ML for Process Industry Series

Machine Learning in Python for Dynamic Process Systems

A practitioner's guide for building process modeling, predictive, and monitoring solutions using dynamic data

2024 Edition

Ankur Kumar, Jesus Flores-Cerrillo

`

Machine Learning in Python for Dynamic Process Systems

A practitioner's guide for building process modeling, predictive, and monitoring solutions using dynamic data

Ankur Kumar
Jesus Flores-Cerrillo

`

Dedicated to our spouses, family, friends, motherland, and all the data-science enthusiasts

`

न चोर हार्यं न च राज हार्यं न भात्रू भाज्यं न च भारकारि

व्ययं कृते वर्धत एव नित्यं विद्याधनं सर्वधनप्रधानम

No one can steal it, no king can snatch it,
It cannot be divided among the brothers and it's not heavy to carry,
As you consume or spend, it increases; as you share, it expands,
The wealth of knowledge is the most precious wealth you can have.

- *A popular Sanskrit shloka*

`

Machine Learning in Python for Dynamic Process Systems

www.MLforPSE.com

First published: June 2023
Revision 1 published: January 2024

`

About the Authors

Ankur Kumar holds a PhD degree (2016) in Process Systems Engineering from the University of Texas at Austin and a bachelor's degree (2012) in Chemical Engineering from the Indian Institute of Technology Bombay. He currently works at Linde in the Advanced Digital Technologies & Systems Group in Linde's Center of Excellence, where he has developed several in-house machine learning-based monitoring and process control solutions for Linde's hydrogen and air-separation plants. Ankur's tools have won several awards both within and outside Linde. Ankur's tools have won several awards both within and outside Linde. Most recently, one of his tools, PlantWatch (a plantwide fault detection and diagnosis tool), received the 2021 Industry 4.0 Award by the Confederation of Industry of the Czech Republic. Ankur has authored or co-authored several peer-reviewed journal papers (in the areas of data-driven process modeling and monitoring), is a frequent reviewer for many top-ranked research journals, and has served as Session Chair at several international conferences. Ankur served as an Associate Editor of the Journal of Process Control from 2019 to 2021, and currently serves on the Editorial Advisory Board of Industrial & Engineering Chemistry Research Journal. Most recently, he was included in the 'Engineering Leaders Under 40, Class of 2023' by *Plant Engineering* Magazine.

Jesus Flores-Cerrillo is currently an Associate Director - R&D at Linde and manages the Advanced Digital Technologies & Systems Group in Linde's Center of Excellence. He has over 20 years of experience in the development and implementation of monitoring technologies and advanced process control & optimization solutions. Jesus holds a PhD degree in Chemical Engineering from McMaster University and has authored or co-authored more than 40 peer-reviewed journal papers in the areas of multivariate statistics and advanced process control among others. His team develops and implements novel plant monitoring, machine learning, IIOT solutions to improve the efficiency and reliability of Linde's processes. Jesus's team received the Artificial Intelligence and Advanced Analytics Leadership 2020 award from the National Association of Manufacturers' Manufacturing Leadership Council.

`

Note to the readers

Jupyter notebooks and Spyder scripts with complete code implementations are available for download at https://github.com/ML-PSE/Machine_Learning_for_DPS. Code updates when necessary, will be made and updated on the GitHub repository. Updates to the book's text material will be available on Leanpub (www.leanpub.com) and Google Play (https://play.google.com/store/books). We would greatly appreciate any information about any corrections and/or typos in the book.

`

Series Introduction

In the 21st century, data science has become an integral part of the work culture at every manufacturing industry and process industry is no exception to this modern phenomenon. From predictive maintenance to process monitoring, fault diagnosis to advanced process control, machine learning-based solutions are being used to achieve higher process reliability and efficiency. However, few books are available that adequately cater to the needs of budding process data scientists. The scant available resources include: 1) generic data science books that fail to account for the specific characteristics and needs of process plants 2) process domain-specific books with rigorous and verbose treatment of underlying mathematical details that become too theoretical for industrial practitioners. Understandably, this leaves a lot to be desired. Books are sought that have process systems in the backdrop, stress application aspects, and provide a guided tour of ML techniques that have proven useful in process industry. This series '***Machine Learning for Process Industry***' addresses this gap to reduce the barrier-to-entry for those new to process data science.

The first book of the series '***Machine Learning in Python for Process Systems Engineering***' covers the basic foundations of machine learning and provides an overview of broad spectrum of ML methods primarily suited for static systems. Step-by-step guidance on building ML solutions for process monitoring, soft sensing, predictive maintenance, etc. are provided using real process datasets. Aspects relevant to process systems such as modeling correlated variables via PCA/PLS, handling outliers in noisy multidimensional dataset, controlling processes using reinforcement learning, etc. are covered. This second book of the series is focused on dynamic systems and provides a guided tour along the wide range of available dynamic modeling choices. Emphasis is paid to both the classical methods (ARX, CVA, ARMAX, OE, etc.) and modern neural network methods. Applications on time series analysis, noise modeling, system identification, and process fault detection are illustrated with examples. Future books of the series will continue to focus on other aspects and needs of process industry. It is hoped that these books can help process data scientists find innovative ML solutions to the real-world problems faced by the process industry.

Books of the series will be useful to practicing process engineers looking to 'pick up' machine learning as well as data scientists looking to understand the needs and characteristics of process systems. With the focus on practical guidelines and real industrial case studies, we hope that these books lead to wider spread of data science in the process industry.

`

Preface

Model predictive control (MPC) and real-time optimization (RTO) are among the most critical technologies that drive the process industry. Any experienced process engineer would vouch that the success of MPCs and RTOs depends heavily on the accuracy of the underlying process models. Similarly, the key requirement for other important process technologies like dynamic data reconciliation, process monitoring, etc. is availability of accurate process models. Modeling efforts during commissioning of these tools can easily consume up to 90% of the project cost and time. Modern data revolution, whereby abundant amount of process data are easily available, and practical difficulties in building high-fidelity first-principles dynamic models for complex industrial processes have popularized the usage of empirical data-driven/machine learning (ML) models. Building dynamic models using process data is called system identification (SysID) and it's a very mature field with extensive literature. However, it is also very easy for a process data scientist (PDS) new to this field to get overwhelmed with the SysID mathematics and 'drowned' in the sea of SysID terminology. Several noteworthy ML books have been written on time-series analysis; however, in process industry, input-output models are of greater import. Unfortunately, there aren't many books that cater to the needs of modern PDSs interested in dynamic process modeling (DPM) without weighing them down with too much mathematical details and therein lies our motivation for authoring this book: specifically, a reader-friendly and easy to understand book that provides a comprehensive coverage of ML techniques that have proven useful for building dynamic process models with focus on practical implementations.

