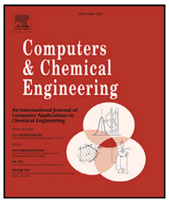
Computers and Chemical Engineering 178 (2023) 108367

Contents lists available at ScienceDirect

Computers and Chemical Engineering

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



Model predictive control of nonlinear processes using neural ordinary differential equation models

Junwei Luo [a], Fahim Abdullah [a], Panagiotis D. Christofides [a,b,*]

[a] Department of Chemical and Biomolecular Engineering, University of California, Los Angeles, CA, 90095-1592, USA
[b] Department of Electrical and Computer Engineering, University of California, Los Angeles, CA 90095-1592, USA

ARTICLE INFO

Keywords:
Neural ordinary differential equations
Model predictive control
Subsampling
Noisy data
Chemical processes

ABSTRACT

Neural Ordinary Differential Equation (NODE) is a recently proposed family of deep learning models that can perform a continuous approximation of a linear/nonlinear dynamic system using time-series data by integrating the neural network model with classical ordinary differential equation solvers. Modeling nonlinear dynamic processes using data has historically been a critical challenge in the field of chemical engineering research. With the development of computer science and neural network technology, recurrent neural networks (RNN) have become a popular black-box approach to accomplish this task and have been utilized to design model predictive control (MPC) systems. However, as a discrete-time approximation model, RNN requires strictly uniform step sequence data to operate, which makes it less robust to an irregular sampling scenario, such as missing data points during operation due to sensor failure or other types of random errors. This paper aims to develop a NODE model and construct an MPC based on this novel continuous-time neural network model. In this context, closed-loop stability is established and robustness to noise is addressed using a variety of data analysis techniques. An example of a chemical process is utilized to evaluate the performance of NODE-based MPC. Furthermore, the performance of the NODE-based MPC under Gaussian and non-Gaussian noise is investigated, and the subsampling method is found to be effective in building models for MPC that are suitable for handling the presence of non-Gaussian noise in the data.

1. Introduction

Model predictive control is an advanced optimization-based control strategy that has been widely used in various industries and applications, ranging from chemical process control (Rohani et al., 1999; Benattia et al., 2016) to autonomous vehicles (Raffo et al., 2009; Ji et al., 2016; Brüdigam et al., 2021). The primary objective of a model predictive controller (MPC) is to regulate the behavior of a dynamical system by predicting its future behavior with a mathematical model and optimizing control inputs accordingly (Morari and Lee, 1999). Specifically, the MPC repeatedly solves an optimization problem at each sampling instant, which takes into account a prediction of the future behavior of the system over a finite time horizon, to generate a control sequence that minimizes a chosen performance criterion over the entire prediction horizon. Therefore, the prediction accuracy of the process model used by the MPC is essential to the MPC's performance, and developing an accurate predictive model continues to be a mathematical and scientific challenge.

First-principles models, developed based on a comprehensive understanding of the process mechanism, are the most robust and accurate predictive models that an MPC can use. However, developing such models is usually a time-consuming task that requires physicochemical information, which may not be available in most cases. To address this challenge, data-driven modeling methods have been used in chemical engineering to facilitate and generalize the modeling process while reducing costs. An example of a data-driven application in classical process control is the Fuzzy logic technique developed in the 1960s (Zadeh, 1965), which has been applied in various chemical engineering research efforts (Yaacob et al., 2001; Osofisan and Obafaiye, 2007; Liao et al., 2008). However, Fuzzy Logic Control (FLC) is usually implemented as a model-free approach (Sarmasti Emami, 2019), which is conceptually different from model-based approaches such as MPC. One advanced recent technique that is conceptually similar to the FLC is reinforcement learning (Nian et al., 2020). With the advent of the 21st century, machine learning methods, which are a subset of data-driven techniques, have been widely used and demonstrated great success in both classification and regression problems. The classical machine learning methods such as Gaussian process (GP) regression (Hewing et al., 2019; Zhang et al., 2022) and support vector regression (Xi et al., 2007; Çıtmacı et al., 2022) became popular choices to model physical

* Corresponding author at: Department of Chemical and Biomolecular Engineering, University of California, Los Angeles, CA, 90095-1592, USA.
E-mail address: pdc@seas.ucla.edu (P.D. Christofides).

https://doi.org/10.1016/j.compchemeng.2023.108367
Received 7 May 2023; Received in revised form 20 July 2023; Accepted 25 July 2023
Available online 1 August 2023

systems using data for process control purposes. With the development of computer software and hardware, more advanced and computationally intensive methods such as neural network (NN) modeling have also become accessible in conventional engineering disciplines. Since then, various NN techniques, including but not limited to feedforward neural network (FNN) (Mohanty, 2009; Kittisupakorn et al., 2009; Yun et al., 2022; Tom et al., 2022), recurrent neural network (RNN), and convolution neural network (CNN) (Han et al., 2019) approaches have led to significant innovations in process control research. In particular, RNN models trained with time-series data have been demonstrated to be an effective method to develop predictive models for nonlinear systems to be used as the process model in MPC for chemical process control (Wong et al., 2018; Xiao et al., 2021; Wu et al., 2021). In addition to NN approaches, other nonlinear methods that have been widely used in the process systems engineering literature to model dynamical systems for process control include nonlinear autoregressive methods (Billings, 2013) and sparse identification for nonlinear dynamics (SINDy) (Abdullah et al., 2021, 2022a; Abdullah and Christofides, 2023).

Recently, a novel class of neural network models, the neural ordinary differential equation (NODE), was proposed and has attracted significant attention due to their ability to learn continuous-time dynamical systems. As stated in Chen et al. (2018), due to its continuous nature, the NODE model can incorporate time-series data having an arbitrary time span between each data point, which gives it an important advantage compared to the RNN, which is a discrete approximation of the time sequence data that requires a consistent time step size in the training set and provides prediction having the same time step sizes. On the other hand, the performance of the SINDy method is highly dependent on the candidate terms in the predefined bank of basis functions. Similar to other NN methods, the NODE has a high degree of freedom in its model structure and weight matrices, which makes it more generalizable than SINDy. Therefore, this new modeling method has been widely tested in various areas to identify and control physical processes. For example, Lai et al. (2021) confirmed the feasibility of using NODE in linear and nonlinear structural identification, demonstrating their results on several numerical and experimental systems. Bradley and Boukouvala (2021) proposed a framework using the NODE model to extract parameters that describe the derivatives of physical states from the data. Furthermore, in Chee et al. (2023), a knowledge-based neural ordinary differential equation (KNODE) model that can capture a dynamic process was developed and implemented by designing a KNODE-based MPC for a quadrotor system. These advances illustrate the potential of NODE-based modeling for physical systems and motivate its use in the development of MPC for chemical processes.

Lastly, the application of all data-driven methods in a practical industrial system faces several critical challenges, one being that data collected from the system is usually corrupted by noise. Therefore, various researchers investigated methodologies to account for noisy data in the NN context. One way to handle noisy data is by smoothing the data with a noise filter in the data preprocessing step (Tran and Ward, 2017). But this method usually involves averaging the measurements within a certain time window, which introduces an undesired delay to the control system. In addition to smoothing the data, algorithms can be used to reduce the effect of noise during the development of the NN model. For example, Luo et al. (2022) used a weighted loss function to train the NN model to account for experimental noisy data, while Wu et al. (2021) demonstrated the Monte Carlo dropout and co-teaching method as effective ways to handle noisy data in LSTM models. However, Liu et al. (2019) stated the dropout method cannot be used with the original NODE structure. Hence, they proposed a modification on the NODE model, namely the Stochastic Differential Equation (SDE), to allow the use of effective regularization methods (e.g., dropout) while preserving most of the design of the NODE model. Finally, Goyal and Benner (2022) proposed an NODE-based framework to extract the ground-truth trajectory from noisy measurements.

Motivated by the theoretical advantage of using the NODE model in modeling continuous-time systems and the recent results supporting its implementation in physical systems, this work aims to develop a Lyapunov-based MPC (LMPC) based on a NODE model. The NODE model is designed to capture complex nonlinear relationships in a chemical process, so that this NODE-based LMPC can potentially be implemented in an industrial chemical process. In terms of the specific novelty of this work with respect to the neural network modeling literature, this study demonstrates that the NODE modeling approach can be used as an additional option to capture the derivative information of the state variables, which is required to enforce the contractive stability constraint of the LMPC. Specifically, differently from RNN modeling, NODE models allow capturing the state derivative with its hidden state while the output is trained to fit the state trajectory. Additionally, the continuous property of the NODE model enables the use of the subsampling method to account for noisy data and deal with measurement noise more effectively compared to RNN models, since NODE can handle irregularly sampled data sets. The rest of this paper is organized into the following sections: in Section 2, the mathematical notation and background of this work will be introduced; the NODE structure, training algorithm, and technical details will be discussed in Section 3 and the LMPC design and stability analysis will be addressed in Section 4. In Section 5, a simulation case study of a chemical process is utilized to show the implementation of NODE modeling and the development of NODE-based LMPC. The performance of the NODE model and the NODE-based LMPC will be evaluated via open- and closed-loop simulations, respectively. Finally, Section 5 further demonstrates the application of NODE-based LMPC while considering the effect of Gaussian and non-Gaussian types of sensor noise.

2. Preliminaries

2.1. Notation

The time-derivative of x is represented by $\dot{x}$, that is, $\dot{x} := \frac{\partial x}{\partial t}$. $\hat{y}$ denotes the prediction of y using a mathematical model, and $\hat{V}(x) = V(\hat{x})$. $\mathbf{v}^{\top}$ represents the transpose of $\mathbf{v}$. "$\backslash$" stands for subtracting one set from another, such that $A\backslash B := \{x \in \mathbb{R}^n | x \in A, x \notin B\}$. A continuous function $\alpha : [0, a) \to [0, \infty)$ belongs to class $\mathcal{K}$ if it is strictly increasing and achieves a value of zero only when evaluated at zero.

2.2. Class of systems

The general class of continuous-time systems considered in this research can be expressed by the following equations:

$$\dot{x} = F(x, u) \tag{1a}$$

$$y = x + v \tag{1b}$$

$$x(t_0) = x_0 \tag{1c}$$

where $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$ are the state vector and the manipulated input vector, respectively. An arbitrary nonlinear function, $F : \mathbb{R}^{n+m} \to \mathbb{R}^n$, mapping the state and input vectors to the time-derivative of the system, is assumed to be continuous and sufficiently smooth. $v \in \mathbb{R}^n$ represents the sensor noise affecting the state measurement y. t_0 and x_0 are used to denote the initial time and the corresponding initial state, respectively. Without loss of generality, the values of t_0 and x_0 are taken to be zero. By assuming $F(0, 0) = 0$ and that the system is in deviation form, i.e., $x_d = x - x_s; u_d = u - u_s$ where subscripts d and s denote deviation variables and the steady-state values of the state and input vectors, respectively, the steady-state of the nominal system with $v(t) = 0$ is located at the origin of the state space.

2.3. Defining Lyapunov-based stability region

To ensure that the nominal system of Eq. (1) can be used to construct a feasible process control problem, we first define an open region D in the state space around a selected set-point, which is a steady state of the system, such that the nominal system is closed-loop stable in the sense that any instantaneous state belonging to D can be brought to the set-point under a certain controller. Specifically, the stability criteria can be mathematically defined as the existence of a Lyapunov function $V(x)$ and a stabilizing controller $\Phi(x)$ such that, $\forall x$ in the region D, the following inequalities hold:

$$c_1|x|^2 \leq V(x) \leq c_2|x|^2 \tag{2a}$$

$$\dot{V}(x) = \frac{\partial V(x)}{\partial x} F(x, \phi(x)) \leq -c_3|x|^2 \tag{2b}$$

$$\left|\frac{\partial V(x)}{\partial x}\right| \leq c_4|x| \tag{2c}$$

where $V(x)$ is a Lyapunov function, and c_1, c_2, c_3, c_4 are positive constants. In other words, Eq. (2) implies the existence of a controller that can ensure exponential stability of the state x around the set-point. One candidate controller can be the universal Sontag controller (Lin and Sontag, 1991). Therefore, one can first find a region (D) where the time-derivative of the Lyapunov function ($\dot{V}$) is negative under the controller $\Phi(x)$. Subsequently, we pick a subset of D, namely Ω_ρ, to be our stability region, such that $\Omega_\rho := \{x \in D \mid V(x) \leq \rho\}$ where $\rho > 0$ and $\Omega_\rho \subset D$. The set of values of x that gives $V(x)$ equal to a positive constant ρ is the boundary of our stability region. Finally, the Lipschitz property of $F(x,u)$ combined with the bound on u implies the existence of positive constants M, L_x, L'_x such that the following inequalities hold for all $x, x' \in D, u \in U$:

$$|F(x,u)| \leq M \tag{3a}$$

$$|F(x,u) - F(x',u)| \leq L_x|x - x'| \tag{3b}$$

$$\left|\frac{\partial V(x)}{\partial x}F(x,u) - \frac{\partial V(x')}{\partial x}F(x',u)\right| \leq L'_x|x - x'| \tag{3c}$$

2.4. Neural network approximation of time-series data

Most popular neural network structures, such as the RNN family (e.g., vanilla RNN, Gated Recurrent Units (GRU), Long Short-Term Memory (LSTM) Units), compute their output using various logistic units (or neurons) to perform nonlinear transformations on the received input and propagate the result as the input of the next neuron. The result of the nonlinear transformations flowing from one neuron to another is named the hidden state. In the case of using an RNN model to capture time-series data, one of the common applications of neural network modeling in process control, the hidden state is passed chronologically until the desired final time step. Therefore, the RNN prediction for a time sequence state can be summarized as the following equation (Chen et al., 2018):

$$h(k+1) = h(k) + f(h(k), x_{rnn}(k)) \tag{4}$$

where $h(k)$, $x_{rnn}(k)$, and $f(\cdot)$ stand for the hidden state of the RNN model for the kth recurrent unit, the input for the k^{th} recurrent unit, which is usually the process measurements and control actions at time step k, and the nonlinear transformation performed on all received information in that unit. One may find that Eq. (4) is similar to the numerical integration step of an explicit integration scheme. Therefore, continuously adding recurrent units to an RNN model is equivalent to using a smaller integration time step, where the nonlinear transformation within the RNN unit, $f(h(k), x_{rnn}(k))$, evaluates the right-hand side of the ordinary differential equation (ODE). The Neural Ordinary Differential Equation (NODE) is designed based on this observation in Chen et al. (2018). The structure of NODE is designed such that by fitting the output of the model to the data, the hidden state of the NODE will fit to the time-derivative of the data. The technical details of NODE will be discussed in Section 3, following the proposed workflow in Chen et al. (2018).

