Journal of Environmental Management 373 (2025) 123870

Contents lists available at ScienceDirect

Journal of Environmental Management

journal homepage: www.elsevier.com/locate/jenvman

Research article

Training stiff neural ordinary differential equations in data-driven wastewater process modelling

Xiangjun Huang [a,*], Kyriakos Kandris [b], Evina Katsou [b]

[a] Department of Civil and Environmental Engineering, Brunel University London, UK
[b] Department of Civil and Environmental Engineering, Imperial College London, UK

1. Introduction

Neural ordinary differential equations (NODEs) have emerged as a powerful tool for data-driven modelling of dynamical systems (R. Chen et al., 2018). Unlike traditional machine learning methods such as multilayer perceptrons and recurrent neural network operate in discrete time steps, NODEs excel in learning underlying complex dynamics evolving over continuous time and extracting mechanistic insights from monitoring data, even when the data are irregular. They have shown advantages in various tasks, including prediction (Dupont et al., 2019; Kidger et al., 2020; Núñez et al., 2023), kinetic parameter estimation (Bradley and Boukouvala, 2021; Kong et al., 2022), hybrid modelling (Quaghebeur et al., 2022), optimal control (Böttcher and Asikis, 2022; Sandoval et al., 2022), and physics-informed neural networks (PINN) (Cuomo et al., 2022; Gusmão et al., 2022; Karniadakis et al., 2021; Xue et al., 2021).

Despite these advancements, the application of NODEs in wastewater modelling remains limited. Effective training of NODEs using monitoring data is crucial for successful implementation. Our attempts to apply standard NODEs to the activated sludge model no.1 (ASM1) resulted in failed training (see Figs. 6 and 7). Similar challenges were faced when modelling N_2O in MATLAB (The MathWorks Inc, 2023) (see Figs. 15 and 16). Stiffness in the system, identified as a key issue (Kim et al., 2021), complicates the training process.

While some studies propose methods to stabilise the training process of stiff NODEs, such as equation scaling along with stabilised gradient calculation (Kim et al., 2021), our experiments with these approaches frequently encountered underflow errors, leading to premature termination. The stiffness, arising from the presence of widely disparate time scales in the dynamics, necessitates small time steps for numerical stability and can lead to pathological gradients, hindering training convergence.

Our work aims to address this issue in data-driven wastewater modelling with NODEs by proposing two methods.

1) Normalisation method: We adapt normalisation techniques from conventional machine learning (Shanker M et al., 1996), to wrap a pair of state normalisation and derivative de-normalisation layers around neural network to scale the state inputs and derivative outputs, alleviating the burden on the ODE solver, and improving the training efficiency.
2) Collocation method: By employing collocation techniques, we bypass the need for ODE solvers, directly interpolating and regressing derivatives at desired points for a faster solution.

We also propose an incremental strategy that starts with the collocation method for initial training, followed by NODEs training with normalisation. This strategy stabilises the learning process, saves time, and refines results more effectively than using either method alone.

Our experiments demonstrate the successful implementation of these methods in modelling the ASM1 and ASM2d-N_2O models using NODEs. As the concept of NODEs is relatively new, we first introduce their methodologies in the following section.

2. Background

2.1. Neural ordinary differential equations

Wastewater process models, such as the activated sludge model series, employ a set of ordinary differential equation systems to describe the dynamics of biochemical reactions between biomass and substrate. (Mogens et al., 2000a,b). Mathematically, they generalise mechanistic relationships between fractionated components and their derivatives with respect to the independent variable, time (t).

$$\frac{d\boldsymbol{S}(\boldsymbol{t})}{d\boldsymbol{t}} = \boldsymbol{f}(\boldsymbol{S}(\boldsymbol{t}), \boldsymbol{t}) \qquad \text{Equation 2-1}$$

where $S(t)$ denotes variables of fractionised wastewater components, such as readily biodegradable substrate (S_S), active heterotrophic

* Corresponding author.
E-mail address: xiangjun.huang@brunel.ac.uk (X. Huang).

https://doi.org/10.1016/j.jenvman.2024.123870
Received 31 July 2024; Received in revised form 9 November 2024; Accepted 23 December 2024
Available online 30 December 2024

biomass (X_{BH}), or particulate biodegradable organic nitrogen (X_{ND}) etc., presenting the states. The function f is the core demanding extensive domain expertise and testing to formulate (Tchobanoglous et al., 2014).

NODEs offer a new opportunity to reveal the function f based on observed data. Drawing upon the universal approximation theorem (Elbrächter et al., 2019; Pinkus, 1999), NODEs construct a neural network that approximates f:

$$\frac{d\boldsymbol{S}(\boldsymbol{t})}{\boldsymbol{dt}} = \boldsymbol{NN}(\boldsymbol{S}(\boldsymbol{t}), \boldsymbol{t}, \boldsymbol{\theta}) \quad \text{Equation 2-2}$$

where NN denotes the neural network with the weight and bias parameters represented by θ.

Training of the NN in NODEs is then designed to be made by means of integration of the ODEs with only state component data through time steps from initial condition Y (t_0), e.g., at the time when influent enters bioreactor, to the final condition at the anticipated time step t_n, e.g., the hydraulic retention time (HRT) when the flow leaves the reactor as effluent.

$$\boldsymbol{S}(\boldsymbol{t_n}) = \boldsymbol{S}(\boldsymbol{t_0}) + \int_{t_0}^{t_n} \boldsymbol{NN}(\boldsymbol{S}(\boldsymbol{t}))\boldsymbol{dt} = \boldsymbol{ODESolver}(\boldsymbol{NN}, \boldsymbol{S}(\boldsymbol{t_0}), (\boldsymbol{t_0}, \boldsymbol{t_n}), \boldsymbol{\theta}) \quad \text{Equation 2-3}$$

The loss function $L(\theta)$ can be defined using the mean absolute error (MAE):

$$\boldsymbol{L}(\boldsymbol{\theta}) = \frac{\boldsymbol{1}}{\boldsymbol{n}} \sum_{i=1}^{n} \left| \boldsymbol{S}(\boldsymbol{t_i})^{\boldsymbol{NN}} - \boldsymbol{S}(\boldsymbol{t_i})^{\boldsymbol{observation}} \right| = \frac{\boldsymbol{1}}{\boldsymbol{n}} \sum_{i=1}^{n} \left| \boldsymbol{ODESolver}(\boldsymbol{NN}, \boldsymbol{S}(\boldsymbol{t_0}), (\boldsymbol{t_0}, \boldsymbol{t_i}), \boldsymbol{\theta}) - \boldsymbol{S}(\boldsymbol{t_i})^{\boldsymbol{observation}} \right| \quad \text{Equation 2-4}$$

Building upon the above loss function, gradient descent algorithms can then be employed to iteratively update and optimise the weights and biases of NN until the desired level of accuracy is attained.

As a remedy to high memory usage and additional numerical errors by forward pass differentiation, the adjoint sensitivity method is proposed for NODEs training (R. Chen et al., 2018). by introducing the hidden states $\boldsymbol{h}(t)$ and the adjoint $a(t)$, which is the gradient of the loss with respect to the hidden state:

$$\boldsymbol{a}(\boldsymbol{t}) = \frac{\partial \boldsymbol{L}}{\partial \boldsymbol{h}(\boldsymbol{t})} \quad \text{Equation 2-5}$$

So, θ, the weight and bias of the NN can be optimised by computing the gradient through integration of $\boldsymbol{h}(t)$ and $a(t)$:

$$\frac{\boldsymbol{dL}}{\boldsymbol{d\theta}} = \int_{t_n}^{t_0} \boldsymbol{a}(\boldsymbol{t})^T \frac{\partial \boldsymbol{NN}(\boldsymbol{h}(\boldsymbol{t}), \boldsymbol{t}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \boldsymbol{dt} \quad \text{Equation 2-6}$$

However, in practice, the numerical errors from reversing ODE solver can propagate and amplify into gradient flow, rending it ill-conditioned. Even a minuscule error in the forward pass can lead to a pronounced deviation in the reversing solution (Kim et al., 2021). This situation deteriorates when stiffness issue already exists in the studied systems.

2.2. Stiff ODE systems in wastewater

Most real-world systems of ODEs require numerical methods, as analytical solutions are often unavailable or impractical. Stiffness arises when certain numerical methods fail to provide stable solutions unless the step size to be taken is extremely small (Hairer and Wanner, 1996a). This phenomenon is surprisingly common in many real-life problems (Kushnir and Rokhlin, 2012). Despite its prevalence, a rigorous mathematical definition for stiffness remains undetermined (Kushnir and Rokhlin, 2012).

The "stiffness ratio" is sometimes utilised to quantify system stiffness, defined as the product of the time span and the ratio of the real part of the fastest eigenvalue ($\bar{\lambda}$) and slowest eigenvalue ($\underline{\lambda}$) of the ODE system's Jacobian:

$$\boldsymbol{Stiffness\ Ratio} = \frac{|\boldsymbol{Re}(\bar{\boldsymbol{\lambda}})|}{|\boldsymbol{Re}(\underline{\boldsymbol{\lambda}})|} (\boldsymbol{t_1} - \boldsymbol{t_0}) \quad \text{Equation 2-7}$$

Empirically, stiff systems often exhibit significant disparities in the rate of change among various components. This disparity manifests as one component evolving slowly over time while another undergoes abrupt or swift changes, attributable to the system's distinctive chemical or biological kinetics. Apparently, the time span plays a crucial role in the issue. For long-time simulations, the issue can become severely problematic.

Wastewater processes exemplify these differences in scales and evolving dynamics. They involve components with high concentrations changing slowly (e.g., heterogeneous biomass) and transient components or intermediate products with low concentrations changing rapidly (e.g., dissolved oxygen, soluble substrate, hydroxylamine). With hydraulic retention time (HRT) of bioreactor typically ranging from 4 to 20 h, stiffness becomes evident.

Given an example from the ASM1 model, to obtain a stable solution within acceptable tolerances, it is advised (Mogens et al., 2000a,b) that when applying numerical methods to the model implementation, the maximum time step size should be less than:

$$\Delta \boldsymbol{t} < \frac{\boldsymbol{V_{\mathscr{k}}}\, \boldsymbol{C_{\mathscr{k}i}}}{\boldsymbol{O_{\mathscr{k}i}} + \boldsymbol{K_{\mathscr{k}i}}} = \boldsymbol{\theta_{\mathscr{k}i}} \quad \text{Equation 2-8}$$

Where $V_{\mathscr{k}}$ is the volume of reactor compartment $\mathscr{k}$. $C_{\mathscr{k}i}$, $O_{\mathscr{k}i}$, and $K_{\mathscr{k}i}$ are the concentration, output transport terms, and consumption terms of component i in reactor compartment $\mathscr{k}$ respectively. The term $\theta_{\mathscr{k}i}$ is the mean residence time of component i in compartment $\mathscr{k}$ at steady state. With default values, $\theta_{\mathscr{k}i}$ is of the order of 10 min for X_{BH}, X_{BA}, X_P, X_S and X_{ND}, of 1 min for S_S, S_{ND}, S_{NH}, S_{ALK} but of 1 s for S_o (Mogens et al., 2000a, b). The time step adopted in ODE solver typically ranges from 5 to 20% of the advised maximum step for a trade-off between sufficient accuracy and acceptable computational cost. If it is large than $\theta_{\mathscr{k}i}$, the correctness of the results cannot be guaranteed.

Fig. 1 illustrates a continuous stirred-tank reactor (CSTR) example modelled using ASM1, evolving from an initial concentration (see Supplement Table 2) over 6 h, with consistent dissolved oxygen (DO) control at 2 mg/l. It demonstrates the rapid evolution of various components and their first-order derivatives with rates of change ranging from approximately −0.02 mg/(l·d) to 8000 mg/(l·d). Additionally, the figure highlights the asynchronous occurrence of steep and flat segments for each component curve, indicating differing temporal dynamics.

