HYBRID MODELING APPROACHES INTEGRATING PHYSICS-BASED MODELS WITH MACHINE LEARNING FOR PREDICTIVE CONTROL OF BIOLOGICAL AND CHEMICAL PROCESSES

A Dissertation

by

MOHAMMED SAAD FAIZAN BANGI

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Joseph Sang-Il Kwon
Committee Members,	Eduardo Gildin
	M M Faruque Hasan
	Costas Kravaris
Head of Department,	Victor Ugaz

August 2022

Major Subject: Chemical Engineering

ABSTRACT

Recently, there has been growing interest in data-based modeling as the amount of data available has increased tremendously. One such method is Dynamic Mode Decomposition with Control technique, which builds temporally local linear models using data. But its limited domain of applicability (DA) hinders its use for prediction purposes. To overcome this challenge, we proposed an algorithm that utilizes multiple "local" training datasets, and it was applied successfully to hydraulic fracturing. Although data-based modeling offers simplicity and ease of construction, it lacks robustness and parametric interpretability, unlike first-principles modeling.

To balance the advantages and disadvantages of data-based models and first-principles models, hybrid modeling was proposed using artificial neural networks (ANNs). Since then, Machine Learning (ML) has advanced where deep neural networks (DNNs) with more than three layers can be trained to approximate any function accurately. In this work, we proposed a deep hybrid modeling (DHM) framework that integrates first-principles with DNNs and successfully applied it to two complex processes, i.e., hydraulic fracturing and full-scale fermentation reactor. Similarly, Universal Differential Equations (UDEs) was proposed in ML where DNNs are represented as ODEs and solved using ODE solvers. We utilized UDEs to successfully build a DHM using simulation and experimental data for batch production of β-carotene. One limitation of DHM is that its DA is affected by the DNN within it, and its accuracy is high within its DA. Therefore, it is important to consider its DA when designing a model-based controller. To this end, we proposed a Control Lyapunov-Barrier Function (CLBF)-MPC to stabilize and ensure that the closed-loop system stays within DA of DHM. Theoretical guarantees were provided for the CLBF-MPC controller, and it was successfully implemented on a CSTR. The idea of integrating physics with ML can be extended to Reinforcement Learning (RL). In case when model-based controller design is not possible, we proposed a model-free Deep RL (DRL) controller that utilizes prior knowledge in its reward function to quicken the learning process. This DRL controller was successfully applied to hydraulic fracturing wherein Nolte's law was included in the reward function for fast convergence.

DEDICATION

To my teachers, friends, and family.

ACKNOWLEDGMENTS

Firstly, I am grateful to Allah for my good health, well-being, the opportunity to pursue research at Texas A&M University, and for all the innumerable blessings upon me.

Secondly, I am thankful to my advisor Dr. Joseph Sang-il Kwon for his timely help, support, and patience during the course of my study. He gave me the opportunity to take full responsibility of this work, but steered me in the right direction whenever I needed it. I cannot thank him enough for always being there to help me.

Thirdly, I would like to extend my gratitude towards my committee members Dr. Costas Kravaris, Dr. M. M. Faruque Hasan, and Dr. Eduardo Gildin for their help in guiding and reviewing my work.

Fourthly, I would like to thank Harwinder, Abhinav, Prashanth, Bhavana, Yeon-Pyeong, Parth, Ziyan, Pallavi, Silabrata, Kaiyu, Choi, Niranjan for all the stimulating discussions we had, and for their contributions. Thanks to all my friends and colleagues at Texas A&M for all the good memories that I will cherish for the rest of my life.

Lastly, I must express my very profound gratitude to my wife and parents for being an inspiration to me. I cannot thank them enough for their unconditional support and encouragement throughout this journey.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supervised by a thesis committee consisting of Dr. Joseph Sang-il Kwon [principal advisor] of the Department of Chemical Engineering and Texas A&M Energy Institute, Dr. M. M. Faruque Hasan of the Department of Chemical Engineering and Texas A&M Energy Institute, Dr. Costas Kravaris of the Department of Chemical Engineering, and Dr. Eduardo Gildin of the Department of Petroleum Engineering.

All the work conducted for the thesis was completed by the student independently.

Funding Sources

Financial support from the Artie McFerrin Department of Chemical Engineering and the Texas A&M Energy Institute are gratefully acknowledged.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1.1 Data-based modeling

Process modeling is the task of obtaining a mathematical representation for knowledge about any physical process [2]. Depending on the nature of knowledge, models can be classified into various categories. First principles or mechanistic models also known as 'white box' models are obtained using the mass and energy conservation laws, kinetic laws, thermodynamic laws, transport laws, etc. This class of models is transparent, easy-to-understand as they usually contain parameters with physical meaning, valid over a wide range of operating conditions of the process, but are complex and computationally expensive to solve. Motivated by the computationally expensive nature of first-principles models, data-based model order reduction (MOR) techniques have found traction in the field of modeling in order to develop reduced-order models (ROMs) that can represent the process dynamics approximately with a fraction of computational time. MOR techniques assume that solutions of large-scale systems can be often found in sub-spaces with dimensions smaller than that of the original system.

One of the earliest data-based subspace identification techniques is Multi-variable Output Error State Space (MOESP) which has been extensively used in the design and study of feedback controllers for hydraulic fracturing process [3, 4, 5]. Also, one of the most widely used MOR techniques is Proper Orthogonal Decomposition (POD), also known as Karhunen-Loeve analysis. In this method, spatial-temporal data is used to capture the dominant spatial patterns via a set of empirical basis functions. The method of snapshots is used to compute the basis functions, which assumes that each basis function can be represented as linear combination of the snapshots. These basis functions are then used in a projection method such as Galerkin's projection method in order to obtain low-dimensional ODEs which are an approximate representation of the original system. Additionally, temporal clustering can be incorporated in the POD-Galerkin MOR technique to construct temporally local eigen functions, which are then used to obtain a set of temporally lo-

cal ROMs with each of them being represented by low dimensional ODEs. These set of ODEs together represent the complex hydraulic fracturing process and have been shown to be more accurate when compared to their temporally global counterparts [6]. Unlike POD that captures the dominant spatial patterns that carry most of the flow energy in a dynamic system, Dynamic Mode Decomposition (DMD), a recently developed ROM technique, captures those spatial patterns that contribute to the long-term dynamics of the system [7]. This technique of DMD has been extended to control applications in the form of DMDc (DMD with control) which considers the manipulated inputs to extract the underlying dynamics from the measurements of a system. In ROM context, DMDc provides a linear model that best represents the underlying dynamics [8]. But for highly non-linear and complex systems like hydraulic fracturing, a linear relationship is not an accurate representation. Therefore, local DMDc was developed which involves temporal clustering of the spatial-temporal data of hydraulic fracturing system and building temporally local DMDc-based ROM for each cluster. It has been shown that together, these local ROMs are an accurate representation of the hydraulic fracturing system [9, 10]. Alternatively, building linear models that are valid over a larger domain is possible through Koopman operator theory which states that any finite-dimensional nonlinear system can be represented linearly in the space of all possible functions of the system states. In this infinite-dimensional space, the evolution of the system is governed by an infinite-dimensional linear operator called the Koopman operator. A finite approximation of the Koopman operator can be calculated using Extended DMD (EDMD) [11]. This concept of EDMD was successfully extended to design Koopman-based ROMs for control purposes [12, 13, 14] and used them in the feedback control of fracture geometry and spatial proppant concentration profiles in hydraulic fracturing process [15].

Another data-driven model identification method is Sparse identification of nonlinear dynamics (SINDy) which identifies a mathematical model describing the underlying dynamics of a nonlinear process. Specifically, SINDy uses sparse regression techniques to select significant functions relevant to the process model from an over-full library of candidate functions. Consequently, the model identified using SINDy is sparse and physically interpretable. Because of its simplicity, the SINDy

algorithm has been implemented in various process safety and control applications [16, 17, 18, 19]. These MOR and model identification techniques fall in the category of data-driven models or 'black box' models, and are computationally inexpensive to solve, but are usually difficult to interpret as the nature of parameters is unknown, and have narrow domain of applicability. To overcome this challenge, hybrid models or 'grey box' models have been developed which are a combination of white box and black box sub-models [20].

1.2 Hybrid modeling

In process modeling, the concept of hybrid (grey box) models evolved from the field of neural networks [21, 22, 23, 24]. The idea was to build neural network based hybrid models through the use of first principles knowledge. This resulted in hybrid models with better prediction accuracy compared to the first principles models, and better interpolation, extrapolation, and interpretation compared to solely neural networks based models. There exist other kinds of grey box models that combine different types of first principles knowledge and/or empirical submodels. During the 90s, the term 'grey box' models appeared in systems and control theory wherein the structural information from the first principles models was incorporated into the data-based models [25, 26, 27]. But understanding of the term 'grey box' has evolved to represent all types of hybrid models that combine first principles and data-based submodels. Nonetheless, hybrid modeling balances the advantages and disadvantages of strictly first principles and data-based modeling, and offers desirable benefits such as high prediction accuracy, better extrapolation capabilities, ease of calibration, and better interpretability. For these reasons, hybrid modeling has numerous applications in chemical and biochemical engineering. For instance in modeling of chemical reactor [28, 29, 30, 31], polymerization processes [32, 33], crystallization [34, 35], metallurgic processes [36, 37, 38], distillation columns [39, 40], drying processes [41], thermal devices [42], mechanical reactors [43], milling [44, 45], modeling of yeast fermentations [46, 47, 48], modeling of fungi cultivations [49, 50], modeling of bacteria cultivations [51, 52], modeling of mammalian cell cultivations [53, 54], modeling of insect cell cultivations [55], modeling of hybridoma cell

cultivations, [56, 57], etc. For more information one can view [58, 59], excellent review papers on hybrid modeling in the field of process systems engineering. Additionally, to understand the path forward and the challenges that lie ahead in chemical engineering related to integrating data-driven processing with scientific reasoning, one can review this excellent review paper [60].

Recall, hybrid modeling started in 1992 from the use of neural networks along with first principles knowledge [21, 22, 23, 24]. Neural networks are connectionist models that map an input space to an output space and were inspired by the biological networks present in the brain. Each neural network contains elements called 'neurons' or 'nodes' that process an input and give an output, and comprises of layers containing many such nodes. Each node in a particular layer is connected to every node in an adjoining layer, and the strength of each connection is represented by an assigned weight. Consequently, a node in a particular layer receives inputs from all the nodes in the previous layer which are added together after weights are applied on each of them, and a 'bias' is added to the sum. If the neural network only contains weights and biases, then it behaves as a linear function. In order to capture nonlinearity in the input-output data, certain nonlinear functions called 'Activation functions' are used in each node. With activation functions, the input to the neural network undergoes nonlinear transformation through each layer and is collected as the output at the final layer. Overall, the number of nodes, number of layers, weights in each connection, and the biases are the internal parameters of the neural network. Under certain assumptions, neural networks, if sufficiently large, have been shown to capture any nonlinear continuous function accurately [61]. With the advancements in Machine Learning, the field of neural networks has evolved from the use of a single hidden layer to multiple hidden layers resulting in deep neural networks.

1.3 Organization and objectives of the proposed research

The growing availability of data provides a tremendous opportunity to find novel ways to integrate deep neural networks with existing process knowledge in the context of modeling and control. To this end, the overall objective of this doctoral study is to develop mathematical frameworks

which combine deep neural networks with available process knowledge for hybrid modeling, hybrid model-based predictive control and Reinforcement Learning-based control. Sections 2, 3, 4, and 5 outline the results obtained by utilizing the proposed deep hybrid modeling, hybrid model-based predictive control, and DRL (deep reinforcement learning) control frameworks.

In recent years, there has been a lot of interest in data-based modeling as the amount of data measured and stored has increased tremendously, and the resulting data-based models are simple and easy to construct. Some of the data-based modeling methods that are widely used are Dynamic Mode Decomposition (DMD), DMD with control (DMDc), Proper Orthogonal Decomposition (POD), neural networks, etc. One such method is the Local Dynamic Mode Decomposition with Control (LDMDc) technique, which builds temporally local linear models using data only. But the limited domain of attraction (DOA) of LDMDc hinders its widespread use for prediction purposes. To systematically enlarge the DOA of the LDMDc technique, in Section 2, a mathematical framework was developed that utilizes multiple "training" data-sets, implements a clustering strategy to divide the data into clusters, uses DMDc to build multiple local models, and implements the k-nearest neighbors technique to make a selection among the set of local models during prediction. The proposed algorithm was applied successfully to hydraulic fracturing.

Although data-based modeling offers simplicity and ease of construction, it lacks robustness and parametric interpretability, unlike modeling based on first-principles. To balance the advantages and disadvantages of data-based models and first-principles models, hybrid modeling was proposed using artificial neural networks (ANNs). ANNs are connectionist models containing three layers with multiple neurons in each of them and are widely used as function approximators. Since the introduction of ANNs for hybrid modeling, the field of Machine Learning has advanced where deep neural networks (DNNs) with more than three layers can be efficiently trained to approximate any function accurately. In Section 3.1, a deep hybrid modeling framework is developed using DNNs and Levenberg-Marquardt training algorithm, and is successfully applied to build a deep hybrid model for hydraulic fracturing. In the hydraulic fracturing, the unknown process parameters, i.e., fluid leak-off rates were predicted by the DNN and then utilized by the

first-principles model to calculate the hybrid model outputs. This deep hybrid model was easier to analyze, interpret, and extrapolate compared to a data-based model, and showed higher accuracy compared to the first-principles model.

In Section 3.2, to further prove the efficacy of the proposed deep hybrid modeling framework, it was utilized to model a full-scale bio-fermentation reactor with a volume of over 100,000 gallons. The resulting deep hybrid model was more accurate and robust than the (original and improved) first-principles models, and it captured unknown time-varying dependencies among parameters.

Similar to the concept of deep hybrid modeling, Universal Differential Equations (UDEs) was proposed in the field of Machine Learning. The concept of UDEs is a natural extension to another concept called Neural ODEs wherein neural networks are represented as ODEs and solved using ODE solvers. Now, in UDEs, neural networks are integrated with any physics-based knowledge of the system in the form of differential equations, and the resultant UDE model is solved using appropriate ODE solvers. This concept of UDEs was utilized to build a hybrid model for the batch production of β-carotene in two scenarios. In Section 3.3, theoretical assumptions about the unknown dynamics in the existing kinetic model of this process were made, and it was showed that a trained UDE is able to accurately capture these dynamics which were assumed to be unknown initially. In Section 3.4, a UDE model was built for the lab-scale batch production of β-carotene. The existing kinetic model of this process suffers from poor accuracy when compared to the experimental data. Therefore, the concept of UDE was build to build a superior hybrid model. It was showed that the trained UDE model outperforms the existing kinetic model, both in training and testing scenarios.

Integrating physics with DNNs either in a modeling or a control framework comes with few limitations. One such limitation in the case of the deep hybrid modeling framework is that the Domain of Applicability (DA) of the deep hybrid model is greatly influenced by the capabilities of the DNN contained within it. Subsequently, the accuracy of the deep hybrid model is high within its DA and vice versa. Therefore, it is important to take into account the DA of the deep hybrid model when using it to design a model-based controller. In Section 4, a Control Lyapunov-Barrier

Function (CLBF)-based MPC was developed that combines a Control Lyapunov Function (CLF) and a Control Barrier Function (CBF) to stabilize as well as ensure that the closed-loop states stay within the DA of the deep hybrid model where its prediction accuracy is high. Theoretical guarantees were provided on the performance of this CLBF-MPC controller, and it was successfully implemented on a CSTR system.

The idea of integrating prior knowledge about a process with a Machine Learning framework can also be extended to Reinforcement Learning-based process control. Reinforcement Learning (RL) is a field within Machine Learning that deals with designing data-based controllers which learn an optimal control policy by directly utilizing data from the process. In Section 5, a Deep Reinforcement Learning (DRL) controller was developed based on the actor-critic approach that directly utilizes prior knowledge about the process in its reward function formulation to quickly learn an optimal control policy. This DRL control framework was successfully applied to hydraulic fracturing process wherein prior knowledge about Nolte's power law pumping schedule was included in the form of a reward function to obtain uniform proppant concentration along the length of the fracture at the end of the proppant injection process. The inclusion of Nolte's law helped the DRL controller quickly reach convergence to an optimal policy. In Section 6, a brief summary and conclusion of the doctoral research was provided, followed by list of references cited in this doctoral study.

2.1 Enlarging the Domain of Attraction of the Local Dynamic Mode Decomposition with Control Technique: Application to Hydraulic Fracturing*

Many chemical processes are usually represented by high-dimensional complex models which accurately describe the dynamics of the system but their utility in the design of feedback control systems is limited due to the model complexity which puts considerable strain on the computational resources. Nonetheless, the solutions of such large-scale complex systems can be approximately explained by a very specific set of low-dimensional equations. For example, only three ODEs were required to represent the essential features of a laminar fluid flow passing a 2D cylinder [62]. Many model order reduction techniques are based on this idea and they have been widely used in industrially important engineering problems to deal with high-dimensional models without losing much accuracy. One ROM technique is network-based wherein complex chemical systems are divided into a network of small units, the characteristics of each unit can be defined by very few ordinary differential equations, and solving these equations for each unit would give a distribution of properties such as mass, energy, and momentum. More recently, this network-based ROM technique has been applied to gasifier which is represented as a network of ideal reactors consisting of plug flow reactors (PFRs) and continuous stirred tank reactors (CSTRs) [63, 64, 65]. Two of the commonly used modal decomposition techniques in model order reduction methods are Proper Orthogonal Decomposition (POD) and Dynamic Mode Decomposition (DMD). Both of these techniques extract coherent structures within the system by analyzing sequential data obtained either by simulation of the high-fidelity model of the high-dimensional system, or obtained

*Reprinted with permission from "Enlarging the Domain of Attraction of the Local Dynamic Mode Decomposition with Control Technique: Application to Hydraulic Fracturing" by Bangi, M. S. F., Narasingam, A., Siddhamshetty, P. and Kwon, J. S. 2019. Ind. Eng. Chem. Res., 58, 5588-5601, Copyright 2019 American Chemical Society.

via experimental studies. POD technique extracts structures that capture the most energy [66] and can be used to build a ROM for the system.The POD technique has been applied to build ROMs for various applications [67, 68, 69, 70, 71, 72, 6, 73]. But using energy as a criterion for identifying these coherent structures is not always useful as it ignores those structures with zero-energy but are dynamically relevant [74].

DMD was initially introduced in the fluid community to extract flow structures by observing the high-dimensional data that can accurately represent the dynamics of the flow [75]. In comparison to POD, this method extracts those structures that are dynamically relevant and contribute towards the long-term dynamics of the system [76] rather than selecting those that carry the most energy. Mathematically, DMD assumes that nonlinear systems with complex models can be represented using a linear form and this may seem inaccurate at first but understanding DMD as a numerical approximation of Koopman spectral analysis has validated this representation [77, 78, 79]. DMD has been successfully applied to both numerical [7, 80, 81, 82, 83] and experimental [84, 85, 86, 87, 88, 89, 90] fluid flow data to represent relevant physical mechanisms in a linear form. Many works have been carried out regarding the numerics of the DMD algorithm which include the development of memory efficient algorithms [91, 92], a method for selection of a sparse basis of DMD modes [93], and an error analysis of DMD growth rates [90]. Apart from this, theoretical works have been carried out to explore and understand its relationship with other methods such as Fourier analysis [94], POD [7], and Koopman spectral analysis [77, 78, 79]. Also, different methods have been proposed as variations of DMD such as Optimized DMD [94] and Optimal Mode Decomposition [95, 96].

Within this context, DMDc, a purely data-driven modal decomposition technique, was developed to represent nonlinear systems, especially those whose dynamics are influenced by external inputs, in a discrete state-space form by extracting dynamically relevant spatial structures using both measurements of the system and the external inputs applied on it [8]. DMDc provides an understanding of the input-to-output behavior, which can be utilized to predict and design feedback control systems. However, for a highly nonlinear system, a global linear representation might

not be a good approximation considering the fewer degrees of freedom associated with the linear model. Because of this limitation, the global method may fail to capture the effect of the changes in the process parameters such as permeability and Young's modulus of the rock formation on the local dynamics in the case of hydraulic fracturing as these constants are space-dependent. In order to better capture the local dynamics, temporal clustering can be integrated to DMDc to develop local ROMs that better represent the dynamics of the overall system and this technique was introduced as LDMDc [9]. Local DMDc divides the snapshot data into different clusters and for each cluster it obtains a pair of linear operators which together represents a local ROM for their corresponding cluster. The local model will approximately explain the dynamics of the system under the conditions in which the snapshots belonging to that particular cluster were obtained. It has been shown that LDMDc performs better than global DMDc when a single data set obtained under a particular operating condition is used to build the models [9]. However, the drawback of these models is that their domain of attraction (DOA) is limited by the data used for model training; in other words, these models will perform poorly when used for prediction under other operating conditions.

Our contribution in this work is to enlarge the DOA of LDMDc technique by implementing supervised and unsupervised learning techniques on multiple 'training' data sets to build and utilize multiple local ROMs, respectively. These data sets are obtained under different operating conditions by performing simulations of the high-fidelity model. In order to obtain highly-accurate local ROMs, we implement a particular clustering strategy instead of the conventional approaches available in the literature. The clustering strategy involves considering each 'training' data set individually and clustering it using only the information of the inputs such that the optimal number of clusters are obtained along with the clustered output. The reason we opted for the clustering strategy is that it is easier to implement, and the resulting clustered output satisfies constraints required to build highly accurate ROMs which will be discussed later in this text. Using the clustered output, LDMDc-based ROMs are built and these ROMs will be used for prediction. During prediction, at any instance, the selection of a ROM is accomplished by utilizing the k-nearest neighbors (kNN) classification technique. Another novelty is that in our proposed algorithm, we utilize both

the states of the system and the external inputs applied on it which is necessary because the dynamics of the system are influenced by both the state and the applied external input. This aspect of our algorithm makes it different from the LDMDc technique proposed by Narasingam and Kwon [9] wherein only the states of the system were utilized to cluster the data. Also, in the LDMDc technique proposed by Narasingam and Kwon [9] selection of ROM was not necessary as only one 'training' input was used to build the model. On the contrary, in our proposed algorithm any input profile satisfying an imposed constraint within the enlarged DOA can be utilized for prediction and this necessitates the use of the kNN classification technique. Despite the differences, our proposed technique still holds the advantages of the LDMDc technique proposed by Narasingam and Kwon [9] in that it is completely data-driven and captures local dynamics efficiently all while requiring no knowledge in terms of the system model.

2.1.1 Local Dynamic Mode Decomposition with Control

Recall, the technique of DMDc represents the underlying dynamics of a nonlinear system in a linear state-space form by utilizing both the measurements of the system and the external input applied on it. Mathematically, this would mean that the snapshots are related to each other by a linear operator pair. But considering that most of the systems are inherently nonlinear, this linear representation will not be accurate. To accurately represent a nonlinear system using DMDc, Narasingam and Kwon [9] proposed a framework to divide the snapshots into clusters wherein in each cluster its underlying local dynamics can be captured and be represented in a linear form by using DMDc on the snapshots within that cluster.

2.1.1.1 Capturing local dynamics

DMDc is applied to each cluster using its corresponding snapshots to obtain a linear operator pair $\mathbf{A}$ and $\mathbf{B}$ to build a ROM that will capture the cluster's underlying local dynamics. Algorithm below describes how to apply DMDc to each cluster and obtain the pairs $(\mathbf{A}_j, \mathbf{B}_j)$ for every cluster in detail. The ROM to describe the dynamics of the system in each cluster can be formulated as

Algorithm 1 DMDc for each cluster

1: Suppose the j^{th} cluster contains the states $\{\mathbf{x}_1\ \mathbf{x}_2\ \dots\ \mathbf{x}_n\}_j$ and the corresponding inputs applied on them are $\{\mathbf{u}_1\ \mathbf{u}_2\ \dots\ \mathbf{u}_n\}_j$. Arrange the data matrix $\{\mathbf{x}_1\ \mathbf{x}_2\ \dots\ \mathbf{x}_n\}_j$ into matrices $\mathbf{X}$ and $\mathbf{Y}$ such that

$$\mathbf{X} = \{\mathbf{x}_1\ \mathbf{x}_2\ \dots\ \mathbf{x}_{n-1}\}_j,\ \mathbf{Y} = \{\mathbf{x}_2\ \mathbf{x}_3\ \dots\ \mathbf{x}_n\}_j$$

where $\mathbf{X} \in \mathbb{R}^{s\times(n-1)}$ and $\mathbf{Y} \in \mathbb{R}^{s\times(n-1)}$. The corresponding inputs for the states in $\mathbf{X}$ be $\mathbf{\Gamma}$ such that

$$\mathbf{\Gamma} = \{\mathbf{u}_1\ \mathbf{u}_2\ \dots\ \mathbf{u}_{n-1}\}_j$$

where $\mathbf{\Gamma} \in \mathbb{R}^{l\times(n-1)}$

2: The state space form corresponding to this cluster can be defined as

$$\mathbf{Y} = \mathbf{A}_j * \mathbf{X} + \mathbf{B}_j * \mathbf{\Gamma}$$

where $\mathbf{A}_j \in \mathbb{R}^{s\times s}$ and $\mathbf{B}_j \in \mathbb{R}^{s\times l}$

3: The equation can be rewritten in an augmented form as

$$\mathbf{Y} = [\mathbf{A}_j\ \mathbf{B}_j] * \begin{bmatrix}\mathbf{X}\\ \mathbf{\Gamma}\end{bmatrix} = \mathbf{G} * \mathbf{\Omega}$$

where $\mathbf{G} \in \mathbb{R}^{s\times(s+l)}$ and $\mathbf{\Omega} \in \mathbb{R}^{(s+l)\times(n-1)}$

4: Compute the Singular Value Decomposition (SVD) of the augmented matrix $\mathbf{\Omega}$ as

$$\mathbf{\Omega} = \hat{\mathbf{U}}\ \hat{\mathbf{\Sigma}}\ \hat{\mathbf{V}}^*$$

5: Reduce the order of the augmented system $\mathbf{\Omega}$ from '$s + l$' to 'p' by selecting an appropriate tolerance limit for the singular values of $\mathbf{\Omega}$

6: Compute $\hat{\mathbf{U}}_1^* \in \mathbb{R}^{s\times p}$, and $\hat{\mathbf{U}}_2^* \in \mathbb{R}^{l\times p}$ such that

$$\hat{\mathbf{U}} = \begin{bmatrix}\hat{\mathbf{U}}_1\\ \hat{\mathbf{U}}_2\end{bmatrix}$$

7: Compute the SVD of the shifted snapshot sequence $\mathbf{Y}$ as

$$\mathbf{Y} = \tilde{\mathbf{U}}\ \tilde{\mathbf{\Sigma}}\ \tilde{\mathbf{V}}^*$$

8: Reduce the order of the subspace of $\mathbf{Y}$ from 's' to 'r' by selecting an appropriate tolerance limit for the singular values of $\mathbf{Y}$. Here, $\tilde{\mathbf{U}} \in \mathbb{R}^{s\times r}$, $\tilde{\mathbf{\Sigma}} \in \mathbb{R}^{r\times r}$, and $\tilde{\mathbf{V}} \in \mathbb{R}^{(n-1)\times r}$

9: Compute the low dimensional representation of the system matrices as

$$\hat{\mathbf{A}}_j = \tilde{\mathbf{U}}^*\ \mathbf{Y}\ \hat{\mathbf{V}}\ \hat{\mathbf{\Sigma}}^{-1}\ \hat{\mathbf{U}}_1^*\ \tilde{\mathbf{U}} \qquad \hat{\mathbf{B}}_j = \tilde{\mathbf{U}}^*\ \mathbf{Y}\ \hat{\mathbf{V}}\ \hat{\mathbf{\Sigma}}^{-1}\ \hat{\mathbf{U}}_2^*$$

where $\hat{\mathbf{A}}_j \in \mathbb{R}^{r\times r}$ and $\hat{\mathbf{B}}_j \in \mathbb{R}^{r\times l}$

10: Apply the inverse transformation to project the approximate system matrices, $\hat{\mathbf{A}}_j$ and $\hat{\mathbf{B}}_j$, in the r-dimensional subspace to the full order space

$$\mathbf{A}_j = \tilde{\mathbf{U}}\ \hat{\mathbf{A}}_j\ \tilde{\mathbf{U}}^* \qquad \mathbf{B}_j = \tilde{\mathbf{U}}\ \hat{\mathbf{B}}_j$$

where $\mathbf{A}_j \in \mathbb{R}^{s\times s}$ and $\mathbf{B}_j \in \mathbb{R}^{s\times l}$

follows:

$$\mathbf{x}_{i+1} = \mathbf{A}_j * \mathbf{x}_i + \mathbf{B}_j * \mathbf{u}_i \tag{2.1}$$

The above equation represents the system as a discrete-time linear state space model. Therefore, we recognize that DMDc can be used for system identification of a high dimensional, nonlinear system as a linear state-space model by capturing its underlying dynamics using the data obtained from its high-fidelity model.

2.1.2 Enlarging the DOA of Local DMDc

As mentioned previously, the DOA of the LDMDc technique proposed by Narasingam and Kwon [9] is narrow with respect to both the input and the state space, meaning that the model built using this technique can only reproduce accurately the 'training' data when the 'training' input is applied and applying any other input on the model will produce unsatisfactory results. In this work we use a variety of operating conditions to obtain the 'training' data which would then be used to enlarge the DOA of LDMDc.

Suppose $\mathbb{X}_h$ is the trajectory followed by the state $x \in \mathbb{R}^s$ when an input $u \in \mathbb{R}^l$ is applied on the high fidelity model, and $\mathbb{X}_r$ is the trajectory followed by the state $x_r \in \mathbb{R}^s$ when the same input $u \in \mathbb{R}^l$ is applied on the reduced-order model obtained from the proposed algorithm. Then,

$$\begin{aligned}
\mathbb{X}_h &:= \{x \in \mathbb{R}^s : \dot{x}(t) = f(x(t), u(t)) \quad s.t. \quad x(0) = 0, u \in \mathbb{R}^l, \forall t \in [0, t_e]\} \\
\mathbb{X}_r &:= \{x_r \in \mathbb{R}^s : x_r(k+1) = A_k * x_r(k) + B_k * u(k)) \quad s.t. \\
&\qquad\qquad x_r(0) = 0,\ A_k \in A,\ B_k \in B,\ u \in \mathbb{R}^l,\ \forall k \in [0, k_{t_e}]\}
\end{aligned} \tag{2.2}$$

where f represents the high-fidelity model as a function of state x and input u, t_e is the end of fracturing time, and A_k and B_k are the linear operator pairs obtained by using the 'training' data. The DOA $\mathbb{D}$ is then defined as

$$\mathbb{D} := \{x, x_r \in \mathbb{R}^s : \mid x - x_r \mid < \epsilon\} \tag{2.3}$$

where ϵ is the error. In other words, the DOA is the set of all states obtained from the reduced-order models which can satisfactorily describe the system under well-defined conditions.

2.1.2.1 Data Generation

The 'training' data can be obtained by implementing various input profiles on the system either on an experimental basis or by carrying out simulations of the high fidelity model. In this work we choose the latter option for data generation. An important point to remember here is that it is very crucial to identify the DOA for which the model is intended to be built, and to select inputs within this region. To identify this region, it is necessary to understand the application of the ROM. One of the important applications of ROMs is in the design of controllers and one of the most widely used control techniques in these days is an optimization-based control scheme to obtain the optimal control action. Therefore, it would be ideal to select a region which would contain the solution (optimal control action) to the optimization (control) problem. Also, these multiple inputs need to be spread all across this finite region to make sure that the 'training' data is 'rich'. Consequently, the resultant model will be able to accurately predict for any input under a constraint, which will be discussed later, within this finite region. Finally, the number of 'training' inputs to be used is up to the discretion of the users. Large amounts of data would definitely improve the accuracy of the model but this would come at the cost of high computational expenses. Having a priori knowledge of the system, and an understanding of the application of the model will help in deciding the number of 'training' inputs necessary to build the model. To summarize, the following guidelines should be taken into account when defining the 'training' inputs.

1. To identify the region, information from the existing literature/experimental studies must be considered along with other system and practical constraints.

2. The selected 'training' inputs should cover the entire identified input region to maximize the predictive capability of the reduced-order model.

3. To reduce the computational expenses, the 'training' inputs should be unique but within the defined region.

Once the 'training' inputs have been identified, 'training' data sets can be generated by performing numerical simulations of the high-fidelity model. Assuming that 'd' 'training' data sets are generated and each data set contains 'n' snapshots, the nomenclatures used to represent the state and input matrices are shown in [97].

2.1.2.2 Temporal Clustering

Recall, the LDMDc technique divides the generated snapshots of data temporally into clusters. The GOS algorithm was used to partition the snapshots into clusters and it was sufficient to use only the state vectors as only one 'training' input profile was used to obtain a ROM via Local DMDc [9]. But in this work, since multiple 'training' inputs are considered, we propose to use both the state vector and the input by stacking them vertically to form an 'augmented' vector, and these 'augmented' vectors will now constitute our 'training' data sets. Before applying any clustering technique, it is essential to normalize the data as the components of the 'augmented' vector operate in two different spaces (i.e., the state space and the input space) and the range of each component varies with the other. To perform normalization, we first concatenate all the 'augmented' vectors horizontally to form one 'combined' data matrix. To further understand this 'combined' data matrix, its rows represent the components of the 'augmented' vector and its columns represent the time instances. Each row of the 'combined' data matrix must be normalized individually across all the columns of the 'combined' data matrix. The nomenclatures used to represent above-mentioned matrices are shown in [97]. In Algorithm 1, the state vectors and the input vectors have been defined to contain s and l components respectively. As a result, the 'augmented' vector contains $s + l$ components. Therefore, the normalization process is repeated for the entire $s + l$ rows in the 'combined' data matrix.

A point to consider here is the dimensions of the state vector and the input vector. It is usually the case where the dimension of the state vector is much larger compared to the dimension of the input vector. Furthermore, considering that clustering algorithms typically use 'distance' as a metric based on which snapshots are divided into clusters, it is quite possible that the contribution of the state vector towards this metric might numerically outweigh the contribution of the input

vector. To overcome this imbalance, we propose to apply two weights on the 'augmented' vector, wherein one weight is equally divided among all the components of the 'augmented' vector that represent the state vector, and similarly, the other weight is applied on the input component of the 'augmented' vector. Numerically, these weights have to be ascertained by trial and error as they depend on the system, and the type of 'training' inputs used to generate 'training' data to build the model. We then perform Principal Component Analysis (PCA) on the weighted matrix. PCA helps in reducing the order of the model which helps in minimizing the number of dimensions that we have to deal with when clustering the data, validating the model, and using the model for prediction. Applying PCA transforms the weighted matrix into its PCA scores (PCSs) which represent the data in the principal component space, and we will only use those scores whose corresponding components can together be used to represent at least 90% of the variance in the data. Now, we have the necessary transformed data to implement the clustering strategy.

The clustering strategy to be implemented is highly dependent on the 'training' data used, and the application in which the resultant model will be used. Nonetheless, it should satisfy the following output criteria: (a) no two data points from two different 'training' data sets will lie in the same cluster, (b) no two data points with different inputs will lie in the same cluster. The reasons for the above criteria are that when snapshots belonging to different 'training' data sets, or belonging to the same 'training' data set but having different inputs are kept in the same cluster, the resultant operator pair (A, B) will capture the local dynamics inaccurately for that cluster and this linear operator pair will give inaccurate results when used for prediction. Conventional clustering techniques can be applied but it would be much easier to satisfy the above mentioned criteria by implementing a clustering strategy which involves considering each 'training' data set individually and clustering based on the inputs such that the optimal number of clusters is obtained along with the clustered output. Once clustering is done, the cluster centers can be calculated in terms of the PCSs by calculating the average of all the PCSs of the data points within each cluster, and these centers will be used in the selection of the local ROM which is explained in the section below.

2.1.2.3 Local ROM Selection

After building a ROM for each cluster, we use this set of local ROMs for validation and for model prediction. In both the cases we adopt the same approach to select the appropriate local ROM. Recall, at a given time instance, the future state of the system is dependent on both the current state of the system and the input to be applied on it. Hence, we will use both the information in the selection of the appropriate ROM.

At this stage it is important to understand that each local ROM is developed for a cluster of state vectors and their corresponding inputs. At any time instance, given the state and the input, we need to find a snapshot in the 'combined' matrix whose state vector and the input closely match with the ones in consideration. Next, we locate the cluster in which this selected snapshot belongs to and use that particular cluster's ROM to predict the future state for an applied input profile. But finding the closest snapshot in the 'combined' matrix is a computationally expensive task considering the huge amounts of data in use. Instead it would be much easier to find the nearest cluster center to both the state vector and the input in consideration and use the corresponding local ROM to predict the future state trajectory.

Considering that the state vector and the input operate in two different spaces, it is difficult to make a selection of which ROM to use without an appropriate transformation. To overcome this, we apply the transformation similar to the one used in the clustering step of our algorithm. We first stack them vertically to form the 'augmented' vector, normalize each row of the 'augmented' vector using the corresponding mean and variance of that row in the 'combined' matrix obtained in the clustering step, and apply weights on the state vector and the input in the exact manner as done in the clustering step. Recall, in the clustering step, we perform PCA on the weighted matrix to reduce the number of dimensions we have to deal with in various steps of our algorithm, which includes the selection of the local ROM. Considering that we transformed the 'training' data into PCSs of its dominant PCA components in the clustering step, we similarly calculate the PCS of the weighted vector in consideration by using the same dominant PCA components. Now the transformation of the state vector and the input in consideration is complete and the above

calculated PCSs will be further used in the ROM selection step.

We use kNN technique to select the appropriate local ROM. Recall, kNN technique selects 'k' points from a data set that are closest to the query point with respect to the Euclidean distance metric. In our method the above calculated dominant PCSs of the weighted vector in consideration is the query point. The set of cluster centers given to the kNN technique as the input data set will not comprise all the cluster centers. Using all the cluster centers will result in the incorrect selection of the local ROM because it is entirely possible that there exists two 'augmented' vectors whose states and inputs are dissimilar respectively but may have approximately the same PCSs. To avoid such scenarios we include the following constraint in our algorithm: at the given time instance, those cluster centers are selected in the subset of cluster centers to the kNN technique whose respective clusters contain the snapshot having the same time instance. Implementing the kNN technique with the above constraint will help the algorithm in selecting the correct local ROM. Given a $(\mathbf{x}_i, \mathbf{u}_i)$ at the time instance $\mathbf{t}_i$, the right pair of $(\mathbf{A}_j, \mathbf{B}_j)$ can be selected and the next state of the system can be calculated as defined in Eq. (2.1). The proposed method is summarized in Algorithm 2.

2.1.3 Application to hydraulic fracturing

Shale gas is natural gas trapped within rocks of low porosity and low permeability, and hydraulic fracturing is a technique to obtain shale gas by stimulation of such rocks by controlled explosions along the length of the wellbore resulting in the formation of fractures. A clean fluid called pad is then introduced inside the wellbore at high pressures to extend the length of the initial fractures. A fracturing fluid containing water, proppant and additives is then introduced to further extend the fractures. Once pumping is stopped, the remaining fluid is allowed to leak off into the reservoir resulting in the formation of a medium of proppant in the fractures. The natural stresses in the rocks cause the closure of fractures, thereby, trapping the proppant which would then act as a conductive medium for the extraction of the gas present in the reservoir. Two control objectives usually associated with hydraulic fracturing during proppant injection is to obtain uniform prop-

Algorithm 2 Proposed methodology for enlarging the DOA of LDMDc

1: Say the d^{th} data set contains the state matrix $\mathbf{X}^d = \{\mathbf{x}_1\ \mathbf{x}_2\ \dots\ \mathbf{x}_n\}_d$ and the corresponding inputs applied on them are $\mathbf{U}^d = \{\mathbf{u}_1\ \mathbf{u}_2\ \dots\ \mathbf{u}_n\}_d$, respectively.

2: For every data set, stack the state and input matrices to construct the 'augmented' matrix $\mathbf{\Omega}^d$

$$\mathbf{\Omega}^d = \begin{bmatrix} \mathbf{X}^d \\ \mathbf{U}^d \end{bmatrix}$$

3: Form the 'combined' matrix $\mathbf{\Omega}$ by stacking each 'augmented' matrix horizontally

$$\mathbf{\Omega} = \{\mathbf{\Omega}^1\ \mathbf{\Omega}^2\ \dots\ \mathbf{\Omega}^d\}$$

4: Normalize each row of the matrix $\mathbf{\Omega}$ to form the 'normalized' matrix $\mathbf{\Omega}_N$

$$\mathbf{\Omega}_N = \{\mathbf{\Omega}_N^1\ \mathbf{\Omega}_N^2\ \dots\ \mathbf{\Omega}_N^d\} = \begin{bmatrix} \mathbf{X}_N^1 & \mathbf{X}_N^2 & \dots & \mathbf{X}_N^d \\ \mathbf{U}_N^1 & \mathbf{U}_N^2 & \dots & \mathbf{U}_N^d \end{bmatrix}$$

5: Apply weights $\mathbf{w_x}$ and $\mathbf{w_u}$ on the 'normalized' matrix $\mathbf{\Omega}_N$ such that $\mathbf{w_x}$ and $\mathbf{w_u}$ will be equally divided among the components of the state vector and input vector, respectively. The 'weighted' matrix is $\mathbf{\Omega}_w$

$$\mathbf{\Omega}_w = [\mathbf{w}_x\ \mathbf{w}_u] \begin{bmatrix} \mathbf{X}_N^1 & \mathbf{X}_N^2 & \dots & \mathbf{X}_N^d \\ \mathbf{U}_N^1 & \mathbf{U}_N^2 & \dots & \mathbf{U}_N^d \end{bmatrix} = \begin{bmatrix} \mathbf{\Omega}_w^1\ \mathbf{\Omega}_w^2\ \dots\ \mathbf{\Omega}_w^d \end{bmatrix}$$

6: Perform PCA on the 'weighted' matrix $\mathbf{\Omega}_w$ and select the first 'p' Principal Components such that the cumulative sum of their Principal Component Variances is at least 90%.

7: Transform the data in the matrix $\mathbf{\Omega}_w$ to their corresponding PCSs. As we are only considering the first 'p' Principal Components, the dimension of the matrix $\mathbf{\Omega}_w$ reduces to 'p'.

$$\mathbf{PCA}(\mathbf{\Omega}_w) = \begin{bmatrix} \mathbf{PCS_1}(\mathbf{\Omega}_w^1)\ \mathbf{PCS_1}(\mathbf{\Omega}_w^2)\ \dots\ \mathbf{PCS_1}(\mathbf{\Omega}_w^d) \\ \mathbf{PCS_2}(\mathbf{\Omega}_w^1)\ \mathbf{PCS_2}(\mathbf{\Omega}_w^2)\ \dots\ \mathbf{PCS_2}(\mathbf{\Omega}_w^d) \\ \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ \mathbf{PCS_p}(\mathbf{\Omega}_w^1)\ \mathbf{PCS_p}(\mathbf{\Omega}_w^2)\ \dots\ \mathbf{PCS_p}(\mathbf{\Omega}_w^d) \end{bmatrix}$$

8: Cluster the matrix $\mathbf{PCA}(\mathbf{\Omega}_w)$ using any clustering technique such that the clustering output satisfies the criteria mentioned in the clustering subsection. Obtain the cluster centers in terms of the average PCSs of the snapshots within the respective clusters.

9: For every cluster j, obtain the corresponding linear operator pair $(\mathbf{A}_j, \mathbf{B}_j)$ using the original 'training' data of snapshots within that cluster.

10: To select the correct local ROM, implement the kNN technique with k value as 1 as only one local ROM is required to calculate the next state of the system. A subset of cluster centers is selected as explained previously. Also the query point is transformed as described in Steps 2 to 5, its PCSs will be obtained using the same 'p' PCA components obtained in Steps 6 to 7, and then will be used in the kNN technique.

pant concentration throughout the length of the fracture, and to obtain a desired fracture geometry. In this section we applied our proposed methodology to build a LDMDc-based ROM which can be used to predict the proppant concentration at various locations of the fracture at various times

of the proppant injection process for a wide range of proppant pumping schedules.

2.1.3.1 Dynamic modeling of hydraulic fracturing process

Hydraulic Fracturing can be classified into 3 subprocesses which are as follows: (1) Fracture propagation, (2) Proppant transport, and (3) Proppant bank formation.

Fracture propagation: The fracture propagation is assumed to follow the Perkins, Kern, and Nordgren (PKN) model [1, 98] which is shown in Figure 2.1. The other assumptions considered with regard to the fracture propagation are as follows: (1) the fracture length is much greater than its width, and hence, the fluid pressure along the vertical direction remains constant; (2) large stresses in the rock layers above and below the fracture resulting in the fracture being confined to a single layer; and (3) the rock properties such as Young's modulus and Poisson's ratio remain constant with respect to both time and space, and the fracturing fluid is incompressible. Considering the above assumptions, it must be noted that the fracture will take an elliptical shape and its cross-sectional area will be rectangular.

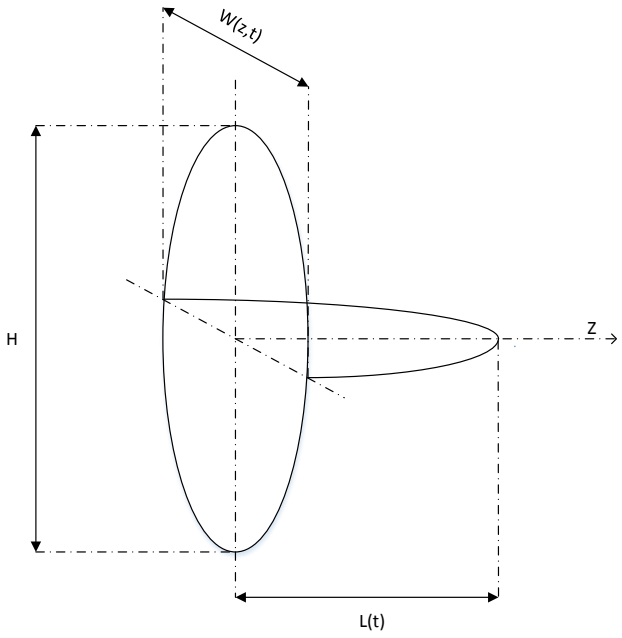


Figure 2.1: The PKN fracture model [1].

Fluid momentum is explained using the lubrication theory which relates the fluid flow-rate in the horizontal direction, q_z, to the sustained pressure gradient, $-\frac{\partial P}{\partial z}\hat{z}$, as follows:

$$q_z = -\frac{\pi H W^3}{64\mu}\frac{\partial P}{\partial z} \tag{2.4}$$

where P is the net pressure varying with the z coordinate, H is the fracture height, W is the width of the fracture, and μ is the fracturing fluid viscosity. The maximum width of the fracture can be related to the net pressure exerted by the fracturing fluids as follows:

$$W = \frac{2PH(1-\nu^2)}{E} \tag{2.5}$$

where E is the Young's modulus and ν is the Poisson's ratio of the formation. The continuity equation obtained by local mass conservation of an incompressible fluid is given by:

$$\frac{\partial A}{\partial t} + \frac{\partial q_z}{\partial z} + HU = 0 \tag{2.6}$$

where $A = \pi WH/4$ is the cross-sectional area of the fracture, t is the time elapsed since the beginning of the fracturing process, z is the time-dependent spatial coordinate in the horizontal direction, and U is the fluid leak off rate per unit height into the reservoir. The fluid leak off rate is in the orthogonal direction to the fracture plane and is given by [99, 100]:

$$U = \frac{2C_{\text{leak}}}{\sqrt{t-\tau(z)}} \tag{2.7}$$

where C_{leak} is the overall leak off coefficient, and $\tau(z)$ is the time instance at which the fracturing fluid reached the coordinate z for the first time. Plugging Eqs. (2.4)-(2.5) into Eq. (2.6) results in the following partial differential equation:

$$\frac{\pi H}{4}\frac{\partial W}{\partial t} - \frac{\pi E}{128\mu(1-\nu^2)}\left[3W^2\Big(\frac{\partial W}{\partial z}\Big)^2 + W^3\frac{\partial^2 W}{\partial z^2}\right] + HU = 0 \tag{2.8}$$

The two boundary conditions and an initial condition for the process are formulated as follows [101, 102]:

$$q_z(0,t) = Q_0 \qquad W(L(t),t) = 0, \tag{2.9}$$

$$W(z,0) = 0 \tag{2.10}$$

where Q_0 is the fluid injection rate at the wellbore, and $L(t)$ is the fracture tip varying with time.

