Front. Chem. Sci. Eng. 2022, 16(2): 237–250
https://doi.org/10.1007/s11705-021-2058-6

RESEARCH ARTICLE

An integrated approach for machine-learning-based system identification of dynamical systems under control: application towards the model predictive control of a highly nonlinear reactor system

Ewan Chee[1], Wee Chin Wong[2], Xiaonan Wang (✉)[1]

1 Department of Chemical & Biomolecular Engineering, Faculty of Engineering, National University of Singapore, Singapore 117585, Singapore
2 Chemical Engineering & Food Technology Cluster, Singapore Institute of Technology, Singapore 138683, Singapore

Abstract Advanced model-based control strategies, e.g., model predictive control, can offer superior control of key process variables for multiple-input multiple-output systems. The quality of the system model is critical to controller performance and should adequately describe the process dynamics across its operating range while remaining amenable to fast optimization. This work articulates an integrated system identification procedure for deriving black-box nonlinear continuous-time multiple-input multiple-output system models for nonlinear model predictive control. To showcase this approach, five candidate models for polynomial and interaction features of both output and manipulated variables were trained on simulated data and integrated into a nonlinear model predictive controller for a highly nonlinear continuous stirred tank reactor system. This procedure successfully identified system models that enabled effective control in both servo and regulator problems across wider operating ranges. These controllers also had reasonable per-iteration times of ca. 0.1 s. This demonstration of how such system models could be identified for nonlinear model predictive control without prior knowledge of system dynamics opens further possibilities for direct data-driven methodologies for model-based control which, in the face of process uncertainties or modelling limitations, allow rapid and stable control over wider operating ranges.

Keywords nonlinear model predictive control, black-box modeling, continuous-time system identification, machine learning, industrial applications of process control

Received January 2, 2021; accepted March 16, 2021

E-mail: chewxia@nus.edu.sg

1 Introduction

Against a backdrop of data proliferation and a surging enthusiasm in the applications of artificial intelligence in industry, there has been a growing interest in the community to incorporate the latest results reported in the machine learning (ML) literature to infer dynamical systems from data more effectively [1]. This inference of dynamical systems is also known as system identification (SysID). This discipline has vital industrial pertinence because it yields representations of the process that, firstly, improve system understanding, and secondly, enable the simulation of complex systems to obtain time-series predictions. While this discipline is anchored in statistics and applied mathematics and is thus well-situated to take advantage of the data-driven industry 4.0 revolution, it remains an art form in practice, not least due to the numerous design decisions that practitioners encounter as well as the large number of parameters and hyperparameters that need to be estimated.

1.1 Model predictive control

A prominent application of SysID can be found in advanced model-based process control strategies such as model predictive control (MPC). MPC incorporates knowledge about the process through a system model and solves a dynamic optimization problem at each time step to yield an optimal control sequence. It then applies the first control action in this sequence to the system before proceeding to re-solve the optimization problem at the next time step [2]. MPC can be especially useful because, unlike common applications of conventional process controllers, it natively supports multiple-input multiple-output

(MIMO) formulations. It also allows system constraints, which represent process limits, to be directly included in the problem formulation, thereby ensuring plant safety and reliability in an explicit manner [3]. MPC has seen successful adoption in industry to ensure safe, reliable, and profitable system operation, with key examples present in process manufacturing, the control of complex machinery, as well as other value-adding activities [3].

The system model plays a determinant role in MPC. It should adequately describe the process dynamics across the controller operating range while remaining simple enough to allow fast optimization [4]. These models could either be constructed from first principles, which might be difficult and costly for complex processes [5], or directly inferred from empirical data [6]. These models could also have different forms: linear, nonlinear, hybrid or nonparametric, among others. MPC products have typically relied on linear models to exploit efficient optimization algorithms [7]. However, such linear MPCs might struggle to offer effective control outside a limited operating range [5,8]. Indeed, achieving tight control across a process' entire operating range is generally difficult, with other conventional applications of widely-used methods like proportional-integral-derivative controllers also ill-equipped to handle systems that are highly nonlinear around their operating points [9,10]. Nonlinear MPCs (NMPCs) can overcome these limitations by employing nonlinear system models and system constraints, which have more flexible forms that permit better representation of the process over a wider operating range. This comes however at the expense of needing to solve non-convex optimization problems quickly and precisely [11]. References [3] and [12] describe successful applications of different MPC formulations in a broad class of academic and industrial problems.

Recent advances in computing and statistics have generated novel techniques that could further improve the viability of black-box nonlinear SysID methods for model-based control. Such black-box methods are useful when process understanding is limited to begin with, or when mechanistic models derived from first principles take prohibitively long to evaluate for fast-sampling applications. Reference [13] demonstrated the neural network-based identification of single-input single-output system models for a pH neutralization process and showed that the resulting NMPC was able to track set-point changes better than a linear MPC. References [14] and [15] reported the use of recurrent neural networks and ensemble techniques to generate MIMO system models for NMPC of a continuous stirred tank reactor (CSTR) system, showing that these NMPCs performed better than linear MPCs at rejecting process disturbances.

1.2 Key contributions

Related works in the SysID literature have typically focused on presenting methodologies with special properties. Reference [1] reported a specific methodology for learning efficient models through sparse regression, for instance. The present work is instead designed to be expository and comprehensive in nature, and its main contribution is to articulate an integrated, overarching approach for learning the dynamics of continuous-time systems under control. This work is targeted at interested or starting practitioners in need of a systematic, integrated, and rigorous methodology for applying ML SysID techniques for model-based control purposes.

The key steps of this integrated approach are as follows: (i) the experimental design for data generation, (ii) the regression workflow which consists of data preprocessing techniques and the critical but often overlooked hyperparameter optimization (HO) phase, and (iii) the integration of the learnt model into an ML-NMPC controller. To formalize the goals at each step of this framework, a rigorous exposition of the ML techniques employed at each step, which include feature generation and Bayesian optimization (BO), is provided. While this framework imposes a logical structure to the flow of the data-driven tasks, it remains sufficiently flexible to allow different techniques to be employed at each stage, which future-proofs this framework in the face of further advances in ML.

This approach is exemplified through the case study of a highly nonlinear MIMO CSTR process. The resulting ML-NMPCs were evaluated based on both control performance and solution times on both servo and regulator problems to ensure the solution's applicability and practicality in real-world industrial settings. The resulting ML-NMPCs were finally benchmarked against an NMPC that employs the true system model. This NMPC is demonstrated to be able to solve all the control problems effectively.

This work is organized as follows. Section 2 formalizes the proposed integrated approach. Section 3 details the CSTR case study, presents the three control scenarios, and reports practical details in implementing this approach. Section 4 presents results and discussions. Section 5 concludes this piece and offers recommendations for future research.

2 Methodology

In the following exposition, $x \in \mathcal{K}^q$ is a column vector, with $\mathcal{K}$ representing some field and $q \geqslant 1$. The notation $[a,b]$ represents a row vector when $(a,b) \in \mathcal{K}^2$, and the T superscript represents the vector transpose operation, such that $[a,b]^{\mathrm{T}}$ is a column vector. The $\dashv\vdash$ symbol represents column vector concatenation, and it returns another column vector, such that $x_1 \dashv\vdash x_2 = [x_1^{\mathrm{T}},x_2^{\mathrm{T}}]^{\mathrm{T}}$ and $\dashv\vdash_{i\in\{1,\ldots,n\}} x_i = [x_1^{\mathrm{T}},\ldots,x_n^{\mathrm{T}}]^{\mathrm{T}}$, where $x_i \in \mathcal{K}^{q_i}$. An $n \times p$ matrix with values in $\mathcal{K}$ is denoted $\mathcal{M}_{n,p}(\mathcal{K})$, while

n-order square matrices are $\mathcal{M}_n(\mathcal{K})$. The $\dashv$ symbol is also used to represent the vertical concatenation of matrices with the same number of columns, such that $\dashv_{i\in\{1,\ldots,n\}} B_i = [B_1^{\mathrm{T}},\ldots,B_n^{\mathrm{T}}]^{\mathrm{T}}$, where $B_i \in \mathcal{M}_{q_i,p}(\mathcal{K})$. Sets are represented by $\{1,\ldots,k\}$. Closed intervals are defined as $[\alpha;\beta]$, with α and β real numbers. Left and right half-open intervals are $]\alpha;\beta]$ and $[\alpha;\beta[$ respectively, with open intervals denoted as $]\alpha;\beta[$. Given $x(t) \in \mathcal{K}^q$ a continuous-time vector-valued quantity, where $t \in \mathbb{R}_+$, $x[k]$ represents its value associated with the discrete time-step k, such that $x[k] = x(t_k)$, where t_k is the real time associated with the discrete time-step k. Δ finally represents the difference operator, such that $\Delta x[k] := x[k+1] - x[k]$.

Equations (1) and (2) describe a general continuous-time state representation for a dynamical system.

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

$$y = g(x,u) + \epsilon, \tag{2}$$

where $x \in \mathbb{R}^a$, $u \in \mathbb{R}^b$ and $y \in \mathbb{R}^c$ are the state, control, and output vectors respectively, with $f : \mathbb{R}^a \times \mathbb{R}^b \to \mathbb{R}^a$ a general nonlinear state equation and $g : \mathbb{R}^a \times \mathbb{R}^b \to \mathbb{R}^c$ a general nonlinear output equation; ϵ is a random variable corresponding to measurement noise. In what follows, we assume complete state information, such that $y = x + \epsilon$. For a MIMO system, a, b and c are integers that are strictly greater than 1.

If f is a highly nonlinear function, conventional applications of widely-used strategies like proportional-integral-derivative control and linear MPC might not offer effective control beyond a limited range. This motivates the study of nonlinear control strategies like NMPC, which the following subsection describes.