The stiffness issue can be further intensified in more complex models, such as the ASM2d-N_2O model (Massara et al., 2018), which encompasses more volatile and intermediate components and more intricate reactions. For instance, the consumption rate of fermentable substrate (S_f), can rapidly decline from 9000 mg/(l·d) to nearly zero within 1 min, while oxygen uptake rate (OUR) may fluctuate around 6000 mg/(l·d). In contrast, nitric oxide (S_{NO}) evolves considerably slowly, ranging

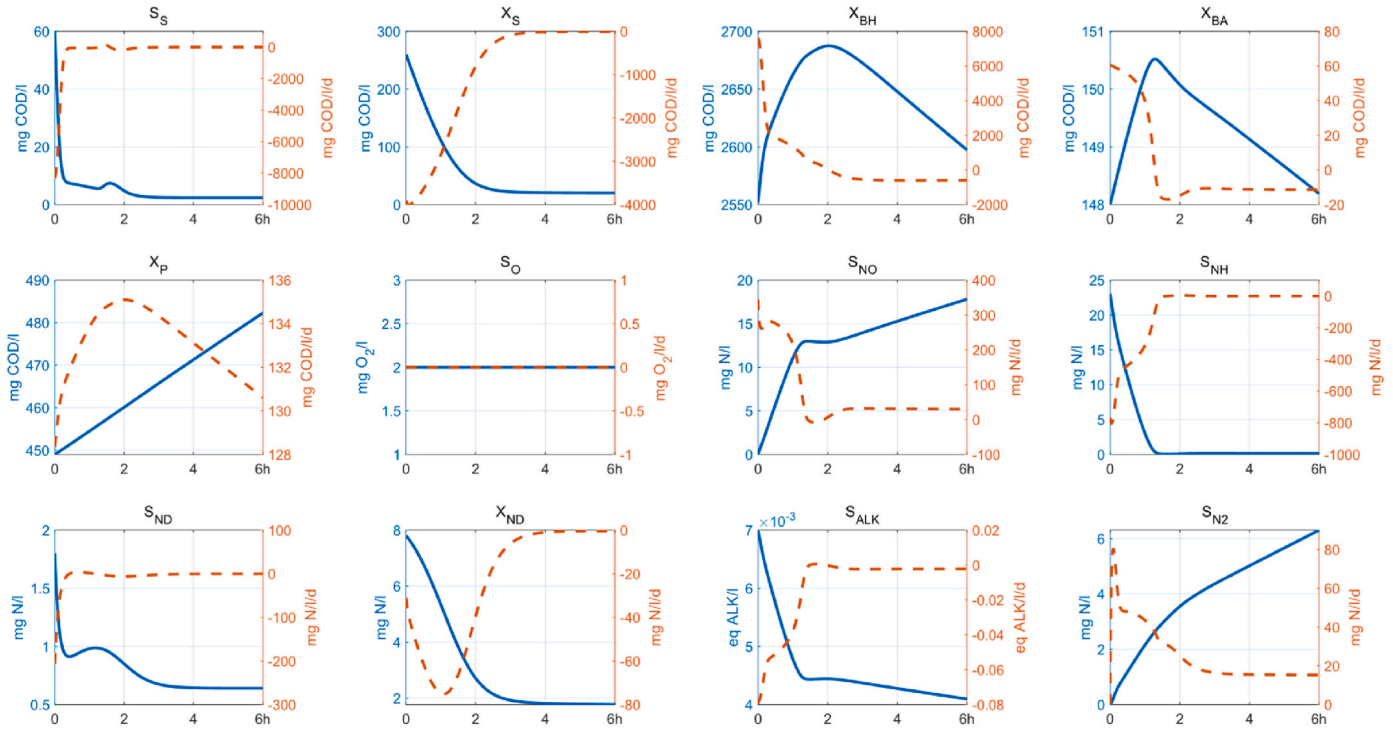

Fig. 1. Components (solid line) and its derivatives (dot line) in a CSTR of ASM1 model with constant DO control at 2 mg/l.

between 0 and 5×10^{-4} mg/l throughout the entire period.

Differential algebraic equations (DAEs) present another source of stiffness. In wastewater modelling, process controls, such as aeration and external carbon input, are often expressed in algebraic form, making DAEs inevitable. DAEs are considered a form of infinite stiffness (Hairer and Wanner, 1996b; Linda, 1982) and some ODEs are incompatible with them.

Stiffness remains a significant challenge in numerical analysis of differential equations (Postawa et al., 2020). While the advancement of specialised algorithms offers promising solutions, the effectiveness of solvers varies greatly. Some solvers employing adaptive or implicit methods can effectively tackle stiff ODEs. However, others may struggle or fail entirely when applied to the stiff systems. This highlights that there is no single "best" algorithm for all stiff problems (Kushnir and Rokhlin, 2012). Instead, suitability of a solver depends on the specific characteristics of the system. Consequently, researchers often resort to trial-and-error method to identify an appropriate solution based on solver features and the studied system's behaviour.

2.3. Training of stiff NODEs

Despite employing carefully selected solvers, known to be effective for stiff mechanistic ODEs, training of the corresponding NODEs with the same solvers remains a challenging task. This discrepancy stems from the differences in stiffness between NODEs and their mechanistic counterparts.

In mechanistic ODE systems, stiffness is inherent in the mathematical formulation of the problem. Instead, stiffness in NODEs emerges from the learned underlying dynamics, influenced by factors such as the neural network's architecture, randomness in initialization and updating, and noise or uncertainty in the training data. Deep neural networks in NODEs, initialized with random weights and biases, can lead to regions with vastly different rates of change, especially without adequate regularization. This can exacerbate stiffness. During training, the constantly evolving neural network function, due to gradient descent optimisation, introduces variations and randomness into the Jacobian of the approximated ODEs. This dynamic nature can amplify stiffness, potentially causing instability and hindering convergence.

Furthermore, real-world measurement data inevitably contains noise, which disrupts smoothness and amplifies errors in derivatives. Slight deviations in an ODE's initial state can result in significant divergences over time due to the accumulation of truncation and round-off errors (Gear, 1981; Hairer and Wanner, 1996a). In the context of NODEs, noise introduced by the data can behave similarly, with tiny errors propagating and amplifying through future steps, potentially causing the model to derail after a certain period. Consequently, training stiff NODEs with noisy data presents an additional challenge.

In summary, NODEs can exhibit more stiffness compared to the mechanistic ODE systems, unless specific techniques are employed to handle it (Fronk and Petzold, 2024).

2.3.1. Stiffness-induced numerical errors

Numerical errors are inevitable in solving real-world problems. It is not a concern if they can be controlled within the tolerance in the solution. The real concern in stiff NODEs is the accumulation and amplification of these errors to a significant level during numerical integration (Zhu et al., 2022). For example, the adjoint method, commonly used for backpropagation in NODEs, requiring double times calls of ODE solver, may encounter exponentially amplification of errors into the adjoint calculations, potentially leading to numerical blow-up (Kim et al., 2021).

In the training of stiff NODEs, stiffness-induced numerical errors can cause inaccurate gradients and unstable training. These errors can grow exponentially over time, causing the training process fail to converge on a meaningful solution. (Zhu et al., 2024). The amplified errors can also result in poor generalization of the trained model as the model learns the artifacts of the numerical errors rather than the actual dynamics of the system. (Kim et al., 2021).

Researchers are dedicated to deal with numerical integration errors for NODEs (Du et al., 2022; Fronk and Petzold, 2024; Zhu et al., 2022, 2024). They offer techniques for error estimation and analysis, which can be helpful in understanding and mitigating stiffness-induced errors.

2.3.2. Stiffness-induced scale separation

More importantly, the randomness and amplified numerical errors can intensify scale separation that already exists in stiff systems. Such enlarged scale separation can lead to loss imbalance, causing the optimiser to prioritise updates for rapidly changing variables and neglect slower ones. It can also cause vanishing or exploding gradients during backpropagation.

In summary, training stiff NODEs is challenging due to several factors. Firstly, stiff systems involve processes with vastly different scales, making it difficult for neural network training algorithms to learn both fast and slow dynamics simultaneously. Secondly, standard gradient descent methods can suffer from amplification of stiffness-induced numerical errors and ill-conditioned gradients during backpropagation, hindering effective weight adjustment. Thirdly, the use of multistep methods in ODE solvers, which rely on past information, introduces complexity and slows down training, especially when dealing with noisy data.

3. Methods

Stiffness challenges in NODEs are often problem-specific, demanding empirical solutions through experimentation. The key to successfully training stiff NODEs lies in maintaining stable gradient computations, avoiding ill-conditioning and balancing the loss contribution. We propose two methods and a strategy to tackle this issue in data-driven modelling using NODEs.

1) **Normalisation method:** We propose a readily deployable normalisation pair within the NODEs neural network that effectively stabilise the training process.
2) **Collocation method:** As an alternative, we present a collocation-based approach that bypasses the need for an ODE solver entirely, thereby circumventing stiffness problems.
3) **Incremental training strategy:** We introduce an incremental strategy that leverages both collocation and direct NODE training sequentially for a more efficient optimisation process when dealing with stiff systems.

3.1. Normalisation method

In machine learning regression tasks, data transformation through scaling is often imperative. This is because algorithms used in the training process, such as gradient descent adopted in our NODEs training, are sensitive to feature variance (Amari, 1993). Scaling or normalisation transforms data to be dimensionless and/or have comparable distribution scales. Especially for the training data which exhibit large feature variability, normalisation ensures each feature contributes equally, and prevents features with higher magnitudes from dominating. Empirical and theoretical evidence supports that normalisation can reduce the risk of vanishing/exploding gradient problem, ensures a well-conditioned optimisation, and accelerates convergence (Huang et al., 2023).

Unnormalized data with large scale separation can cause steep oscillations in the cost function, obstructing the gradient descent optimisation process. This hindrance can result in slow or failed convergence, as gradient descent may struggle to find the optimal solution efficiently (Bhanja and Das, 2018; Cabello-Solorzano et al., 2023).

In the context of stiff NODEs, which employ deep neural networks and involve training data with high scale separation, normalisation is therefore particularly beneficial. However, it is crucial to consider features with physical meaning in wastewater system modelling, as their scale or magnitude must be preserved.

In NODEs, the neural network maps input state variables (denoted as X) to their time derivatives (X'). Our proposed normalisation method employs a pair of state normalisation and derivative de-normalisation layers to wrap the deep neural network (see Fig. 2). The normalisation layer scales the input component concentration to a common range of -1 to 1, while the denormalization layer restores the reaction rate (derivatives) from -1 to 1 range to their normal scale. All these normalisation operations are performed inside the approximated ODE function unit, thereby not affecting the solver scales. In this way, the proposed method ensures the underlying physical dynamics to be learnt are not altered.

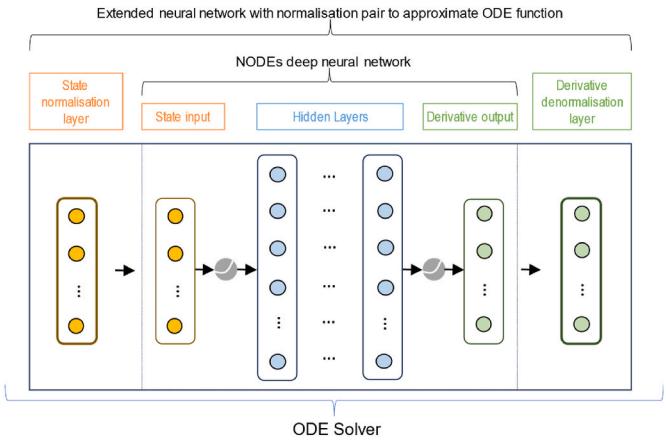

Fig. 2. Illustration of NODEs normalisation pair layout.

The normalisation layer is applied to the neural network input, using standardisation (Z-score normalisation), a common technique in machine learning (Shanker M et al., 1996). This transforms the data to have a zero mean and unit standard deviation.

$$\boldsymbol{X}_i^* = \frac{\boldsymbol{X_i} - \boldsymbol{\mu_X}}{\boldsymbol{\delta_X}} \qquad \text{Equation 3-1}$$

where, μ_X and δ_X denote the mean and standard deviation of the component state data sequence X, respectively.