2.5. Subsampling method

Subsampling is an effective statistical method that is used to reduce the size of the original data set by creating a subset of the original data (Hansen and Johnson, 2011). Subsampling is widely used to derive inferences on a larger data set by using a representative subsample at a lower computational cost. For example, subsampling was used to downsample large biomedical data sets while preserving the distribution in Lötsch et al. (2021), and computational power requirements for regression tasks were reduced through subsampling in Dai et al. (2023). Two of the most important tools used in developing machine learning models, train-test split and cross-validation, also belong to the subsampling family. In this work, we use subsampling to account for noisy data based on the assumption that some of the data points are more affected by noise than others. Therefore, by randomly selecting data points to create a subset that will be used to develop a prediction model, the impact of measurement noise can be mitigated, leading to the development of a better model. Subsampling has also been used in the data-driven modeling literature to account for data noise and has shown promising results (Huys and Paninski, 2009; Abdullah et al., 2022b), inspiring the use of subsampling to handle noisy data in our work.

3. Neural Ordinary Differential Equations (NODE)

3.1. NODE architecture

The NODE model in our work is developed to predict the state of a dynamical system using the following equation:

$$\hat{x}(t_k+t_p) = \text{ODESolver}\left(x(t_k), t_k, t_k+t_p, \mathfrak{f}(\hat{x},u)\right) = F_{nn}(x_0, u, t_k, t_k+t_p, \mathfrak{f}) \tag{5}$$

where $\hat{x} \in \mathbb{R}^n$ is the NODE state vector and $x(t_k)$ represents the state of the nominal system of Eq. (1) at time step t_k. t_p stands for the time span from the initial state measurement at t_k. $\mathfrak{f}$ is the NN model used to capture the nonlinear dynamic relationship of the system, i.e., the right-hand side of the ODE representing the time-derivative,

$$\dot{\hat{x}} = \mathfrak{f}(\hat{x}, u) \tag{6}$$

Since we are working on a regression task with numerical data, an FNN model is used as the function $\mathfrak{f}$. For convenience, we refer to the NN model $\mathfrak{f}$ used in the NODE as the "core model" in the remainder of this manuscript, while F_{nn} is used to denote the overall NODE model, whose output is the state prediction itself following integration. The fundamental mathematical idea behind the NODE model is to find the optimum weight matrix W^* of the FNN (core) model, such that the output of the FNN model can be used by the ODE solver to numerically calculate the predicted state from an arbitrary initial state measurement $x(t_k)$. Based on the universal approximation theorem (Hornik et al., 1990; Hornik, 1991), the core model of a perfectly trained NODE model can capture the right-hand side of the ODE of the desired system. Training of the NODE model refers to the process of optimizing the FNN weights based on the data.

The architecture of the NODE model is demonstrated in Fig. 1. Specifically, when using a trained NODE model to make a prediction, the instantaneous state measurement needs to be provided and will be used as the initial value of the ODE solver. The sequence of control inputs also needs to be provided to the NODE model, which will be used as the input of the FNN model to predict the "time-derivative of the state". The ODE solver will recursively call the FNN model to find the time-derivative and update the predicted state from the initial state value until the final time step of the prediction is reached. Finally, the prediction computed by the NODE model can be returned as a single value or as a vector of the state predictions at the desired time steps. If the prediction is returned as a time sequence, the intermediate time steps in the returning sequence need to be pre-defined and used as an argument of the prediction function. However, more intermediate

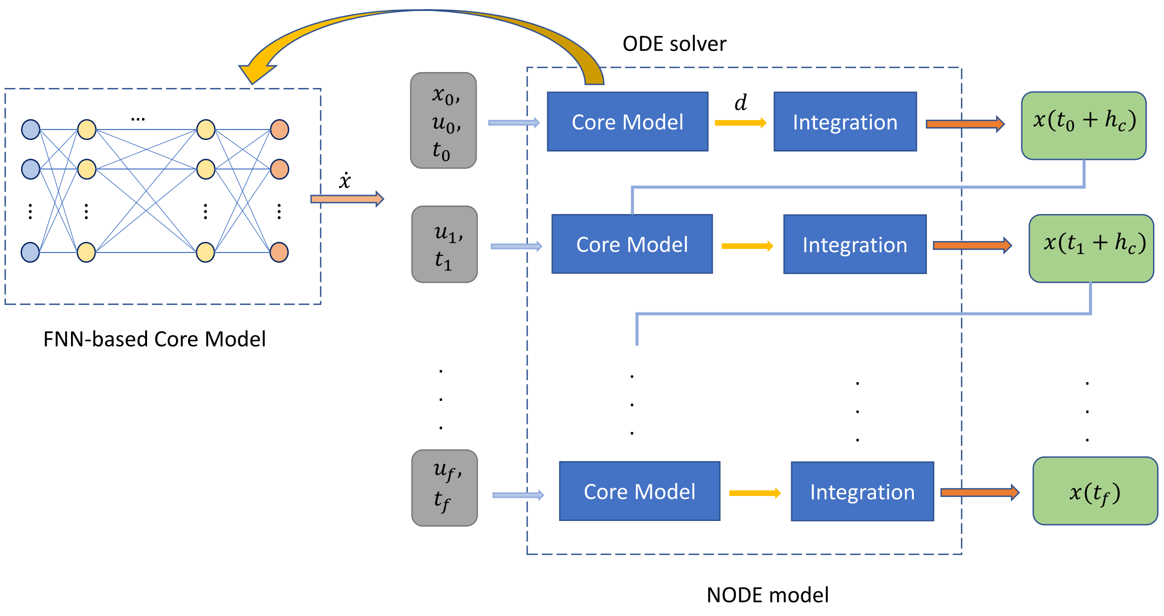


Fig. 1. The architecture of the neural ordinary differential equation (NODE) model. The NODE model contains a nonlinear core function that maps the input to its hidden state, such that the time-series state prediction can be found by integrating the hidden state using an ODE solver. The core function used in this work is a feedforward neural network model whose structure is shown on the left. The blue circles in the FNN represent the input information represented in gray in the figure on the right-hand side.

states in the output will result in higher computational costs. Lastly, the output of the core model can be considered the hidden state of the NODE model and has a very similar value to the state derivative. However, although we defined and used the output of the core model in the NODE model as the time-derivative of the states, it actually does not have any physical meaning. Therefore, various types of regressive models can be used as the core model, but in the case of regressing a physical system, the FNN model is a popular candidate. In contrast, for an image classification task, for example, using a convolutional neural network (CNN) as the core model is the most common approach.

3.2. Back-propagation

In the last subsection, we introduced the architecture of the NODE model and the unique characteristic of this model, which is the integration with an ODE solver within the neural network model. Since the ODE solver is involved in the training of the NODE model and impacts the backpropagation algorithm, the type of ODE solver (e.g., explicit Euler, Runge–Kutta, etc.) becomes one of the hyperparameters to be defined before training. This design differentiates the NODE from the common approach of training an FNN model with state time-derivatives and then computing the state prediction by using an explicit integration method with the output of the FNN model. Specifically, for the second approach, time-derivatives of the state need to be included in the training data set and used as the reference data for the model training. Thus, the state time-derivatives have to be either measured (if measurable, e.g., velocity) or numerically approximated from the raw data of state measurements to develop the training data set. Additionally, in the second approach, the ODE solver is detached from the FNN model, so it is not involved in the model training.

The NODE model used in this study is an example of a supervised learning technique, which optimizes the parameters of a model by minimizing the difference between the model output and a reference data set. Training a neural network model includes two major steps: forward and backward propagation (Dongare et al., 2012; Cilimkovic, 2015; Ren et al., 2022). The forward propagation of the neural network model is a straightforward process that propagates the input through all the layers of the neural network model to compute the output (Zhang et al., 2021). Subsequently, the model output will be compared with the reference data set to calculate the loss of the model with respect to the user-defined loss function. The mean squared error (MSE) is a popular loss function for regression problems and is used in this study. Once calculated, the loss will be propagated backward, from the output layer to the input layer, to compute the derivative of the loss with respect to each weight parameter of the model. Finally, the weight parameters are optimized by the gradient descent method described by Eq. (7a) below:

$$\mathbf{W}^{k+1} = \mathbf{W}^{k} - \alpha \frac{\partial Loss}{\partial \mathbf{W}} \tag{7a}$$

$$Loss = \frac{1}{2}\sum_{i=1}^{N_d} (\hat{Y}_i - Y_i)^{\top} (\hat{Y}_i - Y_i) \tag{7b}$$

where α is known as the learning rate, N_d denotes the number of data points in the training set, and $\mathbf{W} = \{w_i \mid i = 1, 2, \dots, N\}$ is the weight matrix containing all N weight parameters in the neural network model. $\frac{\partial Loss}{\partial \mathbf{W}}$ is the loss gradient with respect to each weight parameter, $Y_i = [y_{1,i}, y_{2,i}, \dots, y_{n,i}]$ is the i^{th} state measurement, and $\hat{Y}$ is the model prediction of Y. In an FNN model, neurons are densely connected to each other, multiplying their respective weight values during the propagation phases. Therefore, the loss gradient with respect to the weight parameters can be easily computed by applying the chain rule iteratively from the output layer to the input layer. However, in NODE, the output given by the output layer of the FNN model is not directly used in computing the model loss. Considering the MSE of Eq. (7b) and the NODE definition of Eq. (5), the NODE prediction is computed by the ODE solver, which means the chain rule cannot be applied directly to propagate the gradient of loss.

3.2.1. Adjoint sensitivity method

Chen et al. (2018) proposed to use the adjoint sensitivity method to propagate the gradient of loss through the ODE solver in the NODE model. The adjoint sensitivity method, proposed in Pontryagin (1987), is a popular method used in scientific research to efficiently compute the gradient of a model loss with respect to model parameters (or inputs). In Errico (1997), a detailed process to develop an adjoint model is introduced and an example of its application in meteorology demonstrated. Chen et al. (2018) provided the mathematical derivation and proof of applying the adjoint method in the development of NODE. In short, an adjoint state will be created to represent any derivative information that is useful to train the neural network model. By doing so, the time-derivative of the adjoint state can be formulated based on the chain rule. Finally, the loss gradients can be computed by integrating the time-derivative of the adjoint states backward in time. A detailed implementation of this training algorithm is demonstrated in the following steps:

1. **Define adjoint states:** The first step in training the NODE model is to define the adjoint states. One can start this step by first identifying what gradient information is needed to train their NN model. The first adjoint state is defined as the loss derivative with respect to the NODE output, which can be represented as $\mathbf{a} = \frac{\partial Loss}{\partial \hat{Y}}$. Since the loss function is usually selected to be an explicit quantity (e.g., MSE, MAE), the gradient $\frac{\partial Loss}{\partial \hat{Y}}$ can be computed analytically using the model prediction $\hat{Y}$. Subsequently, considering the case of using FNN as the core model, the second adjoint state is defined to represent the loss derivative with respect to the FNN weight parameters, that is, $\mathbf{a}_\mathbf{W} = \frac{\partial Loss}{\partial \mathbf{W}}$. In this study, the ODE function of the system is assumed to not contain any explicit term in time. Therefore, time is not an input of the core model, but it can be included following a similar approach. Lastly, all adjoint states are augmented into a column vector to perform the next step of the calculation.
2. **Set up the time-derivative of the adjoint states:** With an augmented adjoint state vector, $\mathbf{a}_{\text{aug}} = [\mathbf{a}\ \mathbf{a}_\mathbf{W}]^\top$, following the derivation in Chen et al. (2018), the time-derivative of the adjoint state can be expressed in the following form:

$$\frac{\partial\left(\frac{\partial Loss}{\partial \hat{Y}}(t)\right)}{\partial t} = \frac{\partial \mathbf{a}(t)}{\partial t} = -\mathbf{a}(t)\frac{\partial \mathfrak{f}}{\partial \hat{Y}} \tag{8a}$$

$$\frac{\partial\left(\frac{\partial Loss}{\partial \mathbf{W}}(t)\right)}{\partial t} = \frac{\partial \mathbf{a}_\mathbf{W}(t)}{\partial t} = -\mathbf{a}(t)\frac{\partial \mathfrak{f}}{\partial \mathbf{W}} \tag{8b}$$

Since $\mathbf{W}$ is the weight vector that contains all the N weight parameters in the core model, Eq. (8b) is a vector of N equations, which can be represented as $\frac{\partial \mathbf{a}_{\mathbf{W}_i}(t)}{\partial t} = -\mathbf{a}(t)\frac{\partial \mathfrak{f}}{\partial \mathbf{W}_i}, i = 1, \ldots, N$ respectively.
3. **Integrate backward in time:** Based on the definition of the adjoint states and their time-derivatives in the previous steps, we can compute the numerical values of the adjoint states by integrating Eq. (8) backward in time. Specifically, we denote the initial time step by t_0 and the final time step by t_f. Based on Eq. (7b), the adjoint state $\mathbf{a}(t_f)$ will simply be the sum of the model predictions at the final time step for each trajectory in the training set, i.e., $\sum \hat{Y}(t_f)$. With the adjoint state $\mathbf{a}(t_f)$ known, if we assume $\mathbf{a}_\mathbf{W}(t_f)$ to be zero, the adjoint states at t_0 can be found by the following expression, where t' is a notational substitute for t:

$$\mathbf{a}(t_0) = \mathbf{a}(t_f) - \int_{t_f}^{t_0} \mathbf{a}(t')\frac{\partial \mathfrak{f}}{\partial \hat{Y}}\,\mathrm{d}t' \tag{9a}$$

$$\mathbf{a}_\mathbf{W}(t_0) = \mathbf{a}_\mathbf{W}(t_f) - \int_{t_f}^{t_0} \mathbf{a}(t')\frac{\partial \mathfrak{f}}{\partial \mathbf{W}}\,\mathrm{d}t' \tag{9b}$$

which can be solved with an ODE solver by approximating the partial derivative terms in Eq. (9) using the automatic differentiation method. At the end, the gradient of the loss with respect to weights at the initial time step, $\mathbf{a}_\mathbf{W}(t_0)$, is used to update the model based on Eq. (7a).

3.2.2. *Automatic differentiation*

Automatic differentiation (AD) is an efficient and cheap method to approximate the gradient between two variables and is widely used in the development of neural network models. The AD method is well-developed and supported in modern machine learning Application Programming Interfaces (APIs). For example, AD is provided as the `autograd` function in PyTorch and the `GradientTape` function in TensorFlow. Both machine learning APIs use a computational graph to implement the AD method. A simplified demonstration of a computational graph is a map that reflects all the connections in a neural network model and has a database for the derivative of various common math operations. For example, if a result (c) in a computational graph is computed by multiplying two inputs ($c = a \times b$), then the derivative of the output with respect to either of the inputs is simply the other input ($\frac{\partial c}{\partial a} = b$), and the derivative of the output with respect to all of its inputs will be stored in the computational graph. When performing the forward propagation of the neural network model, all necessary gradients will be computed and stored. Therefore, during backpropagation, the loss of the gradient with respect to any parameter can be systematically computed based on the chain rule. In this work, since we used PyTorch to develop our model, the derivative terms in Eq. (9) are approximated using the `autograd` function.

4. Lyapunov-based model predictive control using NODE models

This section formulates the design of an LMPC designed using the NODE model of Eq. (5) to predict the future state trajectory, and then presents a closed-loop stability analysis of the nonlinear system of Eq. (1) under the proposed NODE-based LMPC. Due to the sample-and-hold implementation of the controller, the closed-loop states can only be driven to a small neighborhood around the origin. We clarify, for the subsequent propositions and proofs, that the core model of the NODE model represents the time-derivative of the state, i.e., $\hat{\dot{x}}$, which is also the right-hand side of the ODE model to be captured.