Proppant transport: In this model, the proppant is assumed to travel with the superficial velocity of the fracturing fluid in the horizontal direction, and it is assumed to travel with the settling velocity relative to the fracturing fluid in the vertical direction due to the effect of gravity. The other assumptions adopted are as follows: (1) the proppant particle size is assumed to be large enough to neglect the diffusive flux while only convective flux is taken into consideration; (2) the interactions between the proppant particles are neglected while only drag and gravity effects are considered; and (3) the proppant particles have a uniform size. Based on these assumptions, the advection of proppant in the z direction can be computed as:

$$\frac{\partial(WC)}{\partial t} + \frac{\partial}{\partial z}(WCV_p) = 0 \tag{2.11}$$

$$C(0,t) = C_0(t) \quad and \quad C(z,0) = 0 \tag{2.12}$$

where $C(z,t)$ is the suspended proppant concentration at the coordinate z, and at time t. $C_0(t)$ is the proppant concentration injected at the wellbore. V_p is the net velocity of the proppant particles and is obtained by [103]:

$$V_p = V - (1-C)V_s \tag{2.13}$$

where V is the superficial fluid velocity in the horizontal direction, and V_s is the gravitational settling velocity which can be computed as [104]:

$$V_s = \frac{(1-C)^2(\rho_{sd} - \rho_f)gd^2}{10^{1.82C}18\mu} \tag{2.14}$$

where ρ_{sd} is the proppant particle density, ρ_f is the pure fluid density, d is the proppant particle diameter, g is the gravitational constant, and μ is the fracture fluid viscosity which can be related to the proppant concentration as follows [105]:

$$\mu(C) = \mu_0 \left(1 - \frac{C}{C_{max}}\right)^{-\alpha} \tag{2.15}$$

where μ_0 is the pure fluid viscosity, C_{max} is the maximum theoretical concentration determined by $C_{max} = (1-\phi)\rho_{sd}$ where ϕ is the proppant bank porosity, and α is an exponent in the range of 1.2 to 1.8.

Proppant bank formation: The proppant settling results in the formation of a proppant bank and the variation of the bank height, δ, can be explained using the following equations [101, 106]:

$$\frac{d(\delta W)}{dt} = \frac{C V_s W}{(1-\phi)} \tag{2.16}$$

$$\delta(z, 0) = 0 \tag{2.17}$$

where Eq. (2.17) is the initial condition for Eq. (2.16). More information about the first-principles model of hydraulic fracturing process can be obtained from [4, 3, 5].

We solve the dynamic model of the hydraulic fracturing process for various input profiles in the selected finite region of the input space. A numerical scheme is adopted considering the highly nonlinear nature of the model, and the moving boundary of the system [4, 107]. We used a fixed mesh strategy to solve the high-fidelity model. A finite region of the input space was selected in which a total of 13 distinct 'training' input profiles were chosen as shown in Figure 2.2. Note that the proppant injection was started at $t = 220$ s. The reason for this design of the 'training' input profiles is to closely imitate the practically viable inlet proppant concentration in the field which is usually an increasing staircase profile. The step increases have been kept constant at 0.5 in all the cases. Another reason for this kind of pattern is to make sure that we cover the entire finite region so as to obtain rich 'training' data sets and the amount of data used in model building would be

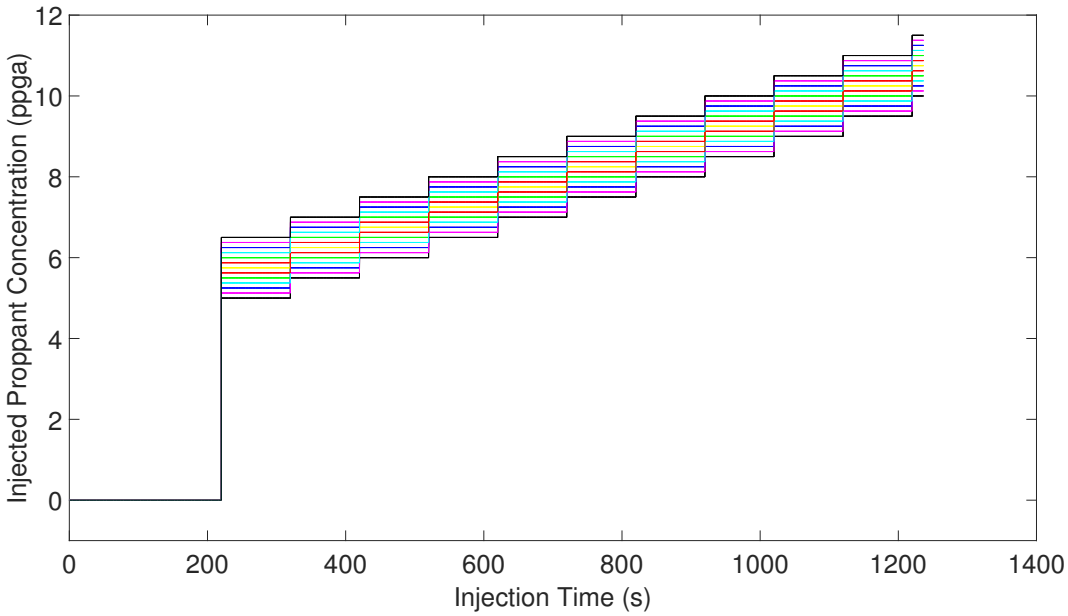


Figure 2.2: Different 'training' input profiles used to generate open-loop simulation data for model training.

optimal. Random input profiles can be considered within this region but the number of 'training' inputs required to cover the entire region would be larger. Also, to avoid using many 'training' inputs and to cover the entire region would require the step increase to be greater than 0.5, which is usually not practical to be implemented in the field.

Each 'training' input profile is implemented on the open-loop system, and the corresponding response of the system is obtained by solving the high fidelity model. The simulations are carried out for $t_f = 1236.4$ s which resulted in a total of 12365 snapshots in each 'training' data set. The high-order discretization scheme resulted in a total of 501 spatial points across the length of the fracture out of which only 101 were selected at equidistant points. The same discretization scheme was applied to the simulations of all the 'training' input profiles. The 'training' data obtained in these simulations were used in building ROMs through LDMDc.

2.1.3.2 Building LDMDc-based ROMs

Our algorithm can be divided into 3 sections: (1) Temporal clustering, (2) Building ROMs for each cluster, and (3) Model selection for validation/prediction. Note that only the data after the

proppant injection began was used in this work.

We first built the 'augmented' vectors by stacking the state vector with its corresponding input vertically, formed the 'combined' matrix by stacking horizontally all the 'augmented' vectors from all the 'training' data sets, normalized each row of the 'combined' matrix, and applied weights on the resultant 'normalized' matrix. In this work we applied equal weights [0.5, 0.5] on the state vector and the input. And, the weight 0.5 on the state vector was equally divided among the components of the state vector whereas the weight on the input was kept the same. The reason we chose this weights is that both the current state of the system and the input applied on it are equally important in propelling the system forward. In other applications it is possible that this may not be the case, and therefore, we suggest that a trial and error scheme needs to be adopted to obtain these weights. We performed PCA on weighted matrix and found that the 1^{st} principal component (PC) was able to capture 99.59 $\%$ of the total variance. Therefore, it was sufficient for us to just use the 1^{st} PCS of all the data points in the clustering step as well as in the model selection step of the algorithm. The clustering strategy was implemented that satisfies the criteria mentioned previously; that is, no two data points from two different 'training' data sets will lie in the same cluster, and no two data points having the same input will lie in the same cluster. We obtained a total of 143 clusters and the output of this clustering strategy in the input space is shown in Figure 2.3 wherein each color represents a cluster. The cluster centers were computed in terms of the 1^{st} PCS and stored to be used in the local ROM selection step of the algorithm.

We applied the DMDc method to every cluster wherein we set the tolerance limit on the singular values as 1 to determine the corresponding p and r values for the purpose of model order reduction. For each cluster, we obtained a pair of linear operators, $(\mathbf{A}_j, \mathbf{B}_j)$, that captures the underlying local dynamics exhibited by the snapshots of that particular cluster. This pair of linear operators is then used to build the linear state-space model which will be then used for model validation and for prediction in the case of random inputs selected within the finite region.

During model validation or model prediction, at any time step, a local ROM needs to be selected based on the current state of the system and the input to be applied on it to further propagate the

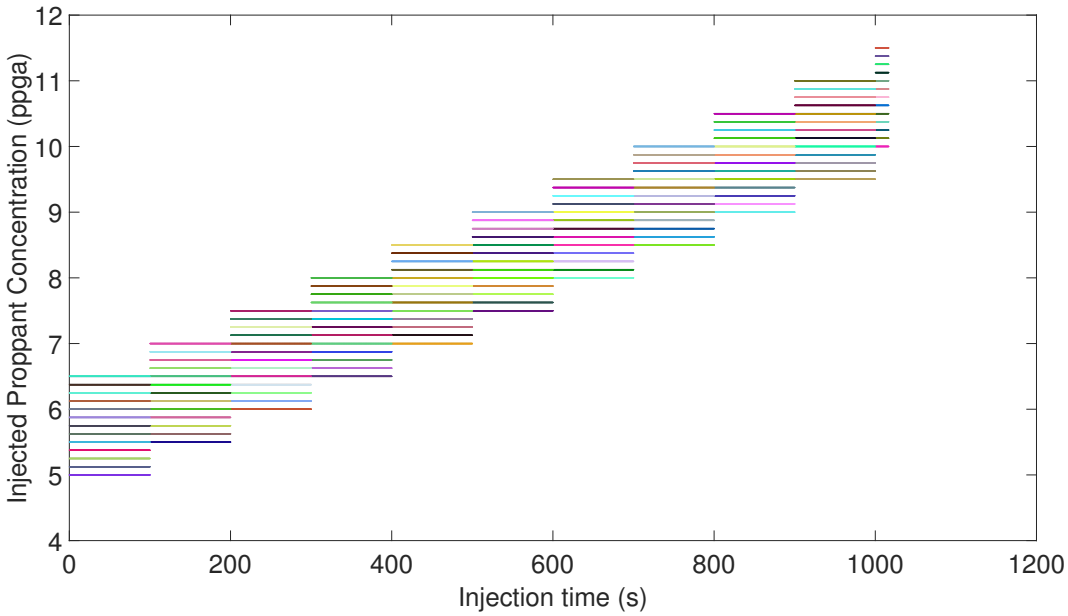


Figure 2.3: Clustering output representation in the input space.

system. To achieve this objective, we implemented the kNN technique by setting the k value as 1 as we only need to select one local ROM. To obtain the subset of cluster centers, we implemented the constraint that at any time step only those cluster centers will be used in the selection process whose corresponding clusters contain the snapshot which was obtained at the same time instance during the open-loop data generation process. Also, at every time step, the query points (i.e., the current state and the input) were transformed in the exact manner adopted in the clustering step. Recall, the parameter used in the selection process is the PCSs with respect to the dominant components. In this work, the 1^{st} PCSs of the subset of cluster centers, and of the query point were used as the 1^{st} PC was able to capture 99.59 $\%$ of the total variance in the data.

2.1.4 Model validation

To verify the accuracy of our proposed methodology, we implemented one of the 'training' inputs which is shown in Figure 2.4. We utilized the developed LDMDc-based ROMs to compute the output for the selected 'training' input and it is compared against the output of the full-order model. Figure 2.5 shows the comparison of these two models with respect to the evolution of the proppant concentration at four different locations during the injection process. It can be seen that

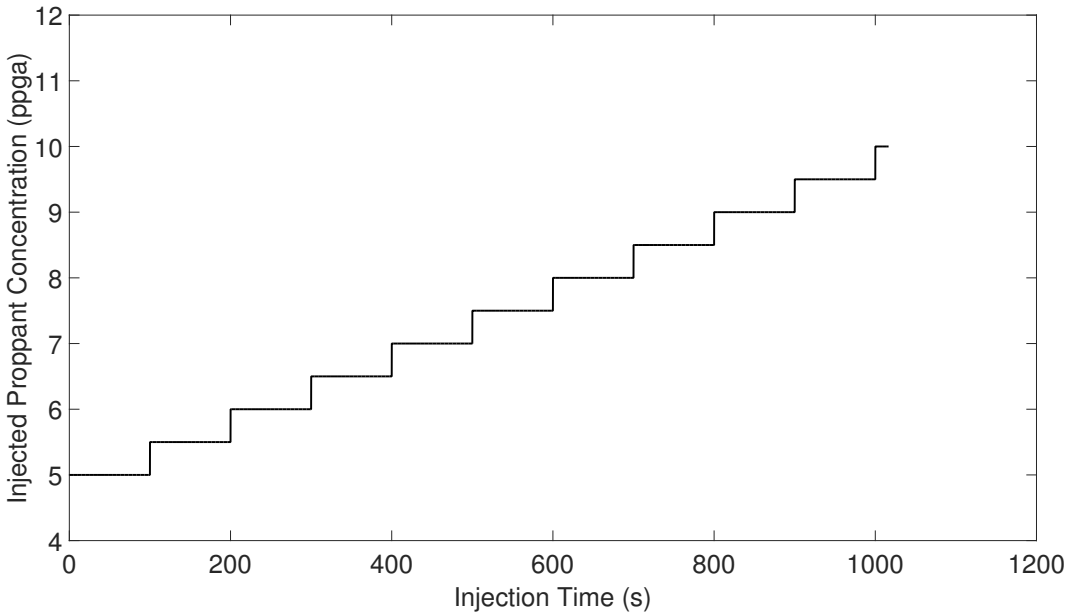
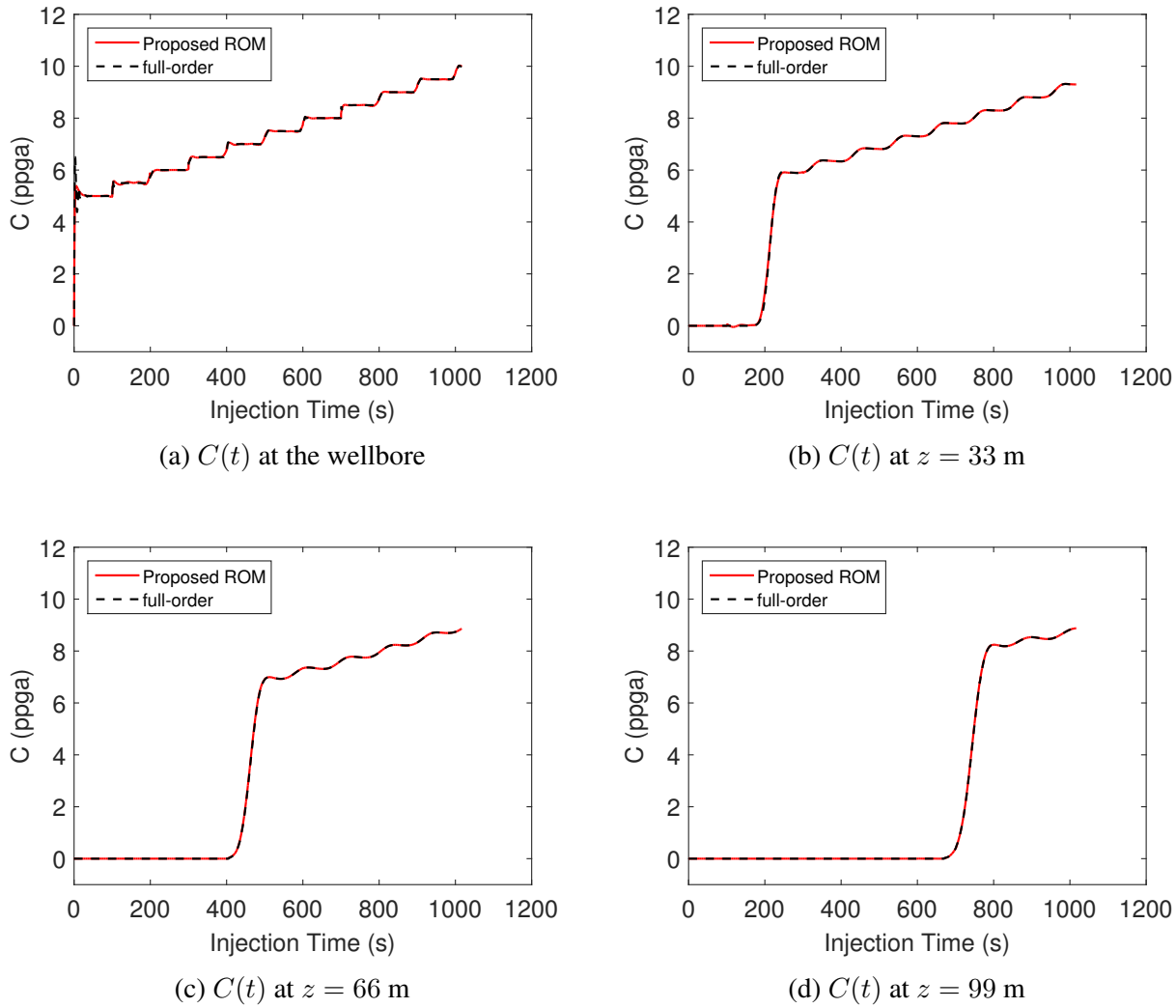


Figure 2.4: Validation input.

the output obtained using the proposed algorithm mimics the output from the full-order model. We used a relative error metric, $E(t)$, to quantify the performance of our proposed methodology in comparison to the full-order solution. The relative error is calculated by using the Frobenius norms of the state vectors as follows:

$$E(t) = \frac{\|\mathbf{x}_{full} - \mathbf{x}_{rom}\|_{fro}}{\|\mathbf{x}_{full}\|_{fro}} \tag{2.18}$$

where $\|.\|_{fro}$ is the Frobenius norm, $\mathbf{x}_{full}$ is the state vector obtained from the full-order solution, and $\mathbf{x}_{rom}$ is the state vector obtained from a ROM developed by the proposed methodology. The relative error for the approximate solution obtained from our proposed methodology is presented in Figure 2.6. From the plot we observe that the proposed methodology is able to provide an accurate approximation when compared to the full-order solution. Similarly, our proposed methodology will give accurate solutions when any of the other 12 'training' inputs are used for validation purposes. Thus, these results validate the proposed methodology and warrant its use for the purposes of prediction when random inputs are used within the selected finite input space.



(a) $C(t)$ at the wellbore

(b) $C(t)$ at $z = 33$ m

(c) $C(t)$ at $z = 66$ m

(d) $C(t)$ at $z = 99$ m

Figure 2.5: Comparison of the approximate solution computed using LDMDc-based ROMs with the full-order solution.

2.1.5 Model prediction

Equipped with the set of LDMDc-based ROMs, we used our model selection step to predict the output when a random input is considered within the selected finite input region.

2.1.5.1 Random input

In this case a random input was generated with the constraint that the injected proppant concentration was varied at every 100s as in the 'training' inputs. The generated random input is shown in Figure 2.7. Note that in all the 'training' inputs, as described in Figure 2.3, the step increase of

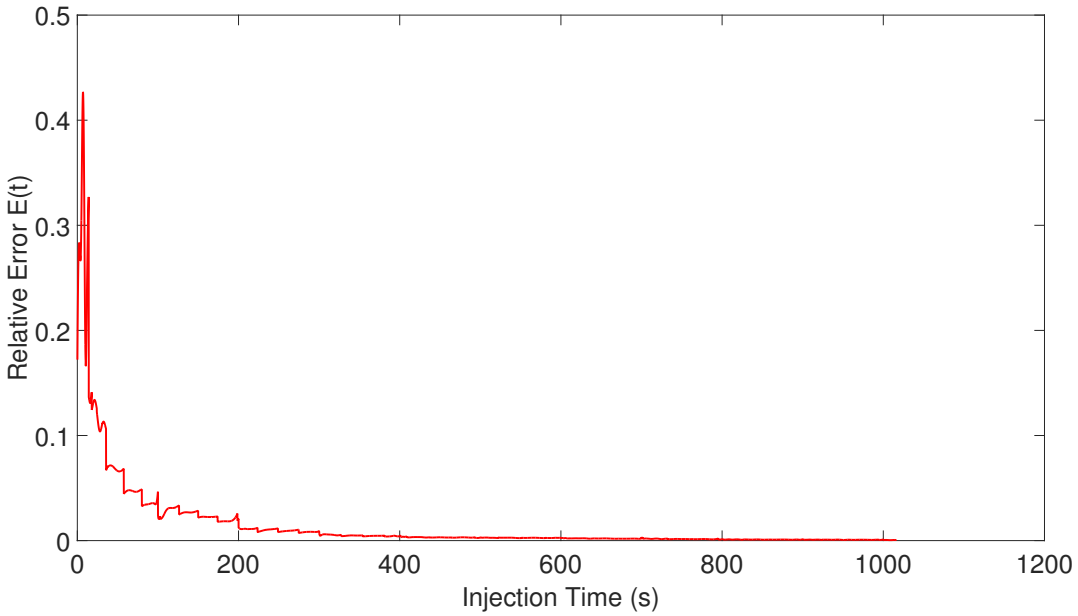


Figure 2.6: Profile for $E(t)$ with time for solution obtained from our proposed methodology when the validation input is used.

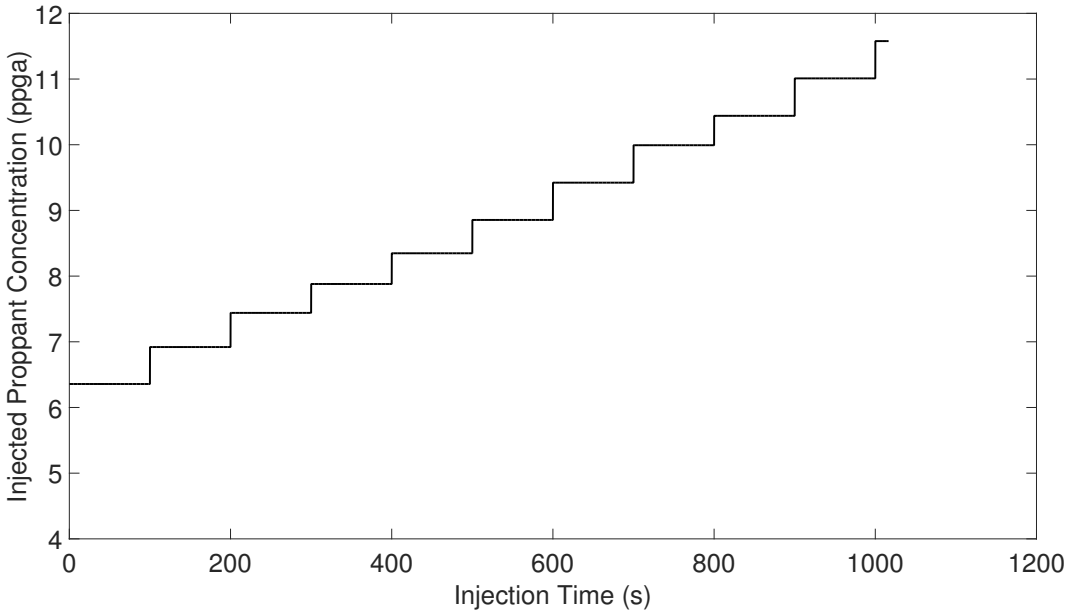
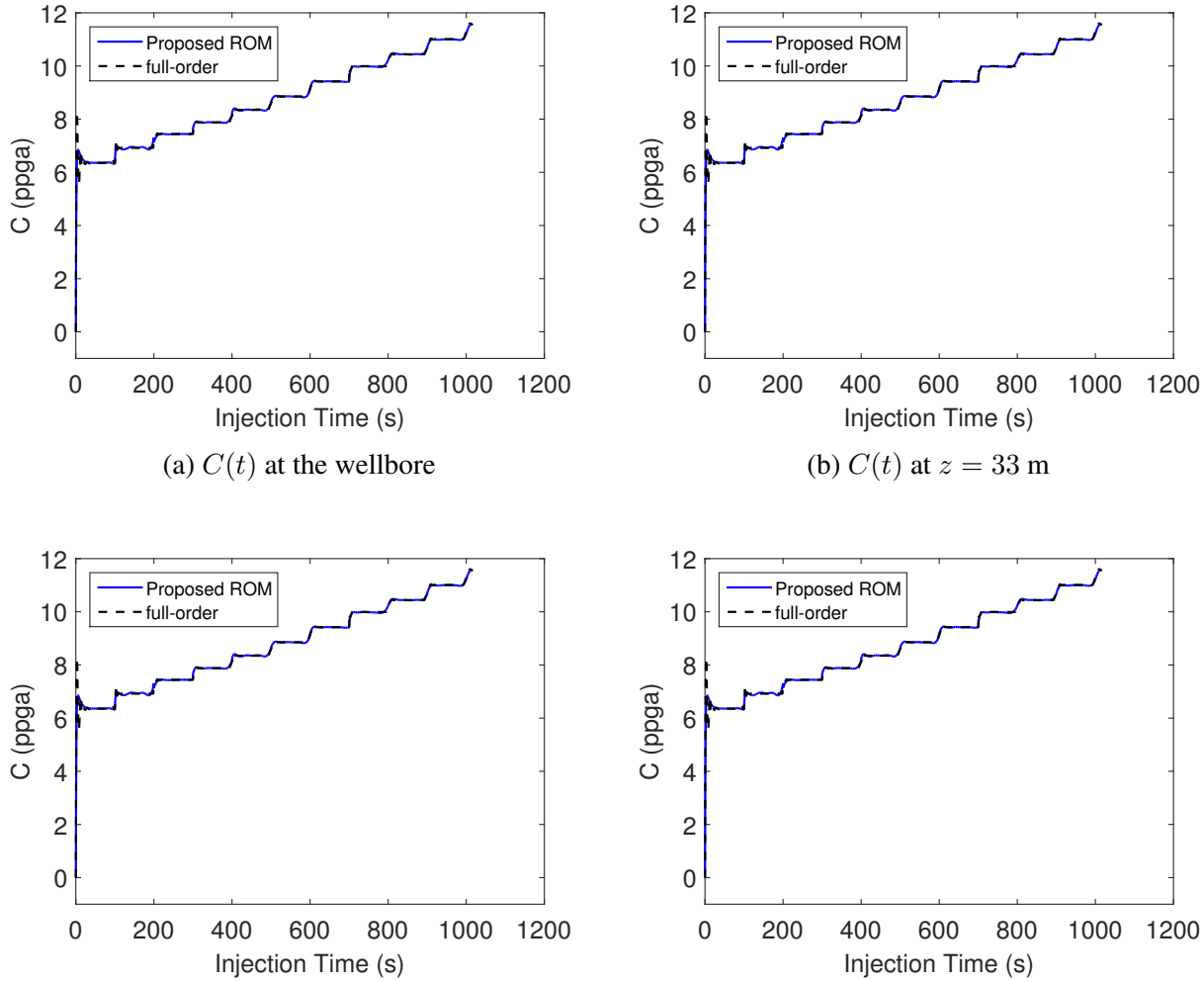


Figure 2.7: Random input profile used for model prediction.

0.5 in the injected proppant concentration was kept constant throughout the pumping process. But in the case of this model prediction a different constraint was implemented that when generating

the random input the step increase in the injected proppant concentration will lie in the range of $[0.425\ 0.575]$. The reason we implemented this constraint is that it makes the generated random input closely mimic the 'training' inputs and also allows flexibility to deviate from them to a limited extent. The output predicted by the proposed algorithm is compared with the output from the high fidelity model at 4 different locations along the length of the fracture and are presented in Figure 2.8. From the figure we observe that the proposed methodology is able to accurately predict



(a) $C(t)$ at the wellbore

(b) $C(t)$ at $z = 33$ m

(c) $C(t)$ at $z = 66$ m

(d) $C(t)$ at $z = 99$ m

Figure 2.8: Comparison of the prediction output computed using LDMDc-based ROMs with the full-order solution.

the concentration at 4 different locations when compared to the full-order solution. To quantify the accuracy of the prediction, we calculated the relative error as defined in Eq. (3.21) and is plotted at

various times during the injection process as shown in Figure 2.9.

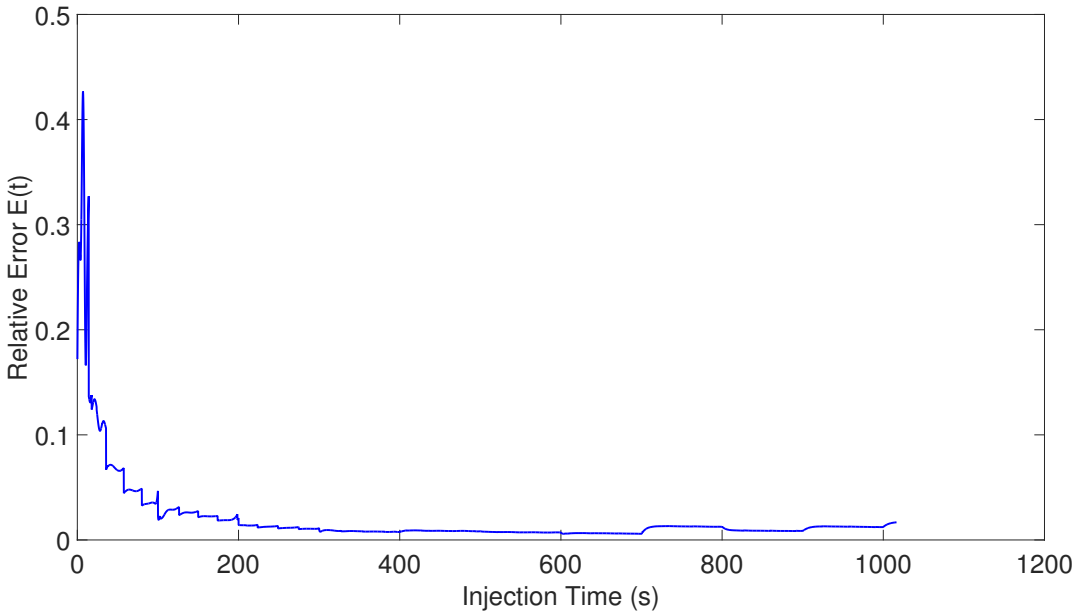


Figure 2.9: Profile for $E(t)$ with time for the prediction when the random input is used.

2.1.6 Comparison with LDMDc

In this section, we illustrate the superior performance of our proposed methodology in comparison to the LDMDc technique proposed by [9] for a randomly generated, constrained input within the finite region of input space. To do so, we first generated the required data by carrying out open-loop simulations of the high-fidelity model. The 'training' input used mimics the ones used in our proposed methodology; in particular, the step increase in the concentration of the injected proppant is kept constant at 0.5 all throughout the injection process as shown in Figure 2.10. We then used just the information of the states to cluster the data into 11 clusters, where for each cluster we captured the local dynamics using LDMDc-based ROMs. These ROMs are then used to calculate the approximate solution of the full-order model. Now that the LDMDc technique has been used to build the model, we used the prediction input shown in Figure 2.7 to compare its performance with the performance of our proposed methodology by plotting the relative error profiles for both the techniques as shown in Figure 2.11. Note that the relative errors for each technique was obtained

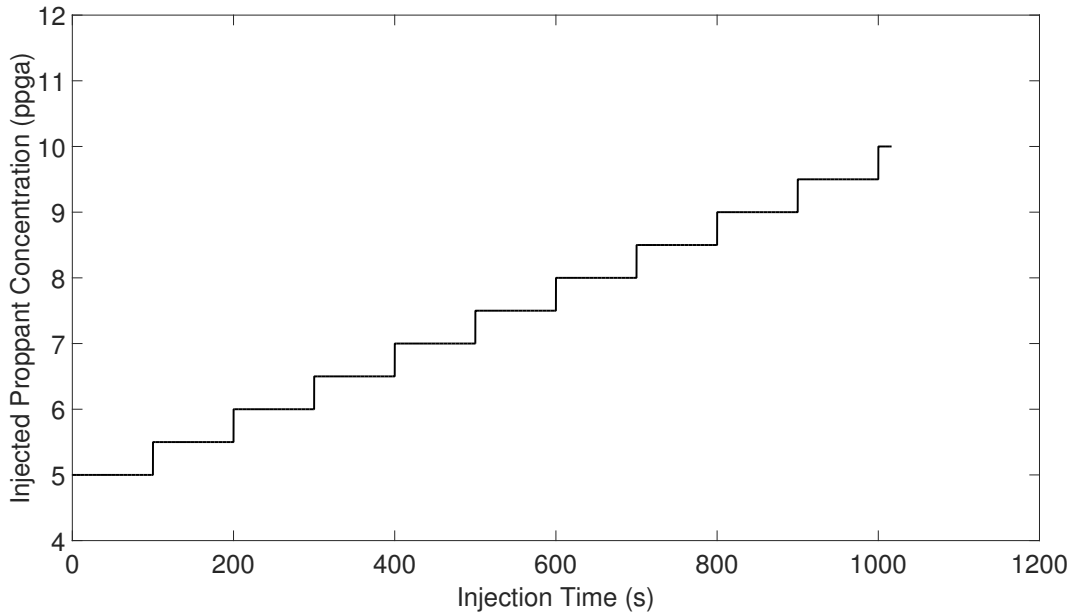


Figure 2.10: Input used to build a LDMDc-based ROM.

by comparing their corresponding approximate solutions to the solution of the high-fidelity model. From the plot we observe the limitation of the LDMDc technique, which is its poor performance

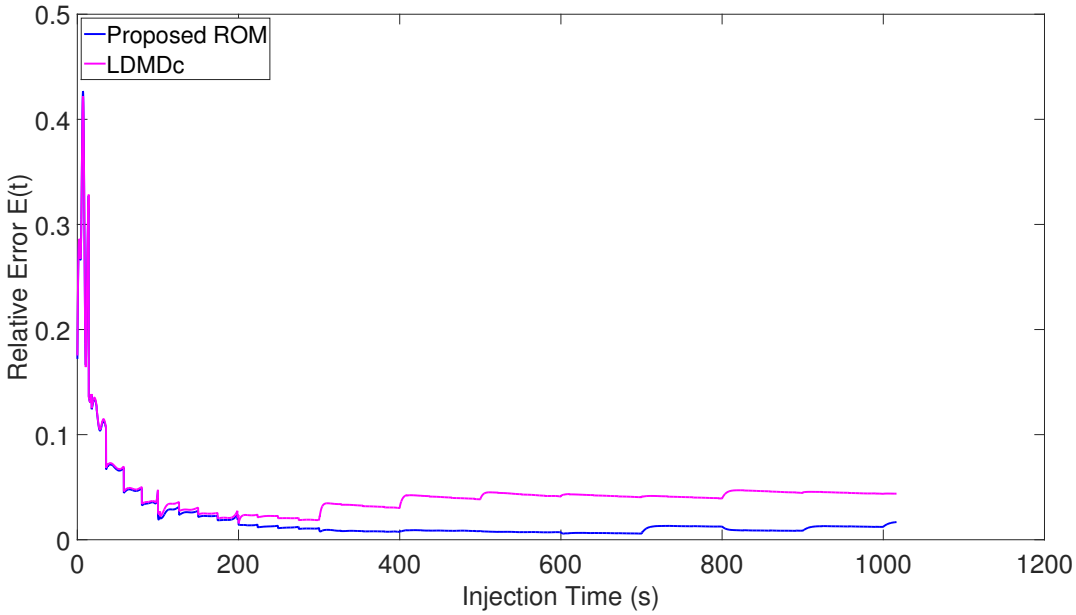


Figure 2.11: Comparison of the relative error profiles of our proposed methodology and the LD-MDc technique.

when a random input is used, which can be overcome by using our proposed methodology.

3.1 Deep hybrid modeling of chemical processes: Application to hydraulic fracturing*

Data-based models are easy to construct but lack in robustness and extrapolation properties unlike first-principles model. In order to balance the advantages and disadvantages of data-based models and first-principles models, hybrid models have been developed which integrate first-principle models with data-based models. Such models have superior accuracy compared to the first-principles model, and better extrapolation properties compared to the data-based models. In process modeling, the concept of hybrid (grey box) models evolved from the field of neural networks [21, 22, 23, 24]. The idea was to build neural network based hybrid models through the use of first principles knowledge. This resulted in hybrid models with better prediction accuracy compared to the first principles models, and better interpolation, extrapolation, and interpretation compared to solely neural networks based models. There exist other kinds of grey box models that combine different types of first principles knowledge and/or empirical submodels. During the 90s, the term 'grey box' models appeared in systems and control theory wherein structural information from the first principles models was incorporated into the data-based models [25, 26, 27]. But the understanding of the term 'grey box' has evolved to represent all types of hybrid models that combines first principles and data-based submodels. Nonetheless, hybrid modeling balances the advantages and disadvantages of strictly first principles and data-based modeling, and offers desirable benefits such as high prediction accuracy, better extrapolation capabilities, ease of calibration, and better interpretability.

Neural networks are connectionist models that map an input space to an output space and were inspired by the biological networks present in the brain. Each neural network contains elements

*Reprinted with permission from "Deep hybrid modeling of chemical process: Application to hydraulic fracturing" by Bangi, M. S. F., and Kwon, J. S. 2020. Comput. Chem. Eng., 134, 106696, Copyright 2020 Elsevier.

called ‘neurons’ or ‘nodes’ that process an input and give an output, and comprises of layers containing many such nodes. Each node in a particular layer is connected to every node in an adjoining layer and the strength of each connection is represented by an assigned weight. Consequently, a node in a particular layer receives inputs from all the nodes in the previous layer which are added together after weights are applied on each of them, and a ‘bias’ is added to the sum. If the neural network only contains weights and biases, then it behaves as a linear function. In order to capture nonlinearity in the input-output data, certain nonlinear functions called as ‘Activation functions’ are used in each node. With activation functions, the input to the neural network undergoes nonlinear transformation through each layer and is collected as the output at the final layer. Overall, the number of nodes, number of layers, weights in each connection, and the biases are the internal parameters of the neural network. Under certain assumptions, neural networks, if sufficiently large, have been shown to capture any nonlinear continuous function accurately [61].

With the advancements in Machine Learning, the field of neural networks has evolved from the use of a single hidden layer to multiple hidden layers resulting in deep neural networks. Recently, there has been lots of interests in understanding and comparing the capabilities of shallow neural networks and deep neural networks as function approximators. For instance, it has been proven that deep sum-product networks for polynomial functions [108], deep neural networks for piecewise smooth functions [109], and three layer network for a specific function [110] require an exponentially less number of neurons than their shallow counterparts. Also, it has been shown that the number of linear regions of the functions approximated by deep neural networks is exponentially large compared to shallow neural networks [111]. These works demonstrate that shallow networks require an exponentially large number of neurons compared to deep networks for specific functions, and hence, depict the power of deep networks over shallow networks.

In this work, we develop a hybrid model for a hydraulic fracturing process that combines its first principles model with a deep neural network that acts as an estimator of its unmeasured process parameters. The hydraulic fracturing process is a complex system in which the fracturing fluid containing proppant is pumped into the reservoir to create and sustain the created fractures in order

to extract oil/gas through them from the reservoir. There are two main issues when modeling this process which are its moving boundary nature and numerous process uncertainties. One such uncertainty is the leak off rate of the fracturing fluid into the reservoir which is difficult to accurately explain using first principles, and building a first principles based model with such a knowledge gap will result in its inaccuracy. Therefore, the novelty in this work is two fold. First, the use of deep neural networks for the purpose of building hybrid models. Second, the application of this hybrid modeling methodology to a complex hydraulic fracturing process in order to explain one of its underlying fluid leak-off phenomena and build a superior model.

3.1.1 Deep neural networks

Deep neural networks are the neural networks with more than three layers and each layer contains multiple neurons. The neurons in each layer are fully connected to the neurons in the subsequent layer as shown in Figure 3.1. The strength of each connection is assigned by its weight w, and each neuron processes the input through a nonlinear function called activation function f.

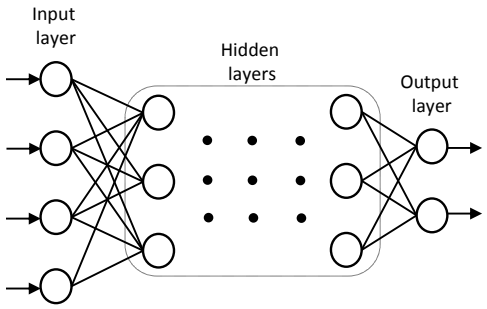


Figure 3.1: Deep neural networks.

Let $n^{k+1}(i)$ be the net input to each unit i in layer $k+1$ which is given by

$$n^{k+1}(i) = \sum_{j=1}^{Sk} w^{k+1}(i,j)a^k(j) + b^{k+1}(i) \tag{3.1}$$

Let $a^{k+1}(i)$ be the output of unit i in layer $k+1$ which is given by

$$a^{k+1}(i) = f^{k+1}(n^{k+1}(i)) \tag{3.2}$$

Assuming there are M layers in the network, the equations in matrix form can be represented as

$$A^{k+1} = F^{k+1}(W^{k+1}A^k + B^{k+1}), \qquad k = 0, 1, ..., M-1 \tag{3.3}$$

$$A^0 = u_q, \qquad q = 1, 2, ..., Q \tag{3.4}$$

where u_q is the input vector to the neural network and its corresponding output is A_q^M. The matrices A^k, F^k, W^k, and B^k contain the outputs, the activation functions, weights, and biases of all the neurons in layer k, respectively. The aim of the neural network is to learn the relationship between the input-output pairs $\{(u_1, y_1), (u_2, y_2), ...(u_Q, y_Q)\}$. The performance of the neural network is measured as follows:

$$V = \frac{1}{2}\sum_{q=1}^{Q} e_q^T e_q \tag{3.5}$$

$$e_q = (y_q - A_q^M) \tag{3.6}$$

where e_q is the error when the q^{th} input is given to the neural network. The error matrix E can be defined as follows:

$$E = [e_1\ e_2 \dots e_Q]^T \tag{3.7}$$

3.1.2 Levenberg-Marquardt training

The Error Backpropagation (EBP) algorithm [112, 113] is one of the most significant breakthroughs in regard to the training of neural networks, but it has an issue of slow convergence, which can be explained using the following two reasons. First, the step size should be small to prevent oscillations around the required minima but this would lead to slow training process. Second, the curvature of the error surface could vary in different directions, so the classical 'error

valley' problem [114] could exist and may lead to slow convergence rate. Despite its issue of slow convergence, the steepest descent algorithm is widely used to train neural networks. Its update rule utilizes the gradient g, which is the first-order derivative of the total error function, is defined as follows:

$$g = \frac{\partial V(u,w)}{\partial w} = \begin{bmatrix} \frac{\partial V}{\partial w_1} & \frac{\partial V}{\partial w_2} & \cdots & \frac{\partial V}{\partial w_N} \end{bmatrix}^T \tag{3.8}$$

The update rule of the steepest descent algorithm is written as:

$$w_{k+1} = w_k - \alpha g_k \tag{3.9}$$

where α is the learning constant.

This issue of slow convergence in the steepest descent algorithm can be improved by the Gauss-Newton algorithm [114]. The Gauss-Newton algorithm utilizes the second-order derivatives of the error function to gauge the curvature of error surface and find the appropriate step size for each direction. The convergence is fast if the error function has a quadratic surface, without which this algorithm would be mostly divergent. The update rule of the Gauss-Newton method is defined as:

$$w_{k+1} = w_k - (J_k^T J_k)^{-1} J_k E_k \tag{3.10}$$

where J is the Jacobian matrix which is defined as:

$$J = \begin{bmatrix} \frac{\partial e_1}{\partial w_1} & \frac{\partial e_1}{\partial w_2} & \cdots & \frac{\partial e_1}{\partial w_N} \\ \frac{\partial e_2}{\partial w_1} & \frac{\partial e_2}{\partial w_2} & \cdots & \frac{\partial e_2}{\partial w_N} \\ & \vdots \; \vdots \; \vdots & & \\ \frac{\partial e_Q}{\partial w_1} & \frac{\partial e_Q}{\partial w_2} & \cdots & \frac{\partial e_Q}{\partial w_N} \end{bmatrix} \tag{3.11}$$

The Levenberg-Marquardt algorithm [115, 116] combines the steepest descent method and the Gauss-Newton algorithm. Consequently, it inherits the stability characteristic of the steepest descent algorithm and the speed of the Gauss-Newton algorithm. The Levenberg-Marquardt

algorithm functions as the steepest descent algorithm when the curvature of error surface is complex until the local curvature takes a reasonable approximation of a quadratic surface wherein the Levenberg-Marquardt can behave as the Gauss-Newton algorithm. Essentially, the Levenberg-Marquardt algorithm utilizes the superior aspects of the steepest descent algorithm and the Gauss-Newton algorithm alternatively based on the situational requirements. The update rule of the Levenberg-Marquardt algorithm is given by

$$w_{k+1} = w_k - (J_k^T J_k + \mu I)^{-1} J_k E_k \tag{3.12}$$

where μ is the combination coefficient, and I is the identity matrix. When the μ value is large, then the Levenberg-Marquardt algorithm behaves as the steepest descent method; otherwise, it behaves as the Gauss-Newton method as summarized in Table 3.1.

Algorithm	Parameter updates	Convergence	Computational issue
EBP	$w_{k+1} = w_k - \alpha g_k$	stable, slow	Gradient
Gauss-Newton	$w_{k+1} = w_k - (J_k^T J_k)^{-1} J_k E_k$	unstable, fast	Jacobian
Levenberg-Marquardt	$w_{k+1} = w_k - (J_k^T J_k + \mu I)^{-1} J_k E_k$	stable, fast	Jacobian

Table 3.1: Summary of the training algorithm.

3.1.3 Proposed deep hybrid model

Consider a dynamical system with the following generic representation:

$$\frac{dx}{dt} = f(x, u, p) \tag{3.13}$$

$$p = g(x, u) \tag{3.14}$$

where x, u, and p denote the states, control inputs, and model parameters, respectively. The dynamics of the states of the system is explained by the equations in f which are dependent on parameters p. These parameters p are related to the states x and the inputs u through the equations in g. Consider the dynamical system defined using Eqs. (3.13), (3.14) along with the following

equation:

$$y = h(x) \tag{3.15}$$

where y denotes the output of the system which is related to the states through the equation in h. In this proposed methodology, a DNN will be trained using data to predict the originally unknown parameter values p, and this trained DNN alongside a first principles model, will be used as shown in Figure 3.2 to build a hybrid model.

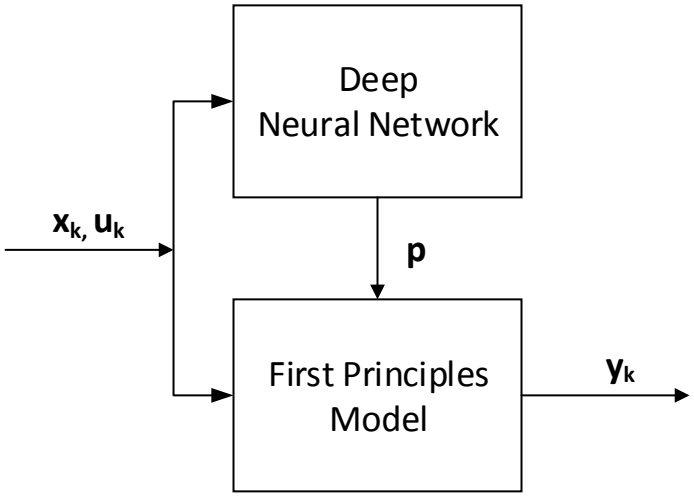


Figure 3.2: Proposed deep hybrid model.

A DNN is initialized by assigning some values for the number of layers, number of neurons in each layer, type of activation function, and the initial values of weights and biases. A DNN has more than one hidden layer apart from the input and output layers. There are many kinds of activation functions being used today but widely used are Sigmoid, Hyperbolic tangent, Rectified linear unit (ReLU), and Leaky rectified linear unit (Leaky ReLU).

