MOX-Report No. 53/2024

Neural ordinary differential equations for model order reduction of stiff systems

Caldana, M.; Hesthaven, J. S.

Neural ordinary differential equations for model order reduction of stiff systems

Matteo Caldana[a], [*] Jan S. Hesthaven[b]

[a] *MOX, Dipartimento di Matematica, Politecnico di Milano,*
Piazza Leonardo da Vinci 32, 20133 Milano, Italy
[b] *Chair of Computational Mathematics and Simulation Science, École Polytechnique Fédérale de Lausanne,*
1015 Lausanne, Switzerland

August 12, 2024

Abstract

Neural Ordinary Differential Equations (ODEs) represent a significant advancement at the intersection of machine learning and dynamical systems, offering a continuous-time analog to discrete neural networks. Despite their promise, deploying neural ODEs in practical applications often encounters the challenge of stiffness, a condition where rapid variations in some components of the solution demand prohibitively small time steps for explicit solvers. This work addresses the stiffness issue when employing neural ODEs for model order reduction by introducing a suitable reparametrization in time. The considered map is data-driven and it is induced by the adaptive time-stepping of an implicit solver on a reference solution. We show the map produces a nonstiff system that can be cheaply solved with an explicit time integration scheme. The original, stiff, time dynamic is recovered by means of a map learnt by a neural network that connects the state space to the time reparametrization. We validate our method through extensive experiments, demonstrating improvements in efficiency for the neural ODE inference while maintaining robustness and accuracy. The neural network model also showcases good generalization properties for times beyond the training data.

Key words: Ordinary differential equations, Stiff equation, Reduced order model, Neural networks, Runge-Kutta.
AMS subject classification: 65L99, 68T07.

1 Introduction

In computational science and engineering, the complexity and scale of numerical simulations have grown exponentially, driven by advances in hardware and algorithms. High-fidelity models can now capture intricate details of physical phenomena across diverse domains such as fluid dynamics, structural analysis, climate modeling, and materials science. However, the increasing complexity of these models often comes at a significant computational cost, rendering them impractical for many applications that require repeated model evaluations over a large number of parameter values. Indeed, the computational cost is driven up by the curse of dimensionality [4].

Reduced Order Models (ROMs) have emerged as a powerful solution to this challenge, offering a balance between accuracy and computational efficiency. By compressing the essential features of high-fidelity models into a more manageable form, usually called latent space, ROMs enable the efficient simulation and analysis of complex systems, especially those requiring real-time analysis [53], inverse problems [22, 25, 39, 45], uncertainty quantification [12, 18, 62], shape optimization [46] or optimal control [7, 49, 57].

The field of ROMs has seen substantial growth, providing practitioners with a wide range of techniques to choose from. Each method comes with its own set of advantages and drawbacks, and the optimal choice of ROM often depends on the specific requirements of the application. One of the most well-established paradigms for model order reduction is proper orthogonal decomposition (POD) [6, 33]. A linear projection technique that provides a high compression rate and, for instance, achieves a particularly high level of accuracy in the case of diffusion processes.

If the governing equations of the high-fidelity model are explicitly employed in the ROM, they are called intrusive [2, 5, 11, 32, 56]. Usually, a greedy algorithm or POD is employed to project the high-fidelity model onto a lower-dimensional space. However, for nonlinear models, additional techniques like the (discrete) empirical interpolation method are required to handle the complexities, which can lead to an unfavorable

---

[*] Corresponding author: `matteo.caldana@polimi.it`

trade-off between accuracy and computational efficiency [3, 15, 17]. Moreover, in [50] it has been shown that for transport-dominated problems, a large latent dimension must be used to achieve reasonable accuracy when using POD, thus limiting their practical use.

In recent years, data-driven methods based on machine learning and deep learning gained large traction due to advances in GPU-based hardware and developments in algorithms and software supporting the artificial intelligence ecosystem. This paradigm is based on the availability of a large quantity of high-fidelity data that can be used to train the model. The approach proves effective if the large amount of resources employed in the offline phase (generation of the solutions and training of the model) is repaid by a large enough number of (cheap) evaluations of the trained model (online phase). These models are called non-intrusive since only knowledge of the data is needed to build them [34, 66]. The field is ever-growing, with a large number of techniques being developed and published. To the best of our knowledge, the most relevant examples of data-driven approaches are the following. Closure models are a hybrid between POD and data-driven approaches where a non-linear term (a neural network) is added to the linear ROM (usually POD-Galerkin) to account for the effect of the unresolved dynamics [67]. In [9, 26, 47, 50, 65], densely connected or convolutional autoencoders have been used to extract the full latent variables. Other techniques have been proposed to enhance the accuracy. For instance, graph neural networks (GNNs) tackle the problem of complicated geometries discretized with unstructured meshes, offering a more natural representation of variables using a graph [52, 54]. Fourier neural operators are a very successful approach that instead exploits the frequency space to enrich the layers of the model [40, 41, 70]. Depending on the considered model, time evolution may be treated just as any parameter of the system or explicitly learned, examples are DeepONets [44, 50], dynamic mode decomposition [8, 60], sparse identification of reduced latent dynamics (SINDy) [10, 16], RNNs [42], LSTMs [21, 65] or Gaussian processes [29].

Another time evolution approach of particular interest is neural ODEs [19, 55, 58, 59]. The core concept of this method is straightforward: it involves learning the right-hand side of a dynamical system with a dense neural network, which can be viewed as the continuous limit of a residual neural network. This approach offers numerous advantages, including a more natural representation of continuous-time data and enhanced parameter and memory efficiency. The primary challenge with neural ODEs is that they are in all effects ODEs that must be solved. This often leads to issues with stiffness, complicating their solution and potentially affecting their stability and accuracy. Stiffness is the condition where there are rapid changes in some components of the solution while other components change slowly. This phenomenon creates numerical challenges when solving the equations, as explicit numerical methods might become inefficient or unstable [69]. Thus, the necessity of employing implicit solvers significantly drives up the cost.

In the present work, we tackle the problem of building a ROM of a stiff system. In this case, the main bottleneck is not represented by the large number of variables of the system but by the stiffness of the system. Indeed, stiffness is usually connected to features of the system, such as the presence of multiple time scales, the spectrum of the Jacobian of the system, or numerical instability, that make building a ROM a challenging task. In this case, employing fixed-time timesteppers is unfeasible since the rapidly varying scales require unreasonably small steps. On the other hand, treating the time like an input of a dense neural network also proves problematic since neural networks are known to perform badly on high frequency [68]. Still, in literature, there can be found several examples of stiff problems solved with machine learning. A possible solution was first proposed in [37], where the authors employ quasi-steady-state assumption (QSSA) to reduce the stiffness of the ODE systems and show that a physic-informed neural network (PINN) then can be successfully applied to the converted non-/mild-stiff systems. This is particularly interesting since the system can be solved with an explicit method. In [38] the authors show that computing a stabilized gradient and suitable scaling of the network outputs enables learning stiff neural ODEs. In [23], PINNs are with Extreme Learning Machine to solve stiff problems. In [1], stiff quantitative systems pharmacology models are accelerated with echo state networks. In [72], an autoencoder is used to produce a reduced order model of a chemical kinetic model for the simulation of a combustion system. Recently, in [28] DeepONets are used to learn the discretized solution of challenging stiff chemical kinetics.

Our strategy is the following, we aim to employ a neural ODE to learn a time-reparametrizated stiff system, where the time map is suitably built so to reduce the stiffness of the problem. The insight is to exploit the advantages of the neural ODEs we previously described while keeping the (online) cost of solving the ODE under control by using an explicit solver. Indeed, our main contribution is to employ the time reparametrization induced by the time-stepping of an implicit solver as a way to reduce the stiffness of the system. An advantage of this procedure is that it is purely data-driven, meaning that it is possible to avoid manually deriving explicit algebraic expressions for the QSS species as it was done in [37]. Once the nonstiff system is solved, the solution is mapped back to the stiff dynamics using a map learnt with a neural network. Particular care is taken in the definition of the map so that it has no dependence on time. Hence, the model maintains good generalization in time since there is only explicit dependence on the state space.

We demonstrate the accuracy and speed of the proposed ROM on a set of five test problems widely used in literature to benchmark stiff ODE solvers [48]. The results are compared with a state-of-the-art

Runge-Kutta implicit solver of the Radau II A kind of the fifth order [69].

The remainder of the paper is organized as follows. First, in Section 2 we introduce the mathematical settings and notation for the problems of interest. In particular, in Section 2.4 and 2.5 we review neural networks and the neural ODEs approach, which are the foundations of this work. In Section 3 we introduce our methodology, discussing also the implementation details. In Section 4 we present a comprehensive set of numerical results. Finally, in Section 5 we draw some conclusions, discussing the strengths and limitations of the proposed approach.

2 Problem formulation

In this section, we first present the construction of ROMs in algebraic terms for approximating a parametric dynamical system. Then, we provide a concise overview of fundamental deep learning concepts and formulate the data-driven model order reduction problem within the framework of neural ODEs.

2.1 Parametric systems of ordinary differential equations

We consider the following parameterized, finite-dimensional dynamical system, described by a set of first-order ODE:

$$\begin{cases} \dot{\boldsymbol{u}}(t) = \boldsymbol{f}(t, \boldsymbol{u}; \boldsymbol{\mu}), & t \in (0, T) \\ \boldsymbol{u}(0; \boldsymbol{\mu}) = \boldsymbol{u}_0(\boldsymbol{\mu}). \end{cases} \tag{1}$$

where $\boldsymbol{\mu} \in \Gamma$ is a vector containing all the parameters belonging to a compact set $\Gamma \subset \mathbb{R}^{N_\mu}$, $\boldsymbol{u} : [0, T) \times \Gamma \to \mathbb{R}^{N_u}$ is the parameterized solution, $\boldsymbol{f} : [0, T) \times \mathbb{R}^{N_u} \to \mathbb{R}^{N_u}$ is the right-hand side function (encoding the dynamical system), $t \in [0, T)$ is the time variable, $\boldsymbol{u}_0 : \Gamma \to \mathbb{R}^{N_u}$ is the initial condition, and $\dot{\boldsymbol{u}}$ is the total derivative with respect to time t. Without loss of generality, we are considering that the initial time for the system is zero; indeed there always exists a translation in time that brings us to this case.

This formulation encompasses not only ODEs but also problems stemming from the semi-discrete formulation of systems of partial differential equations (PDEs) through suitable methods (e.g.: finite elements [73], spectral elements [14], discontinuous Galerkin [20, 35], etc.). In these cases, one may observe an increase in the computational cost of the simulation since the dimension of the state space N_u is usually large. Indeed, these methods usually rely on a fine mesh to discretize the spatial derivatives.

The meaning of the parameter $\boldsymbol{\mu} \in \Gamma$ is broad, indeed it may represent physical properties (e.g.: material properties), initial conditions, or geometrical properties (e.g.: the shape of the domain). In this work, we focus on cases where $\boldsymbol{\mu}$ models physical properties.

2.2 Solving ordinary differential equations

There is a wide literature of methods to solve ODEs [69]. In this work, we concentrate on the well-known Runge-Kutta methods. Namely, let $b_i, a_{ij}, (i, j = 1, ..., s)$ be real numbers and let c_i be defined as

$$c_i = \sum_{j=1}^{i-1} a_{ij}, \quad \forall i = 1, ..., s.$$

Then, we call the s-stage Runge-Kutta method the following discretization of Eq. (1) at time $t_0 + \Delta t$

$$\begin{aligned} \boldsymbol{k}_i &= \boldsymbol{f}\left(t_0 + c_i \Delta t, \boldsymbol{u}_0 + \Delta t \sum_{j=1}^{s} a_{ij} \boldsymbol{k}_j\right), \quad i = 1, ..., s \\ \boldsymbol{u}_1 &= \boldsymbol{u}_0 + \Delta t \sum_{i=1}^{s} b_i \boldsymbol{k}_i, \end{aligned} \tag{2}$$

where $\boldsymbol{u}_1 = \boldsymbol{u}(t_0 + \Delta t)$ and $t_0 = 0$. If $a_{ij} = 0$ for $i \leq j$, each stage of Eq. (2) depends only on the previous ones, thus the method is explicit. If $a_{ij} = 0$ for $i < j$ and at least one $a_{ii} = 0$, the method is called diagonally implicit. In all the other cases, it is called implicit.

Of particular interest are embedded Runge-Kutta methods, which automatically adjust the step size to achieve a prescribed tolerance of the local error. The embedded method exploits a second set of scalars $b_i^*, i = 1, ..., s$ to compute another approximation of the solution of lower order $\boldsymbol{u}^*$. Namely, we want to satisfy the component-wise inequality

$$|(\boldsymbol{u}_1)_i - (\boldsymbol{u}_1^*)_i| \leq \delta_i, \qquad \delta_i = \mathtt{atol} + \mathtt{rtol} \max(|(\boldsymbol{u}_1)_i|, |(\boldsymbol{u}_0)_i|), \qquad i = 1, ..., N_u$$

where `atol` and `rtol` are user-defined (positive) absolute and relative tolerances, respectively. Then, the error is estimated as

$$\varepsilon = \sqrt{\frac{1}{N_u}\sum_{i=1}^{N_u}\left(\frac{(\boldsymbol{u}_1)_i - (\boldsymbol{u}_1^*)_i}{\delta_i}\right)^2}$$

and it is compared to one to find the next step size, namely

$$\Delta t_{\text{new}} = \Delta t \min(10, \max(0.1, (1/\varepsilon)^{1/(q+1)})),$$

where q is usually chosen to be the order of the method, and 10 and 0.1 are the maximum and minimum change factors, respectively. The value of these factors can be tuned depending on the problem. If the normalized error is larger than one, the timestep is rejected and is computed again with the updated Δt.