2.1 Nonlinear model predictive control

Given a discrete-time controller, Eq. (3) defines the cost function J at any given discrete time-step $k \in \mathbb{N}$:

$$J(\Delta U) := \mathrm{tr}((Y - Y^*)^{\mathrm{T}} Q (Y - Y^*)) + \mathrm{tr}(\Delta U^{\mathrm{T}} R \Delta U), \tag{3}$$

where $\Delta U := [\Delta u\,[k],\ \Delta u\,[k+1],\ \ldots,\ \Delta u\,[k+m-1]]^{\mathrm{T}} \in \mathcal{M}_{m,b}(\mathbb{R})$ is the control sequence over the control horizon $m \in \mathbb{N}^*$, $Y := [y[k+1|k],\ldots,y[k+p|k]]^{\mathrm{T}} \in \mathcal{M}_{p,c}(\mathbb{R})$ is the output trajectory over the prediction horizon $p \in \mathbb{N}^*$, $Y^* \in \mathcal{M}_{p,c}(\mathbb{R})$ is the system set-point over p, $Q \in \mathcal{M}_c(\mathbb{R}_+)$ and $R \in \mathcal{M}_b(\mathbb{R}_+)$ are diagonal non-negative weight matrices whose coefficients reflect the relative importance of the corresponding terms in the cost function, and $\mathrm{tr}(\cdot)$ represents the trace of the matrix. The control actions are applied in a zero-order hold fashion throughout the sampling interval, such that $u(t) = u[k]$, $\forall t \in [t_k;t_{k+1}]$.

Equations (4–8) formulate the NMPC problem, which is solved in a receding horizon fashion:

$$\min_{\Delta U} J(\Delta U)\ , \tag{4}$$

$$s.t.\ Y = \mathcal{F}(x[k],u[k],\Delta U \dashv 0_{p-m,b}), \tag{5}$$

$$\Delta u_{\min} \leqslant \Delta u[l] \leqslant \Delta u_{\max},\ \forall l \in \{k,\ldots,k+m-1\}, \tag{6}$$

$$u_{\min} \leqslant u[l] \leqslant u_{\max},\ \forall l \in \{k,\ldots,k+m-1\}, \tag{7}$$

$$y_{\min} \leqslant y[l] \leqslant y_{\max},\ \forall l \in \{k+1,\ldots,k+p\}, \tag{8}$$

where $\mathcal{F}$ is a nonlinear function which generates the discrete-time output trajectory given the system's initial state and the control sequence, and $0_{p-m,b}$ is a zero matrix of dimension $(p-m) \times b$. The concatenation of $0_{p-m,b}$ is equivalent to defining $\Delta u[l] = 0, \forall l \in \{k+m,\ldots,k+p-1\}$, then appending these zero values to ΔU such that the resulting matrix has p rows. Therefore, u remains constant after the control horizon. $\mathcal{F}$ can be generated from f as Eq. (9) shows:

$$\mathcal{F}(x[k],u[k],\Delta U \dashv 0_{p-m,b}) = [\mathcal{G}(1),\ldots,\mathcal{G}(p)]^{\mathrm{T}}, \tag{9}$$

where $\mathcal{G}(l) := x(t_k) + \int_{t_k}^{t_{k+l}} f\Big(x(\tau),u(\tau)\Big)\ \mathrm{d}\tau, \forall l \in \{1,\ldots,p\}$. Equations (6–8) represent constraints on the control actions' rates of change, constraints on their magnitude, and constraints on system outputs, respectively.

This NMPC problem is a constrained nonlinear optimization problem which can be solved using active set or interior point methods. As f and $\mathcal{F}$ both return the same vector Y, they are equivalent from the controller's perspective. We thus refer to f as the NMPC system model in what follows. The continuous-time SysID approach was pursued here because it may be more intuitive to scientists and engineers [16]. With that said, the proposed approach is potentially also amenable to discrete-time formulations after suitable modifications.

If f or $\mathcal{F}$ is not known a priori, it becomes necessary to first identify it from empirical data. The following subsection describes how recent advances in ML could be applied in the context of a broader integrative SysID approach for nonlinear control.

2.2 Black-box direct continuous-time SysID

Figure 1 illustrates the three-step framework for directly identifying f as NMPC system models from empirical data. The data generation procedure involves performing experiments that perturb the system across the controller's operating region to reveal system dynamics. The regression workflow then proceeds to infer nonlinear continuous-time functions as NMPC system models. These system models are finally integrated into the ML-NMPC controllers, which are subsequently tested to evaluate their closed-loop control performance.

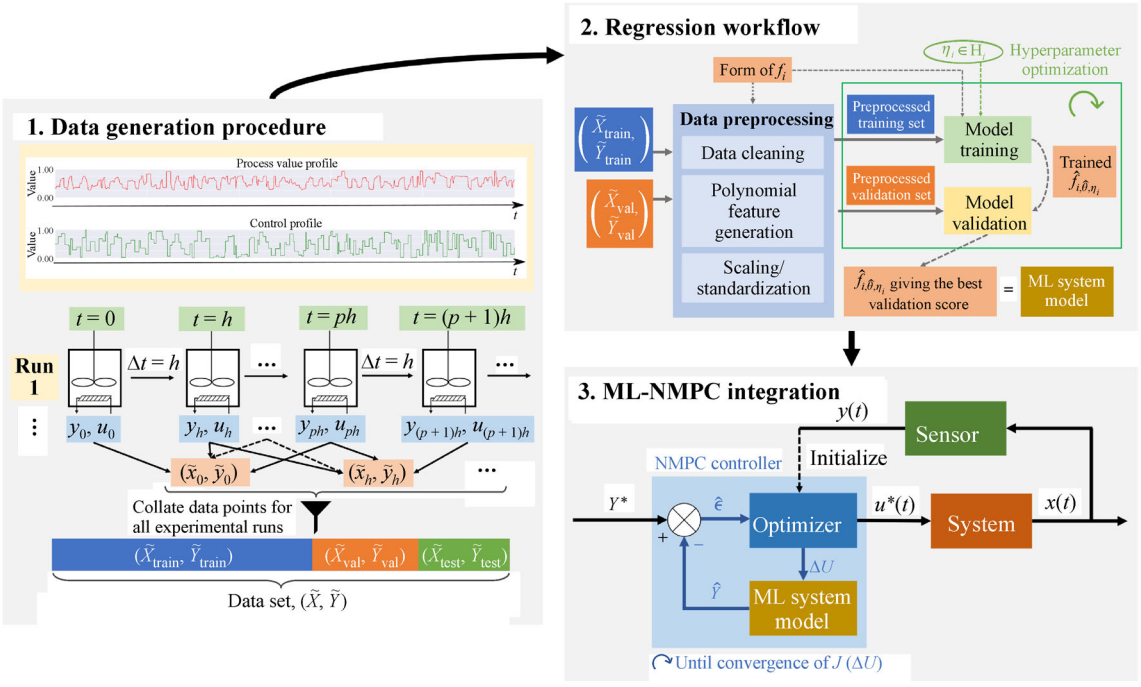

Fig. 1 Integrated data-driven approach for identification of continuous-time NMPC system models.

2.2.1 Data generation procedure

The design of the input signal is critical to generating a rich data set, and the signal proposed in this work consists of simultaneous random step changes for all manipulated variables. The experimental sampling time h is also an important parameter as the following discussion will show.

To generate an input signal of time length $t_{sim} \in \mathbb{R}_+^*$ for a single manipulated variable, a partition of the interval $[0; t_{sim}]$ is first performed by performing a random jump forward in time from the start of the interval, with the jump length in time units (tus) sampled from a discrete uniform distribution $\mathcal{U}(\{l_{min}, l_{min}+h, \ldots, l_{max}\})$, where l_{min} and l_{max} are the minimum and maximum jump lengths in tus respectively and are both integer multiples of h, then repeating these jumps until t_{sim} is reached. The value of the manipulated variable at each subinterval is then sampled from a continuous uniform distribution $\mathcal{U}([u_{min}; u_{max}])$, where u_{min} and u_{max} are the lower and upper bounds of the manipulated variable as specified in the NMPC formulation, respectively. This process is repeated for each manipulated variable.

Figure 2 shows an example of an input signal and the system response when the input signal is applied to the system. As the control values reached are uniformly represented across the controller's operating region, it facilitates the learning of ML models that interpolate well to give good dynamical predictions within this region.

Given an input signal $(u_i)_{i\in\{0,\ldots,t_{sim}/h\}}$ and the system response $(y_i)_{i\in\{0,\ldots,t_{sim}/h\}}$, where u_i and y_i are potentially multidimensional, we pose $\tilde{x}_i := y_i \# u_i$ and $\tilde{y}_i := [(y_{i+1} - y_i)/h]$, $\forall i \in \{0,\ldots,t_{sim}/h - 1\}$. $\tilde{x}_i$ and $\tilde{y}_i$ represent data points and their corresponding labels, which are alternatively the independent and dependent variables respectively, and this supervised learning task is formalized in Section 2.2.2.