3.1.4 Training algorithm

We use an input-output training data set (u_1, y_1), (u_2, y_2), ...(u_q, y_q), ...(u_Q, y_Q) for the purpose of training the hybrid model which includes a DNN to approximate the unknown parameters p. The inputs u are presented to the hybrid model, specifically to the first principles model and

the DNN. The inputs to the DNN are propagated through its layers and the network's outputs are obtained at the final layer. Eqs. (3.1), (3.4) are utilized to calculate the outputs of the DNN which are the predicted parameter values p. These predictions are used as inputs to the first principles model to calculate the outputs of the hybrid model $y^{'}$. For the deep hybrid model, the squared prediction error of the output for all Q training patterns was minimized as:

$$\hat{V} = \frac{1}{2}\sum_{q=1}^{Q}(e_q)^T(e_q) \tag{3.16}$$

$$e_q = y_q - y_q^{'} \tag{3.17}$$

The DNN's output p_q does not explicitly appear in the above error equation as it is generated and utilized internally in the hybrid model. In order to update the parameters of DNN using Eqs. (3.11), (3.12), the effect of the DNN's output p_q on the prediction error of the hybrid model e_q needs to be quantified. For this purpose, we utilize finite difference methods to calculate the gradient of the hybrid model's output $y_q^{'}$ with respect to the DNN's output p_q. Hence, we obtain the following equations:

$$\frac{\partial e_q}{\partial y_q^{'}} = -1 \tag{3.18}$$

$$\frac{\partial y_q^{'}}{\partial p_q} = \frac{y_{q+1}^{'} - 2y_q^{'} + y_{q-1}^{'}}{p_{q+1} - p_{q-1}} \tag{3.19}$$

$$\frac{\partial e_q}{\partial p_q} = \frac{\partial e_q}{\partial y_q^{'}}\frac{\partial y_q^{'}}{\partial p_q} = -\frac{y_{q+1}^{'} - 2y_q^{'} + y_{q-1}^{'}}{p_{q+1} - p_{q-1}} \tag{3.20}$$

Let's define the sensitivity of the error e_q to changes in the net input of unit i in layer k as:

$$\delta_q^k(i) = \frac{\partial e_q}{\partial n_q^k(i)} \tag{3.21}$$

Now, the above equation can be rewritten using Eq. (3.2) as:

$$\delta_q^k(i) = \frac{\partial e_q}{\partial n_q^k(i)} = \frac{\partial e_q}{\partial a_q^k(i)}\frac{\partial a_q^k(i)}{\partial n_q^k(i)} = \frac{\partial e_q}{\partial a_q^k(i)}\,\dot{f}^k(n^k(i)) \tag{3.22}$$

For the last layer M, the above equation leads to:

$$\delta_q^M = \frac{\partial e_q}{\partial A_q^M}\ \dot{F}^M(n_q^M) \tag{3.23}$$

But the output A_q^M from the final layer M is the predicted parameter p_q which results in

$$\frac{\partial e_q}{\partial p_q} = \frac{\partial e_q}{\partial A_q^M} \tag{3.24}$$

Using Eqs. (3.20), (3.23), (3.24), the δ_q^M value can be calculated. Recall, the Jacobian matrix contains the sensitivities of the error e_q to changes in the parameters of the DNN, i.e., W^k and B^k. In mathematical form, these sensitivities can be represented as $\frac{\partial e_q}{\partial W^k}$ and $\frac{\partial e_q}{\partial B^k}$, and can be calculated for the neurons in the final layer M using δ_q^M and Eq. (3.1) in the following way:

$$\frac{\partial e_q}{\partial W^M} = \delta_q^M\ A_q^{M-1} \tag{3.25}$$

$$\frac{\partial e_q}{\partial B^M} = \delta_q^M \tag{3.26}$$

For other layers, $k = 1,\ ...,\ M-1$, δ_q^k value can be calculated using the following recurrence relation:

$$\delta_q^k = \dot{F^k}(n_q^k)\ {W^{k+1}}^T\ \delta_q^{k+1} \tag{3.27}$$

Using δ_q^k value, the Jacobian matrix can be calculated in a manner similar to when obtaining Eqs. (3.25), (3.26). Following the calculation of the Jacobian matrix, the parameters of the DNN can be updated using Eq. (3.12). The Levenberg-Marquardt training algorithm starts with an initial value for μ but it is multiplied by a factor β whenever an update would result in the increase of $\hat{V}$. On the other hand, when an update reduces the $\hat{V}$ value, then μ is divided by β. The training algorithm is repeated until the value of $\hat{V}$ reaches a predefined tolerance. The proposed algorithm has been summarized in Algorithm 3.

Also, the flow diagram of the proposed hybrid modeling framework is presented with details

Algorithm 3 Deep hybrid model training

1: Present all the inputs u_q to the hybrid model and calculate its corresponding output $y_q^{'}$. Compute the errors e_q using Eq. (3.17) and the sum of the squared of errors over all inputs $\hat{V}$ using Eq. (3.16).
2: Compute the Jacobian matrix using Eqs. (3.11), (3.20), (3.21), (3.22), (3.23), (3.24), (3.25), (3.26), (3.27)
3: Update the parameters of the DNN using Eq. (3.12) and recalculate $\hat{V}$ using Eq. (3.16). If it is smaller than that computed in Step 1, then reduce μ by β, and proceed to Step 1 with the new parameters of the DNN. If the $\hat{V}$ is greater than that computed in Step 1, then increase μ by β and repeat this step.
4: The algorithm is terminated when $\hat{V}$ is less than a predetermined value.

in Figure 3.3.

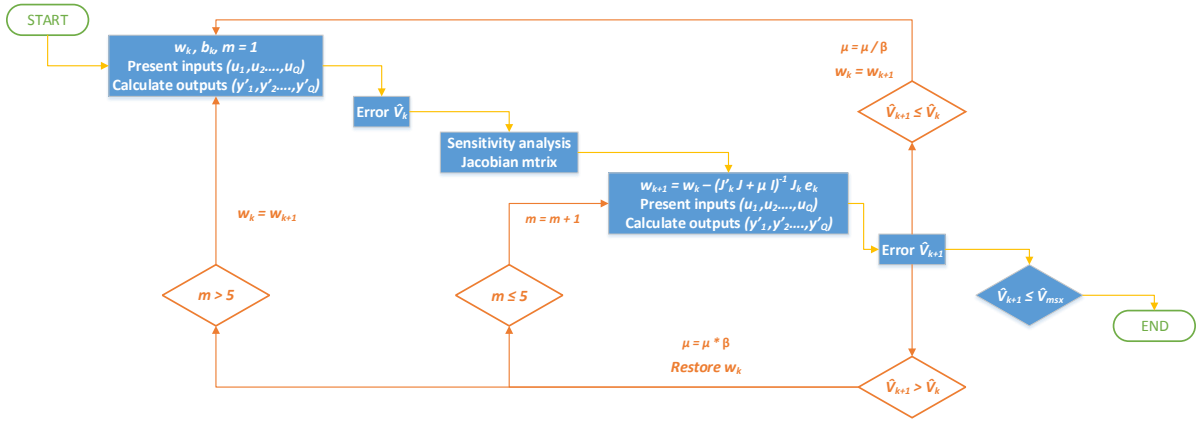


Figure 3.3: Block diagram for Levenberg-Marquardt based deep hybrid model training.

3.1.5 Hydraulic fracturing process

Shale gas is the natural gas trapped in rocks with low porosity and permeability which is difficult to extract using conventional oil/gas extraction techniques. Horizontal drilling and hydraulic fracturing process enabled the extraction of oil/gas from such rocks. In a hydraulic fracturing process, controlled explosions are carried out inside the wellbore to create initial fractures followed by pumping of a clean fluid called pad to extend the geometry of these fractures. Later, a fracturing fluid containing water, proppant and additives is pumped inside the system to further extend these fractures. During and after the pumping process, the fracturing fluid leaks off into the reservoir leaving behind proppant in the fractures. The natural stresses present in the reservoir cause the closure of these fractures with proppant inside them. The presence of proppant creates a conductive medium for the oil/gas to flow through these fractures making their extraction possible. The phe-

nomenon of interest in this work is the leak off of the fracturing fluid into the reservoir during the hydraulic fracturing process as its knowledge is essential to obtain the optimal fracture geometry. Obtaining the fracture geometry requires huge amounts of fracturing fluid, and hence, fluid leak off indirectly affects the economic efficiency of the hydraulic fracturing process. In our past work on modeling and control of hydraulic fracturing process, we utilized a mathematical expression of time-dependent fracturing fluid leak-off rate. In practice, advancement in pressure analysis has made it possible to estimate the leak-off rate from pressure decline following injection of fracturing fluid. But this method requires the knowledge of gross fracture height, and hence, this method is suitable for the formations with a large net permeable height. As previously described, in this work, we utilize a DNN to estimate the fluid leak-off rate from input-output data of the hydraulic fracturing process as explained in the previous section. To achieve this objective, we also utilize a first principles based model of hydraulic fracturing process as described in Eqs. (2.4)-(2.17) [4].

3.1.6 Deep hybrid model for hydraulic fracturing process

Recall, the deep hybrid model structure as shown in Figure 3.1 is utilized to predict the unknown parameters of the first principles model. We utilized the first principles model of the hydraulic fracturing process discussed above along with a DNN to build a hybrid model. The objective of the DNN, once it is trained, is to predict the leak off rate U of the fracturing fluid into the reservoir by using time t as input. The input to the hybrid model is the fracturing fluid injection rate $Q_0(t)$ and the output of the hybrid model is the width of the fracture W_z at various points $z = 1, 2, 3.....251$ along the length of the fracture in the horizontal direction z. The schematic of the deep hybrid model for hydraulic fracturing process is shown in Figure 3.4. Unlike the schematic of hybrid model shown in Figure 3.1, wherein both the state and the input applied on the system are used as inputs to the DNN, we only utilize the time t as input to the DNN as seen in Figure 3.4. For the DNN, we considered a fully connected network consisting of 5 layers, i.e., 3 hidden layers, 1 input layer, and 1 output layer. Each hidden layer contains 20 neurons, and the input and output layer contain 1 neuron each. Hyperbolic tangent was used as the activation

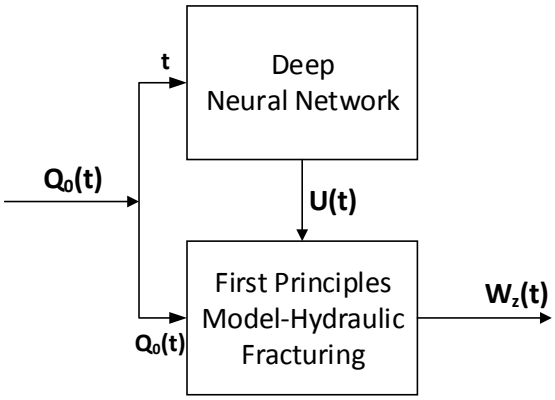


Figure 3.4: Schematic of deep hybrid model for hydraulic fracturing process.

function in the hidden layers and the linear function in the outer layers. To generate training data in order to train the hybrid model, we simulated the first principles model using a constant input flow rate of $Q_0(t) = 0.03\ \mathrm{m}^3/\mathrm{s}$ and the leak off model, as shown in Eq. (3.28) [99, 100], with the assumption that the leak off rate U is independent of spatial location z. The other parameters of the first principles model used in our process calculations are as follows [101]: $H = 20\ m$, $\mu = 0.56$ $\mathrm{Pa} \cdot \mathrm{s}$, $E = 5 \times 10^3$ MPa, and $\nu = 0.2$. The output from the simulation was the widths W_z at various points $z = 1, 2, 3.....251$ along the length of the fracture. These 251 points are spaced equidistantly with a distance of $0.2\ m$ between two consecutive spatial points and we collected 2841 time snapshots of the output data with a time difference of $0.1s$ between two consecutive snapshots. Since we have used the model in Eq. (3.28) for leak off rate when generating the simulation data, we expect the DNN in the hybrid model to capture a similar variation of the leak off rate U.

$$U = \frac{2C_{\mathrm{leak}}}{\sqrt{t - \tau(z)}} \tag{3.28}$$

Now that the hybrid model structure and its components have been defined, and the training data is available, we initialized the parameters of DNN (i.e., weights W and biases B), and began the Levenberg-Marquadt based training algorithm as discussed above in Algorithm 3. The inputs $Q_0(t)$ were presented to the hybrid model with the DNN utilizing t as its input and the first principles model utilizing $Q_0(t)$ as its input. The input t to the DNN undergoes transformation through

its layers and gives the output $U(t)$ which is presented to the first principles model as an input. The first principles model utilizes $Q_0(t)$ along with $U(t)$ to calculate the outputs $W_z^{'}(t)$. Since the parameters of DNN were randomly initialized, the initial predictions of U(t) by the DNN is more likely inaccurate than its actual value which in turn will affect the error $e(t)$ calculated using Eq. (3.17). As the algorithm progresses, the parameters of the DNN are updated, and hence, its predictions of $U(t)$ will move closer to its actual value. A point to note here is that despite the hybrid model predicting widths at 251 locations, we only considered just the width at the wellbore, i.e., $W_1^{'}(t)$ in the objective function calculation Eq. (3.16), and hence in error $e(t)$ calculation Eq. (3.12) as well. This is because we assumed earlier that U does not vary with spatial location, and hence, the $U(t)$ approximated by the DNN is applicable at all locations which implies that when the $W_1^{'}(t)$ values predicted by the hybrid model move towards convergence by updating the parameters of the DNN, the widths at other locations i.e., $W_z^{'}$ where $z = 2, 3.....251$ also move towards convergence.

In order to reduce the error and reach the tolerance of 10^{-6} for $\hat{V}$, we needed to update the parameters of DNN, i.e., its weights W and biases B using Eq. (3.12). To do so, we calculate the Jacobian matrix as explained previously which includes initializing the value of μ. We used an initial value of μ as 10^{-2} and β as 2, and updated the weights. Using the updated weights, we recalculated $\hat{V}$ and continued with the training process as explained in Algorithm 3. Once the tolerance for $\hat{V}$ was reached, we stopped the training process. We compared the outputs (i.e., W_z of the hybrid model and the actual outputs from the training data) at the wellbore. It can be seen from Figure 3.5 that the outputs obtained using the proposed deep hybrid model closely mimic the output from the training data. This indicates that the DNN has been well trained and the hybrid model accurately predicts the outputs W_z. We used a relative error metric, $RE(t)$, to quantify the performance of the hybrid model in comparison to the training data. The relative error is calculated by using the Frobenius norms of the state vectors as follows:

$$RE(t) = \frac{\|\mathbf{W}_{training}(t) - \mathbf{W}_{hybrid}(t)\|_{fro}}{\|\mathbf{W}_{training}(t)\|_{fro}} \tag{3.29}$$

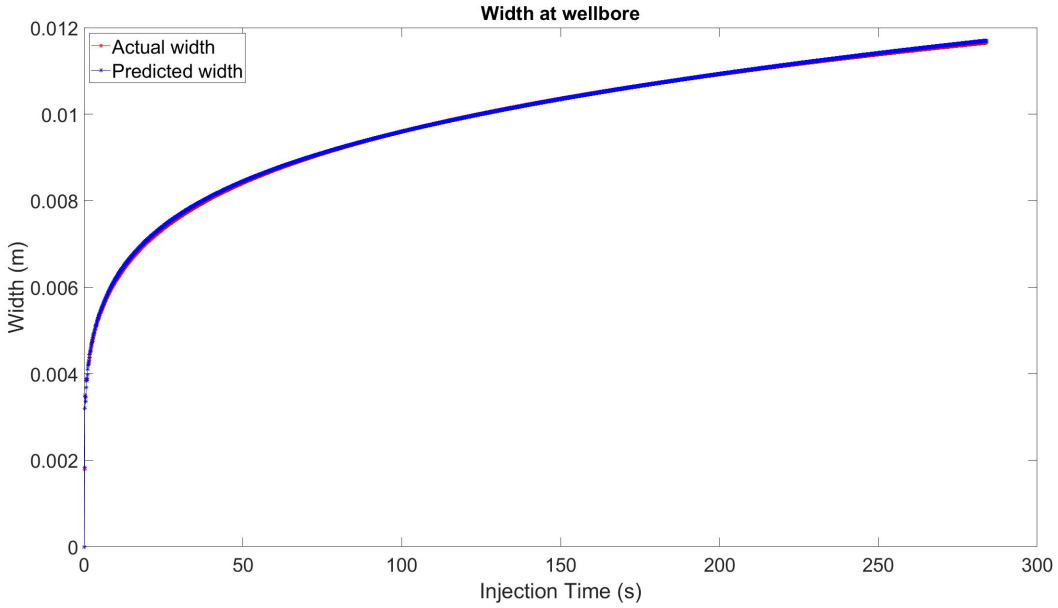


Figure 3.5: Comparison of wellbore widths obtained from the hybrid model, and the training data.

where $\|.\|_{fro}$ is the Frobenius norm, $\mathbf{W}_{training}(t)$ and $\mathbf{W}_{hybrid}(t)$ are the width vectors obtained from the training data and the hybrid model, respectively, at time t. The relative error in the widths predicted by the hybrid model in comparison with the training data at different times is presented in Figure 3.6. In order to compare the performance of the DNN in approximating the underlying phenomena of leak off, we used the obtained DNN and computed the leak off rate U values by presenting time inputs t. The predicted U values are compared against the actual leak off rate values obtained using Eq. (3.28), and the comparison is shown in Figure 3.7.

From Figure 3.6, we note that the DNN has been well trained and was able to accurately approximate the underlying leak off rate in the training data. A point to observe here is that there is slight inaccuracy in the approximation of the leak off rate U by the DNN in the initial stages of pumping whose effect can also be seen in the larger relative errors as shown in Figure 3.6.

3.1.6.1 Comparison of Deep hybrid model and black box model

The deep hybrid model utilizes a DNN to approximate the unobserved process parameters. As discussed previously, the structure of the hybrid model is similar to the first principles model except the DNN which does not alter the nature and characteristics of the known parameters in the first

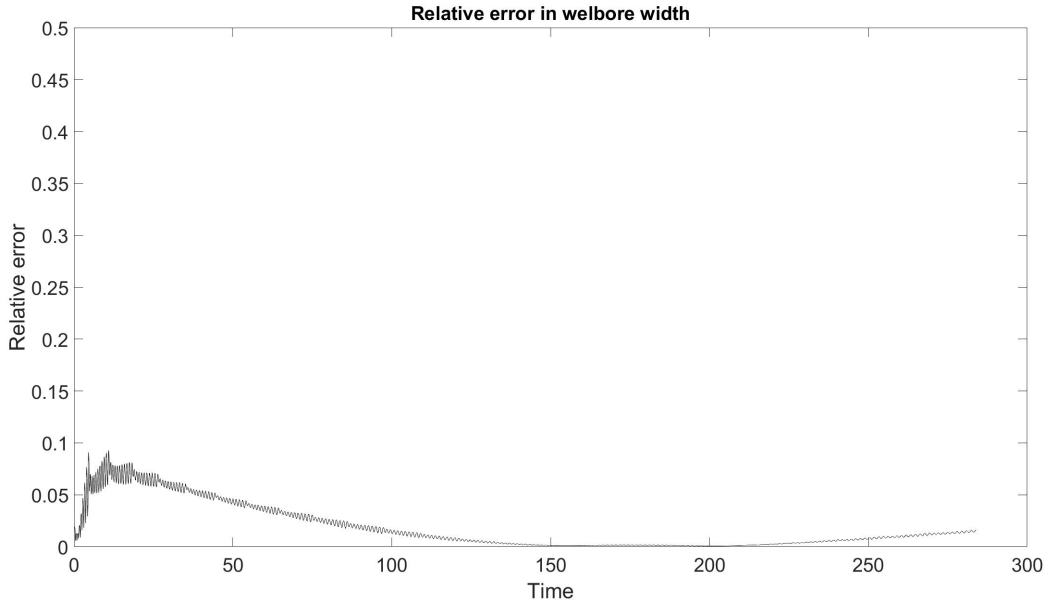


Figure 3.6: Relative error of the hybrid model predictions in comparison to the training data.

principles model. The presence of the first principles model in the hybrid model enables it with better extrapolation properties. On the other hand, the parameters of a black box model, which are purely data-driven, do not carry any physical meaning and such models have narrow applicability. To quantitatively prove this point, we built a black box model using a DNN containing 2 hidden layers, and 2 outer layers. We used the same training data as was in the case of hybrid model training in order to train the black box model. Once trained, we compared the output of the black box model with the actual output from the training data. It can be seen from Figure 3.8 that the outputs obtained using the black box closely mimics the output from the training data. This indicates that the DNN has been well trained and the black box model accurately predicts the width when the training input $Q_0 = 0.03\ \mathrm{m}^3/\mathrm{s}$ is presented to it. In contrast to the hybrid model, the parameters of the black box model (i.e., the DNN parameters) do not carry any physical meaning and hence, cannot give any insights about the process. To compare the extrapolation properties of the deep hybrid model and the black box model, we presented to them another input of $Q_0 = 0.04$ m^3/s, which is a small deviation from the training input.

Figure 3.9 shows the comparison of the widths W_z from the hybrid model, black box model and the true data at the wellbore. Although the DNN-based black box model captures the trend in

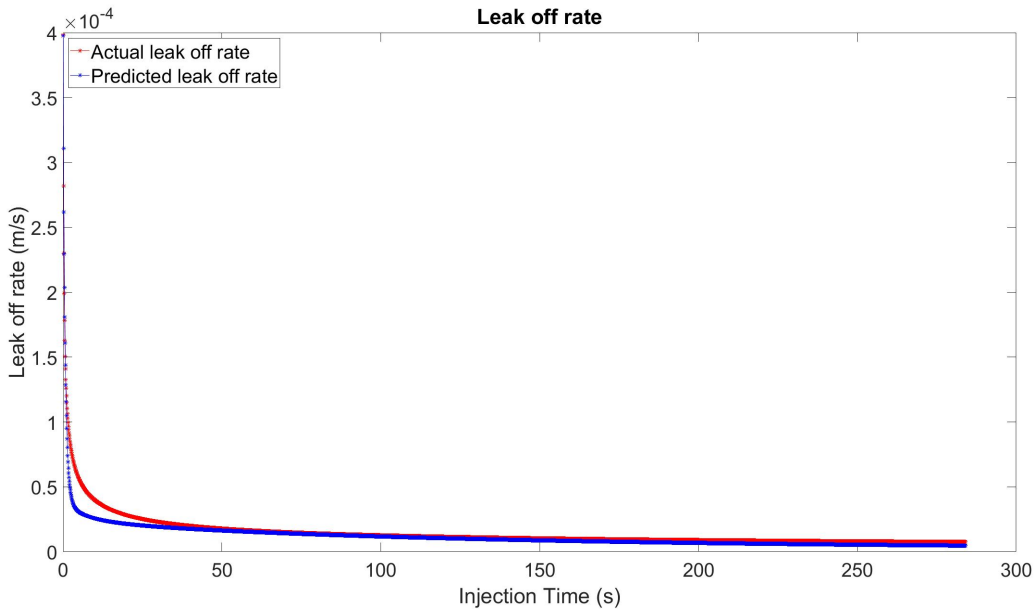


Figure 3.7: Comparison of leak off rates predicted from the DNN and actual values calculated using Eq. (3.28).

the outputs but clearly has poor accuracy, whereas the hybrid model accurately predicts the output when presented with the test input. An important point to note here is that the difference between the training input ($Q_0 = 0.03\ \mathrm{m}^3/\mathrm{s}$) and the test input ($Q_0 = 0.04\ \mathrm{m}^3/\mathrm{s}$) is small; however, the performance of the black box model varies considerably in both these cases as shown in Figure 3.9. As explained previously, this inaccurately stems from the fact that the black box model can accurately approximate the relationship between the training inputs and the training outputs by utilizing various parameters which do not carry any physical significance. Hence, when a test input different from the training input is presented to it, the black-box model fails to replicate the accuracy seen in the case of training data. On the other hand, since the hybrid model retains the structure of the first principles model and its parameters carry physical meaning, it is able to accurately predict the output in the case of test input.

3.1.6.2 Comparison of Deep hybrid model and first principles model

The main objective of the deep hybrid model is to develop a DNN that enhances the performance of the first principles model by accurately predicting its unknown parameters. In this work, we developed a DNN to extract the parameter associated with the leak-off rate from the training

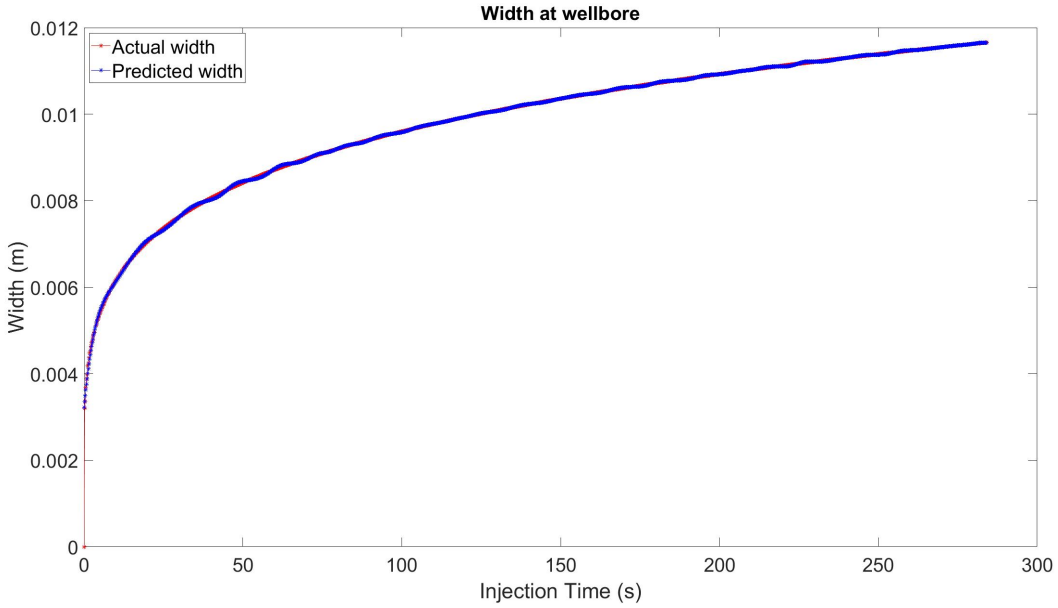


Figure 3.8: Comparison of wellbore widths obtained from the DNN-based black box model, and the training data.

data. In order to compare the performance of the hybrid model with only the first principles model, we assume a constant value for the leak off rate in the first principles model, and we do not make any changes to the deep hybrid model developed above. The three different cases of the first principles model were considered with three different values $(U = 1.2 \times 10^{-5}, 1.2 \times 10^{-6}, 6.3 \times 10^{-6})$ for the leak off rate. From Figure 3.10, it can be seen that the hybrid model accurately predicts, as expected, when compared to the first principles model. This superior performance is attributed to the accurate prediction of the parameter associated with the leak-off rate by the DNN in the hybrid model. On the other hand, the first principles model with $(U = 1.2 \times 10^{-5})$ performed on par with the hybrid model but the process described by it takes longer time to reach the desired fracture length, and the first principles models with $(U = 1.2 \times 10^{-6}\ and\ 6.3 \times 10^{-6})$ predicted poorly both in terms of accuracy as well as the time to reach the desired fracture length. The above comparison proves the superiority of hybrid model over first principles model in terms of accuracy as the hybrid model contains a DNN to accurately predict the leak off rate U values. A point to be noted here is that in all of the above results we utilized simulation data. In our future work, we plan to apply our proposed methodology for field/experimental data.

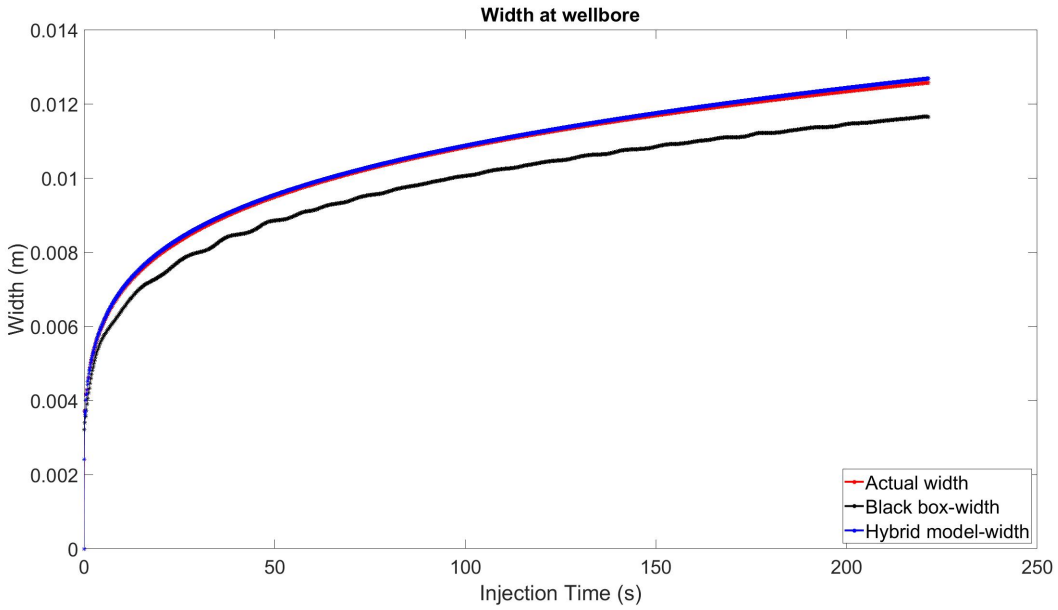


Figure 3.9: Comparison of wellbore widths obtained from the hybrid model, black box model and the test data.

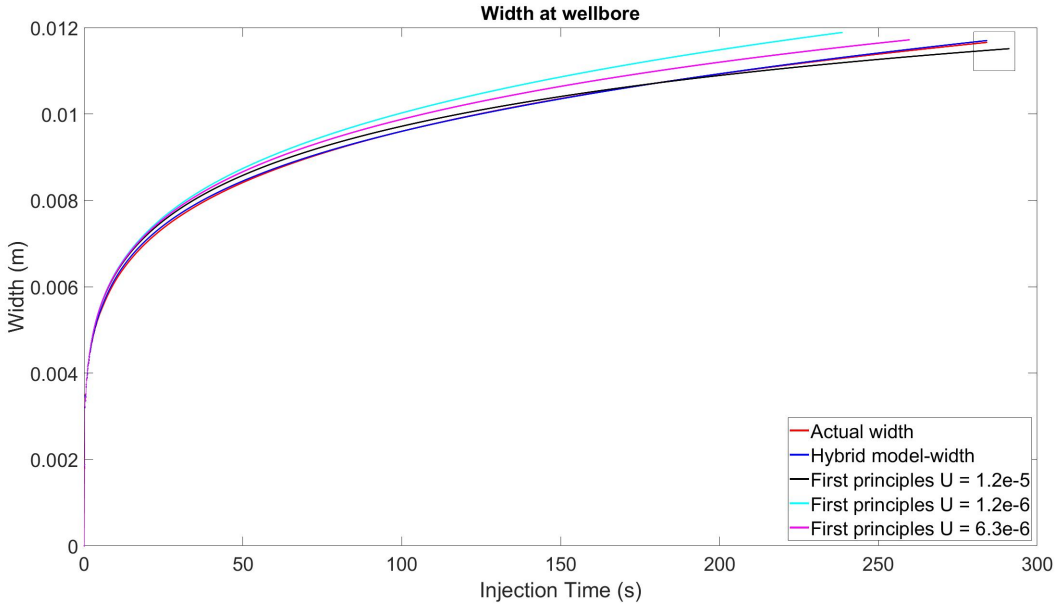


Figure 3.10: Comparison of wellbore widths obtained from the hybrid model, first principles model and the actual data.

3.2 Deep neural network-based hybrid modeling and experimental validation for an industry-scale fermentation process: Identification of time-varying dependencies among parameters*

Bio-fermentation processes are widely used for industrial production of many useful products such as chemicals, enzymes, food products, and pharmaceuticals. They involve the use of micro-organisms as 'catalysts' which convert substrates to products of interest. These micro-organisms can be bacteria, fungi, mammalian cells, etc., and are often optimized and engineered to achieve greater yields of product than observed in naturally occurring systems. These processes are advantageous over chemical processes as they are sustainable due to there low-temperature and low-pressure operations, and no requirements for harsh chemicals [117]. A typical bio-fermentation process is carried out in two phases. During the first phase, a bulk of substrate is combined with micro-organisms and other essential nutrients which are required for their growth. During this phase, the micro-organisms consume the available substrate, and consequently, there is an increase in the biomass concentration. In the second phase, other additional substrates are continuously fed into the reactor, and the rates of feeding are heavily regulated to avoid overfeeding or underfeeding which can significantly reduce the productivity of the process. Product is recovered from the reactor either continuously during the process or at the end of the second phase.

Modeling a bio-fermentation process is a challenging task given the complex interactions that occur within it. Usually, a first-principles model is developed using mass and energy conservation laws, kinetic laws, thermodynamic laws, etc., and it is able to capture the essential dynamics of the process. For this reason, building such a model requires significant time, resources, and

*Reprinted with permission from "Deep neural network-based hybrid modeling and experimental validation for an industry-scale fermentation process: Identification of time-varying dependencies among parameters" by Shah, P., Sheriff, M. Z., Bangi, M. S. F., Kravaris, C., Kwon, J. S., Botre, C., and Hirota, J. 2020. Chem. Eng. J., 441, 135643, Copyright 2022 Elsevier.

process insight. Additionally, due to the complex nature of the process, some mechanisms within the process are not understood to a level that they can be accurately modeled, and in such cases, empirical formulations are introduced in the first-principles model. The overall accuracy of the first-principles model is dependent on these empirical relationships. On the other hand, a data-based model can be developed using historical process data, which is easy to build and is accurate in its training regime, but will not be robust over a wide range of operating conditions of the process [9, 118, 97]. Another class of models called hybrid models exist which are a combination of first-principles models and data-based models [20].

Given the advantages of a hybrid model, it has been widely used to model lab-scale or pilot-scale bio-fermentation processes. It usually involves an ANN as a function approximator predicting unknown parameters or states to be combined with a first-principles model [119, 120, 121, 122]. The resulting hybrid model shows superior model accuracy compared to the first-principles model. But the field of neural networks has evolved from the use of a single hidden layer in an ANN to the use of multiple hidden layers in a DNN which requires exponentially less number of neurons than their shallow counterparts to approximate a specific function [123]. Additionally, a hybrid modeling approach has never been applied to a full-scale bio-fermentation reactor.

Motivated by these limitations, we developed a DNN-based hybrid model approach for a full-scale bio-fermentation process with a volume of over 100,000 gallons. Specifically, prior to building the hybrid model, we first improved the first-principles model by adding additional components and parameters to its equations based on process knowledge acquired from literature studies. This improved first-principles model was tested against multiple experimental datasets provided by an industry sponsor. Then, we identified critical parameters in the improved first-principles model which highly influence its outputs using local and global sensitivity analysis. Finally, these time-varying parameters were then estimated using a data-clustering approach, and approximated using a DNN which was then combined with the first-principles model to build a hybrid model. This hybrid model's performance was tested against multiple batches of process data provided by the industry sponsor, and compared against the accuracy of the first-principles model and the improved

first-principles model [124].

3.2.1 First-principles model of the bio-fermentation process

The first-principles model presented and discussed throughout this work is based on real process data provided by an industry sponsor, where the original first-principles model is similar to one used in [125]. The growth rate model is as follows:

$$\mu = \mu_{S_1} + \mu_{S_2} + \mu_I \tag{3.30}$$

$$\mu_{S_1} = \frac{\mu_{max,S_1} \cdot \xi_{S_2} \cdot \xi_I \cdot S_1}{K_{S,S_1} + S_1 + a_{S_1,S_2} \cdot S_2 + a_{S_1,I} \cdot I} \tag{3.31}$$

$$\mu_{S_2} = \frac{\mu_{max,S_2} \cdot S_2}{K_{S,S_2} + S_2 + a_{S_2,S_1} \cdot S_1 + a_{S_2,I} \cdot I} \tag{3.32}$$

$$\mu_I = \frac{\mu_{max,I} \cdot I}{K_{S,I} + I + a_{I,S_1} \cdot S_1 + a_{I,S_2} \cdot S_2} \tag{3.33}$$

where, μ, μ_i, and $\mu_{max,i}$ refer to the overall growth rate, the growth rates associated with each component (i.e., Substrate 1 (S_1), Substrate 2 (S_2), Intermediate (I)), and the maximum specific growth rate of the micro-organisms associated with each component, respectively. $K_{S,i}$ and $a_{i,j}$ refer to the half-velocity constant associated with each component, and the inhibitory effect of component i on utilization of component j by the micro-organisms, respectively. It is important to note that $\xi_{S_2} \cdot \xi_I$ in Eq. (3.31) is incorporated with μ_{max,S_1} to account for any inhibitory effects Substrate 2 and Intermediate have on the growth rate associated with Substrate 1. The process has two operation modes, i.e., phase 1 and phase 2, and since the nutrient source is different in these two phases, the respective reactor models for each phase are different.

During phase 1, the reactor model can be described by the following equations:

$$\mu = \mu_{S_1} \tag{3.34}$$

$$\mu_{S_1} = \frac{\mu_{max,S_1} \cdot \xi_{S_2} \cdot \xi_{S_I} \cdot S_1}{K_{S,S_1} + S_1 + a_{s_1,s_2} \cdot S_2 + a_{s_1,I} \cdot I} \tag{3.35}$$

$$\frac{dB}{dt} = (\mu_{S_1} + \mu_{S_2} + \mu_I) \cdot B \tag{3.36}$$

$$\frac{dS_1}{dt} = -\frac{\mu_{S_1} \cdot B}{Y_{B/S_1}} \tag{3.37}$$

where B refers to Biomass, and Y_{B/S_1} refers to the yield coefficient of Biomass associated with Substrate 1.

During phase 2, the reactor model can be described by the following equations:

$$\frac{dB}{dt} = (\mu_{S_1} + \mu_{S_2} + \mu_I) \cdot B - mp_1 \cdot \frac{F_{in}}{V} \cdot B \tag{3.38}$$

$$\frac{dS_1}{dt} = -\frac{\mu_{S_1} \cdot B}{Y_{B/S_1}} - \frac{F_{in}}{V} \cdot S_1 \tag{3.39}$$

$$\frac{dS_2}{dt} = k_1 \cdot \mu_{S_1} \cdot B - \frac{\mu_{S_2} \cdot B}{Y_{B/S_2}} - \frac{F_{in}}{V} \cdot (S_2 - S_{2_{initial}}) \tag{3.40}$$

$$\frac{dI}{dt} = (k_2 \cdot \mu_{S_1} + k_3 \cdot \mu_{S_2}) \cdot B - \frac{\mu_I \cdot B}{Y_{B/I}} - \frac{F_{in}}{V} \cdot I \tag{3.41}$$

$$\frac{dP}{dt} = (\alpha_1 \cdot \mu_{S_1} + \alpha_2 \cdot \mu_{S_2} + \alpha_3 \cdot \mu_I) \cdot B + \beta \cdot B - \frac{F_{in}}{V} \cdot P \tag{3.42}$$

$$\frac{dV}{dt} = F_{in} \tag{3.43}$$

where P is the Product concentration, V is the reactor volume, α_i is the coefficient linked to the growth rate responsible for increase in Product, k_i refers to the coefficient linked to the growth rate responsible for the increase in Substrate 2 and Intermediate, β is the coefficient linked to the non-growth associated term responsible for the increase in Product, and F_{in} refers to the feed flow rate of Substrate 2. The parameters Y_{B/S_1}, Y_{B/S_2}, and $Y_{B/I}$ refer to the yield coefficients of Biomass associated with each component. It should be noted that the coefficients linked to the growth rate, k_i, incorporate temperature dependence through Arrhenius equation as follows:

$$k_i = c_i \cdot e^{\frac{-Ea_i}{RT}} \tag{3.44}$$

where c_i is the pre-exponential factor, E_{a_i} is the activation energy, R is the universal gas constant,

and T refers to the temperature.

It is important to note that this is the original model that was provided by the industry sponsor, who also introduced a manipulated parameter (mp_1) to the second term of Eq. (3.38) associated with Biomass in order to obtain a better fit. The following section will provide a summary of modifications that were carried out on the original first-principles model to improve its prediction capability. It is also important to note that all results presented in this work are normalized, at the request of the industry sponsor.

3.2.2 Improving the first-principles model

As the estimation results for the original first-principles model are not ideal for certain states, an improvement in the original first-principles model was first pursued through the incorporation of two additional components. These components may account for discrepancies encountered when utilizing the original first-principles model, and these modifications will be discussed next. These modifications include the incorporation of two components X_1 and X_2. Component X_1 is a manipulated input for which continuous values are available, and it behaves similar to catalyst when added during phase 2 of the process. Component X_2 is an essential chemical which ensures optimal conditions for micro-organisms. Measurements of component X_2 are also available, and the incorporation of both components in the original first-principles model is highly desirable.

3.2.2.1 Incorporation of component X_1

Through information obtained from historical operation data, the addition of component X_1 in phase 2 was observed to increase the production of the micro-organisms, consequently resulting in an increase in the consumption of Substrate 2, and thus, an increase in the production of Product. This information was utilized to update Eqs. (3.40) and (3.42) through the addition of empirical terms as follows:

$$\frac{dS_2}{dt} = k_1 \cdot \mu_{S_1} \cdot B - \frac{\mu_{S_2} \cdot B}{Y_{B/S_2}} - \frac{F_{in}}{V} \cdot (S_2 - S_{2_{initial}}) - p_1 \cdot X_1 \tag{3.45}$$

$$\frac{dP}{dt} = (\alpha_1 \cdot \mu_{S_1} + \alpha_2 \cdot \mu_{S_2} + \alpha_3 \cdot \mu_I) \cdot B + \beta \cdot B - \frac{F_{in}}{V} \cdot P + p_2 \cdot X_1 \tag{3.46}$$

where p_1 and p_2 are empirical coefficients that allow X_1 to be incorporated as consumption and production terms in the equations for Substrate 2 and Product, respectively.

It is important to note that since X_2 functions in a way similar to catalyst, it was initially incorporated in the growth rate coefficient related to μ_{S_2} in Eq. (3.32). Unfortunately, satisfactory results were not obtained through this approach, and the empirical terms shown in Eqs. (3.45) and (3.46) had to be introduced instead.

3.2.2.2 Incorporation of component X_2

Through information obtained from historical operation, it is understood that measurements from an online sensor are available for X_2. Therefore, it is desirable to incorporate component X_2 as well. X_2 is expected to play a role similar to the role played by oxygen in bio-fermentation processes, and was hence incorporated as follows [126]:

$$\mu = (\mu_{S_1} + \mu_{S_2} + \mu_I) \cdot \left(\frac{X_2}{K_{X_2} + X_2} \right) \tag{3.47}$$

$$\frac{dX_2}{dt} = k_L a \cdot (X_{2_{max}} - X_2) - q_{X_2} \cdot B \tag{3.48}$$

where $X_{2_{max}}$ and K_{X_2} are the maximum value of X_2 during a particular batch run, and the half-velocity constant associated with X_2, respectively. $k_L a$ and q_{X_2} are the mass transfer coefficient, and uptake rate of X_2, respectively. It should be noted that online measurements for X_2 are available throughout phase 1 and phase 2 of the bio-fermentation process. An additional objective of the industry sponsor was to incorporate X_2 in the original first-principles model and predict it like the other states, so that its estimates can be tracked in real-time.

3.2.3 Sensitivity analysis and clustering

To develop a hybrid model that utilizes all available information (i.e., first-principles), sensitivity analysis needs to be carried out to identify highly sensitive model parameters.

3.2.3.1 Sensitivity analysis

Before modifying the model further by considering new measurements or using data-driven approaches to improve the existing model, it is essential to perform a sensitivity analysis to understand which parameters greatly influence the outputs. In this section, a local and global sensitivity analysis of the model is presented.

A local sensitivity analysis around the nominal values of the model parameters is initially carried out to understand how the model parameters and initial conditions influence the different outputs, i.e., Substrate 1, Substrate 2, Biomass, Product, Intermediate, and X_2.

A sensitivity matrix is first derived in order to come up with the parameter set which affects the outputs. The sensitivity matrix shows the dependencies of the outputs with respect to the parameters and initial conditions, as shown below:

$$S = \begin{bmatrix} \partial y_1(t_1)/\partial\theta_1 & \dots & \partial y_1(t_1)/\partial\theta_{n_\theta} \\ \vdots & \ddots & \vdots \\ \partial y_1(t_{n_t})/\partial\theta_1 & \dots & \partial y_1(t_{n_t})/\partial\theta_{n_\theta} \\ \vdots & \ddots & \vdots \\ \partial y_{n_y}(t_1)/\partial\theta_1 & \dots & \partial y_{n_y}(t_1)/\partial\theta_{n_\theta} \\ \vdots & \ddots & \vdots \\ \partial y_{n_y}(t_{n_t})/\partial\theta_1 & \dots & \partial y_{n_y}(t_{n_t})/\partial\theta_{n_\theta} \end{bmatrix} \tag{3.49}$$

Here, yϵR^{n_y} represents the output states, and $\theta\epsilon R^{n_\theta}$ represents the parameters and initial condition whose sensitivity analysis is carried out. These sensitivity matrix values are typically normalized by multiplying with the nominal values of parameters and by dividing through the nominal values of the outputs to ensure that different units for the parameters/outputs do not affect the sensitivity analysis results.

To capture the effect that these parameters have on the outputs, a criterion called Fisher information matrix is calculated in the form of sensitivity matrix as follows:

$$FIM = S^T \Sigma S \tag{3.50}$$

where Σ is an identity matrix. A specific criterion is required to evaluate the information contained in the Fisher information matrix, and for this purpose, the D-optimality criterion (ϕ_D) is used. It minimizes the logarithm of the determinant of the inverse of the Fisher information matrix. Using inverse determinant property, we get:

$$\phi_D^* = max\ \phi_D\ (FIM) = max\ log\ det\ (FIM) \tag{3.51}$$

For the purpose of this sensitivity analysis, one parameter is evaluated at a time, and ϕ_D is computed to show the effect it has on an output. A higher ϕ_D value implies that the concerned model parameter has a higher influence on the given output. As the process states in the first-principles model are measurable, these states are the model outputs in the bio-fermentation process, i.e., y = x where x is the state. It is important to note that the study is carried out for the outputs of the reactor model for phase 2, since it is the most important phase of the bio-fermentation process as Substrate 2 is present in this phase and majority of Product is formed in this phase. Substrate 2 is tied to the operating cost of the process and is the main energy source for the micro-organisms. As the supply of Substrate 1 is limited, it is primarily used for the initial growth of Biomass, and only a relatively small percentage of product is formed in phase 1 when compared to phase 2. To calculate the sensitivity of states with respect to the parameters, the following equation is solved:

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial x}{\partial \theta_i} = \frac{\partial f}{\partial x^T}\frac{\partial x}{\partial \theta_i} + \frac{\partial f}{\partial \theta_i} \tag{3.52}$$

When θ represents an initial condition of the state in Eq. (3.57), the second term ($\frac{\partial x}{\partial \theta_i}$) is 1, and for all other parameters, this term is 0. For the case of this bio-fermentation process, the dimension of the overall sensitivity matrix is $[195, 6, 38]$ where 195 denotes the number of time instants in the process, 6 denotes the number of states (i.e., Substrate 1, Biomass, Product, Substrate 2, Intermediate, added component X_2), and 38 is the number of parameters including the initial conditions.

Equal weight to outputs	ϕ_D	Substrate 2/Product with 5 times weight	ϕ_D
$S_{2_{initial}}$	22.7	V_0	68.6
Y_{B/S_2}	22.6	$S_{2_{initial}}$	68.01
V_0	22.3	Y_{B/S_2}	67.8
B_0	9.02	α_2	27.4
μ_{max,S_2}	8.66	μ_{max,S_2}	25.9
K_{S,S_2}	8.24	K_{S,S_2}	24.6
α_2	5.47	B_0	6.08
man_para	0.858	β	2.08
β	0.416	G_0	0.018
G_0	0.018	P_0	-5.77

Table 3.2: Local sensitivity analysis: a list of sensitive parameters with D-optimality criterion (ϕ_D) values for (a) when output states in the model for phase 2 are equally important, and (b) when Substrate 2 and Product are 5 times more weighted than the other states.

Overall, in this sensitivity analysis study, we consider the effect of these 38 parameters on the output states. The parameters consist of six initial conditions corresponding to the six states, and 32 parameters from the growth rate and phase 2 reactor model.