It would be clear to you by now that this book is designed to teach working process engineers and budding PDSs about DPM. While doing so, this book attempts to avoid a pitfall that several generic ML books fall into: overemphasis on 'modern' and complex ML techniques such as artificial neural networks (ANNs) and undertreatment of classical DPM methods. Classical techniques like FIR and ARX still dominate the dynamic modeling solutions offered by commercial vendors of industrial solutions and are no less 'machine-learning' than the ANNs. These two along with other classical techniques (such as OE, ARIMAX, CVA) predate the ANN-craze era and have stood the test of time in providing equal (if not superior) performance compared to ANNs. Correspondingly, along with modern ML techniques like RNNs, considerable portion of the book is devoted to classical dynamic models with the emphasis on understanding the implications of modeling decisions such as the impact of implicit noise model assumption with ARX model, the implication of differencing data, etc.

`

Guided by our own experience from building process models for varied industrial applications over the past several years, this book covers a curated set of ML techniques that have proven useful for DPM. The broad objectives of the book can be summarized as follows:

- reduce barrier-to-entry for those new to the field of SysID
- provide working-level knowledge of SysID techniques to the readers
- enable readers to make judicious selection of a SysID technique appropriate for their problems through intuitive understanding of the advantages and drawbacks of different methods
- provide step-by-step guidance for developing tools for soft sensing, process monitoring, predictive maintenance, etc. using SysID methods

This book adopts a tutorial-style approach. The focus is on guidelines and practical illustrations with a delicate balance between theory and conceptual insights. Hands-on-learning is emphasized and therefore detailed code examples with industrial-scale datasets are provided to concretize the implementation details. A deliberate attempt is made to not weigh readers down with mathematical details, but rather use it as a vehicle for better conceptual understanding. Complete code implementations have been provided in the GitHub repository. Although most of the existing literature on SysID use MATLAB as the programming environment, we have adopted Python in this book due to its immense popularity among the broad ML community. Several Python libraries are now available which makes DPM using Python convenient.

We are quite confident that this text will enable its readers to build dynamic models for challenging problems with confidence. We wish them the best of luck in their career.

Who should read this book

The application-oriented approach in this book is meant to give a quick and comprehensive coverage of dynamic modeling methodologies in a coherent, reader-friendly, and easy-to-understand manner. The following categories of readers will find the book useful:

1) Data scientists new to the field of system identification
2) Regular users of commercial process modeling software looking to obtain a deeper understanding of the underlying concepts
3) Practicing process data scientists looking for guidance for developing process modeling and monitoring solutions for dynamic systems
4) Process engineers or process engineering students making their entry into the world of data science

`

Pre-requisites

No prior experience with machine learning or Python is needed. Undergraduate-level knowledge of basic linear algebra and calculus is assumed.

Book organization

Under the broad theme of ML for process systems engineering, this book is an extension of the first book of the series (which dealt with fundamentals of ML and its varied applications in process industry); however, it can also be used as a standalone text. To give due treatment to various aspects of SysID, the book has been divided into three parts**. Part 1** of the book provides a perspective on the importance of ML for dynamic process modeling and lays down the basic foundations of ML-DPM (machine learning for dynamic process modeling). **Part 2** provides in-detail presentation of classical ML techniques and has been written keeping in mind the different modeling requirements and process characteristics that determine a model's suitability for a problem at hand. These include, amongst others, presence of multiple correlated outputs, process nonlinearity, need for low model bias, need to model disturbance signal accurately, etc. **Part 3** is focused on artificial neural networks and deep learning. While deep learning is the current buzzword in ML community, we would like to caution the reader against the temptation to deploy a deep learning model for every problem at hand. For example, the models covered in Part 2 still dominate the portfolio of models used in industrial controllers and can often provide comparable (or even superior) performance compared to ANNs with relatively less hassle.

Symbol notation

The following notation has been adopted in the book for representing different types of variables:

- lower-case letters refer to vectors ($x \in \mathbb{R}^{m\times 1}$) and upper-case letters denote matrices ($X \in \mathbb{R}^{n\times m}$)
- individual element of a vector and a matrix are denoted as x_j and x_{ij}, respectively.
- any *i*th vector in a dataset gets represented as subscripted lower-case letter (e.g., $x_i \in \mathbb{R}^{m\times 1}$). Its distinction from an individual element x_j would be clear from the corresponding context.

`

Installing Required Packages

In this book, SIPPY package (https://github.com/CPCLAB-UNIPI/SIPPY) is used for classical dynamic modeling of input-output systems. Below are a few remarks regarding the installation of the package

- The package code can be downloaded from its GitHub repository. You may put the 'sippy' folder provided in the package's distribution in your working directory to use the provided models
- To run the code provided in this book, Slycot package is not required
- Control package's version should be < 0.9 (https://github.com/CPCLAB-UNIPI/SIPPY/issues/48)

`

Table of Contents

`

`

`

Part 1

Introduction & Fundamentals

Chapter 1

Machine Learning and Dynamic Process Modeling: An Introduction

Process industry operations are dynamic in nature. In complex process plants such as oil refineries, 1000s of process measurements may be recorded every second to capture crucial process trends. Plant engineers often employ dynamic process models to predict future values of process variables in applications such as process control, process monitoring, etc. Machine learning (ML) provides a convenient mechanism to bring together the dynamic process modeling (DPM) needs and the large data resources available in modern process plants.

This chapter provides an overview of what ML has to offer for DPM. This chapter also addresses a dichotomy between ML community and DPM community. While the former generally tends to claim that ML has been reduced to mere execution of 'model-fitting' actions, the later vouches for the need of 'experts' to build 'successful' models. We will attempt to reconcile these two differing viewpoints.