Explicit Runge-Kutta methods are generally unsuitable for the solution of stiff equations because their region of absolute stability is small [69]. Indeed, in the present work, we will employ three different Runge-Kutta methods depending on the application. Namely, we use an implicit Runge-Kutta of the Radau II A kind of the fifth order [30] for solving the stiff problems, and we use a Runge-Kutta of the fourth order (in the adaptive and fixed step size variants) for nonstiff problems. Starting from the paper of Butcher [13], it became customary to symbolize method (2) by the a tableau

$$\begin{array}{c|cccc}
c_1 & a_{11} & a_{12} & \dots & a_{1s} \\
c_2 & a_{21} & a_{22} & \dots & a_{2s} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\
\hline
 & b_1 & b_2 & \dots & b_s \\
 & b_1^* & b_2^* & \dots & b_s^*
\end{array}$$

We have the following table for the Radau II A method

$$\begin{array}{c|ccc}
\frac{2}{5} - \frac{\sqrt{6}}{10} & \frac{11}{45} - \frac{7\sqrt{6}}{360} & \frac{37}{225} - \frac{169\sqrt{6}}{1800} & -\frac{2}{225} + \frac{\sqrt{6}}{75} \\
\frac{2}{5} + \frac{\sqrt{6}}{10} & \frac{37}{225} + \frac{169\sqrt{6}}{1800} & \frac{11}{45} + \frac{7\sqrt{6}}{360} & -\frac{2}{225} - \frac{\sqrt{6}}{75} \\
1 & \frac{4}{9} - \frac{\sqrt{6}}{36} & \frac{4}{9} + \frac{\sqrt{6}}{36} & \frac{1}{9} \\
\hline
 & \frac{4}{9} - \frac{\sqrt{6}}{36} & \frac{4}{9} + \frac{\sqrt{6}}{36} & \frac{1}{9}
\end{array}$$

Due to the super-convergence of the Radau IIA method (classical order $p = 2s - 1$) it is not possible to have an embedded method of order $p - 1$ without extra cost. By taking a linear combination of $\Delta t \boldsymbol{f}(\boldsymbol{x}_0, \boldsymbol{u}_0)$ and the internal stage values $\boldsymbol{k}_i$ it is however possible to get an approximation of order s, we refer the interested reader to [30] for details. The following tableau is used for explicit Runge-Kutta of the fourth-order

0							
1/5	1/5						
3/10	3/40	9/40					
4/5	44/45	−56/15	32/9				
8/9	19372/6561	−25360/2187	64448/6561	−212/729			
1	9017/3168	−355/33	46732/5247	49/176	−5103/18656		
1	35/384	0	500/1113	125/192	−2187/6784	11/84	
	35/384	0	500/1113	125/192	−2187/6784	11/84	0
	5179/57600	0	7571/16695	393/640	−92097/339200	187/2100	1/40

2.3 Data-driven model order reduction

In this section, we discuss the concepts and notation that stand at the basis of data-driven model order reduction. We introduce the manifold of solutions of equation (1)

$$\mathcal{M} = \{\boldsymbol{u}(t;\boldsymbol{\mu}) \,|\, t \in [0,T), \boldsymbol{\mu} \in \Gamma\} \subset \mathbb{R}^{N_u}, \tag{3}$$

which is a collection of the solutions for each parameter $\boldsymbol{\mu} \in \Gamma$ and time $t \in [0,T)$. We suppose that it is possible to find a discretized approximation $\boldsymbol{u}_h(t;\boldsymbol{\mu})$ up to arbitrary accuracy by integrating (1) with a suitable method. By solving the problem for M_μ different choices of $\boldsymbol{\mu}$ we create a discretized manifold

$$\mathcal{M}_h = \{\boldsymbol{u}_h(t;\boldsymbol{\mu}) \,|\, t \in \{0 \le t_1 < ... < t_{M_T} \le T\}, \boldsymbol{\mu} \in \Gamma_h = \{\boldsymbol{\mu}_1 \in \Gamma, ..., \boldsymbol{\mu}_{M_\mu} \in \Gamma\}\} \subset \mathbb{R}^{N_u}, \tag{4}$$

which is a finite set of vectors $\boldsymbol{u}_h \in \mathbb{R}^{N_u}$ that approximate the solution at a certain discretized point in time t and for a certain choice of parameter $\boldsymbol{\mu}$. This set can be reshaped into a so-called, snapshot matrix

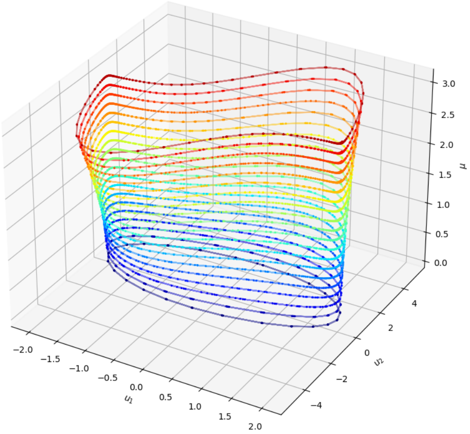

Figure 1: Example of discrete solution manifold $\mathcal{M}_h$ for the Van der Pol oscillator.

$\boldsymbol{U} \in \mathbb{R}^{N_u \times M}$, where $M = |\mathcal{M}_h|$, by stacking all the column vectors of the discrete manifold. Let us stress that the time discretization $\{0 \leq t_1 < ... < t_{M_T} \leq T\}$ may change depending on the value of $\boldsymbol{\mu}$. Indeed, it often happens that $\boldsymbol{\mu}$ changes the dynamics of the system, and thus a different time discretization is generated by the adaptive procedure. For the sake of simplicity, we omit the explicit dependence on $\boldsymbol{\mu}$ of the time discretization.

In practice, one might obtain data also from knowing the exact analytical solution of the problem for some specific values of $\boldsymbol{\mu}$ or from real-world measurements. However, it is of utmost importance that the data gathered in the snapshot matrix $\boldsymbol{U}$ is reliable since it is interpreted as our "ground truth".

Data-driven ROMs aim to leverage the information contained in the discretized solution manifold $\mathcal{M}_h$ to obtain in a fast and reliable way the solution $\boldsymbol{u}_h(t;\mu)$ for $\boldsymbol{\mu} \notin \Gamma_h$ or $t \geq T$. Figure 1 shows a graphical representation of the solution manifold for the Van der Pol oscillator (cf. Section 4.1). The motivation behind ROMs is that classical time integration methods may become prohibitively expensive either due to the size of N_u or due to stiffness (cf. Section 3).

2.4 Neural networks

In this section, we review the basic concepts of deep learning that we will use to build our ROM. Let $N_I, N_O \in \mathbb{N}$ be strictly positive. An artificial neural network is a function $\mathcal{F} : \mathbb{R}^{N_I} \to \mathbb{R}^{N_O}$ that maps an input vector $\boldsymbol{x} \in \mathbb{R}^{N_I}$ to an output vector $\boldsymbol{y} \in \mathbb{R}^{N_O}$ and depends on the set of parameters $\boldsymbol{\theta}$ (we think of all the parameters as reshaped into a one-dimensional vector). As architecture, we consider the dense fully connected feedforward (FNN) neural network (also called multi-layer perceptron) which is defined as follows. Fix two positive integers L and Z. Let $\boldsymbol{y}^{(0)} = \boldsymbol{x}$ and $\boldsymbol{y}^{(L)} = \tilde{\boldsymbol{y}}$. A FNN of depth L and width Z is the composition of L functions called layers defined by

$$\boldsymbol{y}^{(l)} = \sigma^{(l)}(\mathrm{W}^{(l)}\boldsymbol{y}^{(l-1)} + \boldsymbol{b}^{(l)}), \quad l = 1, ..., L,$$

where $\mathrm{W}^{(l)} \in \mathbb{R}^{N_l \times N_{l-1}}$ are matrices of parameters called weights and $\boldsymbol{b}^{(l)} \in \mathbb{R}^{N_l}$ are vectors of parameters called biases. Here, $N_0 = N_I$, $N_L = N_O$ and $N_l = Z$ for $l = 1, ..., L-1$. Finally, at the end of each layer, a scalar non-linear activation function $\sigma^{(l)}$ is applied component-wise. We employ the same activation function (to be specified later, cf. Section 4) for all the layers apart from the last one where we use a linear activation function.

Several learning strategies are available to determine the vector of parameters $\boldsymbol{\theta}$. Among these, one of the most popular paradigms is supervised learning. This method focuses on defining and minimizing a non-linear cost function, known as the loss function $\mathcal{L}$. The loss function is typically formulated using a dataset comprising input-target pairs $(\boldsymbol{x}, \boldsymbol{y})_{i=1}^{N_{\text{train}}}$, which the neural network aims to learn. A common expression for the loss function is:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \|\boldsymbol{y} - \tilde{\boldsymbol{y}}(\boldsymbol{x};\boldsymbol{\theta})\|_p \,, \tag{5}$$

where $\|\cdot\|_p$ denotes the discrete p-norm, typically with $p = 1$ or $p = 2$.

To minimize the loss function gradient-based optimization is commonly employed. The gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}$ is efficiently computed using automatic differentiation techniques [51]. An epoch, defined as one complete pass through the dataset, consists of multiple gradient updates and is the unit often used to quantity the training

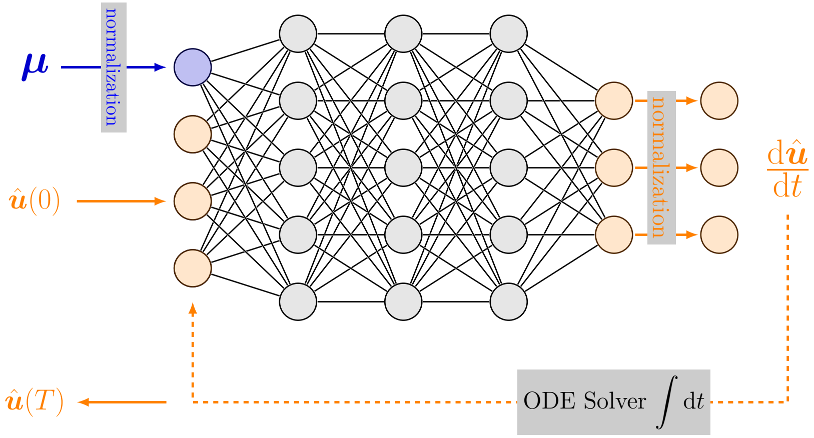

Figure 2: Architecture of the Neural ODE for model order reduction.

length. Properly optimized neural networks exhibit the ability to generalize, making accurate predictions on unseen data. For further reading on neural networks, comprehensive resources include [27, 71].

2.5 Neural ODEs

The model that we choose to use to build the ROM is neural ODEs. Neural ODEs have emerged as powerful tools to model continuous-time dynamics using a differential equation. This approach provides several key benefits. Firstly, Neural ODEs offer superior flexibility in modeling time-series data, as they can naturally handle irregularly sampled data and varying time steps. Secondly, they ensure smooth trajectories and better integration with physical laws and other domains where the underlying processes are continuous in nature. Furthermore, neural ODEs can lead to more efficient training and inference, as they require fewer parameters and can leverage advanced numerical solvers for differential equations. Neural ODEs are also more memory efficient compared to traditional recurrent neural networks, especially for long time-series data, because they avoid the need to store intermediate states of the network during backpropagation.

More precisely, a neural ODE is a neural network NN that evolves the dynamics as

$$\dot{\boldsymbol{u}}(t) = \mathcal{NN}(t, \boldsymbol{u}(t); \boldsymbol{\mu}), \tag{6}$$

that is the NN is employed to approximate the right-hand side of the dynamical system (1) by taking as inputs the time t, the current state $\boldsymbol{u}$, and the parameters $\boldsymbol{\mu}$. One of the main drawbacks of this approach is that it is necessary to numerically integrate the system in the online phase. Given the recurrent nature of the architecture, adjoint solvers have been developed to enhance the memory efficiency during the training phase [19]; their cost however remains much larger when compared to FNN.

When used as an architecture for ROMs, the neural ODE is often paired with another neural network to reduce the dimension of the system from N_u to a small latent state space. This is usually achieved by means of autoencoders [9, 26, 47, 50, 65] or other reduction techniques [59]. However, if N_u is small the advantages of using a small latent space representation may not be counterbalanced by the increased cost of evaluating the neural network. In this work, we propose a solution to this latter problem by introducing an appropriate change of variable that significantly reduces the stiffness of the problem. Indeed, we will show that our method makes it possible to use explicit solvers to integrate neural ODEs representing stiff systems.