4.1. *Lyapunov-based control using NODE models*

We assume the existence of a stabilizing feedback controller $u = \Phi_{nn}(x) \in U$ that renders the origin of the core model of Eq. (6) exponentially stable in an open neighborhood around the origin $\hat{\phi}_u \in \mathbb{R}^n$ in the sense that there exist a continuously differentiable control Lyapunov function $\hat{V}(x)$ and positive constants $\hat{c}_1$, $\hat{c}_2$, $\hat{c}_3$, $\hat{c}_4$ such that the following inequalities hold for all $x \in \hat{D}$:

$$\hat{c}_1|x|^2 \le \hat{V}(x) \le \hat{c}_2|x|^2 \tag{10a}$$

$$\dot{\hat{V}}(x) = \frac{\partial \hat{V}(x)}{\partial x}\mathfrak{f}(x, \Phi_{nn}(x)) \le -\hat{c}_3|x|^2 \tag{10b}$$

$$\left|\frac{\partial \hat{V}(x)}{\partial x}\right| \le \hat{c}_4|x| \tag{10c}$$

We begin by characterizing the region $\hat{\phi}_u$ where the constraints of Eq. (10) are met under the controller $u = \Phi_{nn}(x)$, followed by defining the closed-loop stability region of the NODE model of Eq. (6) to be a level set of the Lyapunov function inside $\hat{\phi}_u$: $\Omega_{\hat{\rho}} := \{x \in \hat{\phi}_u \mid \hat{V}(x) \le \hat{\rho}\}$ where $\hat{\rho} > 0$. The assumptions of Eq. (2) and Eq. (10) are the stabilizability requirements of the nonlinear system of Eq. (1) and the NODE model of Eq. (6), respectively.

As the data set for constructing the NODE model is generated via open-loop simulations with $x \in \Omega_\rho$ and $u \in U$, we have $\Omega_{\hat{\rho}} \subseteq \Omega_\rho$. Moreover, there exist positive constants M_{nn} and L_{nn} such that the following inequalities hold for all $x, x' \in \Omega_{\hat{\rho}}$ and $u \in U$:

$$|\mathfrak{f}(x,u)| \le M_{nn} \tag{11a}$$

$$\left|\frac{\partial \hat{V}(x)}{\partial x}\mathfrak{f}(x,u) - \frac{\partial \hat{V}(x')}{\partial x}\mathfrak{f}(x',u)\right| \le L_{nn}|x - x'| \tag{11b}$$

The following proposition is developed to demonstrate that the feedback controller $u = \Phi_{nn}(x)$ can stabilize the nominal system of Eq. (1), despite the model mismatch between Eq. (1) and the NODE model of Eq. (6), if the modeling error is sufficiently small.

Proposition 1 (*c.f. proposition 2 in Wu et al. (2019)*). *Under the assumption that the origin of the closed-loop NODE system of Eq. (6) is rendered exponentially stable under the controller $u = \Phi_{nn}(x) \in U\ \forall x \in \Omega_{\hat{\rho}}$, if there exists a positive real number $\gamma < \hat{c}_3/\hat{c}_4$ that constrains the modeling error $|\nu| = |F(x,u) - \mathfrak{f}(x,u)| \leq \gamma|x|$, $\forall u \in U$ and $\forall x \in \Omega_{\hat{\rho}}$, then the origin of the nominal closed-loop system of Eq. (1) under $u = \Phi_{nn}(x) \in U$ is also exponentially stable $\forall x \in \Omega_{\hat{\rho}}$.*

Proof. To prove that the origin of the nominal system of Eq. (1) may be rendered exponentially stable $\forall x \in \Omega_{\hat{\rho}}$ under the controller designed using the NODE model of Eq. (5), we prove that $\dot{\hat{V}}$ for the nominal system of Eq. (1) is still rendered negative for all $x \in \Omega_{\hat{\rho}}$ under the controller $u = \Phi_{nn}(x)$. The time-derivative of $\hat{V}$ is computed based on Eqs. (10b) and (10c), as follows:

$$\begin{aligned}\dot{\hat{V}} &= \frac{\partial \hat{V}(x)}{\partial x} F(x, \Phi_{nn}(x)) \\ &= \frac{\partial \hat{V}(x)}{\partial x}\big(\mathfrak{f}(x,\Phi_{nn}(x)) + F(x,\Phi_{nn}(x)) - \mathfrak{f}(x,\Phi_{nn}(x))\big) \\ &\leq -\hat{c}_3|x|^2 + \hat{c}_4|x|\big(F(x,\Phi_{nn}(x)) - \mathfrak{f}(x,\Phi_{nn}(x))\big) \\ &\leq -\hat{c}_3|x|^2 + \hat{c}_4\gamma|x|^2\end{aligned} \tag{12}$$

By choosing the modeling error γ to satisfy $\gamma < \hat{c}_3/\hat{c}_4$, it can be ensured that $\dot{\hat{V}} \leq -\tilde{c}_3|x|^2 \leq 0$ where $\tilde{c}_3 = -\hat{c}_3 + \hat{c}_4\gamma > 0$. Consequently, the closed-loop state of the nominal system of Eq. (1) converges to the origin $\forall x_0 \in \Omega_{\hat{\rho}}$ under $u = \Phi_{nn}(x) \in U$. □

Remark 1. In this work, the terminology of "modeling error" is used to describe the difference between Eq. (1) and Eq. (6) because the core model of the trained NODE model is expected to capture the right-hand-side of the ODE function of Eq. (1). However, the NODE model is trained using the measured states, i.e., the solution of Eq. (1b), such that the output of the core model of Eq. (6) is the hidden state of the NODE model. Hence, the training loss of the NODE model refers to the error in the state, while the modeling error refers to the error in the time-derivative.

4.2. Sample-and-hold implementation of Lyapunov-based MPC

Since the Lyapunov-based MPC designed using the NODE model of Eq. (5) is implemented in a sample-and-hold fashion, in the next two propositions, the sample-and-hold properties of the Lyapunov-based controller $u = \Phi_{nn}(x)$ are derived. In particular, the following proposition derives an upper bound for the error between the states calculated by the nominal system of Eq. (1) and the states predicted by the NODE model of Eq. (5).

Proposition 2. *(c.f. proposition 3 in Wu et al. (2019)) For the nonlinear system $\dot{x} = F(x,u)$ of Eq. (1) and the NODE core model $\dot{\hat{x}} = \mathfrak{f}(\hat{x},u)$ of Eq. (6) with the same initial condition $x_0 = \hat{x}_0 \in \Omega_{\hat{\rho}}$, there exist a function $f_w(\cdot)$ of class $\mathcal{K}$ and a positive constant κ such that the following inequalities hold $\forall x, \hat{x} \in \Omega_{\hat{\rho}}$:*

$$|x(t) - \hat{x}(t)| \leq f_w(t) := \frac{\nu_m}{L_x}(e^{L_x t} - 1) \tag{13a}$$

$$\hat{V}(x) \leq \hat{V}(\hat{x}) + \frac{\hat{c}_4\sqrt{\hat{\rho}}}{\sqrt{\hat{c}_1}}|x - \hat{x}| + \kappa|x - \hat{x}|^2 \tag{13b}$$

Proof. The error vector between the solutions of the nonlinear system of Eq. (1) and the NODE model of Eq. (5) is defined as $e(t) = x(t) - \hat{x}(t)$, whose time-derivative can be calculated as follows:

$$\begin{aligned}|\dot{e}(t)| &= |F(x,u) - \mathfrak{f}(\hat{x},u)| \\ &\leq |F(x,u) - F(\hat{x},u)| + |F(\hat{x},u) - \mathfrak{f}(\hat{x},u)|\end{aligned} \tag{14}$$

In Eq. (14), the first term can be bounded using Eq. (3b) as follows:

$$|F(x,u) - F(\hat{x},u)| \leq L_x|x(t) - \hat{x}(t)|, \qquad \forall x, \hat{x} \in \Omega_{\hat{\rho}}, \tag{15}$$

while the second term $|F(\hat{x},u) - \mathfrak{f}(\hat{x},u)|$ represents the modeling error, which is bounded by $|\nu| \leq \nu_m\ \forall \hat{x} \in \Omega_{\hat{\rho}}$. As a result, based on Eq. (15) and the bounded modeling error, $\dot{e}(t)$ is bounded as follows:

$$\begin{aligned}|\dot{e}(t)| &\leq L_x|x(t) - \hat{x}(t)| + \nu_m \\ &\leq L_x|e(t)| + \nu_m\end{aligned} \tag{16}$$

The norm of the error vector can be bounded as follows for all $x(t), \hat{x}(t) \in \Omega_{\hat{\rho}}$ using the zero initial condition (i.e., $e(0) = 0$):

$$|e(t)| = |x(t) - \hat{x}(t)| \leq \frac{\nu_m}{L_x}(e^{L_x t} - 1) \tag{17}$$

Finally, to derive Eq. (13b) $\forall x, \hat{x} \in \Omega_{\hat{\rho}}$, we derive the Taylor series expansion of $\hat{V}(x)$ around $\hat{x}$,

$$\hat{V}(x) \leq \hat{V}(\hat{x}) + \frac{\partial \hat{V}(\hat{x})}{\partial x}|x - \hat{x}| + \kappa|x - \hat{x}|^2 \tag{18}$$

where κ is a positive real number. Using Eqs. (10a) and (10c), it follows that

$$\hat{V}(x) \leq \hat{V}(\hat{x}) + \frac{\hat{c}_4\sqrt{\hat{\rho}}}{\sqrt{\hat{c}_1}}|x - \hat{x}| + \kappa|x - \hat{x}|^2, \tag{19}$$

which completes the proof of Proposition 2. □

The final proposition below is developed to prove that the closed-loop state of the actual nonlinear system of Eq. (1) remains bounded in $\Omega_{\hat{\rho}}$ for all times, and can be ultimately bounded in a small neighborhood around the origin, denoted by $\Omega_{\rho_{\min}}$, under the sample-and-hold implementation of the Lyapunov-based controller $u = \Phi_{nn}(x) \in U$.

Proposition 3. *Consider the nonlinear system of Eq. (1) under the controller $u = \Phi_{nn}(\hat{x}) \in U$ that meets the conditions of Eq. (10) and is designed to stabilize the NODE system of Eq. (6). The controller is implemented in a sample-and-hold fashion, such that $u(t) = \Phi_{nn}(\hat{x}(t_k))$, $\forall t \in [t_k, t_{k+1})$, where $t_{k+1} := t_k + \Delta$. Furthermore, let ϵ_s, $\epsilon_w > 0$, $\Delta > 0$ and $\hat{\rho} > \rho_{\min} > \rho_{sp} > \rho_s$ satisfy*

$$-\frac{\tilde{c}_3}{\hat{c}_2}\rho_s + L_{nn}M_{nn}\Delta \leq -\epsilon_s \tag{20a}$$

$$-\frac{\tilde{c}_3}{\hat{c}_2}\rho_s + L'_x M\Delta \leq -\epsilon_w \tag{20b}$$

and

$$\rho_{sp} := \max\{\hat{V}(\hat{x}(t+\Delta)) \mid \hat{x}(t) \in \Omega_{\rho_s}, u \in U\} \tag{21a}$$

$$\rho_{\min} \geq \rho_{sp} + \frac{\hat{c}_4\sqrt{\hat{\rho}}}{\sqrt{\hat{c}_1}}f_w(\Delta) + \kappa(f_w(\Delta))^2 \tag{21b}$$

Based on the above assumptions, for any $x(t_k) \in \Omega_{\hat{\rho}}\backslash\Omega_{\rho_s}$, the following inequality holds:

$$\hat{V}(x(t)) \leq \hat{V}\left(x(t_k)\right), \forall t \in [t_k, t_{k+1}) \tag{22}$$

and the state $x(t)$ of the nonlinear system of Eq. (1) is bounded in the level set $\Omega_{\hat{\rho}}$ for all times and ultimately trapped in the smaller level set $\Omega_{\rho_{\min}}$.

Proof. Part 1: Assuming $x(t_k) = \hat{x}(t_k) \in \Omega_{\hat{\rho}}\backslash\Omega_{\rho_s}$, it will be first shown that the value of the Lyapunov function $\hat{V}(\hat{x})$ is decreasing under the controller $u(t) = \Phi_{nn}(x(t_k)) \in U\ \forall t \in [t_k, t_{k+1})$, where $x(t)$ and $\hat{x}(t)$ denote the solutions of the nonlinear system of Eq. (1) and the NODE system of Eq. (5), respectively. The time-derivative of the Lyapunov function along the trajectory $\hat{x}(t)$ of the NODE model of Eq. (5) can be written $\forall t \in [t_k, t_{k+1})$ as

$$\begin{aligned}\dot{\hat{V}}(\hat{x}(t)) &= \frac{\partial \hat{V}(\hat{x}(t))}{\partial \hat{x}}\mathfrak{f}(\hat{x}(t), \Phi_{nn}(\hat{x}(t_k))) \\ &= \frac{\partial \hat{V}(\hat{x}(t_k))}{\partial \hat{x}}\mathfrak{f}(\hat{x}(t_k), \Phi_{nn}(\hat{x}(t_k))) + \frac{\partial \hat{V}(\hat{x}(t))}{\partial \hat{x}}\mathfrak{f}(\hat{x}(t), \Phi_{nn}(\hat{x}(t_k))) \\ &\quad - \frac{\partial \hat{V}(\hat{x}(t_k))}{\partial \hat{x}}\mathfrak{f}(\hat{x}(t_k), \Phi_{nn}(\hat{x}(t_k)))\end{aligned} \tag{23}$$

where the first term can be bounded as follows using Eqs. (10a) and (10b):

$$\dot{\hat{V}}(\hat{x}(t)) \leq -\frac{\hat{c}_3}{\hat{c}_2}\rho_s + \frac{\partial \hat{V}(\hat{x}(t))}{\partial \hat{x}} \mathfrak{f}(\hat{x}(t), \Phi_{nn}(\hat{x}(t_k))) - \frac{\partial \hat{V}(\hat{x}(t_k))}{\partial \hat{x}} \mathfrak{f}(\hat{x}(t_k), \Phi_{nn}(\hat{x}(t_k))) \tag{24}$$

In Eq. (24), bounding the difference using the Lipschitz condition of Eq. (11) with the fact that $\hat{x} \in \Omega_{\hat{\rho}}$, $u \in U$, $\dot{\hat{V}}(\hat{x}(t))$ can be upper-bounded $\forall\, t \in [t_k, t_{k+1})$:

$$\begin{aligned}\dot{\hat{V}}(\hat{x}(t)) &\leq -\frac{\hat{c}_3}{\hat{c}_2}\rho_s + L_{nn}|\hat{x}(t) - \hat{x}(t_k)| \\ &\leq -\frac{\hat{c}_3}{\hat{c}_2}\rho_s + L_{nn}M_{nn}\Delta\end{aligned} \tag{25}$$

Therefore, the satisfaction of the condition of Eq. (20a) ensures that the following inequality holds $\forall\, \hat{x}(t_k) \in \Omega_{\hat{\rho}} \backslash \Omega_{\rho_s}$, $\forall\, t \in [t_k, t_{k+1})$:

$$\dot{\hat{V}}(\hat{x}(t)) \leq -\epsilon_s \tag{26}$$

By integrating the aforementioned differential equation over the time interval $t \in [t_k, t_{k+1})$ with respect to time, it can be deduced that $V(\hat{x}(t_{k+1})) \leq V(\hat{x}(t_k)) - \epsilon_s\Delta$. So far, we have demonstrated that, for all $\hat{x}(t_k) \in \Omega_{\hat{\rho}} \backslash \Omega_{\rho_s}$, the closed-loop state of the NODE system of Eq. (5) remains confined within the closed-loop stability region $\Omega_{\hat{\rho}}$ at all times and progresses towards the origin under the controller $u = \Phi_{nn}(\hat{x}) \in U$ when implemented in a sample-and-hold approach.