Overall, this chapter provides a whirlwind tour of how the power of machine learning is harnessed for dynamic process modeling. Specifically, the following topics are covered

- Introduction to dynamic process modeling and the need for machine learning
- Typical workflow in a ML-based DPM (ML-DPM) project
- Taxonomy of popular ML-DPM methods and models
- Applications of DPM in process industry

Let's now tighten our seat-belts as we embark upon this exciting journey of de-mystifying machine learning for dynamic process modeling.

`

1.1 Process Systems Engineering, Dynamic Process Modeling, and Machine Learning

Process industry is a parent term used to refer to industries like petrochemical, chemical, power, paper, cement, pharmaceutical, etc. These industries use processing plants to manufacture intermediate or final consumer products. As emphasized in Figure 1.1, the prime concerns of the management of these plants include, amongst others, optimal design and operations, high reliability through proactive process monitoring, quality control, and data reconciliation. All these tasks fall under the ambit of process systems engineering (PSE).

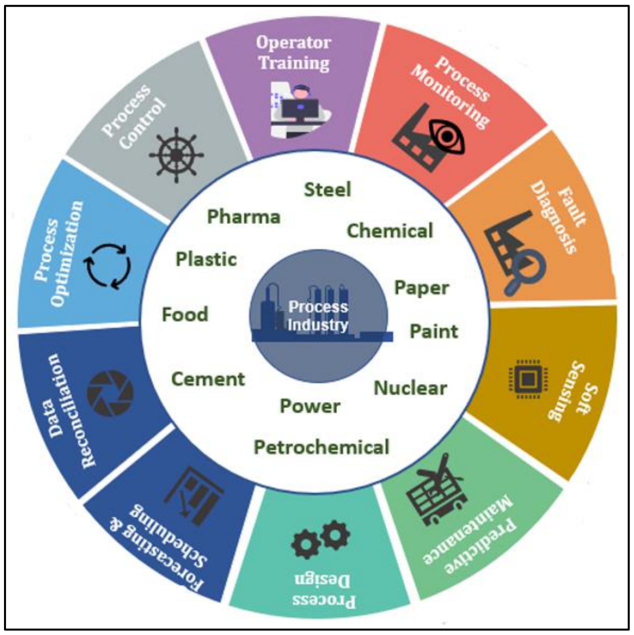


Figure 1.1: Overview of industries that constitute process industry and the tasks process systems engineers perform

The common theme among the PSE tasks is that they all rely on reliable dynamic mathematical process models that can accurately predict the future state of the process using current and past process measurements. Therefore, DPM is one of the defining skills of process systems engineers. Process industry has historically utilized both first principles/phenomenological and empirical/data-based models for DPM. While the former models provide higher fidelity, the later models are easier to build for complex systems. The immense rise in popularity of machine learning/data science in recent years and exponential increase in sensor measurements collected at plants have led to renewed interest in ML-based DPM. Several classical and 'modern' ML methods for DPM are at a process data scientist's disposal. The rest of the book will take you on a whirlwind tour of these methods. Let's now jump straight into the nitty-gritty of ML-DPM.

Components of a dynamic process model

In the context of DPM, there are three types of variables/signals in a process system as shown in Figure 1.2. The measured signals that we desire to predict are termed outputs while the measured signals that can be manipulated to influence the outputs are called inputs. In industrial processes, these outputs are seldom solely influenced or completely explained by inputs. This unexplained component is attributed to the noise signals which can manifest as measurement noise or process noise (unmeasured disturbances or unmeasured signals that impact the system are often the cause of process noise). In the pH neutralization process example shown below, process noise corresponds to unmeasured fluctuations in the wastewater acidity and/or flow which causes disturbances in the process output.