We report in Figure 2 a representation of the neural ODE. The hat symbol $\hat{\cdot}$ represents a suitable normalization of the data, see the next section for details. For the sake of simplicity, we will indicate in the captions the reference solution (the one that is comprised in the dataset) with the subscript "*ref*", and the components of $\boldsymbol{u}$ as predicted by the neural ODE with simply u_i.

2.5.1 Training a Neural ODE

There are two main approaches to training a neural ODE. The first one is the classic supervised approach. Assuming that the value of $\boldsymbol{f}(t, \boldsymbol{u}_h(t); \boldsymbol{\mu})$ is known for each sample in the discrete manifold $\boldsymbol{u}_h \in \mathcal{M}_h$, then it is possible to compute the loss (5) and minimize it through a gradient-based optimizer. However, this kind of data is not always available. Instead, the second approach – the one originally proposed for the training of neural ODEs – minimizes the error with respect to the timeseries data $\boldsymbol{u}_h(t)$. Namely, by using a differentiable ODE solver, we minimize the loss

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \left\| \boldsymbol{u}_h(t; \boldsymbol{\mu}) - \left(\boldsymbol{u}_h(0; \boldsymbol{\mu}) + \int_0^t \mathcal{NN}(\boldsymbol{u}(s; \boldsymbol{\mu}), \boldsymbol{\mu}) \mathrm{d}s \right) \right\|_p, \tag{7}$$

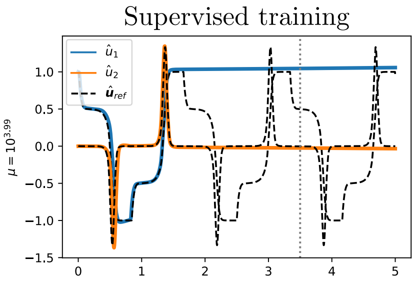

Figure 3: Comparison of the supervised and ODE solver approaches for training the neural ODE from the same randomly initialized model for the Van der Pol oscillator (cf. Section 4.1). In blue and orange the predictions of the model, and in black the reference solution. The supervised training is globally less accurate and stable, but better captures sharp gradients.

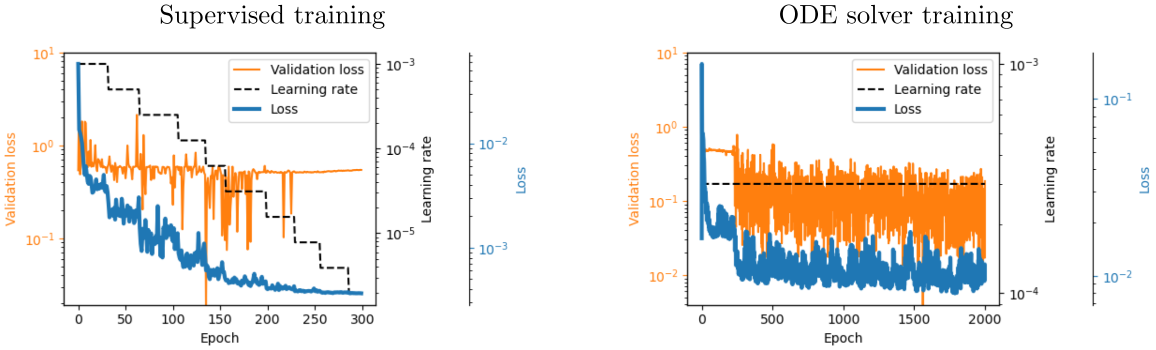

Figure 4: Example of the behavior of the loss, the validation loss, and the learning rate during the training of the ROM with the two-step approach: first the supervised approach and then the ODE solver approach.

where the integral is usually computed with an explicit integration method like forward Euler or an explicit Runge-Kutta method.

The advantage of the supervised approach is that it is less computationally demanding since it does not need to compute the gradient through the ODE solver using automatic differentiation. On the other hand, in the supervised framework, the loss is not always indicative of the accuracy of the predictions on the timeseries data $\boldsymbol{u}(t)$. For this reason, we always employ as validation loss Eq. (7).

Implementation details for Eq. (7) are important when it is employed to train the neural network. We resample the discrete timeseries $\boldsymbol{u}_h(t; \boldsymbol{u})$ to match the output of the time integrator, namely a Runge-Kutta method of the fourth order with a fixed timestep. To minimize the interpolation errors, we use a reference solution with a very fine time discretization. A uniform resampling allows for an efficient usage of the partial results of the ODE solver for all the times smaller than the final one. Moreover, the initial integration time is not zero but is picked uniformly between $[0, T)$ so that each time integration in the dataset is done with the same number of timesteps: this procedure enables vectorization of the computations, greatly reducing the computational cost. This optimization was originally introduced in `torchdiffeq` [19]. The number of timesteps used is called unroll length and is a hyperparameter that plays a key role in the optimization procedure. Finally, as commonly done in supervised learning, the computations are split into mini-batches to increase stochasticity. If not otherwise stated, when using Eq. (7) for validation or testing we employ as a time integrator an embedded Runge-Kutta method of order four with adaptive time-stepping. However, computations are significantly cheaper since we can turn off automatic differentiation.

In our experiments, we employ a two-phase training. First, a standard supervised approach is used to “initialize” the neural network since it is cheap and enables a fast reduction of the loss. Then, we switch to the differentiable ODE solver approach to fine-tune the ROM. Our experiments show that in this case, it is beneficial to use an increasingly larger unroll length to ensure the robustness and accuracy of the model. Further details will be given in Section 4 for each test case. Employing a two-phase training approach not only reduces the computational cost but also increases the quality of the final model. Indeed, from our experiments, we have observed that with the supervised learning approach the neural network is able to learn sharper gradients that may lead to unstable states. On the other hand, learning through the ODE solver produces stable results that might follow less precisely the reference solution. Figure 3 shows an example for two (underfitting) models. In Figure 4 we show the training and validation loss for a model trained in sequence with the supervised and ODE solver approaches.

Another important point of discussion is the explicit dependence on time t of the $\mathcal{NN}$. In practice, we do not use t as input of the neural network $\mathcal{NN}$. However, this does not necessarily limit us to autonomous systems of ODEs. Indeed, it is possible to use an input layer with memory, like an LSTM, to add temporal dependence. This approach is usually preferred to employing t as input of the neural network since it improves generalization properties in time.

To accelerate the training suitable normalizations are applied to the input $\boldsymbol{\mu}$, the output $\boldsymbol{f}$, and the timeseries data $\boldsymbol{u}$. The exact expressions for the normalizations depend on the test case considered and are detailed in Section 4. The insight behind the choice of the normalization is that we aim to have data that has a magnitude close to one, does not have outliers, and has either a uniform distribution or a normal distribution with a standard deviation of magnitude close to one. In our experiments, strictly monotone normalizations achieved better results. We designed the normalizations by manual inspection of the histograms of the data.

3 Addressing the stiffness issue

While many techniques have been developed to build reduced-order models of large systems stemming from semi-discretized PDEs, not many solutions are available for stiff ODEs. Indeed, it is a challenge to learn their complex dynamics, which often feature large gradients and widely different time scales. More rigorously, a commonly used formula to define the stiffness index of a problem is

$$S = \frac{Re(\lambda_{\max})}{Re(\lambda_{\min})} T, \tag{8}$$

where λ_i are the eigenvalues of the Jacobian of the system. However, it is known that this definition is not always helpful. For instance, it does not apply to the classic Robertson chemical kinetics problem since it has two zeros eigenvalues [61]. A more pragmatic definition is given by Harier: "*Stiff equations are problems for which explicit methods do not work*" [69]. That is, explicit numerical time integrators such as the embedded Runge-Kutta method of the fourth order require timesteps so small that implicit methods become cheaper. Stiffness is a particularly important issue when working with neural ODEs. Indeed, it is a well-known fact that neural ODE training may fail due to stiffness. For instance, [38] shows issues of this kind and proposes techniques to improve the training. However, it is still necessary to use a costly implicit solver in the online phase, since the learnt dynamic is stiff.

Hence, to use the neural ODE as a ROM, we aim to reduce the stiffness of the system, allowing for the usage of (cheap) explicit methods. To this end, we propose to change the dynamics of the problem by considering the change of variable in time induced by the adaptive time-stepping of the implicit solver used to solve the full-order model. We define a data-driven reparametrization of the time t called $t_s = t_s(t)$ such that

$$t_s(t^n) = \frac{n}{NT}.$$

That is, the Δt chosen at the n-th step by the implicit time stepper is proportional to the derivative of the function that we employ as a change of variable in time. We remark that $t_s = t_s(t)$ is strictly increasing and thus invertible. We guess this change of variables induces less stiff dynamics since the adaptive time-stepping spreads the sharp peaks and quickly adapts to the large timescales. The system becomes

$$\begin{cases} \dot{\boldsymbol{u}}(t_s) = \boldsymbol{f}_s(t_s, \boldsymbol{u}; \boldsymbol{\mu}), & t \in (0,1) \\ \boldsymbol{u}(0; \boldsymbol{\mu}) = \boldsymbol{u}_0(\boldsymbol{\mu}). \end{cases}$$

To accommodate this change in the neural ODE framework, we need a way to compute $t = t(t_s; \boldsymbol{\mu})$ in the online phase, to map the solution back to the original dynamics. Our experiments show that learning directly the map $t = t(t_s; \boldsymbol{\mu})$ is a hard task for a neural network, probably due to the presence of abrupt changes in the function. Instead, we found that learning $\dot{t}(\boldsymbol{u}(t_s; \boldsymbol{\mu}))$ is a stable approach that produces more accurate results. We also prefer this approach to learning $\dot{t}(t_s; \boldsymbol{\mu})$ to keep the system autonomous (that is, never introduce explicit dependence on time) to improve the generalization properties. The evaluation of the time dynamics is cheap since it can be done in one batch using vectorization. Indeed, in the first step, we compute $\boldsymbol{u}(t_s)$, then, we carry out the cumulative integration of $\dot{t}(\boldsymbol{u}(t_s; \boldsymbol{\mu}))$ with the Simpson's rule. The main advantage of this approach is that it is cheap compared to autoencoders, especially for small problems since the only overhead is the computation of $t = t(t_s; \boldsymbol{\mu})$. However, it is also possible to combine the autoencoder with the described methodology: first, the autoencoder extracts a latent state representation, then, we could apply the time reparametrization in the latent space, reducing the stiffness of the latent dynamics.

3.1 Implementation details

Applying this methodology to the training with the differentiable ODE solver does not require any change. Indeed, the values of the timeseries $\boldsymbol{u}(t_s)$ (that is, the state space) do not change, they are only differently spaced in time. The main difference concerns the supervised learning approach. As explained before, this step

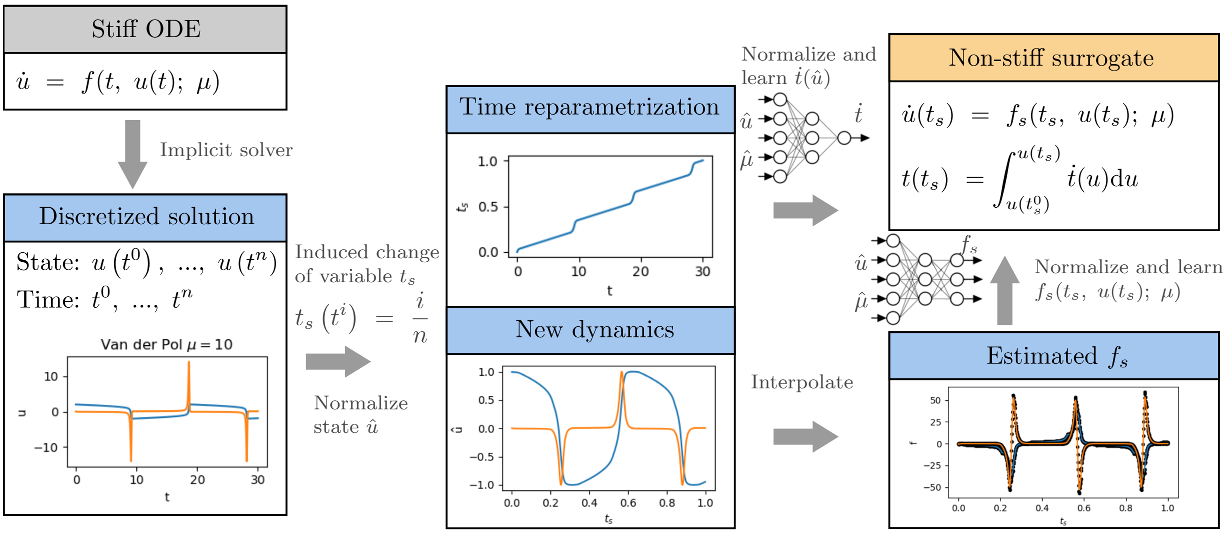

Figure 5: Time reparametrization workflow. Starting from the stiff ODE system (in gray), we compute the discrete solution and obtain the discrete manifold $\mathcal{M}_h$ using an implicit method. We normalize the solution, compute the time reparametrization $t_s(t)$, and estimate f_s (in blue). Finally, we are able to train a nonstiff model that can be solved using an explicit method (in orange).

allows us to "initialize" the neural network in a cheap way and requires us to estimate both $\dot{\boldsymbol{u}}(t_s) = \boldsymbol{f}(t_s, \boldsymbol{u}; \boldsymbol{\mu})$ and $\dot{t}$. The approximation of the derivative is done using a Savitzky-Golay filter with low order (two or three) and small window size (< 10). Figure 5 shows a scheme of the workflow. To accelerate the training also the values of the time derivative are normalized, namely, we have found a logarithmic normalization to work particularly well.