Since the local sensitivity analysis was carried out initially, all the parameters were considered to be at the nominal values. The result from this study is shown in Table 3.2. Two cases are examined, the first case is where all the output states are assumed to be equally important, and the second case is where Substrate 2 and Product are considered 5 times more weighted than the rest, as they are the primary states of interest and need to be predicted accurately. Parameters are listed according to decreasing order of ϕ_D for both the cases. The importance of the parameter is determined by how high it appears in the table.

For the first case where output states are assumed to be equally weighted, it is seen that the yield coefficient associated with Substrate 2 and the initial concentration of Substrate 2 being fed into the reactor are particularly important, along with the initial conditions of the state. Additionally, μ_{max,S_2}, K_{S,S_2}, and α_2 are also considerably important. The initial conditions of the output and the initial concentration of pure Substrate 2 flowing into the fermenter, $S_{2_{initial}}$, cannot be estimated using optimization. This is because, their value is subject to the real-time operation of the bio-fermentation plant, varying with different batches. Thus, the main focus is on the parameters

present in the growth rate and phase 2 that can be better estimated in order to attain a superior output prediction compared to the revised first-principles model.

From the local sensitivity analysis, it can be concluded that regardless of the weight to the states, the initial concentration of Substrate 2 in the feed, the yield coefficient with respect to Substrate 2, and the initial value of Volume are important parameters. The initial conditions of most of the states are important. The half-velocity constant K_{S,S_2}, which is associated with Substrate 2, is the only half-velocity constant that is important. The non-associated growth term β is an important parameter affecting Product. The maximum specific growth rate associated with Substrate 2, μ_{max,S_2}, and the growth rate coefficient associated with Substrate 2 responsible for the increase in Product, α_2, also play important roles in affecting the outputs.

As the parameters generally vary a lot depending on different batches, operating conditions, and changes in measurements, it is important to see how sensitive the outputs are, based on a wide range of parameter values, i.e., through a global sensitivity analysis. From global sensitivity analysis, we can identify the parameters and initial conditions that are the most important and significantly affect the outputs, particularly, Substrate 2 and Product concentrations. To decide the range of parameter values upon which the global sensitivity analysis model would be developed, 5 experimental datasets were run with a slight change in certain conditions, e.g., initial parameter guess, parameter bounds, etc. Based on the estimated parameter values obtained from these runs, an overall range was decided for each parameter and initial condition, leaving a 20-50% margin of error to account for the maximum range of values possible. A wider range is generally preferred because different batches can estimate different parameter values, and it is helpful to see how the outputs might react to parameter values that are far from their nominal value.

For the global sensitivity analysis, we use $lhsdesign$ (Latin Hypercube Sampling) in MATLAB to construct a matrix of random values between 0 to 1 for each parameter. 100 different cases of parameter sets are considered for this analysis and are averaged at the end. The overall combined global sensitivity analysis of the model is done using the formulation below:

No.	Substrate 1	Biomass	Product	Substrate 2	Intermediate	Added component X_2
Parameters						
1	Y_{B/S_1}	Y_{B/S_2}	α_2	$a_{S_2,I}$	$a_{S_2,I}$	V_0
2	G_0	E_0	Y_{B/S_2}	E_0	μ_{max,S_2}	k_La
3	μ_{max,S_1}	$S_{2_{initial}}$	E_0	μ_{max,S_2}	E_0	$S_{2_{initial}}$
4	–	$a_{S_2,I}$	$S_{2_{initial}}$	Y_{B/S_2}	Y_{B/S_2}	q_{X_2}
5	–	man_para	β	a_{S_2,S_1}	a_{S_2,S_1}	Y_{B/S_2}
6	–	B_0	$a_{S_2,I}$	K_{S,S_2}	$a_{S_2,I}$	p_1
7	–	μ_{max,S_2}	μ_{max,S_2}	$S_{2_{initial}}$	K_{S,S_2}	Y_{B/S_1}
8	–	a_{S_2,S_1}	V_0	V_0	$S_{2_{initial}}$	μ_{max,S_2}
9	–	V_0	c_3	man_para	c_3	K_{S,S_2}
10	–	c_3	a_{S_2,S_1}	a_{I,S_2}	$\mu_{max,I}$	B_0
11	–	K_{S,S_2}	P_0	B_0	$Y_{B/I}$	E_0
12	–	$Y_{B/I}$	K_{S,S_2}	c_3	man_para	$Y_{B/I}$

Table 3.3: Global sensitivity analysis: a list of sensitive parameters for each output present in the revised first-principles model for phase 2.

$$\begin{aligned}\phi_D(model) = \phi_D(Substrate\ 1) + \phi_D(Biomass) + \phi_D(Intermediate) \\ +5 \cdot [\phi_D(Substrate\ 2) + \phi_D(Product)]\end{aligned} \tag{3.53}$$

The range (lower and upper bounds LB and UB) of each parameter considered is shown below:

$$Range\ of\ each\ parameter = [LB \quad LB + (UB - LB) \cdot lhsdesign] \tag{3.54}$$

The effect of all these 38 parameters on the outputs is studied, individually and for the model as a whole. The results of global sensitivity analysis for individual outputs are summarized in Table 3.3, where the importance of the parameter is determined by how high it appears in the table. Most of the parameters in Table 3.3 have a positive ϕ_D value and parameters not included in this table are negative, implying that they do not affect the outputs significantly.

It should be noted that the parameters are much more significant for the global sensitivity analysis, compared to the local sensitivity analysis where the initial conditions were of high significance to the outputs. It is seen that the yield coefficient associated with Substrate 2, initial concentration

Substrate 2/Product with 5 times weight	ϕ_D
E_0	65.0
$a_{S_2,I}$	59.9
Y_{B/S_2}	55.4
μ_{max,S_2}	38.6
α_2	37.34
$S_{2_{initial}}$	29.4
β	16.9
a_{S_2,S_1}	6.80
K_{S,S_2}	-7.14
P_0	-8.74

Table 3.4: Global sensitivity analysis: a list of sensitive parameters with D-optimality criterion (ϕ_D) values when Substrate 2 and Product are 5 times more weighted than the other states.

of Substrate 2 when it is fed into the fermenter, inhibition parameters $a_{S_2,I}$ (effect of Substrate 2 on utilization of Intermediate by micro-organisms), and maximum specific growth rate of the micro-organisms associated with Substrate 2 are the most important parameters. Additionally, β (non-growth associated term responsible for the increase in Product) and α_2 (coefficient associated with Substrate 2 responsible for the increase in Product) are sensitive to Product. K_{S,S_2} (Half velocity associated with Substrate 2) is sensitive to Substrate 2. Only three parameters are seen to affect Substrate 1 in phase 2, and the rest of the parameters have a negative or zero ϕ_D value.

In combined global sensitivity analysis of the developed revised first-principles model, the effect of all outputs is considered together. Substrate 2 and Product outputs are given five times more weight than Substrate 1, Biomass, and Intermediate. As mentioned earlier, the reason for that is the need for accurate prediction of these two states. A summary of the results of the combined global sensitivity analysis is shown in Table 3.4. From Table 3.4, the following conclusions can be made:

- Regardless of the weight to the states and global/local analysis, it should be noted that the initial concentration of Substrate 2 in the feed, $S_{2_{initial}}$, the yield coefficient with respect to Substrate 2, Y_{B/S_2}, coefficient associated with Substrate 2 responsible for the increase in Product, α_2, and maximum specific growth rate associated with Substrate 2, μ_{max,S_2}, are important parameters.

- The initial condition of all states is important for local analysis, but not as important for global analysis.
- The half-velocity constant associated with Substrate 2, K_{S,S_2}, appears to be the only half-velocity constant of significant importance.
- Inhibition parameters are very important and sensitive to the model according to the global sensitivity analysis, especially the ones associated with Substrate 2.
- The non-associated growth term, β, is an important parameter affecting Product.

The results of the sensitivity analysis can be utilized in order to examine variation in sensitive model parameters, through parameter clustering as presented in the following section.

3.2.4 Improving the revised first-principles model through clustering

The sensitivity analysis identified μ_{max,S_2}, $\mu_{max,I}$, and K_{S,S_2} as the sensitive model parameters for the growth rate, and Y_{B/S_2}, c_3, α_2, and k_La as the sensitive model parameters for phase 2. Some of the other sensitive parameters like $a_{S_2,I}$ and other inhibition parameters are sensitive to the outputs but they do not vary with time. The identified model parameters can now be utilized in a parameter clustering approach to observe their variation through the course of the bio-fermentation process, and to determine which parameters might benefit from a hybrid model approach. This approach is beneficial if there is no first-principles model to define potential time-varying parameters. In these cases, DNNs are used to develop a relation between frequently available online measurements and estimated parameters. This allows the model accuracy to be improved by utilizing time-varying parameters rather than a single estimate for the given model parameter.

To the knowledge of the authors, there were no first-principles models for any of the sensitive model parameters that were identified, and thus, a clustering approach was pursued to determine if there were large variations in the sensitive model parameters. In this approach, sensitive parameters are estimated separately in different clusters of time. This approach provides different estimates for the sensitive model parameter in each cluster, thus enabling time-varying parameters to be

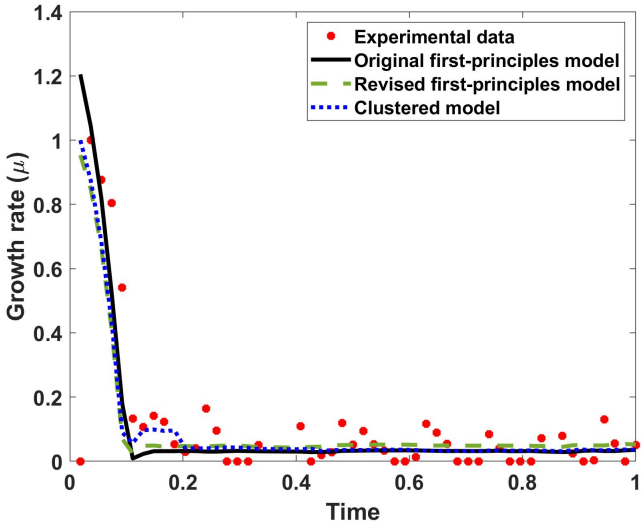


Figure 3.11: A comparison of growth rate parameter estimation using the first-principles model, revised first-principles model, and clustered model.

obtained.

Experimental datasets provided by the industry sponsor included the values at 50 different time instants for each state. These were used to create 5 clusters, each comprised of 10 values. Insensitive growth parameters values were fixed. This is done since limited experimental data is available, and re-estimating all parameters may lead to over-fitting.

A comparison of the simulation results of all the developed models, i.e., first-principles model, revised first-principles model, and clustered model with experimental data for the growth rate, is provided in Figure 3.11. Here, improved estimation of the growth rate using the parameter clustering approach can be observed on the normalized time scale. But still the clustered model is unable to accurately track the time-varying nature of the growth rate characteristics. The parameters estimated by the clustered approach for the growth rate are presented in Table 3.5. Similarly, the parameters estimated by the clustered approach for the reactor model for phase 2 are presented in Table 3.6. These results demonstrate that the sensitive model parameters are time-varying, particularly the growth rate coefficients associated with Substrate 2 and Intermediate, μ_{max,S_2} and $\mu_{max,I}$, and yield coefficient associated with Substrate 2, $Y_{B/S_2,1}$. These parameters may benefit from developing a hybrid model, which will be explored in the following section.

Growth rate parameter	Time period (normalized)	Value	Unit
$\mu_{max,S_2,1}$	0.0-0.2	0.512	hr^{-1}
$\mu_{max,S_2,2}$	0.2-0.4	0.202	hr^{-1}
$\mu_{max,S_2,3}$	0.4-0.6	0.133	hr^{-1}
$\mu_{max,S_2,4}$	0.6-0.8	0.124	hr^{-1}
$\mu_{max,S_2,5}$	0.8-1.0	0.129	hr^{-1}
$K_{S,S_2,1}$	0.0-0.2	68.9	g Substrate 2 L^{-1}
$K_{S,S_2,2}$	0.2-0.4	2.03×10^2	g Substrate 2 L^{-1}
$K_{S,S_2,3}$	0.4-0.6	2.37×10^2	g Substrate 2 L^{-1}
$K_{S,S_2,4}$	0.6-0.8	2.85×10^2	g Substrate 2 L^{-1}
$K_{S,S_2,5}$	0.8-1.0	2.79×10^2	g Substrate 2 L^{-1}
$\mu_{max,I,1}$	0.0-0.2	0.972	hr^{-1}
$\mu_{max,I,2}$	0.2-0.4	0.772	hr^{-1}
$\mu_{max,I,3}$	0.4-0.6	0.687	hr^{-1}
$\mu_{max,I,4}$	0.6-0.8	0.555	hr^{-1}
$\mu_{max,I,5}$	0.8-1.0	0.612	hr^{-1}

Table 3.5: Clustered growth rate parameters.

3.2.5 Development of the hybrid model

3.2.5.1 Improving the revised first-principles model through hybrid modeling

In the previous section, sensitive model parameters were identified and estimated in a clustered manner, where each of the five values estimated for the parameters was used to improve model prediction. The parameters mentioned in Table 3.5 and Table 3.6 show that there was significant variation in their values with time, but parameters such as μ_{max,S_2}, $\mu_{max,I}$, and $Y_{B/S_2,1}$ change more frequently, and nonlinearly in time unlike k_La, and $c_{3,1}$. Thus, to get a more accurate representation of these parameters and capture their complete time-varying nature over all the time instants, a hybrid modeling approach was adopted.

As described previously, a hybrid model is one that utilizes a data-driven model along with a first-principles model. A DNN is trained to estimate μ_{max,S_2}, $\mu_{max,I}$, and $Y_{B/S_2,1}$, which are to be utilized in the improved first-principles model. The inputs to the DNN are the concentrations of Substrate 2, Biomass, Intermediate, Product, Volume, and X_2. As this approach is primarily for phase 2 model, Substrate 1 is not considered since it is experimentally known to be negligible

Phase 2 parameter	Time period (normalized)	Value	Unit
$Y_{B/S_2,1}$	0.0-0.2	0.855	g Cell/g Substrate 2
$Y_{B/S_2,2}$	0.2-0.4	0.155	g Cell/g Substrate 2
$Y_{B/S_2,3}$	0.4-0.6	0.135	g Cell/g Substrate 2
$Y_{B/S_2,4}$	0.6-0.8	0.115	g Cell/g Substrate 2
$Y_{B/S_2,5}$	0.8-1.0	0.156	g Cell/g Substrate 2
$c_{3,1}$	0.0-0.2	0.30	g Intermediate/g Cell
$c_{3,2}$	0.2-0.4	1.00	g Intermediate/g Cell
$c_{3,3}$	0.4-0.6	1.50	g Intermediate/g Cell
$c_{3,4}$	0.6-0.8	1.50	g Intermediate/g Cell
$c_{3,5}$	0.8-1.0	1.50	g Intermediate/g Cell
$\alpha_{2,1}$	0.0-0.2	0.011	g Product/g Substrate 2
$\alpha_{2,2}$	0.2-0.4	0.056	g Product/g Substrate 2
$\alpha_{2,3}$	0.4-0.6	0.051	g Product/g Substrate 2
$\alpha_{2,4}$	0.6-0.8	0.035	g Product/g Substrate 2
$\alpha_{2,5}$	0.8-1.0	0.010	g Product/g Substrate 2
k_La_1	0.0-0.2	13.4	hr^{-1}
k_La_2	0.2-0.4	3.36	hr^{-1}
k_La_3	0.4-0.6	3.36	hr^{-1}
k_La_4	0.6-0.8	2.36	hr^{-1}
k_La_5	0.8-1.0	2.06	hr^{-1}

Table 3.6: Clustered phase 2 parameters.

during this phase as it gets consumed almost completely in phase 1. The first layer is the input, and the last layer is the output. The nodes are connected using weights, and each node has a bias. Rectified Linear (ReLu) activation function is used to calculate the output of each node. The DNN used in this work consists of 3 hidden layers with 5 nodes each and 3 outputs, which are the parameters mentioned above. These parameters are then used in the first-principles model, and the output concentrations are calculated.

Now, say, x_k are the states of the improved first-principles model mentioned earlier, i.e., Substrate 2, Biomass, Product, Intermediate, X_2, and Volume, and u_k are the manipulated inputs to the process which will also be used as an input to the hybrid model such as temperature, added component X_1, alkali flow rate, and Substrate 2 flow rate. It is important to note that alkali flow rate is a critical input to the fermenter as it is added to keep the pH in check as it can neutralize Intermediate and added chemical X_1.

The DNN is initially pre-trained using the MATLAB deep learning toolbox, and the clustering parameter values. These parameters are used in the first-principles model for phase 2 wherein the other parameters are constant. The output concentrations from the hybrid model are represented as x_{k+1}, which will be used as input to the model in the next time step. Y_k is the plant measurement of these output concentrations. Once we have the output from the hybrid model, the SSE is calculated. Jacobian matrix is then calculated which is used to update the weights and biases. Levenberg Marquardt algorithm is used to update the DNN parameters. These updated weights and biases are then used in the next iteration, and the hybrid model gives a new set of outputs. Once again, the error is calculated, and SSE is computed. This process is repeated until the error is less than a tolerance value. It is important to note that unlike the clustering method which had 5 estimated values for each parameter, the DNN has 50 parameter values corresponding to 50 output measurements, thus estimating time-varying parameters in a much more accurate manner.

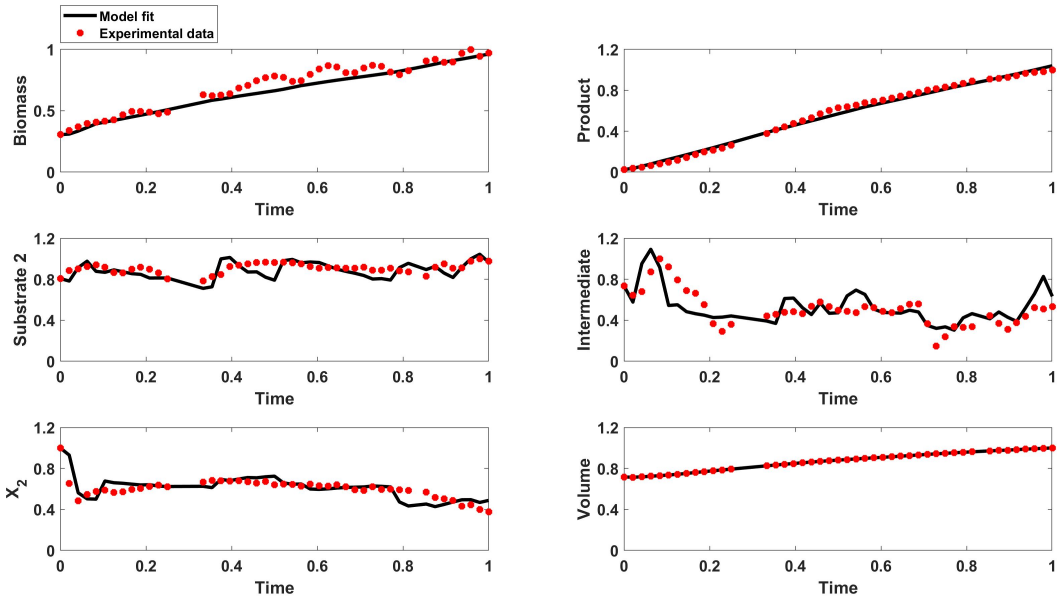


Figure 3.12: A comparison of the hybrid model and training data during phase 2.

The results for the hybrid model using the training data are illustrated in Figure 3.12. It can be seen that all the states are predicted well and the model fit is more accurate than the revised first-principles model's prediction, especially for Product and Substrate 2, which are the primary states of interest in this work. Estimation of states using validation data is shown in Figure 3.13,

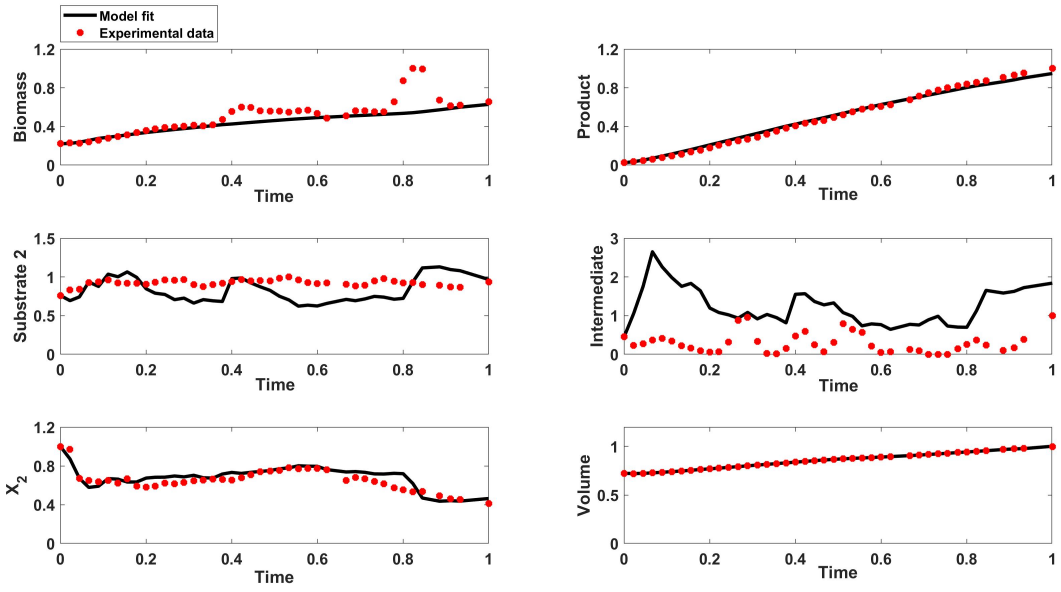


Figure 3.13: A comparison of the hybrid model and validation data during phase 2.

and it shows that all the states except Intermediate are predicted fairly accurately. There is an order of magnitude difference between the measurements of Intermediate from different batches due to uncertainty from the yeast cells and the significant effects X_1 and alkali flow rate have on it. Hence, its prediction for validation batch is not as accurate as the other states. The main concern with Intermediate concentration is regarding identification of abnormality in bio-fermentation process, and there are other means of tracking Intermediate that the industry sponsor uses. The results also show that the prediction accuracy of component X_2 is high thus the developed model can be used to successfully track X_2 along with the other states.

The parameters estimated by the hybrid model were used to further validate 2 additional batches, shown in Figure 3.14, and Figure 3.15. Due to difficulty in taking offline measurements, only Substrate 2 and Product concentrations are measured during normal operation of the bio-fermentation plant. The results show reasonable prediction for both these states using the two batches. For further improving the prediction accuracy, it is crucial to have more experimental data so that the neural network can be trained even more precisely as larger the sample size of data, the better the parameter estimates. For the objective of this work, all these validation plots show that the states, especially Substrate 2 and Product, are predicted reasonably well.

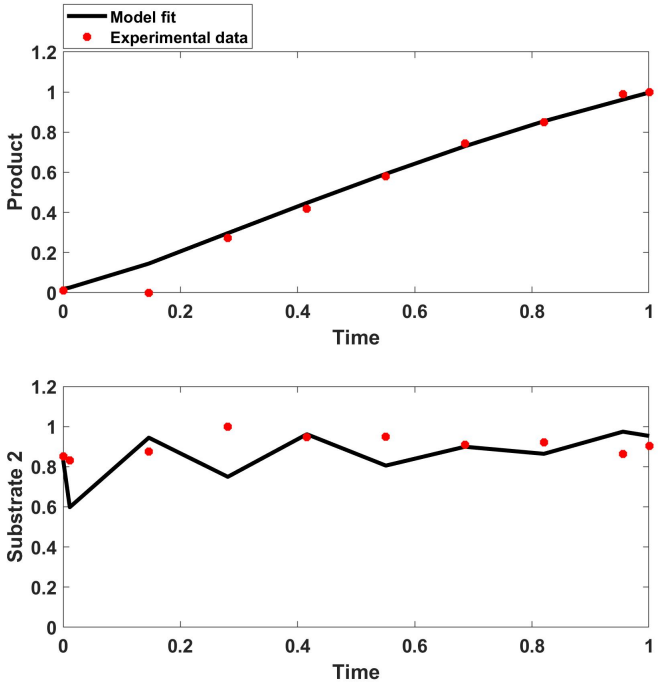


Figure 3.14: A comparison of the hybrid model and additional validation dataset 1, during phase 2.

3.2.6 Error analysis

In order to quantify and compare the performance of the three models, i.e., original first-principles model, revised first-principles model, and hybrid model, we utilize the relative error (RE) formulation as defined below:

$$RE_k = 1 - \frac{x_{k+1}}{Y_k} \tag{3.55}$$

In Eq. (3.55), x_{k+1} are the predicted state concentrations using the models and Y_k are the plant measurements, as described in detail in Section 5.1. First, performance of the three models is compared using the training batch that was used to train the DNN in the hybrid model. The predicted outputs from these three models were utilized to calculate the RE value as defined in Eq. (3.55) at every time step. The RE plots are plotted and compared in Figure 3.16. The key observation here is that the hybrid model outperforms the first-principles model and the revised first-principles model. This can be attributed to the fact that the hybrid model includes a trained

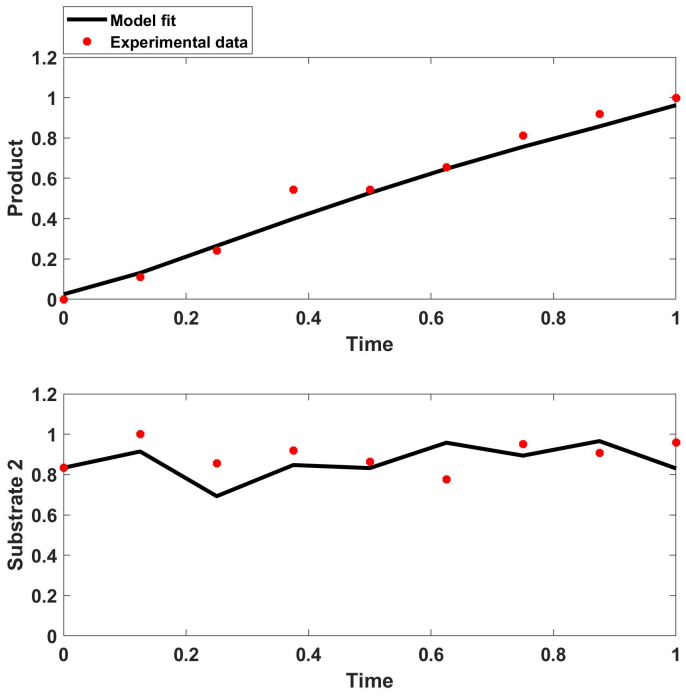


Figure 3.15: A comparison of the hybrid model and additional validation dataset 2, during phase 2.

DNN which accurately predicts the sensitive parameter values as well as the dependencies among themselves, and this results in better prediction of outputs.

Next, the performance of the three models is compared using the validation batch. The predicted outputs were utilized to calculate the RE value as defined in Eq. (3.55). Once again, it can be observed from Figure 3.17 that the hybrid model performs much better than the other two models, except in the case of the prediction of Intermediate, where the performance of the hybrid model is comparable to the first-principles model.

Additionally, the error is numerically quantified using the root mean squared error (RMSE). RMSE values were calculated by comparing the predictions from all three models against the training and validation batches, and are summarized in Table 3.7 and Table 3.8, respectively. From these tables, it can be observed that the hybrid model outperforms the first-principles model and the revised first-principles model in the prediction of all the states except for the Intermediate. Moreover, the RMSE values for Product and Substrate 2 for the two additional validation batches using the hybrid model were found to be low: 0.0625 (Product) and 0.1279 (Substrate 2) for the

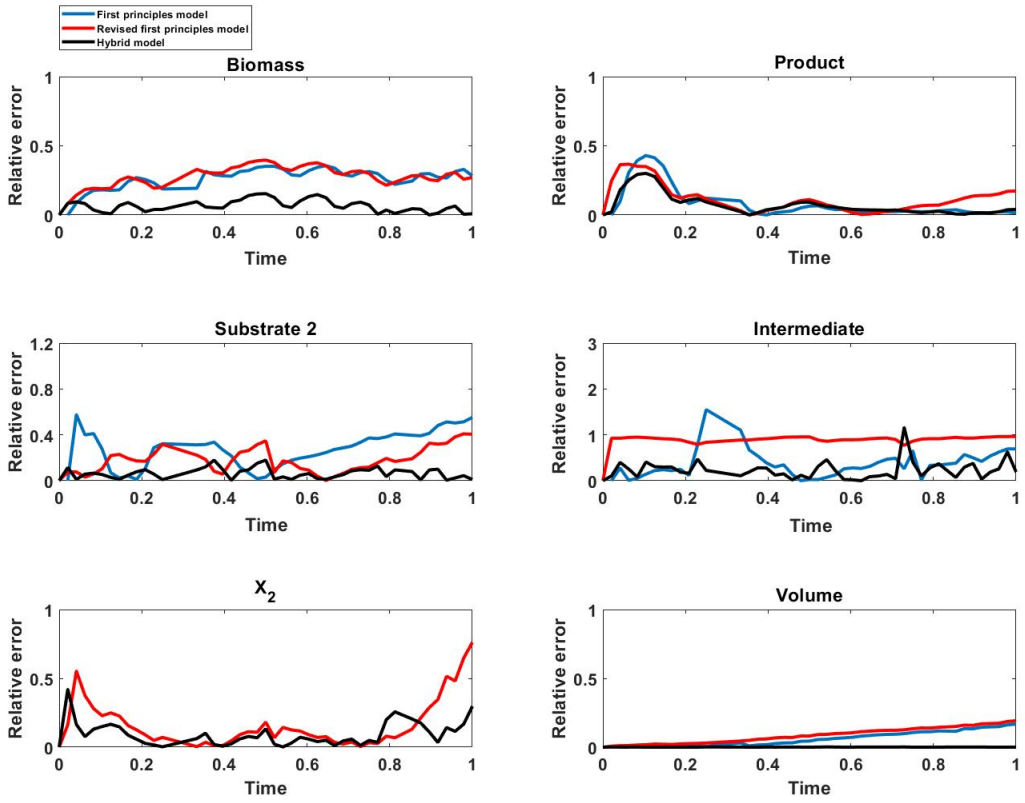


Figure 3.16: Relative errors between the models (i.e, the first-principles model, revised first-principles model, and hybrid model) and the training data obtained from the industry sponsor.

first, and 0.0448 (Product) and 0.1240 (Substrate 2) for the second. In conclusion, the hybrid model shows superior performance as it is equipped with a DNN that predicts time-sensitive parameters accurately.

	Biomass	Product	Substrate 2	Intermediate	X_2	Volume
First-principles	0.2162	0.0260	0.2886	0.2046	-	0.0792
Revised first-principles	0.2218	0.0672	0.1932	0.4851	0.1158	0.0993
Hybrid model	0.0590	0.0278	0.0707	0.1368	0.0719	5.877e-04

Table 3.7: RMSE values for all three models using training data

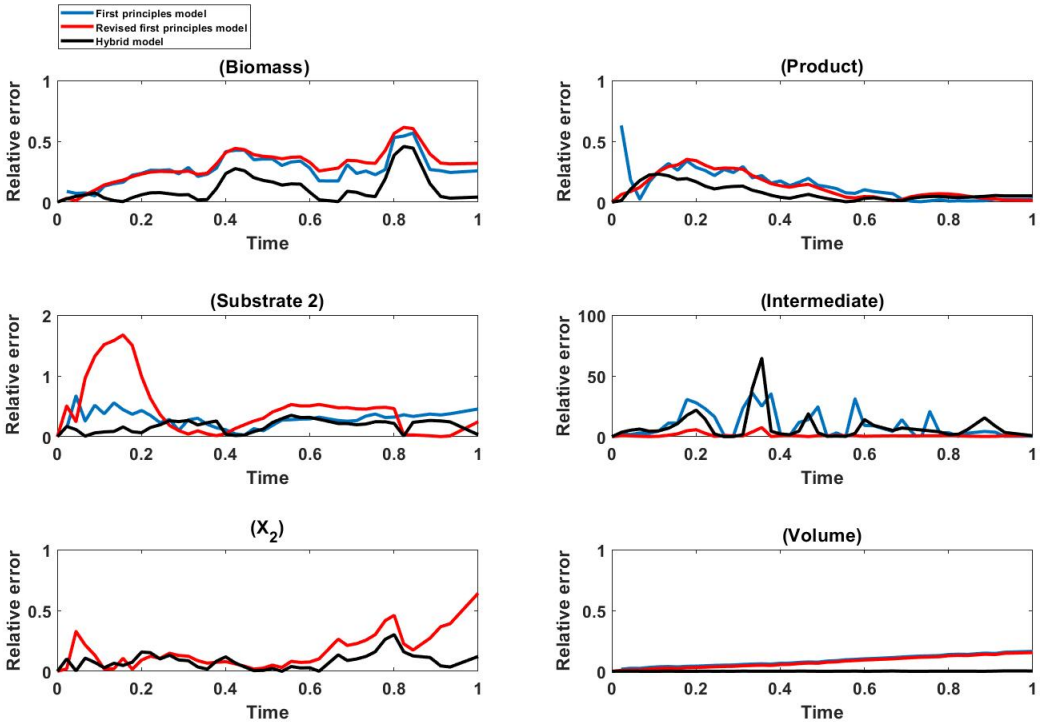


Figure 3.17: Relative errors between the models (i.e, the first-principles model, revised first-principles model, and hybrid model) and the validation data obtained from the industry sponsor.

	Biomass	Product	Substrate 2	Intermediate	X_2	Volume
First-principles	0.1972	0.0480	0.2968	2.536	-	0.0862
Revised first-principles	0.2189	0.0463	0.6030	0.3460	0.1170	0.0776
Hybrid model	0.1274	0.0278	0.1862	1.079	0.0639	9.792e-04

Table 3.8: RMSE values for all three models using validation data

3.3 Universal hybrid modeling of batch kinetics of aerobic carotenoid production using *Saccharomyces Cerevisiae*

In this work, we utilize a physics-informed machine learning method called Universal Ordinary Differential Equations (UDEs) to build a hybrid model for a bio-fermentation process. The concept of UDE is based on Neural ODEs (NODEs) method wherein the derivative of a function is modeled using a DNN which is trained using modern ODE solvers, and in UDEs only a part of the derivative function is modeled using a DNN. Now, in our hybrid modeling approach, the DNN represents the unknown terms in the ODEs of the first-principles model of the bio-fermentation. Essentially, this DNN captures the derivatives of unknown dynamics occurring within the process. This hybrid model is trained using the DifferentialEquations.jl package available in Julia programming language. The key difference between our proposed approach and the deep hybrid model proposed by Bangi and Kwon (2020) is that in the deep hybrid model, the structure of the DNN is fixed at the beginning of the training process, whereas the depth of the DNN in the UDE model is a parameter that is optimized during the training process. Considering the depth of the DNN as a parameter alleviates the problem of tuning the structure of the DNN during the model building process which is usually done using trial-and-error approach. Additionally, this difference is crucial as it means that by making the size of the DNN a parameter during the training process, we are essentially controlling the overall accuracy of the UDE model. This allows us to obtain any desired accuracy and very high accuracy will come at the cost of long training times.

In this work, we develop a UDE-based hybrid model (or Universal hybrid model) for the production of β-carotene using *Saccharomyces cerevisiae* strain mutant SM14 with glucose as the carbon source. Carotenes such as β-carotene improve human health as a precursor of vitamin A. There have been studies that show its positive impacts on human health, antioxidant properties, and protective properties against cancer [127, 128, 129, 130]. Recently, a first-principles model was developed for the batch production of β-carotene using *Saccharomyces cerevisiae* strain mutant SM14 with glucose as the main substrate [131]. This model is also capable of predicting

concentrations of biomass, glucose, ethanol and acetic acid, and hence, it is used in our work, as the first-principles model for the production process of β-carotene.

3.3.1 First-principles model for β-carotene production

Bio-fermentation processes utilize microbes to convert raw materials into useful products such as chemicals, food products, pharmaceuticals, etc. These microbes can be fungi, bacteria, mammalian cells, etc., and they are engineered in such a manner so as to produce greater yields of a desired product. These processes are widely used in the industry as they are sustainable due to their low-temperature and low-pressure operations, and no requirements for harsh chemicals [132]. A bio-fermentation process is usually modeled using first-principles such as conservation of mass, conservation of energy, kinetic laws, thermodynamic laws, transport laws etc., and these models are robust as they capture the essential dynamics that occur within it. The first-principles model from the work of Ordonez et. al. (2016) was adopted in this work. The growth rate model is as follows:

$$\mu = \mu_G + \mu_E + \mu_A \tag{3.56}$$

$$\mu_G = \frac{\mu_{max,G} \cdot \xi_E \cdot \xi_A \cdot G}{K_{S,G} + G + a_{G,E} \cdot E + a_{G,A} \cdot A} \tag{3.57}$$

$$\mu_E = \frac{\mu_{max,E} \cdot E}{K_{S,E} + E + a_{E,G} \cdot G + a_{E,A} \cdot A} \tag{3.58}$$

$$\mu_I = \frac{\mu_{max,A} \cdot A}{K_{S,A} + A + a_{A,G} \cdot G + a_{A,E} \cdot E} \tag{3.59}$$

where μ is the overall growth rate. μ_i, and $\mu_{max,i}$ refer to the growth rates, and the maximum specific growth rates associated with each component (i.e., glucose, ethanol, and acetic acid), respectively. $K_{S,i}$ is the half-velocity constant associated with each component, and $a_{i,j}$ is the inhibitory effect of component i on utilization of component j by the microorganisms. The variables $\xi_E \cdot \xi_A$ in Eq. (3.57) are added in order to account for any inhibition effect that ethanol or acetic acid may have on the glucose growth rate. Now, the cell growth is represented as follows:

$$\frac{dX}{dt} = \mu \cdot X \tag{3.60}$$

The glucose consumption rate is given by Eq. (3.61), where $Y_{X/G}$ is the biomass yield coefficient on glucose.

$$\frac{dG}{dt} = -\frac{\mu_G}{Y_{X/G}} \cdot X \tag{3.61}$$

The production and consumption of ethanol is governed by the following equation:

$$\frac{dE}{dt} = k_1 \cdot \mu_G \cdot X - \frac{\mu_E}{Y_{X/E}} \cdot X \tag{3.62}$$

Similarly, the production and consumption of acetic acid is governed by the following equation:

$$\frac{dA}{dt} = (k_2 \cdot \mu_G + k_3 \cdot \mu_E) \cdot X - \frac{\mu_A}{Y_{X/A}} \cdot X \tag{3.63}$$

Finally, β-carotene production is related to cell growth and biomass concentration as shown in Eq. (3.64).

$$\frac{dP}{dt} = (\alpha_1 \cdot \mu_G + \alpha_2 \cdot \mu_E + \alpha_3 \cdot \mu_A) \cdot X + \beta \cdot X \tag{3.64}$$

where α's represent the coefficients for growth-associated product formation with respect to each substrate, and β is the coefficient for non-growth-associated product formation.

3.3.2 Neural ODEs and UDEs

DNNs such as residual networks and recurrent neural networks learn the relationship between the input-output data by building a sequence of transformations to a hidden state h as follows:

$$h_{t+1} = h_t + f\left(h_t, \theta_t\right) \tag{3.65}$$

where $t \in \{0...T\}$ and $h_t \in \mathbb{R}^D$. The iterative updates as shown in Eq. (3.65) is very similar to the Euler discretization of a continuous transformation. This similarity is the fundamental idea behind

Neural ODEs and UDEs.

3.3.2.1 Neural ODEs

In a neural ODE, the continuous dynamics of hidden state h are parameterized using an ODE specified by a neural network as shown below:

$$\frac{dh(t)}{dt} = f\left(h(t), t, \theta\right) \tag{3.66}$$

where $h(0)$ and $h(T)$ are the values of the hidden state at the input layer and the output layer, respectively. Starting with $h(0)$, the value of $h(T)$ can be computed using a differential equation solver, which evaluates f with desired accuracy whenever necessary. The parameters of the neural network can be trained by performing reverse-mode differentiation through the ODE solver. The ODE solver is treated as a black box, and the gradients are calculated using the Adjoint Sensitivity method. In this method, the gradients are calculated by solving a second ODE backwards in time. This approach has low memory cost and explicitly controls accuracy [133].

3.3.2.2 UDEs

A UDE model is the extension of Neural ODEs to build a hybrid model by combining it with first-principles model. Specifically, part of the ODEs in the first-principles model contains an embedded DNN that learns the dynamics unaccounted in the first-principles model. An example of a UDE is shown below:

$$\frac{du}{dt} = g\left(u, t, U_\theta(u, t)\right) \tag{3.67}$$

where $\frac{du}{dt} = g\left(u, t\right)$ is the first-principles model with missing terms which are represented by U_θ, i.e., a DNN with θ as its parameters [134].

Now, training a UDE means minimizing a cost function $L(\theta)$ calculated with respect to the current solution $u_\theta(t)$, which is the solution of the UDE with respect to parameters θ. If the loss function $L(\theta)$ is minimized using methods such as gradient descent, or Adam, or L-BFGS (Limited memory-Broyden-Fletcher-Goldfarb-Shanno algorithm), then it requires the gradient of L with

respect to θ, i.e., $\frac{dL}{d\theta}$. By chain rule, this requires the calculation of $\frac{du}{d\theta}$. Therefore, training a UDE essentially boils down to obtaining the gradients of the solution of the UDE with respect to DNN parameters [134]. There exists many methods to calculate these gradients called Adjoints in Julia programming language. Also, the UDE model can be built by using the DifferentialEquations.jl package [135]. Specific details about the Julia packages used in this work are provided in the results section.

3.3.3 UDE model for β-carotene production

In our work, we generated data by solving Eqs. (3.56)-(3.64) with initial glucose concentration of 20 g/L and a fermentation time of 72 h. The parameter values used when solving the first-principles model are tabulated in Table 3.9. Now, in a UDE, the embedded DNN learns unknown dynamics that are not accounted for in the first-principles model. The structure of the DNN used in our work consists of 3 hidden layers with 20 neurons in each of them. The activation function in all the three layers is the linear function. The solver used to solve the ODEs is VCABM (an adaptive order adaptive time Adams Moulton method) in Julia [135]. Also, to calculate the gradients of the solution of the UDE model $u_\theta(t)$ with respect to the parameters $L(\theta)$, we used ForwardDiff-Sensitivity method, i.e., an implementation of discrete forward sensitivity analysis through Julia package ForwardDiff.jl [136].

Parameter	Value	Unit	Parameter	Value	Unit
$\mu_{max,G}$	0.2516	h^{-1}	$K_{S,G}$	0.4137	g Glucose L^{-1}
$\mu_{max,E}$	0.0218	h^{-1}	$K_{S,E}$	0.5618	g Ethanol L^{-1}
$\mu_{max,A}$	0.0182	h^{-1}	$K_{S,A}$	0.4506	g Acetic Acid L^{-1}
$a_{G,E}$	1.2964	–	$a_{G,A}$	1.0318	–
$a_{E,G}$	1.0636	–	$a_{E,A}$	1.0058	–
$a_{A,G}$	1.0000	–	$a_{A,E}$	1.0031	$\frac{gCell}{gEthanol}$
$Y_{X,G}$	0.1855	$\frac{gCell}{gGlucose}$	$Y_{X,E}$	0.3637	$\frac{gCell}{gEthanol}$
$Y_{X,A}$	1.0163	$\frac{gCell}{gAceticAcid}$	α_1	0.7545	$\frac{mgProduct}{gGlucose}$
α_2	13.9280	$\frac{mgProduct}{gEthanol}$	α_3	1.1089	$\frac{mgProduct}{gAceticAcid}$
β	0.2804	$\frac{mgProduct}{gCellh}$	k_1	1.7300	$\frac{gEthanol}{gCell}$
k_2	0.0936	$\frac{gAceticAcid}{gCell}$	k_3	0.2937	$\frac{gAceticAcid}{gCell}$

Table 3.9: Parameters used in the first-principles model.

Now, in order to train the parameters of the DNN, we solve an optimization problem using multiple shooting method. In multiple shooting method, the training data is split into multiple groups. The solver is then implemented on each individual group. If the end prediction of any group coincides with the initial prediction of the adjacent group, then the resultant solution is same as solving the optimization problem on the entire training data set. The objective function is the sum of the squared deviation of the hybrid model prediction with respect to the training data and a penalty term. This penalty is added to ensure that the overlapping parts of two consecutive groups coincide [137]. The objective function is shown below:

$$C(\theta) = \min_{\theta} \left(P_{pred} - P_{data}\right)^2 + Penalty \tag{3.68}$$

$$Penalty = P_f \cdot \sum_i |P_{pred,i} - P_{pred,i+1}| \tag{3.69}$$

where P_{pred} and P_{data} are the product values predicted by the UDE model and from the training data, respectively. P_f is a positive factor to ensure that their is continuity in the solution when performing the multiple shooting method. In our work, we used a value of 200 for P_f. $P_{pred,i}$ and $P_{pred,i+1}$ are the last product value predicted in group i and the first product value predicted in group $i+1$, respectively. Now, solving the optimization problem via multiple shooting method is performed in 2 steps. In the first step, the Adam solver is used to get to a minimum, and in the second step, we hone in on the minimum using the BFGS (Broyden-Fletcher-Goldfarb-Shanno algorithm) solver. We considered two hypothetical scenarios to show the capabilities of the UDE model.

3.3.3.1 Case 1

The assumption in this case is that the dynamics of the production of product (i.e., $\frac{dP}{dt}$) are entirely unknown. Now, in order to learn the product dynamics, we embed a DNN in the ODEs of the first-principles model, and train the resultant UDE using the training data. The resultant

product equation in the UDE is as follows:

$$\frac{dP}{dt} = U_\theta(X) \tag{3.70}$$

where U is the DNN, and θ is its set of parameters, i.e., weights and biases. The input to the DNN is the biomass concentration and its output is the approximated product dynamics. Now, the UDE model including Eq. (3.70) is trained, and Figure 3.18 shows the training progress via the objective function value with respect to the iteration number.

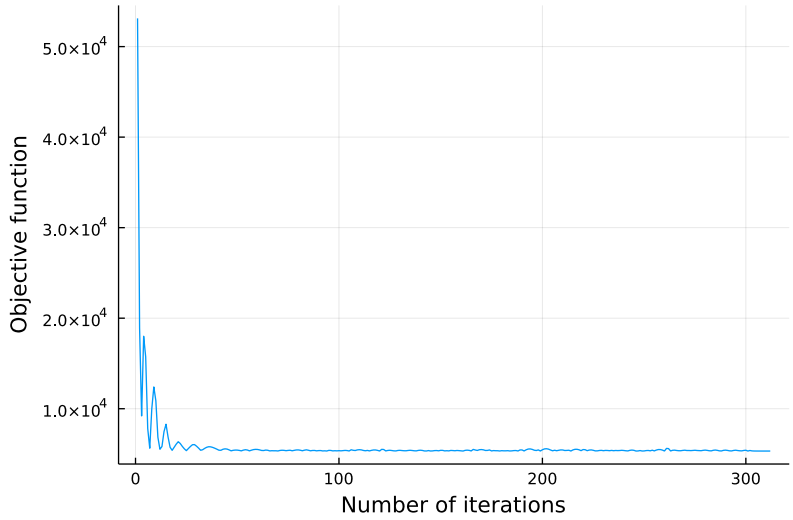


Figure 3.18: Training progress with iteration for Case 1.

In Figure 3.18, we observe that the objective function values are high initially. This is because the parameters of the DNN are randomly initialized and its predictions of product dynamics are highly inaccurate. But as training progresses, the parameter values are optimized and the objective function reaches a minimum. Once training ends, we utilize the UDE model to make predictions for all 5 concentrations (i.e., biomass, glucose, ethanol, acetic acid, and product). The UDE model's predictions are compared against training data, and this comparison is shown in Figure 3.19. From Figure 3.19, we observe that the UDE model's predictions are accurate.

