} \dot{u}_n &= J_{\Psi'} \Lambda \Psi(u_n), \\ u_n(0) &= \Psi'(u_{N0}). \end{aligned}$$

Following the three relevant combinations of encoder-decoder introduced in Section 3.2, we make some observations:

- Linear encoder, linear decoder. We express the encoder in terms of its constant Jacobian matrix $\Psi'(u_N) = J_{\Psi'} u_N$. Given that $u_n = J_{\Psi'} u_N$, the exact n-solution is $u_n(t) = J_{\Psi'} \exp(\Lambda t) u_{N0}$. We observe that the n-solution goes to zero as the original one. Due to the linearity of Ψ' and Ψ, the autoencoder defined in $[0, t_1]$, $t_1 > 0$, remains valid also for $(t_1, t_2]$, $\forall t_2 > t_1$, indeed, $J_{\Psi'}$ is a constant matrix. From a practical standpoint, this means that the autoencoder can be determined on a short interval of time and we can then extrapolate the solution for any future time.
- Linear encoder, nonlinear decoder. Due to the linearity of the encoder, we still have that $\lim_{t\to\infty} u_n = 0$. However, due to its nonlinearity, the decoder has to be determined over the entire manifold $\mathcal{M}$, and in particular for $t \in [0, \infty)$, in order for the autoencoder to reproduce $\mathrm{Id}_{\mathcal{M}}$ (assuming that $\mathcal{M}$ is defined for $t \in [0, \infty)$).
- Nonlinear encoder, nonlinear decoder. Similarly to the previous case, the autoencoder must be determined over the entire time interval of interest. Note that the nonlinearity of the encoder may allow u_n to converge to a non-zero stationary value as $t \to \infty$, despite the fact that $\lim_{t\to\infty} u_N = 0$.

Step 2: region of absolute stability for Forward Euler. We now investigate the effects of time discretization of the n-ODE propagated to the N-ODE, in particular, we want to establish under what conditions the numerical solution tends to zero towards infinite time[†]. For simplicity, we analyze the FE scheme. As in Step 1, the analysis proceeds studying separately three types of autoencoder associated to the different types of $\mathcal{M}$:

- Linear encoder, linear decoder. We express the decoder in terms of its Jacobian matrix $\Psi(u_n) = J_\Psi u_n$. By the linearity of the decoder, the reconstructed solution $J_\Psi u_n$, tends to zero if u_n does.

[†]Note that the stability results hold also for a matrix Λ with positive real part of the eigenvalues [28, 37], although in this paper we are not really interested in this scenario.

Therefore, it is sufficient to check the stability of the n-ODE. By calling $A = J_{\Psi'}\Lambda J_\Psi$, the numerical solution at timestep $k+1$ is:

$$\mathfrak{u}_n^{k+1} = (\mathbb{I} + \Delta t A)^k \mathfrak{u}_{n0},$$

which leads to the well-known upper bound on the timestep size: $\max_i |1 + \Delta t \lambda_i| < 1$, where λ_i is the i^{th} eigenvalue of A.
In the following, we show that the upper bound for the timestep for the n-ODE is at most equal to that of the N-ODE. The n-eigenproblem, $Av = \lambda v$, $v \in \mathbb{R}^n$, can be written as

$$J_\Psi A v = J_\Psi (J_{\Psi'} \Lambda J_\Psi v) = \lambda J_\Psi v,$$

and by calling $w \coloneqq J_\Psi v$, $w \in \mathcal{M}$,

$$(J_\Psi J_{\Psi'})\Lambda w = \lambda w.$$

We observe that the matrix $P = J_\Psi J_{\Psi'}$ is a projection, since

$$(J_\Psi J_{\Psi'})(J_\Psi J_{\Psi'}) = J_\Psi (J_{\Psi'} J_\Psi) J_{\Psi'} = J_\Psi J_{\Psi'}.$$

In general, a projection P may modify the eigenvalues of Λ, unless it is an orthogonal projection. Orthogonality here means that $(w - Pw) \perp Pw$, which is equivalent to $w^\top P^\top (P - I) w = 0$. It can be checked that the projection induced by the autoencoder, $P = J_\Psi J_{\Psi'}$, is an orthogonal projection for points on the manifold:

$$\begin{aligned} u_N^\top P^\top (P - I) u_N &= u_N^\top (J_\Psi J_{\Psi'})^\top (J_\Psi J_{\Psi'} - I) u_N \\ &= u_n^\top J_\Psi^\top (J_\Psi J_{\Psi'})^\top (J_\Psi J_{\Psi'} - I) J_\Psi u_n \\ &= u_n^\top \left((J_\Psi^\top J_{\Psi'}^\top)\, J_\Psi^\top J_\Psi\, (J_{\Psi'} J_\Psi) - (J_\Psi^\top J_{\Psi'}^\top)\, J_\Psi^\top J_\Psi \right) u_n \\ &= u_n^\top \left(I_{n\times n}\, J_\Psi^\top J_\Psi\, I_{n\times n} - \, I_{n\times n} J_\Psi^\top\, J_\Psi \right) u_n = 0. \end{aligned}$$

This result shows that the eigenvalues of A form a subset of those of Λ, which implies that the timestep bound for the n-ODE coincides, in the worst case, with that of the N-ODE.

- Linear encoder, nonlinear decoder. By the linearity of the encoder $\lim_{t\to\infty} u_N = 0 \Rightarrow \lim_{t\to\infty} u_n = 0$ and by the exact reduction we have that $\lim_{t\to\infty} \mathfrak{u}_n = 0 \Rightarrow \lim_{t\to\infty} \mathfrak{u}_N = 0$. So we only need to assure that $\lim_{t\to\infty} \mathfrak{u}_n = 0$. Unfortunately, the nonlinear decoder leads to a nonlinear n-ODE:

$$\mathfrak{u}_n^{k+1} = \mathfrak{u}_n^k + \Delta t J_{\Psi'} \Lambda \Psi(\mathfrak{u}_n^k),$$

which hinders the possibility of assessing the stability by analyzing the eigenvalues of the Jacobian of the right-hand side, J_{F_n} [37], due to the nonlinear nature of the r.h.s. We refer the reader to Section 5.5 for a more in-depth discussion. Nonetheless, a necessary condition for stability is that small perturbations around the equilibrium point, $u_n = 0$, do not lead to divergence. We set $u_n^k = 0$ and we consider a sufficiently small perturbations δu form the equilibrium point $u_n^k = 0$. Using a Taylor expansion, we have

$$u_n^{k+1} = \delta u_n + \Delta t J_{\Psi'} \Lambda \Psi(\delta u_n) = (\mathbb{I} + \Delta t J_{\Psi'} \Lambda J_\Psi(0)) \delta u_n + \mathcal{O}(\delta u_n^2),$$

from which, by setting $J_\Psi = J_\Psi(0)$, conclusions analogous to those of the previous case can be drawn.

- Nonlinear encoder, nonlinear decoder. Due to the presence of nonlinearities, $\lim_{t\to\infty} u_N = 0 \Rightarrow \lim_{t\to\infty} u_n = \alpha$, where α satisfies $\Psi(\alpha) = 0$. For simplicity, without loss of generality, we set $\alpha = 0$ (see Proposition 7). Hence, we still need to analyze the conditions under which $\mathfrak{u}_n$ tends to zero. However, due to the nonlinear nature of the system, only limited conclusions can be drawn, and we refer the reader to Section 5.5 for further discussion.

5.5. Nonlinear stability

Because of the nonlinearities intrinsically present in our problem, it becomes relevant to study the region of nonlinear stability. The goal here is to consider a dissipative N-ODE and check for which timestep size the dissipative property is satisfied also by the numerical solution. The dissipative property means that the error between two solutions, w_N, u_N decreases monotonically over time: $\frac{\partial}{\partial t}\left(\|w_N - u_N\|_2^2\right) < 0$. In what follows, the use of the 2-norm will be necessary. This property is satisfied if the one-sided Lipschitz constant of F_N, namely ν_{F_N}, is negative [6,28,37]. Therefore, given $\nu_{F_N} < 0$, our model problem is

$$\begin{aligned} \dot{u}_N &= F_N(u_N), \quad t \in (0,\infty), \\ u_N(0) &= u_{N0}. \end{aligned}$$

The satisfaction of nonlinear stability implies absolute stability [28]. Indeed, if $\lim_{t\to\infty} u_N = 0$, the dissipative property ensures that any perturbed solution also converges to zero. Moreover, we notice that, if Λ is symmetric, F_N defined in (5.5) satisfies the dissipative condition. Therefore, with the aforementioned conditions, the results shown in this section are sufficient to meet the requirements outlined in Section 5.4 for symmetric Λ. Let us now find the relation between $\|\mathfrak{u}_n^{k+1} - \mathrm{w}_n^{k+1}\|_2^2$ and $\|\mathfrak{u}_n^{k} - \mathrm{w}_n^{k}\|_2^2$ obtained with the FE scheme:

$$\begin{aligned} \|\mathfrak{u}_n^{k+1} - \mathrm{w}_n^{k+1}\|_2^2 &= \langle \mathfrak{u}_n^k + \Delta t F_n(\mathfrak{u}_n^k) - \mathrm{w}_n^k - \Delta t F_n(\mathrm{w}_n^k), \mathfrak{u}_n^k + \Delta t F_n(\mathfrak{u}_n^k) - \mathrm{w}_n^k - \Delta t F_n(\mathrm{w}_n^k)\rangle \\ &= \|\mathfrak{u}_n^k - \mathrm{w}_n^k\|_2^2 + \Delta t^2 \|F_n(\mathfrak{u}_n^k) - F_n(\mathrm{w}_n^k)\|_2^2 + 2\Delta t \langle F_n(\mathfrak{u}_n^k) - F_n(\mathrm{w}_n^k), \mathfrak{u}_n^k - \mathrm{w}_n^k\rangle \\ &< \|\mathfrak{u}_n^k - \mathrm{w}_n^k\|_2^2 + L_{F_n}^2 \Delta t^2 \|\mathfrak{u}_n^k - \mathrm{w}_n^k\|_2^2 + 2\Delta t \nu_{F_n} \|\mathfrak{u}_n^k - \mathrm{w}_n^k\|_2^2 \\ &= (1 + L_{F_n}^2 \Delta t^2 + 2\Delta t \nu_{F_n}) \|\mathfrak{u}_n^k - \mathrm{w}_n^k\|_2^2, \end{aligned} \tag{5.6}$$

where ν_{F_n} is the one-sided Lipschitz constant of F_n, assumed to be negative. For general nonlinear encoders and decoders, the dissipativity property, i.e., $\nu_{F_n} < 0$, is not guaranteed in a straightforward manner, but it can be encouraged, for example, through an appropriate penalization term in the training loss to enforce a negative one-sided Lipschitz constant. For example, the one-sided Lipschitz constant itself computed on the minibatch, or alternatively the quantity $\exp(\langle F_n(\Psi'(u_{N,1})) - F_n(\Psi'(u_{N,2})),\ \Psi'(u_{N,1}) - \Psi'(u_{N,2})\rangle)$, also evaluated selecting $u_{N,1}, u_{N,2}$ on the minibatch, may be used as a penalty term. In this way, the stability properties of the original system can be preserved in the reduced-order surrogate. Then, from (5.6), we derive the bound for the timestep size:

$$1 + L_{F_n}^2 \Delta t^2 + 2\Delta t \nu_{F_n} < 1 \quad \Rightarrow \quad \Delta t < \frac{2|\nu_{F_n}|}{L_{F_n}^2}, \tag{5.7}$$

which guarantees that the error between any two solutions of the n-ODE asymptotically vanishes. The bound in (5.7) can be interpreted as a sufficient condition to enforce absolute stability in the presence of a nonlinear n-ODE.

6. Approximation of n-ODE by neural networks

In this section, we consider three neural networks, $N_{\Psi'}$, N_Ψ, N_{F_n}, with trainable weights $w_{\Psi'}$, w_Ψ, w_{F_n}, to approximate, respectively, Ψ', Ψ, and F_n and we study the effects of this approximation. F_n depends on μ and to account for this dependence we provide μ as input to the neural network, so $N_{F_n} = N_{F_n}(u_n, \mu)$. For the sake of notation, unless it is relevant for the context, we avoid to explicitly write the parameters as argument.

The core idea of the following analysis is to interpret the functions approximated by neural networks as perturbations of their exact counterparts. Accordingly, we make the following definition.

Definition 5 (Discrepancies κ, υ, ω)**.** We denote by κ, υ, ω the discrepancies between the encoder, decoder, right hand side of the n-ODEand their approximations, such that

$$N_{\Psi'}(u_N) = \Psi'(u_N) + \kappa(u_N),\ N_\Psi(u_n) = \Psi(u_n) + \upsilon(u_n),\ N_{F_n}(u_n) = F_n(u_n) + \omega(u_n). \tag{6.1}$$

The universal approximation theorems state that, within appropriate functional spaces, neural networks can approximate functions with arbitrarily small error. We refer to [7, 29, 42, 54] and the references therein for results concerning the approximation of continuous functions, such as Ψ', Ψ, and F_n.

A further observation is in order: since F_n depends on the Jacobian of the encoder (see (4.1)), it is advisable to employ smooth activation functions for $N_{\Psi'}$ (and not necessarily for N_{F_n} nor N_Ψ), such as the ELU, to ensure differentiability.

According to the aforementioned literature, and considering the discussion in Section 3.2, the approximation errors κ, υ, and ω can be bounded by constants provided the neural networks are sufficiently expressive and well trained. Specifically, these errors satisfy $\|\kappa\| \leq \varepsilon_{\Psi'}$, $\|\upsilon\| \leq \varepsilon_\Psi$, and $\|\omega\| \leq \varepsilon_{F_n}$, where each ε depends on the particular function being approximated. The sources of error can be due to

- insufficient expressivity of the neural networks and/or inappropriate architecture choice, represented by $\mathcal{E}_1$ and $\mathcal{E}_2$. However, the approximation theorems in [7, 29, 42, 54] and the discussion in Section 3 ensure that, using a proper architecture, as $|w| \to \infty$, this source of error vanishes;
- optimization procedure used to determine w, which may yield a suboptimal set of parameters due to non-convergence or convergence to a local minimum and/or insufficient training data;
- the use of a time integration scheme with finite timestep size for training the neural networks.

Here, we are not interested in studying the second contribution. Instead, we assume that the optimization process leads to the global optimum and, additionally, that the neural networks are sufficiently large, and that enough samples in $\mathcal{P}$ are available. A brief discussion on the third error source is done in Section 6.1. The upper bounds $\varepsilon_{\Psi'}$, ε_Ψ, and ε_{F_n} include all the aforementioned sources of error.