The optimizer we employ is AdamW [43] with batch size 32 and a learning rate scheduler that geometrically reduces errors on plateaus of a factor of one-half with patience twenty epochs. The training lasts for a maximum of 400 epochs. The best model is chosen using early stopping based on the validation loss. The activation functions, the depth and width of the neural network, the initial learning rate, and the seed are hyperparameters that we optimize by means of random search on a problem-by-problem basis. In particular, the activation function is chosen among GELU [31], SiLU [24], Hard Swish [36], Leaky ReLU and ReLU, the depth between three and ten, the width between five and 100 and the initial learning rate between $2 \cdot 10^{-3}$ and 10^{-4}.

We employ a very similar approach to fine-tune the differentiable solver. The main difference is that the maximum number of epochs considered is 2000. The unroll length and the timestep size are tuned for each problem. As a rule of thumb we employ increasingly larger unroll lengths with smaller and smaller learning rates. Namely, we start with covering about 5% of the time domain and it increases up to 50%.

The neural network $\mathcal{NN}_t$ approximating $\dot{t}(\boldsymbol{u}(t_s; \boldsymbol{\mu}))$ is also trained in a two-step process. First, it is trained in a supervised manner. Similarly to what we do in the other supervised step, we employ as validation loss the accuracy of the predictions on the validation timeseries when employing the ODE solver to integrate both $\dot{t}(\boldsymbol{u}(t_s; \boldsymbol{\mu}))$ and $\boldsymbol{f}(t_s, \boldsymbol{u}; \boldsymbol{\mu})$. Then, it is fine-tuned using as a target the time t. This is achieved by automatic differentiation through Simpson's integration rule. Namely, the loss is

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \left\| t - \int_{\boldsymbol{u}(0)}^{\boldsymbol{u}(t)} \mathcal{NN}_t(\boldsymbol{u}(t_s); \boldsymbol{\theta}) \mathrm{d}t_s \right\|_p ,$$

where the integral is computed cumulatively on the discretization induced by the explicit Runge-Kutta method applied to the trained $\mathcal{NN}$. We tested $p = 1, 2, 4$, the best results were achieved with $p = 1$.

Finally, to achieve the best performances in the online phase (inference time), the model is optimized using the features of modern deep learning libraries, such as oneDNN graph fusion [51].

4 Numerical results

Our methodology is tested in five numerical experiments featuring stiff problems widely used in the literature to benchmark implicit ODE integrators [48, 69]. We do not consider dynamical systems derived from semi-discrete PDEs since it would require the usage of autoencoders to extract the latent dynamics. Indeed, we aim to assess the behavior of our method as a stand-alone improvement to neural ODEs. Introducing other variables in the test cases, such as the error of the autoencoders may be detrimental to this task. We are confident that given the wide range of test cases presented, our methodology could be extended to larger state spaces or semi-discrete PDEs without problems. Indeed, being able to outperform classical ODE solvers on small problems without the need to use a simplified latent space representation is a more difficult task.

Computations were carried out on the HPC cluster at MOX with processor Intel® Xeon® Gold 6238R and on a laptop with an AMD® Ryzen® 7 PRO 7840U. The reported computational cost refers to the latter CPU. For the sake of reproducibility, we rely on the standard (CPU) implementations found in SciPy [64] of the Runge-Kutta methods. Moreover, to ensure a fair and rigorous comparison with the neural network, we executed both methods on a CPU. While leveraging a GPU would likely enable the neural network to significantly outperform the full-order method, our goal was to maintain a balanced evaluation framework.

Before proceeding with the presentation of the test cases, let us introduce some metrics of interest that we will use to assess the performance of the ROMs.

- **time**: elapsed CPU time in seconds to solve the system up to the final time T.
- **# fev**: number of evaluations of the right-hand side (either $\boldsymbol{f}$ or $\mathcal{NN}$).
- **# jev**: number of evaluations of the Jacobian of $\boldsymbol{f}$ (used for the implicit solver).
- **# lu**: number of LU decompositions (used for the implicit solver)
- $\mathbf{MSE}_{t_s}$: mean squared error of $\mathcal{NN}$ in the t_s reparametrization.
- **MSE**: mean squared error in the t parametrization.
- L^2: relative integral error computed with Simpson's rule.
- d_{peak}: average distance of the peaks (any sample whose two direct neighbors have a smaller amplitude) between the reference solution and the low fidelity solution (used for periodic solutions).

4.1 Test case 1: Van der Pol oscillator

The van der Pol oscillator is a second-order differential equation that exhibits limit cycle behavior, which makes it a valuable model for studying phenomena such as electrical circuits, cardiac rhythms, and neuronal firing patterns. The problem is governed by a parameter $\mu > 0$ and we consider its following formulation in the time interval $[0, 3.5\mu]$

$$\begin{cases} \dot{u_1} = u_2, \\ \dot{u_2} = \mu(1 - u_1^2)u_2 - u_1, \\ \boldsymbol{u}(0) = (2, 0). \end{cases} \tag{9}$$

This test case aims to assess the capabilities of our method on an autonomous periodic system. Indeed, the Van der Pol oscillator has two periodic solutions: a non-trivial periodic solution and an unstable zero solution. The parameter $\mu > 0$ determines the significance of the nonlinear component of the equation. When μ is large the system becomes stiff. We set $\Gamma = [10^2, 10^4]$ and consider training dataset composed by a logarithmic discretization Γ_h of Γ with $N_\mu = 51$ points. The validation dataset is the interval midpoints of Γ_h. The reference solutions are computed using the Radau method, with absolute and relative tolerances set to 10^{-12}.

The data for the supervised training is smoothed with a Savitzky-Golay filter of window seven and order two. Data, namely the columns of $\boldsymbol{U}$, are randomly subsampled to have about one thousand points for each period of the timeseries. Fine-tuning is done by increasing the size of the unroll length starting from 20 up to 80 (exact values are a hyperparameter to be tuned) and time-stepping $\Delta t = 1/40$. The starting learning rate is reduced by a couple of orders of magnitude with respect to the supervised step.

Data is normalized using the following functions: parameter normalization: $\hat{\mu} = \log_{10} \mu$; state normalization: $\hat{\boldsymbol{u}} = (u_1/2, u_2/\mu)$; dynamics normalization: $\hat{\boldsymbol{f}} = (f_1/5, f_2/10)/\mu$; time normalization: $\hat{T} = T/\mu$. The model is tested for $N_\mu^{\text{test}} = 4$ values of $\mu \in [10^{2.01}, 10^{3.99}]$ spaced logarithmically (these values are chosen so that they are not present in the training nor the validation dataset). Moreover, we are also testing for time beyond what is present in the training and validation datasets, namely, we are integrating until a final time $\hat{T} = 5$. Figure 6 shows the results of applying the change of dynamics to this problem. On the left, the plot of the problem with the new dynamics shows smaller gradients. On the right, after the change of variable, we recover with good accuracy the original dynamics of the Van der Pol oscillator. In Table 1 we show a comparison of the computational cost. The number of right-hand side evaluations is one order of magnitude smaller for the explicit solver of the neural ODE w.r.t. the implicit solver of the FOM, this proves that change of variable has significantly reduced the stiffness of the problem. Another interesting feature of our approach is that the ROM error is almost independent from μ, on the other hand, the implicit solver

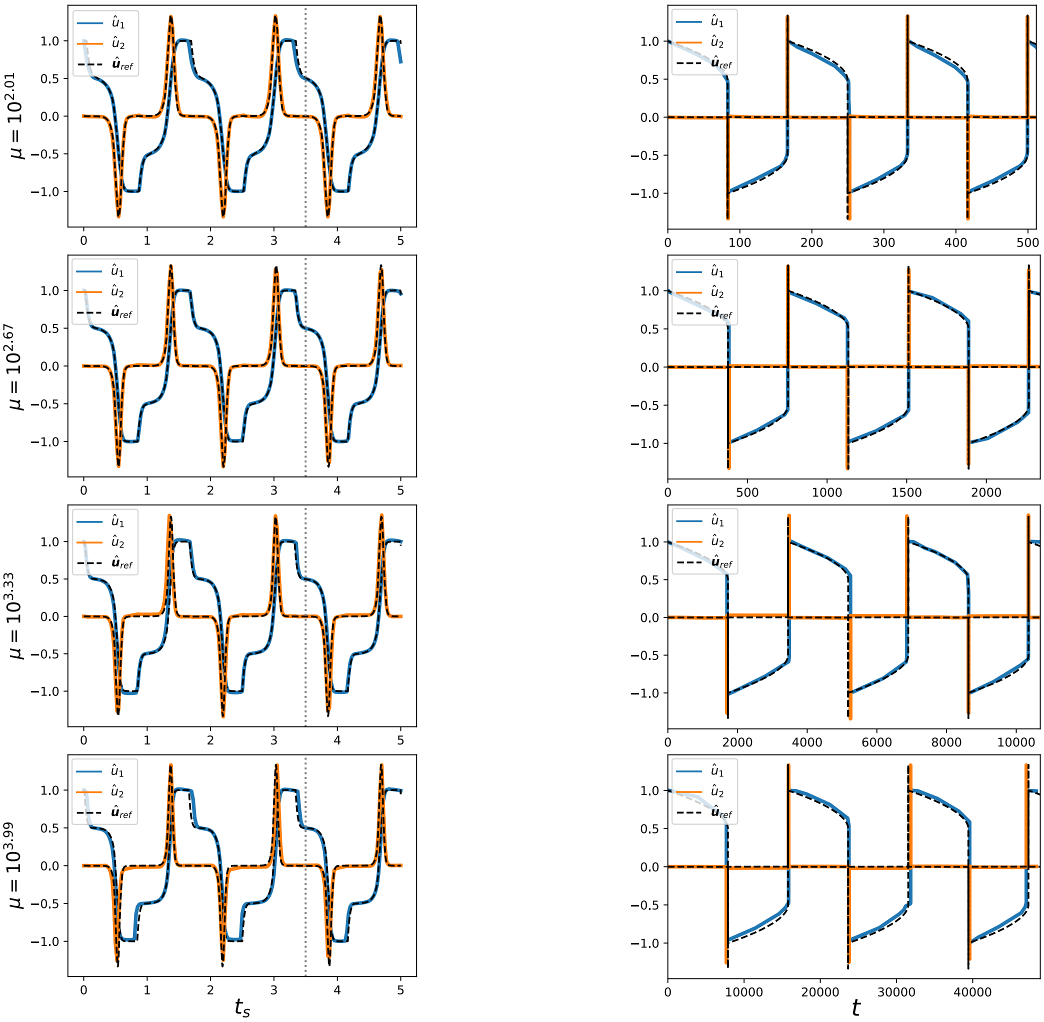

Figure 6: Test case 1: Van der Pol oscillator. On the left, in colors, the neural ODE predictions in t_s. On the right, in colors, the prediction is mapped to the original time t. In black is the reference solution. The vertical line (⋯⋯) represents the final time in the training dataset.

exhibits a correlation among the two, suggesting that for larger values of μ computational gains may be larger. In this case, the L^2 error is computed only with respect to the first component of $\boldsymbol{u}$. Indeed, since u_2 features sharp peaks, small variations in their position make the integral error large and not representative of the accuracy of the solution. For this reason, we have introduced the d_{peak} metric, which tries to measure the accuracy of the ROM in this regard. We stress that for this test case, the ROM is performing well also for unknown times, maintaining an accurate prediction of the period of the system. From the metrics in Table 1 it is also possible to notice that the main bottleneck in terms of accuracy for the ROM is the time mapping. Indeed, the model is very accurate in the t_s parametrization.

In this case, we employ a fixed time-stepping in the online phase. This is a feasible approach only because the dynamics in the t_s parameterization exhibit similar behavior across all considered values of μ.

4.2 Test case 2: OREGO problem

The OREGO problem consists of the following stiff system of 3 non-linear ODEs defined in the time interval $t \in [0, 1000]$

$$\begin{cases} \dot{u}_1 = \mu_1(u_2 - u_1u_2 + u_1 - \mu_3 u_1^2), \\ \dot{u}_2 = \frac{1}{\mu_1}(-u_2 - u_1u_2 + u_3), \\ \dot{u}_3 = \mu_2(u_1 - u_3), \\ \boldsymbol{u}(0) = (1, 2, 3). \end{cases} \tag{10}$$

Similarly to the previous test case, the problem is an autonomous system that features a periodic solution. However, the problem originates from the description of a chemical reaction, thus it is key for the concentration $\boldsymbol{u}$ to stay positive. Moreover, the values of $\boldsymbol{u}$ span six orders of magnitude. The considered parameter

μ	solver	toll	time [s]	# fev	# jev	# lu	MSE$_{t_s}$	L^2	d_{peak}
$10^{2.01}$	ROM	1/40	0.033	1000	0	0	4.13e-3	0.03520	0.662
	Radau	10^{-2}	0.046	2470	110	516	–	0.01447	0.811
$10^{2.67}$	ROM	1/40	0.033	1000	0	0	2.39e-3	0.02262	2.665
	Radau	10^{-2}	0.056	3316	133	688	–	0.00870	2.075
$10^{3.33}$	ROM	1/40	0.033	1000	0	0	5.94e-3	0.03098	19.27
	Radau	10^{-2}	0.072	3850	154	806	–	0.00742	8.083
$10^{3.99}$	ROM	1/40	0.033	1000	0	0	7.23e-3	0.04768	191.0
	Radau	10^{-2}	0.078	4418	156	938	–	0.04377	210.9

Table 1: Test case 1: Van der Pol. Comparison of computational cost and accuracy for the Radau solver and the neural ODE based reduced order model (ROM) on the test dataset. Refer to Section 4 for the definition of the metrics.