In order to quantify the performance of the UDE model, we calculate relative error as shown

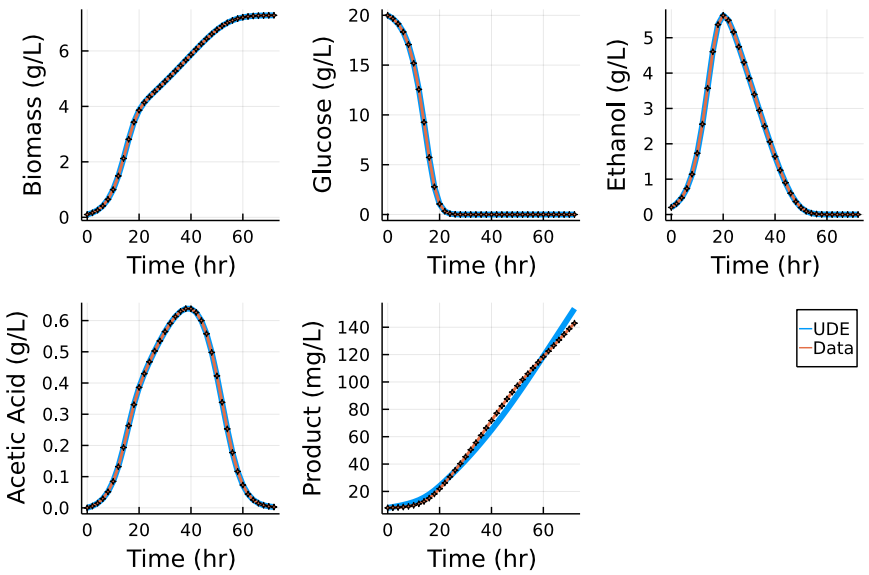


Figure 3.19: Comparison of UDE model predictions versus data for Case1.

below:

$$Error = 1 - \left| \frac{P_{pred}}{P_{data}} \right| \tag{3.71}$$

The error variation with time is plotted in Figure 3.20. In this figure, we observe that the relative error percentage was able to go as low as less than 10% with time.

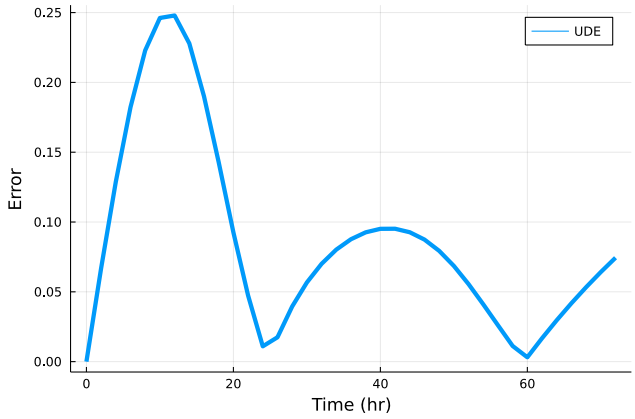


Figure 3.20: Relative error variation for Case 1.

3.3.3.2 Case 2

The assumption in this case is that the dynamics of the production of product are partially unknown (i.e., $\beta \cdot X$ is assumed to be missing in the first-principles model). In order to learn the

unknown product dynamics, we embed a DNN in the ODEs of the first-principles model, and train the resultant UDE using the training data. The resultant product equation in the UDE is as follows:

$$\frac{dP}{dt} = (\alpha_1 \cdot \mu_G + \alpha_2 \cdot \mu_E + \alpha_3 \cdot \mu_A) \cdot X + U_\theta(X) \tag{3.72}$$

where U is the DNN, and θ is its set of parameters, i.e., weights and biases. The input to the DNN is the biomass concentration and its output is the approximated unknown product dynamics. Now, the UDE model including Eq. (3.72) is trained and Figure 3.21 shows the training progress.

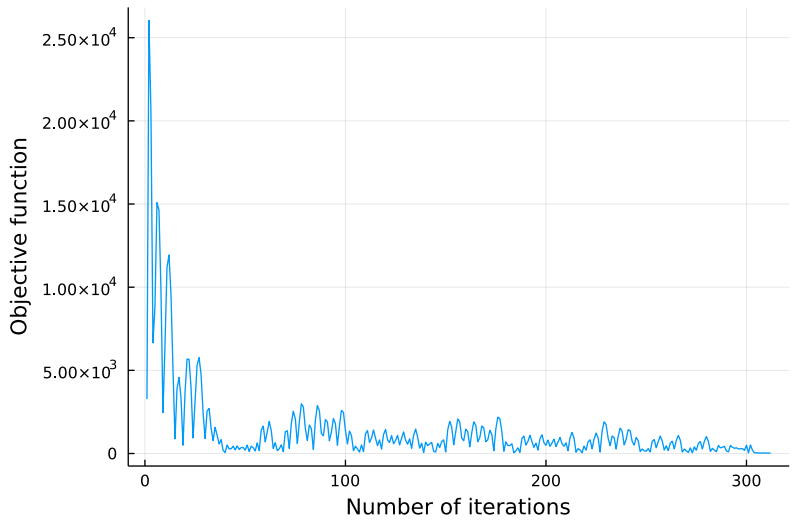


Figure 3.21: Training progress with iteration for Case 2.

From Figure 3.21, we observe that the objective function values are high initially but as training progresses, the performance of the UDE model improves until a minimum is reached. Once training ends, we utilize the UDE model to make predictions and compare it against the first-principles model (with partially known dynamics) and the training data. This comparison is shown in Figure 3.22. From Figure 3.22, we observe that the UDE model's predictions are very accurate but the first-principles model's performance is poor because of the unknown dynamics.

In order to quantify the performance of the UDE model and the first-principles model (with partially known dynamics), we calculate their relative errors as shown in Figure 3.23.

In Figure 3.23, we observe that the relative error for the UDE model is near 0%, whereas,

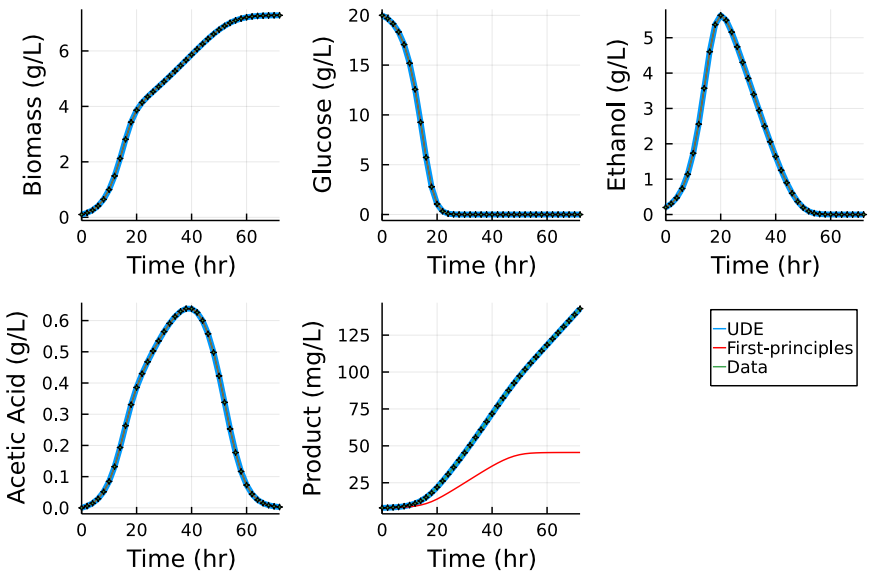


Figure 3.22: Comparison of UDE model predictions versus data for Case2.

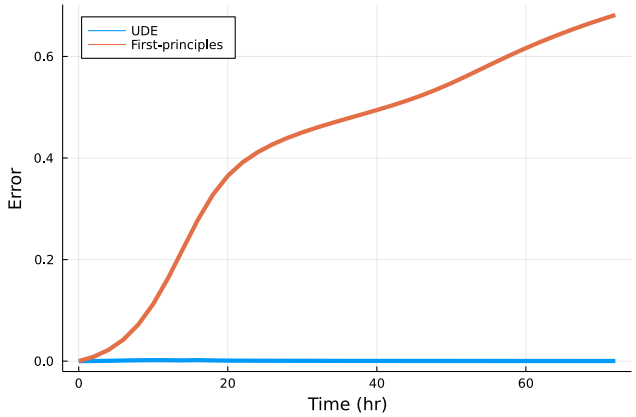


Figure 3.23: Relative error variation for Case 2.

for the first-principles model it goes beyond 50%. Also, the performance of the UDE model in this case is far superior compared to Case 1, and this can be observed by comparing Figs. 3.22 and 3.23. The performance of the UDE model in these two figures highlights the importance of incorporating any known knowledge about the process in the UDE model structure.

3.4 Physics-informed neural networks for hybrid modeling of lab-scale batch fermentation for β-carotene production using *Saccharomyces Cerevisiae*[*]

In the previous section, we built a UDE model to approximate the kinetics involved in the production of β-carotene using *Saccharomyces Cerevisiae*. In order to show the effectiveness of the UDE method, we made few theoretical assumptions about the kinetic model as described in Eqs. (3.56)-(3.64), and built the UDE model. But in this work, we use experimental data to test the effectiveness of the UDE method to build a hybrid model [138].

Carotenoids are a diverse group of yellow-orange pigments which have been extensively used in food pigmentation and as constituents in dietary and vitamin supplements [139, 140]. Specifically, β-carotene, which is a precursor for Vitamin A, has shown to have a positive impact on human health, antioxidant properties and protective properties against cancer [129, 130, 127, 128]. Currently, some of the Carotenoids are produced synthetically using chemical technology but the byproducts in such chemical processes have undesirable side effects when consumed. For this reason, the use of microbial sources for the production of Carotenoids has received lots of attention [141, 140, 142, 143, 144].

An accurate model which can describe biomass growth, substrate consumption, and β-carotene formation is critical for process optimization and control purposes. Usually, a first-principles model is developed to describe the physical and bio-chemical phenomena occurring in the batch fermentation process, which includes fundamental laws such as conservation of mass and energy, kinetic laws, thermodynamic laws, and transport laws. A first-principles model is robust, but it cannot account for all the complex interactions within the process, thereby limiting its accuracy. An alternative to first-principles modeling is data-driven modeling. Recently, there has been a lot of

[*]Reprinted with permission from "Physics-informed neural networks for hybrid modeling of lab-scale batch fermentation for β-carotene production using *Saccharomyces Cerevisiae*" by Bangi, M. S. F., Kao, K., and Kwon, J. S. 2022. Chem. Eng. Res. Des., 179, 415-423, Copyright 2022 Elsevier.

interest in data-driven modeling [145, 146, 147, 148] and control [149, 150] in the field of chemical engineering as the amount of data collected, stored, and utilized has increased tremendously. Data-driven models are computationally inexpensive to solve, but they show poor extrapolation over a range of inputs and operating conditions [97, 17]. To overcome the above-mentioned limitations, hybrid modeling is utilized which combines a first-principles model with a data-driven model. Hybrid models show superior accuracy and better extrapolation properties than first-principles models and data-driven models, respectively [151]. The idea of hybrid models began with combining artificial neural network models with first-principles models [21]. Since then, hybrid modeling has been applied in various chemical and biochemical engineering applications. For example, in modeling of bacteria cultivations [52], crystallization [34, 35], fungi cultivations [50], insect cell cultivations [55], mammalian cell cultivations [54], yeast fermentations [48], intracellular signal pathway [152, 153], chemical reactor [28], mechanical reactors [43], distillation columns [39, 40], drying processes [41], metallurgic processes [38], milling [45], polymerization processes [33], thermal devices [42], etc. For more information on hybrid modeling, one can view [58, 154], which are excellent review papers.

Recall, hybrid modeling started in 1992 from the use of artificial neural networks along with first-principles knowledge [21]. Since then, the field of neural networks has evolved from artificial neural networks with a single hidden layer to deep neural networks (DNNs) with multiple hidden layers. These DNNs require an exponentially fewer number of neurons than artificial neural networks to approximate specific functions [108, 110, 109]. Recently, [151] developed a hybrid model using a DNN for hydraulic fracturing which consists of a discrete number of hidden layers between input and output layers. In a DNN with a discrete number of hidden layers, each layer adds a small error to its output which traverses through the hidden layers. Generally, adding more layers to the DNN structure will reduce the overall modeling error. But with the number of layers increasing, the accuracy of the DNN saturates first and then begins to degrade [155]. Adding new connections in the DNN to form Residual network (ResNet) will solve this degradation problem [155]. Mathematically, these ResNets resemble the solution obtained using Euler's

method for ordinary differential equations (ODEs). This similarity led to the development of a new class of networks called Neural Ordinary Differential Equations (Neural ODEs) in which instead of specifying a discrete sequence of hidden layers between the input and output domains, the progression of the hidden states through the hidden layers becomes continuous [133]. This continuous equation is solved using a black-box differential equation solver to obtain the output of the Neural ODEs. These continuous-depth networks adapt their evaluation strategy to each input, explicitly trade computational speed for accuracy, and have constant memory cost. These Neural ODEs can be combined with an existing first-principles model to build a physics-informed neural network model called Universal Differential Equations (UDEs) [134].

In this work, the UDE framework is utilized to build a hybrid model for batch production of β-carotene using *Saccharomyces cerevisiae*. The key difference between the UDE-based hybrid model approach and the deep hybrid model proposed by Bangi and Kwon (2020) is that the depth of the neural network in the UDE-based hybrid model is a parameter during the model training process. On the other hand, in Bangi and Kwon (2020), the structure of the neural network is fixed at the beginning of the training process, and is optimized using trial-and-error approach. Considering the depth of the neural network as a parameter alleviates the challenge of tuning it during the model building stage which is usually done using a trial-and-error approach. Additionally, this difference is crucial as it means that by making the depth of the neural network a parameter during the training process, the overall accuracy of the UDE-based hybrid model can be controlled. This allows us to obtain any desired accuracy, but very high accuracy will come at the cost of long training times. In order to reduce the training time, prior knowledge about the process can be incorporated during the UDE-based hybrid model training. Specifically, careful selection of the input features to the Neural ODE in the UDE-based hybrid model can ensure faster convergence of its parameters. Therefore, the novelty of the proposed work can be summarized as follows: a) building a hybrid model which has superior accuracy compared to the existing kinetic model for batch production of β-carotene using *Saccharomyces cerevisiae*, b) utilizing UDE approach and experimental data to train and validate a hybrid model for a complex batch fermentation process, and c) incorporation

of prior process knowledge to ensure convergence of UDE-based hybrid model parameters.

3.4.1 Microorganism and culture media

As mentioned previously, *Saccharomyces cerevisiae* strain mutant SM14 [156] was used in this work. In the engineered yeast strain, which is the ancestor for SM14, the carotenogenic pathway genes crtYB/crtI/crtE were introduced into yeast S288c strain. The engineered β-carotene producer was subjected to an adaptive evolution experiment. Strain SM14 was a β-carotene hyperproducer that resulted from the evolution experiment. These yeast strains, engineered to produce beta-carotene, were streaked out for single colonies from -80 °C cryostorage onto YPD plates. Fresh plates were streaked out every 3 weeks. Experiments were performed to obtain the optimal initial glucose concentration of 20 g/L. Each colony was inoculated in a flask containing 50 ml YNB (Yeast Nitrogen Base) supplemented with 20 g/L glucose and grown overnight at 200 rpm and 30 °C for use as seed culture for bioreactor runs.

3.4.2 Bioreactor cultivation results

Figure 3.24 shows the concentrations of biomass, β-carotene, ethanol and acetic acid production and subsequent consumption, and glucose consumption of *Saccharomyces Cerevisiae* in a stirred-tank bioreactor with initial glucose concentration of 20 g/L. In the first 24 h, yeast exhibited an exponential growth period until the glucose was completely utilized, and the ethanol concentration had reached a maximum concentration value of 5.42 g/L. After glucose was completely utilized, ethanol was utilized as a carbon source, resulting in the decline of ethanol concentration and a maximum in the acetic acid concentration of 1.19 g/L at 50 h. Thereafter, acetic acid was consumed. β-carotene production increased during the growth phase and continued throughout the cultivation period. The β-carotene production reached nearly 120 mg/L at the end of a 72 h period.

3.4.3 UDE model for lab-scale β-carotene production

The kinetic model as described in Eqs. (3.56)-(3.64) is used as the first-principles model. In this work, we utilized data as shown in Figure 3.24 with initial glucose concentration of 20 g/L and

a fermentation time of 72 h. The parameter values used when solving the first-principles model are tabulated in Table 3.9. Now, in a UDE, the embedded DNN learns unknown dynamics that

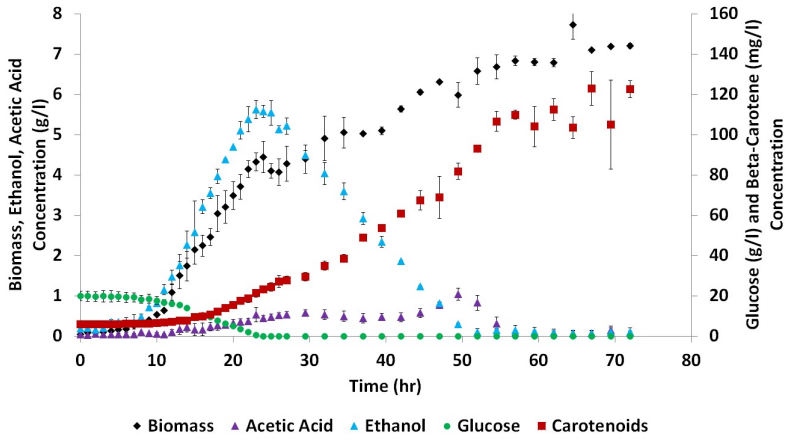


Figure 3.24: Biomass, glucose consumption, ethanol and acetic acid concentration and carotenoids production in batch cultures of *Saccharomyces Cerevisiae* with 20 g/L initial glucose

are not accounted for in the first-principles model, and we utilized 3 DNNs for building the UDE model. Each DNN was utilized to learn and predict the unknown dynamics in the biomass, acetic acid, and product equations, respectively. The inputs to the DNNs are the biomass and ethanol concentrations. The structure of each DNN used in our work consists of 2 hidden layers with 5 neurons in each of them. The activation function in all the three layers is the hyperbolic tangent (tanh) function. The solver used to solve the ODEs is VCABM (an adaptive order adaptive time Adams Moulton method) in Julia [135]. Also, to calculate the gradients of the solution of the UDE model $u_\theta(t)$ with respect to the parameters $L(\theta)$, we used ForwardDiffSensitivity method, i.e., an implementation of discrete forward sensitivity analysis through Julia package ForwardDiff.jl [136].

Now, in order to train the parameters of the DNN, we solve an optimization problem with an objective function as shown in Eq. (3.73):

$$C\left(\theta\right) = \min_{\theta} \left(x_{pred} - x_{data}\right)^2 \tag{3.73}$$

where x_{pred} and x_{data} are the concentration values of biomass, acetic acid, and product predicted by the UDE model and from the training data, respectively. Now, solving the optimization problem is performed in 2 steps. In the first step, the Adam solver is used to get to a minimum, and in the

second step, we hone in on the minimum using the BFGS (Broyden-Fletcher-Goldfarb-Shanno algorithm) solver.

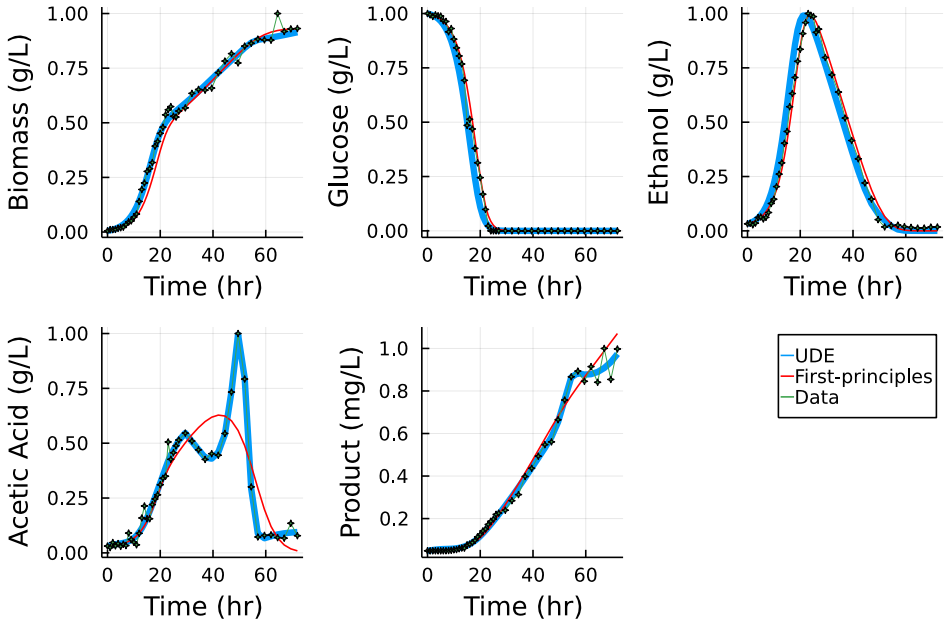


Figure 3.25: Comparison of UDE model predictions after training versus predictions from first-principles model for an initial glucose concentration of 20 g/l.

The training results are shown in Figure 3.25. The biomass and product concentrations have improved when compared to the predictions from the first-principles model. The UDE model outperforms first-principles model vastly when it comes to the prediction of acetic acid. It is able to accurately capture the dynamic behavior in acetic acid concentration especially in the period of 40 to 60 h. This shows the effectiveness of utilizing the UDE method to build a hybrid model to improve the prediction accuracy.

3.4.4 Testing UDE model with different initial concentrations of glucose

The UDE model is tested with other experimental data-set where the initial concentrations of glucose is 22.36 g/. The UDE model predictions are compared against the predictions from the first-principles model, and this comparison is shown in Figure 3.26.

From Figure 3.26, it can be observed that the UDE model is able to outperform the first-principles model even when the initial concentration of glucose is different from the training case.

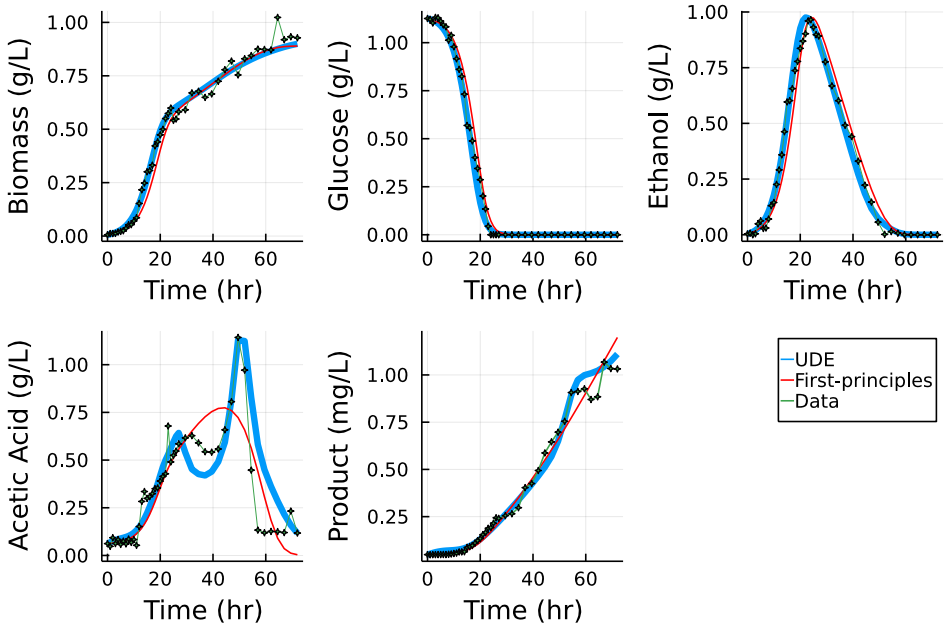


Figure 3.26: Comparison of UDE model predictions versus predictions from first-principles model for an initial glucose concentration of 22.36 g/l.

In order to quantify these comparisons, we utilize the Mean Squared Error (MSE) metric as shown below:

$$MSE = \frac{\sum (x_{pred} - x_{data})^2}{N} \tag{3.74}$$

The MSE values for both the training and testing cases are calculated using Eq. (3.74), and are tabulated in Table 3.10.

MSE	UDE Training	First-principles Training	UDE Testing	First-principles Testing
Biomass	0.0253	0.0492	0.0345	0.0524
Acetic Acid	0.0251	0.1102	0.1317	0.1319
Product	0.0236	0.0411	0.0423	0.0486

Table 3.10: MSE values for training and testing data-sets.

From Table 3.10, it can be observed that the MSE values for the UDE model are smaller than the first-principles model. This is because the DNNs in the UDE model are trained to capture the hidden dynamics of the process which are unaccounted in the first-principles model. This shows the advantage of building a UDE model because of its superior accuracy.

4.1 Stabilization with guarantees on domain of applicability for hybrid model-based predictive control

Historically, process modeling began with the use of first-principles such as conservation of mass and energy balances, thermodynamic laws, kinetic laws, etc. These first-principle models have wide domain of applicability (DA) but can have poor accuracy due to some unexplained physical/chemical phenomena in the process. On the other hand, data-based models such as deep neural networks (DNNs) can be developed using plant measurements or simulation data that are computationally inexpensive, highly accurate within the DA but have a narrow DA. Hybrid models combine first-principles models with data-based models resulting in superior accuracy compared to first-principles models and better extrapolability compared to data-based models. Recently, a deep hybrid model was developed which combines first-principles model with a DNN for hydraulic process.

The gain in extrapolability in deep hybrid model when compared to a purely data-based DNN model is useful in practical applications. Specifically, when designing a model-based predictive controller wherein a wide range of DA of the model allows for a wider search of the input space in order to obtain optimal control actions. Also, a narrow DA would restrict the input space and would lead to sub-optimal control performance. Although, the deep hybrid model has a wider DA compared to a DNN model it is still finite and is influenced by the DA of the DNN within it. Therefore, in this work, we propose to design a deep hybrid model-based controller that obtains the optimal control policy within the DA of the deep hybrid model while guaranteeing the stability of the original system.

In the last couple of decades, model predictive control (MPC) has been utilized widely to control multi-variable processes with linear/nonlinear models and constraints. Specifically, Lyapunov-

based MPC (LMPC) has been developed which uses a Lyapunov-based control law to ensure feasibility and stabilizability within a well defined stability region [157, 158]. In order to incorporate the DA of the deep hybrid model within the LMPC controller, we propose to develop and incorporate within the MPC design a Control Barrier Function (CBF) that has prominence in the field of safety [159, 160, 161, 162, 163, 164, 165]. The developed CBF ensures that the controller stays within the DA of the deep hybrid model which is obtained by implementing the k-nearest neighbors (knn) technique on the training data used for training the DNN in the deep hybrid model. Subsequently, in order to ensure simultaneous stability and guarantees on DA by the MPC, we develop a Control Lyapunov-Barrier Function (CLBF) which combines both the Control Lyapunov Function (CLF) and the CBF. This CLBF is incorporated in the form of various constrains in the MPC formulation in order to ensure closed-loop stable performance within the DA of the deep hybrid model. This work is divided into two parts. In the first part, we provide stability analysis and theoretical guarantees with respect to the DA of the deep hybrid model for a CLBF-MPC ontroller. In the second part, we apply our proposed method on a chemical process example i.e., a continuous stirred tank reactor (CSTR).

4.1.1 Stability analysis and DA guarantees

Consider a continuous-time nonlinear system described by the following state-space form:

$$\dot{x} = F(x, u) = f(x) + g(x)u \tag{4.1}$$

Let its existing first principles model be described by the following equation:

$$\dot{x} = \tilde{F}(x, u) = \tilde{f}(x) + \tilde{g}(x)u \tag{4.2}$$

Assuming that the accuracy of the first-principles model is limited, we utilize a deep neural network $D(x)$ to build a hybrid model in order to improve its accuracy. The following equation represents

the hybrid model:

$$\dot{x}_h = F_h(x_h, u) = \tilde{F}(x_h, u) + D(x_h, \theta) \tag{4.3}$$

Here $\tilde{F}$ is known to us and D represents the unknown dynamics of the system. Let θ be the parameters of D, which can be optimized by minimizing the following objective function:

$$J_T(\theta) = \frac{1}{T}\int_0^T ||\dot{x}(t) - \tilde{F}(x, u) - D(x; \theta)||^2 dt + R(\theta) \tag{4.4}$$

Here $R(\theta)$ is the regularization term which promotes sparsity in the parameter values. In an ideal scenario, an optimal deep neural network D^* with parameters θ^* will fully represent the unknown dynamics of the system such that:

$$\dot{x} = F(x, u) = \tilde{F}(x, u) + D^*(x, \theta^*) \tag{4.5}$$

Based on the universal approximation theorem for neural networks which states that given a sufficient number of neurons, a DNN is able to approximate any dynamic nonlinear system on compact subsets of the state-space for finite time. Extending this principle to the hybrid model system, such that, for any $\delta_o > 0$ the following equation holds true:

$$||D^*(x, \theta^*) - D(x, \theta)|| \leq \delta_o \tag{4.6}$$

Using Eq. (4.6), Eq. (4.5) can be rewritten as:

$$\dot{x} = F(x, u) = F_h(x, u) + e(t) \tag{4.7}$$

such that

$$\sup_{t \in [0,T]} ||e(t)|| \leq \delta_o \tag{4.8}$$

Integrating Eqs. 4.3 and 4.7, for any $t \in [0, T]$, subtracting, and taking norms we obtain

$$||x(t) - x_h(t)|| \leq ||x(0) - x_h(0)|| + \int_0^t ||F_h(x, u) - F_h(x_h, u)||ds + \int_0^t ||e(s)||ds \tag{4.9}$$

Using Eq. (4.8), and assuming F_h is L-Lipschitz, we obtain

$$||x(t) - x_h(t)|| \leq ||x(0) - x_h(0)|| + L \int_0^t ||x(s) - x_h(s)||ds + \delta_o T \tag{4.10}$$

Using the integral form of the Gronwall Lemma, we obtain

$$||x(t) - x_h(t)|| \leq [||x(0) - x_h(0)|| + \delta_o T]\, e^{Lt} \tag{4.11}$$

Assuming the initial state in the hybrid model is identical to that of the actual system, we obtain

$$||x(t) - x_h(t)|| \leq \delta_o T e^{Lt} \tag{4.12}$$

4.1.1.1 Notation

The Euclidean norm of a vector is represented as $|\cdot|$, and the weighted Euclidean norm is represented as $|\cdot|_Q$ where Q is a positive definite matrix. x^T represents the Transpose of x. $\mathbb{R}_+$ represents the set $[0, \infty)$. The null set is represented using ϕ. $L_f V(x)$ represents the standard Lie derivative i.e., $L_f V(x) := \frac{\partial V(x)}{\partial x} f(x)$. A scalar continuous function $V : \mathbb{R}^n \to \mathbb{R}$ is proper if the set $x \in \mathbb{R}^n | V(x) \leq k$ is compact $\forall k \in \mathbb{R}$ which is equivalent to V being radially unbounded. Assume positive real numbers β and ϵ such that $\mathcal{B}_\beta(\epsilon) := \{x \in \mathbb{R}^n |\ |x - \epsilon| < \beta\}$ is an open ball around ϵ with a radius of β. Subtraction of sets is denoted using "\", i.e., $A \backslash B := \{x \in \mathbb{R}^n | x \in A, x \notin B\}$. A function f is of class $\mathcal{C}^1$ if it is continuously differentiable. Given a set $\mathcal{D}$, the boundary and the closure of $\mathcal{D}$ are denoted by ∂D and $\bar{\mathcal{D}}$, respectively. A continuous function $\alpha : [0, a) \to [0, \infty)$ is said to belong to class $\mathcal{K}$ if it is strictly increasing and is 0 only when evaluated at 0.

4.1.1.2 Lyapunov-based control for system stability

We assume that there exists a Control Lyapunov function (CLF) V for system in Eq. (4.1) such that V is proper, positive definite, and satisfies the small control property i.e., $\forall \epsilon > 0, \exists \delta > 0, s.t. \forall x \in \mathcal{B}_\delta(0)$, there exists u that satisfies $|u| < \epsilon$ and $L_f V(x) + L_g V(x) u < 0$, and also the following condition:

$$L_f V(x) < 0, \ \forall x \in \{z \in \mathbb{R}^n \backslash \{0\} | L_g V(z) = 0\} \tag{4.13}$$

This assumption implies that there exists a stabilizing control law $\Phi(x) \in U$ for the system of Eq. (4.1) that renders the origin of the system asymptotically stable $\forall x$ in a neighborhood around the origin such that Eq. (4.13) holds for $u = \Phi(x)$. An example of such a controller is given by the Sontag's formula [166]:

$$k_i(x) = \begin{cases} -\frac{p + \sqrt{p^2 + \gamma |q|^4}}{|q|^2} q & \text{if } q \neq 0 \\ 0, & \text{if } q = 0 \end{cases} \tag{4.14a}$$

$$\Phi_{S,i}(x) = \begin{cases} u_{min} & \text{if } k_i(x) < u_{min} \\ k_i(x) & \text{if } u_{min} \leq k_i(x) \leq u_{max} \\ u_{max} & \text{if } k_i(x) > u_{max} \end{cases} \tag{4.14b}$$

where p denotes $L_f V(x)$, q denotes $L_g V(x)$, and $\gamma > 0$. Also, the constraints on the control action is defined by $u \in U := \{u_{min} \leq u \leq u_{max}\} \subset \mathbb{R}^m$, where u_{min} and u_{max} are the minimum and maximum values of inputs allowed, respectively. Based on the Lyapunov control law $\Phi(x)$, a region ϕ_u can be found under constrained inputs such that the time-derivative of the CLF is negative i.e., $\phi_u = \{x \in \mathbb{R}^n | \dot{V} < 0, u = \Phi(x) \in U\}$. Also, a level set Ω_b inside ϕ_u is defined as $\Omega_b = \{x \in \phi_u | V(x) \leq b, b > 0\}$. Since Ω_b is a forward invariant subset of ϕ_u, it is guaranteed that for any initial state $x_0 \in \Omega_b$, $\forall t \geq t_0$, $x(t)$ of system remains in Ω_b under control law of Eq. (4.14).

4.1.1.3 Hybrid model

A hybrid model (first-principles combined with DNN) as shown in Eq. (4.3) is developed with the following form:

$$\dot{x}_h = F_h(x_h, u) = f_h(x_h) + g_h(x_h)u \tag{4.15}$$

such that

$$f_h(x_h) = \tilde{f}(x_h) + D(x_h, \theta) \tag{4.16}$$

$$g_h(x_h) = \tilde{g}(x_h) \tag{4.17}$$

where $x_h \in \mathbb{R}^n$ is the hybrid model state vector and $u \in \mathbb{R}^m$ is the manipulated input vector. Throughout the manuscript, we use x to represent the state of the system in Eq. (4.1) and x_h is the state obtained using the hybrid model of Eq. (4.15). Also, to ensure that the hybrid model of Eq. (4.15) has the same steady-state as the original system of Eq. (4.1), the generalization error needs to be bounded within the operating region as shown in Eq. (4.12). Additionally, we assume that there exists a CLF V and a stabilizing controller $u = \Phi_h(x) \in U$ that renders the origin of the hybrid model of Eq. (4.15) asymptotically stable.

Now, by the universal approximation theorem, DNNs are able to model any continuous nonlinear functions on compact subsets of the state-space $\mathbb{R}^n$ with sufficient number of neurons. Furthermore, $D(x_h, \theta)$ is the output of a series of nonlinear transformations involving inputs to the DNN, weights and biases. We choose activation functions that are Lipschitz continuous in the compact subset within which the DNN training data is collected, such as $tanh$. All hidden layers and output layer of the DNN model use $tanh$ as the activation function, therefore making the DNN also Lipschitz continuous. Now, assuming $\tilde{F}$ to be continuous, this leads to the conclusion that F_h is also continuous.

4.1.1.4 Characterization of Domain of Applicability (DA)

As mentioned previously, we utilize a training set $\mathcal{S}$ to build the hybrid model as shown in Eq. (4.15). Ideally, the DA of the hybrid model would span the entire set $\mathcal{S}$. But in order to build a

Control Barrier function (CBF), we need two sets of points. One set for defining the DA region and the other for defining the 'not DA' region. Therefore, we divide the training set $\mathcal{S}$ into two sets $\mathcal{D}$ and $\mathcal{U}$ such that $\mathcal{D}$ is considered as DA, and $\mathcal{U}$ is outside $\mathcal{D}$. A Control Lyapunov Barrier function (CLBF)-based predictive controller based on hybrid model will be developed to ensure the closed-loop stability while avoiding $\mathcal{U}$.

Now, $\mathcal{D} \subset \mathbb{R}^n$ is the DA within which the performance of the hybrid model is satisfactory, and $\mathcal{U}$ is the region outside the DA of hybrid model such that $\mathcal{U} \cap \mathcal{D} = \emptyset$ and $\{0\} \subset \mathcal{D}$. The objective is to design a controller such that the system in Eq. (4.1) is stable and remains within $\mathcal{D}$ in the following sense:

Definition 1. *Consider the system of Eq. (4.1). If there exists a control law $u = \Phi(x) \in U$ such that for any initial state $x(t_0) = x_0 \in \mathcal{D}$, $x(t)$ remains inside $\mathcal{D}$, $\forall t \geq 0$ and the origin of the closed-loop system of Eq. (4.1) can be rendered asymptotically stable, we say that the control law $u = \Phi(x)$ maintains the process state within DA at all times.*

4.1.1.5 Control Barrier function (CBF)

Following the definition on stability and DA guarantees, the definition of a CBF is as follows:

Definition 2. *Given a set of points outside the DA in state space $\mathcal{U}$, a $\mathcal{C}^1$ function $B(x) : \mathbb{R}^n \to \mathbb{R}$ is a CBF if it satisfies the following properties:*

$$B(x) > 0 \; \forall x \in \mathcal{U}, \tag{4.18a}$$

$$L_f B(x) \leq 0 \; \forall x \in \{z \in \mathbb{R}^n \backslash \mathcal{U} | L_g B(z) = 0\}, \tag{4.18b}$$

$$\mathcal{X}_B := \{x \in \mathbb{R}^n | B(x) \leq 0\} \neq \emptyset \tag{4.18c}$$

4.1.1.6 Stabilization and DA guarantees via CLBF

We utilize the method proposed in [160], where it has been shown that if a CLBF $W_c(x)$ exists for the system of the form Eq. (4.1), then there exists a controller of the form Eq. (4.14) with $W_c(x)$ replacing V that guarantees both safety and stability. But in our work, we design a CLBF

controller that provides stability and gives guarantees on DA instead of safety.

Definition 3. *Given a set of points $\mathcal{U}$ in state space i.e., 'not DA' of the hybrid model, a $\mathcal{C}^1$ function $W_c(x) : \mathbb{R}^n \to \mathbb{R}$, which is proper and lower-bounded, is a CLBF if it has a minimum at the origin and satisfies the following properties:*

$$W_c(x) > \rho_c,\ \forall x \in \mathcal{U} \subset \phi_{uc} \tag{4.19a}$$

$$\left|\frac{\partial W_c(x)}{\partial x}\right| \leq r(|x|) \tag{4.19b}$$

$$L_{f_h} W_c(x) < 0,\ \forall x \in \{z \in \phi_{uc} \backslash (\mathcal{U} \cup \{0\} \cup \mathcal{X}_e) | L_{g_h} W_c(z) = 0\}, \tag{4.19c}$$

$$\mathcal{U}_{\rho_c} := \{x \in \phi_{uc} | W_c(x) \leq \rho_c\} \neq \emptyset \tag{4.19d}$$

$$\overline{\phi_{uc} \backslash (\mathcal{U} \cup \mathcal{U}_{\rho_c})} \cap \overline{\mathcal{U}} = \emptyset \tag{4.19e}$$

where $\rho_c \in \mathbb{R}$, and $\mathcal{X}_e := \{x \in \phi_{uc} \backslash (\mathcal{U} \cup \{0\}) | \partial W_c(x)/\partial x = 0\}$ is a set of states where $L_{f_h} W_c(x) = 0$ (for $x \neq 0$) due to $\partial W_c(x)/\partial x = 0$.

Under a stabilizing control law $\Phi_h(x)$ i.e., Eq. (4.14) with W_c replacing V, ϕ_{uc} is defined to be the union of the origin, $\mathcal{X}_e$ and the set where the time-derivative of $W_c(x)$ is negative with constrained input: $\phi_{uc} = \{x \in \mathbb{R}^n | \dot{W}_c(x(t), \Phi_h(x)) = L_{f_h} W_c + L_{g_h} W_c u < -\alpha |W_c(x) - W_c(0)|, u = \Phi_h(x) \in U\} \cup \{0\} \cup \mathcal{X}_e$, and α is a positive real number used to characterize the set ϕ_{uc}. Also, we define the set of initial conditions by $\mathcal{X}_{W_c} := \{x \in \phi_{uc} \backslash \mathcal{U}\}$ where $(\{0\} \cup \mathcal{X}_e) \in \mathcal{X}_{W_c}$. The control law $u = \Phi_h(x) \in U$ that renders the origin exponentially stable within an open neighborhood ϕ_{uc} is assumed to exist for the hybrid model of Eq. (4.3) such that there exists a $\mathcal{C}^1$ constrained CLBF function $W_c(x)$ that has a minimum at the origin and satisfies the following equations $\forall x \in \phi_{uc}$:

$$\hat{c}_1 |x|^2 \leq W_c(x) - \rho_0 \leq \hat{c}_2 |x|^2 \tag{4.20a}$$

$$\frac{\partial W_c(x)}{\partial x} F_h(x, \Phi_h(x)) \leq -\hat{c}_3 |x|^2, \forall x \in \phi_{uc} \backslash \mathcal{B}_\delta(x_e) \tag{4.20b}$$

$$\left|\frac{\partial W_c(x)}{\partial x}\right| \leq \hat{c}_4 |x| \tag{4.20c}$$

where $\hat{c}_1, \hat{c}_2, \hat{c}_3, \hat{c}_4$ are positive real numbers, $\mathcal{B}_\delta(x_e)$ is a small neighborhood around $x_e \in \mathcal{X}_e$, and $W_c(0) = \rho_0$ is the global minimum value of $W_c(x)$ in ϕ_{uc}. Additionally, assuming continuity and smoothness for system in Eq. (4.1), there exists positive constants M, L_x, L'_x such that the following equations hold

$$|F(x,u)| \leq M \tag{4.21a}$$

$$|F(x,u) - F(x',u)| \leq L_x|x - x'| \tag{4.21b}$$

$$\left|\frac{\partial W_c(x)}{\partial x}F(x,u) - \frac{\partial W_c(x')}{\partial x}F(x',u)\right| \leq L'_x|x - x'| \tag{4.21c}$$

A constrained CLBF that satisfies Eq. (4.19) can be constructed following the method in [160], wherein a CLF and CBF are constructed separately and then combined together.

Consider the hybrid model as shown in Eq. (4.3) with a constrained CLBF function as shown in Eq. (4.19). Assuming a bounded $\mathcal{U}$, i.e., 'not DA' region, we derive simultaneous stability and DA guarantees for the hybrid model. In the scenario of a bounded $\mathcal{U}$, there exist stationary points $\mathcal{X}_e$ such that the continuous controller cannot render the origin exponentially stable. This problem can be tackled by designing them as saddle points and implementing discontinuous control actions such that the state of the system moves away from them in the direction of $W_c(x)$. In the following theorem we show that the hybrid model of Eq. (4.3) can be rendered exponentially stable at the origin using the control law based on constrained CLBF function of Eq. (4.19) while guaranteeing the state of the hybrid model system remains within its DA.

Theorem 1. *Given a region $\mathcal{U}$ 'not DA' for hybrid model of Eq. (4.3) with input constraints $u \in U$, consider a constrained CLBF $W_c(x) : \mathbb{R}^n \to \mathbb{R}$ with minimum at origin. Then the feedback control law $\Phi_h(x)$ guarantees that the closed-loop system stays in $\mathcal{X}_{W_c}$ and does not enter $\mathcal{U}$ for all times for $x(0) = x_0 \in \mathcal{X}_{W_c}$. Additionally, the origin is rendered exponentially stable provided discontinuous control actions i.e., $u = \bar{u}(x) \in U$ are applied at saddle points x_e such that $W_c(x)$ decreases.*

Proof: We first show that $x(t) \notin \mathcal{U}, \forall t \geq 0$ if $x(0) = x_0 \in \mathcal{X}_{W_c}$.

a) (i) Consider that $x_0 \in \mathcal{U}_{\rho_c}$. According to the definition of ϕ_{uc}, it is guaranteed that $\dot{W}_c < 0$ everywhere in $\mathcal{X}_{W_c} \backslash (\mathcal{X}_e \cup \{0\})$. (ii) If $x \in (\mathcal{X}_e \cup \{0\})$, then $\dot{W}_c = 0$. Therefore, from (i) and (ii), it follows that $\dot{W}_c \leq 0$, implying $x(t) \in \mathcal{U}_{\rho_c}\ \forall t \geq 0$ if $x_0 \in \mathcal{U}_{\rho_c}$. According to the definition of W_c being proper and because of its property $\dot{W}_c \leq 0$, $\mathcal{U}_{\rho_c}$ is a compact invariant set. Because $\mathcal{U}_{\rho_c} \cap \mathcal{U} = \emptyset$, it follows that $\forall x_0 \in \mathcal{U}_{\rho_c}$, $x(t) \notin \mathcal{U}$.

b) Next, we show that $x(t) \notin \mathcal{U}, \forall t \geq 0$ if $x(0) = x_0 \in \phi_{uc} \backslash (\mathcal{U}_{\rho_c} \cup \mathcal{U})$. For any $x_0 \in \phi_{uc} \backslash (\mathcal{U}_{\rho_c} \cup \mathcal{U})$, $W_c(x_0) > \rho_c$ and $\dot{W}_c < 0$ along the trajectory of $x(t)$. Because of Eq. (4.19e), it holds that any trajectory starting in $\phi_{uc} \backslash (\mathcal{U}_{\rho_c} \cup \mathcal{U})$ will reach the boundary of $\phi_{uc} \backslash (\mathcal{U}_{\rho_c} \cup \mathcal{U})$ before reaching $\mathcal{U}$), and that $\overline{\phi_{uc} \backslash (\mathcal{U}_{\rho_c} \cup \mathcal{U})} \cap \mathcal{U}_{\rho_c}$ is a nonempty set. Therefore, since $W_c(x) > \rho_c$ within $\phi_{uc} \backslash (\mathcal{U}_{\rho_c} \cup \mathcal{U})$ and $W_c(x) \leq \rho_c$ within $\mathcal{U}_{\rho_c}$ from Eq. (4.19d), $W_c(x) = \rho_c,\ \forall x \in \partial\phi_{uc} \backslash (\mathcal{U}_{\rho_c} \cup \mathcal{U})$ due to the continuity property of W_c. This means that the trajectory after reaching the boundary of $\phi_{uc} \backslash (\mathcal{U}_{\rho_c} \cup \mathcal{U})$ will enter $\mathcal{U}_{\rho_c}$ and remain there. This concludes the proof that $x(t) \notin \mathcal{U}, \forall t \geq 0$ if $x(0) = x_0 \in \mathcal{X}_{W_c}$.

c) However, exponential stability of the origin is not guaranteed when using the controller $u = \Phi_h(x) \in U$ because it is possible that the state of the hybrid model could converge to a saddle point x_e instead of the origin. In order to overcome this issue, continuous control actions (i.e., $\bar{u} \in U$) need to be implemented at the saddle point x_e such that the state of the hybrid model moves away from x_e in the direction of decreasing W_c. These control actions $\bar{u}$ need to be calculated in advance and applied when necessary.

From a), b), and c), it is concluded that the feedback control law $\Phi_h(x)$ guarantees that the state of the hybrid model stays in $\mathcal{X}_{W_c}$ and does not enter $\mathcal{U}$ for all times for $x(0) = x_0 \in \mathcal{X}_{W_c}$.

4.1.1.7 Design of constrained CLBF

In this section, the method for constructing a constrained CLBF is presented. Specifically, a constrained CLBF can be constructed by combining a CLF and CBF which satisfies the constraints of Eq. (4.19). The following definition gives the guidelines as developed in [165], for choosing the CLF and CBF, and the corresponding weights that renders that $W_c(x)$ has a global minimum at the origin.