Stability - Propagation of neural network's errors. We begin our analysis by proceeding step by step. First, we examine the Lyapunov stability of the exact-in-time problem, and subsequently, we account for the effects of time discretization. From (6.1) we understand that the analysis resembles

that of Section 4.3, but here the perturbed solution, w_n, derives from the application of neural networks to the n-ODE. We have that the error is bounded by the accuracy of the neural networks (see Appendix A for the computations):

$$\|w_N - u_N\| \leq L_\Psi\Big(\varepsilon_{\Psi'} + \varepsilon_{F_n} t\Big)e^{L_{F_n} t} + \varepsilon_\Psi,$$

showing that the approximation error introduced solely by the neural networks is controlled.

Fully approximated problem. We move now to the fully approximated problem, $\mathfrak{P}(\mathfrak{u})$, where $\mathfrak{u} = (\mathfrak{u}_N, \mathfrak{u}_n)$ and $\mathfrak{u}_N$, $\mathfrak{u}_n$ represent, respectively, the reconstructed and the n-solution:

$$\mathfrak{P}(\mathfrak{u}) = \begin{cases} \mathfrak{u}_N^{k+1} = N_\Psi(\mathfrak{u}_n^{k+1}), \\ \mathscr{L}_n(\mathfrak{u}_n^{k+1}, \mathfrak{u}_n^k, \dots, \mathfrak{u}_n^{k+1-P}, N_{F_n}, \Delta t) = 0, \quad k = P-1, P, \dots \\ \mathfrak{u}_n^0 = N_{\Psi'}(u_N^0). \end{cases} \tag{6.2}$$

The main objective is to ensure that the fully-approximated numerical solution can represent accurately the exact one. We treat the time integration scheme combined with the neural networks as a new numerical method, whose accuracy can be controlled through some sets of hyper-parameters: the timestep size separating two consecutive training data, called *training timestep size*, Δt_{train}, the timestep size used during the online phase, Δt, and the number of trainable weights, $|w|$ (Section 3.1). Therefore, the goal is to ensure that $\|\mathfrak{u}_N - u_N\| \to 0$ as $\Delta t_{\text{train}}, \Delta t \to 0$ and $|w| \to \infty$. Before continuing, we need to present the training procedure in Section 6.1 and afterwards discuss the convergence in Section 6.3.

6.1. Training

In this section, we outline the methodology for training the neural networks. The training process is formulated as an optimization problem, in which a loss function is computed and minimized using data. The first step involves the generation of the datasets. We sample the parameters from a uniform distribution over the parameter space, and the corresponding solutions, u_N, along with any additional required quantities, are computed and stored. The resulting data are partitioned into three subsets: training, validation, and test datasets. The validation and test sets each comprise at least 10% of the total data. The second step consists of the minimization procedure, using a gradient-based method, for which we consider two scenarios, detailed below.

In the following, we denote by M_N the integration in time scheme for solving the N-ODE and similarly for M_n. Let v_N^k be the discrete N-solution obtained through M_N. The manifold defined by $\{\mathrm{v}_{N,i}^k\}_{\mu_i \in \mathcal{P}}$ differs from the exact one, $\mathcal{M}$, depending on the accuracy of M_N. Note that we add the subscript i in $\mathrm{v}_{N,i}^k$ to denote the solution for the parameter μ_i. The solution v_N^k are computed with a timestep size Δt_N. Without altering the main results, for the sake of simplicity, we henceforth assume that $\Delta t_{\text{train}} = \Delta t_N$, and we simply write Δt_{train}. We consider the set $\{\mathrm{v}_N^k\}$ to constitute the training data, as is commonly done in practical applications.

6.1.1. Semi data-driven

The first approach consists in jointly training $N_{\Psi'}$ and N_Ψ, followed by the training of N_{F_n} to approximate F_n by directly minimizing the discrepancy between N_{F_n} and F_n. We compute F_n from its definition (4.1), so it is necessary to store the values of F_N together with the solutions u_N for each

timestep size and parameter instance. This procedure is not purely data-driven, as F_N is typically not provided as an output by the solver. Therefore, some understanding of the solver's implementation is required to extract and store F_N. Following the aforementioned ideas, the first step is to force the autoencoder to reproduce the identity by minimizing a functional $\mathscr{J}_1$ computed on a minibatch of size $n_{\text{train}} \times N_{\text{time}}$, containing n_{train} samples in $\mathcal{P}$ and $N_{\text{time}}(\mu)$ timesteps for each parameter sample. Calling $w_{\Psi'}$ and w_{Ψ} the trainable weights of $N_{\Psi'}$ and N_{Ψ}, respectively, the loss $\mathscr{J}_1$ is defined as

$$\mathscr{J}_1 = \frac{1}{n_{\text{train}} N_{\text{time}}} \sum_{i=1}^{n_{\text{train}}} \sum_{j=1}^{N_{\text{time}}} \mathscr{N}_{ij}^1, \qquad \mathscr{N}_{ij}^1(w_{\Psi'} \cup w_{\Psi}) = \|\mathrm{v}_{N,i}^j - N_{\Psi} \circ N_{\Psi'}(\mathrm{v}_{N,i}^j)\|_2^2. \tag{6.3}$$

The second step is to learn F_n by minimizing $\mathscr{J}_2$:

$$\mathscr{J}_2 = \frac{1}{n_{\text{train}} N_{\text{time}}} \sum_{i=1}^{n_{\text{train}}} \sum_{j=1}^{N_{\text{time}}} \mathscr{N}_{ij}^2, \qquad \mathscr{N}_{ij}^2(w_{F_n}) = \|J_{N_{\Psi'}} F_{N,i}^j - N_{F_n,i}^j\|_2^2, \tag{6.4}$$

where w_{F_n} are the trainable weights of N_{F_n}.

6.1.2. Fully data-driven

In this second approach, the data consist only in samples of v_N, not F_N. The training of $N_{\Psi'}$, N_{Ψ}, and N_{F_n} can be made concurrently by minimizing:

$$\mathscr{J}_3 = \frac{1}{n_{\text{train}} N_{\text{time}}} \sum_{i=1}^{n_{\text{train}}} \sum_{j=1}^{N_{\text{time}}} \sum_{r=\{1,3\}} \eta_r \mathscr{N}_{ij}^r,$$

where $\eta_r > 0$ are user-defined scalar weights, $\mathscr{N}_{ij}^1$ is the loss for the autoencoder, defined in (6.3), $\mathscr{N}_{ij}^3$ is the loss contribution to approximate F_n by minimizing the residual of (2.4), defined as:

$$\mathscr{N}_{ij}^3(w_{F_n}) = \left\| \sum_{p=0}^{P} \alpha_p \overline{N}_{\Psi'}(\mathrm{v}_{N,i}^{j+1-p}) - \Delta t_{\text{train}} \sum_{p=0}^{P} \beta_p N_{F_n}(\mu_i, \overline{N}_{\Psi'}(\mathrm{v}_{N,i}^{j+1-p})) \right\|_2^2, \tag{6.5}$$

where $\overline{N}_{\Psi'}(\cdot)$ is the output of Ψ' for fixed value of the trainable weights. Indeed $\mathscr{N}_{ij}^3$ depends only on the trainable weights of N_{F_n}.

It is relevant to note that, thanks to the use of the autoencoder, it is possible to compute the n-solution as $\overline{\Psi'}(\mathrm{v}_{N,i}^k)$ breaking the dependence of $u_{n,i}^k$ on its value at the previous timestep, $u_{n,i}^{k-1}$. Therefore, the temporal loop can be broken, enabling the possibility of a straightforward parallelization of the computation of $\mathscr{N}_{ij}^3$. Doing so, it is possible to fully exploit the computational power of GPUs. Furthermore, by providing data at different times, the risk of overfitting is mitigated.

Moreover, we highlight that the reduced model is trained by minimizing a single loss function, $\mathscr{J}_3$, whose components, $\mathscr{N}_{ij}^1$ and $\mathscr{N}_{ij}^3$, depend exclusively on either $w_{\Psi'} \cup w_{\Psi}$ or w_{F_n}. This separation simplifies the potentially intricate and highly nonlinear structure of the loss function.

6.1.3. Characterization of F_n approximation error.

We now study how accurate the two training strategies are in approximating F_n. We assume that $|w_{F_n}| \to \infty$ and layers large enough that F_n is arbitrarily expressive. This assumption allows us to introduce an additional setting: we assume that the minimum value of the loss function is attained, i.e., $\mathscr{J}_i = 0$, $i = 1, 2, 3$. Our goal is to identify necessary condition under which the minimum value $\mathscr{J}_i$ implies $\max_{k=0,1,\dots} \|N_{F_n}(t_k) - F_n(t_k)\| \to 0$.

Regarding the semi data-driven methods, the minimization of (6.4) implies enforcing a match between N_{F_n} and $J_{N_{\Psi'}} F_N$. It follows that a necessary condition for $\max_{k=0,1,\dots} \|N_{F_n}(t_k) - F_n(t_k)\| \to 0$ is that $\|J_{N_{\Psi'}}(t_k) - J_{\Psi'}(t_k)\|_\infty \to 0$, therefore, an accurate representation of the encoder and its Jacobian is necessary.

Fully data-driven methods should be treated with care. We first notice that, by the definition of the local truncation error, the following equation holds:

$$\left\| \sum_{p=0}^{P} \alpha_p \Psi'(u_N(t_{j+1-p})) - \Delta t_{\text{train}} \sum_{p=0}^{P} \beta_p F_n(u_N(t_{j+1-p})) - \Delta t_{\text{train}} \tau_n^j \right\|_2^2 = 0. \tag{6.6}$$

We call ζ_j the difference between the discrete and the exact N-solutions: $\zeta_j = \mathrm{v}_N^j - u_N(t_j)$. Assuming a convergent M_N, we have that $\zeta_j \to 0$ for $\Delta t_{\text{train}} \to 0$, $j = 0, 1, \dots$. We introduce now the neural network for the encoder, $N_{\Psi'} = \Psi' + \kappa$. We remind that, assuming enough data, accurate training, and proper choice of the architecture, by Theorems 1 and 2, the error κ can be made arbitrarily small. By the last considerations, (6.5) can be rewritten as

$$\begin{aligned} \mathscr{N}_{ij}^3 = \Bigg\| & \sum_{p=0}^{P} \alpha_p \left(\Psi'(u_N(t_{j+1-p}) + \zeta_{j+1-p}) + \kappa(u_N(t_{j+1-p}) + \zeta_{j+1-p}) \right) \\ & - \Delta t_{\text{train}} \sum_{p=0}^{P} \beta_p N_{F_n}(\Psi'(u_N(t_{j+1-p}) + \zeta_{j+1-p}), \mu_i) \Bigg\|_2^2, \end{aligned} \tag{6.7}$$

Given the assumption $\mathscr{J}_i = 0$, we set $\mathscr{N}_{ij}^3 = 0$. Then, setting (6.7) to zero and comparing it to (6.6), we conclude that

Proposition 11. The fully data-driven is a convergent method for training N_{F_n} in the sense that, under proper conditions, $\max_{k=0,1,\dots} \|N_{F_n}(t_k) - F_n(t_k)\| \to 0$. A necessary condition to meet the requirement $\max_{k=0,1,\dots} \|N_{F_n}(t_k) - F_n(t_k)\| \to 0$ is that $\tau_n^k, \zeta_k \to 0$ and $\kappa \to 0$. The former can be achieved with $\Delta t_{\text{train}} \to 0$, the latter can be achieved with $|w_{\Psi'} \cup w_\Psi| \to \infty$, or if $\mathcal{M}$ is linear so that a linear autoencoder, thus a finite number of trainable weights, is sufficient for having $\kappa = 0$.

From a practical point of view, Eq (6.7) states that N_{F_n}, through the optimization process described in Section 6.1.2, approximates the correct right-hand side, up to an error introduced by the discretization of the N-ODE (due to ζ), an error arising from the enforcement of the reference n-solution, obtained via compression of the full-order one, to fit the M_n scheme (due to τ_n), and an additional error due to the approximation of the encoder (due to κ).

Further details can be found in [12], where it is shown that, assuming exact data, which in our framework corresponds to the availability of u_n, the approximation error resulting from the

minimization of the residual is bounded by $C_2(\Delta t_{\text{train}}^P + e_{\mathcal{A}})$, were C_2 is a constant depending on M_n, P is the order of convergence of M_n, and $e_{\mathcal{A}}$ denotes the error introduced by the approximation capability of N_{F_n}.

In the fully data-driven approach, due to the intrinsic errors introduced by (6.7), which leads N_{F_n} to possibly not represents F_n accurately, the evaluation of the trained reduced order model should be performed using the same scheme M_n and the same timestep size employed during training, $\Delta t = \Delta t_{\text{train}}$. On the other hand, the semi data-driven approach allows the possibility to change both M_n and Δt during the online phase, while still yielding reliable solutions. See Section 7.2 for numerical evidence.

6.2. Solution of the fully discrete problem

We consider the P-steps scheme whose difference operator is written in (2.4) and it is rewritten here with a more convenient form for this section:

$$\sum_{p=0}^{P} \alpha_p \mathfrak{u}_n^{k+1-p} = \Delta t \sum_{p=0}^{P} \beta_p N_{F_n}(\mathfrak{u}_n^{k+1-p}), \quad k = P-1, P, \dots$$

with the following auxiliary conditions for the first $P-1$ steps, derived from a one-sided finite difference of the same order of convergence of the P-steps scheme:

$$\sum_{p=0}^{P} \gamma_p \mathfrak{u}_n^{k+p} = \Delta t N_{F_n}(\mathfrak{u}_n^{k}), \quad k = 0, \dots, P-1. \tag{6.8}$$

To ease the notation, we set $n = 1$ although the results are easily extendable to higher dimension. According to the time integration scheme, for $K \geq 2P-1$, we define the following matrices:

$$A = \begin{bmatrix} \alpha_P, \alpha_{P-1}, \dots, \alpha_0, 0, \dots, 0 \\ 0, \alpha_P, \alpha_{P-1}, \dots, \alpha_0, 0, \dots, 0 \\ \vdots \\ 0, \dots, 0, \alpha_P, \alpha_{P-1}, \dots, \alpha_0 \end{bmatrix} \in \mathbb{R}^{(K-P+1)\times K},$$

$$B = \begin{bmatrix} \beta_P, \beta_{P-1}, \dots, \beta_0, 0, \dots, 0 \\ 0, \beta_P, \beta_{P-1}, \dots, \beta_0, 0, \dots, 0 \\ \vdots \\ 0, \dots, 0, \beta_P, \beta_{P-1}, \dots, \beta_0 \end{bmatrix} \in \mathbb{R}^{(K-P+1)\times K},$$

$$\Gamma = \begin{bmatrix} \gamma_0, \dots, \gamma_{P-1}, \gamma_P, 0, \dots, 0 \\ 0, \gamma_0, \dots, \gamma_{P-1}, \gamma_P, 0, \dots, 0 \\ \vdots \\ 0, \dots, 0, \gamma_0, \dots, \gamma_{P-1}, \gamma_P \end{bmatrix} \in \mathbb{R}^{(P-1)\times K},$$

$$C = \begin{bmatrix} 1, 0, \dots, 0 \\ 0, 1, 0, \dots, 0 \\ \vdots \\ 0, \dots, 0, 1, 0, \dots, 0 \end{bmatrix} \in \mathbb{R}^{(P-1)\times K},$$

and the following vector whose elements are the solution at each timesteps: $\underline{\mathfrak{u}}_n = [\mathfrak{u}_n^0, \mathfrak{u}_n^1, \ldots, \mathfrak{u}_n^K]^T$ and similarly for $\underline{N}_{F_n}$. Note that, to avoid any confusion, the arrays of solutions at each timestep are denoted with the underline. The solution in time of the n-ODE is obtained from the following nonlinear system:

$$\begin{bmatrix} \Gamma \\ A \end{bmatrix} \underline{\mathfrak{u}}_n = \Delta t \begin{bmatrix} C \\ B \end{bmatrix} \underline{N}_{F_n}(\underline{\mathfrak{u}}_n). \tag{6.9}$$

6.3. Global convergence

We now proceed to study the error $\|u_N - \mathfrak{u}_N\|$ between the exact solution and the solution of the fully approximated problem (6.2), and the corresponding matrix form with initial conditions included (6.9). This result explicitly relates the total error to the neural network approximation error of the reduced dynamics and the decoder reconstruction accuracy, offering a key theoretical guarantee for the effectiveness of the DRE method. We remark in advance that the results in this section depend on the approximation errors $\varepsilon_{\Psi'}$, ε_{Ψ}, and ε_{F_n}, which are generally unknown since the exact Ψ' and Ψ are not available. It follows that the following discussion is primarily intended as a theoretical result, necessary to establish the convergence property of the proposed method.