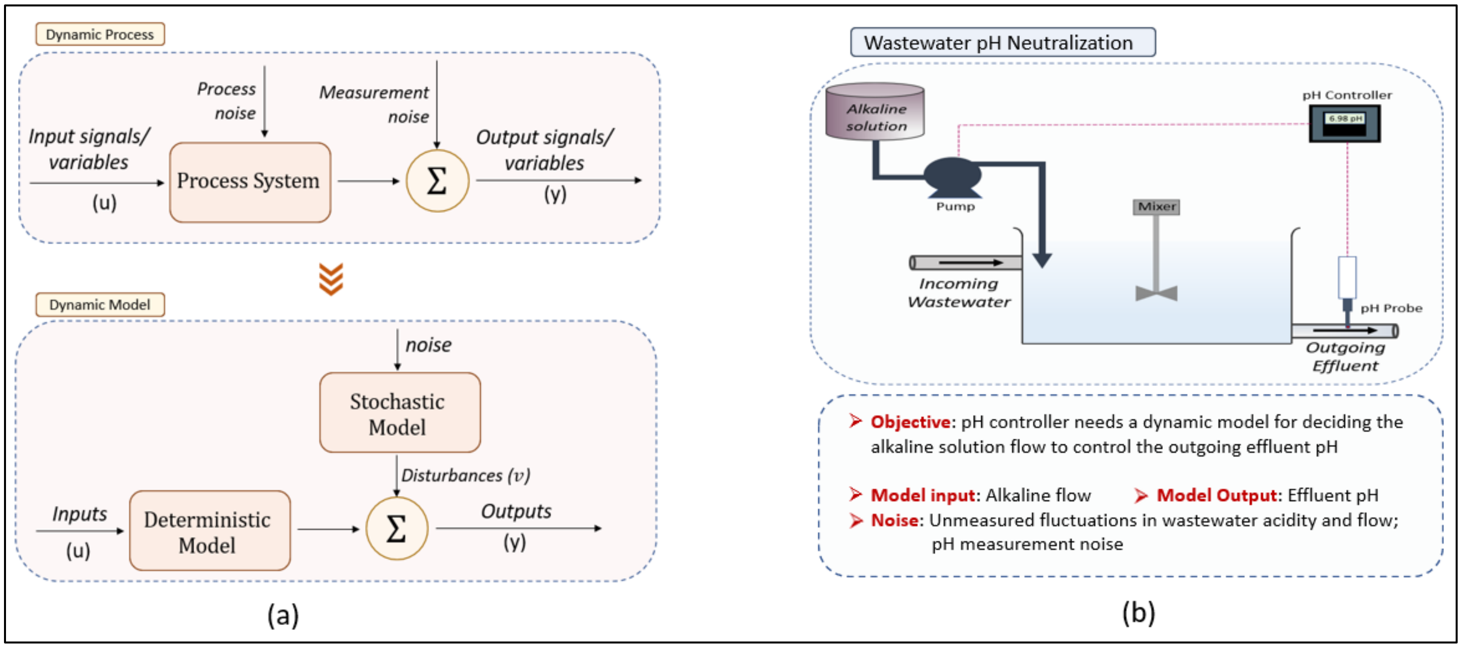

Dynamic Process
Process noise
Measurement noise
Input signals/ variables (u)
Process System
Σ
Output signals/ variables (y)
Dynamic Model
noise
Stochastic Model
Disturbances (v)
Inputs (u)
Deterministic Model
Σ
Outputs (y)
(a)
Wastewater pH Neutralization
Alkaline solution
pH Controller
6.98 pH
Pump
Mixer
Incoming Wastewater
pH Probe
Outgoing Effluent
Objective: pH controller needs a dynamic model for deciding the alkaline solution flow to control the outgoing effluent pH
Model input: Alkaline flow
Model Output: Effluent pH
Noise: Unmeasured fluctuations in wastewater acidity and flow; pH measurement noise
(b)

Figure 1.2: (a) Process system (and its dynamic model) with input, output, and noise signals (b) A pH neutralization process broken down into its dynamic components

The impact of measurement and process noise can be clubbed together and explained via a stochastic model as shown in the figure above. The resulting disturbance signal summarizes all the uncertain characteristics of the process. The stochastic and deterministic models can be estimated simultaneously as well as separately.

In the engineering community, the task of building mathematical models of a dynamic system using measured data is called system identification (SysID). Additionally, the task of modeling dynamic systems without input variables is termed time-series analysis.

`

Dynamic modeling notation

In this book, we will focus on time-invariant discrete-time models wherein the signals will be assumed to be recorded at regular sampling interval of *T* time units. The output *y* at time *kT* (*k* is any integer) will be denoted as *y(k)*. System identification therefore entails finding the relationship between the three signals *y(k), u(k), and v(k)*. For the pH example, a simple discrete-time linear model could look like the following

$$\begin{aligned} \hat{y}_{pH}(k) &= \alpha\hat{y}_{pH}(k-1) + \beta u_{alkaline}(k-1) \\ y_{pH}(k) &= \hat{y}_{pH}(k) + v(k) \end{aligned} \qquad \text{eq. 1}$$

where *α* and *β* are estimable model parameters and the disturbance variable, *v(k)*, summarizes all the uncertainties. Here, the value of the output at the *k*th time instant is predicated upon the past values of both output and input, and the disturbance. In the later chapters, we will study how to characterize *v(k)* using stochastic models.

We cannot emphasize enough the importance of proper handling of process disturbance signals. When in hurry, you may be tempted to ignore the stochastic component. However, doing so would only be inviting disappointment as you may end up with biased deterministic models with unsatisfactory performance. This interlink between stochastic and deterministic models would probably not be obvious to beginner PDSs. By the time you finish this book, this connection will become obvious.

Static vs dynamic model

Before we proceed further, let's take a quick look at how a dynamic model differs from a static model. Consider a SISO process in Figure 1.3 where a step change in input is induced and the corresponding change in output is observed.

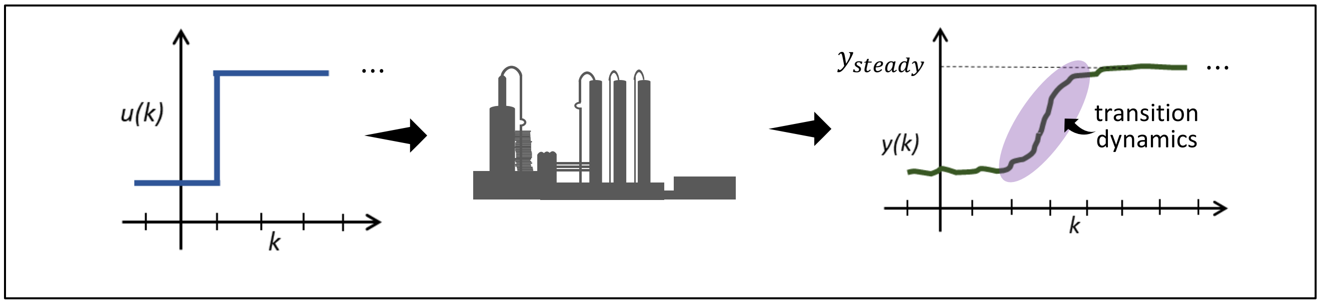


Figure 1.3: Representative dynamic changes in a SISO process output upon a step change in input

`

- In static model, we are only concerned with being able to predict y_{steady} as the impact of the step change in u.
- On the other hand, in dynamic model, we care about the transition dynamics as well.
- It is obvious that a dynamic model estimation is a more demanding problem and the complexity only increases for multivariable systems with stochastic components.

Validity of discrete-time models

It won't be wrong to say that continuous-time description of process systems (which are inherently continuous in nature) is more natural than discrete-time description. However, in computer controlled industrial plants, signals from the underlying continuous process are sampled and made available at discrete time-points. Therefore, there is sound rationale for focusing on discrete-time models.

When using first principles approach, discrete-time model can be obtained as an approximation as well as an exact description of the underlying process. For example, consider the following differential equation

$$\frac{dy}{dt} = f(y(t), u(t))$$

Approximating the derivative via forward difference gives,

$$\frac{y(t+T) - y(t)}{T} = f(y(t), u(t)); \quad T \; is \; sampling \; interval$$

$$\Rightarrow y(t+T) = y(t) + Tf\big(y(t), u(t)\big)$$
$$or$$
$$\boxed{y(k+1) = y(k) \; + \; T\, f(y(k), u(k)); \quad k = \frac{t}{T}}$$

Discrete-time model

In an alternate scenario, if the input variables are constant between samples, then an exact analytical discrete-time representation can be derived for linear processes.