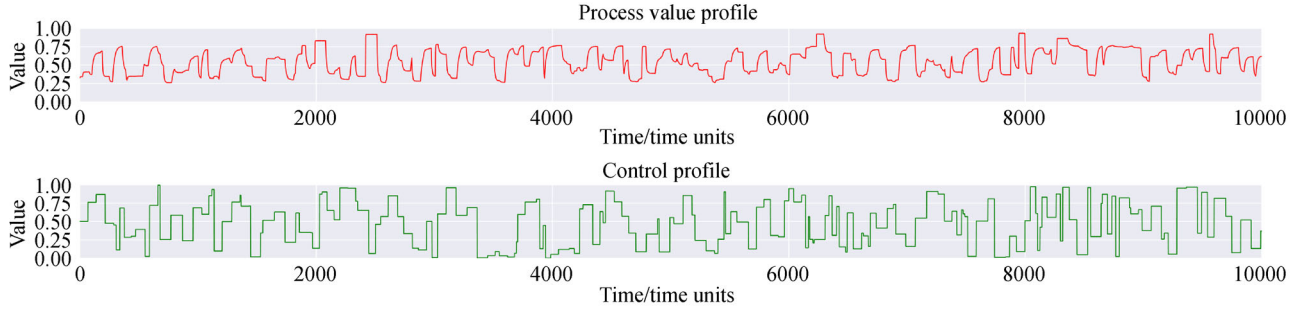

Fig. 2 Example input signal (green) and system response (red), where $t = 10000$, $l_{min} = 10$, $l_{max} = 100$, $u_{min} = 0$, $u_{max} = 1$.

As $\tilde{y}_i$ contain first-order approximations of the continuous derivatives representing the system dynamics, values for h that are small with respect to the system's characteristic time would be ideal, i.e., at least an order of magnitude smaller. In this work, the initial process values are taken to be the set-point values. Combining data points from multiple experimental runs is also straightforward, and it is indeed desirable to use as many data points as could possibly be collected. This is because, in general, model variance decreases as the number of data points increases, such that more precise estimates for the model parameters are achieved.

2.2.2 Regression workflow

We pose $\tilde{X} := [\tilde{x}_i]^{\mathrm{T}}_{i\in\{1,\dots,N\}}$ the design matrix with $\{\tilde{x}_i\}_{i\in\{1,\dots,N\}}$ the data set of size $N \in \mathbb{N}^*$, and $\tilde{Y} := [\tilde{y}_i]^{\mathrm{T}}_{i\in\{1,\dots,N\}}$ the labels matrix with $\{\tilde{y}_i\}_{i\in\{1,\dots,N\}}$ the set of labels associated to $\{\tilde{x}_i\}$.

Before performing the regression task, the data set is split into training, validation, and test sets. The training set contains the data used to fit a given model. The validation set contains the data used to provide an unbiased evaluation of a model fit on the training set while tuning the model hyperparameters. The test data finally contains the data used to provide an unbiased evaluation of the tuned models. The validation and test sets are "held out" from the training set to prevent data leakage, which is when information outside the training set is used to create the model. Data leakage causes the estimates of the final model's general predictive power to be overly optimistic. We note I_{train}, I_{val} and I_{test} the set of indices of $\tilde{X}$ corresponding to these sets, with $\text{card}(I_{\text{train}}) = N_{\text{train}}$, $\text{card}(I_{\text{val}}) = N_{\text{val}}$ and $\text{card}(I_{\text{test}}) = N_{\text{test}}$, such that $N_{\text{train}} + N_{\text{val}} + N_{\text{test}} = N$. $\tilde{X}_{\text{train}} := [\tilde{x}_i]^{\mathrm{T}}_{i\in I_{\text{train}}}$ is the training design matrix with $\tilde{Y}_{\text{train}} := [\tilde{y}_i]^{\mathrm{T}}_{i\in I_{\text{train}}}$ its labels matrix. $\tilde{X}_{\text{val}}$, $\tilde{Y}_{\text{val}}$, $\tilde{X}_{\text{test}}$ and $\tilde{Y}_{\text{test}}$ are defined similarly.

Data preprocessing encompasses a broad set of strategies that prepares the data set for the model training phase in such a way as to facilitate the learning of models that generalize well. It generally involves data cleaning, which removes outliers that might have resulted from sensor faults or data input errors, and feature engineering, which encompasses other techniques like feature transformation, feature generation, and feature extraction. In this work, we focus on the generation of polynomial and interaction features, which represent higher-order and coupled nonlinear terms in the system dynamics. We pose $\tilde{X} = \left[\tilde{X}_j\right]_{j\in\{1,\dots,N_f\}}$, where $\tilde{X}_{j\in\{1,\dots,N_f\}}$ are feature vectors with $N_f \in \mathbb{N}$ the number of features. Equation (10) formalizes the feature generation procedure of degree $d \in \mathbb{N}^*$:

$$\tilde{X}^{(d)} = \left[\tilde{X}_1^{p_1}\tilde{X}_1^{p_2}\dots\tilde{X}_{N_f}^{p_{N_f}}\right]_{(p_1,\dots,p_{N_f})\in\{0,\dots,\ell\}^{N_f}\ \text{s.t.}\ p_1+\dots p_{N_f}\leqslant\ell,\,\forall\ell\in\{1,\dots,d\}} \tag{10}$$

To illustrate this procedure, suppose that $\tilde{X} = \left[\tilde{X}_1,\tilde{X}_2\right]$. A feature generation of degree 2 yields the following matrix:

$$\tilde{X}^{(2)} = \left[1,\tilde{X}_1,\tilde{X}_2,\tilde{X}_1^2,\tilde{X}_1\tilde{X}_2,\tilde{X}_2^2\right]. \tag{11}$$

Data standardization is also performed to allow all features to be considered equally during model training. Equation (12) describes this operation:

$$\tilde{X}^{(d)}_{\cdot,std} = \left[\frac{\tilde{X}^{(d)}_{\cdot,j} - \overline{\tilde{X}}^{(d)}_{\star,j}}{\hat{\sigma}_{\star,j}}\right]_{j\in\{1,\dots,N_f^{(d)}\}}, \tag{12}$$

where $\tilde{X}^{(d)}_{\cdot,j}$ represents the j-th column of $\tilde{X}^{(d)}_{\cdot}$, $\overline{\tilde{X}}^{(d)}_{\star,j}$ the sample mean of $\tilde{X}^{(d)}_{\star,j}$, $\hat{\sigma}_{\star,j}$ the population standard deviation of $\tilde{X}^{(d)}_{\star,j}$, and $N_f^{(d)} \in \mathbb{N}^{\star}$ the total number of polynomial and interaction features. $\cdot$ is a placeholder referring either to the training, validation, or test sets. If $\cdot$ refers to either $\tilde{X}_{\text{train}}$ or $\tilde{X}_{\text{val}}$, then $\star$ refers to $\tilde{X}_{\text{train}}$. If $\cdot$ refers to $\tilde{X}_{\text{test}}$, then $\star$ refers to $\tilde{X}_{\text{train}} \cup \tilde{X}_{\text{val}}$. This mapping of $\cdot$ to $\star$ is done to prevent data leakage.

The model training phase corresponds to solving for a prediction function f such that $\tilde{y} \approx f(\tilde{x})$. Suppose that we have n candidate models, i.e., n different regression forms for f. For each $f_{i\in\{1,\dots,n\}}$, we solve the mean squared error (MSE) minimization problem in Eq. (13):

$$\hat{\theta}_i = \underset{\theta_i\in\Theta_i}{\arg\min}\frac{1}{N_{\text{train}}}\sum\nolimits_{j\in I_{\text{train}}}\left(\tilde{y}_j - f_{i,\theta_i}\left(\tilde{x}_j\right)\right)^2, \tag{13}$$

where Θ_i represents the parameter space for f_i. $f_{i,\hat{\theta}_i}$ finally represents the solution to this regression problem, and this procedure is repeated for each f_i. Let $\eta_i \in \mathrm{H}_i$ be the set of hyperparameters for f_i. HO involves training f_i with different η_i, then selecting $\hat{\eta}_i$ which satisfies Eq. (14):

$$\hat{\eta}_i = \underset{\eta_i\in\mathrm{H}_i}{\arg\min}\frac{1}{N_{\text{val}}}\sum\nolimits_{j\in I_{\text{val}}}\left(\tilde{y}_j - f_{i,\hat{\theta}_i;\eta_i}\left(\tilde{x}_j\right)\right)^2. \tag{14}$$

BO is employed in this work for HO, with exhaustive grid searches used instead if the hyperparameter space is countable and sufficiently small. BO is a class of optimization techniques that applies Bayes Theorem to quickly find the global optima of a general nonlinear multidimensional function [17]. Unlike grid and random searches, BO takes past evaluations into account by using them to build a probabilistic surrogate model of the HO objective function. It uses this model to select the most promising hyperparameters at each iteration. The objective

function is then evaluated at those hyperparameters, with this result subsequently used to update the surrogate model. This iterative procedure therefore yields posteriors that better approximate the HO objective function with each evaluation, from which the best hyperparameters can be hoped to be more easily located [17]. As function evaluations for Eq. (14) are expensive, since an iteration involves training an entire model, such a principled approach which balances exploration of the hyperparameter space and exploitation of best found values is desirable [18,19].

The formalization of BO consists of five key elements. The first element is H_i, the hyperparameter space, and the second is the HO objective function, which was present in Eq. (14) and is explicated here:

$$\mathcal{J}(\eta_i) := \frac{1}{N_{\mathrm{val}}} \sum_{j \in I_{\mathrm{val}}} \left(\tilde{y}_j - f_{i,\hat{\theta}_i;\eta_i}(\tilde{x}_j) \right)^2. \tag{15}$$

The third element is the surrogate model, which is an alternative probabilistic representation of the objective function that is cheaper to evaluate. The fourth is the acquisition function which is defined over H_i and is computed from the surrogate model, and it quantifies the desirability of sampling a given point in H_i next. The final element is the BO history, which is a set $\mathcal{D}_{\mathrm{BO},s} := \{(\eta_{i,v}, \mathcal{J}(\eta_{i,v}))\}_{v \in \{1,\ldots,s\}}$ containing the s samples drawn from $\mathcal{J}$ so far. The surrogate model is referred to as the posterior distribution when it is conditioned by $\mathcal{D}_{\mathrm{BO},s}$, and is represented by $p_{\mathcal{D}_{\mathrm{BO},s}}$. We also drop the i subscript in an abuse of notation for the remaining technical development of BO.