The de-normalisation layer is then applied to the neural network output to restore the state derivative data to its normal range.

$$\boldsymbol{X}_i' = \boldsymbol{X}_i'^* * \boldsymbol{\delta}_{X'} + \boldsymbol{\mu}_{X'} \qquad \text{Equation 3-2}$$

where, $\mu_{X'}$ and $\delta_{X'}$ denote the mean and standard deviation of the state derivative data sequence X', representing biokinetic rate.

It is crucial to note that the normalisation and de-normalisation layers must be applied in pair within the ODE solver, wrapping the neural network. Unlike conventional machine learning, where normalisation is often performed outside the training process, it cannot be applied outside the solver. This is because the component states and their derivatives carry physical meaning in the dynamics addressed by the solver. Scaling the data outside the solver would skew the relationships of the components, substantially distorting the dynamics to be learned.

From Equations (3)–(1) and Equations (3)–(2), we can see that four sets of mean value and standard deviation are required. The mean and standard deviation for input can be calculated straightforwardly with the monitored time-series component state data. However, the derivative data sequence X' are not available explicitly. To estimate the mean and standard deviation of X', we can employ difference quotients applied to the state data sequence X, such as the single-sided difference method.

$$\boldsymbol{X}' = (\boldsymbol{x_2} - \boldsymbol{x_1},\ \boldsymbol{x_3} - \boldsymbol{x_2}, \ldots, \boldsymbol{x_n} - \boldsymbol{x_{n-1}})/\Delta\boldsymbol{t} \qquad \text{Equation 3-3}$$

Or central difference:

$$\boldsymbol{X}' = (\boldsymbol{x_2} - \boldsymbol{x_1}, (\boldsymbol{x_3} - \boldsymbol{x_2})/\boldsymbol{2}, \ldots, (\boldsymbol{x_n} - \boldsymbol{x_{n-2}})/\boldsymbol{2}, (\boldsymbol{x_n} - \boldsymbol{x_{n-1}}))/\Delta\boldsymbol{t} \qquad \text{Equation 3-4}$$

Experiments show that single-sided and central differences yield similar results (see Supplement Table 1). Although the approximation tends to be more accurate with densely sampled state datasets, a relatively small number of discretisation often suffices.

In practice, the mean and standard deviation of component reaction rates can also be estimated or corrected by experienced operators from other sources, such as routine operation records, site measurements, established mechanistic models, and digital twin outputs.

Researchers have proposed equation scaling methods along with stable gradient calculation, demonstrating their effectiveness on several stiff systems of NODEs (Kim et al., 2021). However, when applied to our wastewater systems, these methods proved insufficient. Our experiments revealed that without our proposed normalisation method, training even the simple ASM1 model using NODEs diverged. Conversely, with our normalisation method, training the more complex ASM2d-N_2O model remained effective, even in the presence of noisy data with up to 0.1 standard deviation (SD) amplitude.

Our method distinguishes itself not only by its comprehensive normalisation of each feature, surpassing the "rough" equation scaling approach based on max-min state values and time spans, but also by its emphasis on the denormalization layer. We hypothesize that derivatives are inherently more erratic and sensitive than state sequences. This aligns well with the common focus on reaction rates (derivatives) in wastewater treatment studies, such as OUR, biomass growth rate, and substrate utilization rate. Normalizing these rates can leverage domain knowledge and improve model performance.

As demonstrated in section 4, normalisation acts as a preconditioner in NODEs, significantly improving training stability and efficiency. The paired normalisation and de-normalisation layers balances loss contribution and stabilise gradients, leading to faster convergence and improved training smoothness and efficiency. Notably, the associated computational overhead is minimal, even practically negligible.

3.2. Collocation method

While the aforementioned normalisation method utilises four parameters pre-obtained from the training dataset, the collocation method employs the entire trajectory of the observational training dataset. It estimates the complete trajectory of the corresponding derivatives using traditional mathematical regression methods, then trains the neural network against the collocated pairs of state input data and estimated state derivatives. This approach eliminates the need for an ODE solver, thereby avoiding the stiffness issue.

In NODEs, the direct approach calls out the ODE solver for derivative calculations at every training step, with derivative computation remaining implicit or hidden to users. Conversely, the collocation method explicitly approximates all the derivatives using kernel functions and interpolation/regression methods prior to the training procedure. Although both methods involve optimisation by gradient descent, training of the neural network in collocation method is simpler as it does not need to go through ODE solver at each step, while direct approach must. Consequently, it can be extremely fast and robust to noise. Fig. 3 illustrates the different strategies of these two methods.

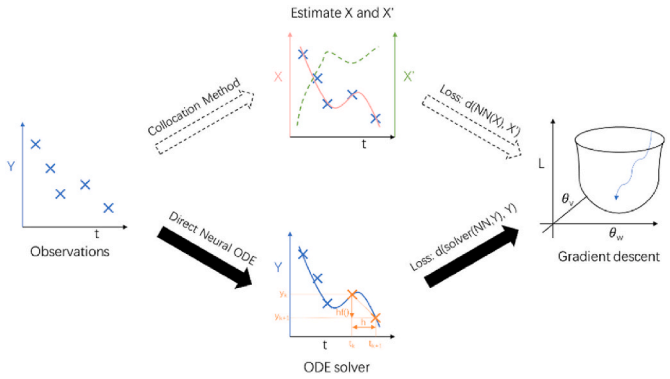

Fig. 3. Different training strategies by direct NODE and collocation method.

The first step of collocation method entails estimating state variables, let's say $X(t)$, and their derivatives, let's say $X'(t)$, from sampled observations (Y_1, Y_2, …, Y_n), at the time points (t_1, t_2, …, t_n), with measurement errors (e_1, e_2, …, e_n), then we have:

$$\boldsymbol{Y}_i = \boldsymbol{X}(\boldsymbol{t}_i) + \boldsymbol{e}_i \quad \boldsymbol{i} = \boldsymbol{1}, \ldots, \boldsymbol{n} \qquad \text{Equation 3-5}$$

To derive $X(t)$ and $X'(t)$ from Y_i, a common practice is to use non-parametric local linear regression for $X(t)$, and local polynomial (often quadratic) regression for $X'(t)$. This approach is based on Talay's formular and criterion of minimizing locally weighted least-squares. Liang and Wu gave a complete deduction process in their paper (Liang and Wu, 2008). We brief the results as follows.

$$\widehat{\boldsymbol{X}}(\boldsymbol{t}) = \boldsymbol{\varepsilon}_1^T \left(\boldsymbol{T}_{1,t}^T \boldsymbol{W}_t \boldsymbol{T}_{1,t}\right)^{-1} \boldsymbol{T}_{1,t}^T \boldsymbol{W}_t \boldsymbol{Y} \qquad \text{Equation 3-6}$$

$$\widehat{\boldsymbol{X}}'(\boldsymbol{t}) = \boldsymbol{\varepsilon}_2^T \left(\boldsymbol{T}_{2,t}^T \boldsymbol{W}_t \boldsymbol{T}_{2,t}\right)^{-1} \boldsymbol{T}_{2,t}^T \boldsymbol{W}_t \boldsymbol{Y} \qquad \text{Equation 3-7}$$

Where,

$$\boldsymbol{\varepsilon}_1 = \begin{bmatrix} \boldsymbol{1} \\ \boldsymbol{0} \end{bmatrix} \quad \boldsymbol{\varepsilon}_2 = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{1} \\ \boldsymbol{0} \end{bmatrix} \qquad \text{Equation 3-8}$$

$$\boldsymbol{T}_{1,t} = \begin{bmatrix} \boldsymbol{1} & \boldsymbol{t}_1 - \boldsymbol{t} \\ \boldsymbol{1} & \boldsymbol{t}_2 - \boldsymbol{t} \\ \vdots & \vdots \\ \boldsymbol{1} & \boldsymbol{t}_n - \boldsymbol{t} \end{bmatrix} \quad \boldsymbol{T}_{2,t} = \begin{bmatrix} \boldsymbol{1} & \boldsymbol{t}_1 - \boldsymbol{t} & (\boldsymbol{t}_1 - \boldsymbol{t})^2 \\ \boldsymbol{1} & \boldsymbol{t}_2 - \boldsymbol{t} & (\boldsymbol{t}_2 - \boldsymbol{t})^2 \\ \vdots & \vdots & \vdots \\ \boldsymbol{1} & \boldsymbol{t}_n - \boldsymbol{t} & (\boldsymbol{t}_n - \boldsymbol{t})^2 \end{bmatrix} \qquad \text{Equation 3-9}$$

$$\boldsymbol{W}_t = \begin{bmatrix} \boldsymbol{K}_h(\boldsymbol{t}_1 - \boldsymbol{t}) & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{K}_h(\boldsymbol{t}_2 - \boldsymbol{t}) & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{K}_h(\boldsymbol{t}_n - \boldsymbol{t}) \end{bmatrix} \qquad \text{Equation 3-10}$$

Where $K(\cdot)$ is a symmetric kernel function. Given an example of Epanechnikov kernel function, we can have:

$$\boldsymbol{K}_h(\boldsymbol{t}_i - \boldsymbol{t}) = \boldsymbol{K}\left(\frac{\boldsymbol{t}_i - \boldsymbol{t}}{\boldsymbol{h}}\right) \Big/ \boldsymbol{h} \qquad \text{Equation 3-11}$$

Where $\boldsymbol{h}$ is a bandwidth:

$$\boldsymbol{h} = \left(\boldsymbol{n}^{-\frac{1}{5}}\right)\left(\boldsymbol{n}^{-\frac{3}{35}}\right)\left((\log(\boldsymbol{n}))^{-\frac{1}{16}}\right) \qquad \text{Equation 3-12}$$

The choice of kernel function depends on the observation data characteristics. For instance, cubic spline is preferred for less noisy or relatively sparse data, while B-spline or Epanechnikov kernel is suitable for noisier datasets.

Due to boundary restriction, derivative estimations at both ends are often inaccurate. This can be mended by excluding data at both ends, to achieve smoother results and reduce excessive changes at the boundaries.

Proper data preprocessing can alleviate subsequent burdens and minimise errors in the training process. To enhance the accuracy of estimated X and X', it is advisable to smooth the noisy observation Y before applying the collocation method. This may involve outlier detection, smoothing techniques and cross validation based on wastewater system knowledge.

The next step involves training the neural network with the estimated $X(t)$ and $X'(t)$ pairs. This process is straightforward, similar to conventional machine learning. The MAE loss function can be

constructed as:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} |NN_\theta(\widehat{X}_i) - \widehat{X}_i| \quad \text{Equation 3-13}$$

The accuracy of the collocation method depends on the characteristics of observational data and estimation methods. While it can be high if the adopted methods and data align well, the results often require further refinement using more elaborate algorithms.

3.3. Incremental strategy

In practice, the results from the collocation method are often "rough" and not sufficiently accurate, although they may be close to the global minima and less prone to local minimum (Rackauckas et al., 2020). To improve the model fidelity to the optimal level, further training using finer or more elaborate methods, such as local minimum algorithms or direct NODEs approach, is expected. Since the collocation method provides a good initial result, subsequent optimisation will experience reduced stiffness and increased effectiveness. In this way, the model fidelity is incrementally improved.

We refer to this practice – applying a coarse method followed by a finer method that builds upon the results of the previous method - as the incremental strategy. The idea is to first provide a rough estimation using the collocation method to narrow the approximated range for the result, then refine it locally by the direct NODE method to achieve higher fidelity.

4. Experiments and results

Data-driven modelling using NODEs primarily involves the repeated solving of initial value problems (IVPs). To demonstrate the feasibility and efficiency of the proposed methods in training NODEs models for wastewater process modelling, we conducted two experimental studies using models from ASM1 to ASM2d-N_2O with increasing complexity but different focuses. We use simpler ASM1 for 100-times efficiency test, saving significant time. ASM2d-N_2O is more complex. Adding noise with level up to 0.1 standard deviation to increase the complexity will suffice to show the effectiveness of our method under more challenging conditions. Both models are used on an IVP in a CSTR.