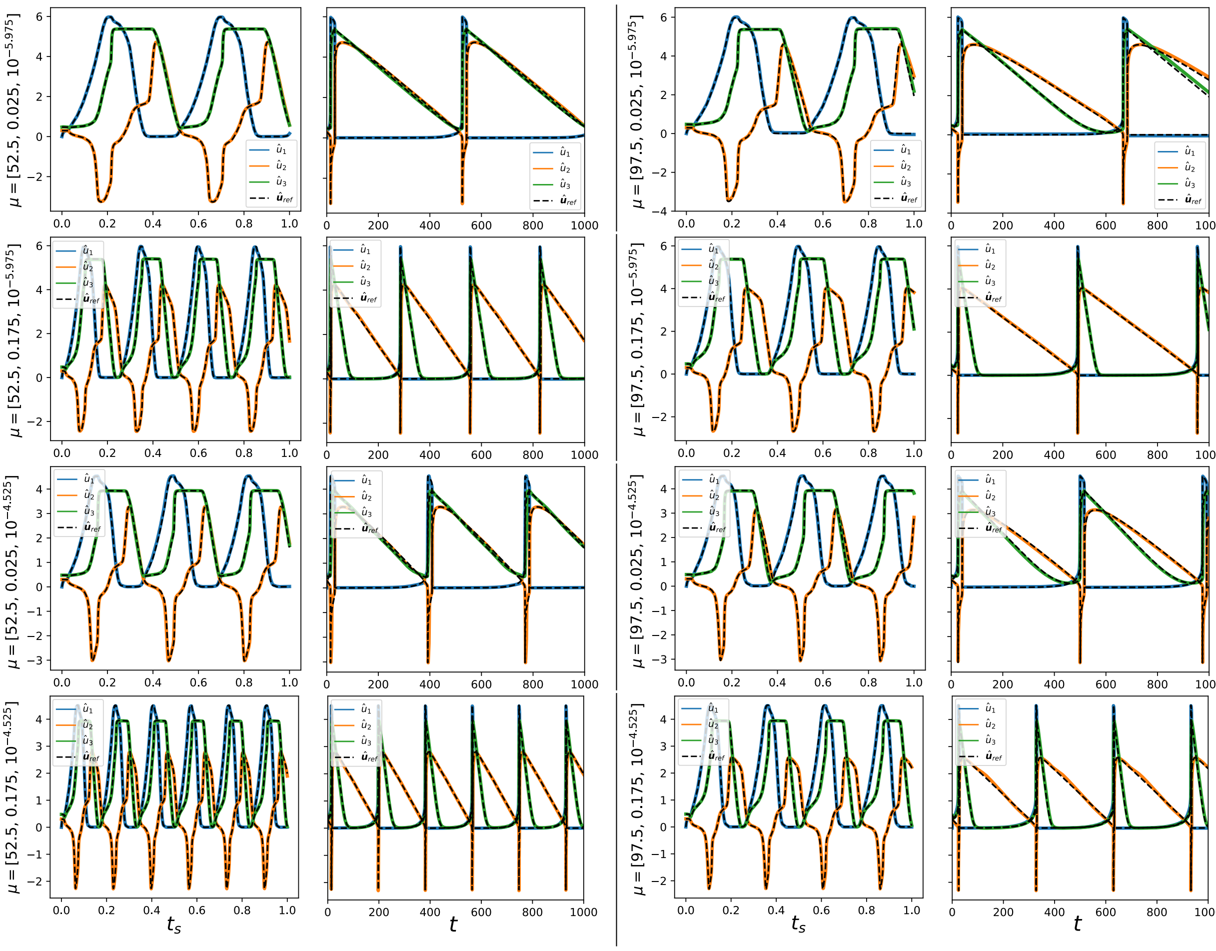

Figure 7: Test case 2: OREGO. On the left, in colors, the neural ODE predictions in t_s. On the right, in colors, the prediction is mapped to the original time t. In black is the reference solution. The state scale is logarithmic.

space is $\Gamma = [50, 100] \times [0.002, 0.02] \times [10^{-6}, 10^{-4}]$, which contains the value $\boldsymbol{\mu} = (77.27, 0.161, 8.375 \cdot 10^{-5})$ usually employed in literature for this test case. The training dataset is built by subsampling a discretization of Γ with a uniform grid of 6, 19, and 21 points in each direction, respectively. The validation dataset is based on a discretization of Γ in the midpoints of the training discretization. Reference solutions are computed with a tolerance of 10^{-10}. We apply the following normalizations: parameter normalization: $\hat{\boldsymbol{\mu}} = (\mu_1/77.27, \mu_2/0.161, \log_{10}\mu_3/5)$; state normalization $\hat{\boldsymbol{u}} = \log_{10}\boldsymbol{u}$; dynamics normalization: $\hat{\boldsymbol{f}} = \boldsymbol{f}$. We test the performance of the ROM for eight parameters

$$\boldsymbol{\mu}_{\text{test}} \in \{52.5, 97.5\} \times \{0.025, 0.175\} \times \{10^{-5.975}, 10^{-4.525}\}.$$

Figure 7 shows that the ROM is able to accurately follow the reference solution and track its period. A quantitative comparison with the Radau solver is reported in Table 2. We remark that a tolerance of 10^{-2} for the Radau solver did not produce a positive solution for all the problems in the test dataset, thus the comparison is made by employing a tolerance of 10^{-3}. The number of right-hand side evaluations of the explicit solver is less than half the one of the implicit one, showing that our method has made the system nonstiff. Also, the computational cost is consistently smaller than the one needed by the Radau solver, however, the latter has better accuracy. On the other hand, the distance of the peaks is comparable, proving

$\boldsymbol{\mu}$	solver	toll	time [s]	# fev	# jev	# lu	MSE_{t_s}	L^2	d_{peak}
$(52.5, 0.025, 10^{-5.975})$	ROM	$2 \cdot 10^{-4}$	0.037	1130	0	0	1.20e-3	1.19e-2	1.668
	Radau	10^{-3}	0.065	3777	167	656	–	1.31e-5	3.032
$(97.5, 0.025, 10^{-5.975})$	ROM	$2 \cdot 10^{-4}$	0.034	1046	0	0	4.52e-3	7.75e-2	5.041
	Radau	10^{-3}	0.067	3445	159	600	–	2.12e-5	4.632
$(52.5, 0.175, 10^{-5.975})$	ROM	$2 \cdot 10^{-4}$	0.065	1970	0	0	2.64e-3	5.67e-2	1.810
	Radau	10^{-3}	0.111	6590	310	1100	–	8.47e-5	1.115
$(97.5, 0.175, 10^{-5.975})$	ROM	$2 \cdot 10^{-4}$	0.057	1694	0	0	1.41e-3	3.08e-2	1.134
	Radau	10^{-3}	0.074	4649	212	744	–	1.43e-5	0.467
$(52.5, 0.025, 10^{-4.525})$	ROM	$2 \cdot 10^{-4}$	0.048	1454	0	0	3.45e-3	5.14e-2	5.976
	Radau	10^{-3}	0.063	3895	183	666	–	3.73e-5	2.885
$(97.5, 0.025, 10^{-4.525})$	ROM	$2 \cdot 10^{-4}$	0.049	1466	0	0	1.85e-3	6.27e-2	2.355
	Radau	10^{-3}	0.062	3809	172	658	–	8.83e-5	7.598
$(97.5, 0.175, 10^{-4.525})$	ROM	$2 \cdot 10^{-4}$	0.088	2690	0	0	2.17e-3	1.17e-2	1.366
	Radau	10^{-3}	0.117	7148	319	1202	–	1.33e-5	1.100
$(52.5, 0.175, 10^{-4.525})$	ROM	$2 \cdot 10^{-4}$	0.059	1770	0	0	2.57e-3	2.28e-2	1.920
	Radau	10^{-3}	0.077	4781	217	814	–	2.35e-5	1.102

Table 2: Test case 2: OREGO. Comparison of computational cost and accuracy for the Radau solver and the neural ODE based reduced order model (ROM) on the test dataset. Refer to Section 4 for the definition of the metrics.

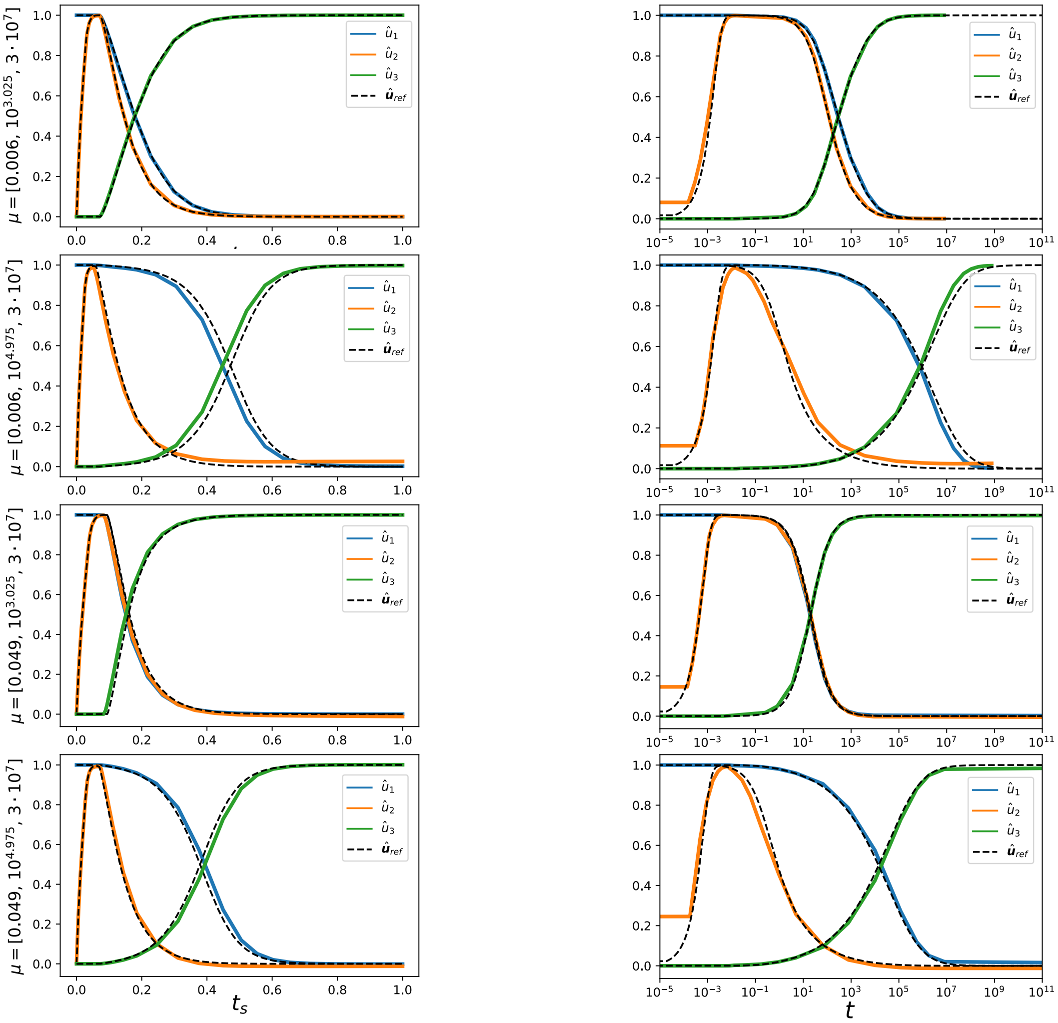

Figure 8: Test case 3: ROBER. On the left, in colors, the neural ODE predictions in t_s. On the right, in colors, the prediction is mapped to the original time t. In black is the reference solution.

that the ROM accurately captures the period of the system.