It should be noted that Eq. (26) may not hold when $x(t_k) = \hat{x}(t_k) \in \Omega_{\rho_s}$, meaning that the state may exit Ω_{ρ_s} within a single sampling period. Therefore, we establish another region $\Omega_{\rho_{sp}}$ based on Eq. (21a) to ensure that the closed-loop state $\hat{x}(t)$ of the NODE model does not depart from $\Omega_{\rho_{sp}}$ over a single sampling period, i.e., during $t \in [t_k, t_{k+1})$, $\forall u \in U$, $\forall \hat{x}(t_k) \in \Omega_{\rho_s}$. If the state $\hat{x}(t_{k+1})$ exits Ω_{ρ_s}, Eq. (26) is satisfied again at $t = t_{k+1}$, thereby reactivating the controller $u = \Phi_{nn}(x(t_{k+1}))$ and directing the state towards Ω_{ρ_s} during the following sampling period. Consequently, it is demonstrated that the state approaches $\Omega_{\rho_{sp}}$ for the closed-loop NODE system of Eq. (5) for all $\hat{x}_0 \in \Omega_{\hat{\rho}}$. In Part 2, we demonstrate that the closed-loop state of the actual nonlinear process of Eq. (1) can also be confined within $\Omega_{\hat{\rho}}$ for all times and eventually contained within a small neighborhood around the origin with the sample-and-hold implementation of the controller $u = \Phi_{nn}(x) \in U$.

Part 2: We repeat the analysis carried out for the NODE system of Eq. (5). First, we suppose $x(t_k) = \hat{x}(t_k) \in \Omega_{\hat{\rho}} \backslash \Omega_{\rho_s}$ and derive the following expression for the time-derivative of $\hat{V}(x)$ for the nonlinear system of Eq. (1):

$$\begin{aligned}\dot{\hat{V}}(x(t)) &= \frac{\partial \hat{V}(x(t))}{\partial x} F(x(t), \Phi_{nn}(x(t_k))) \\ &= \frac{\partial \hat{V}(x(t_k))}{\partial x} F(x(t_k), \Phi_{nn}(x(t_k))) + \frac{\partial \hat{V}(x(t))}{\partial x} F(x(t), \Phi_{nn}(x(t_k))) \\ &\quad - \frac{\partial \hat{V}(x(t_k))}{\partial x} F(x(t_k), \Phi_{nn}(x(t_k)))\end{aligned} \tag{27}$$

From Eq. (12), it can be inferred that $\frac{\partial \hat{V}(x(t_k))}{\partial x} F(x(t_k), \Phi_{nn}(x(t_k))) \leq -\tilde{c}_3|x(t_k)|^2$ $\forall x \in \Omega_{\hat{\rho}} \backslash \Omega_{\rho_s}$. By utilizing Eq. (10a) and the definition of Lipschitz continuity in Eq. (11), the following inequality can be established $\forall\, t \in [t_k, t_{k+1})$ and $\forall\, x(t_k) \in \Omega_{\hat{\rho}} \backslash \Omega_{\rho_s}$:

$$\begin{aligned}\dot{\hat{V}}(x(t)) &\leq -\frac{\tilde{c}_3}{\hat{c}_2}\rho_s + \frac{\partial \hat{V}(x(t))}{\partial x} F(x(t), \Phi_{nn}(x(t_k))) \\ &\quad - \frac{\partial \hat{V}(x(t_k))}{\partial x} F(x(t_k), \Phi_{nn}(x(t_k))) \\ &\leq -\frac{\tilde{c}_3}{\hat{c}_2}\rho_s + L'_x|x(t) - x(t_k)| \\ &\leq -\frac{\tilde{c}_3}{\hat{c}_2}\rho_s + L'_x M\Delta\end{aligned} \tag{28}$$

Thus, if the condition of Eq. (20b) is fulfilled, the inequality below is valid $\forall\, x(t_k) \in \Omega_{\hat{\rho}} \backslash \Omega_{\rho_s}, \forall\, t \in [t_k, t_{k+1})$:

$$\dot{\hat{V}}(x(t)) \leq -\epsilon_w \tag{29}$$

The above differential equation may be integrated in time between any two points within the following sampling period, i.e., $[t_k, t_{k+1})$ to obtain the following inequalities for the Lyapunov function $\hat{V}$ for all $x(t_k) \in \Omega_{\hat{\rho}} \backslash \Omega_{\rho_s}$:

$$\hat{V}(x(t_{k+1})) \leq V(x(t_k)) - \epsilon_w\Delta \tag{30}$$

$$\hat{V}(x(t)) \leq \hat{V}(x(t_k)), \quad \forall\, t \in [t_k, t_{k+1}) \tag{31}$$

As a result, the state of the closed-loop system of Eq. (1) stays within $\Omega_{\hat{\rho}}$ for all time. In addition, the controller $u = \Phi_{nn}(x)$ can continue to steer the state of the nonlinear system of Eq. (1) towards the origin within each sampling period. Furthermore, if the initial state $x(t_k)$ satisfies $x(t_k) \in \Omega_{\rho_s}$, then it was already demonstrated in Part 1 that the state of the NODE model of Eq. (5) is confined within $\Omega_{\rho_{sp}}$ for one sampling period. Given the bounded modeling error between the actual nonlinear system of Eq. (1) and the NODE model of Eq. (5) described by Eq. (13a), there is a compact set $\Omega_{\rho_{\min}} \supset \Omega_{\rho_{sp}}$ satisfying Eq. (21b) such that the state of the nonlinear system of Eq. (1) remains within $\Omega_{\rho_{\min}}$ for one sampling period if the state of the NODE model of Eq. (5) is constricted within $\Omega_{\rho_{sp}}$. If the state $x(t)$ enters $\Omega_{\rho_{\min}} \backslash \Omega_{\rho_s}$, we have already demonstrated that Eq. (31) holds, and thus, under $u = \Phi_{nn}(x)$, the state will be driven back towards the origin during the next sampling period, ultimately constricting the closed-loop system to remain within $\Omega_{\rho_{\min}}$. Thus, the proof of Proposition 3 is completed, having shown that for any $x_0 = \hat{x}_0 \in \Omega_{\hat{\rho}}$, the closed-loop state trajectories of the nonlinear system described by Eq. (1) remain within $\Omega_{\hat{\rho}}$ and ultimately within $\Omega_{\rho_{\min}}$ if the assumptions of Proposition 3 are satisfied. □

4.3. Lyapunov-based MPC formulation

Model predictive control is an advanced process control technique that computes the control action by solving an optimization problem based on a given predictive model and feedback measurement. A Lyapunov-based MPC is a class of MPC with additional constraints based on the value of the Lyapunov function and its time-derivative at the current state. The optimization problem of LMPC can be written as follows:

$$\mathcal{J} = \min_{u \in S(\Delta)} \int_{t_k}^{t_{k+N_h}} L(\hat{x}(t), u(t))\, \mathrm{d}t \tag{32a}$$

$$\text{s.t.} \quad \hat{x}(t) = F_{nn}(x_0, u, t_k, t_{k+N_h}) \tag{32b}$$

$$u(t) \in U, \ \forall\, t \in [t_k, t_{k+N_h}) \tag{32c}$$

$$\hat{x}(t_k) = x(t_k) \tag{32d}$$

$$\dot{\hat{V}}(\hat{x}, u) \leq -k\hat{V}(\hat{x}), \text{ if } x(t_k) \in \Omega_\rho \backslash \Omega_{\rho_{sp}} \tag{32e}$$

$$\hat{V}(\hat{x}) \leq \rho_{sp}, \ \forall\, t \in [t_k, t_{k+N_h}), \text{ if } x(t_k) \in \Omega_{\rho_{sp}} \tag{32f}$$

where $L(\cdot)$ is the cost function based on the state and control input values, such that the objective of the optimization problem is to minimize the integral of the cost function in the time span between t_k and t_{k+N_h}, where N_h is the prediction horizon, which is an integer multiple of Δ. $S(\Delta)$ represents the set of piecewise constant functions with a period Δ, which the input u is restricted to. F_{nn} represents the NODE model, described by Eq. (5), that gives as its output the state prediction $\hat{x}(t)$ over the prediction horizon by integrating the core model of Eq. (6) starting from the initial condition of Eq. (32d), which is the state measurement at t_k. U denotes the allowable range of the control action u, and $\Omega_{\rho_{sp}} := \{x \in \phi(x) \mid \hat{V}(x) \leq \rho_{sp},\ \rho_{sp} < \rho\}$ is a subset of Ω_ρ that defines the process state region considered to be "close enough" to the set-point when deploying the LMPC for set-point tracking. The condition stated in Eq. (32e) guarantees that, when the state $x(t_k)$ is in the region $\Omega_{\hat{\rho}} \backslash \Omega_{\rho_{sp}}$, the controller will drive the state towards the origin. However, as soon as the state $x(t_k)$ enters the region $\Omega_{\rho_{sp}}$, the states predicted by the NODE model from Eq. (32b) will consistently stay within $\Omega_{\rho_{sp}}$ throughout the entire prediction horizon. The goal of the LMPC is to calculate the optimal sequence of control actions $u = u^*(t),\ t \in [t_k, t_{k+N_h})$ and apply the first move of the sequence, $u^*(t_k)$, over the next sampling period. Once the process evolves for one sampling period, the LMPC is resolved again at the next sampling period.

Remark 2. As we demonstrate in the application section, quadratic Lyapunov functions can be used to effectively design the Lyapunov-based stability constraints for the LMPC. It is important to note that, while the operating region is dependent on the choice of the Lyapunov function, the use of a quadratic Lyapunov function does not necessarily mean a small closed-loop stability (and operating) region; it really depends on the properties of the specific application. Non-quadratic Lyapunov functions could be considered for this task as well, but they are harder to construct. The design of Lyapunov functions is an important research area in nonlinear systems and control; various methods have been proposed to design a Lyapunov function (e.g., Grosman and Lewin, 2005; Giesl and Hafstein, 2015).

Remark 3. The motivation for using Lyapunov-based MPC is primarily its closed-loop stability guarantee. Specifically, when the states are initialized from a certain set of initial conditions, closed-loop stability of the LMPC can be guaranteed due to the propositions presented previously and the theorems that will be presented in the next subsection. In contrast, in MPC without Lyapunov-based contractive constraints, the set of initial conditions starting from which closed-loop stabilization of the original system can be achieved under MPC cannot be characterized for nonlinear systems as it depends on the system structure, constraints, horizon length, weight matrices and so on. Imposing a Lyapunov constraint of the type we impose on the MPC via Eq. (32e) (that the control action in the first move of the MPC reduces the value of the Lyapunov function as much as or more than the reduction that would be achieved by an explicit controller for which the stability region is easier to characterize) allows us to characterize the stability region under MPC because the MPC inherits the stability region of the explicit controller when the specific Lyapunov-based constraint is used (Mhaskar et al., 2005, 2006). The use of the specific Lyapunov constraint of Eq. (32e) in the LMPC provides a way to get an explicit characterization of the stability region which cannot be characterized in any other way. This characterization is clearly dependent on the choice of the Lyapunov function $\hat{V}(x)$ in terms of both its form and values of parameters/matrices used and is, of course, conservative in general. This is as far as stability can be guaranteed under MPC for general nonlinear systems today. The present work uses as the process model of the Lyapunov-based MPC a NODE model to calculate future state evolution and state derivatives, but the rest of the MPC formulation is unchanged. As such, the stability results of LMPC combined with the advantages of using NODE models that have already been explained in Section 3 are the motivation for using NODE-based LMPC in this work.

Remark 4. A further advantage of using NODE models in LMPC is its ability to readily compute the time-derivative of the Lyapunov function, which depends on the design of the function $\hat{V}(x)$. Considering the design to be $\hat{V}(x) = x^{\top}Px$ where P is a user-defined positive definite matrix, $\dot{\hat{V}}(x)$ can be expressed as $2x^{\top}P\dot{x}$. The time-derivative of the process states, $\dot{x}$, can be easily found if an explicit ODE of the system is known. In the case of an NODE model, the time-derivative of the process states can be approximated by the hidden state, which is the output of the core model. Therefore, numerical approximation of the output derivative ($\dot{x}$) by methods such as finite-differences, which is needed to develop an LMPC with RNN model, is not necessary. This property will be useful when a numerical approximation method cannot be applied, for example, a process has missing measurements that result in a relatively big time step gap, making numerical calculations of derivatives inaccurate. We also highlight that this advantage is independent of the form of the Lyapunov function, i.e., even for a non-quadratic Lyapunov function, the $\dot{x}$ term remains in its time-derivative and will need to be approximated.

4.4. Closed-loop stability analysis

The LMPC formulation presented in Eq. (32) is used to derive the following theorem, which guarantees recursive feasibility of the LMPC optimization problem and also closed-loop stability of the actual nonlinear system of Eq. (1) under the sample-and-hold implementation of the resulting optimal control actions.

Theorem 1. *Assuming the controller $\Phi_{nn}(x)$ satisfies Eq. (10), the closed-loop system of Eq. (1) under the LMPC of Eq. (32) is considered. Let $\Delta > 0, \epsilon_s > 0$, and $\hat{\rho} > \rho_{\min} > \rho_{sp} > \rho_s$ be such that they satisfy Eqs. (21a) and (21b). If the conditions of Propositions 2 and 3 are met, a feasible solution for the optimization problem of Eq. (32) always exists for any initial state $x_0 \in \Omega_{\hat{\rho}}$. Moreover, it is guaranteed that the LMPC of Eq. (32) maintains $x(t) \in \Omega_{\hat{\rho}}$ for all $t \geq 0$, and that $x(t)$ of the closed-loop system of Eq. (1) eventually converges to $\Omega_{\rho_{\min}}$.*

Proof. We begin by establishing the recursive feasibility of the optimization problem in Eq. (32) for all states $x \in \Omega_{\hat{\rho}}$. In particular, if at time t_k, $x(t_k) \in \Omega_{\hat{\rho}} \backslash \Omega_{\rho_{sp}}$, then the control action $u(t) = \Phi_{nn}(x(t_k)) \in U$, $t = [t_k, t_{k+1})$ computed using the state measurement $x(t_k)$ satisfies the input constraint of Eq. (32c) and the Lyapunov-based constraint of Eq. (32e). Furthermore, if $x(t_k) \in \Omega_{\rho_{sp}}$, the control actions obtained from $\Phi_{nn}(x(t_{k+i}))$, $i = 0, 1, \dots, N_h - 1$ comply with the input constraint of Eq. (32c) and the Lyapunov-based constraint of Eq. (32f) since Proposition 3 shows that the states predicted by the NODE model of Eq. (32b) remain inside $\Omega_{\rho_{sp}}$ under the controller $\Phi_{nn}(x)$. Hence, if $x(t) \in \Omega_{\hat{\rho}}$ for all times, the LMPC optimization problem of Eq. (32) is recursively feasible for all initial states $x_0 \in \Omega_{\hat{\rho}}$.