Common acquisition functions include the probability of improvement, expected improvement (EI), entropy search, and lower confidence bound. At the $(s+1)$-th iteration, these functions have the generic form shown in Eq. (16):

$$A_{s+1}(\eta) = \mathbb{E}_{\mathcal{D}_{\mathrm{BO},s}}[u_{s+1}(\eta)], \tag{16}$$

where $u_{s+1} : \mathrm{H} \to \mathbb{R}$ is a utility function to maximize. *EI* is the most common acquisition function and is used in this study, and it has the following utility function at the $(s+1)$-th iteration:

$$u_{EI,s+1}(\eta) = \max\left(0, \mathcal{J}_s^* - \mathcal{J}(\eta)\right), \tag{17}$$

where $\mathcal{J}_s^* := \min_{v \in \{1,\ldots,s\}} \mathcal{J}(\eta_v)$ is the best value for $\mathcal{J}$ found after s iterations. The acquisition function $A_{EI,s+1}(\eta)$ at the $(s+1)$-th evaluation is thus:

$$\begin{aligned} A_{EI,s+1}(\eta) &= \int_{-\infty}^{\infty} \max(0, \mathcal{J}_s^* - \mathcal{J})\, p_{\mathcal{D}_{BO,s}}(\mathcal{J}|\eta)\, \mathrm{d}\mathcal{J} \\ &= \int_{-\infty}^{\mathcal{J}_s^*} (\mathcal{J}_s^* - \mathcal{J})\, p_{\mathcal{D}_{BO,s}}(\mathcal{J}|\eta)\, \mathrm{d}\mathcal{J}. \end{aligned} \tag{18}$$

This finally yields the following maximization problem at the $(s+1)$-th evaluation:

$$\begin{aligned} \eta_{s+1} &= \arg\max_{\eta \in \mathrm{H}} A_{EI,s+1}(\eta) \\ &= \arg\max_{\eta \in \mathrm{H}} \int_{-\infty}^{\mathcal{J}_s^*} (\mathcal{J}_s^* - \mathcal{J})\, p_{\mathcal{D}_{BO,s}}(\mathcal{J}|\eta)\, \mathrm{d}\mathcal{J}. \end{aligned} \tag{19}$$

This therefore corresponds to finding $\eta \in \mathrm{H}$ which is expected to improve on $\mathcal{J}_s^*$ the most, given the current surrogate model $p_{\mathcal{D}_{BO,s}}(\mathcal{J}|\eta)$.

Common forms for the surrogate model include Gaussian processes and Tree-Parzen estimators (TPEs), with the latter being applied in this work. Technical details for the structure of TPEs and how EI is optimized in the TPE algorithm can be found in ref. [18]. The degree for polynomial feature generation is considered as a hyperparameter for all candidate models, with $d_i \in \{1,\ldots,10\}$, $\forall i$.

Each candidate model f_i whose best hyperparameters $\hat{\eta}_i$ have been found is then trained on $\tilde{X}_{\mathrm{train}} \cup \tilde{X}_{\mathrm{val}}$ and tested on $\tilde{X}_{\mathrm{test}}$ to determine its test MSE, which serves as an indicator of the model's general predictive ability with respect to other candidate models after hyperparameter tuning. Before integration into ML-NMPC, the models are initialized with their best hyperparameters $\hat{\eta}_i$ and trained on the full data set $\tilde{X}$.

In situations where data is sparse, it would be useful to apply cross-validation techniques to obtain better estimates for the model's general predictive power during HO. K-fold cross-validation involves, firstly, extracting $\tilde{X}_{\mathrm{test}}$ from $\tilde{X}$, then splitting the remaining data into K equal folds, i.e., $\{\tilde{X}_i\}_{i \in \{1,\ldots,K\}}$, where $K \leqslant N - N_{\mathrm{test}}$. K number of models are then trained for each hyperparameter configuration, with each run $l \in \{1,\ldots,K\}$ having $\cup_{i \in \{1,\ldots,K\} \setminus l} \tilde{X}_l$ as the training set and $\tilde{X}_l$ as its validation set. This technique returns K validation MSEs, which can be averaged to obtain a less biased estimate of the model's goodness. K is typically selected to be 5 or 10. Leave-one-out cross-validation involves selecting $K = N - N_{\mathrm{test}}$ and, while it is the most computationally expensive, returns the least biased estimate of the model's goodness. As it is assumed that enough data has been generated, cross-validation techniques are not employed in this work.

2.2.3 Integration of ML model into NMPC

In the last step of this framework, the models $f_{i,\hat{\theta}_i;\hat{\eta}_i}$ obtained from the regression workflow are integrated into the ML-NMPC as system models and used to generate predictions for $\dot{x}$, which themselves are used to evolve x in time to generate Y given ΔU. To solve this system of first-order ordinary differential equations, numerical methods like Runge-Kutta methods and linear multistep methods can be used.

Tuning of the ML-NMPC control parameters is done

before testing its closed-loop performance on the case study. In this work, this consists of modifying m to minimize the weighted integrated absolute error (WIAE) for a ±5% set-point step experiment:

$$\text{WIAE} := \int_{t_0}^{t_f} W\|y(t) - y^*(t)\|_1 \, \mathrm{d}t, \tag{17}$$

where $W \in \mathcal{M}_c(\mathbb{R}_+)$ is a weight matrix, and $t_0 \in \mathbb{R}_+$ and $t_f \in \mathbb{R}_+$ are the start and end times of the experiment, respectively. $p = 5$ was taken to be fixed in this work. The WIAE was selected as the figure of merit for its simple interpretation, though other controller tuning statistics that explicitly include ΔU or that have more elaborate forms could also be used without any loss of generality of this procedure (see ref. [20] for other statistics that could be used in place of the WIAE). The controller's sampling time h_{NMPC} is typically chosen to be 5 to 10 times faster than the system's characteristic time as a rule of thumb [21].

3 Case study

3.1 Plant model

The case study, which was also studied in ref. [22], consists of a single CSTR system which houses a reversible chemical reaction described in Eq. (20), where A is the feed species, R the desired product, and S the undesired byproduct. Non-dimensionalized expressions for the system's dynamical behavior are shown in Eqs. (21–23):

$$A \underset{k_4}{\overset{k_1}{\rightleftharpoons}} R \underset{k_3}{\overset{k_2}{\rightleftharpoons}} S, \tag{20}$$

$$\frac{\mathrm{d}C_A}{\mathrm{d}t} = q[C_{A0} - C_A] - k_1 C_A + k_4 C_R, \tag{21}$$

$$\frac{\mathrm{d}C_R}{\mathrm{d}t} = q[1 - C_{A0} - C_R] + k_1 C_A + k_3[1 - C_A - C_R] - [k_2 + k_4]C_R, \tag{22}$$

$$k_j = k_{0,j}\exp\left\{\left[-\frac{E}{RT_0}\right]_j\left[\frac{1}{T} - 1\right]\right\}, j \in \{1,2,3,4\}, \tag{23}$$

where $C_{i\in\{A,R,S\}} \in \mathbb{R}_+$ is the species' reactor concentration, $C_{A0} \in \mathbb{R}_+$ is the feed concentration of A, $q \in \mathbb{R}_+$ is the feed and the exit flow rate, $k_{j\in\{1,\ldots,4\}} \in \mathbb{R}_+$ are the reaction rate constants, $k_{0,j\in\{1,\ldots,4\}} \in \mathbb{R}_+$ are the Arrhenius pre-exponential constants, $[E/RT_0]_{j\in\{1,\ldots,4\}} \in \mathbb{R}_+$ are the normalized activation energies, and $T \in \mathbb{R}_+$ is the system temperature. q and T are the manipulated variables in this study. The values for the model parameters are reported in Appendix A (cf. Electronic Supplementary Material, ESM).

Figure 3 schematizes the CSTR plant and plots the system's steady-state conditions as a function of T when q is 0.8 and at its nominal point. To simplify the system, it is assumed that the low-level flow and temperature control loops have negligible dead times and present no steady-state errors. The system's set-point was selected to maximize the ratio of C_R to C_A. This corresponds also to maximizing product yield and minimizing any downstream separations costs. There are strong non-linearities in how both C_A and C_R vary with T at the set-point. There is also input multiplicity, since two different values of T could yield the same value for C_R, which suggests that there are sign changes in the determinant of the steady-state gain matrix within the operating region that make linear controllers with integral action unstable [23]. The control of this system at this set-point is therefore a relevant and interesting problem.

To numerically simulate this system, a linear multistep method implemented through the LSODA option in integrate.solve_ivp in the *SciPy* package (version 1.5) was used [24]. A Gaussian noise $\epsilon \sim \mathcal{N}(0,10^{-3})$ was included in every measurement of C_A and C_R to simulate measurement noise. Given the range of values for C_A and C_R, this corresponds to a measurement error of ca. 1%.