For comparison purposes, we utilised simulated trajectory data generated from these mathematical models to train the NODEs models. This allows for a direct assessment of the NODEs' performance against well-established wastewater treatment models. All the code and results can be found at https://github.com/Xiangjun-Huang/solving-stiffness-of-NODE.

4.1. ASM1 model

The ASM1 is one of the simplest models for wastewater biological process modelling. Introduced in 1987 and revised over the years, it has been widely for simulating organic matter and nitrogen removal in wastewater treatment. The version we employed consists of 15 components (including two additional components S_{N2} and X_{inorg} for N balance and TSS calculation) and 8 reactions (Mogens et al., 2000a,b). We used default values for stoichiometric and kinetic parameters. For detailed model information, please refer to the cited document.

We generated trajectory data of 1000 points using the ASM1 mathematical model, simulating a CSTR from a defined initial state with a fixed DO level of 2 mg O_2/l over 6 h. The initial values (see Supplement Table 2) were adapted from the steady state of the bioreactor in benchmark simulation model no. 1 (BSM1) (Alex et al., 2018). Fig. 4 illustrate the maximum eigenvalues of the Jacobian over the IVP solution trajectory, showing peak stiffness at approximately 1.7 h.

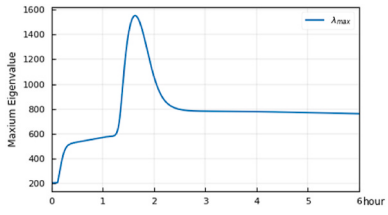

Fig. 4. Maximum eigenvalues of the Jacobian over the IVP solution trajectory.

4.1.1. Training options

We conducted experiments using Python 11 and the *torchdiffeq* package (R. T. Q. Chen, 2018). After testing various solvers from both *torchdiffeq* and the *SciPy* package (Virtanen et al., 2020), we selected *dopri5* for our experiments. Our tests revealed minimal differences between loss functions, with Huber loss performing slightly better. However, MAE was deemed sufficient for process modelling and thus adopted in our experiments.

After testing structures with hidden layers from 2 to 6 and nodes from 20 up to 200, the neural network for the NODEs was constructed as a multilayer perceptron with four layers and 50 nodes in each hidden layer, using activation functions between the layers. While Chen used the Tanh activation function in most of his NODEs examples (R. Chen et al., 2018), we also tested *Gelu* activation function. *Gelu,* a relatively new function, bridges stochastic regularisers with non-linearities, distinguishing it from other activation functions (Hendrycks and Gimpel, 2016). It has demonstrated higher accuracy compared to *ReLU*, and *ELU* (Devlin et al., 2019).

Our experiments showed that *Gelu* outperforms *Relu* and Tanh in NODEs training for wastewater modelling. Fig. 5 compares Gelu and Tanh functions in loss changes for the IVP trajectory training based on ASM1 model, clearly indicating considerably enhanced performance with Gelu. It is worth noting that despite nearly 10^2 orders of magnitude loss decline activated by Gelu without normalisation as shown in Fig. 5, the results remained unsatisfactory.

We trained the NODEs using the ADAM optimiser (Kingma and Ba, 2014) with a varying learning rate, as shown in Fig. 5. The training began with a high learning rate 0.1 to harness speed advantages, then switched to a lower rate 0.001 to refine results in response to loss function changes. Training was conducted for 2000 iterations with a sampling batch size of 512 and 16 steps of the interval calculated each time by the solver. For brevity, three non-reactive components (S_i, X_i, X_{inorg}) and constant DO are not shown in the following results.

4.1.2. Normalisation

The efficacy of the proposed method is assessed by comparing the trained model predictions against ground truth trajectories generated by the mathematical model under identical initial conditions. Fig. 6 illustrates a representative training example without normalisation. The

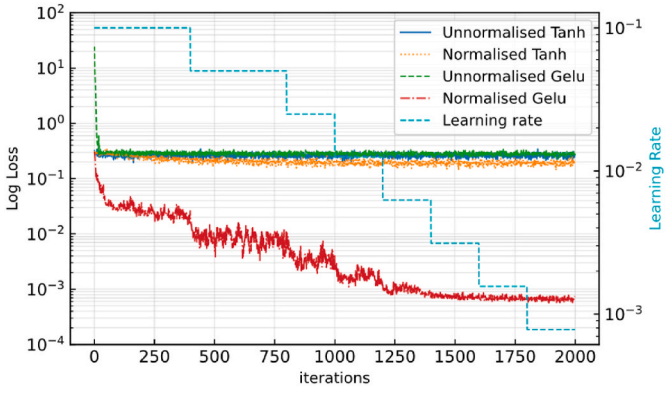

Fig. 5. Comparison in loss between Tanh and Gelu under normalised and unnormalized conditions.

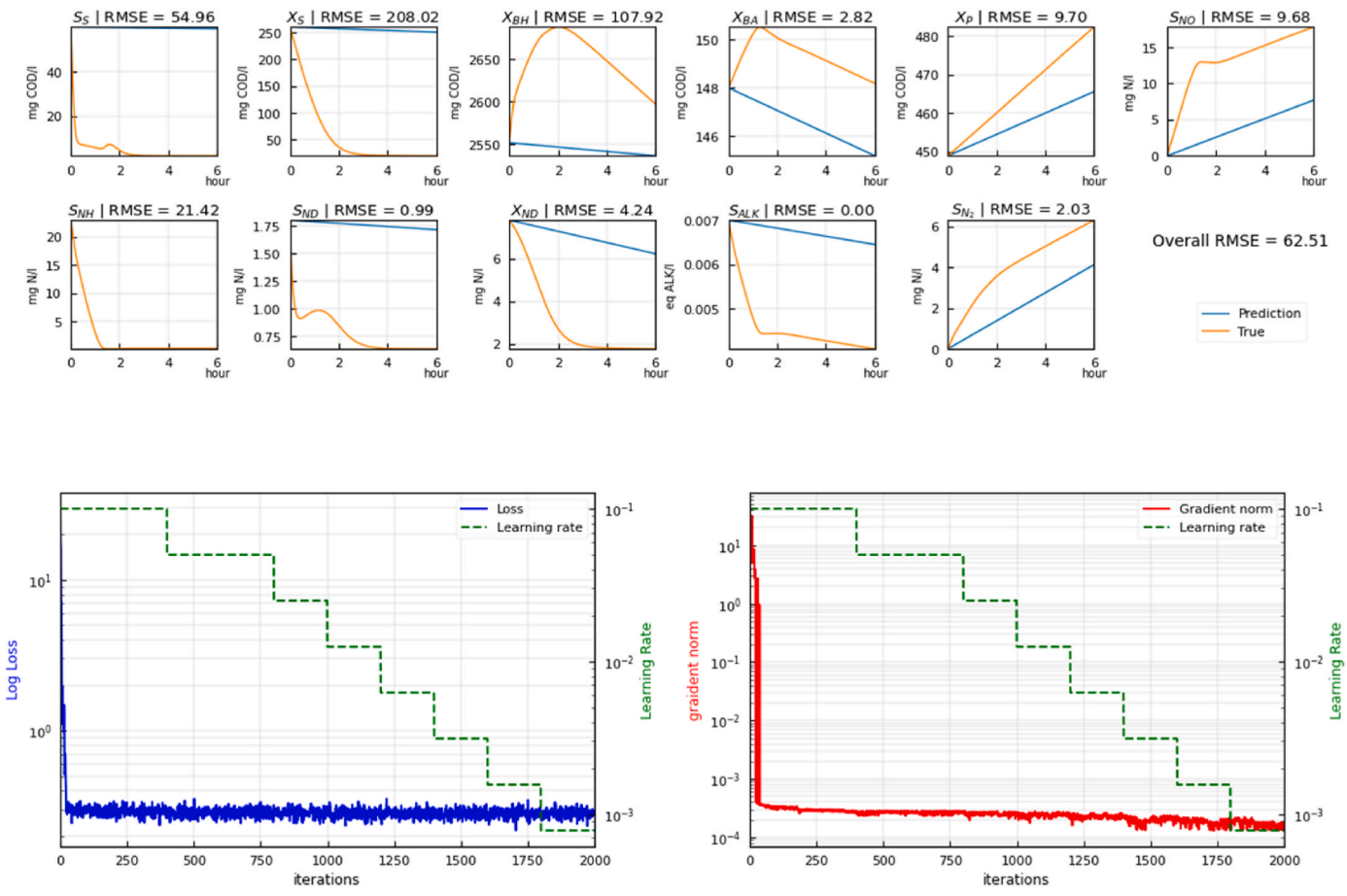

Fig. 6. NODE training without normalisation. Upper: training results; Lower left: loss; Lower right: grad norm.

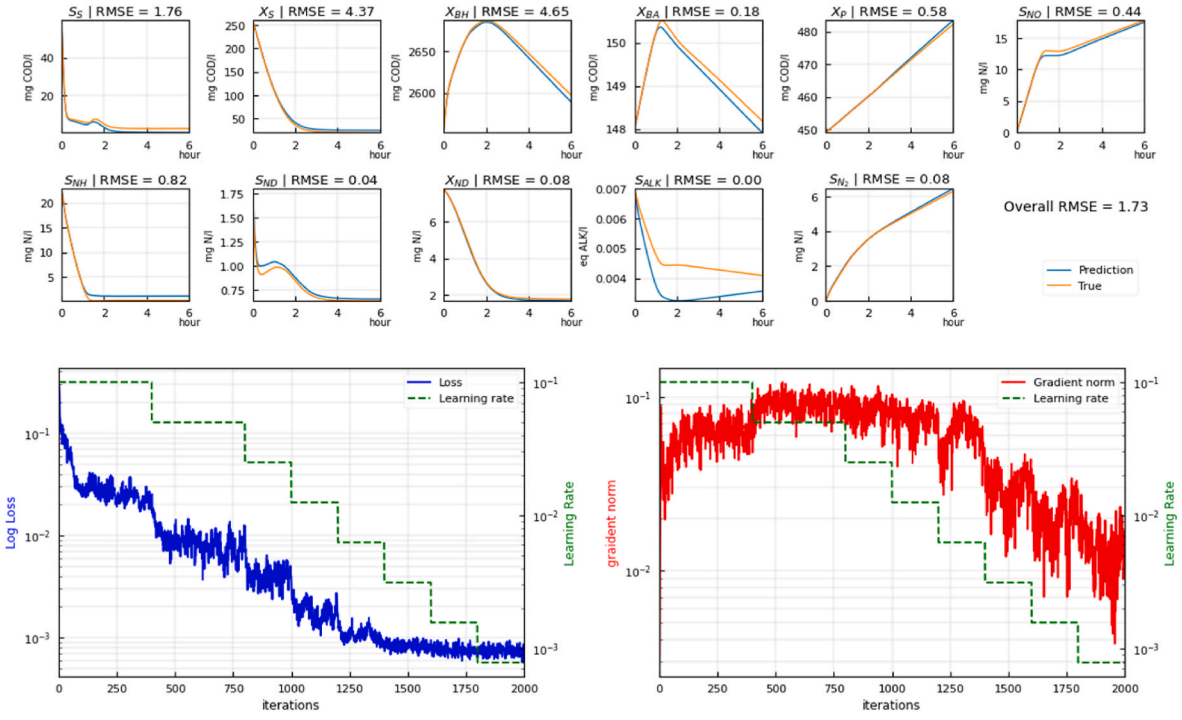

Fig. 7. NODE training with normalisation Upper: training results; Lower left: loss; Lower right: grad norm.

results indicate that the neural network fails to effectively learn from the data, as evidenced by the loss function plateauing after an initial rapid decrease within the first few iterations. The gradient norm exhibits a pathological pattern, remaining at a consistently low level, which reflects the stagnation of the loss throughout the training process. Consequently, the predicted component curves do not align well with the ground truth, resulting in a high overall root mean squared error (RMSE) of 62.51.