Definition 4. *Given a region $\mathcal{U}$ 'not DA' for nominal system of Eq. (4.1), assume that there exists a $\mathcal{C}^1$ function $V : \mathbb{R}^n \to \mathbb{R}_+$, and a $\mathcal{C}^1$ function $B : \mathbb{R}^n \to \mathbb{R}$, such that:*

$$c_1|x|^2 \leq V(x) \leq c_2|x|^2, \forall x \in \mathbb{R}^n, c_2 > c_1 > 0 \tag{4.22}$$

$$\mathcal{U} \subset \mathcal{H} \subset \phi_{uc},\ 0 \notin \mathcal{H} \tag{4.23}$$

$$B(x) = -\eta < 0,\ \forall x \in \mathbb{R}^n \backslash \mathcal{H};\ B(x) \geq -\eta,\ \forall x \in \mathcal{H};\ B(x) > 0\ \forall x \in \mathcal{U} \tag{4.24}$$

where $\mathcal{H}$ is a compact and connected set inside ϕ_{uc}. Define $W_c(x)$ to have the form $W_c(x) := V(x) + \mu B(x) + \nu$, where :

$$\left| \frac{\partial W_c(x)}{\partial x} \right| \leq r(|x|) \tag{4.25}$$

$$L_{f_h} W_c(x) < 0,\ \forall x \in \{z \in \phi_{uc} \backslash (\mathcal{U} \cup \{0\} \cup \mathcal{X}_e) | L_{g_h} W_c(z) = 0\} \tag{4.26}$$

$$\mu > \frac{c_2 c_3 - c_1 c_4}{\eta}, \tag{4.27a}$$

$$\nu = \rho_c - c_1 c_4, \tag{4.27b}$$

$$c_3 := \max_{x \in \partial \mathcal{H}} |x|^2 \tag{4.27c}$$

$$c_4 := \min_{x \in \partial \mathcal{U}} |x|^2 \tag{4.27d}$$

then for initial states $x_0 \in \phi_{uc} \backslash \mathcal{U}_\mathcal{H}$, where $\mathcal{U}_\mathcal{H} := \{x \in \mathcal{H} | W_c(x) > \rho_c\}$, the control law $\Phi_h(x)$ with $W_c(x)$ replacing $V(x)$ guarantees that the closed-loop system's state is bounded in $\phi_{uc} \backslash \mathcal{U}_\mathcal{H}$ and does not enter the region $\mathcal{U}_\mathcal{H}$ for all times.

4.1.1.8 CLBF-based model predictive control

In this section, a CLBF-MPC is designed using the hybrid model to optimize process performance while driving the process states to a small bound around the origin. First, we show that the stability and DA guarantees of Theorem 1 hold for the original system of Eq. (4.1) using the controller $u = \Phi_h(x)$ which is designed for the hybrid model system. Next, the CLBF-MPC formulation is presented which drives the state of the original system of Eq. (4.1) to a small neigh-

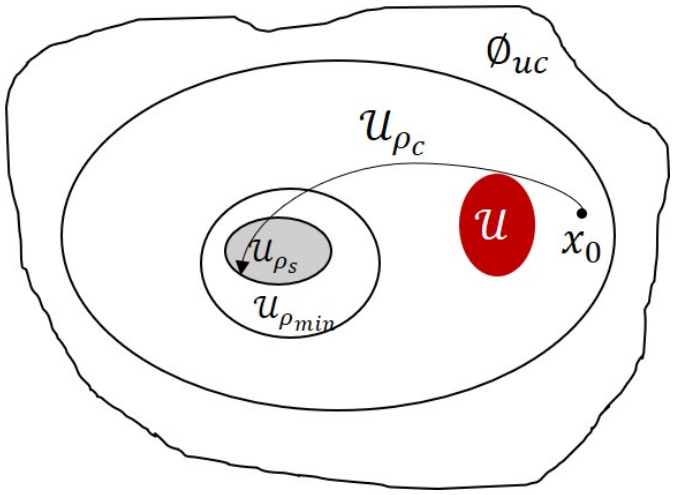


Figure 4.1: Simple schematic representing the various sets $\mathcal{U}_{\rho_c}, \mathcal{U}_{\rho_{min}}, \mathcal{U}_{\rho_s}$ and the bounded region $\mathcal{U}$ which is 'not DA' of the hybrid model.

borhood around the origin under sample-and-hold implementation as shown in Figure 4.1.

Proposition 1. *Consider the original system of Eq. (4.1), and assuming the hybrid model system of Eq. (4.15) has the same initial condition as the system in Eq. (4.1) (i.e., $x(0) = x_h(0)$), there exists a positive constant k such that following inequality hold:*

$$W_c(x) \leq W_c(x_h) + \frac{\hat{c}_4\sqrt{\rho_c - \rho_0}}{\sqrt{\hat{c}_1}}(\delta_o T e^{Lt}) + k(\delta_o T e^{Lt})^2 \tag{4.28}$$

Proof: Since $W_c(x)$ is a continuous function and bounded on compact sets, Taylor series is used to expand it around x_h such that

$$W_c(x) \leq W_c(x_h) + \frac{\partial W_c(x_h)}{\partial x}|x - x_h| + k|x - x_h|^2 \tag{4.29}$$

In Taylor series expansion there exist higher order terms but in our work we bound our expansion using the term $k|x - x_h|^2$. From Eqs. (4.20a) and (4.20c) we obtain

$$W_c(x) \leq W_c(x_h) + \frac{\hat{c}_4\sqrt{\rho_c - \rho_0}}{\sqrt{\hat{c}_1}}|x - x_h| + k|x - x_h|^2 \tag{4.30}$$

The above equation can be further simplified using Eq. (4.12) as follows

$$W_c(x) \leq W_c(x_h) + \frac{\hat{c}_4\sqrt{\rho_c - \rho_0}}{\sqrt{\hat{c}_1}}(\delta_o T e^{Lt}) + k(\delta_o T e^{Lt})^2 \tag{4.31}$$

This completes the proof.

Proposition 2. *Consider the original system of Eq. (4.1) under the controller $u = \Phi_h(x)$ with a bounded region $\mathcal{U}$ i.e., 'not DA' of the hybrid model in Eq. (4.15). Then the stability and DA guarantees of Theorem 1 also hold for the original system of Eq. (4.1) under $u = \Phi_h(x) \in U$ and $u = \bar{u}(x) \in U$ given that Eq. (4.32) is satisfied under discontinuous control actions $u = \bar{u}(x) \in U$ when $x(t_k) = x_h(t_k) \in \mathcal{B}_\delta(x_e)$.*

$$W_c(x_h(t)) < W_c(x_h(t_k)) - \frac{\hat{c}_4\sqrt{\rho_c - \rho_0}}{\sqrt{\hat{c}_1}}(\delta_o T e^{L(t-t_k)}) - k(\delta_o T e^{L(t-t_k)})^2, \forall t > t_k \tag{4.32}$$

Proof: First, to show that the origin of the original system of Eq. (4.1) can be rendered exponentially stable under $u = \Phi_h(x)$, we prove that $\dot{W}_c(x) \leq 0 \; \forall x \in \phi_{uc} \backslash \mathcal{B}_\delta(x_e)$ as follows

$$\dot{W}_c = \frac{\partial W_c(x)}{\partial x} F(x, \Phi_h(x)) = \frac{\partial W_c(x)}{\partial x}(F_h(x, \Phi_h(x)) + F(x, \Phi_h(x)) - F_h(x, \Phi_h(x))) \tag{4.33a}$$

$$= \frac{\partial W_c(x)}{\partial x}(F_h(x, \Phi_h(x))) + \frac{\partial W_c(x)}{\partial x}(F(x, \Phi_h(x)) - F_h(x, \Phi_h(x))) \tag{4.33b}$$

$$(Using\ Eqs.\ (4.20b), (4.20c), assuming\ |F(x,u) - F_h(x,u)| \leq \gamma|x|) \tag{4.33c}$$

$$\leq -\hat{c}_3|x|^2 + \hat{c}_4|x|\gamma|x| \tag{4.33d}$$

$$\leq -\hat{c}_3|x|^2 + \hat{c}_4\gamma|x|^2 \tag{4.33e}$$

Assuming $\gamma < \hat{c}_3/\hat{c}_4$ implies that $\dot{W}_c \leq 0 \; \forall x \in \phi_{uc} \backslash \mathcal{B}_\delta(x_e)$. Next, we show that at the saddle points, discontinuous control actions $u = \bar{u}(x) \in U$ can drive the state of the original system away from the saddle points in the direction of decreasing $W_c(c)$. Assuming that the state of the original system enters a neighborhood around the saddle points at $t = t_k$ such that $x_h(t_k) = x(t_k) \in \mathcal{B}_\delta(x_e)$, Eq. (4.32) holds true for discontinuous control actions $\bar{u}(x_h)$.

Then $\forall t > t_k$, using Eq. (4.31), we get the following:

$$W_c(x(t)) \leq W_c(x_h(t)) + \frac{\hat{c}_4\sqrt{\rho_c - \rho_0}}{\sqrt{\hat{c}_1}}(\delta_o T e^{L(t-t_k)}) + k(\delta_o T e^{L(t-t_k)})^2 \tag{4.34a}$$

$$W_c(x(t)) - \frac{\hat{c}_4\sqrt{\rho_c - \rho_0}}{\sqrt{\hat{c}_1}}(\delta_o T e^{L(t-t_k)}) - k(\delta_o T e^{L(t-t_k)})^2 \leq W_c(x_h(t)) \tag{4.34b}$$

(Using Eq. (4.32))

$$W_c(x(t)) - \frac{\hat{c}_4\sqrt{\rho_c - \rho_0}}{\sqrt{\hat{c}_1}}(\delta_o T e^{L(t-t_k)}) - k(\delta_o T e^{L(t-t_k)})^2 < W_c(x_h(t_k)) - \frac{\hat{c}_4\sqrt{\rho_c - \rho_0}}{\sqrt{\hat{c}_1}}(\delta_o T e^{L(t-t_k)}) - k(\delta_o T e^{L(t-t_k)})^2 \tag{4.34c}$$

$$W_c(x(t)) < W_c(x_h(t_k)) \tag{4.34d}$$

From Eq. (4.34), we can conclude that the state of the original system will move away from the saddle points in the direction of decreasing $W_c(x)$ provided that the discontinuous control actions satisfy Eq. (4.32). Using the results from Eqs. (4.33)-(4.34), we can conclude that the closed-loop state of the original system can be driven to the origin using the control actions $u = \Phi_h(x) \in U$ and $u = \bar{u}(x) \in U$ while avoiding the region $\mathcal{U}$.

4.1.1.9 Sample-and-hold implementation

In this section, we derive the stability properties of the CLBF-based controller under sample-and-hold implementation. Specifically, we show that the closed-loop state of the original system is bounded in $\mathcal{U}_{\rho_c}$ and will be driven to a small neighborhood around the origin $\mathcal{U}_{\rho_{min}}$. The control actions of $u = \Phi_h(x) \in U$ and $u = \bar{u}(x) \in U$ are implemented in sample-and-hold fashion i.e., $\forall t \in [t_k, t_{k+1}), u(t) = u(t_k)$, where $t_{k+1} = t_k + \Delta$, and Δ is the sampling period of the CLBF-MPC.

Proposition 3. *Consider the original system of Eq. (4.1) under the sample-and-hold implementation of controller $u = \Phi_h(x)$ and $u = \bar{u}(x) \in U$ with a bounded region $\mathcal{U}$ i.e., 'not DA' of the hybrid model in Eq. (4.15). Given that Eq. (4.32) is satisfied under sample-and-hold implementation of discontinuous control actions $u = \bar{u}(x) \in U$ when $x \in \mathcal{B}_\delta(x_e)$, and there exist $\epsilon_w > 0$,*

$\Delta > 0$, $\rho > \rho_{min} > \rho_h > \rho_s$ *such that*

$$-\frac{-\hat{c}_3 + \hat{c}_4\gamma}{\hat{c}_2}(\rho_s - \rho_0) + L'_x M\Delta \leq -\epsilon_w \tag{4.35}$$

$$\rho_h := max\{W_c(x_h(t+\Delta))|x_h(t) \in \mathcal{U}_{\rho_s}, u \in U\} \tag{4.36}$$

$$\rho_{min} \geq \rho_h + \frac{\hat{c}_4\sqrt{\rho_c - \rho_0}}{\sqrt{\hat{c}_1}}(\delta_o T e^{L\Delta}) + k(\delta_o T e^{L\Delta})^2, \forall t > t_k \tag{4.37}$$

then $W_c(x(t))$ decreases within every sampling period, and thus, the state of the system remains in $\mathcal{U}_\rho$ for all times and is ultimately bounded in $\mathcal{U}_{\rho_{min}}$.

Proof: Assuming $x(t_k) = x_h(t_k) \in \mathcal{U}_\rho \backslash \mathcal{U}_{\rho_s}$, the time derivative of $W_c(x(t))$ for the original system of Eq. (4.1) is shown below:

$$\dot{W}_c(x(t)) = \frac{\partial W_c(x(t))}{\partial x} F(x(t), \Phi_h(x(t_k))) \tag{4.38a}$$

$$\dot{W}_c(x(t)) = \frac{\partial W_c(x(t_k))}{\partial x} F(x(t_k), \Phi_h(x(t_k))) + \frac{\partial W_c(x(t))}{\partial x} F(x(t), \Phi_h(x(t_k))) \tag{4.38b}$$

$$- \frac{\partial W_c(x(t_k))}{\partial x} F(x(t_k), \Phi_h(x(t_k))) \tag{4.38c}$$

Using Eqs. (4.20),(4.21), and (4.33), we obtain

$$\dot{W}_c(x(t)) = -\frac{-\hat{c}_3 + \hat{c}_4\gamma}{\hat{c}_2}(\rho_s - \rho_0) + L'_x M\Delta \tag{4.39}$$

The above equation does not hold around the neighborhood of saddle points $\mathcal{B}_\delta(x_e)$ since Eqs. (4.20) and (4.33) may not be valid there. Using Eq. (4.39) and assuming Eq. (4.35) holds true, we obtain the following equation $\forall x(t_k) \in \mathcal{U}_\rho \backslash \mathcal{U}_{\rho_s}$ and $t \in [t_k, t_{k+1})$:

$$\dot{W}_c(x(t)) \leq -\epsilon_w \tag{4.40}$$

The above equation ensures that the closed-loop state of the original system stays within $\mathcal{U}_\rho$ under sample-and-hold implementation. Also, if Eq. (4.32) is satisfied under sample-and-hold im-

plementation of discontinuous control actions $u = \bar{u}(x) \in U$, it is proven in Eq. (4.34) that $W_c(x(t)) < W_c(x(t_k))$ holds true for the original system of Eq. (4.1), $\forall t > t_k$. Therefore, under sample-and-hold implementation the state of the original system leaves the neighborhood around saddle points.

Finally, we show that once the state enters $\mathcal{U}_{\rho_s}$ i.e., $x(t_k) = x_h(t_k) \in \mathcal{U}_{\rho_s}$, it is bounded within $\mathcal{U}_{\rho_{min}}$ for the remaining time $t \geq t_k$. Eq. (4.36) states that $\mathcal{U}_{\rho_h}$ is the largest level set of $W_c(x_h)$ that the state of the hybrid model can reach starting from $\mathcal{U}_{\rho_s}$ within one sampling period. Additionally, according to Eq. (4.37), $\mathcal{U}_{\rho_{min}}$ is the largest level set of $W_c(x)$ based on the state of the original system when the hybrid model state x_h is bounded in $\mathcal{U}_{\rho_h}$. Since, $\dot{W}_c(x(t)) \leq -\epsilon_w$ may not always hold true in $\mathcal{U}_{\rho_s}$ under the sample-and-hold implementation of $u = \Phi_h(x) \in U$, the sets $\mathcal{U}_{\rho_h}$ and $\mathcal{U}_{\rho_{min}}$ are defined to guarantee that the states of the original system and the hybrid model are bounded around the origin in a neighborhoods that are bigger than $\mathcal{U}_{\rho_s}$.

This completes the proof that the state of the original system stays within the DA and is bounded within $\mathcal{U}_\rho$ for all times, and is ultimately driven to $\mathcal{U}_{\rho_{min}}$ under the sample-and-hold implementation of the controller $u = \Phi_h(x)$ and $u = \bar{u}(x) \in U$.

4.1.1.10 Mathematical formulation of CLBF-MPC

The optimization problem solved in the CLBF-MPC is as follows:

$$\mathcal{J} = \min_{u \in S(\delta)} \int_{t_k}^{t_{k+N}} L(x_h(t), u(t)) dt, \tag{4.41a}$$

$$s.t.\ \dot{x}_h(t) = F_h(x_h(t), u(t)), \tag{4.41b}$$

$$x_h(t_k) = x(t_k), \tag{4.41c}$$

$$u(t) \in U, \forall t \in [t_k, t_{k+N}), \tag{4.41d}$$

$$\dot{W}_c(x(t_k), u(t_k)) \leq \dot{W}_c(x(t_k), \Phi_h(t_k)), \tag{4.41e}$$

$$if\ x(t_k) \notin \mathcal{B}_\delta(x_e)\ and\ W_c(x(t_k)) > \rho_h$$

$$W_c(x_h(t)) \leq \rho_h, \forall t \in [t_k, t_{k+N}), \tag{4.41f}$$

$$if\ W_c(x(t_k)) \leq \rho_h$$

$$W_c(x_h(t)) < W_c(x(t_k)) - \frac{\hat{c}_4\sqrt{\rho_c - \rho_0}}{\sqrt{\hat{c}_1}}(\delta_o T e^{L(t-t_k)}) - k(\delta_o T e^{L(t-t_k)})^2, \forall t \in (t_k, t_{k+N}), \quad (4.41g)$$
$$if\ x(t_k) \in \mathcal{B}_\delta(x_e)$$

where $x_h(t)$ is the predicted state from the hybrid model, N is the number of sampling times in the prediction horizon, and $S(\Delta)$ is the set of piece-wise constant functions with sampling period Δ. Eq. (4.41a) is the objective function of the MPC optimization problem which is usually in a quadratic form i.e., $x_h^T Q x_h + u^T R u$, where Q and R are positive definite matrices. Eq. (4.41b) is the hybrid model and Eq. (4.41d) are the input constraints. Eqs. (4.41e)-(4.41g) ensure closed-loop stability and DA guarantees for the system of Eq. (3.13).

Theorem 2. *Consider the original system of Eq. (4.1) with a constrained CLBF W_c that satisfies Eq. (4.19) and has a minimum at the origin. For any $x_0 \in \mathcal{U}_\rho$, it is guaranteed that CLBF-MPC optimization problem of Eq. (4.41) can be solved with recursive feasibility for all times. Also, under the sample-and-hold implementation of CLBF-MPC using the hybrid model that satisfy the conditions in Proposition 3, it is guaranteed that the state is bounded in $\mathcal{U}_\rho$ $\forall t \geq 0$ for any $x_0 \in \mathcal{U}_\rho$ and is ultimately bounded in $\mathcal{U}_{\min}$.*

Proof: The proof consists of two parts. In first part we show the recursive feasibility of the CLBF-MPC optimization problem. In second part we prove the simultaneous stability and DA guarantees of the original system Eq. (4.1) under the CLBF-MPC which utilizes a hybrid model of Eq. (4.3).

Part 1: In Propositions 1-3, it has been shown that the controller $u = \Phi_h(x) \in U$ and $u = \bar{u}(x) \in U$ under sample-and-hold implementation satisfy the CLBF-MPC constraints of Eqs. (4.41e)-(4.41g). Specifically, the control actions $u = \Phi_h(x) \in U$ and $u = \bar{u}(x) \in U$ satisfy the input constraint of Eq. (4.41d). Eq. (4.41e) is satisfied by letting $u(t_k) = \Phi_h(x(t_k))$ when $x(t_k) \in \mathcal{U}_\rho \backslash \mathcal{B}_\delta(x_e) \cup \mathcal{U}_{\rho_h}$. Also, it is shown in Proposition 3 that once the state enters $\mathcal{U}_{\rho_s}$ under the controller $u = \Phi_h(x) \in U$, it will not leave $\mathcal{U}_{\rho_h}$ within one sampling time for any $u \in U$. Therefore the constraint of Eq. (4.41f) is satisfied. Lastly, the constraint of Eq. (4.41g) is satisfied as the control actions $u = \bar{u}(x) \in U$ are designed to take the state of the system away from the

saddle points $\mathcal{B}_\delta(x_e)$. This completes the proof of recursive feasibility of the optimization problem in Eq. (4.41).

Part 2: The proof for closed-loop stability and DA guarantees is presented here. Given $x_0 \in \mathcal{U}_\rho \backslash \mathcal{U}_{\rho_h}$, Eq. (4.41e) drives the state towards the origin. If a saddle point is encountered then Eq. (4.41g) becomes active and moves the state away from the saddle point in the direction of decreasing $W_c(x)$. Once the state moves away from the neighborhood of the saddle points $\mathcal{B}_\delta(x_e)$, then closed-loop stability and DA guarantees are obtained using Eqs. (4.41e)-(4.41f). This means that the state of the system stays in $\mathcal{U}_\rho$ for all times and ultimately converges to $\mathcal{U}_{\rho_{min}}$. This completes the proof of simultaneous stability and DA guarantees of the original system Eq. (4.1) under the CLBF-MPC.

4.1.2 Application to a CSTR

In this section, a CSTR example is utilized to show the effectiveness of the proposed CLBF-MPC controller which utilizes a deep hybrid model for predicting the states of the system. We consider a well mixed, nonisothermal CSTR where an irreversible and exothermic reaction $A \rightarrow B$ takes place with second order kinetics. The dynamics of the CSTR system is explained using based on mass and energy conservation laws as shown below:

$$\frac{dC_A}{dt} = \frac{F}{V}(C_{A0} - C_A) - k_0 exp(-\frac{E}{RT})C_A^2 \tag{4.42}$$

$$\frac{dT}{dt} = \frac{F}{V}(T_0 - T) + \frac{-\Delta H}{\rho_L C_p} k_0 exp(-\frac{E}{RT})C_A^2 + \frac{Q}{\rho_L C_p V} \tag{4.43}$$

where C_A is the concentration of reactant A in the CSTR, F is the volumetric flow rate of feed, V is the volume of the CSTR, C_{A0} is the concentration of reactant A in the feed, k_0 is the pre-exponential constant, E is the activation energy, R is the ideal gas constant, T is the temperature of the CSTR, T_0 is the temperature of the feed, ΔH is the enthalpy of reaction, ρ_L is the density of reacting liquid, C_p is the heat capacity of the reacting liquid, and Q is heat input rate. The values of these parameters are shown in Table 4.1.

$F = 5\ m^3/hrK$
$V = 1\ m^3$
$k_0 = 8.46 * 10^6 m^3/kmol\ hr$
$E = 5 * 10^4\ kJ/kmol$
$R = 8.314\ kJ/kmol\ K$
$\Delta H = -1.15 * 10^4\ kJ/kmol$
$\rho_L = 1000\ kg/m^3$
$C_p = 0.231\ kJ/kg\ K$
$T_0 = 300\ K$

Table 4.1: Parameters used in CSTR system.

The set-point for the CLBF-MPC controller is (C_{As}, T_s) =$(1.95\ kmol/m^3,\ 402\ K)$ which is the unstable steady state of the CSTR system. The manipulated inputs are the feed concentration of C_{A0} and the heat input rate Q. The limits on the manipulated input are as follows: $0.5\ kmol/m^3 \leq C_{A0} \leq 7.5\ kmol/m^3$ and $-5 * 10^5\ kJ/hr \leq Q \leq 5 * 10^5\ kJ/hr$. The states and the inputs of the closed-loop CLBF-MPC controller are $x^T = [C_A - C_{As}\ T - T_s]$ and $u = [C_{A0}\ Q]$ such that $(x_s^*,\ u_s^*) = (0, 0)$. The sampling time for the controller is $0.01\ hr$ and the integration time-step to solve the ODEs is $10^{-3}\ hr$.

To build a deep hybrid model, it is assumed that the available first-principles model of the CSTR is as follows:

$$\frac{dC_A}{dt} = \frac{F}{V}(C_{A0} - C_A) - C_A^2 \tag{4.44}$$

$$\frac{dT}{dt} = \frac{F}{V}(T_0 - T) + \frac{-\Delta H}{\rho_L C_p} C_A^2 + \frac{Q}{\rho_L C_p V} \tag{4.45}$$

Numerous open loop simulations of the original system as shown in Eqs. (4.42)-(4.43) are performed to generate the training data. Specifically, different initial conditions are considered in the state-space and inputs within the constraints (i.e., $u \in U$) are implemented such that the generated training data set is large enough to represent the operating region. Next, a DNN $D(x)$ is constructed using 2 hidden layers with 50 neurons each. The two hidden layers have leaky rectified linear unit and linear unit as their activation functions, respectively. Using this DNN, the equations for the proposed deep hybrid model are as follows:

$$\frac{dC_A}{dt} = \frac{F}{V}(C_{A0} - C_A) - C_A^2 + D(T)C_A^2 \tag{4.46}$$

$$\frac{dT}{dt} = \frac{F}{V}(T_0 - T) + \frac{-\Delta H}{\rho_L C_p}C_A^2 + \frac{Q}{\rho_L C_p V} + \frac{-\Delta H}{\rho_L C_p}D(T)C_A^2 \tag{4.47}$$

Using Adam optimizer and an initial learning rate of 0.001, the DNN in the deep hybrid model was trained. Figure 4.2 shows the comparison between the predictions from the deep hybrid model and the true data for a given inputs and initial condition. From Figure 4.2 it is observed that the deep

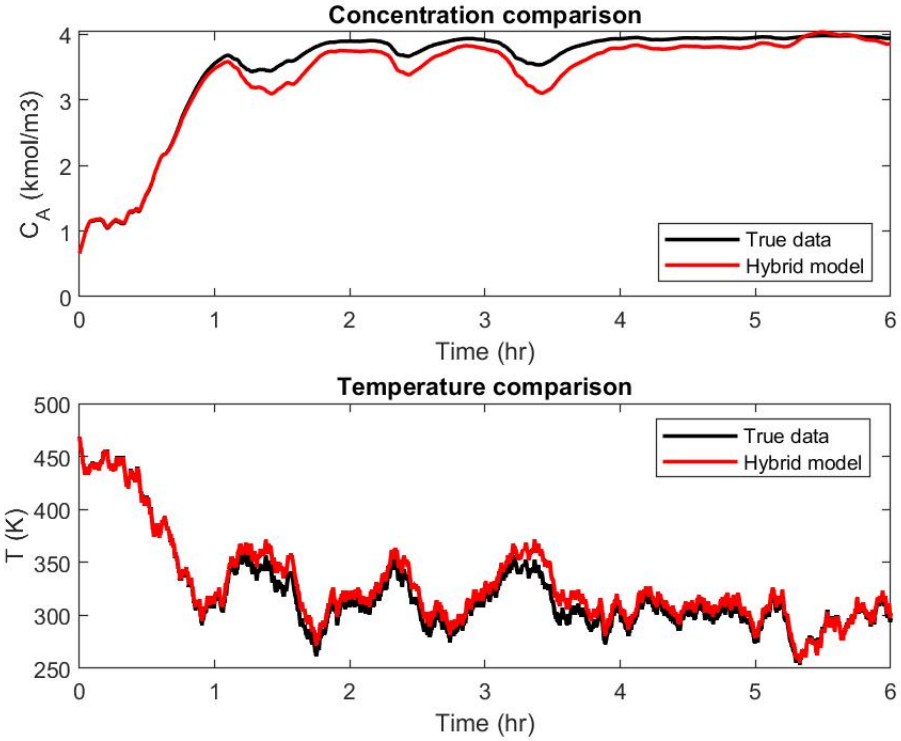


Figure 4.2: Comparison of hybrid model predictions versus actual data.

hybrid model is well trained and can be utilized for building the CLBF-MPC controller.

The objective of the CLBF-MPC controller is to operate the CSTR system at the unstable stead-state by identifying optimal control actions within the DA of the hybrid model. Usually, predictions from the hybrid model are compared to the 'true' model but in real world chemical applications the 'true' model is not available. Therefore, a different method needs to be adopted to identify the DA. In order to identify the DA region for the developed deep hybrid model, the

knn technique is used. For each candidate training point, we identify 10 of its nearest neighbors, and calculate the average Euclidean distance between the candidate point and its neighbors. A tolerance of 0.01 is set for this average distance. Training points with average distance less than the tolerance are considered as within the DA of the deep hybrid model, and training points with average distance greater than the tolerance are considered as outside the DA. Figure 4.3 shows all the training points within the operating region. The training points within the red curve were found to lie outside the DA, and therefore, this small region is not included within the DA of the deep hybrid model. The rest of the training points in the operating region are considered to be part of the DA. The region not considered to be part of the DA of the deep hybrid model can is defined as $\mathcal{U} :=$

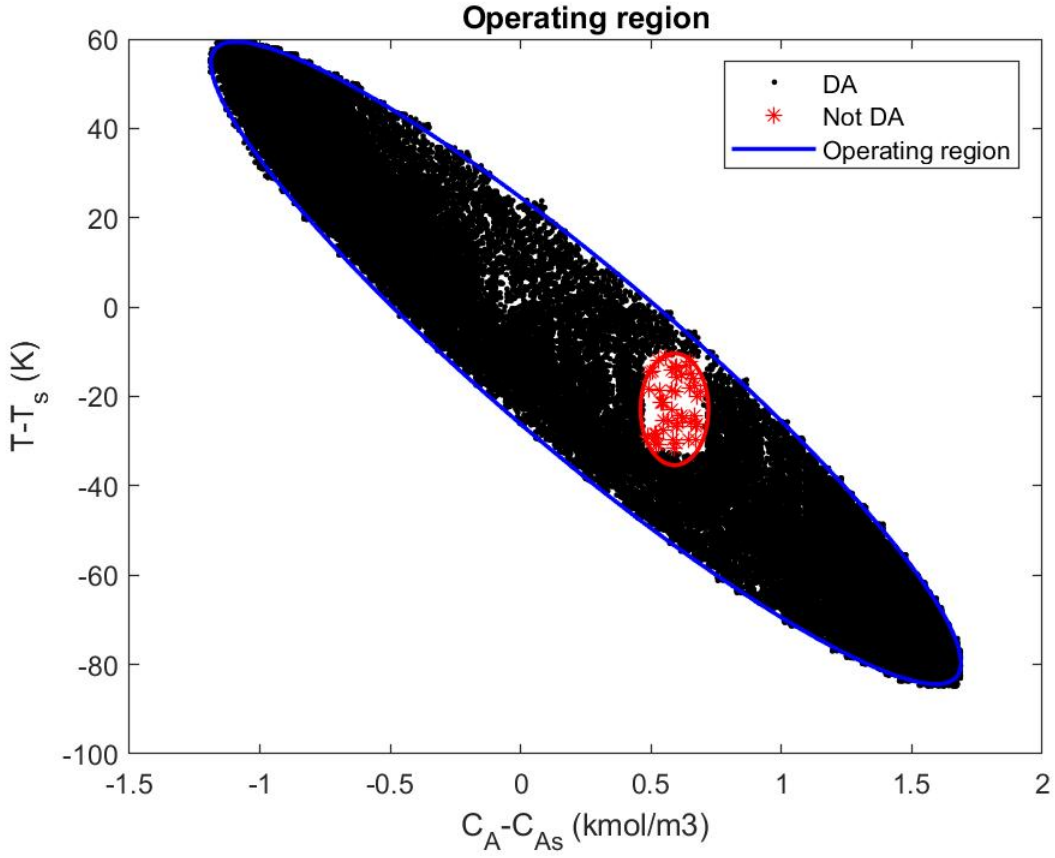


Figure 4.3: Training data and the identified DA of the deep hybrid model.

$\{x \in \mathbb{R}^2 | F_{\mathcal{U}}(x) = \frac{(x(1)-0.5897)^2}{0.9192^2} + \frac{(x(2)+22.8613)^2}{88.3883^2} < 0.02\}$. Considering the practical considerations such as the modeling error between the hybrid model and the actual system, we consider a slightly

larger region for $\mathcal{U}$ such that $\mathcal{U} := \{x \in \mathbb{R}^2 | F_{\mathcal{U}}(x) = \frac{(x(1)-0.5897)^2}{0.9192^2} + \frac{(x(2)+22.8613)^2}{88.3883^2} < 0.03\}$. Using this definition of $\mathcal{U}$, the CBF is defined as follows:

$$B(x) = \begin{cases} e^2 - e^{\frac{F_{\mathcal{U}}(x)}{F_{\mathcal{U}}(x)-0.03}}, & \text{if } x \in \mathcal{U} \\ -e^2, & \text{if } x \notin \mathcal{U} \end{cases} \tag{4.48}$$

The CLF is constructed in the standard quadratic form of $V(x) = x^T P x$ such that P is as follows:

$$P = \begin{bmatrix} 1060 & 22 \\ 22 & 0.52 \end{bmatrix} \tag{4.49}$$

Using the CLF $V(x)$ and CBF $B(x)$, the CLBF $W_c(x) = V(x) + \mu B(x) + \nu$ is constructed with the values of the parameters as follows: $\rho = 0$, $c_1 = 0.1$, $c_2 = 1061$, $c_3 = 1190$, $c_4 = 90$, $\nu = \rho - c_1 c_4 = -9$, and $\mu = 1.71 * 10^5$.

We choose a quadratic objective function for the CLBF-MPC in order to drive the system to the defined set-point while minimizing the inputs i.e., concentration of A in feed as well as the heat supplied/removed. The objective function is given as follows:

$$L(x_h, u) = |x_h(t)|^2_{Q_L} + |u(t)|^2_{R_L} \tag{4.50}$$

where the weight matrices Q_L and R_L are as follows:

$$Q_L = \begin{bmatrix} 20 & 0 \\ 0 & 0.01 \end{bmatrix} \tag{4.51}$$

$$R_L = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.05 \end{bmatrix} \tag{4.52}$$

The MPC sampling time was 0.01 hr and the prediction horizon was chosen to be 15. The constrained nonlinear optimization was solved in MATLAB.

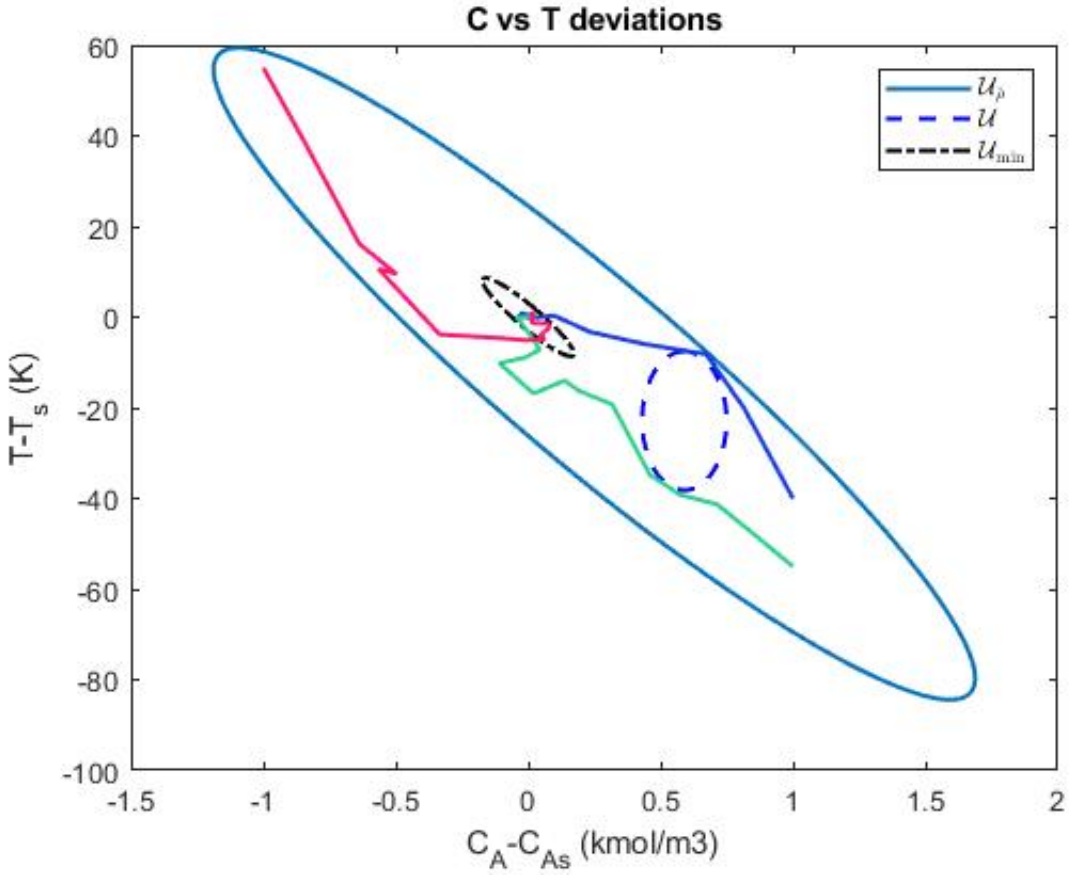


Figure 4.4: Closed loop trajectories of the original CSTR system using deep hybrid model-based CLBF-MPC.

We first show the implementation of the CLBF-MPC in order to obtain an optimal input trajectory that such that the close-loop state of the original CSTR system stays within the DA of the deep hybrid model. In this regard, we select three initial conditions in the subset of $\mathcal{U}_{\hat{\rho}}$ i.e., $(2.95\ kmol/m^3, 362\ K)$, $(2.95\ kmol/m^3, 347\ K)$, and $(0.95\ kmol/m^3, 457\ K)$. Figure 4.4 shows the closed-loop trajectory starting from these initial conditions under the CLBF-MPC controller. Figure 4.5 and Figure 4.6 are the inputs implemented by the CLBF-MPC controller. Figure 4.4 shows that for any initial state within $\mathcal{U}_{\hat{\rho}}$ (a subset of $\mathcal{U}_{\rho}$), the closed-loop states of the original CSTR system stay within the DA of the deep hybrid model, avoid the region $\mathcal{U}$. and ultimately converge to $\mathcal{U}_{\min}$ under the deep hybrid model-based CLBF-MPC.

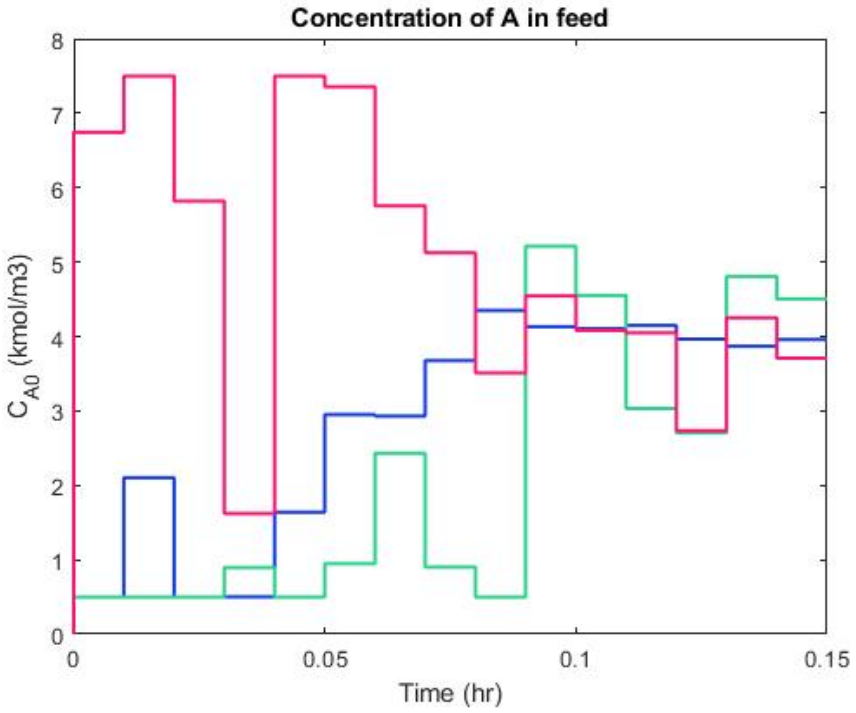


Figure 4.5: Profile of concentration of A in feed for the three initial conditions under CLBF-MPC controller.

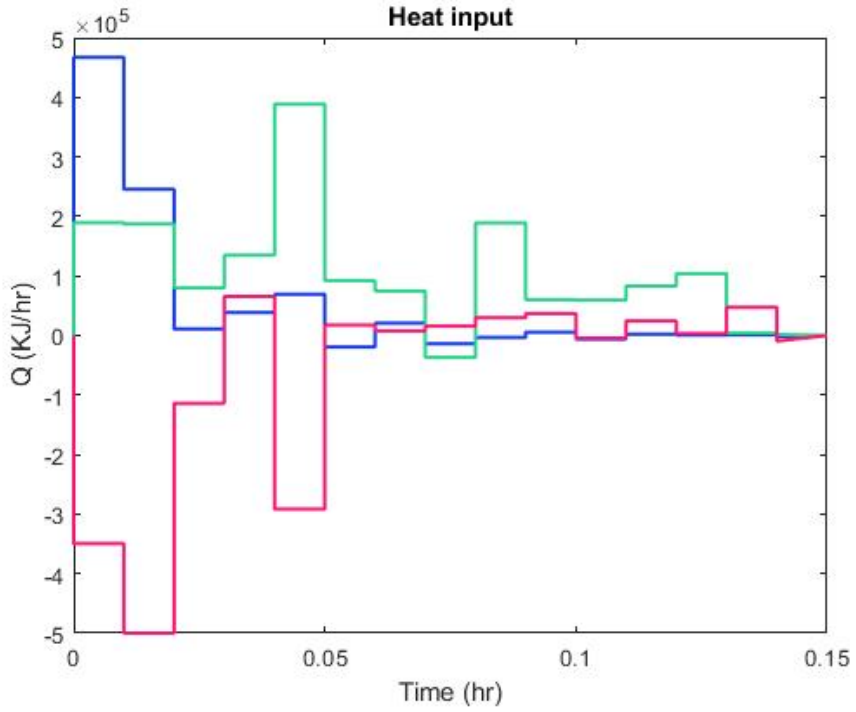


Figure 4.6: Heat input profile for the three initial conditions under CLBF-MPC controller.

5.1 Deep reinforcement learning control of hydraulic fracturing*

Hydraulic fracturing is performed for extraction of oil and gas from rocks that have low porosity and low permeability [168]. This is achieved by first carrying out controlled explosions within the formation to create initial fracture paths. Next, a fluid called pad is injected at high pressures to extend the initial fracture paths, which is followed by injection of a fracturing fluid which consists of water, additives, and proppant at high pressures in order to further extend these fractures into the rock formation. Materials other than water have also been considered to enhance the productivity of the hydraulic fracturing process [169, 170, 171, 172]. Once pumping is stopped, these fractures close due to the natural stresses in the rock formation as the fracturing fluid leaks into the reservoir, leaving behind proppant in the fractures. The trapped proppant acts as a highly conductive medium for easier extraction of oil and gas. The proppant concentration and the fracture geometry are the two main factors that affect the efficiency of the hydraulic fracturing process. To achieve the desired values for these attributes, it is necessary to design an optimal pumping schedule. Many works have been conducted in this direction [173, 174, 4]. These works consider this control problem in an open-loop formulation. Additionally, there have been efforts to design model predictive control (MPC) schemes for hydraulic fracturing processes after advances in real-time measurement techniques such as downhole pressure analysis and microseismic monitoring [102, 5, 3, 175, 176, 177, 107, 178, 179]. A ROM-based feedback control system was designed to maximize the net present value (NPV) of oil produced from a horizontal well before gas breakthrough by computing optimal oil production profile [180]. An optimization framework was developed to obtain optimal multi-size proppant pumping schedule that maximizes shale gas

*Reprinted with permission from "Deep reinforcement learning control of hydraulic fracturing" by Bangi, M. S. F., and Kwon, J. S. 2021. Comput. Chem. Eng., 154, 107489, Copyright 2021 Elsevier.

production from unconventional reservoirs using the MP-PIC model [181]. The effect of varying proppant diameters across pumping stages on shale gas production was modeled to obtain optimal pumping schedule that maximizes cumulative shale gas production volume [182]. A control framework was proposed to incorporate sustainability considerations in hydraulic fracturing via the removal of total dissolved solids (TDS) in flowback water from fractured wells using thermal membrane distillation [183]. A novel model-based controller was developed to maximize the net profit from shale gas development while keeping the total cost associated with water management to a minimum [184]. For efficient and sustainable water management, a comprehensive study was conducted to better understand freshwater consumption and flowback and produced water production for shale gas wells to expand and upgrade the existing water and shale gas network [185]. An optimization model was developed which uses 'enhanced gas recovery by carbon dioxide (CO2) injectionâĂŹ (EGR-CO2) technology to identify the optimal shale gas supply chain network configuration in a MILP problem that maximizes the profit obtained from shale gas production [186]. An optimal pumping schedule was designed to obtain optimal fracture geometry and proppant concentration in a fracture which was obtained using a non-Newtonian high-viscosity gel [187]. However, these model-based control systems require an accurate model which is difficult to build given the various uncertainties in the rock formation and is an ongoing research area [188, 189, 190, 118, 97, 191, 151, 192]. Additionally, the performance of a MPC system depends on its tuning parameters, and the accuracy of the process model. It is a common practice to continuously monitor the controller's performance and begin a model re-identification process or re-tune the parameters of the controller in case the controller performance degrades, which is time-consuming and is resource intensive. To summarize, there are two challenges with designing a model-based controller for hydraulic fracturing process: a) Its first principles model involves highly-coupled partial differential equations (PDEs) with moving boundaries which are computationally expensive to solve at each sampling time, and b) Regular re-tuning of controller and model parameters.

To address the limitations of a model-based controller, we design a model-free data-based con-

troller suitable for nonlinear chemical processes, specifically for hydraulic fracturing, by combining concepts from reinforcement learning (RL) and deep learning (DL). RL is a branch of machine learning which deals with solving complex decision making problems, and it involves an agent which interacts with the environment (process) to derive an optimal policy in order to reach the desired target [193, 194]. RL has delivered tremendous success in computer games [195, 196], board games [197, 198], robotics [199, 200], etc. Furthermore, developments in DL have enabled its combination with RL, which has achieved huge success such as AlphaGo defeating human champions in the game Go [201]. Another RL algorithm called deep-Q-network (DQN) [195] has achieved the human level performance in Atari video games. Despite its success in other domains, application of RL to process control has been very limited [202, 203, 204, 205, 206, 148] even though many process control problems can be defined as Markov decision processes (MDPs) [207]. The challenge is the lack of RL algorithms that can efficiently deal with continuous state and action spaces, which is usually the case with process control applications. It is possible to discretize and use Dynamic programming (DP) to obtain a solution to the RL problem in such cases, but there is an exponential growth in computational complexity with respect to the number of states and actions which is referred to as the curse of dimensionality [208]. Approximate dynamic programming (ADP) was proposed, which utilizes simulations and function approximators to overcome this challenge (i.e., curse of dimensionality). In addition to ADP, many other RL algorithms have been proposed for continuous-time nonlinear systems [209, 210]. But these algorithms require high-accuracy models that are either available or that can be identified using system identification methods. Given the limitations of model-based RL methods, several data-based RL methods have been proposed, which come with their own limitations [207]. The recent success of combining RL with DL has led to a resurgence of interest in data-based RL for continuous state and action spaces.

In this work, we design a deep reinforcement learning (DRL) controller which is based on actor-critic approach and temporal difference (TD) learning [149, 211]. The actor (controller) interacts with the process iteratively and implements control actions that give maximum rewards.