We will show that $\forall x_0 \in \Omega_{\hat{\rho}}$, the state of the closed-loop system of Eq. (1) under the LMPC scheme of Eq. (32) remains bounded in $\Omega_{\hat{\rho}}$ $\forall t$ and ultimately converges to a small neighborhood around the origin $\Omega_{\rho_{\min}}$ as described by Eq. (21b).

Assume $x(t_k) \in \Omega_{\hat{\rho}} \backslash \Omega_{\rho_{sp}}$ at time t_k. In this case, the constraint of Eq. (32e) is activated, and the control action u is computed to decrease the value of $\hat{V}(\hat{x})$ based on the predicted states obtained from the NODE model of Eq. (32b) over the next sampling period. Furthermore, Eq. (31) shows that, if the constraint of Eq. (32e) is met, then $\dot{\hat{V}}(x) \leq -\epsilon_w$ holds for $t \in [t_k, t_{k+1})$ when applying the control action $u^*(t_k)$ to the nonlinear system of Eq. (1). Consequently, the value of the Lyapunov function calculated using the state of the actual nonlinear system of Eq. (1), $\hat{V}(x)$, decreases within the following sampling period, implying that the closed-loop state can be driven into $\Omega_{\rho_{sp}}$ within a finite number of sampling periods.

Once the state enters $\Omega_{\rho_{sp}}$, the constraint of Eq. (32f) is activated to ensure that the predicted states of the NODE model of Eq. (32b) remain in $\Omega_{\rho_{sp}}$ throughout the prediction horizon. While there may exist a mismatch between the NODE model of Eq. (32b) and the nonlinear system of Eq. (1), based on the results of Proposition 3, we can guarantee that the state $x(t)$ of the nonlinear system of Eq. (1) remains bounded in $\Omega_{\rho_{\min}}$ $\forall t \in [t_k, t_{k+1})$ as characterized by Eq. (21b) if the state predicted by the NODE model of Eq. (32b) stays in $\Omega_{\rho_{sp}}$.

Thus, at the next sampling period $t = t_{k+1}$, if the state $x(t_{k+1})$ is still within $\Omega_{\rho_{sp}}$, then the constraint of Eq. (32f) ensures that the predicted state $\hat{x}$ of the NODE model of Eq. (32b) remains in $\Omega_{\rho_{sp}}$, thereby keeping the state x of the actual nonlinear system of Eq. (1) inside $\Omega_{\rho_{\min}}$. However, if the state exits $\Omega_{\rho_{sp}}$ such that $x(t_{k+1}) \in \Omega_{\rho_{\min}} \backslash \Omega_{\rho_{sp}}$, we can retrace the proof for $x(t_k) \in \Omega_{\hat{\rho}} \backslash \Omega_{\rho_{sp}}$ to show that the state will

Table 1
Parameters of the CSTR example.

$T_0 = 300$ K	$k_0 = 8.46 \times 10^6$ m^3/kmol h
$C_p = 0.231$ kJ/kg K	$\rho_L = 1000$ kg/m^3
$F = 5$ m^3/h	$E = 5 \times 10^4$ kJ/kmol
$R = 8.314$ kJ/kmol K	

once again be driven towards the origin since the constraint of Eq. (32e) will be triggered. This concludes the proof of the states of the actual nonlinear system of Eq. (1) being bounded under the sample-and-hold implementation of the controller $u = \Phi_{nn}(x)$ and being ultimately driven to a small neighborhood around the origin $\Omega_{\rho_{\min}}$ $\forall x_0 \in \Omega_{\hat{\rho}}$. □

5. Application of NODE-based model predictive control in chemical process

In this section, a chemical process involving a continuous stirred tank reactor (CSTR), which facilitates a reaction converting reactant A to product B, is utilized to demonstrate the development and application of NODE-based LMPC in a chemical process setting. Based on mass and energy balances, the following ODEs are used to describe the CSTR system:

$$\frac{\mathrm{d}C_A}{\mathrm{d}t} = \frac{F}{V}(C_{A0} - C_A) - k_0 \mathrm{e}^{\frac{-E}{RT}} C_A^2 \tag{33a}$$

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \frac{F}{V}(T_0 - T) + \frac{-\Delta H}{\rho_L C_p} k_0 \mathrm{e}^{\frac{-E}{RT}} C_A^2 + \frac{Q}{\rho_L C_p V} \tag{33b}$$

where C_{A0} is the concentration of reactant A in the feed flow, while T and C_A are the temperature and the concentration of A, respectively, in the CSTR. ρ_L and C_p represent the density and heat capacity of the liquid mixture in the CSTR, respectively, and are assumed to be constant. ΔH denotes the enthalpy of the reaction. The constant parameters are listed in Table 1.

The unstable steady state of the system at ($C_{As} = 1.95$ kmol/m^3, $T_s =$ 402 K) is selected as the set-point of this example, and the control objective of the LMPC is to maintain the state of the system around this set-point. The manipulated control variables in this example are the inlet concentration of reactant A and the heat duty of the coolant jacket of the CSTR, with the steady state control actions set to $C_{A0s} = 4$ kmol/m^3, $Q_s = 0$ kJ/h. The bounds of the manipulated control variables are $C_{A0} \in C_{A0s} \pm 3.5$ kmol/m^3 and $Q \in Q_s \pm 5 \times 10^5$ kJ/h.

5.1. Noise-free example

5.1.1. Data collection and preprocessing

Training data used to develop the NODE model is obtained by performing open-loop simulations. Specifically, the first step of the data collection is to define the open region D, which requires the design of a Lyapunov function $\hat{V}(x)$. In our work, the control Lyapunov function is designed to be $\hat{V}(x) = x^\top P x$, where P is the positive definite matrix $P = \begin{bmatrix} 1060 & 22 \\ 22 & 0.52 \end{bmatrix}$. Subsequently, a level set where $\rho = 375$ is selected to be the closed-loop stability region Ω_ρ, which is shown as the ellipse in Fig. 2. Since the desired region of operation is Ω_ρ, this is also chosen to be the sampling region for data collection.

After determining the sampling region Ω_ρ, numerous open-loop state trajectories are obtained by integrating Eq. (33) from randomly drawn initial states under random, constant control inputs using the explicit Euler method over a time span of 0.045 h. A very small step size (i.e., $h_c \ll 0.005$ h) is used in the explicit Euler method to generate the state trajectories, but since the process is assumed to have a sampling period of 0.005 h, the generated state trajectories are then evenly down-sampled to have a time interval of 0.005 h between data points. As a result, the training data set is made up of various state trajectories generated from a wide range of initial conditions and control inputs. Each trajectory in the training data set contains 10 samples or data points. Moreover, the data is generated in deviation form, $x = (C_A - C_{As}, T - T_s)$ and $u = (C_{A0} - C_{A0s}, Q - Q_s)$, such that the steady state is at the origin (i.e., $x_s = (0,0)$, $u_s = (0,0)$). Following this method, a training data set containing 1000 trajectories was collected and found to be enough to train the NODE model in this example.

Similar to other neural network models, data needs to be scaled prior to being used to train an NODE model. The scaling step is a part of data preprocessing. The MaxAbs scaler, which scales the data set by dividing each variable by the maximum absolute value of each variable in the data set (without any subtraction), respectively, is adopted in this work to maintain the steady-state of the scaled data set at the origin. As a result, the training data is scaled to a range between −1 and 1. Besides scaling the data, the training data set is split into two parts, such that 80% of the trajectories are used to train the model and the rest are used to validate the model performance. Specifically, the model weights are updated based on the loss with respect to 80% of all trajectories and the remaining 20%, usually named the validation set, is not included in the calculation of the loss for the weight update but are used to compute the validation loss, which is an important metric to ensure that the model does not overfit the training data. Lastly, there are additional 100 trajectories that are reserved for testing the trained model's performance, which is usually called the test set, and these are generated in the same manner as the training data.

The data preprocessing step, including scaling and splitting the data, is handled using Scikit-learn, a popular Python-based machine learning package. Conventionally, in most standard machine learning procedures, the inputs of the neural network model and the reference/output data are first augmented into two different tensors. After that, the Scikit-learn scaler function is applied to fit and transform the input and output tensors, respectively. However, this scaling strategy will cause a mismatch between the model input and first time step of the output trajectory. Specifically, the prediction of our NODE model is designed to include the initial value of the state, which is provided by the input tensor during the model training. Therefore, the initial value of each trajectory in the output tensor corresponds to t_0 and should be identical to the states values in the input tensor. If the tensors are scaled separately, scaling factors (e.g., mean, maximum absolute value, etc.) will be different for the input and the output tensors. This is because, using time-series data as an example, data in the output tensor is evolved in time from the data in the input tensor, which is very likely to yield different statistics for the two tensors. If the different means are subtracted from the input and output tensors, the initial value of each trajectory corresponding to t_0 will be different for each tensor. To avoid this mismatch, the same scaling factor should be applied to the variables representing the same physical quantities. Lastly, to ensure no information leakage occurs during the model building process, the maximum absolute value used to scale the data should be based on only the 80% of the training data set.

Remark 5. In practice, the knowledge of the operating region can be obtained from the process design and operational purpose as well as past experience of operating the process. For example, the operating region of the process should be determined from the previous operation experience if the objective is to update an existing control system or from process design and simulation using reliable software used in industry, such as Aspen Hysys or Aveva PRO/II, if the objective is to develop and implement a control system to a new process. The training data in those cases will be collected from the past operation or process simulation, respectively.

5.1.2. NODE training

In this subsection, the details of the NODE model development using PyTorch are provided. Specifically, the core model of the NODE is chosen to be an FNN model to map neural network inputs to the hidden state of the NODE model. The core model has two hidden

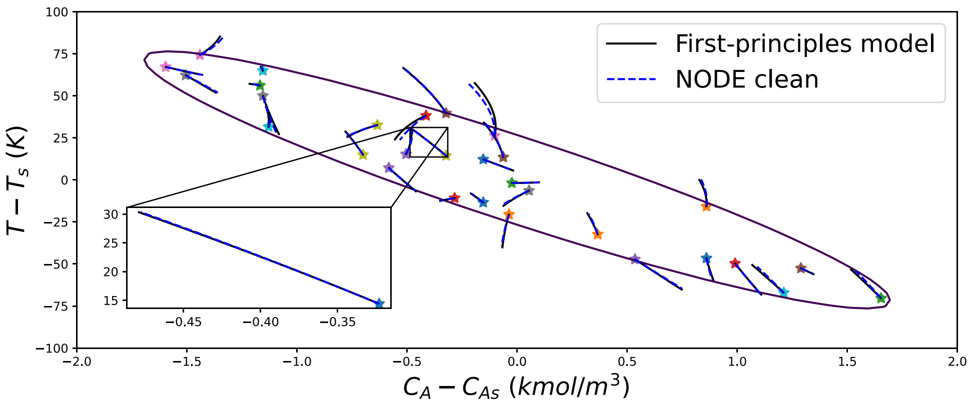


Fig. 2. Open-loop state-space trajectories of the CSTR given by the first-principle model (black line) and the trained NODE model (blue dash). The purple ellipse denotes the boundary of the pre-defined stability region, Ω_ρ, of the CSTR system. Star icons are the initial state pairs randomly drawn within the stability region.

layers and each hidden layer contains 64 neurons activated by the hyperbolic tangent (tanh) function. Dupont et al. (2019) suggested that the NODE model should be differentiable everywhere, but activation functions like ReLU, which are not, have been used in NODE-based research (Dupont et al., 2019; Kidger et al., 2020). Nevertheless, to avoid any potential failure caused by the lack of differentiability of the activation function, tanh is used as the activation function in our core model. The Adaptive Moment Estimation (ADAM) optimizer is used to update the weight matrices in the core model. Mini-batches are used for the gradient descent method with a batch size of 32 trajectories. Lastly, an explicit Euler solver is used as the ODE solver in the NODE model.

Hyperparameter tuning is a critical step in neural network development and is done via a coarse, exploratory search followed by manual fine-tuning in this work. Specifically, we observed that using two hidden layers in the core model can significantly improve the learning ability of the NODE model but having more than two hidden layers did not provide any significant improvement despite the uptick in computational resource usage. For the number of neurons to be used in the hidden layers, we applied a two-dimensional grid search using 8, 16, 32, 64, and 128 neurons for each hidden layer and assessed the model performance at each combination. It was found that using more than 64 neurons did not significantly improve the model performance. Therefore, we used 64 neurons in our hidden layer. The NODE model is trained with 300 epochs, and the testing loss of the trained model is 3.1×10^{-4}. Furthermore, the mean absolute errors were 0.015 $\mathrm{kmol/m^3}$ and 0.8 K for the prediction of C_A and T, respectively.

Remark 6. The model obtained using the aforementioned hyperparameter tuning strategy may not be the most accurate possible model. However, due to the low dimension of our system and the complexity of the neural network structure, further fine-tuning of the hyperparameter is not expected to significantly improve the model performance. We note that finding the best possible model or the most effective hyperparameter tuning strategy are not the major objectives of this study.

Remark 7. Detailing the coding implementation of the NODE is not the objective of this paper. However, since the NODE is a recently proposed model, such that its training is not fully supported in the commonly used machine learning APIs such as PyTorch and TensorFlow, we outline some key steps of our implementation. An official NODE python package, `torchdiffeq`, is provided in Chen et al. (2018). In our study, the intermediate steps of the data sequence are important when evaluating the loss of the model. Therefore, the integral of the adjoint state $\mathbf{a}(t)$ is updated at each intermediate observation during the backpropagation. We developed our code based on an open-source GitHub project (Surtsukov, 2019). Modifications needed to be added to the ODE solver and the backpropagation function to correctly handle the control actions in the neural network input.

5.1.3. NODE-based LMPC performance

After training the NODE model, we conduct open-loop simulations using the NODE model and benchmark it against the first-principles (FP) equation to evaluate its prediction accuracy. Specifically, open-loop simulations are conducted under fixed control inputs over two sampling periods for both the NODE model and the FP model and the state trajectories compared. The simulations start from randomly selected initial conditions inside the stability region Ω_ρ. Subsequently, a random, constant control input is applied to the process for two sampling periods. Fig. 2 depicts the open-loop trajectories for both the NODE model and the FP model of Eq. (33) under identical input signals as described. The close agreement between the state trajectories predicted by both models throughout the two sampling periods supports the fact that the NODE model prediction has been trained to a high level of accuracy.