3.2 Control scenarios

Controller performance was evaluated based on 3

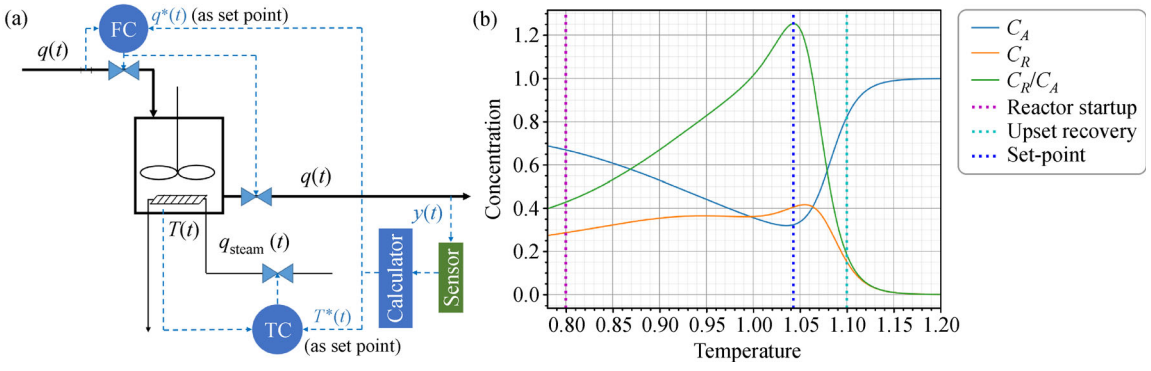

Fig. 3 (a) Schematic diagram of CSTR plant, where q^* and T^* represent set-points for q and T respectively; (b) system steady-state conditions for $q = 0.8$ ("Reactor startup" and "Upset recovery" refer to start points for the control problems in Section 3.2).

scenarios. The first scenario was a servo control problem representing ±5% step changes to the C_R set-point. This step size was selected because the control actions needed to stabilize the system at the new set-points are large enough to pose a control problem that is challenging enough to be studied meaningfully. Each experimental run began at the set-point and was followed by the +5% step and the −5% step before reverting to the set-point, such that the experiment had the following C_R profile: [0.406, 0.426, 0.386, 0.406]. Each step lasted for 5 tus for a total experimental length of 20 tus.

The second and third scenarios were regulator control problems corresponding to "reactor startup" and "upset recovery" respectively. These two process disturbances lay at opposite sides of the set-point, with the former having low T and C_R values and the latter having an abnormally high T value. A process controller would therefore need to optimize correctly for T values within an extended range of 0.8 and 1.1 to perform well in both these scenarios. All experimental runs for these two scenarios lasted for 10 tus.

3.3 Framework application

To infer f for this case study, we pose $\tilde{x}_i := [C_A, C_R, q, T]_i^{\mathrm{T}}$ and $\tilde{y}_i := [\mathrm{d}C_A/\mathrm{d}t, \mathrm{d}C_R/\mathrm{d}t]_i^{\mathrm{T}}$. Note that the polynomial and interaction features for $\tilde{x}_j$ up to degree d would be appended to this vector before the model training phase as part of the feature generation procedure described in Section 2.2.2. In this work, $h = 0.1$ tu, $t_{\mathrm{sim}} = 1000$, $l_{\min} = 1$, $l_{\max} = 10$, $q_{\min} = 0.5$, $q_{\max} = 0.9$, $T_{\min} = 0.7$ and $T_{\max} = 1.1$. 5 runs were performed, and an example of an experimental run is shown in Fig. 4.

In this work, a 60%, 20%, 20% split for the training, validation and test sets was used. The polynomial feature generation and data standardization operations were performed with preprocessing.PolynomialFeatures and preprocessing.StandardScaler from scikit-learn (version 0.23) respectively [25]. The scikit-learn package was also used to perform model training for different candidate models. The optuna package (version 2.3) in Python 3.8 was used to perform BO [26]. The f_i studied in this work are common regression forms and include linear regression, which we henceforth refer to as "polynomial" regression, support vector regression (SVR), decision tree (DT) regression, extra trees (ET) regression and gradient boosted (GB) regression. The model hyperparameters η_i for each f_i are reported in Appendix B (cf. ESM).

The NMPC problem was solved using the sequential least squares quadratic programming method which was implemented in optimize.minimize from *SciPy* (version 1.5) [24]. The constraints on the control actions' magnitude and system outputs, which are Eqs. (7) and (8) respectively, were manifested as soft constraints, with the coefficients of their penalty terms in the objective function taken to be 1000. The nonlinear optimizer was initialized with values sampled from $\mathcal{U}([-10^{-3}, 10^{-3}])$ at the first time step and was warm-started with the solution from the previous time step in subsequent time steps. In this study $h_{\mathrm{NMPC}} = 0.2$ tu.

To evolve the system in time with the ML models in the ML-NMPC, the RK23 solver in optimize.minimize from *SciPy* (version 1.5) was used. $W = \mathrm{diag}([1,3])$ was chosen for the WIAE to punish C_R deviations 3 times more heavily than C_A deviations.

4 Results and discussion

We report firstly the closed-loop performances of an NMPC controller with the exact system model, which will serve as a benchmark for ML-NMPC. The performance of the ML models from the SysID procedure will be quantified through their validation and test R^2 and MSE values, following which the closed-loop performances of these ML-NMPCs on the control scenarios will be presented.

In what follows, we refer to the NMPC with the exact system model as the "exact NMPC". For the ML-NMPCs,

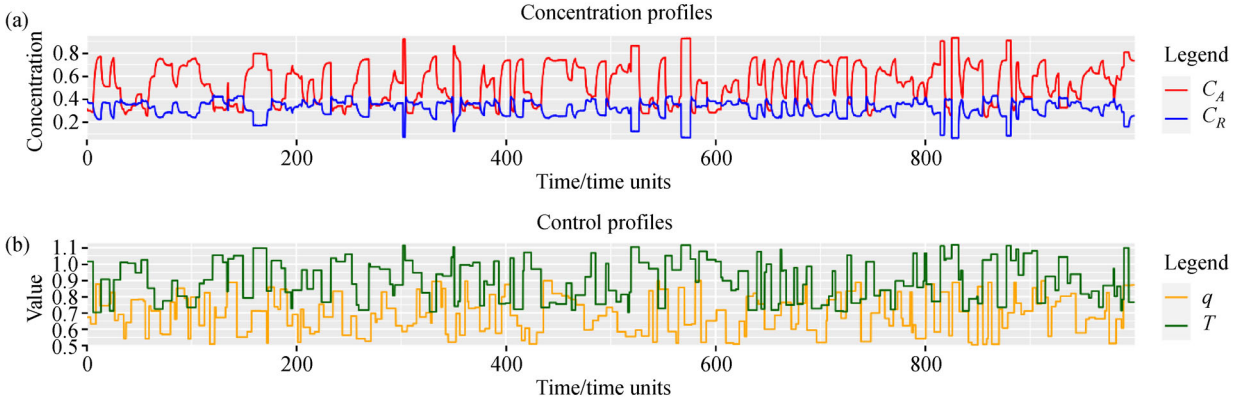

Fig. 4 Example system response (a) to an input signal (b) for the CSTR case study.

we use the shorthand "polynomial-NMPC" to refer to ML-NMPCs with a system model that is linear in the polynomial and interaction features, "DT-NMPC" to refer to ML-NMPCs with a DT system model, and so on for the other ML models. All calculations were performed in a Python 3.8 environment on a computer with 16 GB of RAM and a 4-core Intel® Core™ i7-4790 CPU running at 3.60 GHz.

4.1 Nonlinear MPC with exact system model

Figure 5 shows the results for the exact NMPC with $p = 5$ and different values of m, and it can be observed that $m = 1$ offers the lowest WIAE on the step experiments while also taking the least time to solve. Figure 6 shows the performance for the tuned exact NMPC on the three control scenarios. Stable closed-loop performance was achieved for all control scenarios. The servo, startup, and recovery problems took 3.65, 2.24 and 1.82 s to solve respectively for the $m = 1$ case, demonstrating average per-iteration solution times in the order of 10 ms. This therefore validates the use of the exact NMPC as a useful benchmark for rapid and stable control of this system. Appendix C (cf. ESM) reports the parameters for this exact NMPC.

While it is expected that greater values of m provide tighter control, albeit with longer solution times, $m = 1$ was shown to give lower values for WIAE. However, this came with more aggressive control actions, as Fig. 7 shows. The larger the value for m, the smoother the control profiles, with this demonstrating how the changes in the control action constitute an important term in the NMPC objective function. Having selected WIAE as our figure of merit for the sake of interpretability, we keep $m = 1$ as the tuned m value, making again a further mention that other statistics for tuning m could also be employed which

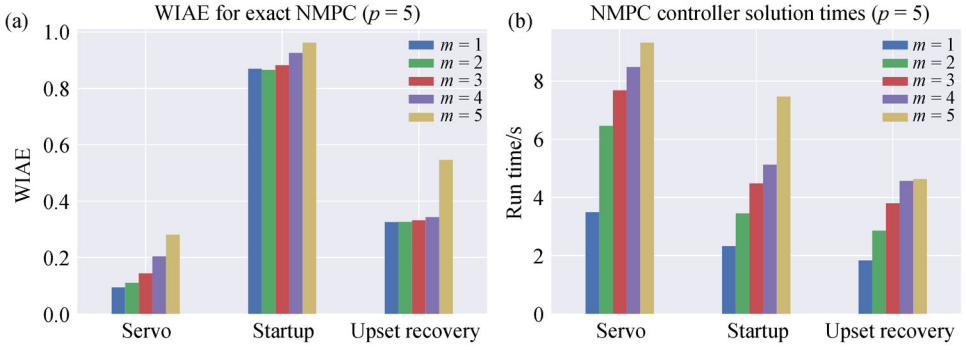

Fig. 5 (a) WIAE for exact NMPC with $p = 5$; (b) exact NMPC solution times with $p = 5$.