4.3 Test case 3: ROBER problem

The ROBER problem is a stiff system of 3 non-linear ODEs defined as follows:

$\boldsymbol{\mu}$	solver	rtoll (atoll)	time [s]	# fev	# jev	# lu	MSE_{t_s}	MSE
$(0.006, 10^{3.025}, 3\cdot 10^7)$	ROM	$2\cdot 10^{-4}$	0.008	230	0	0	1.21e-5	2.40e-4
	Radau	$10^{-4}(10^{-7})$	0.019	927	44	210	–	6.72e-6
$(0.006, 10^{4.975}, 3\cdot 10^7)$	ROM	$2\cdot 10^{-4}$	0.009	272	0	0	1.19e-3	1.11e-3
	Radau	$10^{-4}(10^{-7})$	0.021	1100	52	240	–	5.74e-6
$(0.049, 10^{3.025}, 3\cdot 10^7)$	ROM	$2\cdot 10^{-4}$	0.007	206	0	0	4.34e-4	4.97e-5
	Radau	$10^{-4}(10^{-7})$	0.021	1081	46	256	–	4.49e-6
$(0.049, 10^{4.975}, 3\cdot 10^7)$	ROM	$2\cdot 10^{-4}$	0.009	260	0	0	4.46e-4	9.80e-4
	Radau	$10^{-4}(10^{-7})$	0.024	1121	55	248	–	3.35e-6

Table 3: Test case 3: ROBER. Comparison of computational cost and accuracy for the Radau solver and the neural ODE based reduced order model (ROM) on the test dataset. Refer to Section 4 for the definition of the metrics.

$$\begin{cases} \dot{u_1} = -\mu_1 u_1 + \mu_2 u_2 u_3, \\ \dot{u_2} = \mu_1 u_1 - \mu_2 u_2 u_3 - \mu_3 u_2^2, \\ \dot{u_3} = \mu_3 u_2^2, \\ \boldsymbol{u}(0) = (1, 0, 0). \end{cases} \tag{11}$$

The system describes the kinetics of an autocatalytic chemical reaction and is considered one of the most popular benchmarks for stiff solvers. When used in this context, the time integration window used is usually large, namely $t \in [0, 10^{11})$ and the parameters are $\boldsymbol{\mu} = [0.04, 10^4, 3\cdot 10^7]$. Indeed, codes might fail if u_2 accidentally becomes negative, since it then tends to $-\infty$, causing overflow. This feature makes the ROBER problem ideal for testing the ability of our ROM to be accurate across different time scales.

We set $\Gamma = [0.005, 0.05] \times [10^3, 10^5] \times \{3\cdot 10^7\}$ and consider training dataset composed by a logarithmic discretization Γ_h of Γ with 16 uniform points and 31 logarithmically spaced points in the first two directions, respectively. The validation dataset is the interval midpoints of Γ_h. The reference solutions are computed using the Radau method, with absolute and relative tolerances set to `rtol`$= 10^{-10}$, `atol`$= 10^{-14}$.

Data is normalized using the following functions: $\hat{\boldsymbol{\mu}} = (-\log_{10}\mu_1, \log_{10}\mu_2/4, \log_{10}\mu_3/7)$; state normalization $\hat{\boldsymbol{u}} = (u_1, u_2/10^{(\hat{\mu_1}-\hat{\mu_3})/2}, u_3)$; dynamics normalization: $\hat{\boldsymbol{f}} = \boldsymbol{f}$. The model is tested for four values of the parameters

$$\boldsymbol{\mu}_{\text{test}} \in \{0.006, 0.49\} \times \{10^{3.025}, 10^{4.975}\}.$$

Figure 8 shows that the ROM is able to accurately follow the reference solution across the different time scales. A quantitative comparison with the Radau solver is reported in Table 3. We remark that a tolerance of `rtol`$= 10^{-4}$, `atol`$= 10^{-7}$ for the Radau solver is among the smallest pair of tolerances that produce a positive solution for all the problems in the test dataset. The number of right-hand side evaluations of the explicit solver is almost one order of magnitude smaller than the implicit one, showing that our method has made the system nonstiff. Moreover, the computational cost is consistently smaller than the one needed by the Radau solver. However, the latter achieves a far smaller error. We highlight that the ROM sometimes fails to reach the correct final time, namely, the map to t sometimes stops at a time much smaller than 10^{11}. Nevertheless, the ROM still captures the interesting part of the reaction, having issues only when the solution reaches a plateau.

4.4 Test case 4: E5 problem

The E5 problem consists of the following stiff system of 4 non-linear ordinary differential equations:

$$\begin{cases} \dot{u_1} = -\mu_1 u_1 + \mu_2 u_1 u_3, \\ \dot{u_2} = \mu_1 u_1 - \mu_3 \mu_4 u_2 u_3, \\ \dot{u_3} = \mu_1 u_1 - \mu_2 u_1 u_3 - \mu_3 \mu_4 u_2 u_3 + \mu_3 u_4, \\ \dot{u_4} = \mu_2 u_1 u_3 - \mu_3 u_4, \\ \boldsymbol{u}(0) = (1.76\cdot 10^{-3}, 0, 0, 0). \end{cases} \tag{12}$$

The problem describes a chemical pyrolysis where $\boldsymbol{u}$ represents the concentration of the reactants. The largely different rates of reaction that occur in the same system are the cause of stiffness. As a test problem, it is usually integrated in a large time window, namely, we choose the interval $t \in [0, 10^{11})$. This means the problem features widely different scales in time and space. The problem is particularly challenging. Indeed, the formulation (12) is affected by the cancellation of digits and it is recommended to use the relation $\dot{u}_3 = \dot{u}_2 - \dot{u}_4$ to compute $\boldsymbol{f}(t, \boldsymbol{u})$. Moreover, it has been shown that a very small tolerance on the absolute scalar error must be used to obtain a reliable reference solution. Indeed, they are computed using the Radau method, with absolute and relative tolerances set to `atol`$= 10^{-24}$, `rtol`$= 10^{-10}$.

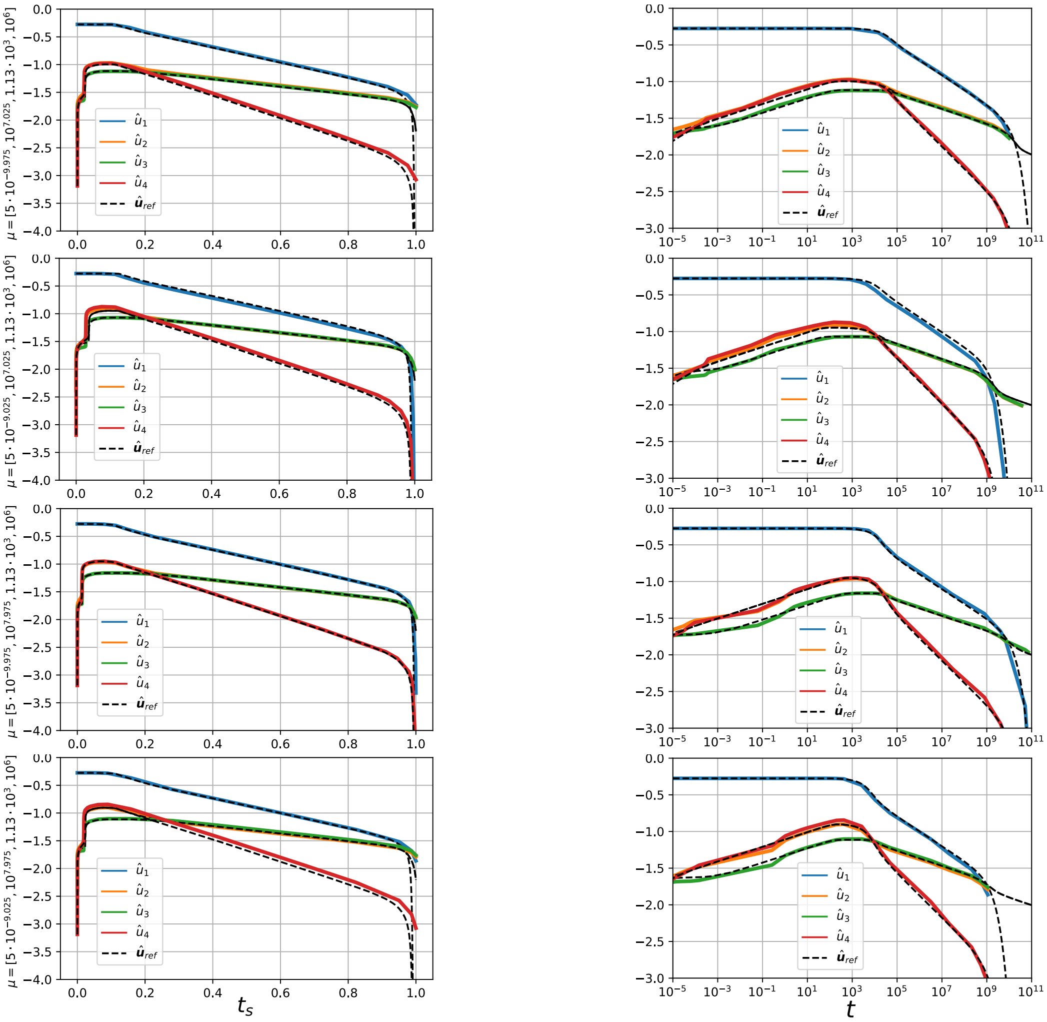

Figure 9: Test case 4: E5. On the left, in colors, the neural ODE predictions in t_s. On the right, in colors, the prediction is mapped to the original time t. In black is the reference solution. The state scale is logarithmic.

We set $\Gamma = [5 \cdot 10^{-10}, 5 \cdot 10^{-9}] \times [10^7, 10^8] \times \{1.13 \cdot 10^3\} \times \{10^6\}$ which contains the parameter $\boldsymbol{\mu} = (7.89 \cdot 10^{-10}, 1.1 \cdot 10^7, 1.13 \cdot 10^3, 10^6)$, which is the usual choice for the benchmark. The training dataset is built by a discretization Γ_h of Γ with 11 logarithmically spaced points in the first two directions. The validation dataset is the interval midpoints of Γ_h. Data is normalized using the following functions: $\hat{\boldsymbol{\mu}} = \log_{10} \mu$; state normalization $\hat{\boldsymbol{u}} = \log_{10} \boldsymbol{u}/10$; dynamics normalization:

$$\hat{\boldsymbol{f}} = \begin{cases} f_i & \text{if } |f_i| < 2, \\ \operatorname{sgn}(f_i)(\log(|f_i| - 1) + 2) & \text{otherwise.} \end{cases}$$

The insight behind this normalization is simple: the data features very long tails due to the presence of large gradients. Thus, we apply a logarithmic transformation to these large values. Suitable constants are added to connect with continuity with the linear part of the transformation.

The model is tested for four values of the parameters

$$\boldsymbol{\mu}_{\text{test}} \in \{5 \cdot 10^{-9.975}, 5 \cdot 10^{-9.025}\} \times \{10^{7.025}, 10^{7.975}\} \times \{1.13 \cdot 10^3\} \times \{10^6\}.$$

Figure 9 shows that the ROM is able to accurately follow the reference solution across the different time and space scales. A quantitative comparison with the Radau solver is reported in Table 4. We remark that a tolerance of `rtol`$= 10^{-6}$, `atol`$= 10^{-20}$ for the Radau solver is among the smallest pair of tolerances that produce a positive and stable solution for all the problems in the test dataset. The number of right-hand side evaluations of the explicit solver is more than one order of magnitude smaller than the implicit one, showing that our method has indeed made the system nonstiff. The work precision tradeoff is particularly favorable for our ROM. Indeed, it has comparable accuracy at a much lower computational cost. The main limitation of our model is that it sometimes fails to reach the correct final time, namely, the map to t sometimes stops at a time much smaller than 10^{11}. However, the ROM still captures the interesting part of the reaction, having issues only when the concentration plummets to values close to zero ($< 10^{-20}$).

$\boldsymbol{\mu}$	solver	rtoll (atoll)	time [s]	# fev	# jev	# lu	MSE$_{t_s}$	MSE
$(5 \cdot 10^{-9.975}, 10^{7.025}, 1.13 \cdot 10^3, 10^6)$	ROM	$2 \cdot 10^{-4}$	0.013	308	0	0	4.12e-2	5.06e-3
	Radau	$10^{-6}(10^{-20})$	0.064	4119	93	416	–	8.36e-4
$(5 \cdot 10^{-9.025}, 10^{7.025}, 1.13 \cdot 10^3, 10^6)$	ROM	$2 \cdot 10^{-4}$	0.015	344	0	0	4.97e-2	4.00e-3
	Radau	$10^{-6}(10^{-20})$	0.065	4156	102	426	–	1.84e-2
$(5 \cdot 10^{-9.975}, 10^{7.975}, 1.13 \cdot 10^3, 10^6)$	ROM	$2 \cdot 10^{-4}$	0.013	302	0	0	1.56e-2	4.80e-3
	Radau	$10^{-6}(10^{-20})$	0.063	4003	103	430	–	9.87e-4
$(5 \cdot 10^{-9.025}, 10^{7.975}, 1.13 \cdot 10^3, 10^6)$	ROM	$2 \cdot 10^{-4}$	0.011	260	0	0	6.48e-2	5.53e-3
	Radau	$10^{-6}(10^{-20})$	0.068	4055	112	438	–	1.88e-2

Table 4: Test case 4: E5. Comparison of computational cost and accuracy for the Radau solver and the neural ODE based reduced order model (ROM) on the test dataset. Refer to Section 4 for the definition of the metrics.