After ensuring the accuracy of the NODE model, closed-loop simulations under the LMPC of Eq. (32) using the NODE process model are conducted. The objective function of the LMPC is defined to be $L(x,u) = x^\top \bar{Q} x + u^\top R u$, where $\bar{Q}$ and R are the parameter matrices of the state and input penalty terms in the objective function, respectively. Fig. 3 illustrates the state and input profiles of the CSTR in closed-loop under the NODE-based LMPC, where the process state is brought from a randomly drawn initial state in deviation form $x_0 = (C_A - C_{As} = 1.6\ \mathrm{kmol/m^3}, T - T_s = -64.5\ \mathrm{K})$ to the steady-state set-point and maintained within $\Omega_{\rho_{sp}}$. Furthermore, the closed-loop performance of the NODE-based LMPC and the FP-based LMPC are found to be very similar, indicating that the NODE-based LMPC can perform as well as the FP-based LMPC due to the high accuracy of the NODE model. Finally, Fig. 4 shows closed-loop state-space trajectories from several random initial conditions in Ω_ρ under the NODE-based LMPC, all of which are found to be successfully stabilized around the set-point.

Remark 8. Although changing the type of ODE solver in the NODE is not recommended, the parameters used in the ODE solver can be changed based on the model performance without any negative impact. For example, the integration time step in the explicit Euler solver used in this study was tuned when developing the LMPC to balance prediction accuracy with computational cost. Besides this, in the PyTorch framework, additional mathematical operations, such as a lambda layer, can be added to the core model after training. Specifically, a lambda layer can be designed to ensure that the output of the core model is equal to zero when the neural network inputs are all at the steady-state. By adding such a lambda layer after training, the NODE model can be easily programmed to provide correct information

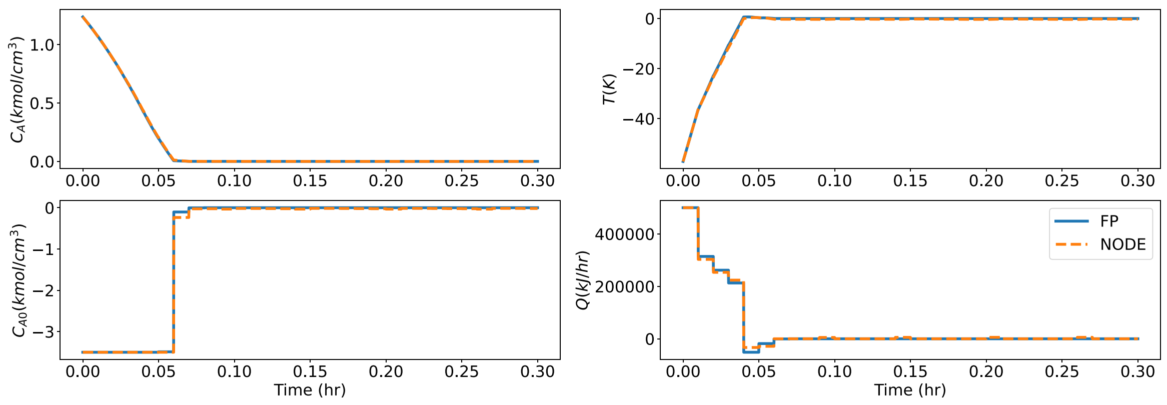


Fig. 3. State and input profiles in deviation form for the CSTR under the LMPC using the first-principles process model (blue line) and the NODE process model (orange line).

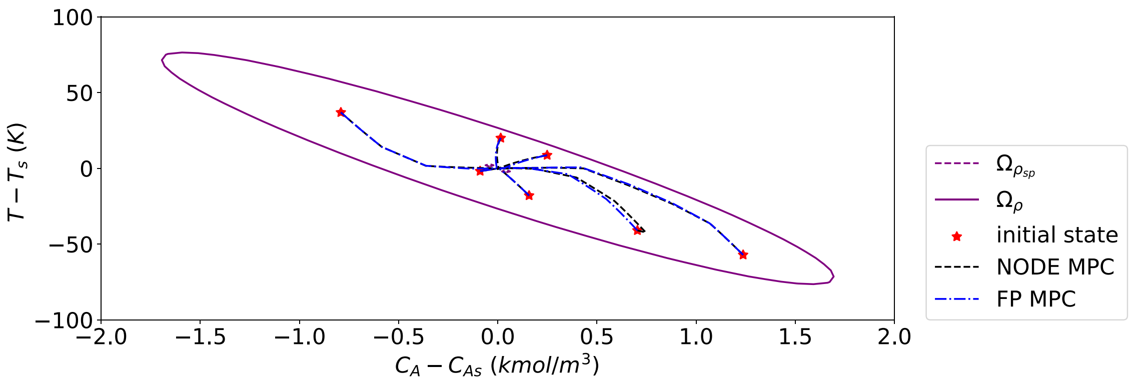


Fig. 4. NODE-based LMPC performance in closed-loop simulation. The closed-loop simulation used the NODE-based LMPC to stabilize the CSTR from various initial conditions represented by red stars. The closed-loop state trajectories under the LMPC are demonstrated in black dashed line and compared with the state trajectories under an LMPC based on FP equation. The NODE-based LMPC successfully bring the state to the desired region $\Omega_{\rho_{sp}}$ and having a very similar performance comparing to the LMPC designed using FP equation.

for some critical well-known process conditions without harming the training process (i.e., including the lambda layer during the training process may affect the gradient descent step of the neural network model training). In this work, it is found that adding a lambda layer to ensure zero output at the steady state did not have a significant effect on the LMPC performance. Therefore, the simulations in this work are carried out without involving any lambda layer.

Remark 9. While the operating region (closed-loop stability region) is dependent on the form of the Lyapunov function, the choice of a quadratic Lyapunov function is motivated by several factors. Firstly, in a quadratic formulation, the sign definiteness of the Lyapunov function $V = x^\top Px$ follows directly from the sign definiteness of the matrix P, i.e., if P is positive (semi-)definite, then V is also positive (semi-)definite. Secondly, it cannot be ascertained that the operating region obtained using a quadratic Lyapunov function is small with respect to the entire stability region, say X, as it depends heavily on the system itself. Even in the case that the characterized region is a small subset of X, this is not a drawback since there is no better alternative for the general class of nonlinear systems considered to explicitly characterize the closed-loop system stability region. Finally, chemical processes do not generally exhibit any specific mathematical structures that can be exploited to design a non-quadratic, application-specific Lyapunov function. This is different from many electrical or mechanical systems, where, for example, a total energy of the system function is a natural candidate for the Lyapunov function. Based on the specific mechanical system being studied, the energy function can be obtained using involved physics and domain knowledge, and will likely be the best candidate for the Lyapunov function. For a simple pendulum, for example, the energy function is combination of a quadratic part and a trigonometric part (sum of kinetic and potential energy), which is a natural Lyapunov function for the pendulum system but is highly specific to this system and cannot generally be applied to another system; certainly not to a chemical process model. In fact, even the addition of friction in the pendulum requires the Lyapunov function to be modified accordingly to retain negative definiteness of its time-derivative. Hence, in our chemical process example, we use a quadratic Lyapunov function that can be obtained for many (but not necessarily all) process systems. This is especially important and meaningful because we work from a data-based paradigm, assuming zero *a priori* knowledge of the physics of the system. Once the quadratic form of the Lyapunov function is selected, the next step is the design of the P matrix, which is carried out based on the stabilizing controller $\Phi_{nn}(x)$ (e.g., a Sontag Lyapunov-based controller or P-controller). We first obtain closed-loop trajectories of the state under the stabilizing controller from various initial conditions and record both the states and applied inputs as functions of time, while the state is driven to the origin. At every point along these recorded trajectories, we then calculate the value of the time-derivative of the Lyapunov-function, $\dot{V}$, for many random P matrices (alternately, possibly more rigorously, one may linearize the nonlinear system around the steady-state of interest and solve the corresponding Riccati equation using the A matrix from linearization to find a valid P matrix or an initial estimate of a valid P matrix). A P matrix that renders $\dot{V}$ negative throughout the trajectories is considered a valid P matrix, and the P matrix that maximizes the size of the operating region is selected. Using this procedure, the P matrix was found in this work.

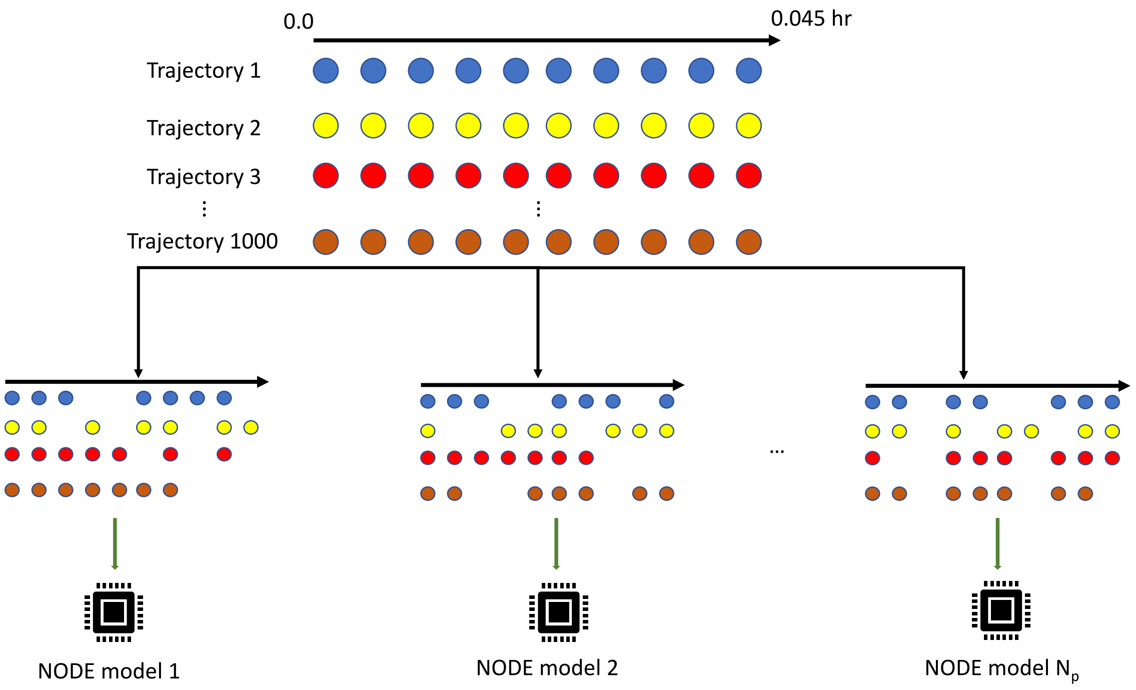


Fig. 5. The workflow of using subsampling method with a subsampling factor $p = 0.3$ to develop neural ordinary differential equation (NODE) models. By using the subsampling method, different training data can be created in each attempt, which allows for training of a new model. N_p in this figure is used to denotes the number of NODE models to generate in the workflow.

5.2. Noisy data example: Gaussian noise

After showing the NODE-based LMPC is capable of controlling a noise-free system, we further investigate if a NODE-based LMPC can be used in the case of noisy data, which is more practical in an industrial setting. For the first scenario, we assumed that the process noise follows a Gaussian distribution. The process noise follows the distribution, $v \sim \mathcal{N}(0, \sigma^2)$, and is added to the clean data set reported in Section 5.1.1 to generate the noisy data set. Three noisy data sets corrupted by increasing strength of process noise are generated for this study and the details of the noise levels are listed in Table 2. Subsequently, each noisy data set is used to train an NODE model having the same structure as the one developed using the clean data set. Table 2 also includes the testing loss of the NODE model under each noise level.

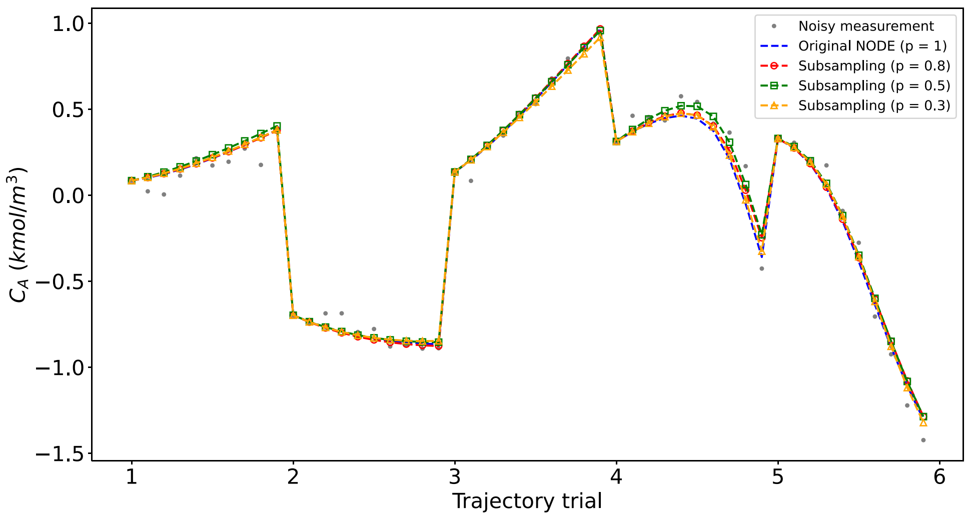
The testing loss for the model developed with noisy data is calculated using the reference data also corrupted by the same strength of noise (e.g., if the NODE model is developed using the data set corrupted by the weak level of noise, the testing data used to calculate its loss is also corrupted by the weak level of noise), instead of comparing with the clean data. This is because a clean data set is not available in a practical system with measurement noise. Therefore, the models are compared in the sense of how well they can fit the available data. As a result, the NODE model trained with the weak noise has a slightly higher loss than the one trained with clean data, and the loss increases with increasing noise levels, which demonstrates the negative impact of measurement noise on the model training.

To account for noisy measurements, the subsampling method, following the workflow shown in Fig. 5, is adopted in this study. We introduce a tuning parameter for the subsampling method, named subsampling factor p, which denotes the percentage of data points randomly selected to be kept in the data set while dropping out the rest of the data. Specifically, the number of trajectories in the training set will remain the same, but some data points in each trajectory will be dropped out randomly to match the desired p value. Additionally, since the data points to be dropped are randomly selected, redoing the subsampling with the same p value will result in a different training data set each time, which leads to training a different NODE model each time. Three p values (i.e., 30%, 50%, 80%) are used to subsample the data, and 5 NODE models are trained for each value of p. Finally, only the lowest training loss among the 5 models is reported in Table 2, and the corresponding model is used to develop an LMPC for the next step. The training loss in Table 2 shows that using the subsampling method does not have a significant impact on the NODE model performance. Open-loop simulations are further conducted to evaluate the model performance. Fig. 6 demonstrates the performance of the NODE models for different values of p by showing the predicted state trajectories, which are overlapping with each other.