The normalisation method is then applied with the estimated mean and standard deviation of the derivatives sequence using single-sided difference quotient. The neural network was wrapped by the normalisation and de-normalisation pair with the estimated parameters. As illustrated in Fig. 7, this normalisation technique yields significant improvements. The predicted trajectory curves now closely align with the ground truth, resulting in a substantially reduced overall RMSE of 1.73. Moreover, the loss function exhibits a gradual decrease throughout the training iterations, while the gradient norm demonstrates stable behaviour.

4.1.3. Incremental strategy

The incremental strategy was implemented by first training the normalised model using the collocation method, followed by direct NODE training. Figs. 8 and 9 illustrate the results of the collocation method utilising the Epanechnikov kernel function, comparing the ground truth with the collocated trajectory and its derivatives. The smoothed trajectory demonstrated a close fit to the ground truth, as evidenced by a low RMSE of 6.39. However, the derivatives exhibited significant disparity, with a high RMSE of 325.5, indicating challenges in accurate estimation.

Fig. 10 illustrates a representative result from the collocation training stage. As the number of iterations increases, the loss consistently decreases, while the gradient norm remains stable. Despite this apparent progress, the RMSE remains high at 31.44 after 2000 iterations. This persistent discrepancy can be attributed to substantial errors in derivative estimation using the collocation method.

Fig. 11 displays the results of the subsequent direct NODEs training stage. Throughout this stage, the loss exhibits a generally consistent decrease, while the gradient norm maintains stability at a relatively low level compared to collocation training phase.

Due to the stochastic nature of neural networks, results may vary slightly across different training runs. To assess efficiency, we conducted 100 trials of each training method, each comprising 2000 iterations under identical conditions. The tests were performed on a computer with an Intel® Core™ i7 CPU (2.8 GHz), 16 MB RAM, without a dedicated GPU. The test program was executed within the VSCode IDE on a Windows 10 64-bit operating system.

Fig. 12 presents the results of this efficiency test (detailed data available on the project's GitHub repository). The analysis reveals that the incremental training strategy, compared to the NODE-only training, consumes 24.3% less time on average and yields a 24.7% lower RMSE. Notably, when collocation training precedes NODE training, the resulting RMSE demonstrates a smaller standard deviation (1.2) compared to the method without collocation integration (1.4), suggesting enhanced training stability.

4.2. ASM2d-N_2O model

The growing concern over climate change has intensified focus on greenhouse gas emissions from wastewater treatment plants, particularly nitrous oxide (N_2O). Recent research (Ye et al., 2022) identifies four potential pathways for N_2O generation during biological nitrogen removal in wastewater treatment: (i) hydroxylamine oxidation, (ii) nitrifier denitrification, (iii) heterotrophic denitrification, and (iv) abiotic reactions. While these pathways may coexist at varying ratios, the significance of the fourth pathway remains under debate.

This experiment employs the ASM2d-N_2O model, an extension of the ASM2d model that incorporates N_2O emissions. This comprehensive model describes 40 reactions involving 24 fractionated components, encompassing the biological removal of carbon, nitrogen, and phosphorus, including the three major N_2O emission pathways. We utilised the stoichiometric and kinetic parameters reported in the original paper (Massara et al., 2018).

The model's complexity arises from its inclusion of greenhouse gas emissions like N_2O and other transient, low-concentration byproducts such as nitric oxide (NO). Additionally, it represents intricate biochemical reactions. These factors collectively contribute to a significant degree of stiffness in the system due to the vast differences in scales and magnitudes between various components. This characteristic makes the ASM2d-N_2O model a suitable test case for evaluating the proposed solutions.

We employed MATLAB as the programming language for this experiment. The data originated from an ASM2d-N_2O model simulation of a CSTR for 6 h under constant dissolved oxygen control at 2 mg/L. The simulation began from an initial condition detailed in Supplementary Table 3. The generated trajectory data for the IVP was discretised into 1000 points.

To simulate the real situation and evaluate how the proposed methods behave on noisy monitoring data, we prepared three datasets for our experiment. One set contained the original, noise-free data. The remaining two sets were corrupted with varying levels of white noise, each with different SD amplitudes, 0.05, and 0.1. To mitigate the effect of noise on training, we applied a Gaussian filter with a window size of 50 for smoothing before training the models.

4.2.1. Training options

MATLAB offers a comprehensive suite of ODE solvers, yet currently

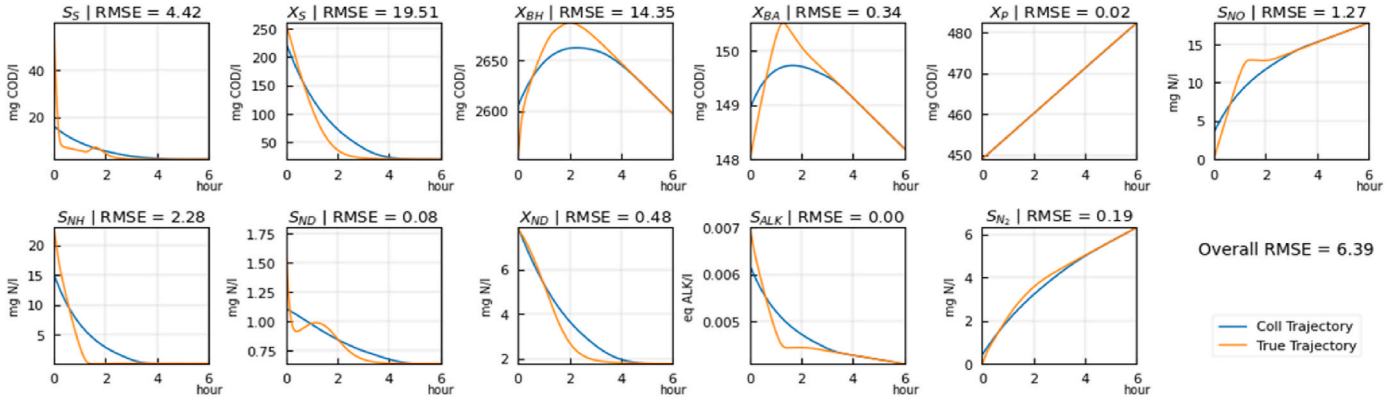

Fig. 8. Comparison of collocated data and ground truth of the trajectory.

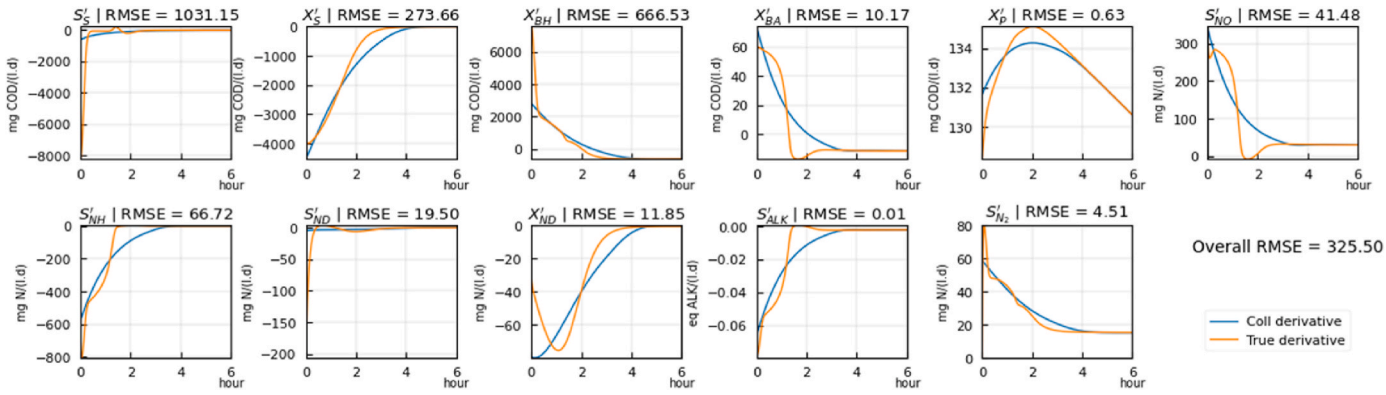

Fig. 9. Comparison of collocated derivative and ground truth derivative.

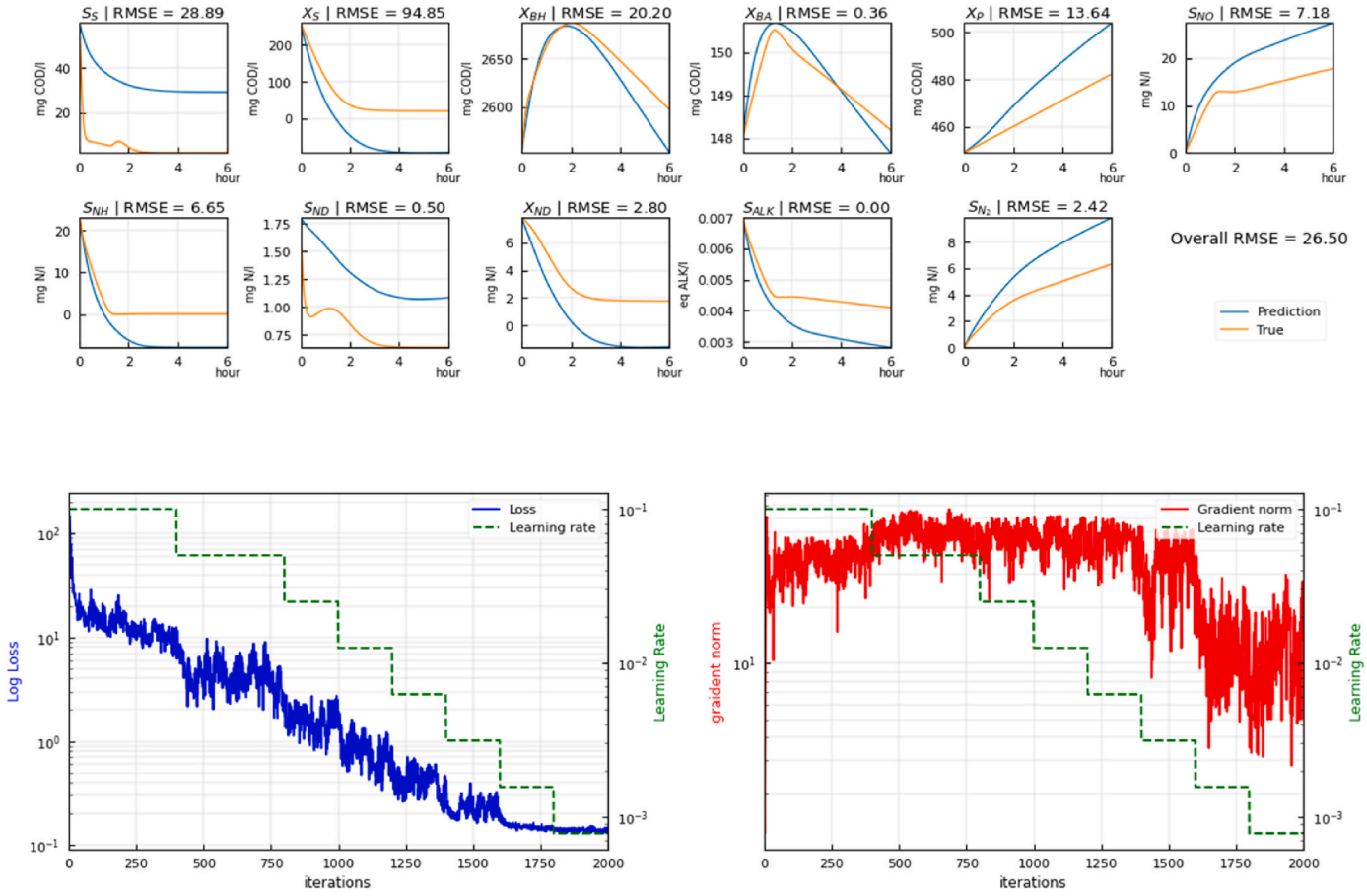

Fig. 10. Collocation training stage by incremental strategy. Upper: training results; Lower left: loss; Lower right: grad norm.

provides only one solver specifically designed for NODEs problems. This limitation reflects the relative novelty of the NODE approach, underscoring the need for further development in this area. Despite the *dlode45* solver being documented for non-stiff problems (MATLAB, 2024), we employed it in our specific case for experimentation.