`

1.2 ML-DPM Workflow

Figure 1.4 shows the typical steps involved in system identification. As you can see, SysID is more than just curve fitting. Overall, there are five broad tasks: data collection, exploratory data analysis, data pre-treatment, model identification, and model validation. Although we will study each step in detail in Chapter 4, let's take a quick overview:

- ***Data collection and exploratory data analysis****:* Availability of 'proper' data is absolutely critical and the requirement of 'richness' of training data is indispensable. In the previous section we saw that dynamic modeling is more demanding than static modeling and consequently, there are more stringent requirements on training data for dynamic modeling. Training data may be taken from historical database or fresh experiments may be performed. The training inputs should be such that the output data contains the dynamic variations of interest.

  Exploratory data analysis (EDA) involves preliminary (and usually manual) investigation of data to get a 'feel' of the system's characteristics. The activities may include generating some graphical plots to check the presence of trends, seasonality, and non-stationarity in data. Inferences made during EDA help make the right choices in the subsequent steps of SysID.

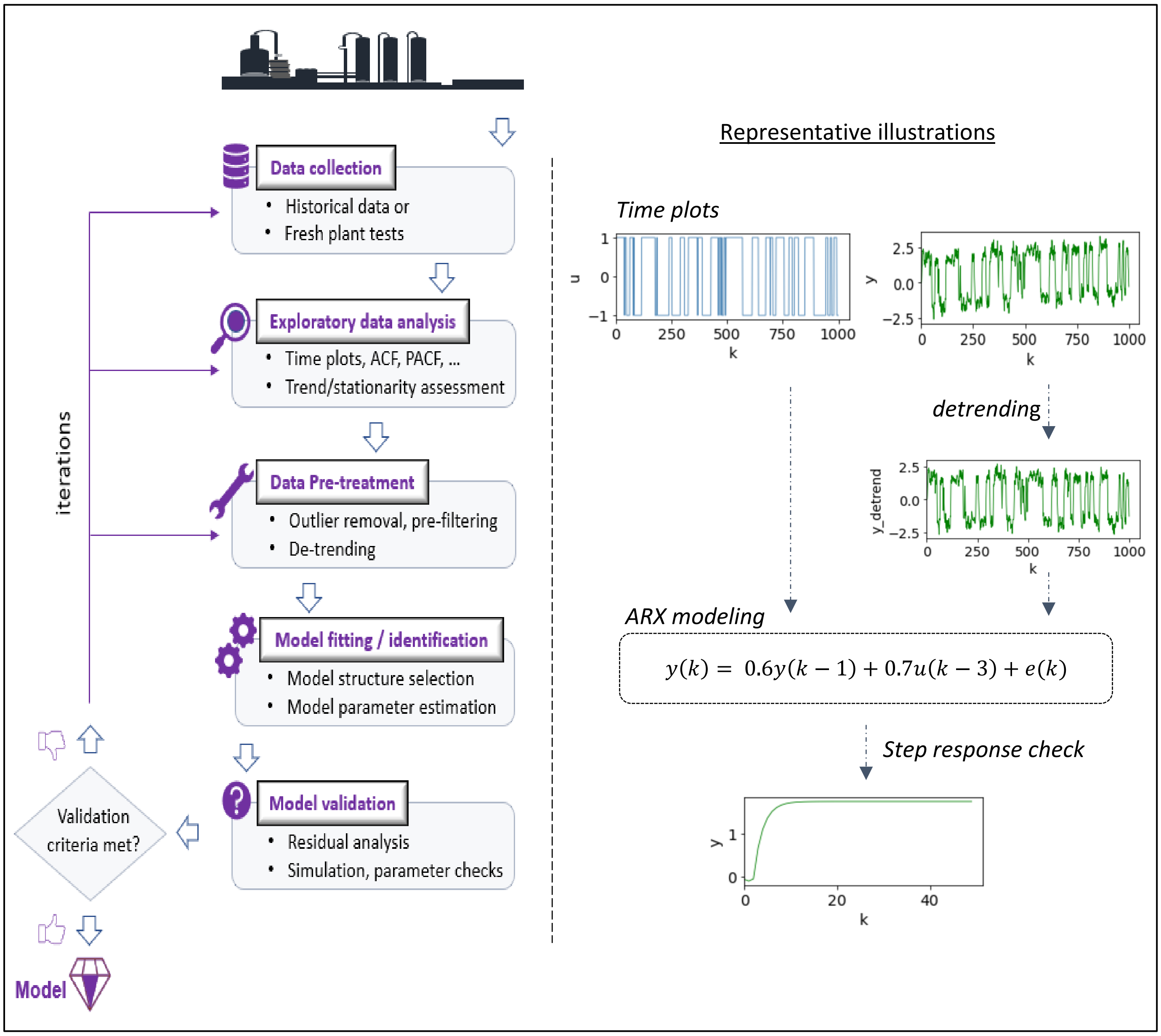
- ***Data pre-treatment***: This step consists of several activities that are designed to remove the portions of training data that are unimportant (or even detrimental) to model identification. It may entail removal of outliers and noise, removal of trends, etc. Alternatively, training data can also be massaged to manipulate the model's accuracy as suited for the end purpose of the model. For example, if the model is to be used for control purpose, then the training data may be pre-filtered to bolster model's accuracy for high frequency signals. Overall, the generic guideline is that you have better chances of a successful SysID with a better conditioned dataset.

- ***Model training***: Model training is the most critical step in SysID and entails a few sub-steps. First, a choice must be made on the type of disturbance model and deterministic model. The end use of the model, the available *a priori* system knowledge, and the desired degree of modeling complexity dictate these selections. For example, if the model is to be used for simulation purposes, then OE[1] structure may be preferred over ARX; if process disturbance has different dynamics compared to process input, then ARMAX will be preferable; if computational convenience is sought, then ARX is usually

[1] We will cover these model structures in the upcoming chapters

`

the first choice and is preferred over other complex structures such as recurrent neural networks. Model structure selection is followed by model parameter estimation. This sub-step is mostly 'hands-off' due to the availability of several specialized libraries that perform parameter estimation.