Theorem 6 (Global convergence)**.** *We make the following assumptions: Ψ', Ψ, and F_n are continuous functions; the data are generated by approximating the N-ODE using a convergent scheme, M_N; the training leads to the global minimum of the loss functions; we use a convergent time integration scheme M_n of order P. Then, we have the following bound:*

$$\max_{\mu} \|\underline{u}_N - \underline{\mathfrak{u}}_N\|_\infty \leq L_\Psi(\varepsilon_{\Psi'}(w) + \varepsilon_{F_n}(\Delta t_{train}, w)T)e^{L_{F_n}T} + L_\Psi C_1 \Delta t^P + \varepsilon_\Psi(w). \tag{6.10}$$

So, for $|w| \to \infty$ and $\Delta t, \Delta t_{train} \to 0$, we have $\max_\mu \|\underline{u}_N - \underline{\mathfrak{u}}_N\|_\infty \to 0$.

Proof. The error between the fully approximated solution and the exact one can be decomposed as

$$\|\underline{u}_n - \underline{\mathfrak{u}}_n\|_\infty \leq \underbrace{\|\underline{u}_n - \underline{w}_n\|_\infty}_{H} + \underbrace{\|\underline{w}_n - \underline{\mathfrak{u}}_n\|_\infty}_{Q}, \tag{6.11}$$

where w_n is the exact solution of the following n-ODE

$$\begin{aligned} \dot{w}_n &= N_{F_n}(w_n), \\ w_n(0) &= N_{\Psi'}(u_{N0}). \end{aligned} \tag{6.12}$$

In (6.11), H is due to neural network approximation of the right-hand side, and Q is due to the time discretization. Using (6.1), Eq (6.12) can be written as

$$\begin{aligned} \dot{w}_n &= F_n(w_n) + \omega(w_n), \\ w_n(0) &= \Psi'(u_{N0}) + \kappa(u_{N0}). \end{aligned} \tag{6.13}$$

Subtracting (2.3) to (6.13), we have

$$\begin{aligned} \dot{w}_n - \dot{u}_n &= F_n(w_n) - F_n(u_n) + \omega(w_n), \\ w_n(0) &= \kappa(u_{N0}). \end{aligned} \tag{6.14}$$

Applying Grönwall's lemma yields

$$H = \|\underline{w}_n - \underline{u}_n\|_\infty \leq (\varepsilon_{\Psi'} + \varepsilon_{F_n} T) e^{L_{F_n,\mu} T}. \tag{6.15}$$

The term Q in (6.11) describes the error introduced by the time discretization of (6.12), which depends on the specific time integration scheme used and can generally be bounded by [6, 28, 37]

$$Q = \|\underline{w}_n - \underline{\mathfrak{u}}_n\|_\infty \leq C_{1,\mu} \Delta t^P, \tag{6.16}$$

where $C_{1,\mu} > 0$ generally depends of the specific r.h.s. at hand and time integration scheme. Finally, replacing (6.15) and (6.16) into (6.11) we obtain the following bound

$$\|\underline{u}_n - \underline{\mathfrak{u}}_n\|_\infty \leq (\varepsilon_{\Psi'}(w) + \varepsilon_{F_n}(\Delta t_{\text{train}}, w) T) e^{L_{F_n,\mu} T} + C_{1,\mu} \Delta t^P, \tag{6.17}$$

where we have emphasized the dependence of the neural network approximation error on the trainable weights, w, and training timestep, Δt_{train}.

From (6.11), and using (6.1), we obtain the error on the reconstructed solution for one trajectory:

$$\begin{aligned}
\|\underline{u}_N - \underline{\mathfrak{u}}_N\|_\infty &= \|\underline{\Psi}(\underline{u}_n) - \underline{\Psi}(\underline{\mathfrak{u}}_n) - \underline{\upsilon}(\underline{\mathfrak{u}}_n)\| \\
&= \|\underline{\Psi}(\underline{u}_n) - \underline{\Psi}(\underline{\mathfrak{u}}_N) - \underline{\upsilon}(\underline{\mathfrak{u}}_n) - \underline{\Psi}(\underline{w}_n) + \underline{\Psi}(\underline{w}_n)\|_\infty \\
&\leq L_\Psi \|\underline{u}_n - \underline{w}_n\|_\infty + L_\Psi \|\underline{w}_n - \underline{\mathfrak{u}}_n\|_\infty + \varepsilon_\Psi \\
&\leq L_\Psi (\varepsilon_{\Psi'} + \varepsilon_{F_n} T) e^{L_{F_n,\mu} T} + L_\Psi C_{1,\mu} \Delta t^P + \varepsilon_\Psi.
\end{aligned}$$

Taking the maximum across the trajectories leads to the global error:

$$\max_\mu \|\underline{u}_N - \underline{\mathfrak{u}}_N\|_\infty \leq L_\Psi (\varepsilon_{\Psi'}(w) + \varepsilon_{F_n}(\Delta t_{\text{train}}, w) T) e^{L_{F_n} T} + L_\Psi C_1 \Delta t^P + \varepsilon_\Psi(w), \tag{6.18}$$

where $C_1 = \max_\mu C_{1,\mu}$. Using the fully data-driven training, by Proposition 11, we have that $\varepsilon_{F_n} \to 0$ as $|w| \to \infty$ and $\Delta t_{\text{train}} \to 0$. By the assumption of sufficient training data and that a global minimum of the loss is reached, the term $\varepsilon_\Psi \to 0$ as $|w| \to \infty$. Finally, the error introduced by the integration in time scheme for the solution of the n-ODE $L_\Psi C_1 \Delta t^P \to 0$ as $\Delta t \to 0$. □

We observe that the encoder appears in the error constant that multiplies the exponential ($\varepsilon_{\Psi'}$) and in L_{F_n} (see (4.2)). The decoder appears in terms of its Lipschitz constant, in L_{F_n}, and as an additional term with ε_Ψ. Therefore, we understand that it is important to ensure that both the encoder and the decoder are accurately approximated.

Remark 4. Similarity with proper orthogonal decomposition: The DRE can be interpreted as a nonlinear generalization of Proper Orthogonal Decomposition (POD) [32, 52, 56], in which the linear transition matrices obtained via singular value decomposition of the snapshot matrix are replaced by neural networks. Furthermore, our methodology also involves approximating the right-hand side F_n of the n-ODE, in order to further accelerate the overall computational procedure.

7. Numerical tests

In this section, we present a few test cases to numerically verify the theory developed and show a generic application of DRE. The search of the optimal neural network is not the goal of this paper, so only a preliminary manual optimization of the hyperparameters, such as the number of data, batch size, number of layers, learning rate and its schedule, and type of activation function, is done. In all test cases, we use the Adam [35] optimizer implemented in PyTorch v2.6 [1] using the default settings to perform minimization.

7.1. Test 1: absolute stability

Here we study numerically the stability after reduction (Section 5.4). To properly isolate this property from other discretization error sources, we do not approximate F_n with N_{F_n} but, instead, we write it as its definition (4.1). For practical simplicity, Ψ' and Ψ are approximated with $N_{\Psi'}$ and N_Ψ, respectively. When not linear, these neural networks are made highly expressive to ensure a negligible approximation error for the purpose of this study. Due to the limited theoretical results available for the general case, we focus here on two specific scenarios: the linear autoencoder and the linear–nonlinear autoencoder.

7.1.1. Linear encoder – Linear decoder

The N-ODE taken into account is

$$\begin{aligned} \dot{u}_N &= \Lambda u_N, \\ u_N(0) &= [\mu, 1, 1]^T, \end{aligned} \tag{7.1}$$

where $u_N \in \mathbb{R}^3$, $\Lambda \in \mathbb{R}^{3\times 3}$ is a diagonal matrix with entries $[-3, -5, -5]$ along the diagonal, and $\mu \sim \mathcal{U}(1, 5)$ is the parameter. We observe that a linear ODE system, such as (7.1), may arise from the spatial discretization of a PDE representing the heat equation. As mentioned in Section 5.4, in this scenario we expect an arbitrarily long extrapolation in time without instabilities.

The reduction maps a vector in $\mathbb{R}^3$ to a vector in $\mathbb{R}^2$. A low-dimensional setting is intentionally chosen to ease the graphical visualization of the results. Since Λ has real eigenvalues, no oscillatory behavior is observed, and the manifold is linear. Indeed, by calling $x = e^{-3t}$ and $y = x^{5/3}$, we have $[u_{n,1}, u_{n,2}, u_{n,3}] = [\mu e^{-3t}, e^{-5t}, e^{-5t}] = [\mu x, y, y]$, which represents a flat surface. Therefore, a linear autoencoder is sufficient to approximate the identity map $\mathrm{Id}_{\mathcal{M}}$. In this case, Ψ' and Ψ are linear and, for practical convenience, they are approximated by $N_{\Psi'}$ and N_Ψ, that are matrices whose entries are still determined through an optimization process to minimize the loss in (6.3). The networks' architectures are reported in Table B2.

We report below the numerical values, up to the third digit, of the Jacobians of the encoder and decoder, together with the reduced matrix $A = J_{N_{\Psi'}} \Lambda J_{N_\Psi}$ and its eigenvalues:

$$J_{N_{\Psi'}} = \begin{bmatrix} 1.221 & 0.200 & 1.097 \\ -0.785 & -0.020 & 0.204 \end{bmatrix}, \quad J_{N_\Psi} = \begin{bmatrix} 0.149 & -1.043 \\ 0.631 & 0.982 \\ 0.631 & 0.982 \end{bmatrix},$$

$$A = \begin{bmatrix} -4.637 & -2.547 \\ -0.233 & -3.363 \end{bmatrix}, \qquad \mathrm{eig}(A) = -5, -3.$$

As expected, the numerical values of the eigenvalues of A coincide with those of Λ (see Section 5.4), which will lead to a timestep size bound for the n-ODE equal to that of the N-ODE. The manifold $\mathcal{M}$ is composed of 100 distinct trajectories, each defined over the time interval $[0, T]$ with $T = 0.5$. In Figure 5, we present the components of the reconstructed solution, $\Psi(\mathfrak{u}_n)$, as a function of time, where $\mathfrak{u}_n$ is obtained by solving the n-ODE using the FE scheme. The n-ODE is solved using two different timestep sizes. The first is $0.99\Delta t_{\max} = 0.99 \times \frac{2}{5}$ and is shown in the first row of Figure 5; the second is $0.5\Delta t_{\max} = 0.5 \times \frac{2}{5}$ and is shown in the second row of the same figure. As expected, for $\Delta t = 0.99\Delta t_{\max}$ the solution exhibits oscillations, although it decays to zero as time tends to infinity. No oscillations are observed for the smaller time-step size. Additionally, we remark that the end of the training time, $T = 0.5$, is highlighted in blue in Figure 5, while the solution is extrapolated up to $t = 100$.

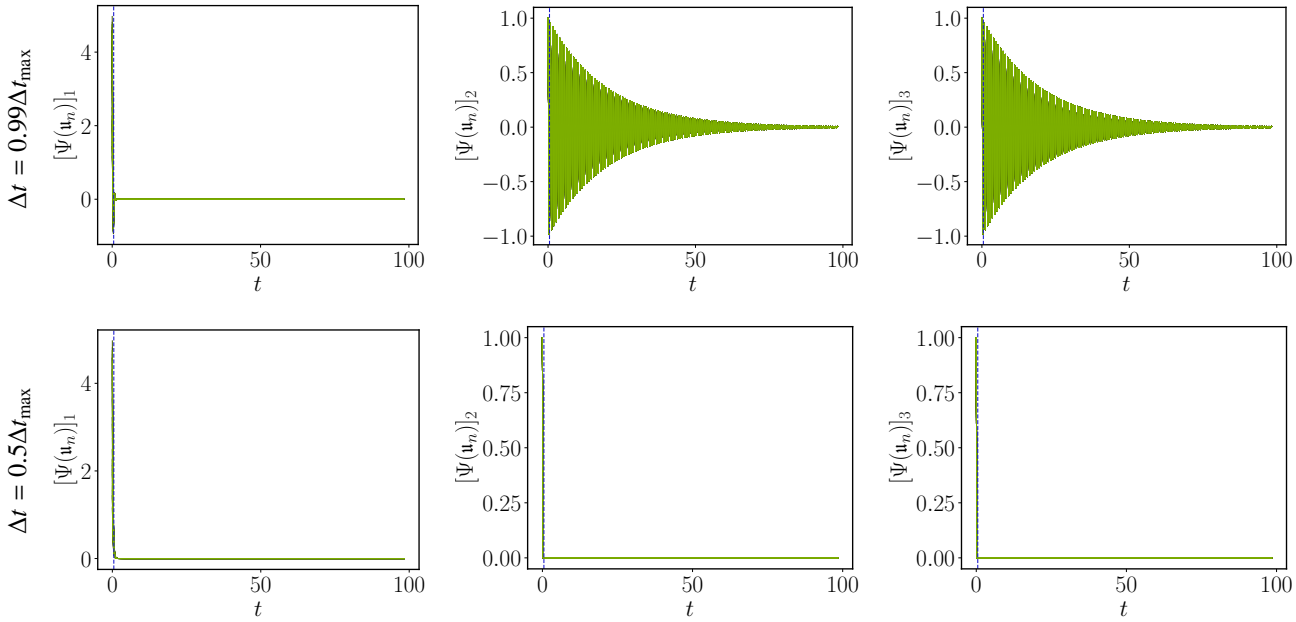


Figure 5. Linear case. Three components, one for each panel, of the reconstructed solutions in the test dataset. First row: the numerical solutions are computed using FE with $\Delta t = 0.99\Delta t_{\max}$. Second row: The timestep size is reduced to $\Delta t = 0.5\Delta t_{\max}$, at which the oscillations are absent. The vertical blue-dashed line is located at the end of the training time.