For the NODE model, we constructed a MLP architecture with four hidden layers, each containing 50 nodes. The Gelu activation function was employed between layers for improved performance. Xavier Glorot initialization was utilised for the weight matrices within the neural network to address vanishing/exploding gradients. The MAE served as the loss function throughout our experiments, aiming to minimise the absolute difference between predicted and ground truth values.

The NODEs were trained using the *Adamupdate* optimiser (Kingma and Ba, 2014) with a gradient decay factor of 0.9, a squared gradient decay factor of 0.999, and a global learning rate of 0.01. We implemented custom loops to manage the training process. The collocation training stage utilised 3000 iterations, followed by 1000 iterations for direct NODE training. A batch size of 200 and time steps of 800 were employed during training.

4.2.2. *Normalisation*

We initially evaluated the performance of direct NODE training without normalisation. To validate the results, we compared the predictions from the trained model with solutions generated by the mathematical model for the same IVP. Fig. 13 presents a typical example of training with data containing 0.05 SD noise and without normalisation.

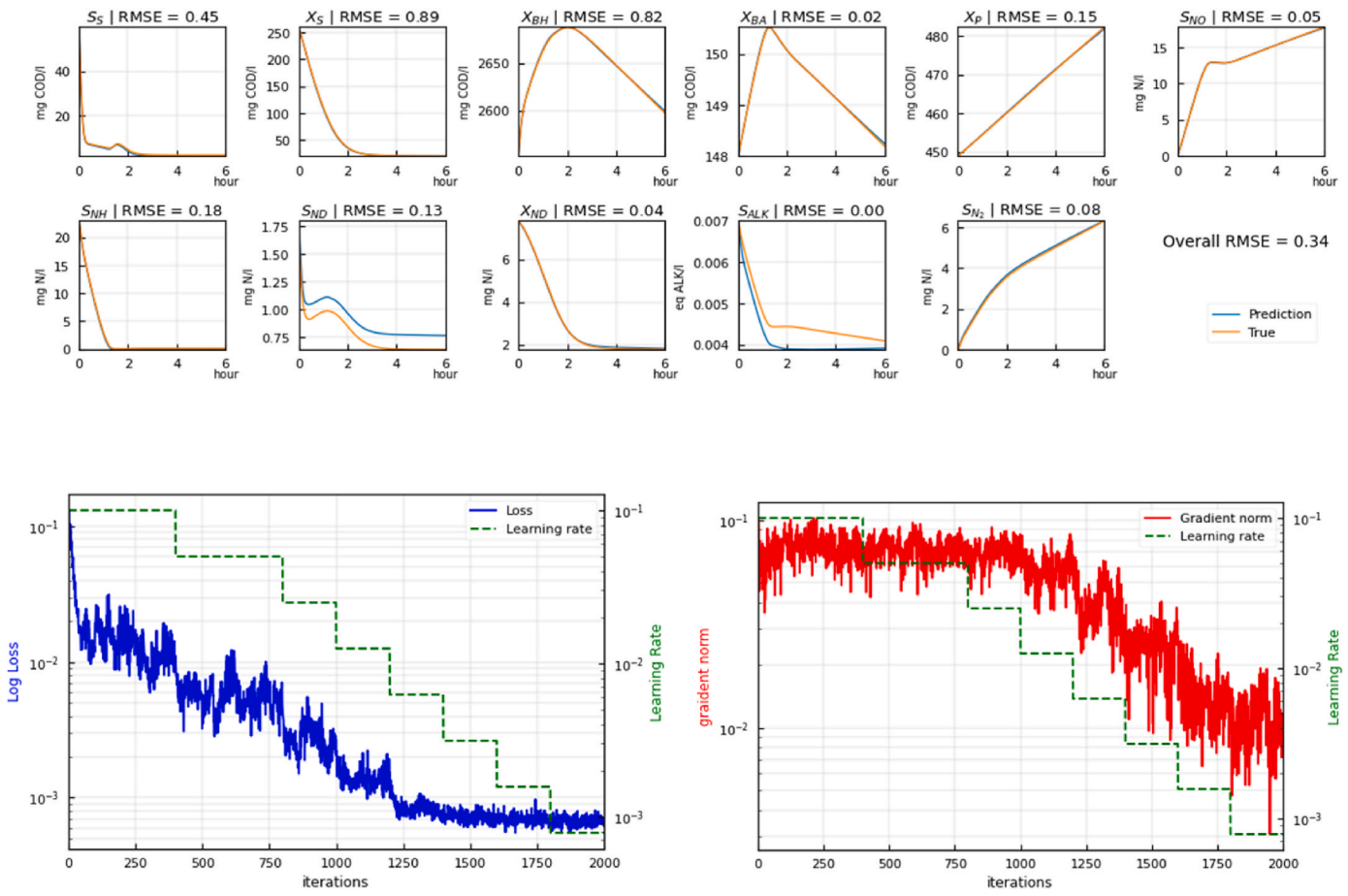

Fig. 11. Direct NODE training stage by incremental strategy. Upper: training results; Lower left: loss; Lower right: grad norm.

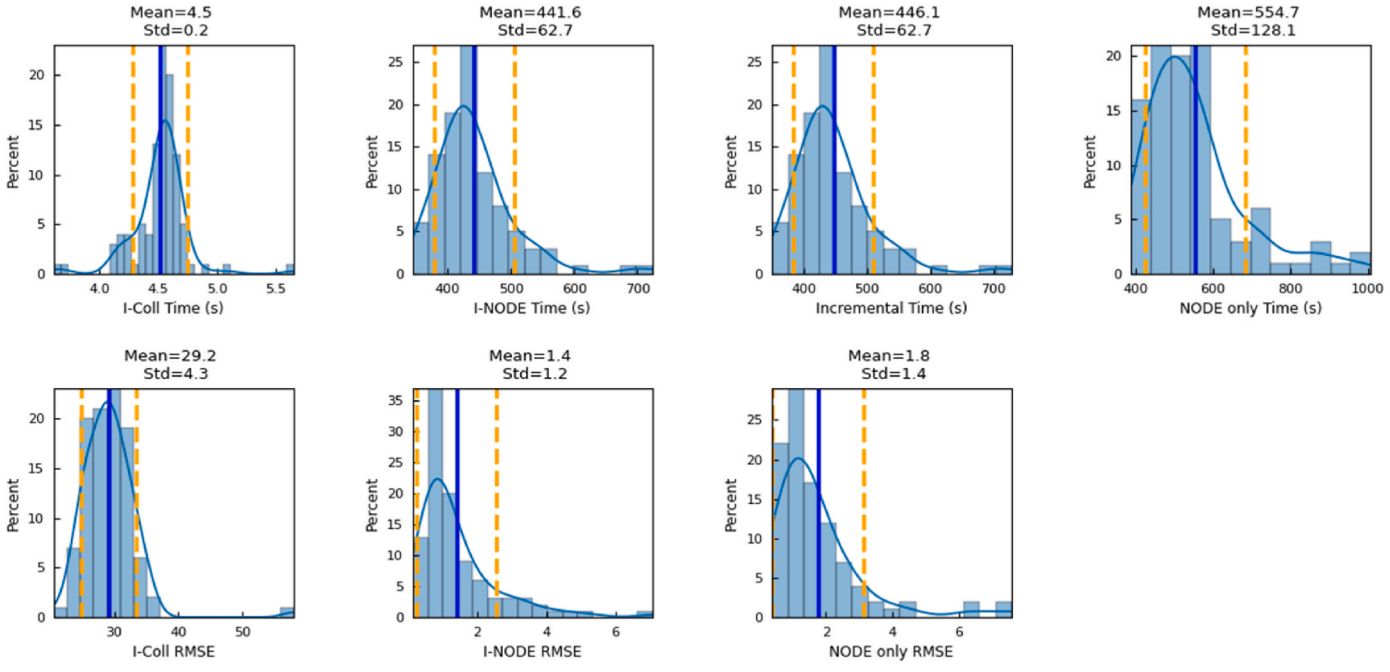

Fig. 12. Distribution of time consumption and RMSE for 100 times running by different training methods. (I-Coll: Incremental collocation training part, I-NODE: Incremental NODE training part).

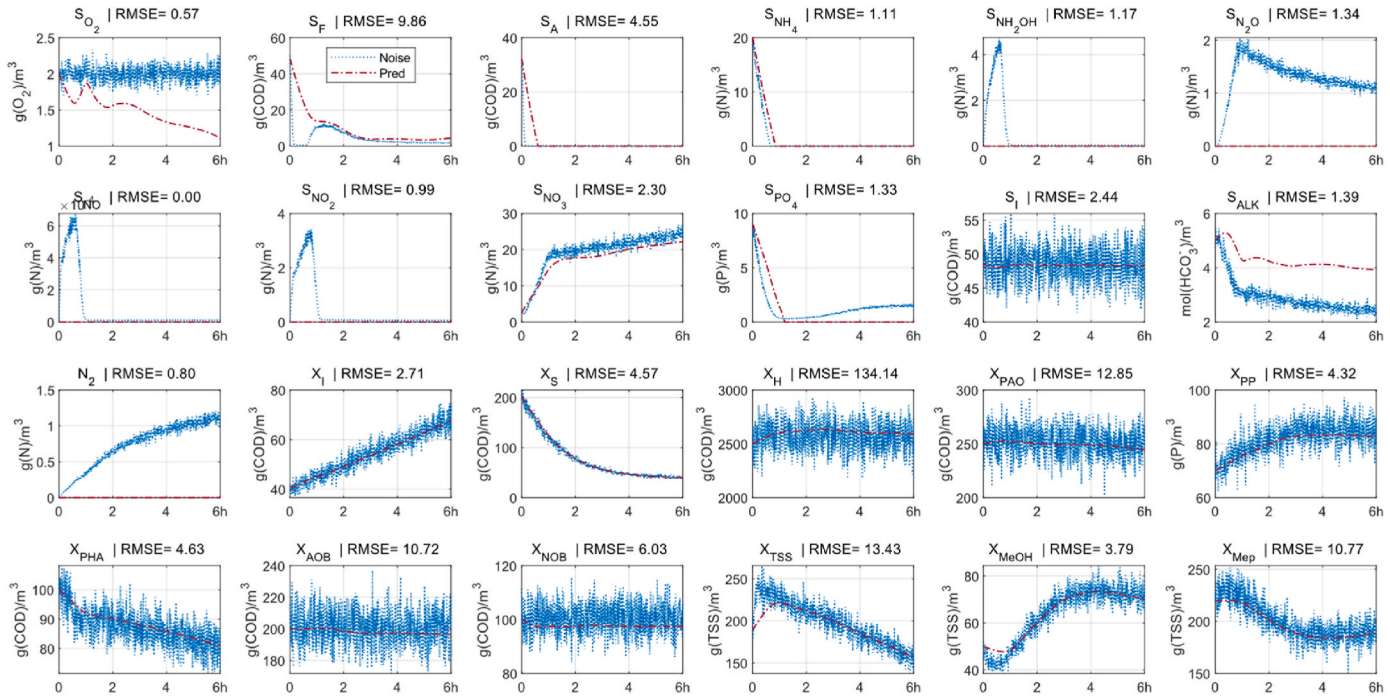

Fig. 13. Result of training without normalisation with data containing 0.05 SD noise.