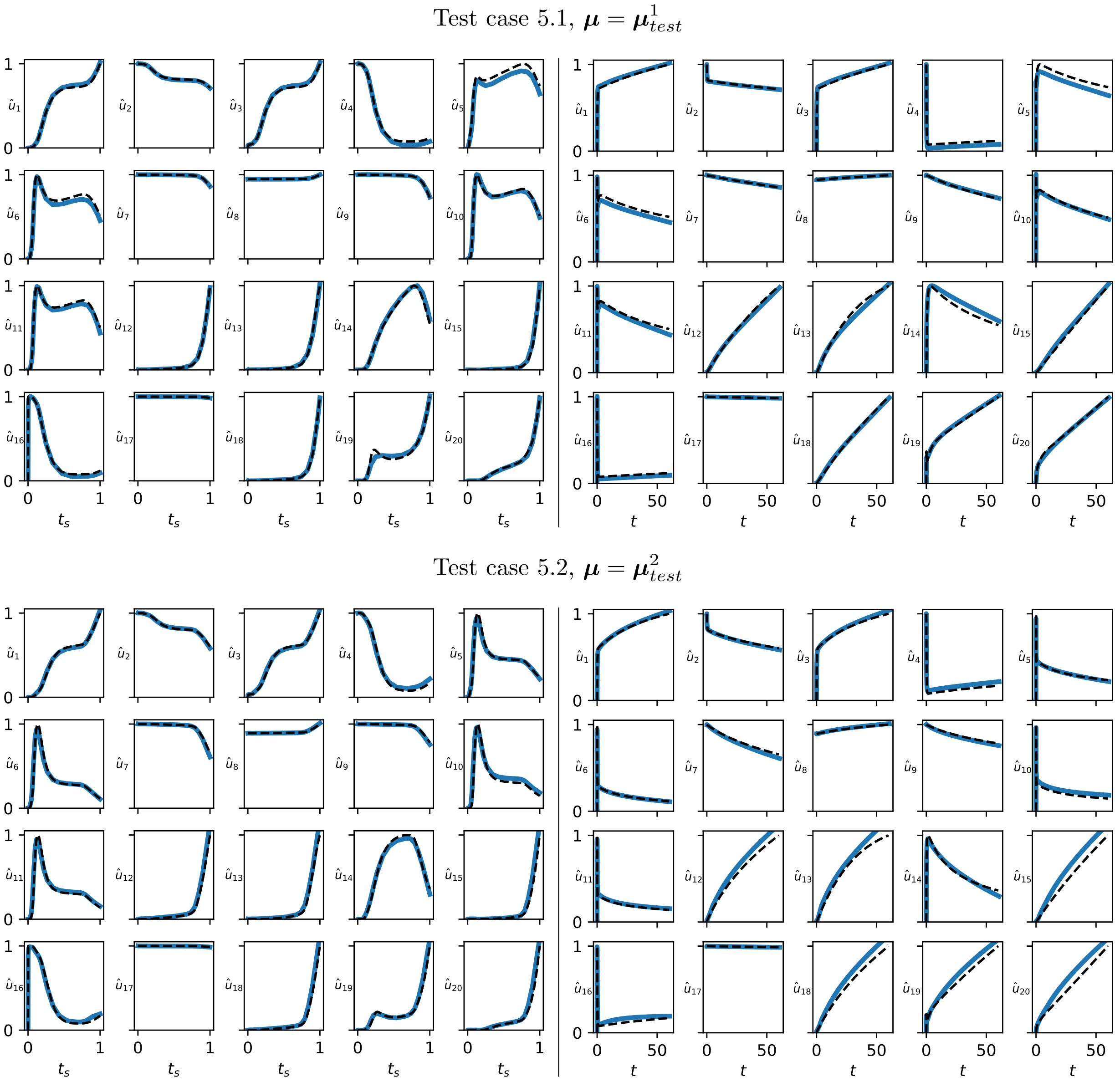

Figure 10: Test case 5: POLLU. On the left, the neural ODE predictions in t_s. On the right, the prediction is mapped to the original time t. In blue (—) the neural ODE prediction of $\hat{u}$, in black (- - -) the reference solution.

4.5 Test case 5: POLLU problem

To show the effectiveness of the proposed method on large state spaces, we test our methodology on the POLLU problem: as a stiff system of 20 non-linear ODEs. The system is the chemical reaction part of the air pollution model developed at the Dutch National Institute of Public Health and Environmental Protection. The problem features 25 reactions ($\boldsymbol{\mu} \in \mathbb{R}^{25}$) and 20 reacting compounds ($\boldsymbol{u} \in \mathbb{R}^{20}$). Namely, we have that the system is defined by

Reaction	$\boldsymbol{r}$	$\boldsymbol{\mu}^0$	Reaction	$\boldsymbol{r}$	$\boldsymbol{\mu}^0$	Reaction	$\boldsymbol{r}$	$\boldsymbol{\mu}^0$
1	$\mu_1 u_1$	0.350e0	10	$\mu_{10} u_{11} u_1$	0.900e4	19	$\mu_{19} u_{16}$	0.444e12
2	$\mu_2 u_2 u_4$	0.266e2	11	$\mu_{11} u_{13}$	0.220e-1	20	$\mu_{20} u_{17} u_6$	0.124e4
3	$\mu_3 u_5 u_2$	0.120e5	12	$\mu_{12} u_{10} u_2$	0.120e5	21	$\mu_{21} u_{19}$	0.210e1
4	$\mu_4 u_7$	0.860e-3	13	$\mu_{13} u_{14}$	0.188e1	22	$\mu_{22} u_{19}$	0.578e1
5	$\mu_5 u_7$	0.820e-3	14	$\mu_{14} u_1 u_6$	0.163e5	23	$\mu_{23} u_1 u_4$	0.474e-1
6	$\mu_6 u_7 u_6$	0.150e5	15	$\mu_{15} u_3$	0.480e7	24	$\mu_{24} u_{19} u_1$	0.178e4
7	$\mu_7 u_9$	0.130e-3	16	$\mu_{16} u_4$	0.350e-3	25	$\mu_{25} u_{20}$	0.312e1
8	$\mu_8 u_9 u_6$	0.240e5	17	$\mu_{17} u_4$	0.175e-1			
9	$\mu_9 u_{11} u_2$	0.165e5	18	$\mu_{18} u_{16}$	0.100e9			

Table 5: Reference reaction rates for the POLLU model that define $\boldsymbol{\mu}^0$.

$$\boldsymbol{f}(\boldsymbol{u};\boldsymbol{\mu}) = \begin{bmatrix} -r_1 - r_{10} - r_{14} - r_{23} - r_{24} + r_2 + r_3 + r_9 + r_{11} + r_{12} + r_{22} + r_{25} \\ -r_2 - r_3 - r_9 - r_{12} + r_1 + r_{21} \\ -r_{15} + r_1 + r_{17} + r_{19} + r_{22} \\ -r_2 - r_{16} - r_{17} - r_{23} + r_{15} \\ -r_3 + 2r_4 + r_6 + r_7 + r_{13} + r_{20} \\ -r_6 - r_8 - r_{14} - r_{20} + r_3 + 2r_{18} \\ -r_4 - r_5 - r_6 + r_{13} \\ r_4 + r_5 + r_6 + r_7 \\ -r_7 - r_8 \\ -r_{12} + r_7 + r_9 \\ -r_9 - r_{10} + r_8 + r_{11} \\ r_9 \\ -r_{11} + r_{10} \\ -r_{13} + r_{12} \\ r_{14} \\ -r_{18} - r_{19} + r_{16} \\ -r_{20} \\ r_{20} \\ -r_{21} - r_{22} - r_{24} + r_{23} + r_{25} \\ -r_{25} + r_{24} \end{bmatrix},$$

where the reactions $\boldsymbol{r}$ are connected to $\boldsymbol{u}$ and $\boldsymbol{\mu}^0$ by rates reported in Table 5. The initial condition is

$$\boldsymbol{u}_0 = (0, 0.2, 0, 0.04, 0, 0, 0.1, 0.3, 0.01, 0, 0, 0, 0, 0, 0, 0, 0.007, 0, 0, 0).$$

It is usually considered that the time interval $t \in [0, 60)$ is representative of the behavior of the reactants sufficiently. The reference solution is obtained by using a tolerance of 10^{-10}. Full details about the model can be found in [63]. The parameter space Γ is built by changing the three components of $\boldsymbol{\mu}$ that most impact the system. Namely, a preliminary sensibility analysis has shown that the components $4, 6$, and 14 of $\boldsymbol{\mu}$ are the most relevant, that is they produced the largest variation in the solution $\boldsymbol{u}$ when perturbed. Hence, we define the parameter space as

$$\Gamma = \prod_{i \notin \{4,6,14\}} \{\mu_i^0\} \times \prod_{i \in \{4,6,14\}} \left[\frac{1}{2}\mu_i^0, 2\mu_i^0\right],$$

where $\boldsymbol{\mu}^0$ is the reference parameter value used for the system, reported in Table 5. The intervals are discretized with 16 uniform points in each coordinate direction. The validation dataset is made by the midpoints of the training datasets.

The model is tested for two parameters $\boldsymbol{\mu}^1_{test}, \boldsymbol{\mu}^2_{test}$ which are equal to $\boldsymbol{\mu}^0$ apart from the three most relevant components, namely $\mu^1_{test,i} = 0.525\mu_i^0$, $\mu^2_{test,i} = 1.975\mu_i^0$, $i = 4, 6, 14$. Results are shown in Figure 10. The ROM is able to follow the reference solution across all the reactions. A quantitative comparison with the Radau solver is reported in Table 6. The number of right-hand side evaluations of the explicit solver is similar to the implicit one. On the other hand, each evaluation of the right-hand side, each evaluation of the Jacobian, and each LU decomposition is more costly than in the previous test cases since N_u is larger. However, the cost of evaluating the neural network is similar to the previous test cases since the dimensions of the networks are comparable. Indeed, the cost of the ROM is still smaller than the cost of the Radau solver while keeping comparable accuracy. These results are of particular significance because they prove that as the state space gets larger our methodology becomes more cost effective.

$\boldsymbol{\mu}$	solver	toll	time [s]	# fev	# jev	# lu	MSE_{t_s}	L^2
$\mu_i = 0.525\mu_i^0$, $i = 4, 6, 14$	ROM	10^{-4}	0.006	194	0	0	6.89e-4	9.80e-4
	Radau	10^{-3}	0.011	182	10	50	–	3.81e-3
$\mu_i = 1.975\mu_i^0$, $i = 4, 6, 14$	ROM	10^{-4}	0.007	212	0	0	1.16e-3	1.69e-3
	Radau	10^{-3}	0.017	268	14	70	–	9.03e-4

Table 6: Test case 5: POLLU. Comparison of computational cost and accuracy for the Radau solver and the neural ODE based reduced order model (ROM) on the test dataset. Refer to Section 4 for the definition of the metrics.

5 Conclusions

In this work, we have developed a novel methodology to tackle the stiffness in neural ODEs. The approach hinges on a suitable time reparametrization of the system that significantly reduces the stiffness, making it possible to efficiently apply explicit solvers to the neural ODEs. The construction of the time reparametrization is general and completely data-driven, making it ideal for an application to ROMs.

By leveraging the intrinsic capabilities of neural ODEs to handle continuous-time data, we capture the dynamics of stiff systems effectively. The accuracy, robustness, and efficiency of the methodology were tested in five famous benchmark test cases from the literature. Namely, when applied to periodic systems, the model generalizes to data well beyond the time training interval. We attribute this characteristic primarily to the autonomous nature of the neural ODE and the time reparametrization model. When applied to chemical equations, our model always maintained the positivity of the concentration, even when applied to problems suffering from numerical instability. Moreover, the work/precision tradeoff of our models always proved comparable or favorable when confronted with a state-of-the-art Runge-Kutta implicit solver of the Radau II A kind.

In conclusion, our work highlights the promising potential of neural ODEs in creating efficient and accurate reduced-order models for stiff ODEs. Future research is directed towards improving the accuracy of the time mapping that, as of right now, stands as the major accuracy bottleneck of the ROM. We also aim to test this framework to handle the semi-discrete formulation of partial differential equations and other higher-dimensional systems. Moreover, techniques like model distillation, network pruning, and better hyperparameter tuning could further reduce the computational cost of our approach.

Declaration of competing interests. 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.

Acknowledgments. MC has been partially funded by PRIN2020 n. 20204LN5N5 funded by the Italian Ministry of Universities and Research (MUR). MC is a member of INdAM-GNCS.

CRediT authorship contribution statement. MC: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Visualization, Writing - Original draft. JSH: Conceptualization, Methodology, Resources, Writing - Review and editing, Supervision, Project administration.

References

[1] R. Anantharaman, A. Abdelrehim, A. Jain, A. Pal, D. Sharp, Utkarsh, A. Edelman, and C. Rackauckas. Stably accelerating stiff quantitative systems pharmacology models: Continuous-time echo state networks as implicit machine learning. *IFAC-PapersOnLine*, 55(23):1–6, 2022. 9th IFAC Conference on Foundations of Systems Biology in Engineering FOSBE 2022.

[2] A. C. Antoulas. *Approximation of large-scale dynamical systems.* SIAM, 2005.