7.1.2. Linear encoder – Nonlinear decoder

We make here a parametric study with $N \in \{3, 100, 500\}$ to ensure the consistency of the results for different sizes of the N-ODE. For the sake of clarity, we describe below in detail only the case $N = 3$; analogous settings are used for the remaining two test cases. For each case $n = 2$.

The N-ODE taken into account is that in (7.1) with same parameter and initial condition; the only difference is the matrix Λ which, up to the third digit, is equal to

$$\Lambda = \begin{bmatrix} -4.75 & -1.55 & -0.79 \\ -1.55 & -4.31 & -1.02 \\ -0.79 & -1.02 & -5.2 \end{bmatrix},$$

with eigenvalues $\lambda_1 = -7.0$, $\lambda_2 = -2.93$, $\lambda_3 = -4.34$. In this test case, despite the only change compared to (7.1) being Λ, a nonlinear decoder is necessary since the manifold is not flat, see Figure 6.

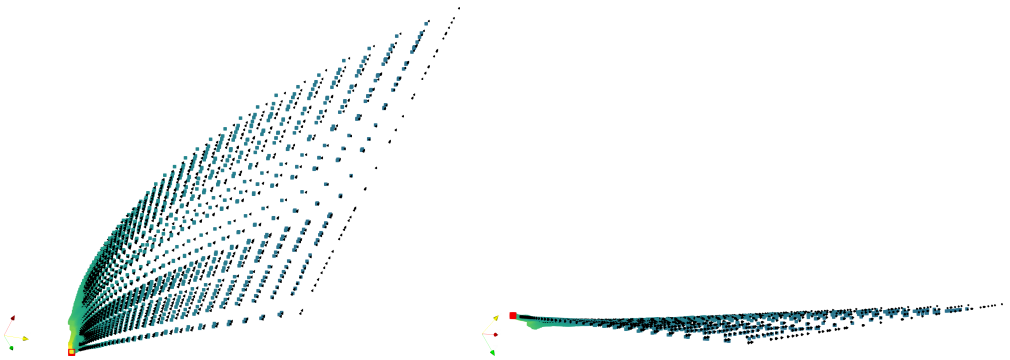
The networks' architectures are reported in Table B2. The training is accomplished on a dataset of 100 samples and the results computed here are computed on the test dataset made of 50 samples, those represented in Figure 6. The training is interrupted after 50 epochs when the loss reaches a low value of about 10^{-4}. After the training, ν_{F_n} and L_{F_n} are computed numerically on the test dataset, following the procedure presented in [13]. Up to the third digit, they result to be $\nu_{F_n} = -3.14$, which is similar to that of F_N ($\nu_{F_N} = \lambda_2 = -2.93$), and $L_{F_n} = 10.4$, from which follows the bound on the timestep size for FE:

$$\Delta t \leq \Delta t_{\max} = \frac{-2\nu_{F_n}}{L_{F_n}} = 0.058,$$

which is about five time smaller to that to assure the stability of the N-ODE if solved directly, $\Delta t \leq \frac{2}{\max_i |\lambda_i|} \approx 2/7.0 \approx 0.28$ and it is about the half to that for assuring the dissipative behavior on the numerical solution of the N-ODE

$$\Delta t \leq \frac{-2\nu_{F_N}}{L_{F_N}} \approx \frac{-2(-2.93)}{7.0^2} \approx 0.120.$$

In the first row of Figure 7, we report the components of the reconstructed solution, $\Psi(\mathfrak{u}_n)$, as a function of time, where $\mathfrak{u}_n$ is obtained by solving the n-ODE using the FE scheme. The time integration is performed with a timestep size of $0.99\Delta t_{\max}$. Despite the approximations introduced by time discretization and the neural network models, and despite the training on partial trajectories truncated at early times, the solution stabilizes at the equilibrium value close to zero. The training interval, $T = 0.5$, is highlighted in blue in Figure 7. We emphasize that, despite the training on a subset of $\mathcal{M}$, due to the simplicity of the problem at hand, it was possible to successfully extrapolate up to grater times, e.g., $t = 10$ in Figure 7, without any instability. Similar results are obtained for $N = 100$ and $N = 500$, as depicted in Figure 7.



(a) view 1 **(b)** view 2

Figure 6. Two perspectives, **(a)** and **(b)**, of the curved manifold $\mathcal{M}$. Black cones indicate the data used to train the autoencoder, while the colored points represent the reconstructed $\Psi(\mathfrak{u}_n)$. The color scale (from blue to yellow) is related to the time, solely for the purpose ease the readability. It is worth noting that, in this test case, despite the extrapolation in time, the reconstructed solution still converges to a stable value near the origin (red point). The axes have been translated away from the origin (red point) to improve readability.

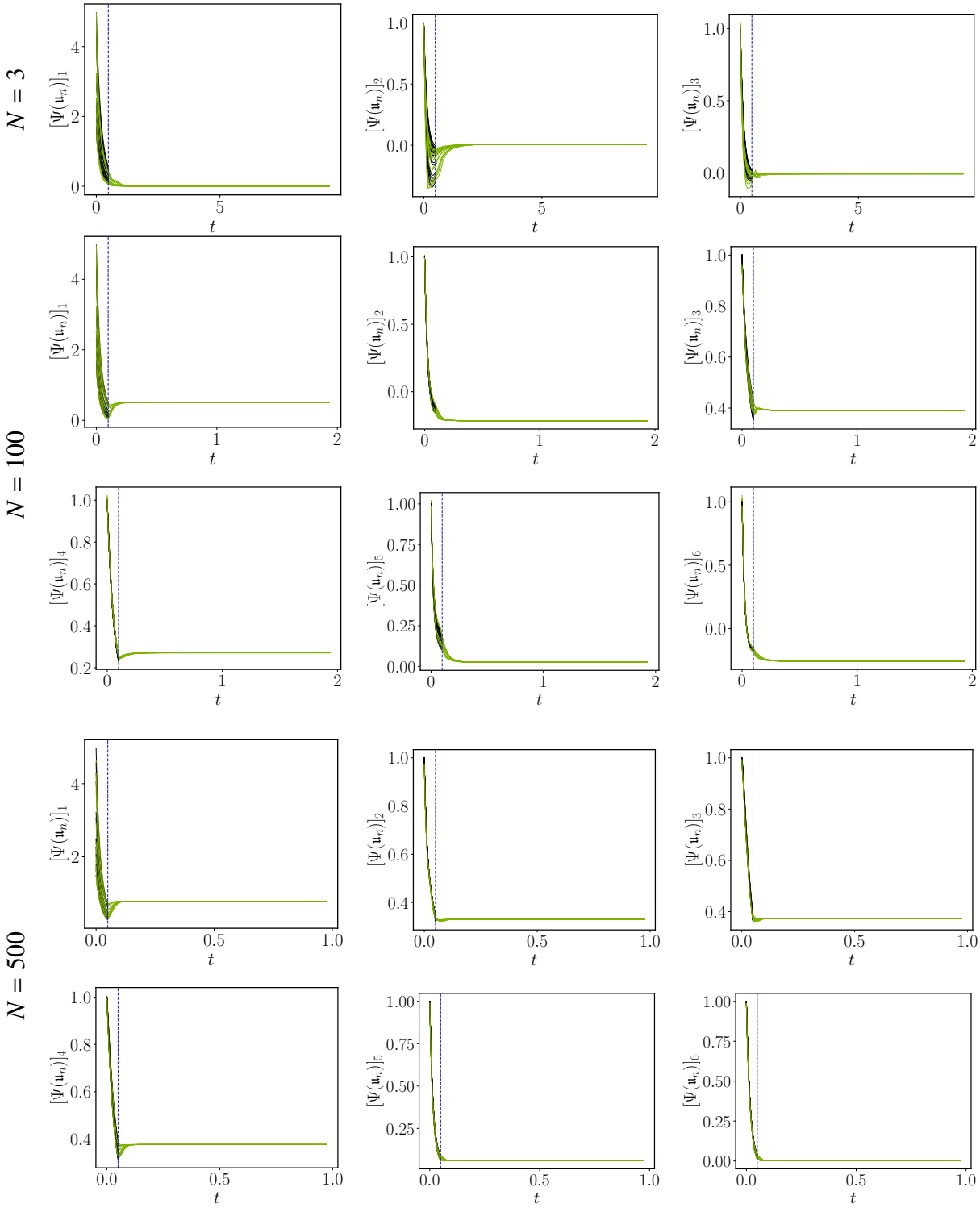


Figure 7. Non linear case. Components of $\Psi(\mathfrak{u}_n)$, one for each panel. First row of panels corresponds to $N = 3$, second and third row corresponds to $N = 100$, while the last two rows corresponds to $N = 500$. For visualization purposes, only the first six components of $\Psi(\mathfrak{u}_n)$ are displayed. $\mathfrak{u}_n$ is computed using FE with $\Delta t = 0.99\Delta t_{\max}$. The vertical blue-dashed line is located at the end of the training time. Note the use of different time ranges for each N for graphical clarity.

7.2. Test 2: convergence in time

In this section, we aim to numerically verify the convergence of the method as $\Delta t \to 0$, after the training of the neural networks. The global error in (6.18) depends on both the accuracy of the time integration method and the neural networks. Here, our goal is to assess convergence by isolating the contribution of the time integration scheme in (6.18). The first case, Section 7.2.1, concerns a manufactured problem which has an analytical solution, whereas the second case, Section 7.2.2, addresses a more general setting. To ease the readability, we report here the common setup of the two cases.

Since our goal is to analyze temporal convergence, in light of (6.18), the approximation error of the neural networks must be reduced as much as possible to isolate the term $L_\Psi C_1 \Delta t^p$. To this end, we employ deep neural networks with multiple layers and a large number of trainable parameters for both cases. Their specific architecture will be detailed in the following.

The width of the layers is chosen in agreement to what reported in Section 3 and the guidelines provided in [7, 42, 54]. In order to ensure continuity to F_n, the activation function used in the encoder are ELU $\in C^1$. We investigate three training scenarios:

(1) *Exact rhs*: F_n is computed directly from its definition, $J_{\Psi'} F_N(\Psi(\cdot))$, yielding the most accurate result since only the autoencoder is approximated by neural networks.
(2) *Semi data-driven*: F_n is approximated by N_{F_n} using augmented data and trained according to the strategy outlined in Section 6.1.1.
(3) *Fully data-driven*: F_n is approximated by N_{F_n} which is trained following the methodology described in Section 6.1.2. The training is accomplished with the FE difference operator, see (6.5).

After training, the quality of the DRE is assessed by computing the average relative error over the test dataset:

$$e_{\text{multi}} = \left(\frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \sum_{j=1}^{N_{\text{time}}} \frac{\|u_{N,i}^j - N_\Psi(\mathfrak{u}_{N,i}^j)\|_2^2}{\|u_{N,i}^j\|_2^2} \right)^{1/2},$$

where n_{test} is the number of data in the test dataset.

7.2.1. Analytical case – Manufactured solution

In order to properly isolate the error associated with time discretization, we introduce a manufactured solution providing analytical expressions for N-ODE and n-ODE. In particular, we derive explicit analytical expressions for Ψ', Ψ, and F_n. Nevertheless, we also report results in which the aforementioned quantities are approximated by properly trained neural networks, namely $N_{\Psi'}$, N_Ψ, and N_{F_n}, whose architectures are reported in Table B3. This strategy allows for a clear separation between the error contribution arising from time discretization and the additional error introduced by the approximation capabilities of the neural networks. This is achieved by comparing the results obtained using the analytical expressions of Ψ', Ψ, and F_n with those obtained by replacing them with $N_{\Psi'}$, N_Ψ, and N_{F_n}, respectively. In the following, we set $W \in \mathbb{R}^{N \times n}$ to be a matrix whose entries are randomly sampled from a uniform distribution over $[0, 1)$. Its pseudoinverse is denoted by $W^\dagger$. Here, $N = 3$ and $n = 2$. We denote by w_1 the first column of W. We then consider the following N-ODE:

$$\begin{aligned} &\dot{u}_N = \mu u_N \log(u_N), \quad \text{for } t \in (0, T], \\ &u_N(0) = \exp(w_1), \end{aligned} \tag{7.2}$$

where $\log(\cdot)$ and $\exp(\cdot)$ are to be understood component-wise; $T = 1$ is the final time. The decoder is defined as $\Psi(u_n) = I_{N\times N} \exp(W u_n)$ while the encoder is defined as $\Psi'(u_N) = W^\dagger \log(I_{N\times N} u_N)$. Note that here the biases of the layers are null. We highlight that, in this setting, non-standard activation functions, namely exp and log, are employed. This choice facilitates the analytical expressions of the quantities involved in this test case, which is the primary focus of the present analysis.

From the above definitions, we can obtain the expression for the n-ODE:

$$[\dot{u}_n]_i = [J_{\Psi'} F_N]_i = \sum_k \left(\sum_j W^\dagger_{ij} \frac{1}{[u_N]_j} \delta_{ik} \right) [F_N]_k,$$
$$u_n(0) = [1, 0]^\top.$$

Given the above definitions, the exact N- and n-solutions are given, respectively, by $u_N = \exp\left(\exp(\mu t) w_1\right)$ and $u_n = \exp(\mu t)[1, 0]^\top$. Note that the vanishing second component of u_n arises from the choice of selecting only the first column of W, namely w_1, in the definition of u_N and of the associated N-ODE. Moreover, we observe that, due to the expression of u_N, the resulting manifold is nonlinear.

The parameter μ is sampled 200 times from a uniform distribution, $\mu \sim \mathcal{U}(0.1, 2)$. Among these samples, 100 are used to form the training dataset, 50 the validation dataset (used to compare the loss during the training), and 50 the test dataset (used to assess the final accuracy of the model after training).

The neural networks, whose architectures are reported in Table B3, are trained using the FE scheme with timestep $\Delta t_{\text{train}} = 0.01$. During the online phase, the timestep size Δt can be varied.

In Figure 8, we report the values of e_{multi} for both the case employing analytical expressions for Ψ', Ψ, and F_n (panel (a)) and the case in which the reduced model is constructed using $N_{\Psi'}$, N_Ψ, and N_{F_n} (panel (b)).

The values of e_{multi} are shown for the three aforementioned strategies, namely "Exact rhs", "Semi data-driven", and "Fully data-driven".