Representative illustrations
Data collection
Historical data or
Fresh plant tests
Exploratory data analysis
Time plots, ACF, PACF, ...
Trend/stationarity assessment
Data Pre-treatment
Outlier removal, pre-filtering
De-trending
Model fitting / identification
Model structure selection
Model parameter estimation
Model validation
Residual analysis
Simulation, parameter checks
iterations
Validation criteria met?
Model
Time plots
detrending
ARX modeling
$y(k) = 0.6y(k-1) + 0.7u(k-3) + e(k)$
Step response check

Figure 1.4: Steps (with sample sub-steps) involved in a typical ML-based dynamic modeling

`

- ***Model validation***: Once a model has been fitted, the next step is to check if the model truly represents the underlying process. Several techniques are at a modeler's disposal. For example, you could check the model's performance on a dataset that has not been used during parameter estimation, plot the modeling errors to look for leftover patterns, or check if the model agrees with the *a priori* information about the system. Often, your first model will fail the validation procedure. An expert modeler can use the validation results to infer the reasons for the failure. Some examples of the cause could be
    - ➢ Training data was not 'rich' enough
    - ➢ Training data was not pre-treated adequately
    - ➢ Choice of model structure was wrong
    - ➢ Estimation algorithm did not converge

  Once diagnosed, appropriate corrections are made, and the iterative procedure continues.

We hope that you get the understanding that SysID is more than just black-box application of modeling algorithm for estimation of model parameters. There are several practical aspects that require active participation of the modeler during the SysID process. Without exaggeration, we can say that SysID is an art, and the rest of the book will help you obtain the necessary skills to become the SysID artist!

1.3 Taxonomy of ML-based Dynamic Models

The field of system identification is overwhelmingly extensive owing to the decades of research that has led to the development of several specialized techniques. Figure 1.5 gives an overview of some of the popular methods and models that we will cover in this book. These models represent only a subset of all the SysID models out there and this selection is based on our experience regarding the relevance of these models for process systems modeling. We largely focus on time-invariant[2], discrete-time models and these shown models can help you handle a large majority of DPM problems you will encounter in process industry.

[2] Models where time variable does not appear as an explicit variable.

`

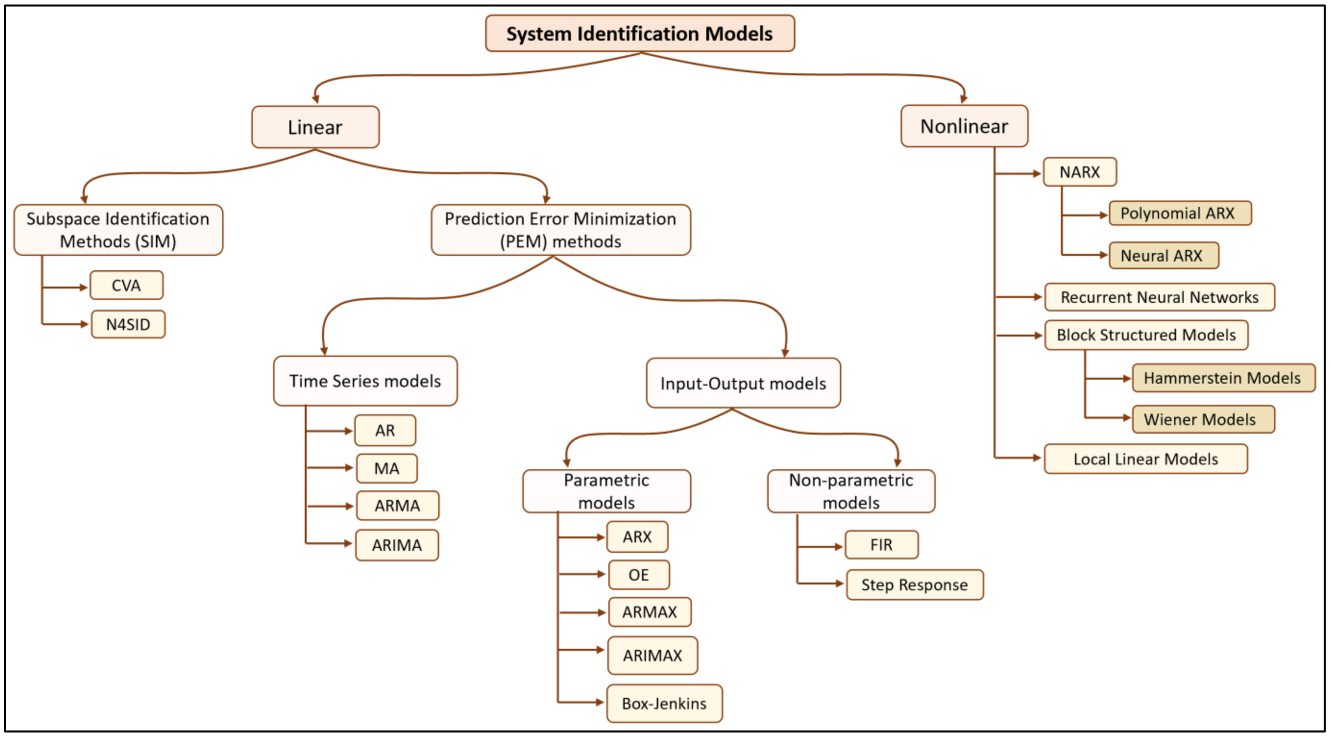


Figure 1.5: Some popular SysID methods and models covered in this book

There are several noteworthy points in Figure 1.5. First, the apparent domination of linear models: you may have expected nonlinear models to be preferable for modeling complex industrial process systems, however, as it turns out, linear models are often justified for DPM. The justification stems from the fact that industrial processes often operate around an optimal point making the linear models pretty good approximation for usage in process control and process monitoring applications. If the end use of the model is process simulation or design where the system response over wide range of input variables is of interest, then nonlinear models can prove to be more suitable. Within linear category, different modeling options exist to cater to process systems with different characteristics, viz, multivariable outputs, presence of correlated noise, disturbances sharing dynamics with inputs, presence of measurement noise only, presence of drifts or non-stationarities, etc.