The critic, as the name suggests, evaluates the control policy followed by the actor and modifies it to achieve the optimal policy. In the DRL controller, the actor and the critic are both represented by two deep neural networks (DNNs) in order to effectively generalize them for continuous-time variables. The DRL controller also utilizes deep deterministic policy gradient (DDPG) algorithm [212] which is usually used for continuous action spaces. We also utilized concepts such as replay memory (RM), target networks and constrained action spaces to make learning more suitable for complex systems like hydraulic fracturing [200]. Replay memory (RM) is used to break the temporal correlation between two consecutive experiences obtained from the process. Without the RM, the DRL controller will learn from temporally correlated tuples of online data which will lead to inefficient learning. Taking random experiences from the RM breaks this correlation, and hence, speeds up the learning process. Target networks are separate networks used to stabilize the learning process by providing stable targets to the actor and the critic networks. At the beginning of the learning process, these networks are initialized as the copies of the actor and the critic networks, and during the learning process their parameters are constrained to change slowly, which enhances their stability. Action spaces are to RL what control input spaces are to process control. The relationship between the controller's action in the DRL framework and the control input from the process control perspective is direct. The DRL controller traverses the action space to obtain an optimal control policy for the defined control problem. In order to enforce constraints on the action space, we included an action-based reward function in the overall reward calculation. Also, based on the prior knowledge about the process from the literature, we know that the optimal solution should follow a monotonically increasing profile [173]. But Nolte's power law pumping schedule is practically infeasible to implement. Hence, a step-wise increasing profile which is practical to implement is desired from the DRL controller. This can be obtained by enforcing a constraint on the amount of change in two consecutive inputs. We include this information in the overall reward function. Additionally, we utilized Principal Component Analysis to reduce the dimension of the RL state before using it in the learning of the actor and the critic. Transforming the RL state to the reduced PCA space helped in faster learning of the control policy. Moreover, we utilized transfer

learning wherein the DRL controller learns offline using a data-based reduced-order model (ROM) first and then learns online from the process. We summarize the novelty of our framework as follows [150]: 1) We propose to use a data-based model-free DRL controller for control of a hydraulic fracturing process which is a complex moving-boundary system and is difficult to develop a highly accurate model; 2) We propose to utilize PCA in the DRL control framework to reduce the dimension of the RL state; 3) We propose a cumulative reward function to handle various process constraints of the hydraulic fracturing process; and 4) We propose the use of transfer learning for the DRL controller to reduce the online learning time.

5.1.1 Background

5.1.1.1 Reinforcement learning

In RL framework, an agent interacts with the environment E at its current state s_t by implementing control action a_t and receiving a reward of r_t. The cumulative future discounted reward is given by

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{5.1}$$

where the discount factor is $0 < \gamma \leq 1$. The expected return Q after implementing action a_t on state s_t is defined as

$$Q(s,a) = \mathbb{E}[R_t \big| s_t = s, a_t = a] \tag{5.2}$$

The optimal action-value is defined as the maximum expected return after implementing action a_t on state s_t, and is given as

$$Q^*(s,a) = max\mathbb{E}[R_t \big| s_t = s, a_t = a] \tag{5.3}$$

The optimal action-value Q^* can be calculated by iteratively solving the Bellman equation which is shown below:

$$Q^*(s,a) = max\mathbb{E}[r + \gamma max Q^*(s',a') \big| s, a] \tag{5.4}$$

where s' and a' are the subsequent state and action, respectively. Overall, the solution to a RL problem is obtained by implementing control actions on the environment and by learning an optimal control policy by receiving data from it.

5.1.2 Actor-Critic framework

The actor-critic framework is a widely used RL algorithm as it can be generalized to systems with continuous spaces. It has two components, i.e., the actor and the critic. The actor finds an optimal policy, and the job of the critic is to evaluate the policy calculated by the actor. With each iteration of learning, the actor is updated by the policy gradient theorem by adjusting the parameters of the policy function which can be represented using function approximators like neural networks. The critic estimates the action-value function Q which can be represented using a neural network and its parameters are updated using stochastic gradient descent (SGD) method such that the Bellman optimality condition is reached. A schematic diagram of the actor-critic algorithm is shown in Figure 5.1.

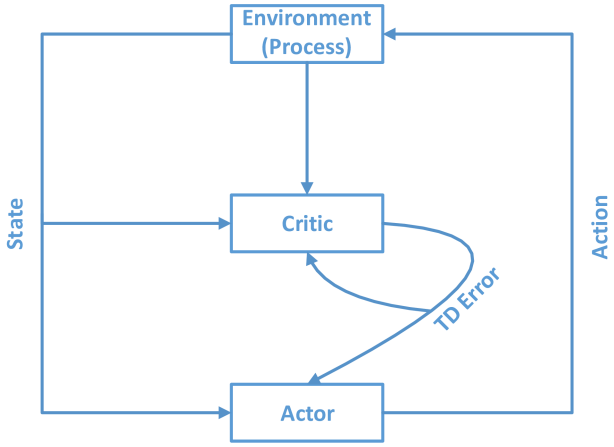


Figure 5.1: A schematic of the actor-critic framework.

5.1.3 Deep reinforcement learning (DRL) controller

The DRL controller [149] is a model-free controller based on the actor-critic framework. It utilizes two DNNs; one to generalize the actor in the continuous state space, and the other to

generalize the critic in the continuous state and action spaces.

5.1.3.1 States and actions

Let u_t and y_t be the input applied on the system at time t and output from the system, respectively. The RL action a_t is same as the input u_t as understood from a control perspective. Therefore, $a_t = u_t$. But the relationship between RL state $s_t \in S$ and the state of the system is different as the RL state s_t should contain information about the system deemed necessary for the successful working of the RL controller. An example of a RL state definition can contain past outputs and the current deviation from the set-point as shown below:

$$s_t := [y_t, y_{t-1}, ...y_{t-d_y}, (y_t - y_{sp})] \tag{5.5}$$

where d_y is the number of past outputs to be included in the DRL controller. The state of the system is initialized as y_0 in every episode, and y_{sp} is the defined set-point for the controller. During the learning process, in every episode, and at every time step, the RL state s is updated as we obtain measurements from the system. Additionally, the RL controller computes a deterministic control policy μ for each state $s_t \in S$ such that $\mu : S \rightarrow A$, where $a_t \in A$.

5.1.3.2 Reward functions

The goal of the RL controller is to reach the set-point using an optimal policy μ that maximizes the aggregate reward that the agent gains from the system. Two examples of reward functions $r : S \times A \times S \rightarrow \Re$ are shown below:

$$r(s_t, a_t, s_{t+1}) = \begin{cases} c & \text{if } |y_{i,t} - y_{i,sp}| \leq \epsilon \\ -\sum_{i=1}^{n} |y_{i,t} - y_{i,sp}| & \text{otherwise} \end{cases} \tag{5.6}$$

$$r(s_t, a_t, s_{t+1}) = \begin{cases} 0 & \text{if } |y_{i,t} - y_{i,sp}| > |y_{i,t+1} - y_{i,sp}| \quad \forall i \in \{1, ...n\} \\ -1 & \text{otherwise} \end{cases} \tag{5.7}$$

where $y_{i,t}$ is the i^{th} output, $y_{i,sp}$ is the set-point for the i^{th} output, $y_{i,t+1}$ is the subsequent i^{th} output, and ϵ is the user defined. As per Eq. (5.6), the agent receives a reward of $c > 0$ only if all the outputs are within the tolerance limit. The reward function of Eq. (5.6) leads to faster tracking but has the disadvantage of obtaining an aggressive control strategy. On the other hand, Eq. (5.7), a polar reward function, assigns a reward of 0 if the deviation from the set-point is monotonically decreasing for all n outputs at each sampling time, and assigns a reward of -1 otherwise. This reward function incentivizes gradual improvement towards the set-point, which results in a smoother tracking performance and a less aggressive control strategy.

5.1.3.3 DNNs as function approximators

The DRL controller utilizes two DNNs to approximate the policy and the Q functions. The actor utilizes a DNN with parameters W_a to generalize the policy function over the continuous action space such that given the state s_t, it produces control actions $a_t = \mu(s_t, W_a)$. Likewise, the critic utilizes a DNN with parameters W_c to generalize the Q function such that given the state s_t and action a_t, it produces Q values as network outputs, i.e., $Q^{\mu}(s_t, a_t, W_c)$.

5.1.3.4 DRL training

The objective of DRL controller training is to calculate the parameter values W_a and W_c such that, once the networks are trained, the actor network can be used to obtain the optimal control actions for any given state.

The DDPG algorithm [200] was proposed in order to improve the trainability of the existing policy gradient theorem. DDPG borrows a concept of mini-batch training from DQN method which involves learning in mini-batches rather than learning directly from online data. DQN utilizes a RM to store historical data in the form of $[s^{(i)}, a^{(i)}, r^{(i)}, s'^{(i+1)}]$, and during the training process, mini-batches of data are sampled from the RM which are used to update the parameters of the DNNs that represent the actor and the critic. Using mini-batches of size M from the RM helps in stabilizing the training process, and randomly sampling tuples from the RM helps in breaking the temporal correlations between the samples. At each sampling time, the latest tuple is stored

in the RM, and an old tuple is discarded from the RM in order to keep its size constant. The parameters of the critic are updated using SGD and samples from the RM so that the TD error is minimized and the update equation is given as follows:

$$W_c \leftarrow W_c + \frac{\alpha_c}{M}\sum_{i=1}^{M}[(\tilde{y}^{(i)} - Q(s^{(i)}, \mu(s^{(i)}, W_a), W_c)) * \nabla_{W_c} Q^{\mu}(s^{(i)}, \mu(s^{(i)}, W_a), W_c)] \tag{5.8}$$

where

$$\tilde{y}^{(i)} \leftarrow r^{(i)} + \gamma Q^{\mu}(s'^{(i)}, \mu(s'^{(i)}, W_a), W_c), \forall i = 1, ..., M \tag{5.9}$$

The parameters of the actor are updated in a batch-wise manner using M samples from the RM, following the DPG theorem, and the update equation is as follows:

$$W_a \leftarrow W_a + \frac{\alpha_a}{M}\sum_{i=1}^{M}[\nabla_{W_a}\mu(s^{(i)}, W_a)\nabla_a Q^{\mu}(s^{(i)}, a, W_c)\big|_{a=a^{(i)}}] \tag{5.10}$$

To further stabilize the learning process, target networks are used to provide stable targets to the critic. In Eq. (5.8), the parameters W_c are utilized to calculate $\tilde{y}$ in Eq. (5.9), which is a target for the critic network. If the target updates are erratic then the parameter updates are also erratic, which may cause the network to diverge. Hence, separate neural networks called target networks are utilized to provide stable targets to be used in Eq. (5.9). Suppose the parameters of the target networks are W'_a and W'_c, then, Eq. (5.9) changes as follows:

$$\tilde{y}^{(i)} \leftarrow r^{(i)} + \gamma Q^{\mu}(s'^{(i)}, \mu(s'^{(i)}, W'_a), W'_c), \forall i = 1, ..., M \tag{5.11}$$

where

$$W'_c \leftarrow \tau W_c + (1 - \tau)W'_c \tag{5.12}$$

$$W'_a \leftarrow \tau W_a + (1 - \tau)W'_a \tag{5.13}$$

where τ is the target network update rate. As per Eqs. (5.12) and (5.13), the parameters of the target networks slowly track the critic and the actor network, which ensures that the targets are

changed slowly, thereby, stabilizing the learning process.

In practical applications, even though the action space is usually continuous, it is bounded too. This is because the control action is usually a representative of physical quantities like flow rate, pressure, and these physical quantities have limits. In order to ensure that the control actions are predicted within the set control limits, it is necessary to bound the actor network. If the actor network is not bounded, then the critic will continue to push the actor to predict control actions outside the control limits. To avoid this scenario, in our work we use gradient clipping to bound the output layer of the actor network. This is done by multiplying the gradient used in the update step, Eq. (5.10), with an appropriate factor. Suppose the action space is bounded in the interval $[a_L, a_H]$ such that $a_L < a_H$, then the gradient clipping is done as follows:

$$\nabla_a Q^\mu(s,a,w) \leftarrow \nabla_a Q^\mu(s,a,w) * \begin{cases} (a_H - a)/(a_H - a_L) & \text{if } \nabla_a Q^\mu(s,a,w) \text{ increases } a \\ (a - a_L)/(a_H - a_L) & \text{otherwise} \end{cases} \tag{5.14}$$

With gradient clipping in place, the control actions will saturate at the upper bound a_H when the critic continually recommends increasing the control actions. On the other hand, the gradient clipping will ensure that the control actions do not decrease beyond the lower limit a_L when the critic continually recommends decreasing the control actions. Combining the above described concepts, the Algorithm 4 lists the steps in the training of the DRL controller. As presented in Algorithm 4, RL training is started by first initializing the parameters of the actor and the critic network. Then the target networks are initialized as outlined in Line 3 of the algorithm. In Line 4, RM is initialized with tuples of historical data. Then, for each episode, a set-point is defined (Line 5). Now for each time step of the episode, the RL state s is defined (Line 8), a control action a_t is obtained from actor, which is implemented on the system to obtain a new output y_{t+1} and reward r_t (Lines 9-10), and the latest tuple (s, a_t, s', r_t) is stored in the RM (Line 12). Now, M samples are uniformly drawn from the RM to update the actor network, the critic network, and the target networks (Lines 13-23). Lines 8-23 are repeated until the end of the episode.

Algorithm 4 DRL controller training

Output: Optimal control policy $\mu(s, W_a)$
Initialize W_a, W_c
Initialize $W'_a \leftarrow W_a$, $W'_c \leftarrow W_c$
Initialize RM with historical data
for each episode **do**
  Set the set-point for episode as y_{sp}
  for each step $t = 0, 1, ...T-1$ **do**
    Set $s \leftarrow [y_t, y_{t-1}, ...y_{t-d_y}, (y_t - y_{sp})]$
    Set $a_t \leftarrow \mu(s, W_a)$
    Implement a_t and obtain y_{t+1} and r_t
    Set $s' \leftarrow [y_t, y_{t-1}, ...y_{t-d_y}, (y_t - y_{sp})]$
    Store tuple (s, a_t, s', r_t) in RM
    Obtain M tuples from RM
    for $i = 1, ...M$ **do**
      $\tilde{y}^{(i)} \leftarrow r^{(i)} + \gamma Q^{\mu}(s'^{(i)}, \mu(s'^{(i)}, W'_a), W'_c)$
    end
    Update W_c using Eq. (5.8)
    for $i = 1, ...M$ **do**
      Calculate $\nabla_a Q^{\mu}(s^{(i)}, a, W_c)\big|_{a=\mu(s^{(i)}, W_a)}$
      Clip gradient using Eq. (5.14)
    end
    Update W_a using Eq. (5.10)
    Update W'_c and W'_a using Eqs. (5.12) and (5.13), respectively
  end
end

5.1.4 Design of DRL controller for hydraulic fracturing

We used a dynamic model of the hydraulic fracturing process as described in Eqs. (2.4)-(2.17) and for more details one can refer to [4]. In our work, we design a DRL controller with the objective of obtaining an optimal control policy that leads to a uniform proppant concentration profile at the end of the pumping process.

5.1.4.1 RL state definition and dimensionality reduction

In order to obtain a uniform concentration profile at the end of the pumping process, we define a set-point C_{sp} for the concentration at 6 different locations along the length of the fracture and use these concentration variables as the outputs to be controlled. Let these be variables be represented as C_i where $i = 1, ..., 6$. Another important parameter in the hydraulic fracturing process is the total amount of proppant injected into the fracture A_t by time t. The total amount to be injected in

the entire pumping process is prefixed, and this criteria has to be met by the DRL controller at the end of the pumping process. In order to ensure that this constraint is satisfied, we include A_t in the output definition with the prefixed value as its set-point A_{sp}. Therefore, the output vector at time t is as follows:

$$y_t = [C_{1t} \quad C_{2t} \quad C_{3t} \quad C_{4t} \quad C_{5t} \quad C_{6t} \quad A_t]^T \tag{5.15}$$

In our work, unlike Eq. (5.5), we propose to use a simpler definition of RL state as follows:

$$s_t := [y_t] \tag{5.16}$$

In order to quicken the learning process, we propose to reduce the dimension of the RL state by using Principal Component Analysis (PCA) on the concentration vector, i.e., $[C_{1t}\ C_{2t}\ C_{3t}\ C_{4t}\ C_{5t}\ C_{6t}]$. We use historical data to obtain the Principal Components (PCs) and select the most dominant one to calculate the corresponding PCA score. Therefore, the reduced output vector at time t is as follows:

$$y_{r,t} = [C_{r,t}\ A_t]^T \tag{5.17}$$

Using the reduced output vector, the RL state definition is changed as follows:

$$s_t := [y_{r,t}] \tag{5.18}$$

5.1.4.2 Action

Action a_t is the concentration of proppant injected into the fracture at time t. The unit of a_t is in terms of ppga which means one pound of proppant added to a gallon of water. The range of a_t in ppga is $[0\ 10]$ but is normalized to $[0\ 1]$, and the sampling time is 100 seconds.

5.1.4.3 Reward function

At each time step, the controller receives a reward r_t from the process whose aggregate value is to be maximized by the controller. Since the objective of the controller in set-point tracking is

to minimize the tracking error, we incorporate this in the reward function r_1 as follows:

$$r_1(t) = \begin{cases} 1 - r_{tracking}(t) & \text{if } t < t_{end} \\ 1 & \text{if } |y_{i,end} - y_{i,sp}| \leq \epsilon \quad \forall i \in \{1, 2...7\} \\ penalty * [1 - r_{tracking}(t)] & \text{if } |y_{i,end} - y_{i,sp}| > \epsilon \quad \forall i \in \{1, 2...7\} \end{cases} \tag{5.19}$$

where $r_{tracking}$ is the squared euclidean distance of the elements of the output vector from their respective set-points, and is defined as:

$$r_{tracking}(t) = \omega_1 * \sum_{i=1}^{6} \frac{|C_{it} - C_{sp}|^2}{6} + \omega_2 * |A_t - A_{sp}|^2 \tag{5.20}$$

where since the variables C_i and A are normalized between 0 and 1, the range of r_1 is $[0\ 1]$. The weights ω_1 and ω_2 indicate the significance of the variables C_i and A, respectively, to the reward function r_1.

Additionally, in our work, we do not use the gradient clipping technique to ensure that the control actions are within the feasible range $[a_L\ a_H]$ as specified in Algorithm 1. Instead we use a reward function r_2 to achieve this goal. The reward function r_2 is defined as follows:

$$r_2(t) = \begin{cases} 0 & \text{if } a(t) \in [0\ 1] \\ -\frac{a(t)*(a(t)-1)}{0.25} & \text{otherwise} \end{cases} \tag{5.21}$$

Since the range of $[a_L\ a_H]$ is normalized to $[0\ 1]$, we use the product term $a(t) * (a(t) - 1)$ in r_2 to penalize the controller if the control actions predicted are outside the feasible range.

Also, based on the knowledge about the process from the literature, we know that the optimal solution should follow a monotonically increasing profile [173]. But Nolte's power law pumping schedule is practically infeasible to implement. Hence, a step-wise increasing profile which is practical to implement is desired from the DRL controller. This can be obtained by enforcing a constraint on the amount of change in two consecutive inputs. We include this information in the

form of a reward function r_3, which is defined as follows:

$$r_3(t) = \begin{cases} 0 & \text{if } \Delta a(t) \in [0\ 0.3] \\ -\frac{\Delta a(t) * (\Delta a(t) - 0.3)}{0.0225} & \text{otherwise} \end{cases} \tag{5.22}$$

where $\Delta a(t) = a(t) - a(t-1)$. The reward function r_3 ensures that the controller learns a control policy which is monotonically increasing with an increment less than 4 ppga/stage. The upper limit of 4 ppga/stage when normalized is equal to 0.3.

Therefore, considering all the constraints, the net reward r_t that the controller receives at time t is the cumulative sum of the rewards obtained using Eqs. (5.19)-(5.22) as shown below:

$$r_t = r_1(t) + r_2(t) + r_3(t) \tag{5.23}$$

Considering the reduced RL state definition and the tailor-made reward function, the algorithm for training the DRL controller for the hydraulic fracturing process is shown in Algorithm 5.

5.1.4.4 DRL controller learning

In our work, we use two stages of learning. In the first stage, the controller learns using a reduced-order-model (ROM) of the process and the learning process is terminated when the controller learns a sub-optimal policy. In the second stage of learning, transfer learning is used wherein the sub-optimal controller parameters are used as initial values, and the controller continues to learn by interacting with the process directly. A schematic of our learning strategy for the DRL controller is shown in Figure 5.2.

5.1.4.5 DRL controller hyperparameters

The actor and the critic are each represented using a DNN. Each of the DNNs has two hidden layers, where the first hidden layer consists of 400 neurons and the second hidden layer consists of 300 neurons. A large number of neurons are utilized because these networks have to represent complex policy and value functions for continuous state and action spaces. Rectified linear unit

Algorithm 5 DRL algorithm for hydraulic fracturing

Output: Optimal control policy $\mu(s, W_a)$
Initialize W_a, W_c
Initialize $W'_a \leftarrow W_a$, $W'_c \leftarrow W_c$
Initialize RM with historical data
Calculate the dominant PC
Set the set-point for DRL controller as y_{sp}
for each episode **do**
  for each step $t = 0, 1, ...T-1$ **do**
    Calculate $y_{r,t}$ using y_t and PC
    Set $s \leftarrow [y_{r,t}]$
    Set $a_t \leftarrow \mu(s, W_a)$
    Implement a_t and obtain y_{t+1}
    Calculate $y_{r,t+1}$ using y_{t+1} and PC
    Set $s' \leftarrow [y_{r,t+1}]$
    Calculate r_t using Eq. (5.23)
    Store tuple (s, a_t, s', r_t) in RM
    Obtain M tuples from RM
    for $i = 1, ...M$ **do**
      $\tilde{y}^{(i)} \leftarrow r^{(i)} + \gamma Q^{\mu}(s'^{(i)}, \mu(s'^{(i)}, W'_a), W'_c)$
    end
    Update W_c using Eq. (5.8)
    for $i = 1, ...M$ **do**
      Calculate $\nabla_a Q^{\mu}(s^{(i)}, a, W_c)\big|_{a=\mu(s^{(i)}, W_a)}$
    end
    Update W_a using Eq. (5.10)
    Update W'_c and W'_a using Eqs. (5.12) and (5.13), respectively
  end
end

and linear activation functions were used in the hidden layers and output layer, respectively. For the first stage of learning, the parameters of the actor and the critic network, i.e., the weights and the biases, were initialized using Xavier initialization as this helps in maintaining constant variance in the outputs from the neurons across every layer. This constant variance helps prevent vanishing or exploding gradients. Also, batch normalization [213] was used in the hidden layers in order to ensure that the training is effective as different variables could have different units and could vary on different scales. Finally, we used Adam optimizer [214] in order to train the networks as it is computationally efficient and well-suited for our optimization problem with a large number of parameters. Adam optimizer combines the advantages of AdaGrad and RMSProp, two popular stochastic optimization methods, by computing adaptive learning rates for each parameter using

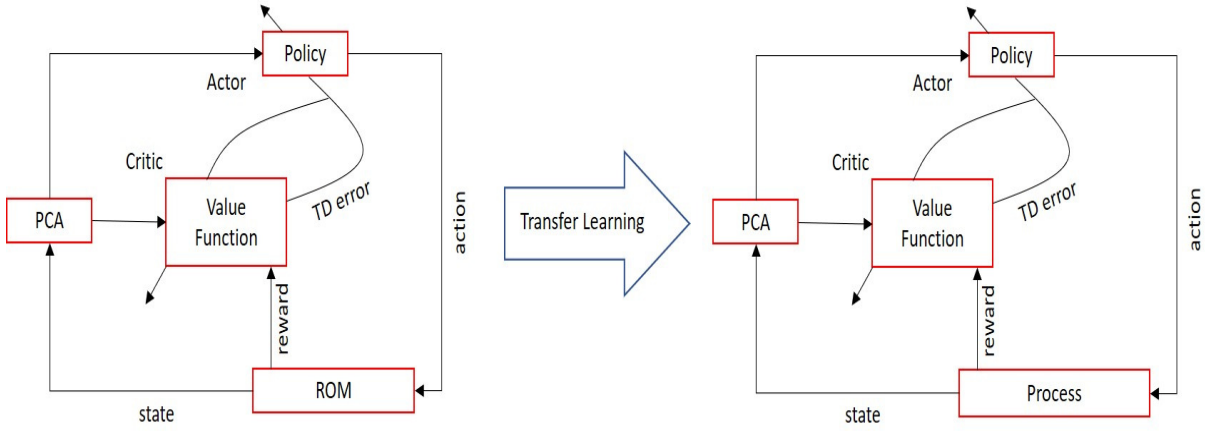


Figure 5.2: A schematic of the proposed learning strategy.

estimates of the first and the second moments of the first-order gradients [214]. The hyperparameters used in our work are given in Table 5.1.

Actor learning rate	0.01
Critic learning rate	0.01
Target network update rate	0.001
Minibatch size	16
RM size	4110
Reward discount factor	0.9
Control action limits	[0, 1]

Table 5.1: Hyperparameter values for the DRL controller

5.1.4.6 ROM for hydraulic fracturing

In our work, we developed a ROM by applying the multivariate output error state space (MOESP) algorithm [215] to regress a linear time-invariant state-space model of the hydraulic fracturing process, which is presented in the following form:

$$x(t_{k+1}) = Ax(t_k) + Bu(t_k) \tag{5.24}$$

$$y(t_k) = Hx(t_k) \tag{5.25}$$

where $y(t_k)$ is the concentration of proppant at 25 different locations, i.e., $[cc_w, cc_9, ..., cc_{216}]$, where cc_{z_i} is the concentration at location z_i with $z_i - z_{i-1} = 0.5\ m$, and $u(t_k) = [cc_w(t_k), ...cc_0(t_k -$

$\theta_{216})]$ is the input to the state space model where cc_w is the concentration at the wellbore, and θ_{z_i} is the input time-delay due to the time required for the proppant to travel from the wellbore to location z_i. In order to obtain the ROM, we obtained training data from the first principles model presented in the previous section by giving input as shown in Figure 5.3.

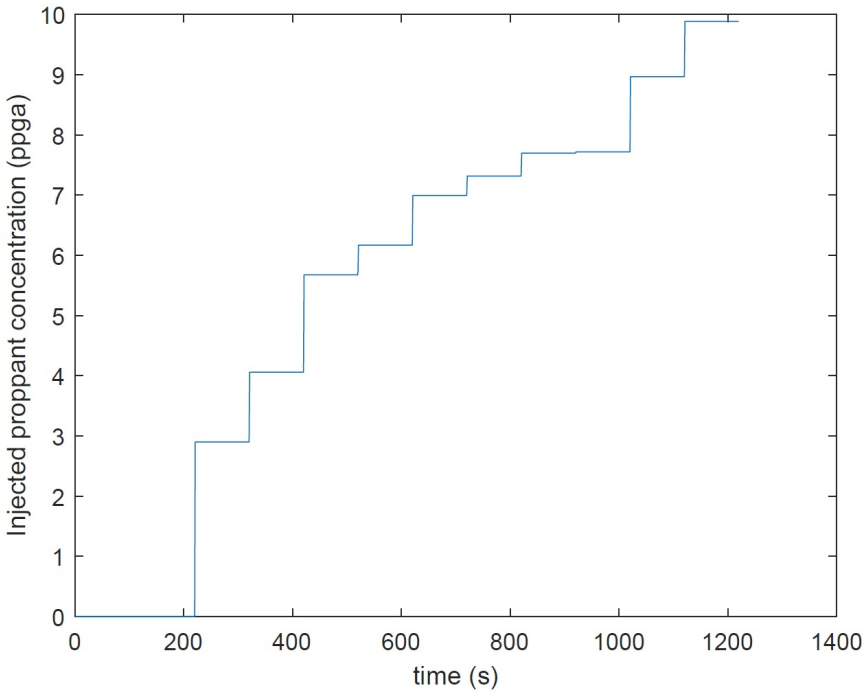


Figure 5.3: Training input for building ROMs.

The locations that are of interest for the purpose of designing the DRL controller are $z = 36, 72, 108, 144, 172, 216$ which are included in the output of the ROM. Figure 5.4 shows the comparison between the predictions from the ROM and from the first principles model at the wellbore and at these 6 locations. It can be observed that the predictions of the proppant concentration at these locations across the fracture are fairly accurate.

5.1.5 DRL controller results

5.1.5.1 Initializing the learning process

To briefly summarize the hydraulic fracturing process, a fracturing fluid along with proppant is injected at high pressures to extend the fracture and to deposit proppant inside the fracture which

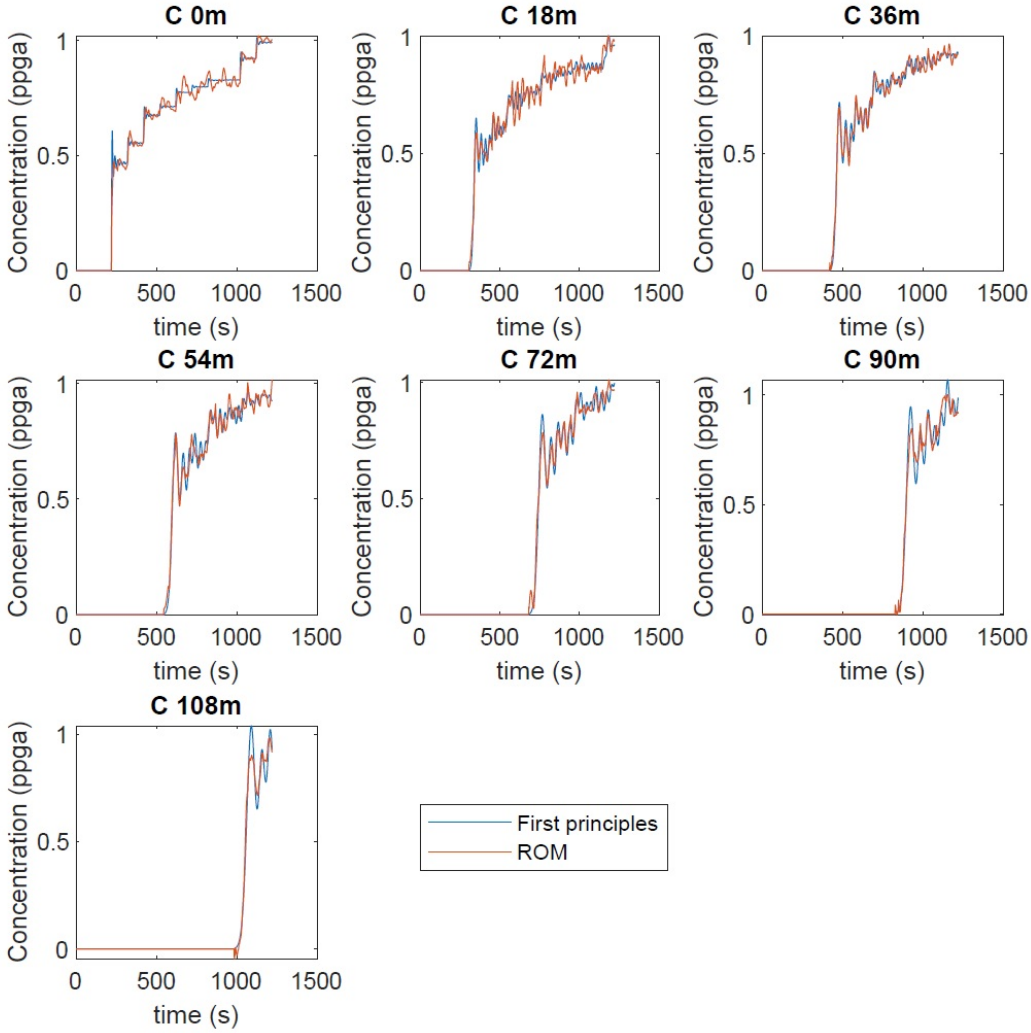


Figure 5.4: Output predictions from the ROMs at the wellbore and 6 other locations.

acts as an artificial medium for the easier extraction of oil and gas. The objective of the DRL controller is to learn a control policy with injected proppant concentration as the manipulated variable and the concentration at 6 locations, i.e., $z = 36, 72, 108, 144, 172, 216$ from the wellbore as the controlled outputs with the objective of obtaining a uniform concentration of 10 $ppga$ at these locations. The limits for the control actions are 0 and 10 $ppga$. The total fracking time considered is 1220 s. During the first 220 s of the injection process, called pad time, no proppant is injected, and thereafter, proppant injection begins. The pad time of 220 s was fixed in order to prevent premature termination of the hydraulic fracturing process, due to the tip screen-out. So the DRL controller learns a control policy from 220 s onwards. The injection process occurs over 10

stages with a constant fracturing fluid rate of $Q = 0.03\ m^3/s$ in order to reach a fracture length of $135\ m$.

For the construction of the RM, we generated simulation data by implementing 411 input profiles on the first-principles model and collected 4110 snapshots of input-output data. PCA was used on the RM data to calculate the dominant PC in order to reduce the RL state during the learning process. In each episode of learning, the set-point for C_i is $10\ ppga$ and for A is $24000\ kgs$ but after normalization these values change to 1. The tolerance for C_i is 0.08 and for A is 0.0417 after normalization. These tolerance values should be selected carefully as stricter tolerances will require longer training times, and laxer tolerances will result in poor performance of the DRL controller. The parameters used in the rewards calculation are shown in Table 5.2.

ω_1	0.1
ω_2	0.9
penalty	0.1

Table 5.2: Hyperparameter values used in rewards calculation

5.1.5.2 First stage of learning

In the first stage of the learning process, the parameters of the DRL controller are initialized as described in the previous section, and the learning process using the ROM was started. The DRL controller implements the control action as predicted by the actor and implements it on the ROM to obtain the outputs. The rewards are calculated using Eq. (5.23), and the tuple (s, a, s', r) is stored in the RM. Then $M = 16$ tuples are randomly selected from the RM and used to update the actor, critic, and the target networks. The learning process is terminated when the following criteria are satisfied: (a) the net reward gained in an episode is 0.75 times the theoretical maximum (obtained from literature); and (b) the total amount of proppant injected is within the tolerance.

The DRL controller reaches the above-mentioned criterion at 603 episodes and in order to track the learning process, the net reward gained in each episode is plotted as shown in Figure 5.5. Initially, since the weights and the biases of the DNNs are randomly initialized, the DRL controller shows poor performance. From episode 17, the controller performs reasonably well as observed in

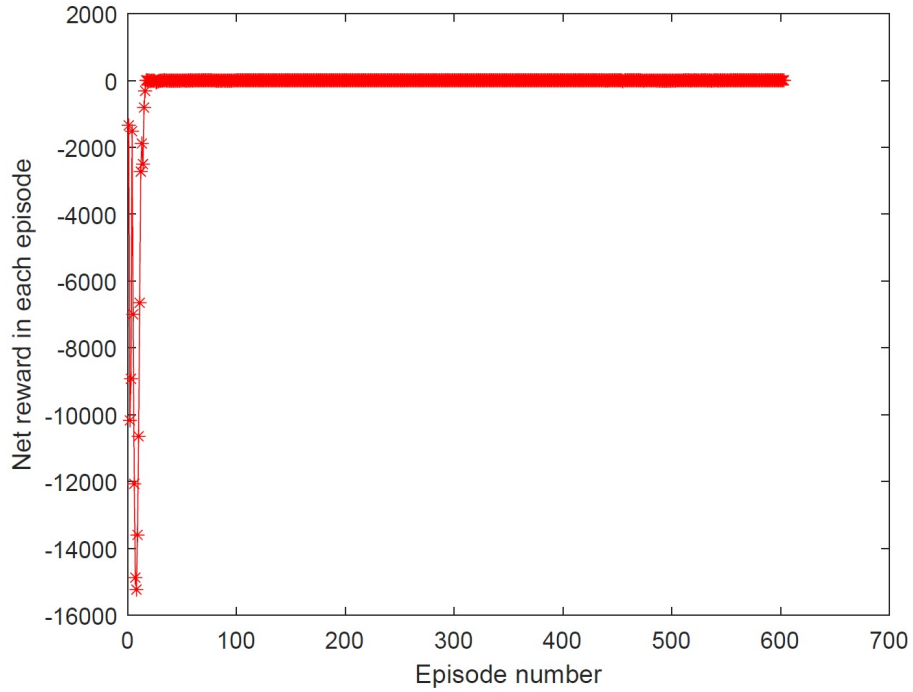


Figure 5.5: Net reward gained in each episode during the ROM learning.

Figure 5.5, but does not meet the criterion for termination until episode 603 wherein it gains a net reward of 3.481.

5.1.5.3 Second stage of learning

In the second stage of the learning process, the weights and the biases are initialized using the parameters obtained from the last episode of learning from the previous stage. The learning process is repeated and terminated when all the states (i.e., the concentrations at 6 locations and the total amount of proppant injected) reach their respective set-points. The DRL controller is able to meet the criteria by episode 724. The learning curve in terms of net rewards per episode for both the stages is shown in Figure 5.6. Initially, the curve undergoes fluctuations as the parameters are randomly initialized in the first stage, and thereafter, the controller performance improves until episode 603 where the criteria for the first stage learning is satisfied. The DRL controller continues to improve even in the second stage until episode 724 wherein the controller reaches the desired control objectives.

Figure 5.7 shows the input profile implemented by the controller in the last episode of learning,

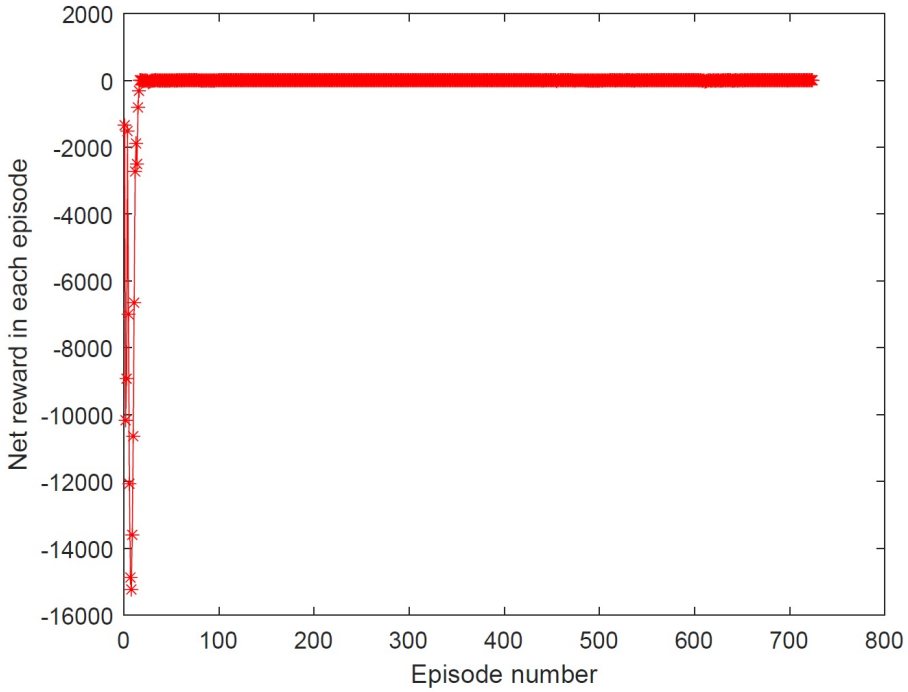


Figure 5.6: Net reward gained in each episode during the entire learning process of the DRL controller. Please note that the learning curve of the second stage continues from Figure 5.5, and corresponds to the episodes between 603 and 724 in this figure.

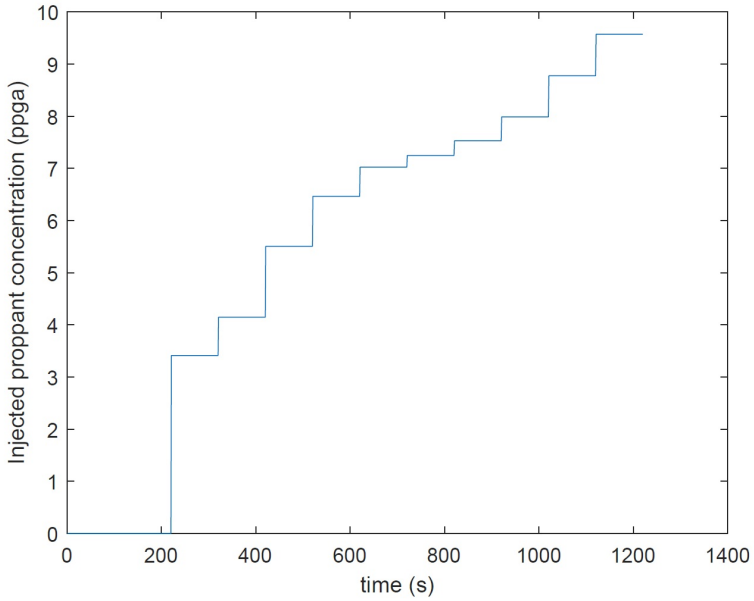


Figure 5.7: The input profile implemented in the last episode.

and Figure 5.8 shows the evolution of the concentrations at the selected locations in the same episode. The input profile obtained is a step-wise increasing profile with the injected proppant concentration values within the specified control limits of $0 - 10$ $ppga$, and a maximum step increase of 3 $ppga$ between two control actions. Additionally, as seen in Figure 5.8, all the states are within their specified tolerance limits from their respective set-points. Hence, learning was terminated at the end of this episode.

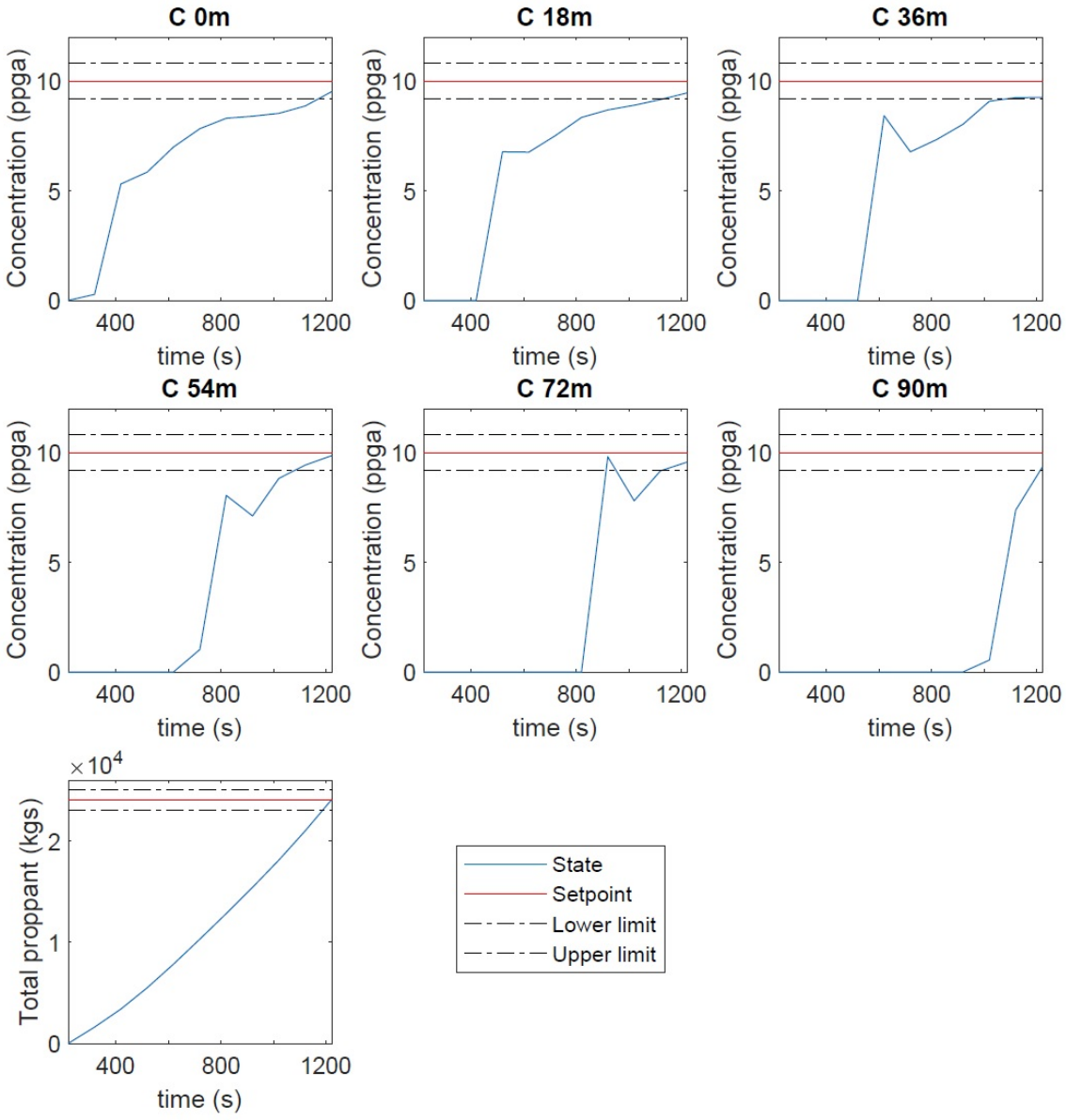


Figure 5.8: Evolution of states in the last episode.

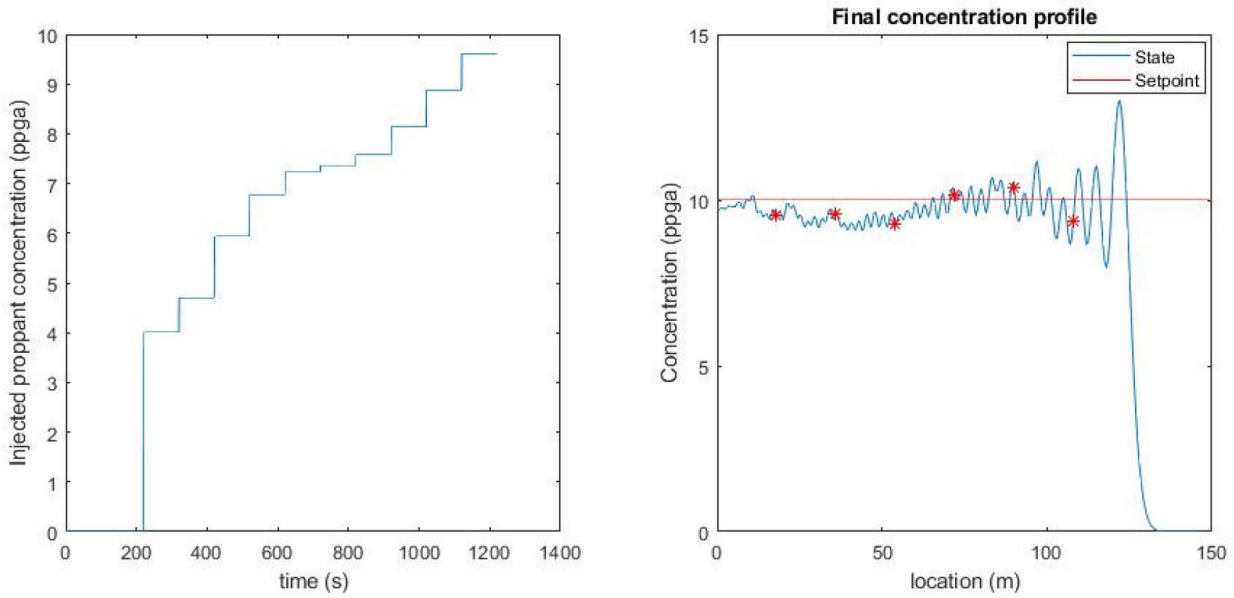


Figure 5.9: The input profile obtained from the DRL controller (left) and the concentration profile at the end of pumping process (right) are presented.

In order to test the DRL controller's performance, we stop the learning process, utilize the actor to predict control actions, and obtain the corresponding outputs. Figure 5.9 shows the inputs predicted by the DRL controller, and the concentration profile across the fracture length at the end of the proppant injection process. The input profile predicted by the controller satisfies the constraints, and the concentrations at the 6 selected locations are within the tolerance limits from the set-point.