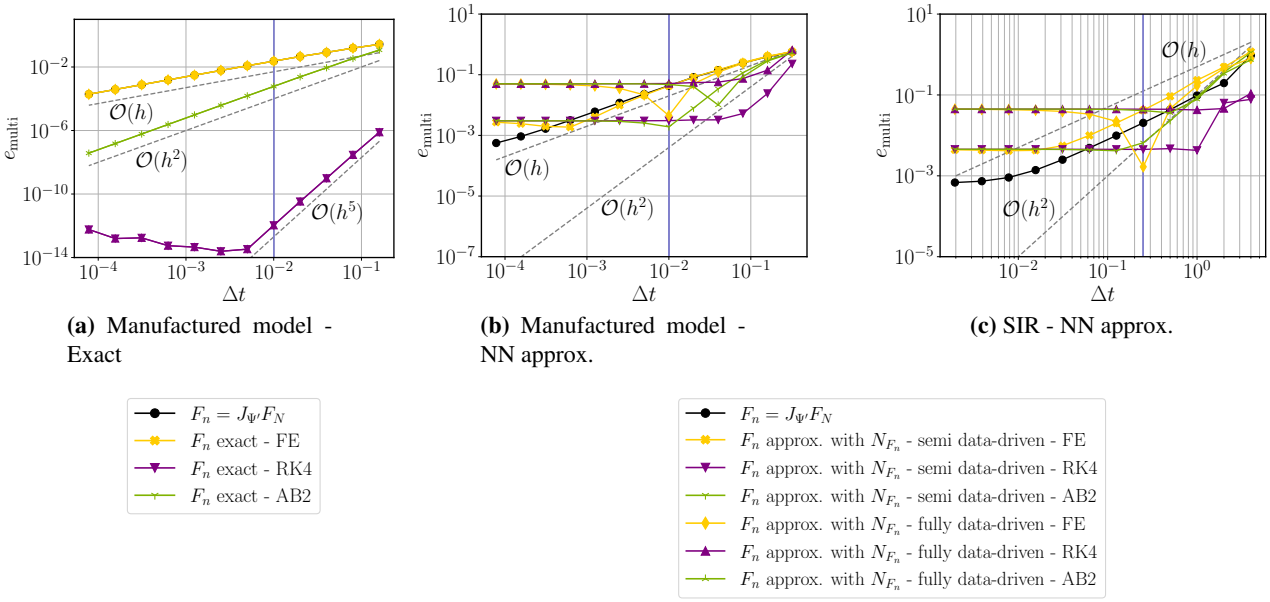
The trained neural networks are employed in combination with different time-integration schemes, possibly different from the one used during training. Specifically, we consider FE (black and yellow lines), the linear multistep Adams–Bashforth 2 method (AB2, green lines), and, for completeness, the multi-stage Runge–Kutta 5 method (RK5, purple lines). The training timestep size, Δt_{train}, is also indicated by a vertical line.

For the analytical case (panel (a)), we observe that the convergence order is the expected one and that the error steadily decreases, reaching very small values for RK5, where the observed stagnation can be attributed to truncation errors. Note that, in panel (a), due to the use of exact Ψ', Ψ, and F_n, some scenarios collapse onto a single line.

Conversely, when these quantities are approximated using neural networks (panel (b)), the error exhibits stagnation to higher values than that of panel (a), indicating that the approximation error introduced by the neural networks becomes dominant. It is important to note that the semi data-driven training strategy allows for improving the accuracy of the method by reducing the timestep during the online phase to values *smaller* than those used during training, therefore enabling temporal over-resolution.

Within the fully data-driven strategy, instead, no improvement is observed when decreasing the timestep. Moreover, the error behavior displays a local minimum at the training timestep value, which

can be explained by the fact that N_{F_n} approximates F_n together with correction terms, some of which depend on the specific Δt_{train}, see (6.7).



(a) Manufactured model - Exact

(b) Manufactured model - NN approx.

(c) SIR - NN approx.

Figure 8. Time convergence for different strategies and integration schemes is shown. (a) the N-ODE is defined in (7.2). The reduced order model is constructed using analytical Ψ', Ψ, F_n. (b) the N-ODE is defined in (7.2). The reduced order model is constructed using trained $N_{\Psi'}$, N_{Ψ}, N_{F_n}. (c) SIR model, see (4.6). The observed orders of convergence match the expected ones up to a plateau, which arises due to the approximation error introduced by the neural networks.

7.2.2. Generic case – SIR

To strengthen the discussion on time convergence, we present here a case for which the analytical expressions of Ψ', Ψ, and F_n are not available. As a model case, we consider the SIR system (4.6), with β treated as a parameter. The main settings are as follows: $\beta \sim \mathcal{U}(0.5, 2.5)$, $\gamma = 0.5$, $N = 100$, and initial condition $u_{N0} = [S_0, I_0, R_0]^T = [90, 10, 0]^T$. The full dataset comprises 300 trajectories in the time range $[0, T]$, $T = 20$, each corresponding to a different realization of β. These data are partitioned into 200 samples for training, 50 for validation , and 50 for testing. Each trajectory is computed using the Runge–Kutta 4(5) method [6], as implemented in Scipy [67], with adaptive time-stepping and stringent tolerances: atol $= 10^{-16}$ and rtol $= 10^{-12}$. These settings ensure that the time discretization error of the N-ODE can be neglected.

The neural networks' architectures are summarized in Appendix B. The training is carried out adopting the FE scheme for all the three aforementioned options for 2000 epochs using a minibatch size of 32. The learning rate is set to 10^{-3} for epochs < 500, reduced to 10^{-4} for $500 \leq$ epochs < 1500, and further to 10^{-5} thereafter. Across all strategies, training decreases the loss function by approximately 6-7 orders of magnitude relative to its initial value, typically reaching a minimum around 10^{-3}. Given that the loss function is the mean squared error, this leads to an estimation of the relative error in u_N

and F_n of the order of magnitude of 10^{-3}. Training is performed on data sampled with timestep size $\Delta t_{\text{train}} = 10^{-3}$. As in the previous case, during the online phase, the timestep size Δt can be varied.

In Figure 8 panel (c), we report the values of e_{multi} for the three strategies "Exact rhs", "Semi data-driven", and "Fully data-driven" as previously mentioned. The trained neural networks are used in combination of different schemes, possibly different from that used during the training: FE, AB2, and, for completeness, RK5. The training timestep size, Δt_{train}, is also indicated with a vertical line. First, we observe that with strategy 1 (black line), the error exhibits first-order convergence up to a stagnation point. This plateau can be explained with a balance between the approximation error of the neural network and the time discretization error, as predicted by (6.18). Looking at the yellow lines (FE), strategy 3 displays a minimum error near the training timestep size. This minimum is also close to the lowest value across all configurations. Reducing Δt further leads to an increase in the error. This behavior is expected, as N_{F_n} approximates F_n together with correction terms, some of which depend on the specific Δt_{train}, see (6.7). This phenomenon disappears when adopting strategy 2. Indeed, its corresponding line shows a higher error at Δt_{train} but continues to decrease with order 1 until reaching a stagnation point for smaller timestep sizes.

We now consider the green lines (AB2). As expected, we observe second-order convergence, and the minimum error is comparable to that obtained with FE. However, since N_{F_n} is trained to minimize (6.5) using the FE difference operator, the error minimum valley around Δt_{train} does not appear.

Regarding the RK5 scheme (purple lines), we observe that, due to its higher order of convergence, the overall error remains dominated by the neural network approximation even for relatively large timestep sizes. Moreover, the stagnation values coincides with those of the other methods, namely FE and AB2.

We emphasize that e_{multi} is computed on the test dataset, so unseen data, thus confirming the stability of the results, the attainment of the expected convergence order, and the generalization capability of the trained models.

7.3. Test 3: chemical reactions kinetic

In this case, we investigate a generic transient system of chemical reactions [23, 63]. This test case serves to demonstrate the application of the DRE to a general nonlinear ODE. The main challenges arise from the strong nonlinearities of the N-ODE and from the larger value of N compared to the previously analyzed cases.

We consider a total of $n_r = 6$ generic forward and backward reactions involving chemical species A_i, for $i = 1, \dots, n_s$, with $n_s = 19$ denoting the total number of species. A generic reaction takes the following form:

$$\sum_{i \in \overline{\mathcal{K}}} s_{ij} A_i \rightarrow \sum_{k \in \mathcal{K}} -s_{kj} A_k, \quad \forall j = 1, \dots, n_r.$$

Here, $\overline{\mathcal{K}}$ and $\mathcal{K}$ represent the sets of indices corresponding to reactants and products, respectively, and s_{ij} denote the stoichiometric coefficients. We denote by $u_N : [0, T] \rightarrow \mathbb{R}^{n_s}$ the concentration vector of the chemical species, by $r : \mathbb{R}^{n_r} \rightarrow \mathbb{R}^{n_r}$ the reaction rate vector, and by $S \in \mathbb{R}^{n_s \times n_r}$ the stoichiometric

matrix, whose entries are the coefficients s_{ij}. We choose the following S:

$$S^\top = \begin{bmatrix} 1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & 0 \\ -1 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & 1 \\ 0 & -1 & 1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 1 & -1 \end{bmatrix}.$$

The time evolution of the concentrations is governed by the following N-ODE:

$$\begin{aligned} \dot{u}_N &= S\, r(u_N), \\ u_N(0) &= [0.79, \mu_1, 0.92, 0.41, 0.32, 0.39, 0.68, 0.89, 0.02, 0.28, 0.58, 0.94, 0.1, 0.9, 0.83, 0.72, 0.72, 0.50, 0.84]. \end{aligned} \tag{7.3}$$

The first parameter of this problem is $\mu_1 \sim \mathcal{U}(0.74, 0.94)$, which influences the initial condition of the second chemical species. For the reaction rates, we employ the following model:

$$r_j = k_j \prod_{i=1}^{n_s} [u_N]_i^{|S_{ij}| \cdot \mathbf{1}_{S_{ij}<0}} \quad \text{for } j = 1, \dots, n_r,$$

where $\mathbf{1}_{S_{ij}<0}$ denotes the indicator function, which equals 1 if $S_{ij} < 0$ and 0 otherwise, and k_j is the forward rate constant for the j-th reaction. The latter is typically subject to uncertainties in real experimental settings. To account for this, we treat the second component of the vector k as the second parameter of the problem, denoted by μ_2, and assume it follows the distribution $\mu_2 = k_2 \sim \mathcal{U}(5, 11)$. We assume the parameters to be independently distributed. The forward rate vector is given by $k = [5, \mu_2, 5, 5, 5, 5]$.

The DRE is trained using a fully data-driven strategy, relying on the Adams–Bashforth 2 integration scheme for the loss contribution (6.5).

We are dealing with a conservative system, which means that the following equation should be satisfied at all times

$$\sum_{k=1}^{N} [u_N(t)]_k = \sum_{k=1}^{N} [u_{N0}]_k, \quad \forall t \in [0, T].$$

It is therefore relevant, in this scenario, that the reduced order model preserves this property. To this aim, a simple strategy is to modify the loss function by adding a contribution whose minimization enforces conservation:

$$\mathscr{N}_{ij}^{\text{cons}} = \left\| \sum_{k=1}^{N=19} [u_{N,i}^j]_k - [N_\Psi(\mathfrak{u}_{n,i}^j)]_k \right\|_2^2.$$

The training dataset consists of 200 samples uniformly distributed in the parameter space, with 100 time steps per sample. The validation and test datasets each contain 10% of the number of samples used for training.

After training, the quality of the DRE is assessed by computing the time-dependent average relative

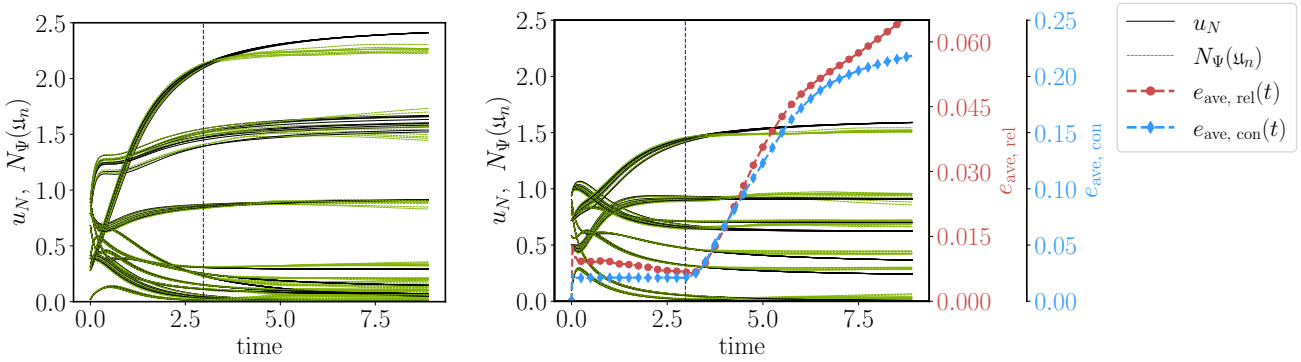
error, $e_{\text{ave, rel}}(t)$, and the average of discrepancy of the total concentration, $e_{\text{ave, con}}(t)$:

$$
\begin{aligned}
e_{\text{ave, rel}}(t_j) &= \left(\frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \frac{\|u_{N,i}(t_j) - N_\Psi(\mathfrak{u}_{n,i}^j)\|_2^2}{\|u_{N,i}\|_2^2} \right)^{1/2}, \\
e_{\text{ave, con}}(t_j) &= \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \sum_{k=1}^{N=19} [u_{N,i}(t_j)]_k - [N_\Psi(\mathfrak{u}_{n,i}^j)]_k,
\end{aligned}
\tag{7.4}
$$

where $j = 0, 1, \ldots$ and $t_j = j\Delta t$.

Figure 9 shows the reference concentrations of each component,$[u_N]_i$, $i = 1, \ldots, n_s$, obtained by solving (7.3) with the Runge-Kutta 4(5) method implemented in SciPy [67], using strict tolerances (atol $= 10^{-16}$, rtol $= 10^{-12}$), represented by solid black lines. The reconstructed solution, $N_\Psi(\mathfrak{u}_n)$, which is computed using a timestep size $\Delta t = 0.025$, is shown with dashed green lines. The reduced model is also queried at future, unseen times, $3 < t < 8.9$, although training was performed only up to $t = 3$. We observe that both the conservation error and the relative error increase steadily at later times. Nevertheless, despite the evaluation on unseen data, the DRE provides reliable results for this test case.

In the right panel of the same figure, the average relative error, $e_{\text{ave, rel}}$, is also displayed to provide a quantitative assessment of the DRE's accuracy. In the training time range, the error is low, reaching a peak of less than 2%. This holds even during the initial phase, where the concentrations vary rapidly. For unseen times, the error increases with a fairly linear trend reaching a maximum of about 4%. In the right panel of Figure 9 also $e_{\text{ave, con}}$ is represented. We observe a consistent overestimation of the total concentration. However, this discrepancy is small, as the total concentration is approximately 11.67, and the deviation affects only the third significant digit.