[3] M. Barrault, Y. Maday, N. C. Nguyen, and A. T. Patera. An 'empirical interpolation' method: application to efficient reduced-basis discretization of partial differential equations. *Comptes Rendus Mathematique*, 339(9):667–672, 2004.

[4] P. Benner, S. Gugercin, and K. Willcox. A survey of projection-based model reduction methods for parametric dynamical systems. *SIAM review*, 57(4):483–531, 2015.

[5] P. Benner, V. Mehrmann, and D. C. Sorensen. *Dimension reduction of large-scale systems*, volume 45. Springer, 2005.

[6] P. Benner, M. Ohlberger, A. Cohen, and K. Willcox. *Model reduction and approximation: theory and algorithms.* SIAM, 2017.

[7] M. Bergmann and L. Cordier. Optimal control of the cylinder wake in the laminar regime by trust-region methods and POD reduced-order models. *Journal of Computational Physics*, 227(16):7813–7840, 2008.

[8] S. L. Brunton and J. N. Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control.* Cambridge University Press, 2022.

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

[10] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

[11] T. Bui-Thanh, K. Willcox, and O. Ghattas. Model reduction for large-scale systems with high-dimensional parametric input space. *SIAM Journal on Scientific Computing*, 30(6):3270–3288, 2008.

[12] T. Bui-Thanh, K. Willcox, and O. Ghattas. Parametric reduced-order models for probabilistic analysis of unsteady aerodynamic applications. *AIAA journal*, 46(10):2520–2529, 2008.

[13] J. C. Butcher. On Runge-Kutta processes of high order. *Journal of the Australian Mathematical Society*, 4(2):179–194, 1964.

[14] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang. *Spectral methods: evolution to complex geometries and applications to fluid dynamics*. Springer Science & Business Media, 2007.

[15] C. Canuto, T. Tonn, and K. Urban. A posteriori error analysis of the reduced basis method for nonaffine parametrized nonlinear PDEs. *SIAM Journal on Numerical Analysis*, 47(3):2001–2022, 2009.

[16] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton. Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences*, 116(45):22445–22451, 2019.

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

[18] P. Chen, A. Quarteroni, and G. Rozza. Reduced basis methods for uncertainty quantification. *SIAM/ASA Journal on Uncertainty Quantification*, 5(1):813–869, 2017.

[19] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[20] B. Cockburn, G. E. Karniadakis, and C.-W. Shu. *Discontinuous Galerkin methods: theory, computation and applications*, volume 11. Springer Science & Business Media, 2012.

[21] P. Conti, M. Guo, A. Manzoni, and J. S. Hesthaven. Multi-fidelity surrogate modeling using long short-term memory networks. *Computer methods in applied mechanics and engineering*, 404:115811, 2023.

[22] T. Cui, Y. M. Marzouk, and K. E. Willcox. Data-driven model reduction for the Bayesian solution of inverse problems. *International Journal for Numerical Methods in Engineering*, 102(5):966–990, 2015.

[23] M. De Florio, E. Schiassi, and R. Furfaro. Physics-informed neural networks and functional interpolation for stiff chemical kinetics. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 32(6):063107, 06 2022.

[24] S. Elfwing, E. Uchibe, and K. Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11, 2018.

[25] M. Frangos, Y. Marzouk, K. Willcox, and B. van Bloemen Waanders. Surrogate and reduced-order modeling: a comparison of approaches for large-scale statistical inverse problems. *Large-Scale Inverse Problems and Quantification of Uncertainty*, pages 123–149, 2010.

[26] S. Fresca, L. Dede', and A. Manzoni. A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs. *Journal of Scientific Computing*, 87:1–36, 2021.

[27] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[28] S. Goswami, A. D. Jagtap, H. Babaee, B. T. Susi, and G. E. Karniadakis. Learning stiff chemical kinetics using extended deep neural operators. *Computer Methods in Applied Mechanics and Engineering*, 419:116674, 2024.

[29] M. Guo and J. S. Hesthaven. Data-driven reduced order modeling for time-dependent problems. *Computer Methods in Applied Mechanics and Engineering*, 345:75–99, 2019.

[30] E. Hairer and G. Wanner. Stiff differential equations solved by Radau methods. *Journal of Computational and Applied Mathematics*, 111(1):93–111, 1999.

[31] D. Hendrycks and K. Gimpel. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

[32] J. S. Hesthaven, C. Pagliantini, and G. Rozza. Reduced basis methods for time-dependent problems. *Acta Numerica*, 31:265–345, 2022.

[33] J. S. Hesthaven, G. Rozza, B. Stamm, et al. *Certified reduced basis methods for parametrized partial differential equations*, volume 590. Springer, 2016.

[34] J. S. Hesthaven and S. Ubbiali. Non-intrusive reduced order modeling of nonlinear problems using neural networks. *Journal of Computational Physics*, 363:55–78, 2018.

[35] J. S. Hesthaven and T. Warburton. *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*. Springer Science & Business Media, 2007.

[36] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, et al. Searching for MobileNetV3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.

[37] W. Ji, W. Qiu, Z. Shi, S. Pan, and S. Deng. Stiff-pinn: Physics-informed neural network for stiff chemical kinetics. *The Journal of Physical Chemistry A*, 125(36):8098–8106, 2021.

[38] S. Kim, W. Ji, S. Deng, Y. Ma, and C. Rackauckas. Stiff neural ordinary differential equations. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 31(9), 09 2021.

[39] T. Lähivaara, A. Malehmir, A. Pasanen, L. Kärkkäinen, J. M. Huttunen, and J. S. Hesthaven. Estimation of groundwater storage from seismic data using deep learning. *Geophysical Prospecting*, 67(8):2115–2126, 2019.

[40] Z. Li, D. Z. Huang, B. Liu, and A. Anandkumar. Fourier neural operator with learned deformations for pdes on general geometries. *Journal of Machine Learning Research*, 24(388):1–26, 2023.

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

[42] Y. Liu, J. N. Kutz, and S. L. Brunton. Hierarchical deep learning of multiscale differential equation time-steppers. *Philosophical Transactions of the Royal Society A*, 380(2229):20210200, 2022.

[43] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[44] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature machine intelligence*, 3(3):218–229, 2021.

[45] Y. Maday, A. T. Patera, J. D. Penn, and M. Yano. A parameterized-background data-weak approach to variational data assimilation: formulation, analysis, and application to acoustics. *International Journal for Numerical Methods in Engineering*, 102(5):933–965, 2015.

[46] A. Manzoni, A. Quarteroni, and G. Rozza. Shape optimization for viscous flows by reduced basis methods and free-form deformation. *International Journal for Numerical Methods in Fluids*, 70(5):646–670, 2012.

[47] R. Maulik, B. Lusch, and P. Balaprakash. Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders. *Physics of Fluids*, 33(3), 2021.

[48] F. Mazzia, F. Iavernaro, and C. Magherini. Test set for initial value problem solvers. *Department of Mathematics, University of Bari*, 2003.

[49] F. Negri, G. Rozza, A. Manzoni, and A. Quarteroni. Reduced basis method for parametrized elliptic optimal control problems. *SIAM Journal on Scientific Computing*, 35(5):A2316–A2340, 2013.

[50] V. Oommen, K. Shukla, S. Goswami, R. Dingreville, and G. E. Karniadakis. Learning two-phase microstructure evolution using neural operators and autoencoder architectures. *npj Computational Materials*, 8(1):190, 2022.

[51] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in Pytorch. *NIPS 2017 Workshop on Autodiff*, 2017.

[52] L. Pegolotti, M. R. Pfaller, N. L. Rubio, K. Ding, R. B. Brufau, E. Darve, and A. L. Marsden. Learning reduced-order models for cardiovascular simulations with graph neural networks. *Computers in Biology and Medicine*, 168:107676, 2024.

[53] M. Peirlinck, F. S. Costabal, J. Yao, J. Guccione, S. Tripathy, Y. Wang, D. Ozturk, P. Segars, T. Morrison, S. Levine, et al. Precision medicine in human heart modeling: Perspectives, challenges, and opportunities. *Biomechanics and modeling in mechanobiology*, 20:803–831, 2021.

[54] F. Pichi, B. Moya, and J. S. Hesthaven. A graph convolutional autoencoder approach to model order reduction for parametrized PDEs. *Journal of Computational Physics*, 501:112762, 2024.

[55] G. D. Portwood, P. P. Mitra, M. D. Ribeiro, T. M. Nguyen, B. T. Nadiga, J. A. Saenz, M. Chertkov, A. Garg, A. Anandkumar, A. Dengel, R. Baraniuk, and D. P. Schmidt. Turbulence forecasting via Neural ODE, Nov. 2019. arXiv:1911.05180 [physics].

[56] C. Prud'Homme, D. V. Rovas, K. Veroy, L. Machiels, Y. Maday, A. T. Patera, and G. Turinici. Reliable real-time solution of parametrized partial differential equations: Reduced-basis output bound methods. *J. Fluids Eng.*, 124(1):70–80, 2002.

[57] S. S. Ravindran. A reduced-order approach for optimal control of fluids using proper orthogonal decomposition. *International journal for numerical methods in fluids*, 34(5):425–448, 2000.

[58] F. Regazzoni, L. Dede, and A. Quarteroni. Machine learning for fast and reliable solution of time-dependent differential equations. *Journal of Computational physics*, 397:108852, 2019.

[59] F. Regazzoni, S. Pagani, M. Salvador, L. Dede', and A. Quarteroni. Learning the intrinsic dynamics of spatio-temporal processes through Latent Dynamics Networks. *Nature Communications*, 15(1):1834, Feb 2024.

[60] P. J. Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of fluid mechanics*, 656:5–28, 2010.

[61] L. F. Shampine and S. Thompson. Stiff systems. *Scholarpedia*, 2(3):2855, 2007.

[62] B. Sudret and A. Der Kiureghian. *Stochastic finite element methods and reliability: a state-of-the-art report.* Department of Civil and Environmental Engineering, University of California, 2000.

[63] J. G. Verwer. Gauss–Seidel iteration for stiff ODEs from chemical kinetics. *SIAM Journal on Scientific Computing*, 15(5):1243–1250, 1994.

[64] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[65] P. R. Vlachas, G. Arampatzis, C. Uhler, and P. Koumoutsakos. Multiscale simulations of complex systems by learning their effective dynamics. *Nature Machine Intelligence*, 4(4):359–366, 2022.

[66] Q. Wang, J. S. Hesthaven, and D. Ray. Non-intrusive reduced order modeling of unsteady flows using artificial neural networks with application to a combustion problem. *Journal of computational physics*, 384:289–307, 2019.

[67] Q. Wang, N. Ripamonti, and J. S. Hesthaven. Recurrent neural network closure of parametric POD-Galerkin reduced-order models based on the Mori-Zwanzig formalism. *Journal of Computational Physics*, 410:109402, 2020.

[68] S. Wang, X. Yu, and P. Perdikaris. When and why PINNs fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.

[69] G. Wanner and E. Hairer. *Solving ordinary differential equations II*, volume 375. Springer Berlin Heidelberg New York, 1996.

[70] G. Wen, Z. Li, K. Azizzadenesheli, A. Anandkumar, and S. M. Benson. U-FNO – An enhanced Fourier neural operator-based deep-learning model for multiphase flow. *Advances in Water Resources*, 163:104180, 2022.

[71] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. *Dive into Deep Learning.* Cambridge University Press, 2023.

[72] P. Zhang and R. Sankaran. Autoencoder neural network for chemically reacting systems. *Journal of Machine Learning for Modeling and Computing*, 3(4):1–28, 2022.

[73] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu. *The finite element method: its basis and fundamentals.* Elsevier, 2005.

MOX Technical Reports, last issues

Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

Negrini, G.; Parolini, N.; Verani, M.
An Immersed Boundary Method for Polymeric Continuous Mixing

52/2024 Botti, M.; Mascotto, L.
A Necas-Lions inequality with symmetric gradients on star-shaped domains based on a first order Babuska-Aziz inequality

51/2024 Antonietti, P.F.; Corti, M.; Lorenzon, G.
A discontinuous Galerkin method for the three-dimensional heterodimer model with application to prion-like proteins' dynamics

50/2024 Fumagalli, I.; Parolini, N.; Verani, M.
A posteriori error analysis for a coupled Stokes-poroelastic system with multiple compartments

49/2024 Ballini, E.; Formaggia, L.; Fumagalli, A.; Keilegavlen, E.; Scotti, A.
A hybrid upwind scheme for two-phase flow in fractured porous media

Ballini, E.; Formaggia, L.; Fumagalli, A.; Keilegavlen, E.; Scotti, A.
A hybrid upwind scheme for two-phase flow in fractured porous media

48/2024 Cicalese, G.; Ciaramella, G.; Mazzieri, I.
Addressing Atmospheric Absorption in Adaptive Rectangular Decomposition

47/2024 Franco, N.R.; Brugiapaglia, S.
A practical existence theorem for reduced order models based on convolutional autoencoders

46/2024 Riccobelli, D.; Ciarletta, P.; Vitale, G.; Maurini, C.; Truskinovsky, L.
Elastic Instability behind Brittle Fracture

45/2024 Fumagalli, A.; Patacchini, F. S.
Numerical validation of an adaptive model for the determination of nonlinear-flow regions in highly heterogeneos porous media