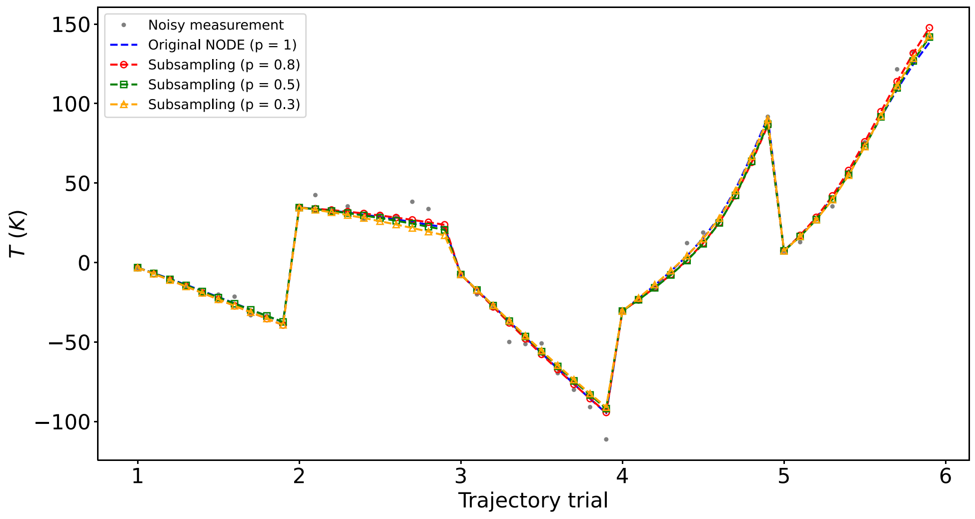
Table 2
Training loss of NODE model using Gaussian noisy data.

	Weak noise	Medium noise	Strong noise
	$\sigma_{C_A} = 0.05$ kmol/m^3 $\sigma_T = 5$ K	$\sigma_{C_A} = 0.15$ kmol/m^3 $\sigma_T = 15$ K	$\sigma_{C_A} = 0.25$ kmol/m^3 $\sigma_T = 25$ K
p		Mean Squared Error (MSE)	
1	0.0076	0.0290	0.0365
0.8	0.0074	0.0320	0.0370
0.5	0.0069	0.0270	0.0370
0.3	0.0068	0.0300	0.0500

Finally, closed-loop simulations are conducted to evaluate the performance of the NODE-based LMPC using different subsampling factors. Specifically, during the closed-loop simulations, a non-zero initial state is first randomly drawn from the stability region $\Omega_\rho \backslash \Omega_{\rho_{sp}}$, following which the respective LMPC is used to bring the process to the set point, which is the origin of the state space. Specifically, the closed-loop simulation is run for a duration of 0.3 h with a sampling time of 0.01 h. Therefore, there are 30 state feedback measurements used by the LMPC in the simulation, which are all corrupted by noise. To ensure a fair comparison between each LMPC, the noise added to the feedback measurements must be consistent. Thus, the sensor noise is only sampled once from a Gaussian distribution and then saved in order to be used in all the closed-loop simulations. Fig. 7 compares the performance of the NODE-based LMPCs developed with four p values in the presence of weak noise (as defined in Table 2). It is observed that



(a)



(b)

Fig. 6. Open-loop simulation results for (a) concentration of reactant A in the CSTR ($C_A - C_{As}$) and (b) temperature of the CSTR ($T - T_s$) using the NODE model trained with Gaussian noisy data.

all the LMPCs successfully stabilized the process from the various initial conditions. However, the clean state may not remain in the designed stability region $\Omega_{\rho_{sp}}$ under the effect of noise. Proposition 4 in Wu et al. (2019) derived how the region of ultimate boundary, $\Omega_{\rho_{min}}$, increases as the disturbance bound and sampling period increase. Based on the closed-loop simulations in Fig. 7, by using $\Omega_{\rho_{sp}} = 2$, we found the region of ultimate boundary expanded to $\Omega_{\rho_{\min}} = 60$ under weak Gaussian noise.

Finally, the LMPC performance is quantified by calculating the integral of the LMPC cost function (Eq. (32)) over the simulation duration, i.e., $\int_{t=0}^{t=0.3h} L(x(\tau), u(\tau))\, d\tau$. The quantified loss shows that the subsampling model with $p = 0.8$ has better performance than the NODE model without subsampling in all five closed-loop simulations. In four out of five simulations, the NODE model with $p = 0.8$ gave an LMPC cost function value approximately 1% lower than the model without subsampling, and in the fifth closed-loop simulation, the NODE model with $p = 0.8$ reduced the LMPC cost function by 15% compared to the model with $p = 1$. The NODE models with $p = 0.5$ and $p = 0.3$ did not have a lower LMPC cost function in all five simulations compared to the model without subsampling, but the differences were within 3%. Therefore, the improvement gains from using subsampling is minor in the case of Gaussian noise and possibly even due to numerical/experimental differences between runs.

5.3. Noisy data example: Non-Gaussian noise

Although the assumption of Gaussian noise is very useful in many applications, non-Gaussian noise is another commonly observed noise distribution in the chemical sector. In this section, we investigate how the NODE model performs under non-Gaussian noise. First, non-Gaussian noise is extracted from an industrial data provided by Aspentech. Fig. 8 shows the non-Gaussian noise distribution for C_A and T, respectively. Next, to generate the non-Gaussian noisy reactor data set, the noise value sampled from the above distribution is added to the clean data set following a similar process as described in Section 5.2. Different levels of noise are added to the clean data for a comprehensive

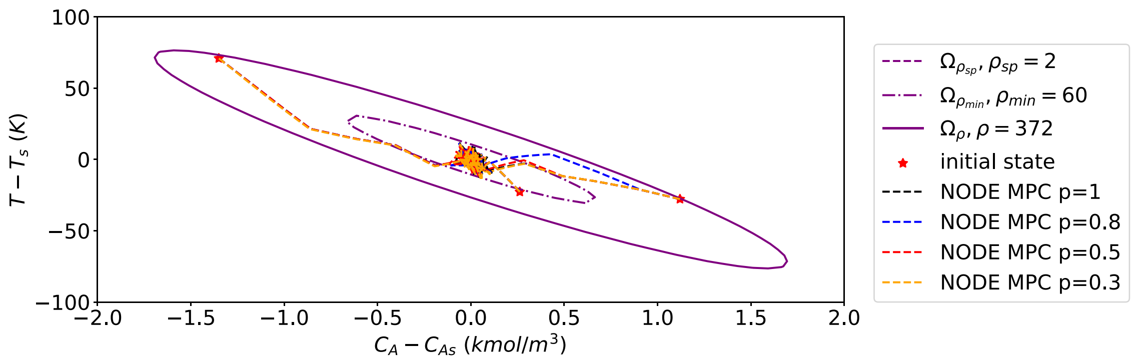


Fig. 7. Closed-loop simulation results under weak Gaussian noise using LMPC based on NODE model developed with different subsampling factors. Red stars represent the initial condition for of each closed-loop simulation and the black, blue, red, and orange dash line are the state trajectories controlled by LMPC based on the NODE model developed with subsampling factor $p = 1, 0.8, 0.5$ and 0.3 respectively.

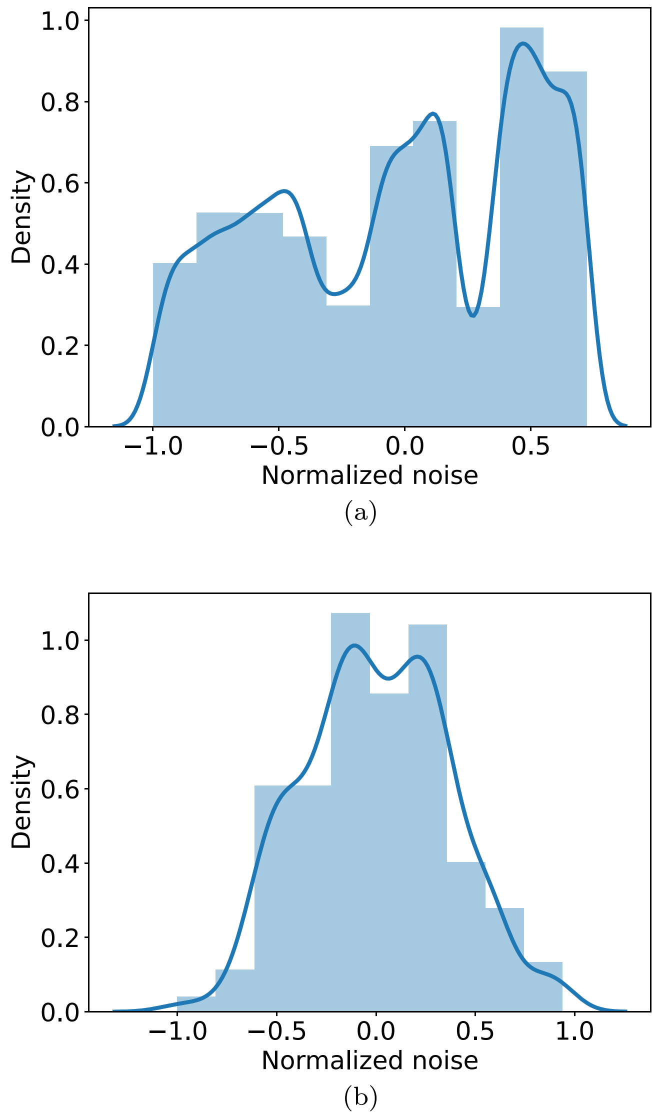


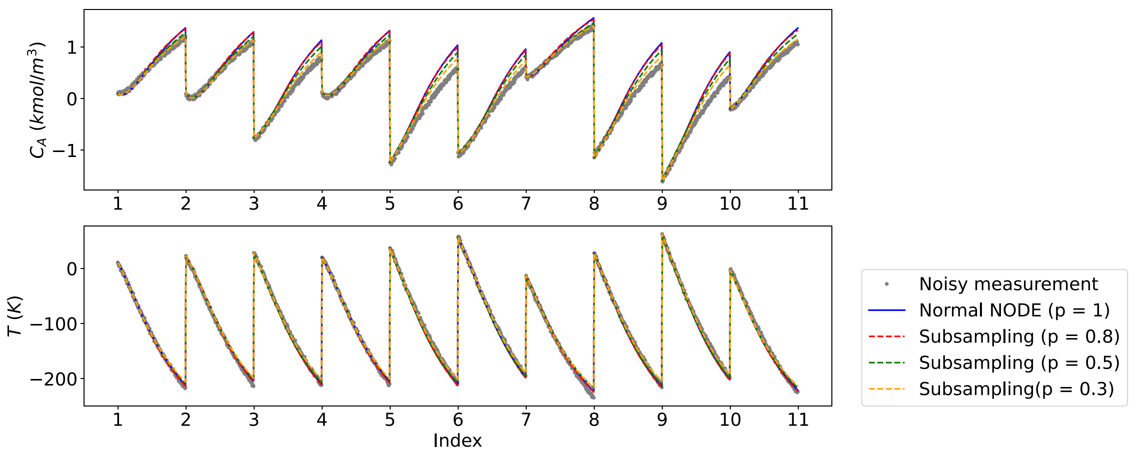
Fig. 8. Non-Gaussian noise distribution for (a) concentration of reactant A in the CSTR (C_A) and (b) temperature (T) of the CSTR. The non-Gaussian noise is scaled to sit between -1.0 to 1.0 and is multiplied by the maximum noise parameter depending on the strength of the noise before adding it to the clean data.

investigation. The level of noise, **O**, is defined by the maximum value of noise that can be added to the clean data set. Specifically, taking the weak non-Gaussian noise as an example, the maximum noise that can be added to the clean data set is 0.05 kmol/m^3 and 5 K for C_A and T, respectively. To add non-Gaussian noise to a single data point in the clean data set, a noise value from -1.0 to 1.0 is first randomly sampled from the non-Gaussian distribution and then multiplied by the maximum noise value of each variable before being added to the respective clean data point.

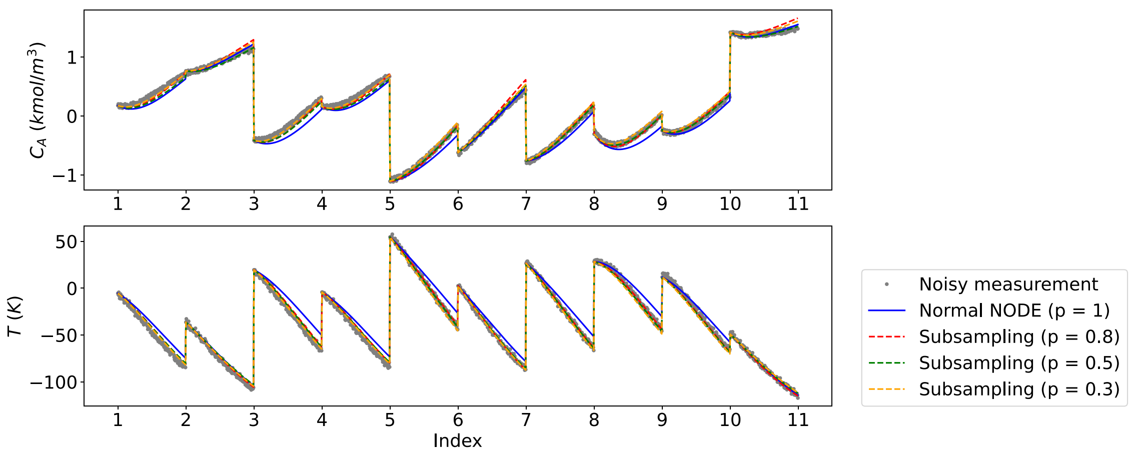
Subsampling with the same range of p values are used to reduce the effect of non-Gaussian noise. The training loss for each value of p and noise level is summarized in Table 3. The maximum noise value of temperature is fixed at 5 K because having a measurement noise of 5 degrees is practically very significant. A sensor that gives a bigger measurement error can be considered to be a dysfunctional sensor and requires maintenance or replacement. On the other hand, the concentration sensor may have more perturbation in its measurement than 0.05 kmol/m^3, so a noise level of 0.1 kmol/m^3 is included in the study as the strong noise level. Based on the training loss values listed in Table 3, the subsampling method successfully improves the model performance when the training data set is corrupted with non-Gaussian noise. The best models to fit the noisy data are the models trained with subsampling factors of $p \leq 0.5$, which reduce the training loss by 28% and 26% for weak and strong non-Gaussian noise, respectively, but the loss for models with $p = 0.5$ and $p = 0.3$ are, in fact, very similar and both show improvement over models with larger p values. The open-loop simulations shown in Fig. 9 further demonstrates the model improvements by using the subsampling method. Specifically, for weak non-Gaussian noise (Fig. 9(a)), the NODE models trained with subsampling method predict C_A better compared to the model without subsampling, but there is no significant difference in terms of the temperature prediction, although this may be due to the small room for improvement for the case of the temperature prediction. For stronger noise (Fig. 9(b)), using the subsampling method improves the predictive performance of the model for both states. Please note that we only change the strength of the C_A noise, but since the temperature and concentration of the CSTR are coupled, increasing the noise level for C_A will also affect the model prediction of the temperature. Closed-loop simulations similar to the Gaussian case are used to evaluate the LMPC performance under strong noise, and the results are shown in Fig. 10. All the NODE-based LMPCs successfully stabilized the process from the non-zero initial state by ultimately maintaining the states within $\Omega_{\rho_{\min}} = 10$. By quantifying the loss over the simulation duration, it is found that using a subsampling factor of $p = 0.3$ gives better LMPC performance compared to the LMPC without subsampling. The largest reduction in the LMPC cost function via subsampling was found to be 34% in closed-loop simulations.