(a) chemical species from 1 to 10

(b) chemical species from 11 to 19

Figure 9. Time evolution of the components of u_N. Solid black lines represent the reference solution obtained by accurately solving the N-ODE; dashed green lines show the reconstructed solution $N_\Psi(\mathfrak{u}_n)$. The two solutions exhibit good agreement visually. The training data, and hence the learned manifold, are restricted to the interval $t \leq 10$, which is marked by a vertical yellow dashed line. The red and blue dashed lines indicate the relative and conservation errors, as defined in (7.4).

Figure 10 shows the set $\mathfrak{u}_n(t;\mu)$ obtained for μ belonging in the training dataset values. The set

is partitioned into two subsets with different color scales: one corresponding to the n-solutions the training time interval $t \in [0, 3]$, called U_t, and the other corresponding to n-solution obtained for future times $t \in (3, 6]$, called U_e. The two subsets are disjoint, indicating that N_{F_n} is operating on unseen input during the training, those for future times. This highlights the challenge of time extrapolation. Indeed, N_{F_n} is trained to approximate $F_n : U_t \to \mathbb{R}^n$ but subsequently evaluated on a larger domain $U_e \supset U_t$. Nonetheless, in this particular case, the extrapolated predictions remain reliable, as also shown in Figure 9. A more accurate extrapolation in time would be feasible if u_n at future times remained within the training region as, for example, in the case where the solution of the N-ODE is periodic, so a training using data sampled in one period of time would be sufficient to characterize virtually all the values of u_n.

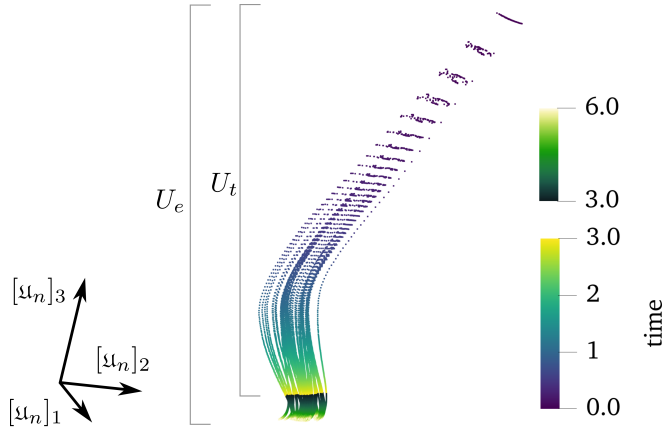


Figure 10. Set of n-solutions, $\mathfrak{u}_n^k$, at timesteps $k = 0, \ldots, K$, with μ sampled from the training dataset, along with the extrapolated solutions for unseen future times.

7.4. Test 4: PDEss – Burgers' model

The purpose of this test is to illustrate an application to a PDE. We consider here the one-dimensional viscous Burgers' model:

$$\begin{aligned} &\dot{u} + u\frac{\partial u}{\partial x} - \mu_1 \frac{\partial^2 u}{\partial x^2} = 0, \qquad && t \in (0, T],\ x \in (a, b), \\ &u(0, x) = \sin(2\pi\mu_2(x - a)/(b - a)) + 1, \end{aligned} \tag{7.5}$$

where $\mu_1 \sim \mathcal{U}(0.0001, 0.001)$ denotes the viscosity coefficient, $\mu_2 \sim \mathcal{U}(1, 1.5)$ is the parameter affecting the initial conditions, and $a = 0$, $b = 1$, $T = 1$. Periodic boundary conditions are imposed at the boundaries. The viscosity is chosen to be small, which ensures the continuity of the solution while it may induce the formation of strong spatial and temporal gradients.

The PDE (7.5) is discretized in space using centered finite-difference schemes for both the advection and diffusion terms, on a fine grid consisting of 1000 nodes over $[a, b]$. We emphasize that the spatial discretization itself is not the focus of this test, rather, the aim is to demonstrate a possible application and effectiveness of the reduced model to a PDE setting. The resulting system of ODEs is then integrated in time using a fifth-order Runge–Kutta method with a timestep size of 0.005. For

simplicity, the discrete solution is subsequently downsampled in space to 50 points. Therefore, the N-ODE has size $N = 50$.

For a qualitative visualization of the underlying dynamics, see Figure 11, which shows three time snapshots of (7.5) for two different parameter configurations.

The neural network architectures are reported in Table B5. They are trained on a dataset consisting of 200 parameter instances and tested on an independent dataset of 50 points. The fully data-driven training strategy is adopted, with a total of 500 epochs.

After training, the solution obtained from the DRE is compared against the full-order model. A first qualitative comparison is shown in Figure 11, where an excellent agreement between the two solutions can be observed.

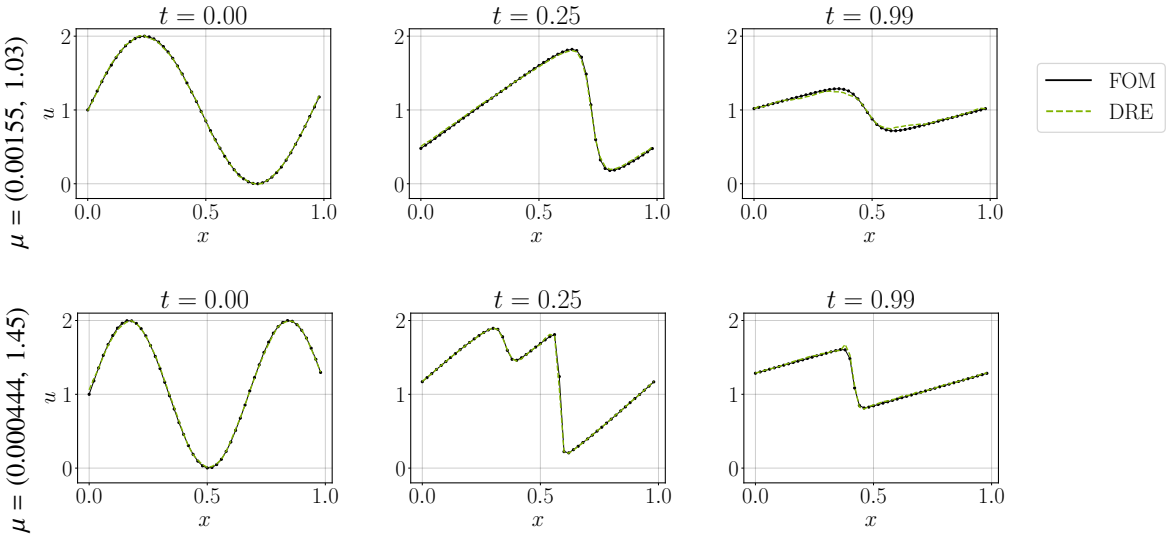
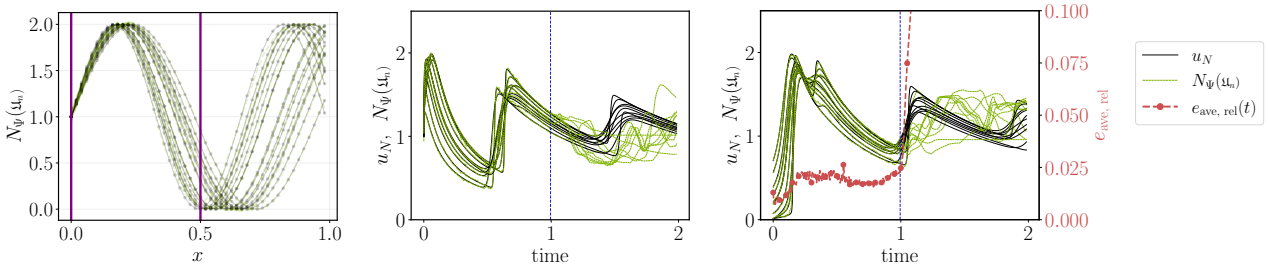


Figure 11. u_N and $\mathfrak{u}_N$ at three time steps for two values of μ. This figure highlights the variability of the dataset and the strong agreement between the FOM and DRE solutions, despite the strong gradients.

The average relative error, $e_{\text{ave, rel}}$, defined in (7.4), is reported in the right-most panel of Figure 12. The results indicate a good level of accuracy, with an error of approximately 2% over the entire training time interval. In contrast, the error increases rapidly for later times. This behavior can be explained by the fact that the functions are approximated over a prescribed domain but subsequently evaluated outside this domain, as discussed in Section 7.3 and illustrated in Figure 10.

Furthermore, in Figure 12, we also report the initial conditions for selected values of μ_2, together with the time evolution of two components of $\mathfrak{u}_N$, corresponding to $x = 0$ and $x = 0.5$, for the same parameter value. For graphical clarity, only two components are displayed. The strong nonlinearity of the problem is evident from the rapid temporal variations of the solution. Nevertheless, the reduced-order model is able to accurately track the time evolution.



(a) Initial conditions **(b)** $\mathfrak{u}_N|_{x=0}$ **(c)** $\mathfrak{u}_N|_{x=0.5}$ and error

Figure 12. Panel **(a)**: initial conditions for varying values of μ_2 and, indicated by vertical purple lines, the components of $N_\Psi(\mathfrak{u}_N)$ displayed in the adjacent panels. Panels **(b)** and **(c)**: time evolution of selected components of u_N. Solid black lines denote the reference solution obtained by accurately solving the N-ODE, while dashed green lines represent the reconstructed solution $N_\Psi(\mathfrak{u}_n)$. The two solutions show good visual agreement. The training data, and consequently the learned manifold, are restricted to the time interval $t \leq 1$, which is indicated by a vertical blue dashed line. Panel **(c)**: the red dashed line indicates the relative error, as defined in (7.4).

7.5. Discussion on computational costs

In this section we comment on the computational cost of the reduced-order model and its different training strategies. Table 1 provides a concise overview of the main steps, from data generation to the evaluation of the reduced model.

Table 1. Main steps for the offline and online phases with associated computational costs. The number of training epochs is n_e. We recall that P is the number of steps of the multistep methods utilized.

	Common	**Semi-datadriven**	**Fully-datadriven**
offline	• If PDE, space discretization and assembly of ODE. • One timestep evolution of N-ODE using explicit or implicit method. • Repeat the previous point for N_{time} times. • Repeat for every parameter. If the problem is non-affine, repeat assembly of the ODE and following steps for each parameter.	• Training $N_{\Psi'}$, N_Ψ. Cost = $O(n_e N^2)$. • Training N_{F_n}. Cost = $O(n_e N^2)$.	• Training $N_{\Psi'}$, N_Ψ, and N_{F_n} concurrently. Cost = $O(n_e N^2)$.
online	• Reduction of initial condition: $\mathfrak{u}_{n0} = N_{\Psi'}(u_{N0})$. Cost = $O(N^2)$. • Evolve in time n-ODE. Cost = $O(PN_{\text{time}}n^2)$. • Reconstruction $\mathfrak{u}_N = N_\Psi(\mathfrak{u}_n)$ for preselected $\overline{m}$ timesteps. Cost = $O(\overline{m}N^2)$.		

In broad terms, the generation and use of the reduced-order model are organized into offline and

online phases. The former includes data generation and network training, while the latter concerns the actual use of the reduced model. In general, the offline phase is the most expensive one, and the overall effectiveness of the reduced model strongly depends on the objective of the task at hand and, in particular, for how many new values of the parameters the reduced model will be evaluated. An example of a multi-query analysis is the gradient-free optimization procedure, where repeated evaluations of the model for a large number of parameter values are required [4].

We begin by discussing some aspects of the offline phase. Since the method is data-driven, dataset generation is necessary. Its cost strongly depends on the specific problem under consideration and on the purpose of the analysis. When the problem of interest is a PDE, as in the example discussed in Section 7.4, the initial step consists in the construction of the N-ODE, that is, the discretization of the problem with respect to all variables except the time, which typically correspond to the spatial coordinates. This step can be particularly expensive, especially when the problem is non-affine, as this requires the reconstruction of the N-ODE for each parameter value; see, among others, [3, 4, 32].

Once the N-ODE has been constructed, it must be solved over the prescribed time interval and for a sufficiently large number of parameter samples in order to guarantee an adequate training of the neural networks. The number of samples is problem-dependent and is dictated by the desired accuracy of the results. Time integration must be performed sequentially, and each single timestep evaluation has a cost which depends on the numerical solver employed. A major influence on this cost arises from the choice between explicit and implicit schemes: the former are typically faster but require small timestep size for stability reasons, whereas the latter are generally more expensive per step but allow for larger timestep size.

Based on the numerical tests presented in this section, the number of timesteps used to generate the full-order solutions appears to be abundant compared to those effectively required for accurately training the neural networks in the desired time range; indeed, a downsampling of the timesteps could be beneficial. Therefore, the reduced model does not seem to impose any limitation on the time evolution of the full-order model, as the settings used to properly solve the full-order model already provide sufficient data to accurately train the networks.

The construction of the reduced model proceeds with the training of the neural networks. The main factors influencing the training time are the desired accuracy, which is directly related to the number of epochs, and the network architecture (smaller networks generally leading to faster training), the optimization algorithm, and the chosen training strategy, namely semi-datadriven versus fully-datadriven. While the former offers the important advantage of accurately approximating F_n, enabling time super-resolution, it is intrinsically slower than the fully-datadriven approach. Indeed, in the semi-datadriven setting the autoencoder and the network N_{F_n} must be trained in two distinct phases: first the autoencoder, and subsequently N_{F_n}, as detailed in Section 6.1. In practice, depending on the numerical cases presented, the semi-datadriven approach requires a computational time that is, roughly speaking, about twice that of the fully-datadriven one.

Once the offline phase is completed, the online phase involves comparatively inexpensive computations. In particular, it includes the time evolution of the n-ODE, which is computationally cheap due to the small dimension n, even when small timesteps are employed. The time evolution starts from the initial condition, which is obtained by a feed-forward evaluation of the encoder network applied to u_{N0}. The high-dimensional solution is then reconstructed through feed-forward evaluations of the decoder network to obtain $\mathfrak{u}_N$ at all, or possibly only $\overline{m} \leq N_{\text{times}}$, timesteps. This task is trivially

parallelizable, and its computational time is typically negligible when compared to the overall time.

Finally, we highlight that the computational cost of the networks is affected by the dimensions N and n. In particular, the cost, C, of the feed-forward evaluation of a generic network scales as $C = O\left(\sum_{\ell=1}^{L} n_\ell n_{\ell-1}\right)$. For the encoder, we have $n_0 = N$ and, unless the network is linear, possibly some intermediate layers of comparable size. The final layer has dimension n. Similarly, for the decoder we have $n_L = N$, together with possibly other preceding layers of similar size. It follows that, for both the encoder and the decoder, the computational cost depends on both N and n: $C_{\text{enc}} = C_{\text{enc}}(N, n) = O(N^2)$ (or $O(Nn)$ in the linear case), $C_{\text{dec}} = C_{\text{dec}}(N, n) = O(N^2)$ (or $O(Nn)$ in the linear case).