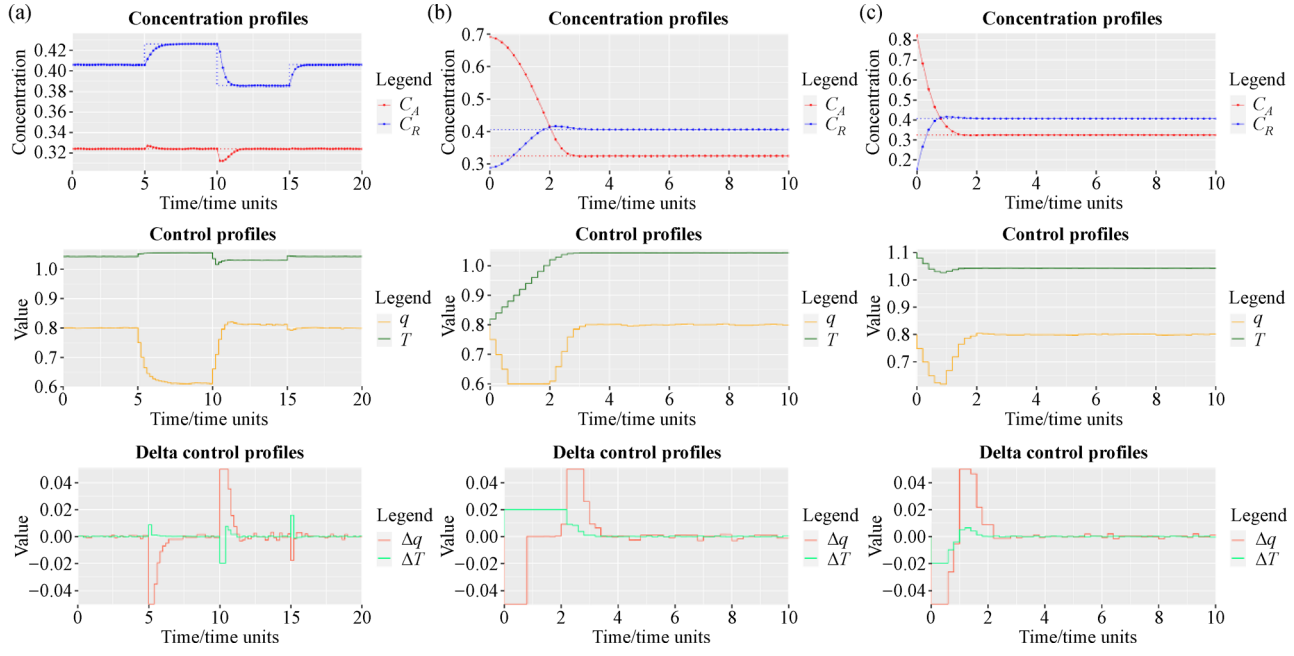

Fig. 6 Exact NMPC performance for $p = 5$ and $m = 1$: (a) servo problem, (b) startup problem, and (c) upset recovery problem.

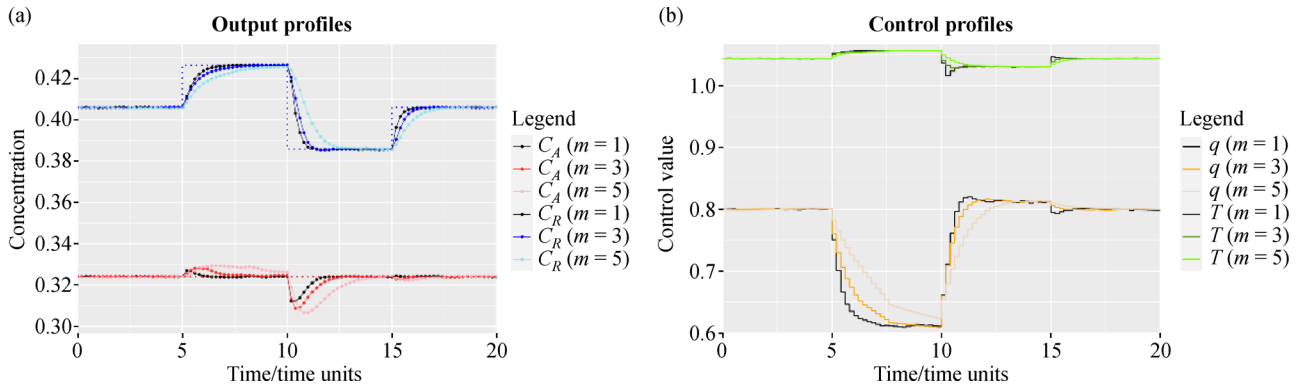

Fig. 7 Exact NMPC performance for $m = 1, 3, 5$ for the servo problem: (a) output profiles, and (b) control profiles.

explicitly considers the control actions. The ML-NMPCs presented in Section 4.3 will also take $m = 1$ as their tuned m value.

4.2 HO

Figure 8 plots the validation and test R^2 scores, and the test MSE scores, for each candidate model. Figure 9 presents more insights into the BO procedure as illustrated through the optimization history plots and slice plots, taking SVR as an example. Appendix D (cf. ESM) reports the best hyperparameters $\hat{\eta}_i$ for each f_i found through BO.

The tuned polynomial model gave the highest test R^2 value and the lowest test MSE value, suggesting that it would return better estimates for $\mathrm{d}C_A/\mathrm{d}t$ and $\mathrm{d}C_R/\mathrm{d}t$ than other candidate models for this case study. The tuned SVR model gave slightly poorer R^2 and MSE scores than the polynomial model, with the tree-based models, i.e., DT, ET, and GB, performing even more poorly than SVR. It is worth noting that different candidate models possess different structures that permit them to model some dynamical systems more efficiently than others. The validation and test results of each model relative to other models can therefore be expected to differ from one case study to another.

The optimization history in Fig. 9(a) illustrates how the BO procedure balances exploration of the hyperparameter space with the exploitation of best found values. The first few trials gave models that returned poor validation scores, and as better hyperparameter configurations were encountered the validation scores of subsequent models remained around similarly good values, suggesting that the sampling strategy began focusing on points close to the best-found hyperparameters. Figure 9(b) reinforces this observation by showing how later trials focused on the region around $\epsilon = 0.02$, with sporadic evaluations at values outside of that region.

4.3 Closed-loop control results

After HO, the tuned candidate models were integrated into ML-NMPC controllers and tested against the three control scenarios. Figure 10 shows the results for ML-NMPC. Figures 11, 12 and 13 show concentration and control paths from the polynomial-NMPC and SVR-NMPC for the servo, startup, and upset recovery problems, respectively. These paths are compared against those from the exact NMPC. Figure 14 shows GB-NMPC performance on all three control problems, with these results being representative of the performances of ML-NMPC employing tree-

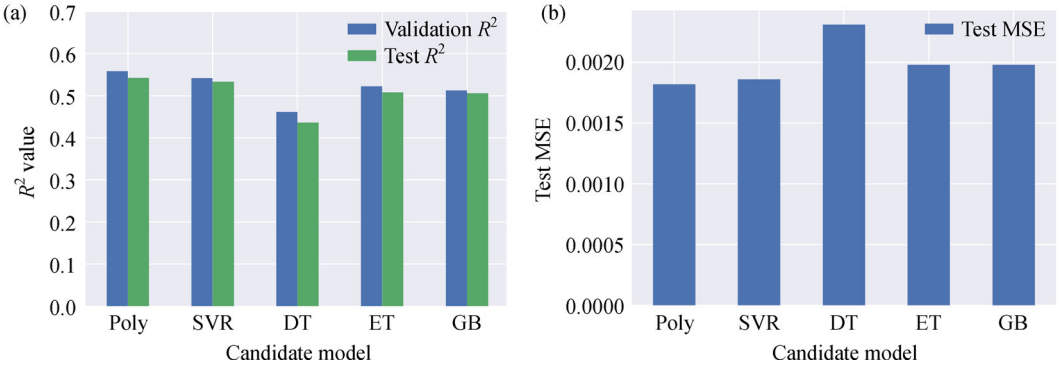

Fig. 8 (a) Validation and test R^2 scores for each candidate model; (b) test MSE score for each candidate model ("Poly" refers to the polynomial regression model).

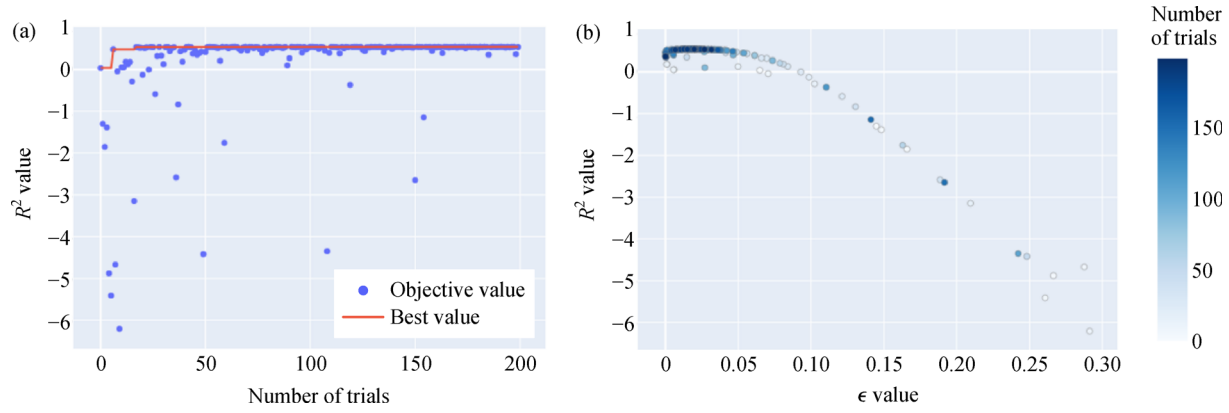

Fig. 9 (a) Optimization history for BO of SVR; (b) slice plot for ϵ hyperparameter for SVR (objective value is validation R^2).