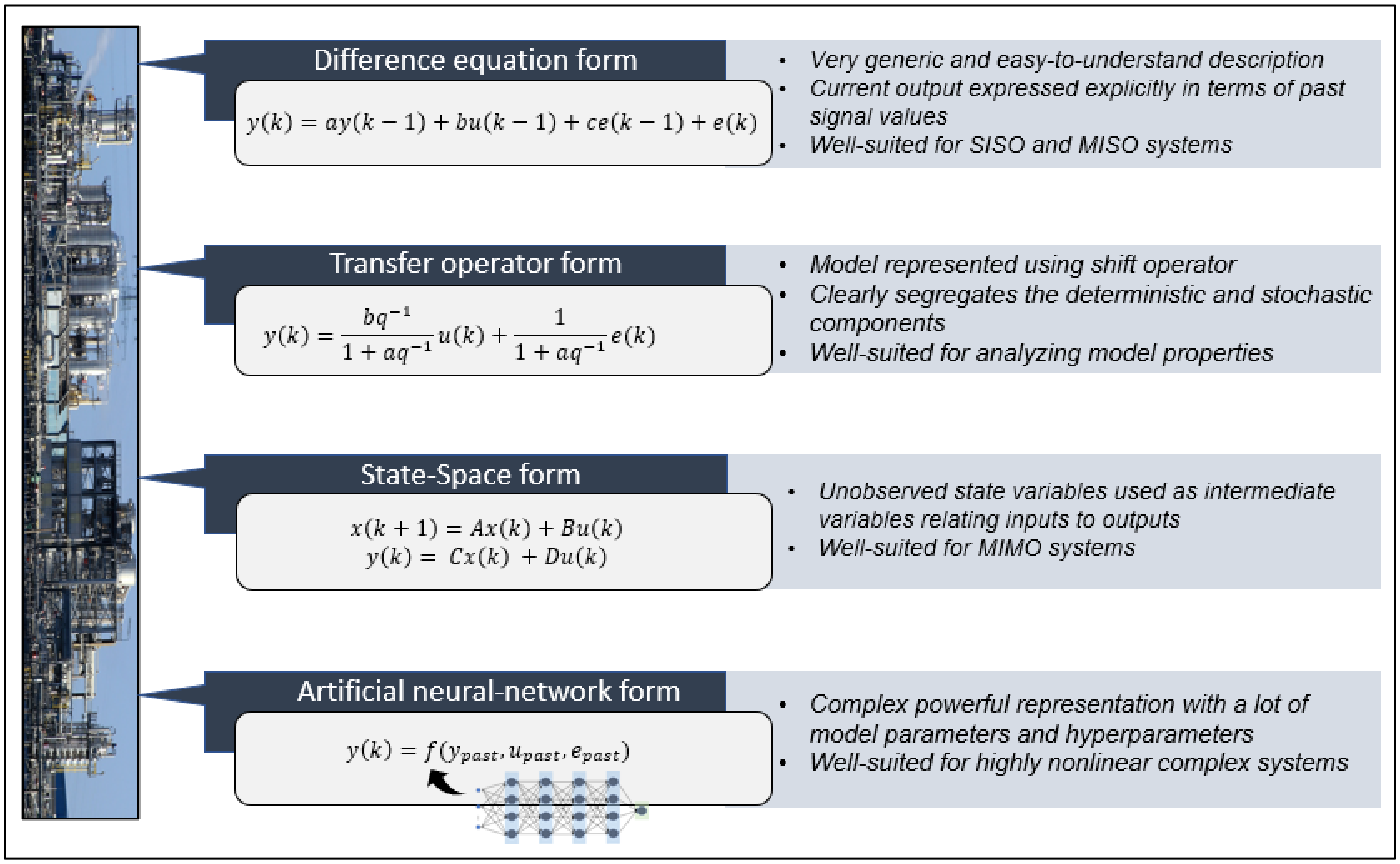
The classification at the root of linear model sub-tree is based on the methodology used for model fitting. While PEM methods employ minimization of prediction errors, subspace methods are based on matrix algebra and do not involve any optimization for parameter optimization.[3] Do not worry if these terms do not make much sense right now; they will soon become ‘obvious’ to you. If not trained about the nuanced differences between these different models and methods, you may not have much idea at the outset about which model would be the best one for your system. This book will help you gain adequate conceptual understanding to become adept at making the right choice and obtaining your coveted model quickly.

[3] Another distinction is that PEM is used to obtain input-output models while SIM is used to obtain state-space models.

`

Types of models

In Eq. 1 we saw one way of representing a dynamic process model. The models from Figure 1.5 can be used to generate other representations of your dynamic systems; the figure below shows the different forms/types of models that we will learn to derive in this book using machine learning. We will also understand the pros and cons of these different model forms.



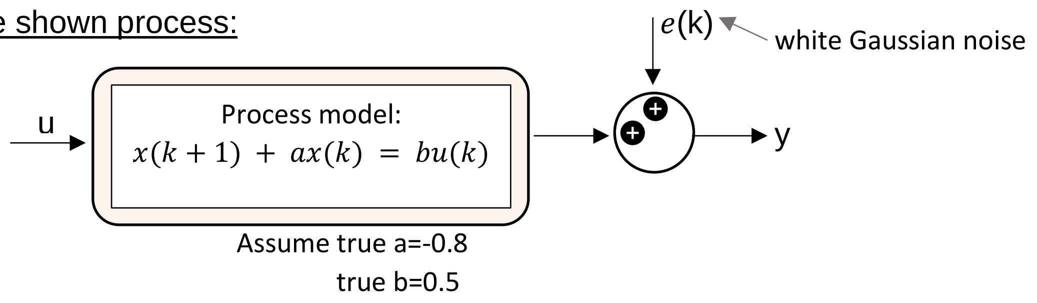
Difference equation form
$y(k) = ay(k-1) + bu(k-1) + ce(k-1) + e(k)$
Very generic and easy-to-understand description
Current output expressed explicitly in terms of past signal values
Well-suited for SISO and MISO systems
Transfer operator form
$y(k) = \frac{bq^{-1}}{1+aq^{-1}}u(k) + \frac{1}{1+aq^{-1}}e(k)$
Model represented using shift operator
Clearly segregates the deterministic and stochastic components
Well-suited for analyzing model properties
State-Space form
$x(k+1) = Ax(k) + Bu(k)$
$y(k) = Cx(k) + Du(k)$
Unobserved state variables used as intermediate variables relating inputs to outputs
Well-suited for MIMO systems
Artificial neural-network form
$y(k) = f(y_{past}, u_{past}, e_{past})$
Complex powerful representation with a lot of model parameters and hyperparameters
Well-suited for highly nonlinear complex systems

Figure 1.6: Different forms of dynamic process model (that we will learn to generate in this book), their defining characteristics, and corresponding representative examples

`

Why careful choice of model matters: A simple illustration

Consider the shown process:



If not careful, you may just ignore the presence of measurement noise and attempt to fit the following input-output model to estimate the model parameters *a* and *b*.