Conversely, the layers of N_{F_n} depend only on the reduced dimension n and the number of parameters, since $n_0 = n + a$ and $n_L = n$. Therefore, the cost associated with N_{F_n} is $C_{F_n} = C_{F_n}(n) = O(n^2)$.

Since backpropagation has a computational cost of the same order of magnitude as the feed-forward evaluation, the training of the autoencoder is intrinsically limited by the problem size, scaling as N^2 (or Nn in the linear case). This step therefore represents the most computationally expensive component of the proposed method. The reader is referred to Table 1 for an indication of the order of magnitude of the computational costs associated with each step involving the neural networks.

8. Conclusions

In this work, we have presented a data-driven reduced-order modeling method entirely based on neural networks. The methodology can be applied to generic ODEs, such as those resulting from the spatial discretization of PDEs as presented in the last test case in Section 7.4.

Several contributions have been made. First, we proved that the set of solutions corresponding to varying parameters forms a m-manifold, which can be parametrized using $n \geq m$ coordinates. We also established a theoretical link between the number of parameters and the minimum latent dimension required for an exact representation.

Second, we provided guidance on the construction of an autoencoder composed of fully connected neural networks, ensuring that no information is lost during the dimensionality reduction phase, i.e., achieving $\varepsilon_1 = 0$.

Third, we identified the n-ODE that governs the reduced dynamics and proved its well-posedness. This system can be solved rapidly in place of the original N-ODE, which is potentially expensive. The relationship between these two systems requires particular treatment, and, consequently, we focused on establishing conditions for stability and convergence such as $\Delta t_{\text{train}}, \Delta t \to 0$ and $|w| \to \infty$, accounting for both time discretization errors and neural network approximation errors.

Fourth, we proposed and analyzed two training strategies, each with distinct strengths and limitations. It is worth highlighting that the semi-data-driven approach enables fine temporal resolution after training, i.e., it allows for $\Delta t < \Delta t_{\text{train}}$.

The methodology was assessed in four test cases: One illustrates stability properties, one shows convergence over time, one evaluates performance in a general application setting, and the last shows an application to a highly nonlinear PDE resulting in strongly nonlinear solution manifold. The numerical results are in agreement with the theoretical findings. In particular, due to the nonlinear nature of both the encoder and decoder, the method is capable of accurately reconstructing the nonlinear N-solution by solving the smallest ODE that allows for a null representation error.

The solid theoretical foundation and promising numerical performance justify the continuation of the research of the presented framework. In the following, we mention some possible future research direction and limitations. We have shown that the injectivity of the solution map is crucial for the representation of $\mathrm{Id}_{\mathcal{M}}$ and that, if absent, it can be recovered by augmenting the data. This observation motivates future investigation of complex manifolds arising from non-injective parameter-to-solution maps.

Moreover, given the analytical complexity of the full stability theory, we have restricted our analysis to relatively simple contexts. An extension of the stability results to more general settings, such as linear multistep or Runge–Kutta schemes, and the definition of the relation autoencoder vs. stability appears particularly interesting. Finally, given the effectiveness of the method, it is interesting to apply it to realistic and engineering-relevant scenarios (see, e.g., [2]) where it is possible to properly assess the effectiveness of the method in terms of computational cost and speedup.

Use of Generative-AI tools declaration

The authors declare that Artificial Intelligence (AI) tools were used only to assist with LaTeX template formatting and English-language editing. The authors are responsible for all scientific content, results, and conclusions of the manuscript.

Acknowledgments

Enrico Ballini and Paolo Zunino acknowledge the partial support the European Union's Euratom research and training programme 2021–2027 under grant agreement No. 101166699 *Radiation safety through biological extended models and digital twins* (TETRIS). This research is part of the activities of the *Dipartimento di Eccellenza* 2023–2027, Department of Mathematics, Politecnico di Milano. Enrico Ballini, Alessio Fumagalli, Luca Formaggia, Anna Scotti, and Paolo Zunino are members of the INdAM Research group GNCS.

Conflict of interest

Prof. Paolo Zunino is an editorial board member for Mathematics in Engineering and was not involved in the editorial review and/or the decision to publish this article.

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The code is publicly available at `https://github.com/enricoballini/dynamical_reduced_embedding.git`.

References

1. J. Ansel, E. Yang, H. He, N. Gimelshein, A. Jain, M. Voznesensky, et al., PyTorch 2: faster machine learning through dynamic Python bytecode transformation and graph compilation, *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, **2** (2024), 929–947. https://doi.org/10.1145/3620665.3640366
2. E. Ballini, A. Cominelli, L. Dovera, A. Forello, L. Formaggia, A. Fumagalli, et al., Enhancing computational efficiency of numerical simulation for subsurface fluid-induced deformation using deep learning reduced order models, *SPE Reservoir Simulation Conference*, 2025. https://doi.org/10.2118/223897-MS
3. E. Ballini, A. Cominelli, L. Dovera, A. Forello, L. Formaggia, A. Fumagalli, et al., Enhancing computational efficiency of numerical simulation for subsurface fluid-induced deformation using deep learning reduced order models, *Comput. Geosci.*, **30** (2026), 14. https://doi.org/10.1007/s10596-026-10409-6
4. E. Ballini, L. Formaggia, A. Fumagalli, A. Scotti, P. Zunino, Application of deep learning reduced-order modeling for single-phase flow in faulted porous media, *Comput. Geosci.*, **28** (2024), 1279–1303. https://doi.org/10.1007/s10596-024-10320-y
5. J. Barnett, C. Farhat, Y. Maday, Neural-network-augmented projection-based model order reduction for mitigating the Kolmogorov barrier to reducibility, *J. Comput. Phys.*, **492** (2023), 112420. https://doi.org/10.1016/j.jcp.2023.112420
6. J. C. Butcher, *Numerical methods for ordinary differential equations*, John Wiley & Sons, Ltd., 2016. https://doi.org/10.1002/9781119121534
7. Y. Cai, Achieve the minimum width of neural networks for universal approximation, *arXiv*, 2022. https://doi.org/10.48550/arXiv.2209.11395
8. P. Y. Chen, J. Xiang, D. H. Cho, Y. Chang, G. A. Pershing, H. T. Maia, et al., CROM: Continuous reduced-order modeling of PDEs using implicit neural representations, *International Conference on Learning Representations (ICLR)*, 2023, 1–41.
9. R. T. Q. Chen, B. Amos, M. Nickel, Learning neural event functions for ordinary differential equations, *arXiv*, 2020. https://doi.org/10.48550/arXiv.2011.03902
10. R. T. Q. Chen, Y. Rubanova, J. Bettencourt, D. K. Duvenaud, Neural ordinary differential equations, In: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett, *Advances in neural information processing systems*, 32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Vol. 31, 2018.
11. P. Conti, M. Guo, A. Manzoni, A. Frangi, S. L. Brunton, J. Nathan Kutz, Multi-fidelity reduced-order surrogate modelling, *Proc. A*, **480** (2283), 2024. https://doi.org/10.1098/rspa.2023.0655
12. Q. Du, Y. Gu, H. Yang, C. Zhou, The discovery of dynamics via linear multistep methods and deep learning: error estimation, *SIAM J. Numer. Anal.*, **60** (2022), 2014–2045. https://doi.org/10.1137/21M140691X
13. R. D'Ambrosio, S. D. Giovacchino, Mean-square contractivity of stochastic theta-methods, *Commun. Nonlinear Sci. Numer. Simul.*, **96** (2021), 105671. https://doi.org/10.1016/j.cnsns.2020.105671

14. N. Farenga, S. Fresca, S. Brivio, A. Manzoni, On latent dynamics learning in nonlinear reduced order modeling, *Neural Networks*, **185** (2025), 107146. https://doi.org/10.1016/j.neunet.2025.107146

15. D. Floryan, M. D. Graham, Data-driven discovery of intrinsic dynamics, *Nat. Mach. Intell.*, **4** (2022), 1113–1120. https://doi.org/10.1038/s42256-022-00575-4

16. N. R. Franco, S. Fresca, F. Tombari, A. Manzoni, Deep learning-based surrogate models for parametrized PDEs: Handling geometric variability through graph neural networks, *Chaos*, **33** (2023), 123121. https://doi.org/10.1063/5.0170101

17. N. R. Franco, A. Manzoni, P. Zunino, A deep learning approach to reduced order modelling of parameter dependent partial differential equations, *Math. Comput.*, **92** (2022), 483–524.

18. N. R. Franco, A. Manzoni, P. Zunino, Mesh-informed neural networks for operator learning in finite element spaces, *J. Sci. Comput.*, **97** (2023), 35. https://doi.org/10.1007/s10915-023-02331-1

19. S. Fresca, L. Dede', A. Manzoni, A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs, *J. Sci. Comput.*, **87** (2021), 61. https://doi.org/10.1007/s10915-021-01462-7

20. S. Fresca, F. Fatone, A. Manzoni, Long-time prediction of nonlinear parametrized dynamical systems by deep learning-based reduced order models, *Math. Eng.*, **5** (2023), 1–36. https://doi.org/10.3934/mine.2023096

21. S. Fresca, A. Manzoni, POD-DL-ROM: Enhancing deep learning-based reduced order models for nonlinear parametrized PDEs by proper orthogonal decomposition, *Comput. Methods Appl. Mech. Eng.*, **388** (2022), 114181. https://doi.org/10.1016/j.cma.2021.114181

22. R. Fu, D. Xiao, I. Navon, F. Fang, L. Yang, C. Wang, et al., A non-linear non-intrusive reduced order model of fluid flow by auto-encoder and self-attention deep learning methods, *Int. J. Numer. Methods Eng.*, **124** (2023), 3087–3111. https://doi.org/10.1002/nme.7240

23. A. Fumagalli, A. Scotti, A mathematical model for thermal single-phase flow and reactive transport in fractured porous media, *J. Comput. Phys.*, **434** (2021), 110205. https://doi.org/10.1016/j.jcp.2021.110205

24. F. J. Gonzalez, M. Balajewicz, Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems, *arXiv*, 2018. https://doi.org/10.48550/arXiv.1808.01346

25. I. Goodfellow, Y. Bengio, A. Courville, *Deep learning*, MIT Press, 2017.

26. R. Gupta, R. Jaiman, Three-dimensional deep learning-based reduced order model for unsteady flow dynamics with variable Reynolds number, *Phys. Fluids*, **34** (2022), 033612. https://doi.org/10.1063/5.0082741

27. I. Gühring, M. Raslan, Approximation rates for neural networks with encodable weights in smoothness spaces, *Neural Networks*, **134** (2021), 107–130. https://doi.org/10.1016/j.neunet.2020.11.010

28. E. Hairer, G. Wanner, *Solving ordinary differential equations II: stiff and differential-algebraic problems*, Springer Series in Computational Mathematics, Vol. 14, Springer Berlin, Heidelberg, 1996. https://doi.org/10.1007/978-3-642-05221-7

29. B. Hanin, M. Sellke, Approximating continuous functions by ReLU nets of minimal width, *arXiv*, 2017. https://doi.org/10.48550/arXiv.1710.11278

30. K. Hasegawa, K. Fukami, T. Murata, K. Fukagata, Machine-learning-based reduced-order modeling for unsteady flows around bluff bodies of various shapes, *Theor. Comput. Fluid Dyn.*, **34** (2020), 367–383. https://doi.org/10.1016/0378-4754(92)90035-F

31. K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, 770–778. https://doi.org/10.1109/CVPR.2016.90

32. J. S. Hesthaven, G. Rozza, B. Stamm, *Certified reduced basis methods for parametrized partial differential equations*, Springer Cham, 2016. https://doi.org/10.1007/978-3-319-22470-1

33. T. Kadeethum, F. Ballarin, Y. Choi, D. O'Malley, H. Yoon, N. Bouklas, Non-intrusive reduced order modeling of natural convection in porous media using convolutional autoencoders: comparison with linear subspace techniques, *Adv. Water Resour.*, **160** (2022), 104098. https://doi.org/10.1016/j.advwatres.2021.104098

34. P. Kidger, *On neural differential equations*, Ph.D. Thesis, University of Oxford, 2021.

35. D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, *arXiv*, 2014. https://doi.org/10.48550/arXiv.1412.6980

36. A. Kumar, R. Hu, S. D. C. Walsh, Development of reduced order hydro-mechanical models of fractured media, *Rock Mech. Rock Eng.*, **55** (2021), 235–248. https://doi.org/10.1007/s00603-021-02668-9

37. J. D. Lambert, *Numerical methods for ordinary differential systems*, Wiley, 1991.

38. J. M. Lee, *Introduction to topological manifolds*, New York: Springer, 2000.

39. K. Lee, K. T. Carlberg, Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders, *J. Comput. Phys.*, **404** (2020), 108973. https://doi.org/10.1016/j.jcp.2019.108973

40. C. Legaard, T. Schranz, G. Schweiger, J. Drgoňa, B. Falay, C. Gomes, et al., Constructing neural network based models for simulating dynamical systems, *ACM Comput. Surv.*, **55** (2023), 1–34. https://doi.org/10.1145/3567591

41. F. Li, Y. Zhang, C. Xiao, Surrogate-based hydraulic fracture propagation prediction using deep neural network proxy, *58th U.S. Rock Mechanics/Geomechanics Symposium*, Golden, Colorado, 2024. https://doi.org/10.56952/ARMA-2024-0056

42. L. Li, Y. Duan, G. Ji, Y. Cai, Minimum width of leaky-ReLU neural networks for uniform universal approximation, *arXiv*, 2023. https://doi.org/10.48550/arXiv.2305.18460

43. Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, et al., Fourier neural operator for parametric partial differential equations, *arXiv*, 2020. https://doi.org/10.48550/arXiv.2010.08895

44. Z. Li, S. Patil, F. Ogoke, D. Shu, W. Zhen, M. Schneier, et al., Latent neural PDE solver: a reduced-order modeling framework for partial differential equations, *J. Comput. Phys.*, **524** (2025), 113705. https://doi.org/10.1016/j.jcp.2024.113705

45. J. Lu, Z. Shen, H. Yang, S. Zhang, Deep network approximation for smooth functions, *SIAM J. Math. Anal.*, **53** (2021), 5465–5506. https://doi.org/10.1137/20M134695X

46. L. Lu, P. Jin, G. Pang, Z. Zhang, G. E. Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, *Nat. Mach. Intell.*, **3** (2021), 218–229. https://doi.org/10.1038/s42256-021-00302-5

47. Y. Lu, A. Zhong, Q. Li, B. Dong, Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations, In: J. Dy, A. Krause, *Proceedings of the 35th International Conference on Machine Learning*, Volume 80: International Conference on Machine Learning, 2018, 3276–3285.