6. SUMMARY

In the first part of the work, we developed a framework to systematically enlarge the domain of attraction for the LDMDc technique and showed its efficacy by applying it to the hydraulic fracturing process. In the second part of the work, we developed a novel deep hybrid modeling framework by integrating deep neural networks with a first-principles model. This deep hybrid model was trained using the Levenberg-Marquardt algorithm. The proposed deep hybrid modeling framework was applied to the hydraulic fracturing process to accurately capture the uncertainty in the leak-off rate by the DNN. We showed the superior accuracy of the proposed deep hybrid model over the existing first-principles model. Also, we proved the superior extrapolation properties of the deep hybrid model over a black-box model. In the third part of the work, we showed the effectiveness of the deep hybrid modeling framework in a real-world case study by building a deep hybrid model for a full-scale bio-fermentation process with a volume of over 100,000 gallons. In this work, we developed a three-step method. Firstly, we improved the accuracy of the first-principles model via incorporating mathematical terms in its equations which are based on obtained process knowledge from a literature study. Secondly, we performed a local and global sensitivity analysis to identify sensitive parameters in the improved first-principles model that have considerable influence on its prediction capability. Finally, we developed a deep hybrid model by integrating the improved first-principles model with a DNN which is trained to predict the identified model parameters. We showed that the resulting deep hybrid model is more accurate and robust than the existing first-principles model. In the fourth part of the work, we developed a UDE model for batch production of β-carotene using synthetic data. We showed that the DNN in the UDE model can effectively capture the unknown dynamics. In the fifth part of the work, we utilized the UDE approach to build a hybrid model for lab-scale batch production of β-carotene. We utilized experimental data to train the UDE model and showed the superior accuracy of the UDE model over the existing kinetic model. Also, we showed the superior extrapolation property of the UDE model using another experimental data set. In the sixth part of the work, we

designed a CLBF-MPC controller using a deep hybrid model which simultaneously stabilizes as well as gives guarantees on the DA of the deep hybrid model. We provided theoretical guarantees on the performance of the CLBF-MPC controller, and successfully implemented it on a chemical process example. Finally, we integrated prior knowledge about the hydraulic fracturing process in the design of a data-based DRL controller. Nolte's law was formulated as a constraint in the reward function for the DRL controller, and we showed that the resulting DRL controller was able to quickly achieve convergence towards an optimal control policy to obtain uniform proppant concentration at the end of the proppant injection process.

REFERENCES

[1] T. K. Perkins and L. R. Kern, "Widths of hydraulic fractures," *J. Pet. Technol.*, vol. 13, pp. 937–949, 1961.

[2] I. T. Cameron and K. Hangos, *Process Modelling and Model Analysis*. Academic Press, 2001.

[3] P. Siddhamshetty, S. Yang, and J. S.-I. Kwon, "Modeling of hydraulic fracturing and designing of online pumping schedules to achieve uniform proppant concentration in conventional oil reservoirs," *Comput. Chem. Eng.*, vol. 114, pp. 306 – 317, 2018. FOCAPO/CPC 2017.

[4] S. Yang, P. Siddhamshetty, and J. S. Kwon, "Optimal pumping schedule design to achieve a uniform proppant concentration level in hydraulic fracturing.," *Comput. Chem. Eng.*, vol. 101, no. C, pp. 138–147, 2017.

[5] P. Siddhamshetty, J. S. Kwon, S. Liu, and P. P. Valko, "Feedback control of proppant bank heights during hydraulic fracturing for enhanced productivity in shale formations," *AIChE J.*, vol. 64, pp. 1638–1650, November 2017.

[6] A. Narasingam, P. Siddhamshetty, and J. S. Kwon, "Temporal clustering for order reduction of nonlinear parabolic PDE systems with time-dependent spatial domains: Application to a hydraulic fracturing process," *AIChE J.*, vol. 63, no. 9, pp. 3818–3831, 2017.

[7] P. J. Schmid, "Dynamic mode decomposition of numerical and experimental data.," *J. Fluid Mech.*, vol. 656, pp. 5–28, 2010.

[8] J. L. Proctor, S. L. Brunton, and J. N. Kutz, "Dynamic mode decomposition with control.," *SIAM J. Appl. Dyn. Syst.*, vol. 15, no. 1, pp. 142–161, 2016.

[9] A. Narasingam and J. S. Kwon, "Development of local dynamic mode decomposition with control: Application to model predictive control of hydraulic fracturing," *Comput. Chem. Eng.*, vol. 106, pp. 501–511, 2017.

[10] A. Bao, E. Gildin, A. Narasingam, and J. S. Kwon, "Data-driven model reduction for coupled flow and geomechanics based on dmd methods," *Fluids*, vol. 4, no. 3, 2019.

[11] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley, "A data-driven approximation of the Koopman operator: Extending dynamic mode decomposition.," *J. Nonlinear Sci.*, vol. 25, no. 6, pp. 1307–1346, 2015.

[12] A. Narasingam and J. S.-I. Kwon, "Koopman lyapunov-based model predictive control of nonlinear chemical process systems," *AIChE Journal*, vol. 65, no. 11, p. e16743, 2019.

[13] A. Narasingam, S. H. Son, and J. S.-I. Kwon, "Data-driven feedback stabilisation of nonlinear systems: Koopman-based model predictive control," *International Journal of Control*, vol. 0, no. 0, pp. 1–12, 2022.

[14] S. H. Son, A. Narasingam, and J. S.-I. Kwon, "Handling plant-model mismatch in koopman lyapunov-based model predictive control via offset-free control framework," *arXiv:2010.07239*, 2020.

[15] A. Narasingam and J. S.-I. Kwon, "Application of Koopman operator for model-based control of fracture propagation and proppant transport in hydraulic fracturing operation," *Journal of Process Control*, vol. 91, pp. 25–36, 2020.

[16] B. Bhadriraju, A. Narasingam, and J. S.-I. Kwon, "Machine learning-based adaptive model identification of systems: Application to a chemical process," *Chem. Eng. Res. Des.*, vol. 152, pp. 372–383, 2019.

[17] B. Bhadriraju, M. S. F. Bangi, A. Narasingam, and J. S. Kwon, "Operable adaptive sparse identification of systems: Application to chemical processes," *AIChE J.*, vol. 66, no. 11, p. e16980, 2020.

[18] B. Bhadriraju, J. S.-I. Kwon, and F. Khan, "OASIS-P: Operable Adaptive Sparse Identification of Systems for fault prognosis of chemical processes," *Journal of Process Control*, vol. 107, pp. 114–126, 2021.

[19] B. Bhadriraju, J. S.-I. Kwon, and F. Khan, "Risk-based fault prediction of chemical processes using operable adaptive sparse identification of systems (OASIS)," *Comput. Chem. Eng.*, vol. 152, p. 107378, 2021.

[20] M. L. Thompson and M. A. Kramer, "Modeling chemical processes using prior knowledge and neural networks," *AIChE. J.*, vol. 40, no. 8, pp. 1328–1340, 1994.

[21] D. C. Psichogios and L. H. Ungar, "A hybrid neural network-first principles approach to process modeling," *AIChE. J.*, vol. 38, no. 10, pp. 1499–1511, 1992.

[22] M. A. Kramer, M. L. Thompson, and P. M. Bhagat, "Embedding theoretical models in neural networks," in *1992 American Control Conference*, pp. 475–479, 1992.

[23] T. A. Johansen and B. A. Foss, "Representing and learning unmodeled dynamics with neural network memories," in *1992 American Control Conference*, pp. 3037–3043, 1992.

[24] H. Su, N. Bhat, P. A. Minderman, and T. J. McAvoy, "Integrating neural networks with first principles models for dynamic modeling," in *IFAC Symposium on Dynamics and Control of Chemical Reactors, Distillation Columns and Batch Processes*, pp. 327–332, 1993.

[25] T. Bohlin and S. F. Graebe, "Issues in nonlinear stochastic grey box identification," *International Journal of Adaptive Control and Signal Processing*, vol. 9, no. 6, pp. 465–490, 1995.

[26] S. B. Jorgensen and K. M. Hangos, "Grey box modelling for control: Qualitative models as a unifying framework," *International Journal of Adaptive Control and Signal Processing*, vol. 9, no. 6, pp. 547–562, 1995.

[27] H. J. A. F. Tulleken, "Grey-box modelling and identification using physical knowledge and bayesian techniques," *Automatica*, vol. 29, no. 2, pp. 285 – 308, 1993.

[28] G. Zahedi, A. Lohi, and K. Mahdi, "Hybrid modeling of ethylene to ethylene oxide heterogeneous reactor," *Fuel Processing Technology*, vol. 92, no. 9, pp. 1725 – 1732, 2011.

[29] S. Gupta, P.-H. Liu, S. A. Svoronos, R. Sharma, N. A. Abdek-Khalek, Y. Cheng, and H. El-Shall, "Hybrid first-principles/neural networks model for column flotation," *AIChE J.*, vol. 45, no. 3, pp. 557–566, 1999.

[30] E. Molga and R. Cherbański, "Hybrid first-principle-neural-network approach to modelling of the liquid-liquid reacting system," *Chem. Eng. Sci.*, vol. 54, no. 13, pp. 2467 – 2473, 1999.

[31] H. Qi, X.-G. Zhou, L.-H. Liu, and W.-K. Yuan, "A hybrid neural network-first principles model for fixed-bed reactor," *Chem. Eng. Sci.*, vol. 54, no. 13, pp. 2521 – 2526, 1999.

[32] A. Y. D. Tsen, S. S. Jang, D. S. H. Wong, and B. Joseph, "Predictive control of quality in batch polymerization using hybrid ann models," *AIChE J.*, vol. 45, no. 2, pp. 455–465, 1996.

[33] B. Fiedler and A. Schuppert, "Local identification of scalar hybrid models with tree structure," *IMA Journal of Applied Mathematics*, vol. 73, no. 3, pp. 449 – 476, 2008.

[34] P. Lauret, H. Boyer, and J. Gatina, "Hybrid modelling of a sugar boiling process," *Control Engineering Practice*, vol. 8, no. 3, pp. 299 – 310, 2000.

[35] P. Georgieva and S. de Azevedo, *Computational intelligence techniques for bioprocess modelling, supervision and control, Volume 218*. Berlin/Heidelberg:Springer, 2009.

[36] M. Reuter, J. V. Deventer, and T. V. D. Walt, "A generalized neural-net kinetic rate equation," *Chem. Eng. Sci.*, vol. 48, no. 7, pp. 1281 – 1297, 1993.

[37] G. Hu, Z. Mao, D. He, and F. Yang, "Hybrid modeling for the prediction of leaching rate in leaching process based on negative correlation learning bagging ensemble algorithm," *Comput. Chem. Eng.*, vol. 35, no. 12, pp. 2611 – 2617, 2011.

[38] R. da Jia, Z. zhong Mao, Y. qing Chang, and L. ping Zhao, "Soft-sensor for copper extraction process in cobalt hydrometallurgy based on adaptive hybrid model," *Chem. Eng. Res. Des.*, vol. 89, no. 6, pp. 722 – 728, 2011.

[39] A. Safavi, A. Nooraii, and J. Romagnoli, "A hybrid model formulation for a distillation column and the on-line optimisation study," *Journal of Process Control*, vol. 9, no. 2, pp. 125 – 134, 1999.

[40] V. Mahalec and Y. Sanchez, "Inferential monitoring and optimization of crude separation units via hybrid models," *Comput. Chem. Eng.*, vol. 45, pp. 15 – 26, 2012.

[41] F. A. Cubillos and G. Acuña, "Adaptive control using a grey box neural model: An experimental application," in *Advances in Neural Networks – ISNN 2007* (D. Liu, S. Fei, Z.-G. Hou, H. Zhang, and C. Sun, eds.), (Berlin, Heidelberg), pp. 311–318, Springer, 2007.

[42] M. R. Arahal, C. M. Cirre, and M. Berenguel, "Serial grey-box model of a stratified thermal tank for hierarchical control of a solar plant," *Solar Energy*, vol. 82, no. 5, pp. 441–451, 2008.

[43] C. A. O. Nascimento, R. Giudici, and N. Scherbakoff, "Modeling of industrial nylon-6,6 polymerization process in a twin-screw extruder reactor. ii. neural networks and hybrid models," *Journal of Applied Polymer Science*, vol. 72, no. 7, pp. 905–912, 1999.

[44] H. C. Aguiar and R. M. Filho, "Neural network and hybrid model: a discussion about different modeling techniques to predict pulping degree with industrial data," *Chem. Eng. Sci.*, vol. 56, no. 2, pp. 565 – 570, 2001.

[45] P. Kumar Akkisetty, U. Lee, G. V. Reklaitis, and V. Venkatasubramanian, "Population balance model-based hybrid neural network for a pharmaceutical milling process," *Journal of Pharmaceutical Innovation*, vol. 5, no. 4, pp. 161–168, 2010.

[46] J. Schubert, R. Simutis, M. Dors, I. Havlik, and A. Lübbert, "Bioprocess optimization and control: Application of hybrid modelling," *Journal of Biotechnology*, vol. 35, no. 1, pp. 51 – 68, 1994.

[47] J. Schubert, R. Simutis, M. Dors, I. Havlik, and A. Luebbert, "Hybrid modelling of yeast production processes âĂŞ combination of a priori knowledge on different levels of sophistication," *Chemical Engineering & Technology*, vol. 17, no. 1, pp. 10–20, 1994.

[48] R. Eslamloueyan and P. Setoodeh, "Optimization of fed-batch recombinant yeast fermentation for ethanol production using a reduced dynamic flux balance model based on artificial neural networks," *Chem. Eng. Comm.*, vol. 198, no. 11, pp. 1309–1338, 2011.

[49] H. Preusting, J. Noordover, R. Simutis, and A. Lübbert, "The use of hybrid modelling for the optimization of the penicillin fermentation process," *CHIMIA International Journal for Chemistry*, vol. 50, no. 9, pp. 416–417, 1996.

[50] X. Wang, J. Chen, C. Liu, and F. Pan, "Hybrid modeling of penicillin fermentation process based on least square support vector machine," *Chem. Eng. Res. Des.*, vol. 88, no. 4, pp. 415 – 420, 2010.

[51] R. Simutis and A. Lübbert, "Exploratory analysis of bioprocesses using artificial neural network-based methods," *AIChE J.*, vol. 13, no. 4, pp. 479–487, 1997.

[52] S. Gnoth, M. Jenzsch, R. Simutis, and A. Lübbert, "Product formation kinetics in genetically modified e. coli bacteria: inclusion body formation," *Bioprocess and Biosystems Engineering*, vol. 31, pp. 41–46, Jan 2008.

[53] M. Dors, R. Simutis, and A. Lübbert, "Advanced supervision of mammalian cell cultures using hybrid process models," in *Computer Applications in Biotechnology*, IFAC Postprint Volume, pp. 72 – 77, Amsterdam: Pergamon, 1995.

[54] A. P. Teixeira, C. Alves, P. M. Alves, M. J. T. Corrondo, and R. Oliveira, "Hybrid elementary flux analysis/nonparametric modeling: application for bioprocess control," *BMC Bioinformatics*, vol. 8, no. 30, 2007.

[55] N. Carinhas, V. Bernal, A. P. Teixeira, M. J. T. Carrondo, P. M. Alves, and R. Oliveira, "Hybrid metabolic flux analysis: combining stoichiometric and statistical constraints to model the formation of complex recombinant products," *BMC Systems Biology*, vol. 5, no. 34, 2011.

[56] P. C. Fu and J. P. Barford, "Integration of mathematical modelling and knowledge-based systems for simulations of biochemical processes," *Expert Systems with Applications*, vol. 9, no. 3, pp. 295 – 307, 1995.

[57] P. C. Fu and J. P. Barford, "A hybrid neural network-first principles approach for modelling of cell metabolism," *Comput. Chem. Eng.*, vol. 20, no. 6, pp. 951–958, 1996.

[58] M. von Stosch, R. Oliveira, J. Peres, and S. F. de Azevedo, "Hybrid semi-parametric modeling in process systems engineering: Past, present and future," *Comput. Chem. Eng.*, vol. 60, pp. 86 – 101, 2014.

[59] J. Sansana, M. N. Joswiak, I. Castillo, Z. Wang, R. Rendall, L. H. Chiang, and M. S. Reis, "Recent trends on hybrid modeling for industry 4.0," *Comput. Chem. Eng.*, vol. 151, p. 107365, 2021.

[60] V. Venkatasubramanian, "The promise of artificial intelligence in chemical engineering: Is it here, finally?," *AIChE J.*, vol. 65, no. 2, pp. 466–478, 2018.

[61] M. Stinchcombe and H. White, "Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions," pp. 613–617, 1989.

[62] B. R. Noack, K. Afanasiev, M. Morzyński, G. Tandmor, and F. Thiele, "A hierarchy of low-dimensional models for the transient and post-transient cylinder wake.," *J. Fluid Mech.*, vol. 497, pp. 335–363, 2003.

[63] M. H. Sahraei, M. A. Duchesne, P. G. Boisvert, R. W. Hughes, and L. A. Ricardez-Sandoval, "Dynamic reduced order modeling of an entrained-flow slagging gasifier using a new recirculation ratio correlation," *Fuel*, vol. 196, pp. 520 – 531, 2017.

[64] M. H. Sahraei, M. A. Duchesne, P. G. Boisvert, R. W. Hughes, and L. A. Ricardez-Sandoval, "Reduced-order modeling of a commercial-scale gasifier using a multielement injector feed system," *Ind. Eng. Chem. Res.*, vol. 56, no. 25, pp. 7285–7300, 2017.

[65] M. H. Sahraei, M. A. Duchesne, R. Yandon, A. Majeski, R. W. Hughes, and L. A. Ricardez-Sandoval, "Reduced order modeling of a short-residence time gasifier," *Fuel*, vol. 161, pp. 222 – 232, 2015.

[66] P. Holmes, J. L. Lumley, and G. Berkooz, *Turbulence, Coherent Structures, Dynamical Systems and Symmetry*. New York: Cambridge University Press, 1996.

[67] A. Armaou and P. D. Christofides, "Finite-dimensional control of nonlinear parabolic PDE systems with time-dependent spatial domains using empirical eigenfunctions.," *Int. J. Appl. Math. Comput. Sci.*, vol. 11, no. 2, pp. 287–317, 2001.

[68] G. Berkooz, P. Holmes, and J. Lumley, "The proper orthogonal decomposition in the analysis of turbulent flows.," *Ann. Rev. Fluid. Mech.*, vol. 25, pp. 539–575, 1993.

[69] E. Christensen, M. Brons, and J. Sorensen, "Evaluation of proper orthogonal decomposition based decomposition techniques applied to parameter-dependent nonturbulent flows.," *SIAM J. Sci. Comput.*, vol. 21, pp. 1419–1434, 2000.

[70] H. M. Park and M. W. Lee, "An efficient method of solving the navier-stokes equations for flow control.," *Int. J. Numer. Methods Eng.*, vol. 41, pp. 1133–1151, 1998.

[71] S. Ravindran, "Proper orthogonal decomposition in optimal control of fluids.," *Int. J. Numer. Methods Fluids*, vol. 34, pp. 425–448, 2000.

[72] H. S. Sidhu, A. Narasingam, P. Siddhamshetty, and J. S. Kwon, "Model order reduction of nonlinear parabolic pde systems with moving boundaries using sparse proper orthogonal decomposition: application to hydraulic fracturing," *Comput. Chem. Eng.*, vol. 112, pp. 92–100, 2018.

[73] S. Pitchaiah and A. Armaou, "Output feedback control of distributed parameter systems using adaptive proper orthogonal decomposition," *Ind. Eng. Chem. Res.*, vol. 49, no. 21, pp. 10496–10509, 2010.

[74] B. R. Noack, M. Schlegel, B. Ahlborn, B. Mutschke, M. Morzyński, P. Comte, and G. Tadmor, "A finite-time thermodynamics formalism for unsteady flows.," *J. Non-Equilib. Thermodyn.*, vol. 33, pp. 103–148, 2008.

[75] P. J. Schmid and J. Sesterhenn, "Dynamic mode decomposition of numerical and experimental data.," *In Bull. Amer. Phys. Soc. 61st APS Meeting, San Antonio, Texas.*, vol. 208, 2008.

[76] M. Ghommem, V. M. Calo, and Y. Efendiev, "Mode decomposition methods for flows in high-contrast porous media. A Global approach," *J. Comput. Phys.*, vol. 257, no. A, pp. 400–413, 2014.

[77] C. W. Rowley, I. Mezić, S. Bagheri, P. Schlatter, and D. S. Henningson, "Spectral analysis of nonlinear flows.," *J. Fluid Mech.*, vol. 641, pp. 115–127, 2009.

[78] I. Mezić, "Analysis of fluid flows via spectral properties of the Koopman operator.," *Ann. Rev. Fluid Mech.*, vol. 45, pp. 357–378, 2013.

[79] S. Bagheri, "Koopman-mode decomposition of the cylinder wake.," *J. Fluid Mech.*, vol. 726, pp. 596–623, 2013.

[80] P. J. Schmid, L. Li, M. P. Juniper, and O. Pust, "Applications of the dynamic mode decomposition.," *Theor. Comput. Fluid Dyn.*, vol. 25, pp. 249–259, 2011.

[81] A. Seena and H. J. Sung, "Dynamic mode decomposition of turbulent cavity flows for self-sustained oscillations.," *Int. J. Heat Fluid Fl.*, vol. 32, pp. 1098–1110, 2011.

[82] Y. Mizuno, D. Duke, C. Atkinson, and J. Soria, "Investigation of wall-bounded turbulent flow using dynamic mode decomposition.," *J. Phys. Conf. Ser.*, vol. 318, p. 042040, 2011.

[83] T. W. Muld, G. Efraimsson, and D. S. Henningson, "Flow structures around high-speed train extracted using proper orthogonal decomposition and dynamic mode decomposition," *Comput. Struct.*, vol. 57, pp. 87–97, 2012.

[84] P. J. Schmid, "Dynamic mode decomposition of experimental data.," *In 8th International Symposium on Particle Image Velocimetry - PIV09*, 2009.

[85] P. J. Schmid, K. E. Meyer, and O. Pust, "Dynamic mode decomposition and proper orthogonal decomposition of flow in a lid-driven cylindrical cavity.," *In 8th International Symposium on Particle Image Velocimetry - PIV09*, 2009.

[86] P. J. Schmid, "Application of the dynamic mode decomposition to experimental data.," *Exp. Fluids.*, vol. 50, pp. 1123–1130, 2011.

[87] C. Pan, D. Yu, and J. Wang, "Dynamical mode decomposition of gurney flap wake flow.," *Theor. Appl. Mech. Lett.*, vol. 1, p. 012002, 2011.

[88] O. Semeraro, G. Bellani, and F. Lundell, "Analysis of time-resolved PIV measurements of a confined turbulent jet using POD and Koopman modes.," *Exp. Fluids*, vol. 53, no. 5, pp. 1203–1220, 2012.

[89] F. Lusseyran, F. Gueniat, J. Basley, C. L. Douay, L. R. Pastur, T. M. Faure, and P. J. Schmid, "Flow coherent structures and frequency signature: Application of the dynamic modes decomposition to open cavity flow," *J. Phys. Conf. Ser.*, vol. 318, p. 042036, 2011.

[90] D. Duke, J. Soria, and D. Honnery, "An error analysis of the dynamic mode decomposition.," *Exp. Fluids*, vol. 52, pp. 529–542, 2012.

[91] J. H. Tu and C. W. Rowley, "An improved algorithm for balanced pod through an analytic treatment of impulse response tails," *J. Comput. Phys.*, vol. 231, no. 16, p. 5317âĂŞ5333, 2012.

[92] B. A. Belson, J. H. Tu, and C. W. Rowley, "Algorithm 945: modredâĂŤa parallelized model reduction library.," *ACM Trans. Math. Softw*, vol. 40, no. 4, 2014.

[93] M. R. Jovanović, P. J. Schmid, and J. W. Nichols, "Sparsity-promoting dynamic mode decomposition.," *Phys. Fluids*, vol. 26, p. 024103, 2014.

[94] K. K. Chen, J. H. Tu, and C. W. Rowley, "Variants of dynamic mode decomposition: Connections between Koopman and Fourier analyses.," *J. Nonlinear Sci.*, vol. 22, no. 6, pp. 897–915, 2012.

[95] P. J. Goulart, A. Wynn, and D. S. Pearson, "Optimal mode decomposition for high dimensional systems.," *Proceedings of the 51st IEEE Conference on Decision and Control. Maui, Hawaii.*, vol. 4965-4970, 2012.

[96] A. Wynn, D. S. Pearson, B. Ganapathisubramani, and P. J. Goulart, "Optimal mode decomposition for unsteady flows.," *J. Fluid Mech.*, vol. 733, pp. 473–503, 2013.

[97] M. S. F. Bangi, A. Narasingam, P. Siddhamshetty, and J. S.-I. Kwon, "Enlarging the domain of attraction of the local dynamic mode decomposition with control technique: Application to hydraulic fracturing," *Ind. Eng. Chem. Res.*, vol. 58, pp. 5588–5601, Apr. 2019.

[98] R. Nordgren, "Propagation of a vertical hydraulic fracture," *Soc. Petrol. Eng. J.*, vol. 12, pp. 306–314, 1972.

[99] G. C. Howard and C. R. Fast, "Optimum fluid characteristics for fracture extension," *Drill. prod. pract.*, vol. 24, pp. 261–270, 1957.

[100] M. J. Economides and K. G. Nolte, *Reservoir stimulation*. Chichester: Wiley, 2000.

[101] Q. Gu and K. A. Hoo, "Evaluating the performance of a fracturing treatment design.," *Ind. Eng. Chem. Res.*, vol. 53, no. 25, pp. 10491–10503, 2014.

[102] Q. Gu and K. A. Hoo, "Model-based closed-loop control of the hydraulic fracturing process.," *Ind. Eng. Chem. Res.*, vol. 54, no. 5, pp. 1585–1594, 2015.

[103] J. Adachi, E. Siebrits, A. Pierce, and J. Desroches, "Computer simulation of hydraulic fractures.," *Int. J. Rock Mech. Min. Sci.*, vol. 44, pp. 739–757, 2007.

[104] A. Daneshy, "Numerical solution of sand transport in hydraulic fracturing.," *J. Pet. Technol.*, vol. 30, pp. 132–140, 1978.

[105] R. Barree and M. Conway, "Experimental and numerical modeling of convective proppant transport.," *J. Pet. Technol.*, vol. 47, pp. 216–222, 1995.

[106] E. J. Novotny, "Proppant transport," in *Proceedings of the 52nd SPE Annual Technical Conference and Exhibition*, vol. (SPE 6813), (Denver, CO), 1977.

[107] P. Siddhamshetty, *Modeling of Hydraulic Fracturing and Design of Online Optimal Pumping Schedule for Enhanced Productivity in Shale Formations*. PhD thesis, Texas A&M University, 2020.

[108] O. Delalleau and Y. Bengio, "Shallow vs. deep sum-product networks," in *Advances in Neural Information Processing Systems 24* (J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, eds.), pp. 666–674, Curran Associates, Inc, 2011.

[109] S. Liang and R. Srikant, "Why deep neural networks for function approximation?," in *5th International Conference on Learning Representations*, 2017.

[110] R. Eldan and O. Shamir, "The power of depth for feedforward neural networks," in *Proceedings of the 29th Annual Conference on Learning Theory* (V. Feldman, A. Rakhlin, and O. Shamir, eds.), vol. 49 of *Proceedings of Machine Learning Research*, (Columbia University, New York, USA), pp. 907–940, PMLR, 23–26 Jun 2016.

[111] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 2924–2932, Curran Associates, Inc., 2014.

[112] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[113] P. J. Werbos, "Backpropagation: past and future," *PSecond International Conference on Neural Network*, vol. 1, pp. 343–353, 1988.

[114] M. R. Osborne, "Fisher's method of scoring," *International Statistic Review*, vol. 86, pp. 271–286, 1992.

[115] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly of Applied Mathematics*, vol. 2, no. 2, pp. 164–168, 1944.

[116] D. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *SIAM J. Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[117] L. Mears, S. M. Stocks, M. O. Albaek, G. Sin, and K. V. Gernaey, "Mechanistic fermentation models for process design, monitoring, and control.," *Trends in Biotechnology*, vol. 35, no. 10, pp. 914–924, 2017.

[118] A. Narasingam and J. S.-I. Kwon, "Data-driven identification of interpretable reduced-order models using sparse regression," *Comput. Chem. Eng.*, vol. 119, no. 2, pp. 101 – 111, 2018.

[119] D. Beluhan and S. Beluhan, "Hybrid modeling approach to on-line estimation of yeast biomass concentration in industrial bioreactor," *Biotechnology Letters*, vol. 22, pp. 631–635, 2000.

[120] R. G. Silva, A. J. Cruz, C. O. Hokka, R. L. Giordano, and R. C. Giordano, "A hybrid neural network algorithm for on-line state inference that accounts for differences in inoculum of cephalosporium acremonium in fed-batch fermentors," *Applied Biochemistry and Biotechnology*, vol. 91-93, pp. 341–352, 2001.

[121] M. Ignova, G. C. Paul, C. A. Kent, C. R. Thomas, G. A. Montague, J. Glassey, and A. C. Ward, "Hybrid modelling for on-line penicillin fermentation optimisation," *IFAC Proceedings Volumes*, vol. 34, no. 1, pp. 395–400, 2002.

[122] S. O. Laursen, D. Webb, and W. F. Ramirez, "Dynamic hybrid neural network model of an industrial fed-batch fermentation process to produce foreign protein," *Comput. Chem. Eng.*, vol. 31, no. 3, pp. 163–170, 2007.

[123] R. Eldan and O. Shamir, "The power of depth for feedforward neural networks," *Proceedings of the Twenty-Ninth Annual Conference on Learning Theory*, vol. 49, pp. 907–940, 2016.

[124] P. Shah, M. Z. Sheriff, M. S. F. Bangi, C. Kravaris, J. S.-I. Kwon, C. Botre, and J. Hirota, "Deep neural network-based hybrid modeling and experimental validation for an industry-scale fermentation process: Identification of time-varying dependencies among parameters," *Chem. Eng. J.*, vol. 441, p. 135643, 2022.

[125] M. C. Ordonez, J. P. Raftery, T. Jaladi, X. Chen, K. Kao, and M. N. Karim, "Modeling of batch kinetics of aerobic carotenoid production using saccharomyces cerevisiae," *Biochemical Engineering Journal*, vol. 114, pp. 226–236, 2016.

[126] Z. Duan, T. Wilms, P. Neubauer, C. Kravaris, and M. N. C. Bournazou, "Model reduction of aerobic bioprocess models for efficient simulation," *Chem. Eng. Sci.*, vol. 217, p. 115512, 2020.

[127] R. Edge, D. J. McGarvey, and T. G. Truscott, "The carotenoids as anti-oxidants-a review," *J. Photochem. Photobiol.*, vol. 41, no. 3, pp. 189–200, 1997.

[128] P. J. Hulshof, T. Kosmeijer-Schuil, C. E. West, and P. C. Hollman, "Quick screening of maize kernels for provitamin a content," *Journal of Food Composition and Analysis*, vol. 20, no. 8, pp. 655–661, 2007.

[129] P. Polazza and N. Krinsky, "Antioxidant effects of carotenoids in vivo and in vitro: an overview," *Methods Enzymol.*, vol. 213, pp. 403–420, 1992.

[130] G. Van Popel and R. A. Goldbohm, "Epidemiologic evidence for beta-carotene and cancer prevention," *Am. J. Clin. Nutr.*, vol. 62, pp. 291–296, 1995.

[131] M. C. OrdoÃśez, J. P. Raftery, T. Jaladi, X. Chen, K. Kao, and M. N. Karim, "Modelling of batch kinetics of aerobic carotenoid production using saccharomyces cerevisiae," *Biochemical Engineering Journal*, vol. 114, pp. 226–236, 2016.

[132] L. M. S. M. Stocks, M. O. Albaek, G. Sin, and K. V. Gernaey, "Mechanistic fermentationmodels for process design, monitoring, and control," *Trends in Biotechnology*, vol. 35, no. 10, pp. 914–924, 2017.

[133] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," (Montreal, Canada), pp. 6571–6583, 2018.

[134] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, and A. Ramadhan, "Universal differential equations for scientific machine learning," *arXiv preprint arXiv:2001.04385*, 2020.

[135] C. Rackauckas and Q. Nie, "Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in Julia," *Journal of Open Research Software*, vol. 5, no. 1, 2017.

[136] J. Revels, M. Lubin, and T. Papamarkou, "Forward-mode automatic differentiation in Julia," *arXiv:1607.07892 [cs.MS]*, 2016.

[137] C. Rackauckas, M. Innes, Y. Ma, J. Bettencourt, L. White, and V. Dixit, "Diffeqflux.jl - A julia library for neural differential equations," *CoRR*, 2019.

[138] M. S. F. Bangi, K. Kao, and J. S.-I. Kwon, "Physics-informed neural networks for hybrid modeling of lab-scale batch fermentation for beta-carotene production using saccharomyces cerevisiae," *Chem. Eng. Res. Des.*, vol. 179, pp. 415–423, 2022.

[139] H. Nelis and A. De Leenheer, "Microbial sources of carotenoid pigments used in foods and feed," *J. Appl. Bacteriol.*, vol. 70, no. 3, pp. 181–191, 1991.

[140] R. Ausich, "Commercial opportunities for carotenoid production by biotechnology," *Pure Appl. Chem.*, vol. 69, no. 10, pp. 2169–2174, 1997.

[141] W. S. E. Johnson, "Microbial carotenoids," *Adv. Biochem. Eng. Biotechnol.*, vol. 53, pp. 119–178, 1995.

[142] P. Lee and C. Schmidt-Dannert, "Metabolic engineering towards biotechnological production of carotenoids in microorganisms," *Appl. Microbiol. Biotechnol.*, vol. 60, pp. 1–11, 2002.

[143] G. Fregova and D. Beshkova, "Carotenoids from rhodotorula and phaffia: yeast of biotechnological importance," *J. Ind. Microbiol. Biotechnol.*, vol. 36, no. 2, pp. 163–180, 2009.

[144] P. B. P. Vachali, P. Bhosale, "Microbial carotenoids from fungi," *Methods Mol. Biol.*, vol. 898, pp. 41–59, 2012.

[145] Z. Ge, "Review on data-driven modeling and monitoring for plant-wide industrial processes," *Chemometrics and Intelligent Laboratory Systems*, vol. 171, pp. 16–25, 2017.

[146] F. J. MontÃąns, F. Chinesta, R. GÃşmez-Bombarelli, and J. N. Kutz, "Data-driven modeling and learning in science and engineering," *Comptes Rendus MÃl'canique*, vol. 347, no. 11, pp. 845–855, 2019. Data-Based Engineering Science and Technology.

[147] C. Fan, D. Yan, F. Xiao, A. Li, J. An, and X. Kang, "Advanced data analytics for enhancing building performances: From data-driven to big data-driven approaches," *Build. Simul.*, vol. 14, pp. 3–24, 2021.

[148] E. N. Pistikopoulos, A. Barbosa-Povoa, J. H. Lee, R. Misener, A. Mitsos, G. V. Reklaitis, V. Venkatasubramanian, F. You, and R. Gani, "Process systems engineering - the generation next?," *Comput. Chem. Eng.*, vol. 147, p. 107252, 2021.

[149] S. Spielberg, A. Tulsyan, N. P. Lawrence, P. D. Loewen, and R. B. Gopaluni, "Toward self-driving processes: A deep reinforcement learning approach to control," *AIChE. J.*, vol. 65, no. 10, p. e16689, 2019.

[150] M. S. F. Bangi and J. S. Kwon, "Deep reinforcement learning control of hydraulic fracturing," *Comput. Chem. Eng.*, vol. 154, p. 107489, 2021.

[151] M. S. F. Bangi and J. S.-I. Kwon, "Deep hybrid modeling of chemical process: Application to hydraulic fracturing," *Comput. Chem. Eng.*, vol. 134, p. 106696, 2020.

[152] D. Lee, A. Jayaraman, and J. S. Kwon, "Development of a hybrid model for a partially known intracellular signaling pathway through correction term estimation and neural network modeling," *PLoS Comput. Biol.*, vol. 16, no. 12, p. e1008472, 2020.

[153] D. Lee, A. Jayaraman, and J. S. Kwon, "Identification of cell-to-cell heterogeneity through systems engineering approaches," *AIChE. J.*, vol. 66, no. 5, p. e16925, 2020.

[154] J. Sansana, M. N. Joswiak, I. Castillo, Z. Wang, R. Rendall, L. H. Chiang, and M. S. Reis, "Recent trends on hybrid modeling for industry 4.0," *Comput. Chem. Eng.*, vol. 151, p. 107365, 2021.

[155] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," (Las Vegas, NV, USA), pp. 770–778, 2016.

[156] L. Reyes, J. Gomez, and K. Kao, "Improving carotenoids production in yeast via adaptive laboratory evolution," *Metab. Eng.*, vol. 21, pp. 26–33, 2014.

[157] P. Mhaskar, N. H. El-Farra, and P. D. Christofides, "Stabilization of nonlinear systems with state and control constraints using Lyapunov-based predictive control," *Systems & Control Letters*, vol. 55, no. 8, pp. 650–659, 2006. New Trends in Nonlinear Control.

[158] D. Munoz de la Pena and P. D. Christofides, "Lyapunov-based model predictive control of nonlinear systems subject to data losses," *IEEE Transactions on Automatic Control*, vol. 53, no. 9, pp. 2076–2089, 2008.

[159] F. Albalawi, H. Durand, and P. D. Christofides, "Process operational safety using model predictive control based on a process safeness index," *Comput. Chem. Eng.*, vol. 104, pp. 76–88, 2017.

[160] M. Z. Romdlony and B. Jayawardhana, "Stabilization with guaranteed safety using control lyapunov-barrier function," *Automatica*, vol. 66, pp. 39–47, 2016.

[161] Z. Wu and P. D. Christofides, "Handling bounded and unbounded unsafe sets in control lyapunov-barrier function-based model predictive control of nonlinear processes," *Chem. Eng. Res. Des.*, vol. 143, pp. 140–149, 2019.

[162] B. Niu and J. Zhao, "Barrier lyapunov functions for the output tracking control of constrained nonlinear switched systems," *Systems & Control Letters*, vol. 62, no. 10, pp. 963–971, 2013.

[163] K. P. Tee, S. S. Ge, and E. H. Tay, "Barrier Lyapunov Functions for the control of output-constrained nonlinear systems," *Automatica*, vol. 45, no. 4, pp. 918–927, 2009.

[164] X. Xu, P. Tabuada, J. W. Grizzle, and A. D. Ames, "Robustness of Control Barrier Functions for safety critical control," *IFAC-PapersOnLine*, vol. 48, no. 27, pp. 54–61, 2015. Analysis and Design of Hybrid Systems ADHS.

[165] Z. Wu, F. Albalawi, Z. Zhang, J. Zhang, H. Durand, and P. D. Christofides, "Control lyapunov-barrier function-based model predictive control of nonlinear systems," *Automatica*, vol. 109, p. 108508, 2019.

[166] Y. Lin and E. D. Sontag, "A universal formula for stabilization with bounded controls," *Systems & Control Letters*, vol. 16, no. 6, pp. 393–397, 1991.

[167] S. Chen, Z. Wu, and P. D. Christofides, "Machine-learning-based construction of barrier functions and models for safe model predictive control," *AIChE Journal, e17456*, 2021.

[168] M. J. Economides, L. T. Watters, and S. Dunn-Normall, *Petroleum well construction*. Chichester: Wiley, 1998.

[169] S. Pahari, B. Bhadriraju, M. Akbulut, and J. S.-I. Kwon, "A slip-spring framework to study relaxation dynamics of entangled wormlike micelles with kinetic monte carlo algorithm," *Journal of Colloid and Interface Science*, vol. 600, pp. 550–560, 2021.

[170] S. Liu, Y. Lin, B. Bhat, K. Kuan, J. S. Kwon, and M. Akbulut, "pH-responsive viscoelastic supramolecular gels based on dynamic complexation of zwitterionic octadecylamidopropyl betaine and triamine for hydraulic fracturing applications," *RSC Advances*, vol. 11, pp. 22517–22529, 2021.

[171] S. Pahari, J. Moon, M. Akbulut, S. Hwang, and J. S.-I. Kwon, "Estimation of microstructural properties of wormlike micelles via a multi-scale multi-recommendation batch bayesian optimization," *Ind. Eng. Chem. Res.*, vol. 60, no. 43, pp. 15669–15678, 2021.

[172] B. Bhat, S. Liu, Y.-T. Lin, M. L. Sentmanat, J. Kwon, and M. Akbulut, "Supramolecular dynamic binary complexes with ph and salt-responsive properties for use in unconventional reservoirs," *PLOS ONE*, vol. 16, pp. 1–16, 12 2021.

[173] K. G. Nolte, "Determination of proppant and fluid schedules from fracturing-pressure decline," *SPE Prod Eng*, vol. 1, pp. 255–265, July 1986.

[174] H. Gu and J. Desroches, "New pump schedule generator for hydraulic fracturing treatment design," in *Proceedings of the SPE Latin American and Caribbean Petroleum Engineering Conference*, (Port-of-Spain, Trinidad and Tobago), Apr. 2003.

[175] P. Siddhamshetty, K. Wu, and J. S.-I. Kwon, "Modeling and control of proppant distribution of multistage hydraulic fracturing in horizontal shale wells," *Ind. Eng. Chem. Res.*, vol. 58, no. 8, pp. 3159–3169, 2019.

[176] Y. Ahn, P. Siddhamshetty, K. Cao, J. Han, and J. S.-I. Kwon, "Optimal design of shale gas supply chain network considering MPC-based pumping schedule of hydraulic fracturing in unconventional reservoirs," *Chem. Eng. Res. Des.*, vol. 147, pp. 412–429, 2019.

[177] P. Siddhamshetty and J. S.-I. Kwon, "Simultaneous measurement uncertainty reduction and proppant bank height control of hydraulic fracturing," *Comput. Chem. Eng.*, vol. 127, pp. 272–281, 2019.

[178] P. Siddhamshetty, P. Bhandakkar, and J. S.-I. Kwon, "Enhancing total fracture surface area in naturally fractured unconventional reservoirs via model predictive control," *J. Petr. Sci. and Eng.*, vol. 184, p. 106525, 2020.

[179] K. Cao, S. H. Son, J. Moon, and J. S.-I. Kwon, "A closed-loop integration of scheduling and control for hydraulic fracturing using offset-free model predictive control," *Applied Energy*, vol. 302, p. 117487, 2021.

[180] P. Siddhamshetty and J. S.-I. Kwon, "Model-based feedback control of oil production in oil-rim reservoirs under gas coning conditions," *Comput. Chem. Eng.*, vol. 112, pp. 112–120, 2018.

[181] P. Siddhamshetty, S. Mao, K. Wu, and J. S.-I. Kwon, "Multi-size proppant pumping schedule of hydraulic fracturing: Application to a MP-PIC model of unconventional reservoir for enhanced gas production," *Processes*, vol. 8, no. 5, 2020.

[182] P. Bhandakkar, P. Siddhamshetty, and J. S.-I. Kwon, "Numerical study of the effect of propped surface area and fracture conductivity on shale gas production: Application for multi-size proppant pumping schedule design," *Journal of Natural Gas Science and Engineering*, vol. 79, p. 103349, 2020.

[183] P. Etoughe, P. Siddhamshetty, K. Cao, R. Mukherjee, and J. S.-I. Kwon, "Incorporation of sustainability in process control of hydraulic fracturing in unconventional reservoirs," *Chem. Eng. Res. and Des.*, vol. 139, pp. 62–76, 2018.

[184] K. Cao, P. Siddhamshetty, Y. Ahn, R. Mukherjee, and J. S.-I. Kwon, "Economic model-based controller design framework for hydraulic fracturing to optimize shale gas production and water usage," *Ind. Eng. Chem.Res.*, vol. 58, no. 27, pp. 12097–12115, 2019.

[185] K. Cao, P. Siddhamshetty, Y. Ahn, M. M. El-Halwagi, and J. S.-I. Kwon, "Evaluating the spatiotemporal variability of water recovery ratios of shale gas wells and their effects on shale gas development," *Journal of Cleaner Production*, vol. 276, p. 123171, 2020.

[186] Y. Ahn, J. Kim, and J. S.-I. Kwon, "Optimal design of supply chain network with carbon dioxide injection for enhanced shale gas recovery," *Applied Energy*, vol. 274, p. 115334, 2020.

[187] S. Pahari, P. Bhandakkar, M. Akbulut, and J. S.-I. Kwon, "Optimal pumping schedule with high-viscosity gel for uniform distribution of proppant in unconventional reservoirs," *Energy*, vol. 216, p. 119231, 2021.

[188] A. Narasingam, P. Siddhamshetty, and J. S. Kwon, "Handling spatial heterogeneity in reservoir parameters using proper orthogonal decomposition based ensemble Kalman filter for model-based feedback control of hydraulic fracturing," *Ind. Eng. Chem. Res.*, vol. 57, no. 11, pp. 3977–3989, 2018.

[189] P. Siddhamshetty, K. Wu, and J. S.-I. Kwon, "Optimization of simultaneously propagating multiple fractures in hydraulic fracturing to achieve uniform growth using data-based model reduction," *Chem. Eng. Res. Des.*, vol. 136, pp. 675–686, 2018.

[190] H. Singh Sidhu, P. Siddhamshetty, and J. S. Kwon, "Approximate dynamic programming based control of proppant concentration in hydraulic fracturing," *Mathematics*, vol. 6, no. 8, 2018.

[191] P. Siddhamshetty, M. Ahammad, R. Hasan, and J. Kwon, "Understanding wellhead ignition as a blowout response," *Fuel*, vol. 243, pp. 622–629, 2019.

[192] S. Mao, P. Siddhamshetty, Z. Zhang, W. Yu, T. Chun, J. S.-I. Kwon, and K. Wu, "Impact of proppant pumping schedule on well production for slickwater fracturing," *SPE Journal*, vol. 26, pp. 342–358, 02 2021.

[193] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge: MIT Press, 1998.

[194] M. Suglyama, *Statistical Reinforcement Learning: Modern Machine Learning Approaches*. Florida: CRC Press, 2015.

[195] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[196] V. Mnih, K. Kavukcuoglu, and D. e. a. Silver, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[197] G. Tesauro, "Temporal difference learning and TD-gammon," *Commun. of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[198] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and

D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 2016.

[199] L. Lehnert and D. Precup, "Policy gradient methods for off-policy control," *arXiv Preprint, arXiv:1512.04105*, 2015.

[200] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv Preprint, arXiv:1509.02971*, 2015.

[201] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, pp. 354–359, 2017.

[202] L. A. Brujeni, J. M. Lee, and S. L. Shah, "Dynamic tuning of PI-controllers based on model-free reinforcement learning methods," in *Proceedings of the International Conference on Control, Automation and Systems (ICCAS)*, pp. 453–458, 2010.

[203] T. A. Badgwell, J. H. Lee, and K.-H. Liu, "Reinforcement learning - overview of recent progress and implications for process control," *Comput. Aid. Chem. Eng.*, vol. 44, pp. 71–85, 2018.

[204] J. Shin, T. A. Badgwell, J. H. Lee, and K.-H. Liu, "Reinforcement learning - overview of recent progress and implications for process control," *Comput. Chem. Eng.*, vol. 127, pp. 282–294, 2019.

[205] J. W. Kim, B. J. Park, H. Yoo, T. H. Oh, J. H. Lee, and J. M. Lee, "A model-based deep reinforcement learning method applied to finite-horizon optimal control of nonlinear control-affine system," *J. Process Control*, vol. 87, pp. 166–178, 2020.

[206] H. Yoo, B. Kim, J. W. Kim, and J. H. Lee, "Reinforcement learning based optimal control of batch processes using Monte-Carlo deep deterministic policy gradient with phase segmentation," *Comput. Chem. Eng.*, vol. 144, p. 107133, 2021.

[207] J. H. Lee and J. M. Lee, "Approximate dynamic programming based approach to process control and scheduling," *Comput. Chem. Eng.*, vol. 30, pp. 1603–1618, 2006.

[208] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. MA: Athena Scientific, 2005.

[209] D. Vrabie and F. Lewis, "Neural network approach to continuous-time direct adaptive optimal control for partially unknown nonlinear systems," *Neural Networks*, vol. 22, no. 3, pp. 237–246, 2009.

[210] K. G. Vamvoudakis and F. L. Lewis, "Online actor-critic algorithm to solve the continuous-time infinite horizon optimal control problem," *Automatica*, vol. 46, no. 5, pp. 878–888, 2010.

[211] Y. Ma, W. Zhu, M. G. Benton, and J. Romagnoli, "Continuous control of a polymerization system with deep reinforcement learning," *J. Process Control*, vol. 75, pp. 40–47, 2019.

[212] D. Silver, G. Lever, N. Hess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning, PMLR*, vol. 32, (Beijing, China), pp. 387–395, 2014.

[213] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, vol. 37 of *JMLR Workshop and Conference Proceedings*, (Lille, France), pp. 448–456, July 2015.

[214] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA*, May 2015.

[215] P. Van Overschee and B. De Moor, *Subspace Identification for Linear Systems: Theory - Implementation - Applications*. New York: Springer, 1996.