6. Conclusion

In this work, we developed a Lyapunov-based model predictive control system using a neural ordinary differential equation (NODE) model



(a)



(b)

Fig. 9. Open-loop simulation results using NODE model training with (a) weak and (b) strong non-Gaussian noisy data.

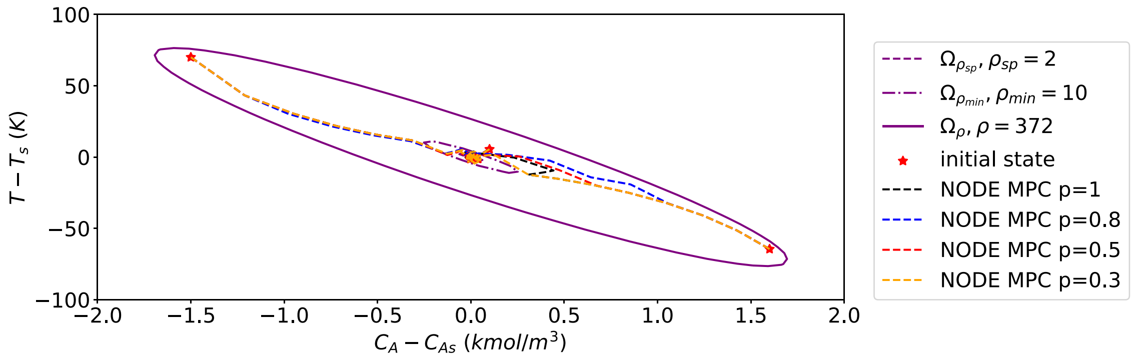


Fig. 10. Closed-loop simulation results under strong Gaussian noise using LMPC based on NODE model developed with different subsampling factors. Red stars represent the initial condition for of each closed-loop simulation and the black, blue, red, and orange dash line are the state trajectories controlled by LMPC based on the NODE model developed with subsampling factor $p = 1, 0.8, 0.5$ and 0.3 respectively.

constructed from process data as the process model, and elucidated the training process of an NODE model and its use in LMPC using a simulated data set. The results demonstrated the ability of the NODE to provide a continuous prediction for the nonlinear system and to be used as the process model in an LMPC. Specifically, the core model of the NODE can capture the time-derivatives of the states, which can be considered an additional method to compute the derivative information other than numerical approximation methods. The state derivatives predicted by the NODE model were used in the LMPC to enforce the constraints and optimize the control objective. Moreover, the NODE model imposes fewer restrictions on the structure of the training data than RNN models, which permits the use of the subsampling method to account for noisy data. NODE models developed with different subsampling factors were used to account for Gaussian and non-Gaussian measurement noise. It was found that using subsampling could not significantly reduce the training loss under Gaussian noise, but could reduce the LMPC cost function in closed-loop simulations by up to 15%. For non-Gaussian noise, using subsampling reduced the training loss by up to 24% and 26% in the case of the weak and strong noises considered in this work, respectively. As for the closed-loop simulations, using subsampling improved the closed-loop performance of the LMPC under non-Gaussian noise by up to 34% in terms of the LMPC cost function.

Table 3
Training loss of NODE model using non-Gaussian noisy data.

	Weak noise	Strong noise
	$\mathbf{O}_{C_A} = 0.05$ kmol/m^3, $\mathbf{O}_T = 5$ K	$\mathbf{O}_{C_A} = 0.10$ kmol/m^3, $\mathbf{O}_T = 5$ K
p	Mean Squared Error (MSE)	
1	0.0025	0.0031
0.8	0.0024	0.0026
0.5	0.0018	0.0024
0.3	0.0019	0.0023

CRediT authorship contribution statement

Junwei Luo: Conceptualization, Methodology, Software, Manuscript writing. **Fahim Abdullah:** Conceptualization, Methodology, Manuscript writing. **Panagiotis D. Christofides:** Manuscript reviewing and editing.

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.

Data availability

Data will be made available on request.

Acknowledgments

Financial support from the National Science Foundation and the Department of Energy is gratefully acknowledged.

References

Abdullah, F., Alhajeri, M.S., Christofides, P.D., 2022a. Modeling and control of nonlinear processes using sparse identification: Using dropout to handle noisy data. Ind. Eng. Chem. Res. 61 (49), 17976–17992.

Abdullah, F., Christofides, P.D., 2023. Data-based modeling and control of nonlinear process systems using sparse identification: An overview of recent results. Comput. Chem. Eng. 174, 108247.

Abdullah, F., Wu, Z., Christofides, P.D., 2021. Sparse-identification-based model predictive control of nonlinear two-time-scale processes. Comput. Chem. Eng. 153, 107411.

Abdullah, F., Wu, Z., Christofides, P.D., 2022b. Handling noisy data in sparse model identification using subsampling and co-teaching. Comput. Chem. Eng. 157, 107628.

Benattia, S.E., Tebbani, S., Dumur, D., 2016. A linearized robust model predictive control applied to bioprocess. In: Proceedings of 55th Conference on Decision and Control. Las Vegas, Nevada, pp. 4046–4052.

Billings, S.A., 2013. Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains. John Wiley & Sons.

Bradley, W., Boukouvala, F., 2021. Two-stage approach to parameter estimation of differential equations using neural ODEs. Ind. Eng. Chem. Res. 60, 16330–16344.

Brüdigam, T., Olbrich, M., Wollherr, D., Leibold, M., 2021. Stochastic model predictive control with a safety guarantee for automated driving. IEEE Trans. Intell. Veh.

Chee, K.Y., Hsieh, M.A., Matni, N., 2023. Learning-enhanced nonlinear model predictive control using knowledge-based neural ordinary differential equations and deep ensembles. In: The 5th Annual Learning for Dynamics and Control Conference. Philadelphia, Pennsylvania, PMLR, pp. 1125–1137.

Chen, R.T., Rubanova, Y., Bettencourt, J., Duvenaud, D.K., 2018. Neural ordinary differential equations. Adv. Neural Inf. Process. Syst. 31.

Cilimkovic, M., 2015. Neural Networks and Back Propagation Algorithm, Vol. 15, No. 1. Institute of Technology Blanchardstown, Blanchardstown Road North Dublin.

Çıtmacı, B., Luo, J., Jang, J.B., Canuso, V., Richard, D., Ren, Y.M., Morales-Guio, C.G., Christofides, P.D., 2022. Machine learning-based ethylene concentration estimation, real-time optimization and feedback control of an experimental electrochemical reactor. Chem. Eng. Res. Des. 185, 87–107.

Dai, W., Song, Y., Wang, D., 2023. A subsampling method for regression problems based on minimum energy criterion. Technometrics 65, 192–205.

Dongare, A., Kharde, R., Kachare, A.D., et al., 2012. Introduction to artificial neural network. Int. J. Eng. Innov. Technol. (IJEIT) 2 (1), 189–194.

Dupont, E., Doucet, A., Teh, Y.W., 2019. Augmented neural ODEs. Adv. Neural Inf. Process. Syst. 32.

Errico, R.M., 1997. What is an adjoint model? Bull. Am. Meteorol. Soc. 78 (11), 2577–2592.

Giesl, P., Hafstein, S., 2015. Review on computational methods for Lyapunov functions. Discrete Contin. Dyn. Syst. Ser. B 20 (8), 2291–2331.

Goyal, P., Benner, P., 2022. Neural ODEs with irregular and noisy data. arXiv preprint arXiv:2205.09479.

Grosman, B., Lewin, D.R., 2005. Automatic generation of Lyapunov functions using genetic programming. IFAC Proc. Vol. 38 (1), 75–80.

Han, L., Yu, C., Xiao, K., Zhao, X., 2019. A new method of mixed gas identification based on a convolutional neural network for time series classification. Sensors 19 (9), 1960.

Hansen, C.D., Johnson, C.R., 2011. Visualization Handbook. Elsevier, Burlington.

Hewing, L., Kabzan, J., Zeilinger, M.N., 2019. Cautious model predictive control using Gaussian process regression. IEEE Trans. Control Syst. Technol. 28, 2736–2743.

Hornik, K., 1991. Approximation capabilities of multilayer feedforward networks. Neural Netw. 4 (2), 251–257.

Hornik, K., Stinchcombe, M., White, H., 1990. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. Neural Netw. 3 (5), 551–560.

Huys, Q.J., Paninski, L., 2009. Smoothing of, and parameter estimation from, noisy biophysical recordings. PLoS Comput. Biol. 5, e1000379.

Ji, J., Khajepour, A., Melek, W.W., Huang, Y., 2016. Path planning and tracking for vehicle collision avoidance based on model predictive control with multiconstraints. IEEE Trans. Veh. Technol. 66, 952–964.

Kidger, P., Morrill, J., Foster, J., Lyons, T., 2020. Neural controlled differential equations for irregular time series. Adv. Neural Inf. Process. Syst. 33, 6696–6707.

Kittisupakorn, P., Thitiyasook, P., Hussain, M.A., Daosud, W., 2009. Neural network based model predictive control for a steel pickling process. J. Process Control 19 (4), 579–590.

Lai, Z., Mylonas, C., Nagarajaiah, S., Chatzi, E., 2021. Structural identification with physics-informed neural ordinary differential equations. J. Sound Vib. 508, 116196.

Liao, R., Chan, C.W., Hromek, J., Huang, G.H., He, L., 2008. Fuzzy logic control for a petroleum separation process. Eng. Appl. Artif. Intell. 21, 835–845.

Lin, Y., Sontag, E.D., 1991. A universal formula for stabilization with bounded controls. Systems Control Lett. 16 (6), 393–397.

Liu, X., Xiao, T., Si, S., Cao, Q., Kumar, S., Hsieh, C.-J., 2019. Neural SDE: Stabilizing neural ODE networks with stochastic noise. arXiv preprint arXiv:1906.02355.

Lötsch, J., Malkusch, S., Ultsch, A., 2021. Optimal distribution-preserving downsampling of large biomedical data sets (opdisdownsampling). PLoS One 16, e0255838.

Luo, J., Canuso, V., Jang, J.B., Wu, Z., Morales-Guio, C.G., Christofides, P.D., 2022. Machine learning-based operational modeling of an electrochemical reactor: Handling data variability and improving empirical models. Ind. Eng. Chem. Res. 61, 8399–8410.

Mhaskar, P., El-Farra, N.H., Christofides, P.D., 2005. Predictive control of switched nonlinear systems with scheduled mode transitions. IEEE Trans. Automat. Control 50 (11), 1670–1680.

Mhaskar, P., El-Farra, N.H., Christofides, P.D., 2006. Stabilization of nonlinear systems with state and control constraints using Lyapunov-based predictive control. Systems Control Lett. 55, 650–659.

Mohanty, S., 2009. Artificial neural network based system identification and model predictive control of a flotation column. J. Process Control 19, 991–999.

Morari, M., Lee, J.H., 1999. Model predictive control: past, present and future. Comput. Chem. Eng. 23, 667–682.

Nian, R., Liu, J., Huang, B., 2020. A review on reinforcement learning: Introduction and applications in industrial process control. Comput. Chem. Eng. 139, 106886.

Osofisan, P., Obafaiye, O., 2007. Fuzzy logic modeling of the fluidized catalytic cracking unit of a petrochemical refinery. Pac. J. Sci. Technol. 8, 59–67.

Pontryagin, L.S., 1987. Mathematical Theory of Optimal Processes. CRC Press, New York.

Raffo, G.V., Gomes, G.K., Normey-Rico, J.E., Kelber, C.R., Becker, L.B., 2009. A predictive controller for autonomous vehicle path tracking. IEEE Trans. Intell. Transp. Syst. 10, 92–102.

Ren, Y.M., Alhajeri, M.S., Luo, J., Chen, S., Abdullah, F., Wu, Z., Christofides, P.D., 2022. A tutorial review of neural network modeling approaches for model predictive control. Comput. Chem. Eng. 165, 107956.

Rohani, S., Haeri, M., Wood, H., 1999. Modeling and control of a continuous crystallization process Part 2. model predictive control. Comput. Chem. Eng. 23 (3), 279–286.

Sarmasti Emami, M.R., 2019. Fuzzy logic applications in chemical processes. J. Math. Comput. Sci 1, 339–348.

Surtsukov, M., 2019. Neural ODEs. GitHub Repository, GitHub, https://github.com/msurtsukov/neural-ode.

Tom, M., Yun, S., Wang, H., Ou, F., Orkoulas, G., Christofides, P.D., 2022. Machine learning-based run-to-run control of a spatial thermal atomic layer etching reactor. Comput. Chem. Eng. 168, 108044.

Tran, G., Ward, R., 2017. Exact recovery of chaotic systems from highly corrupted data. Multiscale Model. Simul. 15 (3), 1108–1129.

Wong, W.C., Chee, E., Li, J., Wang, X., 2018. Recurrent neural network-based model predictive control for continuous pharmaceutical manufacturing. Mathematics 6 (11), 242.

Wu, Z., Rincon, D., Luo, J., Christofides, P.D., 2021. Machine learning modeling and predictive control of nonlinear processes using noisy data. AIChE J. 67 (4), e17164.

Wu, Z., Tran, A., Rincon, D., Christofides, P.D., 2019. Machine learning-based predictive control of nonlinear processes. Part I: theory. AIChE J. 65 (11), e16729.

Xi, X.C., Poo, A.N., Chou, S.K., 2007. Support vector regression model predictive control on a HVAC plant. Control Eng. Pract. 15, 897–908.

Xiao, T., Wu, Z., Christofides, P.D., Armaou, A., Ni, D., 2021. Recurrent neural-network-based model predictive control of a plasma etch process. Ind. Eng. Chem. Res. 61 (1), 638–652.

Yaacob, S., Nagarajan, R., Kin, K.T.T., 2001. Application of predictive fuzzy logic controller in temperature control of phenol-formaldehyde manufacturing: using MATLAB-SIMULINK methodology. In: Intelligent Systems in Design and Manufacturing IV, Vol. 4565. pp. 101–109.

Yun, S., Tom, M., Luo, J., Orkoulas, G., Christofides, P.D., 2022. Microscopic and data-driven modeling and operation of thermal atomic layer etching of aluminum oxide thin films. Chem. Eng. Res. Des. 177, 96–107.

Zadeh, L.A., 1965. Fuzzy sets. Inf. Control 8, 338–353.

Zhang, T., Li, S., Zheng, Y., 2022. Implementable stability guaranteed Lyapunov-based data-driven model predictive control with evolving Gaussian process. Ind. Eng. Chem. Res. 61, 14681–14690.

Zhang, A., Lipton, Z.C., Li, M., Smola, A.J., 2021. Dive into deep learning. arXiv preprint arXiv:2106.11342.