48. J. E. Marsden, T. J. R. Hughes, *Mathematical foundations of elasticity*, Dover Civil and Mechanical Engineering, Dover Publications, New York, 1994.

49. J. R. Munkres, *Topology*, Prentice Hall, Inc., 2000.

50. T. Murata, K. Fukami, K. Fukagata, Nonlinear mode decomposition with convolutional neural networks for fluid dynamics, *J. Fluid Mech.*, **882** (2019), A13. https://doi.org/10.1017/jfm.2019.822

51. J. M. Nordbotten, M. A. Celia, *Geological storage of CO_2 modeling approaches for large-scale simulation*, Wiley, 2011.

52. M. Ohlberger, S. Rave, Reduced basis methods: success, limitations and future challenges, *Proceedings of the Conference Algoritmy*, 2016, 1–12.

53. P. Pant, R. Doshi, P. Bahl, A. Barati Farimani, Deep learning for reduced order modelling and efficient temporal evolution of fluid simulations, *Phys. Fluids*, **33** (2021), 107101. https://doi.org/10.1063/5.0062546

54. S. Park, C. Yun, J. Lee, J. Shin, Minimum width for universal approximation, *arXiv*, 2020. https://doi.org/10.48550/arXiv.2006.08859

55. M. Pintore, B. Després, Computable lipschitz bounds for deep neural networks, *arXiv*, 2024. https://doi.org/10.48550/arXiv.2410.21053

56. A. Quarteroni, A. Manzoni, F. Negri, *Reduced basis methods for partial differential equations: an introduction*, Vol. 92, Springer Cham, 2016. https://doi.org/10.1007/978-3-319-15431-2

57. A. Quarteroni, R. Sacco, F. Saleri, *Numerical mathematics*, Vol. 37, Springer Berlin, Heidelberg, 2007. https://doi.org/10.1007/b98885

58. M. Raissi, P. Perdikaris, G. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.*, **378** (2019), 686–707. https://doi.org/10.1016/j.jcp.2018.10.045

59. F. Regazzoni, S. Pagani, M. Salvador, L. Dede', A. Quarteroni, Learning the intrinsic dynamics of spatio-temporal processes through latent dynamics networks, *Nat. Commun.*, **15** (2024), 1834 . https://doi.org/10.1038/s41467-024-45323-x

60. O. Ronneberger, P. Fischer, T. Brox, U-net: Convolutional networks for biomedical image segmentation, In: N. Navab, J. Hornegger, W. Wells, A. Frangi, *Medical image computing and computer-assisted intervention – MICCAI 2015*, Springer, Cham, **9351** (2015), 234–241. https://doi.org/10.1007/978-3-319-24574-4_28

61. Y. Rubanova, R. T. Q. Chen, D. Duvenaud, Latent ODEs for irregularly-sampled time series, *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019, 5320–5330.

62. A. Sabzi Shahrebabaki, S. Fouladi, E. Holtar, L. Vynnytska, Autoencoder-based generation of subsurface models, *Second EAGE Digitalization Conference and Exhibition*, European Association of Geoscientists & Engineers, **2022** (2022), 1–5. https://doi.org/10.3997/2214-4609.202239082

63. C. I. Steefel, A. C. Lasaga, A coupled model for transport of multiple chemical species and kinetic precipitation/dissolution reactions with application to reactive flow in single phase hydrothermal systems, *Amer. J. Sci.*, **294** (1994), 529–592.

64. P. A. Thompson, *Compressible-fluid dynamics*, McGraw-Hill, Inc., 1972.

65. T. Tripura, S. Chakraborty, Wavelet neural operator for solving parametric partial differential equations in computational mechanics problems, *Comput. Methods Appl. Mech. Eng.*, **404** (2023), 115783. https://doi.org/10.1016/j.cma.2022.115783

66. W. I. T. Uy, D. Hartmann, B. Peherstorfer, Operator inference with roll outs for learning reduced models from scarce and low-quality data, *Comput. Math. Appl.*, **145** (2023), 224–239. https://doi.org/10.1016/j.camwa.2023.06.012

67. P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, et al., SciPy 1.0: Fundamental algorithms for scientific computing in Python, *Nat. Methods*, **17** (2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2

68. G. Ziarelli, N. Parolini, M. Verani, Learning epidemic trajectories through kernel operator learning: from modelling to optimal control, *Numer. Math.*, **18** (2025), 286–324.

Appendix

A. Lipschitz constants and stability

Lipschitz constant L_{F_n}. Let $M_{y(x)} = \sup_x \|y(x)\|$ and L_y the Lipschitz constant of the generic function y. We have

$$
\begin{aligned}
&\|F_n(u_{n1}) - F_n(u_{n2})\| = \|J_{\Psi'}(\Psi(u_{n1}))F_N(\Psi(u_{n1})) - J_{\Psi'}(\Psi(u_{n2}))F_N(\Psi(u_{n2}))\| \\
&\le \|J_{\Psi'}(\Psi(u_{n1}))F_N(\Psi(u_{n1})) - J_{\Psi'}(\Psi(u_{n1}))F_N(\Psi(u_{n2})) + J_{\Psi'}(\Psi(u_{n1}))F_N(\Psi(u_{n2})) - J_{\Psi'}(\Psi(u_{n2}))F_N(\Psi(u_{n2}))\| \\
&\le \|J_{\Psi'}(\Psi(u_{n1}))[F_N(\Psi(u_{n1})) - F_N(\Psi(u_{n2}))]\| + \|F_N(\Psi(u_{n2}))[J_{\Psi'}(\Psi(u_{n1})) - J_{\Psi'}(\Psi(u_{n2}))]\| \\
&\le (M_{J_{\Psi'}} L_{F_N} + M_{F_N} L_{J_{\Psi'}})\|\Psi(u_{n1}) - \Psi(u_{n2})\| \\
&\le \underbrace{(M_{J_{\Psi'}} L_{F_N} + M_{F_N} L_{J_{\Psi'}})L_\Psi}_{=\overline{L}_{F_n,\mu}} \|u_{n1} - u_{n2}\|,
\end{aligned}
$$

where $\overline{L}_{F_n,\mu}$ is a upper bound for $L_{F_n,\mu}$.

Lipschitz constant L_{F_n}, scaled n-ODE. We scale the equations by $K > 0$:

$$
\begin{aligned}
K\dot{u}_n &= K J_{\Psi'} F_N(\Psi(u_n)), \\
K u_n(0) &= K\Psi'(u_{N0}).
\end{aligned}
\tag{A.1}
$$

Replacing $w_n = Ku_n$ into (A.1) we have the dynamics of the scaled n-solution:

$$\begin{aligned} \dot{w}_n &= KJ_{\Psi'}F_N(\Psi(1/Kw_n)) := \tilde{F}_n(w_n), \\ w_n(0) &= K\Psi'(u_{N0}), \end{aligned} \tag{A.2}$$

from which we derive:

$$\begin{aligned} \|\tilde{F}_n(w_{n,1}) - \tilde{F}_n(w_{n,2})\| &= K\|F_n(1/Kw_{n,1}) - F_n(1/Kw_{n,1})\| \\ &\leq KL_{F_n}\|1/Kw_{n,1} - 1/Kw_{n,2}\| \\ &= L_{F_n}\|w_{n,1} - w_{n,2}\|. \end{aligned}$$

Therefore $L_{\tilde{F}_n} = L_{F_n}$.

A modification of the Lipschitz constant can be made by using a nonlinear g, an invertible function with Jacobian J_g and such that $\Psi \circ g^{-1} \circ g \circ \Psi'$ represents the autoencoder $\Psi \circ \Psi'$ but passing through different n-solutions. Let now $w_n = g(u_n)$, then:

$$\begin{aligned} \dot{w}_n &= J_g F_n(g^{-1}(w_n)) := \tilde{F}_n(w_n), \\ w_n(0) &= g(\Psi'(u_{N0})), \end{aligned}$$

where, again, an abuse of notation is done in the definition of $\tilde{F}_n$. In general, depending on g, $L_{\tilde{F}_n}$ differs from L_{F_n}.

Lyapunov stability. First, we move the problem on the N-ODE to the n-ODE:

$$\|w_N - u_N\| \leq L_\Psi \|w_n - u_n\|.$$

Integrating the n-ODE and computing the difference between the reference solution and the perturbed one at time t reads:

$$w_n - u_n = \Psi'(u_{N0} + \Delta_0) + \delta_0 - \Psi'(u_{N0}) + \int_{t_0}^{t} \left(F_n(w_n) + J_{\Psi'}\Delta + \delta - F_n(u_n)\right).$$

Computing the norm and applying the triangular inequality:

$$\begin{aligned} \|w_n - u_n\| \leq \underbrace{\|\Psi'(u_{N0} + \Delta_0) - \Psi'(u_{N0}) + \delta_0\|}_{A} + \underbrace{\| \int_0^t J_{\Psi'}(u_N(t))\Delta\|}_{B} \\ + \underbrace{\| \int_0^t \delta\|}_{C} + \underbrace{\| \int_0^t F_n(w_n) - F_n(u_n)\|}_{D}. \end{aligned}$$

Each term can be bounded as follows:

$$\begin{aligned} A &\leq L_{\Psi'}\|\Delta_0\| + \|\delta_0\|, \\ B &\leq M_{J_{\Psi'}}\|\Delta\|t, \\ C &\leq \|\delta\|t, \end{aligned}$$

$$D \leq \int_0^t L_{F_n} \|w_n - u_n\|.$$

Applying the Grönwall's lemma and the Lipschitz continuity of Ψ, we get the bound

$$\|w_N - u_N\| \leq L_\Psi \Big(L_{\Psi'} \|\Delta_0\| + \|\delta_0\| + (M_{J_{\Psi'}} \|\Delta\| + \|\delta\|) t \Big) e^{L_{F_{n,\mu}} t}.$$

Note that the autoencoder affects $L_{F_{n,\mu}}$, see Proposition 6.

Lyapunov stability – Neural networks. Let us now consider an unperturbed N-ODE, so $\Delta = 0$ and $\Delta_0 = 0$. Interpreting w_n and the reconstructed $w_N = N_\Psi(w_n)$ as perturbed solutions due to the use of neural networks, we have

$$\begin{aligned} \|w_n - u_n\| \leq & \underbrace{\|\Psi'(u_{N0}) - \kappa(u_{n0}) - \Psi'(u_{N0})\|}_{E} \\ & + \underbrace{\| \int_0^t \omega(w_n) \|}_{F} + \| \int_0^t F_n(w_n) - F_n(u_n) \|. \end{aligned}$$

We can bound the first two terms with $E \leq \varepsilon_{\Psi'}$ and $F \leq \varepsilon_{F_n} t$. Therefore, we can apply the Grönwall's lemma. Using the Lipschitz continuity of Ψ we have

$$\begin{aligned} \|w_N - u_N\| &\leq L_\Psi \|w_n - u_n\| + \varepsilon_\Psi \\ &\leq L_\Psi \Big(\varepsilon_{\Psi'} + \varepsilon_{F_n} t \Big) e^{L_{F_n} t} + \varepsilon_\Psi. \end{aligned}$$

B. Neural networks' architectures

We report in Table B2 the neural networks' architectures used in Section 7.1. With "Linear(Affine) $x \times y$" we denote a linear(affine) transformation from $\mathbb{R}^x$ to $\mathbb{R}^y$.

Table B2. Neural networks' architectures used in Section 7.1. $|w|$ is the number of trainable weights for the single neural network, $|w|_{\text{tot}}$ the total number of trainable weights.

		layer	$\lvert w \rvert$	$\lvert w \rvert_{\textbf{tot}}$
Linear-linear	$N_{\Psi'}$	Linear 3×2	6	12
	N_Ψ	Linear 2×3	6	
Linear-nonlinear	$N_{\Psi'}$	Linear 3×2	6	7 678
	N_Ψ	Linear 2×3, PReLU Affine 3×30, PReLU 8×Affine 30×30, PReLU Affine 30×3, PReLU Affine 3×3	7 672	

We report in Table B3 the neural networks' architectures used in Section 7.2. N_{F_n} is trained using two different strategies, as detailed in Section 7.2.

Table B3. Neural networks' architectures used in Section 7.2. $|w|$ is the number of trainable weights for the single neural network, $|w|_{\text{tot}}$ the total number of trainable weights. The analytical and generic tests refer to Section 7.2.1 and Section 7.2.2, respectively.

		layer	$\lvert w\rvert$	$\lvert w\rvert_{\textbf{tot}}$
Analytical	$N_{\Psi'}$	Affine 3×10, ELU 3×Affine 10×10, ELU Linear 10×2	390	6 974
	N_{Ψ}	Linear 2×3, PReLU Affine 3×10, PReLU 4×Affine 10×10, PReLU Affine 10×3, PReLU Affine 3×3	427	
	N_{F_n}	Affine 3×10, PReLU 10×Affine 24×24, PReLU Affine 24×2	6 157	
Generic	$N_{\Psi'}$	Linear 3×2	6	8 463
	N_{Ψ}	Linear 2×3, PReLU Affine 3×30, PReLU 8×Affine 30×30, PReLU Affine 30×3, PReLU Affine 3×3	7 672	
	N_{F_n}	Affine 3×3, PReLU 8×Affine 9×9, PReLU Affine 9×3, PReLU Affine 3×2	785	

We report in Table B4 the neural networks' architectures used in Section 7.3. In this case, the activation functions used in the encoder are smooth to ensure a continuous $J_{\Psi'}$ and, consequently, a continuous F_n.

Table B4. Neural networks' architectures used in Section 7.3. $|w|$ is the number of trainable weights for the single neural network, $|w|_{\text{tot}}$ the total number of trainable weights.

	layer	$\lvert w\rvert$	$\lvert w\rvert_{\text{tot}}$
$N_{\Psi'}$	Affine 20×21, ELU 5×Affine 21×21, ELU Linear 21× 3, ELU	2 814	
N_{Ψ}	Linear 3×20, PReLU Affine 20×21, PReLU 7×Affine 21×21, PReLU Affine 21×20, PReLU Affine 20×20	4 412	9 515
N_{F_n}	Affine 5×20, PReLU 5×Affine 20×20, PReLU Affine 9×3, PReLU Affine 3×3	2 289	

We report in Table B5 the neural networks' architectures used in Section 7.4.

Table B5. Neural networks' architectures used in Section 7.4. $|w|$ is the number of trainable weights for the single neural network, $|w|_{\text{tot}}$ the total number of trainable weights.

	layer	$\lvert w\rvert$	$\lvert w\rvert_{\text{tot}}$
$N_{\Psi'}$	2×Affine 50×50, ELU Linear 50× 3	5 250	
N_{Ψ}	Linear 3×50, PReLU 8×Affine 50×50, PReLU Affine 50×50	20 558	28 097
N_{F_n}	Affine 5×20, PReLU 5×Affine 20×20, PReLU Affine 20×3	2 289