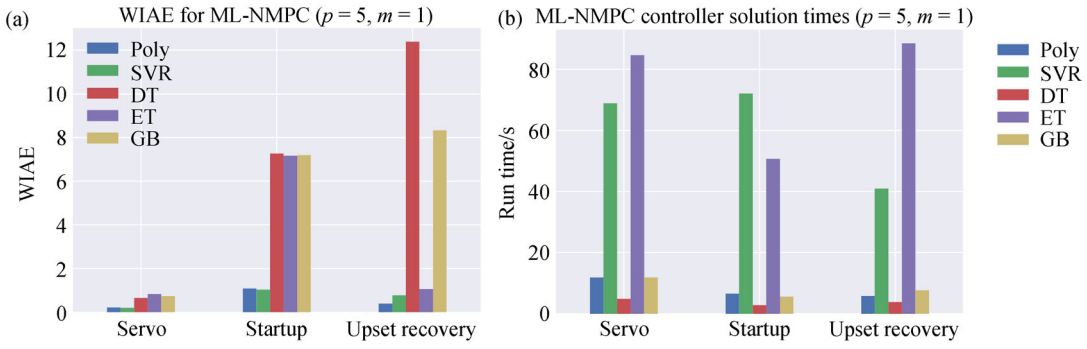

Fig. 10 ML-NMPC performance for $p = 5$ and $m = 1$: (a) WIAE, and (b) solution times.

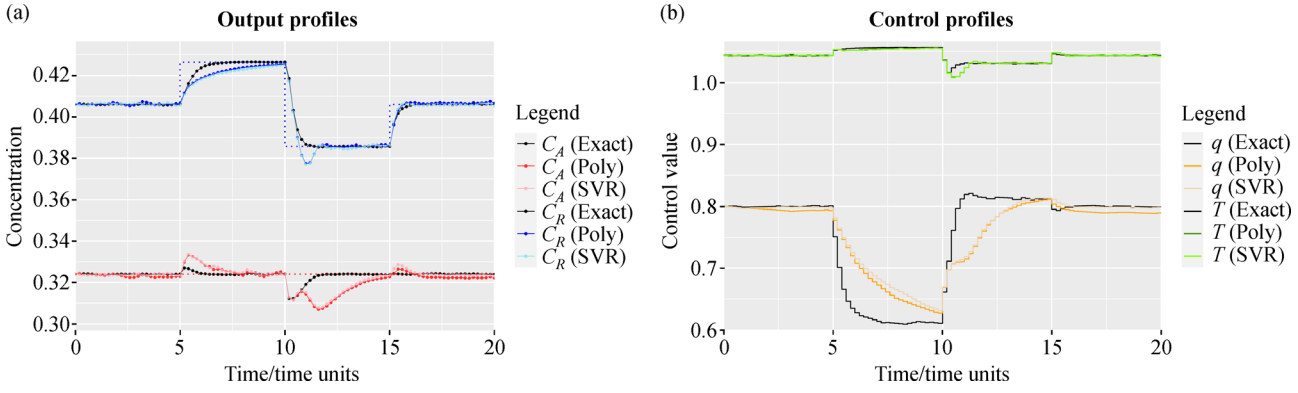

Fig. 11 Exact, polynomial- and SVR- NMPC performance for the servo problem: (a) output profiles, and (b) control profiles.

based system models. The results from the two stable ML-NMPCs, the polynomial-NMPC and SVR-NMPC, are finally benchmarked against the exact NMPC in Tables 1 and 2.

Polynomial- and SVR- NMPCs succeeded in achieving tight control for all three control problems, as Figs. 11–13 show. In the servo problem, C_A and C_R values took slightly longer to settle at their new set-point values for the two ML-NMPCs, with this due to these controllers offering q control which is less aggressive. This suggests that the quality of the continuous-time model identified from the data is sensitive to the selection of the experimental sampling time h for the data generation procedure. While $h = 0.1$ tu was sufficient to generate data that allowed the polynomial and SVR models to learn the right "signs" for the time-derivatives characterizing the dynamical system, smaller h values might offer even better first-order approximations for these time-derivatives that ML models

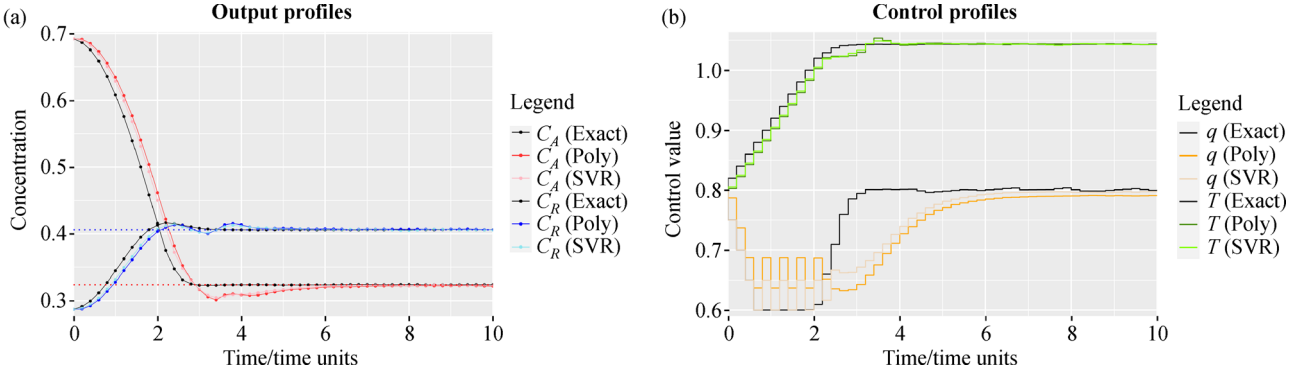

Fig. 12 Exact, polynomial- and SVR- NMPC performance for the startup problem: (a) output profiles, and (b) control profiles.

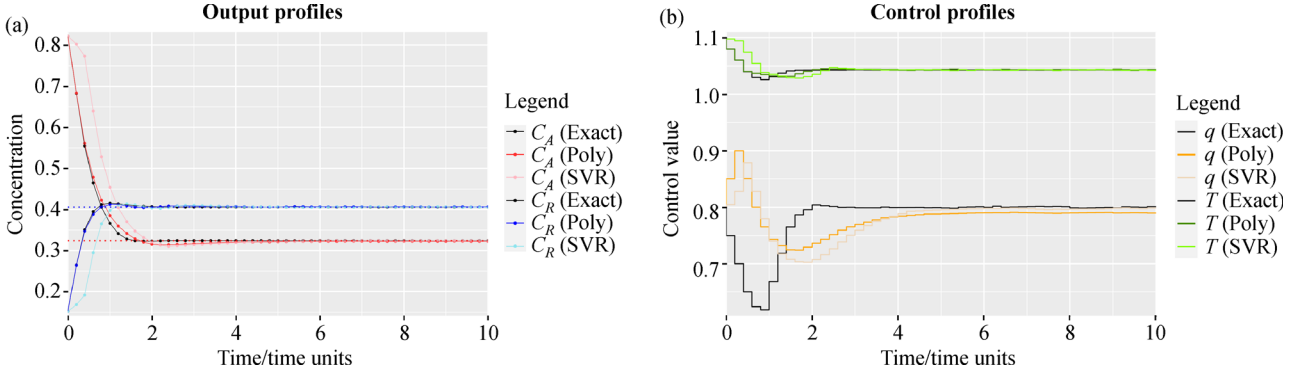

Fig. 13 Exact, polynomial- and SVR- NMPC performance for the upset recovery problem: (a) output profiles, and (b) control profiles.

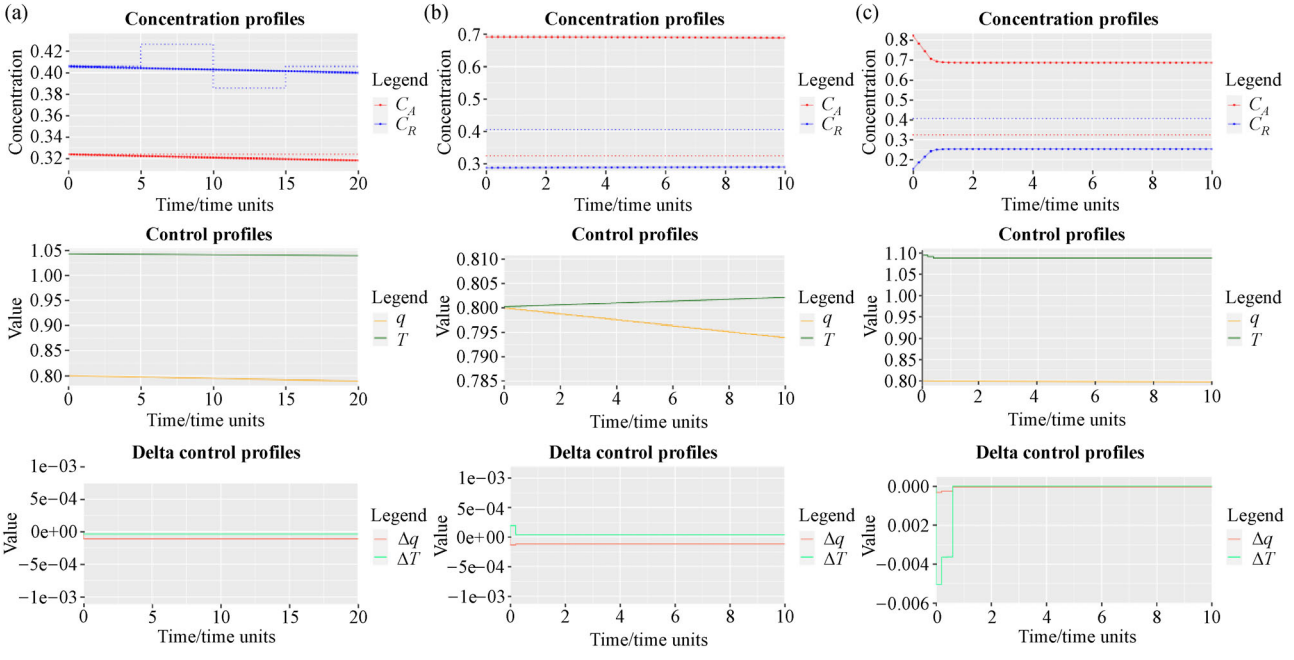

Fig. 14 GB-NMPC performance for $p = 5$ and $m = 1$: (a) servo problem, (b) startup problem, and (c) upset recovery problem.