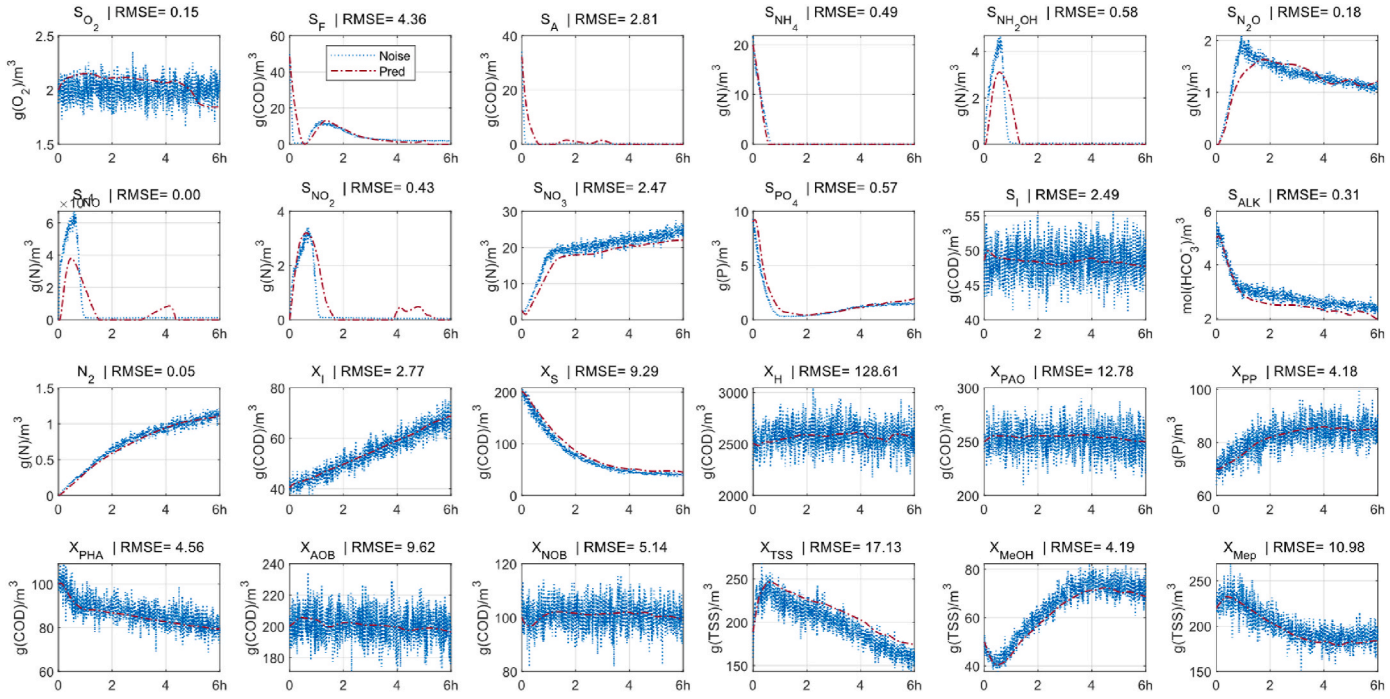

Fig. 14. Result of training with normalisation with data containing 0.05 SD noise.

While the model successfully describes the trajectory of most components, it struggles to capture the trajectories of low-valued scaled components, such as S_{NO}, which have magnitudes on the order of 10^{-4} mg/L.

Following this observation, we implemented the normalisation method. Fig. 14 illustrates the results of direct NODE training with normalisation-denormalization pair. For the normalisation layer, the mean and standard deviation were directly calculated from the smoothed component state data sequence. Differently, the denormalization layer employed the differential quotient method to estimate the mean and standard deviation of the derivative data sequence from the smoothed dataset. Although some turbulences are still evident, the predicted trajectory progressively improves in smoothness with increasing training iterations. Importantly, the model now captures the trajectories of low-valued scaled components like S_{NO}, leading to more satisfactory overall results.

Figure compares training losses between models with and without normalisation, both scaled to the same range.

4.2.3. *Incremental strategy*

We evaluated the performance of incremental training strategies

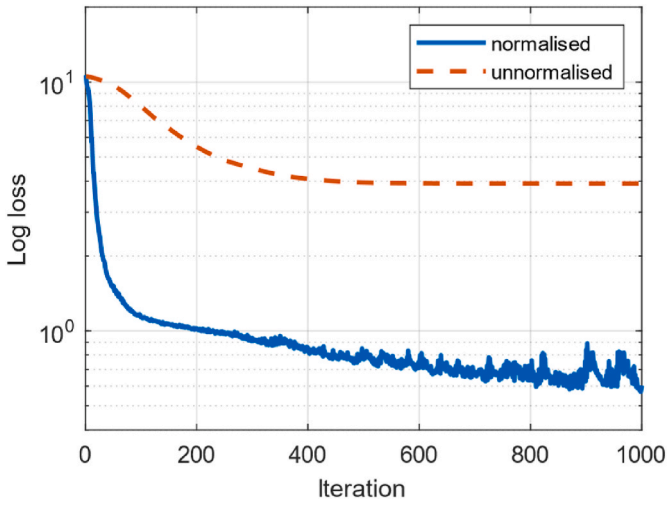

Fig. 15. Comparison in training loss between models with and without normalisation.

using data corrupted with normally distributed noise at three SD levels 0.05, and 0.1. The training process consisted of two stages: first, the model was trained using the collocation method, followed by further training with the direct NODE method. The neural network employed Z-score normalisation at the input layer and de-normalisation at the output layer. An Epanechnikov kernel function was chosen for data collocation.

The collocation training exhibited a fast and stable convergence process, requiring 3000 iterations. Subsequent direct NODE training also demonstrated good convergence, achieving satisfactory results with only 300 iterations, although 1000 iterations were used for further refinement.

As illustrated in Figs. 16, 17 and 17, the predictions were compared to the observations after training with the same number of iterations on data with noise levels of 0.05, and 0.1 SD, respectively. The results clearly demonstrate that the accuracy deteriorates as the noise level increases. Lower noise levels yield better results, as evidenced by the RMSE values of 27.28 and 54.54 for noise levels of 0.05, and 0.1 SD, respectively. This reinforces the importance of minimizing noise through data smoothing techniques before training to avoid training instability and ensure optimal performance.

Our comparison demonstrates that the incremental approach of collocation training followed by the direct NODE method improved the result and enhanced the training stability and speed. Regardless of the chosen approach, normalisation remains crucial for successful training of stiff neural ODE models for wastewater process modelling.

5. Discussion

The training of NODEs for data-driven wastewater processes modelling is particularly challenging due to the inherent stiffness of the underlying dynamics. This stiffness, exacerbated by amplification of numerical errors and enlarged scale separations during optimisation, can lead to unstable training and inaccurate prediction.

To address these challenges, we propose a novel normalisation pair method that significantly enhances the stability and efficiency of NODE training. By normalizing the state input and de-normalizing state derivative output within the ODE solver, we effectively balance the optimisation across disparate scales, leading to smoother gradient flows and reduced risk of gradient vanishing or explosion. This enables the training of deeper and more complex NODEs models. Additionally, the proposed method is simple to implement and can be readily applied to various stiff NODE systems with minimal computational overhead.

To further enhance the training process, we explored an incremental training strategy. By training a NODE model initially using a collocation method and then refining it with a direct NODE approach, we were able to bypass the initial stiffness hurdle and achieve improved efficiency and accuracy. This combined strategy highlights the value of tailoring training methods to specific problem stages.

However, it is important to note that increasing noise levels can rapidly deteriorate the accuracy of NODE models due to heightened derivative sensitivity. To mitigate this issue, pre-emptive data smoothing is highly recommended to reduce noise amplification within the neural network and promote a more stable training process.

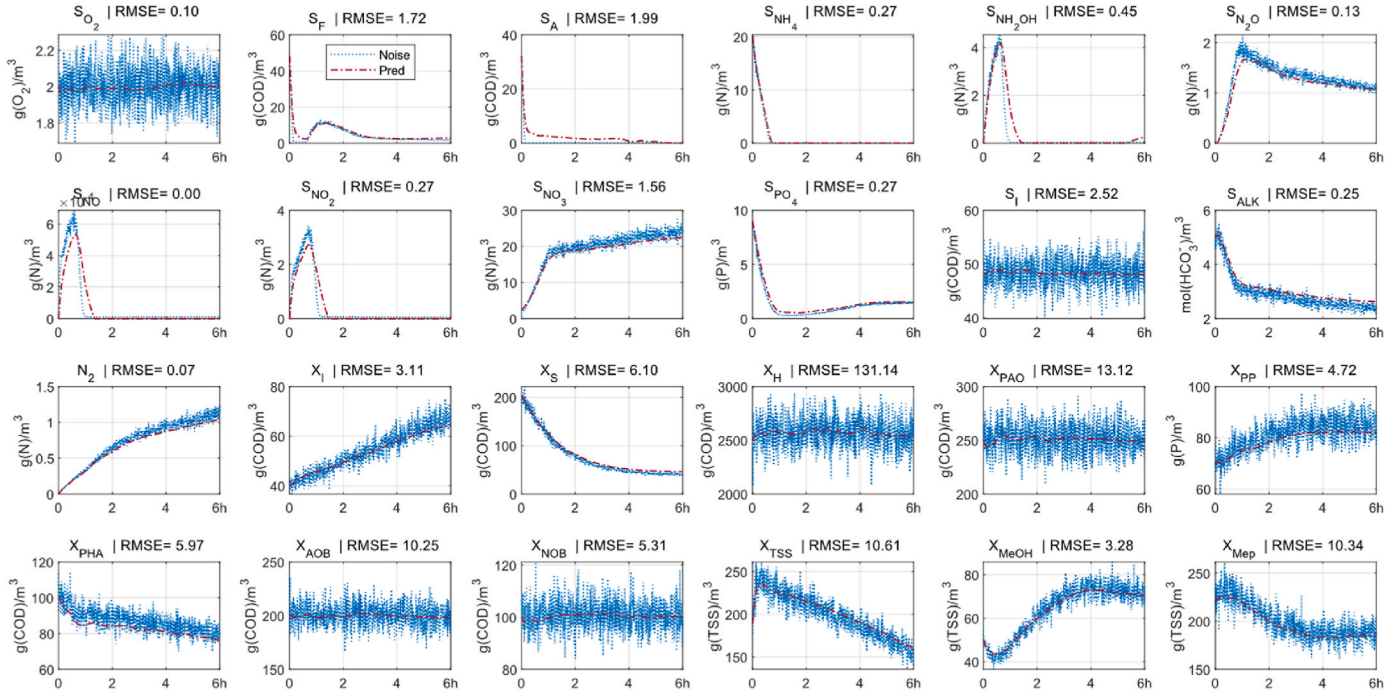

Fig. 16. Validation of model trained by incremental strategy with data containing 0.05 SD noise.

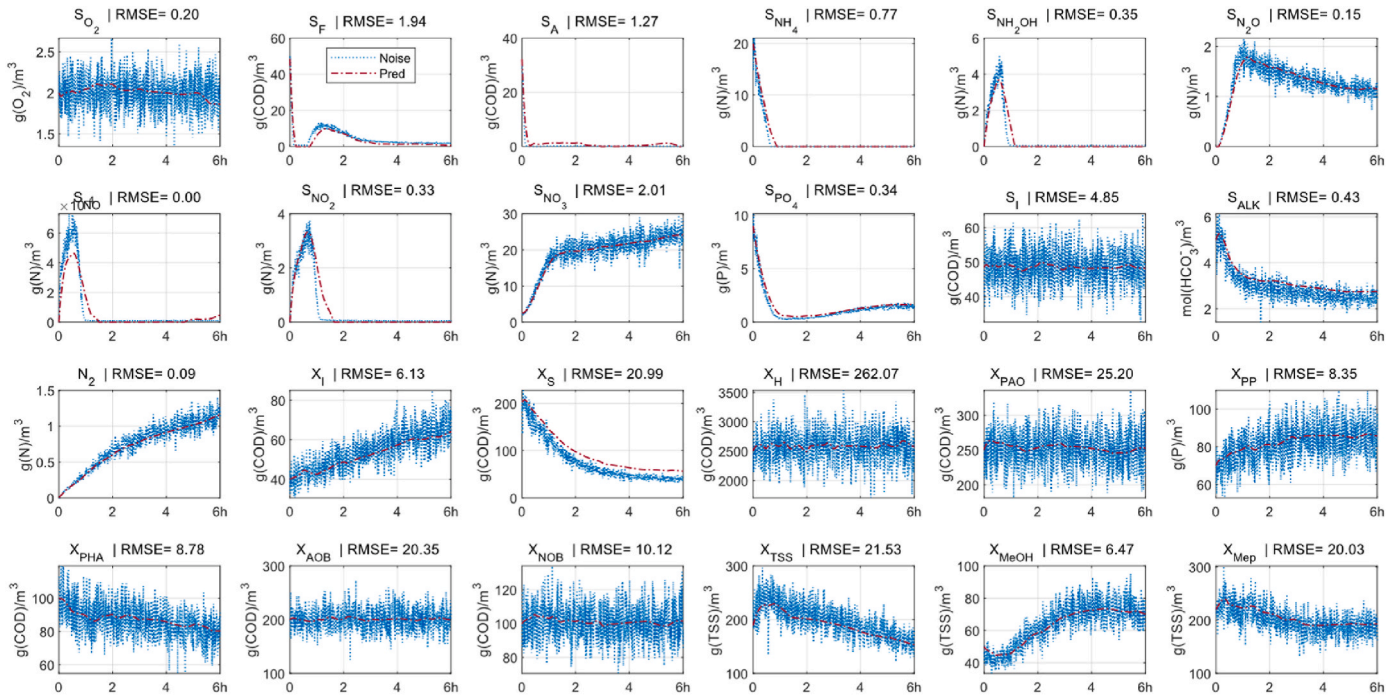

Fig. 17. Validation of model trained by incremental strategy with data containing 0.1 SD noise.

We hypothesize that this normalisation technique may not be optimal for datasets exhibiting minor scale separations, limited feature variability, or high feature correlation. Further investigation can be made to determine an appropriate normalisation parameter for component that simultaneously undergo both reactions of generation from some components and consumption by others. Additionally, the potential of collocation methods can be explored with more sophisticated techniques to enhance accuracy.

6. Conclusion

In conclusion, the proposed normalisation method and incremental training strategy provide a robust and efficient framework for training stiff NODEs in data-driven wastewater modelling. These techniques offer significant advantages in mitigating the numerical stability issues and accelerating the training process. We anticipate that this innovative approach will be validated through further practical applications and pave the way for the widespread adoption of NODEs in various fields, leading to more efficient data-driven management strategies. Beyond wastewater modelling, the proposed normalisation method has the potential to be applied to other fields characterised by stiff ODE systems, noisy data, and complex nonlinear dynamics. Specific applications may include reaction kinetics modelling in chemical engineering, physiological system simulations in biomedical processes, and climate modelling or pollutant dispersion predications in environmental science.

CRediT authorship contribution statement

Xiangjun Huang: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation. **Kyriakos Kandris:** Writing – review & editing, Supervision, Project administration, Investigation. **Evina Katsou:** Writing – review & editing, Supervision, Resources, Project administration, Funding acquisition, Conceptualization.

Declaration of competing interest

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

The work was supported by the CRONUS project (grant agreement ID: 101084405) funded by the European Union under Horizon Europe Research and Innovation Action scheme.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.jenvman.2024.123870.

Data availability

I have shared the link of my data/code in the manuscript uploaded

References

Alex, J., Benedetti, L., Copp, J., Gernaey, K.V., Jeppsson, U., Nopens, I., Pons, M.N., Steyer, J.P., Vanrolleghem, P., 2018. Benchmark simulation model, 1 (BSM1).

Amari, S. ichi, 1993. Backpropagation and stochastic gradient descent method. Neurocomputing 5 (4–5), 185–196. https://doi.org/10.1016/0925-2312(93)90006-O.

Bhanja, S., Das, A., 2018. Impact of Data Normalization on Deep Neural Network for Time Series Forecasting.

Böttcher, L., Asikis, T., 2022. Near-optimal control of dynamical systems with neural ordinary differential equations. Mach. Learn.: Sci. Technol. 3 (4). https://doi.org/10.1088/2632-2153/ac92c3.

Bradley, W., Boukouvala, F., 2021. Two-stage approach to parameter estimation of differential equations using neural ODEs. Ind. Eng. Chem. Res. 60 (45), 16330–16344. https://doi.org/10.1021/acs.iecr.1c00552.

Cabello-Solorzano, K., Ortigosa de Araujo, I., Peña, M., Correia, L., J Tallón-Ballesteros, A., 2023. The Impact of data normalization on the accuracy of machine learning algorithms: a comparative analysis. In: García Bringas, P., Pérez García, H., Martínez de Pisón, F.J., Martínez Álvarez, F., Troncoso Lora, A., Herrero, Á., Calvo Rolle, J.L., Quintián, H., Corchado, E. (Eds.), 18th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2023). Springer, Nature Switzerland, pp. 344–353.

Chen, R., Rubanova, Y., Bettencourt, J., Duvenaud, D., 2018. Neural Ordinary Differential Equations.
Chen, R.T.Q., 2018. *torchdiffeq*. Github. https://github.com/rtqichen/torchdiffeq.
Cuomo, S., Di Cola, V.S., Giampaolo, F., Rozza, G., Raissi, M., Piccialli, F., 2022. Scientific machine learning through physics–informed neural networks: where we are and what's next. J. Sci. Comput. 92 (3). https://doi.org/10.1007/s10915-022-01939-z.
Devlin, J., Chang, M.-W., Lee, K., Toutanova, K., 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. North American Chapter of the Association for Computational Linguistics. https://api.semanticscholar.org/CorpusID:52967399.
Du, Q., Gu, Y., Yang, H., Zhou, C., 2022. The discovery of dynamics via linear multistep methods and deep learning: error estimation. SIAM J. Numer. Anal. 60 (4), 2014–2045.
Dupont, E., Doucet, A., Teh, Y.W., 2019. Augmented neural ODEs. http://arxiv.org/abs/1904.01681.
Elbrächter, D., Perekrestenko, D., Grohs, P., Bölcskei, H., 2019. Deep neural network approximation theory. http://arxiv.org/abs/1901.02220.
Fronk, C., Petzold, L., 2024. Training stiff neural ordinary differential equations with implicit single-step methods. ArXiv. https://arxiv.org/abs/2410.05592.
Gear, C.W., 1981. Numerical solution of ordinary differential equations: is there anything left to do? SIAM Rev. 23 (1), 10–24. https://doi.org/10.1137/1023002.
Gusmão, G.S., Retnanto, A.P., Cunha, S. C. da, Medford, A.J., 2022. Kinetics-informed neural networks. Catal. Today. https://doi.org/10.1016/j.cattod.2022.04.002.
Hairer, E., Wanner, G., 1996a. Solving Ordinary Differential Equations II, second ed.
Hairer, E., Wanner, G., 1996b. Solving Ordinary Differential Equations II - Stiff and Differential Algebraic Problems, vol. 375. Springer.
Hendrycks, D., Gimpel, K., 2016. Gaussian error linear units (GELUs). http://arxiv.org/abs/1606.08415.
Huang, L., Qin, J., Zhou, Y., Zhu, F., Liu, L., Shao, L., 2023. Normalization techniques in training dnns: Methodology, analysis and application. IEEE Trans. Pattern Anal. Mach. Intell. 45 (8), 10173–10196.
Karniadakis, G.E., Kevrekidis, I.G., Lu, L., Perdikaris, P., Wang, S., Yang, L., 2021. Physics-informed machine learning. Nat. Rev. Phy. 3 (6), 422–440. https://doi.org/10.1038/s42254-021-00314-5.
Kidger, P., Morrill, J., Foster, J., Lyons, T., 2020. Neural controlled differential equations for irregular time series. http://arxiv.org/abs/2005.08926.
Kim, S., Ji, W., Deng, S., Ma, Y., Rackauckas, C., 2021. Stiff neural ordinary differential equations. https://doi.org/10.1063/5.0060697.
Kingma, D.P., Ba, J., 2014. Adam: A Method for Stochastic Optimization. http://arxiv.org/abs/1412.6980.
Kong, X., Yamashita, K., Foggo, B., Yu, N., 2022. Dynamic Parameter Estimation with Physics-based Neural Ordinary Differential Equations.
Kushnir, D., Rokhlin, V., 2012. A highly accurate solver for stiff ordinary differential equations. SIAM J. Sci. Comput. 34 (3), A1296–A1315. https://doi.org/10.1137/100810216.
Liang, H., Wu, H., 2008. Parameter estimation for differential equation models using a framework of measurement error in regression models. J. Am. Stat. Assoc. 103 (484), 1570–1583. https://doi.org/10.1198/016214508000000797.
Linda, P., 1982. Differential/Algebraic Equations are not ODE's. SIAM J. Sci. Stat. Comput.
Massara, T.M., Solís, B., Guisasola, A., Katsou, E., Baeza, J.A., 2018. Development of an ASM2d-N2O model to describe nitrous oxide emissions in municipal WWTPs under dynamic conditions. Chem. Eng. J. 335, 185–196. https://doi.org/10.1016/j.cej.2017.10.119.
MATLAB, 2024. Documentation: dlode45 function. https://uk.mathworks.com/help/deeplearning/ref/dlarray.dlode45.html.
Mogens, H., Willi, G., Takashi, M., Mark, van L., 2000a. Activated Sludge Models ASM1, ASM2, ASM2d, and ASM3. IWA Publishing.
Mogens, H., Willi, G., Takashi, M., van, Loosdrecht Mark, 2000b. Activated Sludge Models ASM1,ASM2, ASM2d, and ASM3. IWA Publishing.
Núñez, M., Barreiro, N.L., Barrio, R.A., Rackauckas, C., 2023. Forecasting virus outbreaks with social media data via neural ordinary differential equations. Sci. Rep. 13 (1). https://doi.org/10.1038/s41598-023-37118-9.
Pinkus, A., 1999. Approximation theory of the MLP model in neural networks. Acta Numer. 8, 143–195. https://doi.org/10.1017/S0962492900002919.
Quaghebeur, W., Torfs, E., Baets, B. De, Nopens, I., 2022. Hybrid differential equations: integrating mechanistic and data-driven techniques for modelling of water systems. Water Res. 213. https://doi.org/10.1016/j.watres.2022.118166.
Rackauckas, C., Ma, Y., Martensen, J., Warner, C., Zubov, K., Supekar, R., Skinner, D., Ramadhan, A., Edelman, A., 2020. Universal differential equations for scientific machine learning. ArXiv Preprint ArXiv:2001.04385.
Sandoval, I.O., Petsagkourakis, P., del Rio-Chanona, E.A., 2022. Neural ODEs as feedback policies for nonlinear optimal control. http://arxiv.org/abs/2210.11245.
Shanker, M., Hu, M.Y., Hung, M.S., 1996. Effect of data standardization on neural network training. Int. J. Mgmt Sci 24 (Issue 4).
Tchobanoglous, G., Stensel, H.D., Tsuchihashi, R., Burton, F., 2014. *Wastewater Engineering: Treatment and Resource Recovery* (5th Intern). McGraw-Hill Education.
The MathWorks Inc, 2023. Dynamical system modeling using neural ODE. https://uk.mathworks.com/help/deeplearning/ug/dynamical-system-modeling-using-neural-ode.html.
Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., SciPy 1.0 Contributors, 2020. SciPy 1.0: Fundamental algorithms for scientific computing in Python. Nat. Methods 17, 261–272. https://doi.org/10.1038/s41592-019-0686-2.
Xue, Y., Nasim, M., Zhang, M., Fan, C., Zhang, X., El-Azab, A., 2021. Physics Knowledge Discovery via Neural Differential Equation Embedding.
Ye, L., Porro, J., Nopens, I., 2022. Quantification and Modelling of Fugitive Greenhouse Gas Emissions from Urban Water Systems, first ed. IWA Publishing.
Zhu, A., Jin, P., Zhu, B., Tang, Y., 2022. On numerical integration in neural ordinary differential equations. In: International Conference on Machine Learning, pp. 27527–27547.
Zhu, A., Wu, S., Tang, Y., 2024. Error analysis based on inverse modified differential equations for discovery of dynamics using linear multistep methods and deep learning. SIAM J. Numer. Anal. 62 (5), 2087–2120.