Table 1 Comparison of WIAEs for polynomial- and SVR-NMPCs against exact NMPC for $p = 5$ and $m = 1$

Scenario	Controller performance in WIAE		
	Polynomial	SVR	Exact
Servo	0.209	0.200	0.094
Startup	1.090	1.031	0.870
Upset recovery	0.392	0.774	0.326

Table 2 Comparison of solution times for polynomial- and SVR-NMPCs against exact NMPC for $p = 5$ and $m = 1$

Scenario	Solution times in seconds		
	Polynomial	SVR	Exact
Servo	11.9	69.0	3.50
Startup	6.63	72.2	2.33
Upset recovery	5.88	41.0	1.83

could learn from, potentially allowing the ML-NMPC to solve more accurately for smaller time-steps to achieve tighter control. The control paths in the regulator problems in Figs. 12 and 13 also reveal q and T control for ML-NMPC which lagged slightly behind those for the exact NMPC.

These two ML-NMPCs required an order of magnitude more time per iteration than the exact NMPC, though they remained fast in the order of 0.1 s per iteration. The SVR-NMPC required more time than the polynomial-NMPC due to its increased model complexity, with it taking between 5 and 7 times longer while not offering any consistent improvement in WIAE performance over the polynomial-NMPC. Taking both control performance and evaluation times into account, the polynomial-NMPC would be preferable to the SVR-NMPC for this case study.

The tree-based ML-NMPCs were ineffective in control for all three control problems, as Fig. 14 shows, though an exception exists in ET-NMPC for upset recovery. Tree-based models might not be suitable as system models for ML-NMPC in this study because their piece-wise constant nature [27] could prevent optimizers like SLSQP, which depend on local gradient or Hessian approximations, from functioning well. Understanding this observation for tree-based models lies beyond the scope of this work and can be pursued in a future study.

The results above validate the ability of the proposed integrative SysID approach to identify system models for NMPC that allow effective control of a highly nonlinear MIMO CSTR system for control problems across a wider operating range. Further comments made on observations from this case study reinforce the importance of selecting a suitable h for the data generation procedure, highlight the usefulness of simpler models with shorter ML-NMPC solution times, and hint towards more advanced applications of ML that achieve better bias-variance tradeoffs through this framework. Examples illustrating the latter point include incorporating expert system knowledge to impose a prior structure on the candidate models, thereby reducing model bias. Such models are known as grey-box models. Feature selection techniques such as principal component analysis can also be incorporated along with other model regularization techniques to reduce model variance.

5 Conclusions

In this work, an integrated SysID procedure for deriving black-box nonlinear continuous-time MIMO system models for NMPC was articulated. This procedure was validated through the successful identification of NMPC system models that enabled effective control of a highly nonlinear MIMO CSTR system across a wider operating range. This framework is sufficiently flexible and allows practitioners to employ different data generation methods, apply different HO methods, as well as test different candidate models. The choices presented here reflect the current state-of-the-art or what is reasonable, based on the authors' experience.

It is hoped that these results can motivate further research in direct data-driven methodologies for SysID for MPC. Future directions include the following:

1. Applying more advanced ML techniques to this framework such as feature selection, model regularization. Other candidate models that involve local regression or ensemble methods could also be studied in greater detail. This abundance of ML techniques could be exploited to achieve better bias-variance tradeoffs and yield better system models.
2. Studying the impact of measurement noise, the amount of data available, and the experimental and NMPC controller sampling times on this SysID procedure to quantify its sensitivity to the quality of data and other design decisions.
3. Exploring how a discrete-time model could be derived from a well-learnt continuous-time model, which could serve as a fit-for-purpose NMPC system model requiring smaller solution times.

Acknowledgements The authors thank the MOE AcRF Grant in Singapore for financial support of the projects on Precision Healthcare Development, Manufacturing and Supply Chain Optimization (Grant No. R-279-000-513-133) and Advanced Process Control and Machine Learning Methods for Enhanced Continuous Manufacturing of Pharmaceutical Products (Grant No. R-279-000-541-114).

Code Availability Statement Access to the GitHub repository containing the source code for this project is available upon request.

Electronic Supplementary Material Supplementary material is available in the online version of this article at https://dx.doi.org/10.1007/s11705-021-2058-6 and is accessible for authorized users.

References

1. Kaiser E, Kutz J N, Brunton S L. Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. Proceedings—Royal Society. Mathematical, Physical and Engineering Sciences, 2018, 474(2219): 20180335
2. Sommeregger W, Sissolak B, Kandra K, von Stosch M, Mayer M, Striedner G. Quality by control: towards model predictive control of mammalian cell culture bioprocesses. Biotechnology Journal, 2017, 12(7): 1600546
3. Qin S J, Badgwell T A. A survey of industrial model predictive control technology. Control Engineering Practice, 2003, 11(7): 733–764
4. Öner M, Montes F C C, Ståhlberg T, Stocks S M, Bajtnerb J E, Sin G. Comprehensive evaluation of a data driven control strategy: experimental application to a pharmaceutical crystallization process. Chemical Engineering Research & Design, 2020, 163: 248–261
5. Al Seyab R K, Cao Y. Nonlinear system identification for predictive control using continuous time recurrent neural networks and automatic differentiation. Journal of Process Control, 2008, 18(6): 568–581
6. Ljung L. Perspectives on system identification. Annual Reviews in Control, 2010, 34(1): 1–12
7. Mokhatab S, Poe W A. Handbook of Natural Gas Transmission and Processing. 2nd ed. Boston: Gulf Professional Publishing, 2012, 473–509
8. Venkateswarlu C, Venkat Rao K. Dynamic recurrent radial basis function network model predictive control of unstable nonlinear processes. Chemical Engineering Science, 2005, 60(23): 6718–6732
9. Štampar S, Sokolič S, Karer G, Žnidaršič A, Škrjanc I. Theoretical and fuzzy modelling of a pharmaceutical batch reactor. Mathematical and Computer Modelling, 2011, 53(5-6): 637–645
10. Alanis A Y, Arana-Daniel N, López-Franco C. Artificial Neural Networks for Engineering Applications. Washington: Academic Press, 2019, 55–63
11. Pan Y, Wang J. Model predictive control of unknown nonlinear dynamical systems based on recurrent neural networks. IEEE Transactions on Industrial Electronics, 2012, 59(8): 3089–3101
12. Schoukens J, Ljung L. Nonlinear system identification: a user-oriented road map. IEEE Control Systems, 2019, 39: 28–99
13. Arefi M, Montazeri A, Poshtan J, Jahed-Motlagh M. Nonlinear model predictive control of chemical processes with a wiener identification approach. In: 2006 IEEE International Conference on Industrial Technology. Mumbai: IEEE, 2006, 1735–1740
14. Wu Z, Tran A, Rincon D, Christofides P D. Machine learning-based predictive control of nonlinear processes. Part I: theory. AIChE, 2019, 65(11): e16729
15. Wu Z, Tran A, Rincon D, Christofides P D. Machine-learning-based predictive control of nonlinear processes. Part II: computational implementation. AIChE, 2019, 65(11): e16734
16. Garnier H. Direct continuous-time approaches to system identification. Overview and benefits for practical applications. European Journal of Control, 2015, 24: 50–62
17. Frazier P I. A tutorial on Bayesian optimization. arXiv:1807.02811 [stat.ML], 2018
18. Bergstra J, Bengio Y. Random search for hyper-parameter optimization. Journal of Machine Learning Research, 2012, 13: 281–305
19. Berk J, Nguyen V, Gupta S, Rana S, Venkatesh S. Exploration enhanced expected improvement for bayesian optimization. In: Berlingerio M, Bonchi F, Gärtner T, Hurley N, Ifrim G, eds. Machine Learning and Knowledge Discovery in Databases. Cham: Springer International Publishing, 2019, 621–637
20. Seborg D E, Mellichamp D A, Edgar T F, Doyle F J III. Process dynamics and control. 3rd ed. New York: John Wiley & Sons, 2010
21. Binder B J T, Johansen T A, Imsland L. Improved predictions from measured disturbances in linear model predictive control. Journal of Process Control, 2019, 75: 86–106
22. Wong W C, Chee E, Li J, Wang X. Recurrent neural network-based model predictive control for continuous pharmaceutical manufacturing. Mathematics, 2018, 6(11): 242
23. Koppel L B. Input multiplicities in nonlinear, multivariable control systems. AIChE, 1982, 28(6): 935–945
24. Virtanen P, Gommers R, Oliphant T E, Haberland M, Reddy T, Cournapeau D, Burovski E, Peterson P, Weckesser W, Bright J, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. Nature Methods, 2020, 17(3): 261–272
25. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, et al. Scikit-learn: machine learning in Python. Journal of Machine Learning Research, 2011, 12: 2825–2830
26. Akiba T, Sano S, Yanase T, Ohta T, Koyama M. Optuna: a next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. New York: Association for Computing Machinery, 2019, 2623–2631
27. Shi Y, Li J, Li Z. Gradient boosting with piece-wise linear regression trees. arXiv:1802.05640 [cs.LG], 2019