Fitted Model (ARX form): $y(k+1) + a\,y(k) = bu(k)$

Data file ‘simpleProcess.csv’ contains 1000 samples of *u* and *y* obtained from the true process. Below are the parameter estimates we get using this data,

$$\hat{a} = -0.6628\ (\pm\, 0.016)\ ,\ \hat{b} = 0.7174\ (\pm 0.035)$$

values in bracket denote standard errors

Two things are striking here: the estimates are grossly inaccurate, and the parameter error estimates seem to suggest high confidence in these wrong values! What went wrong in our approach? The reason is that the SNR (signal-to-noise ratio) value is not very high (data was generated with SNR ~ 10) and the fitted ARX model is wrong input-output form of the true process. The correct form would be the following

Correct input-output model

$$\begin{aligned} x(k+1) + ax(k) &= bu(k) \\ y(k) &= x(k) + e(k) \end{aligned} \quad \Rightarrow \quad \boxed{y(k+1) + a\,y(k) = bu(k) + \boxed{e(k+1) + ae(k)}}$$

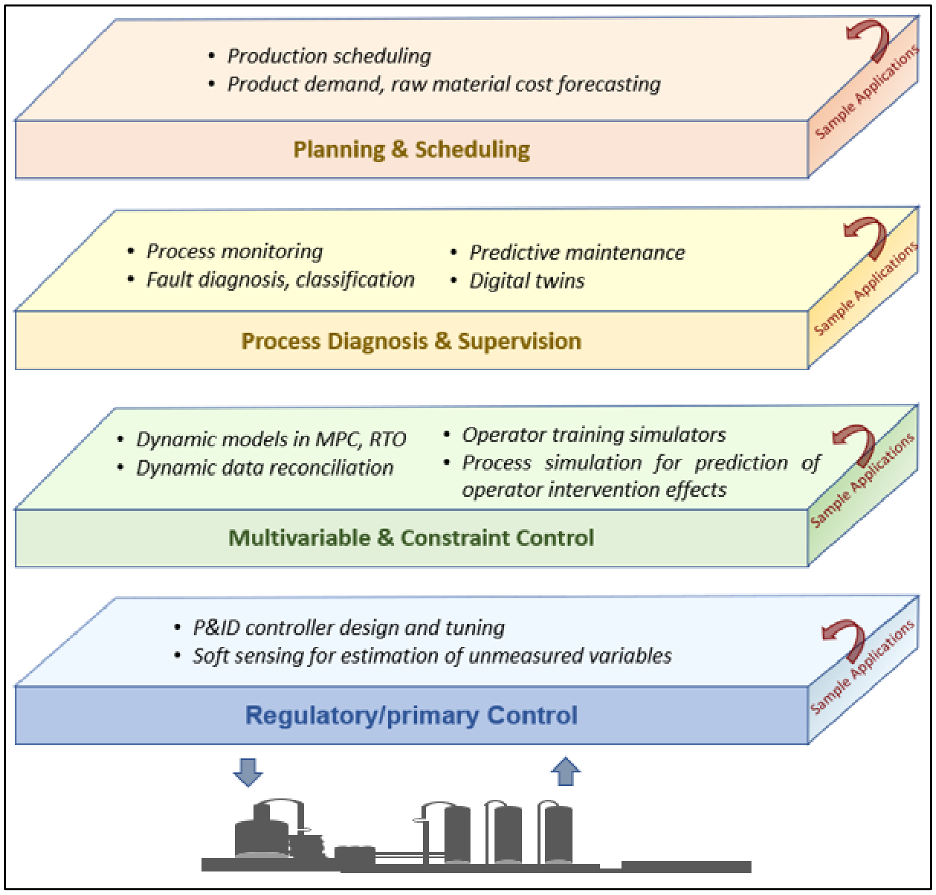
Equation error is not white (as assumed in ARX model) but colored!

Ignoring the presence of correlated equation error leads to biased (inaccurate) parameter estimates. Hopefully, this simple illustration has convinced you against blind application of ‘convenient’ models. Part II of this book will add several modeling tools to your ML-DPM arsenal to help avoid such modeling mistakes.

`

1.4 Applications of DPM in Process Industry

In Figure 1.1, we saw some of the applications of dynamic models in process industry. To provide further perspectives into how a typical plant operator or management may use these varied applications, Figure 1.7 juxtaposes DPM applications alongside the typical decision-making hierarchy in a process plant. Every step of plant operation is now-a-days heavily reliant on DPM-based tools and machine learning has proven to be a useful vehicle for quickly building these tools. You can use the models from Figure 1.5 for building these tools or if you use commercial vendor solutions, you can find them employing these models in their products. For example, in the process control field, FIR models have been the bedrock of industrial MPC controllers. In the last few years, commercial vendors have inducted CVA models in their offering due to the advantages provided by subspace models. The latest offering by Aspen, DMC3[4], incorporates neural networks for MPC and inferential modeling. ARX, BJ models are also used for industrial MPC[5].



Production scheduling
Product demand, raw material cost forecasting
Planning & Scheduling
Sample Applications
Process monitoring
Fault diagnosis, classification
Predictive maintenance
Digital twins
Process Diagnosis & Supervision
Sample Applications
Dynamic models in MPC, RTO
Dynamic data reconciliation
Operator training simulators
Process simulation for prediction of operator intervention effects
Multivariable & Constraint Control
Sample Applications
P&ID controller design and tuning
Soft sensing for estimation of unmeasured variables
Regulatory/primary Control
Sample Applications

Figure 1.7: DPM applications in different layers of operational hierarchy

[4] https://www.aspentech.com/en/products/msc/aspen-dmc3

[5] Qin and Badgwell, A survey of industrial model predictive control technology. *Control Engineering Practice*, 2003

- The regulatory control layer primarily comprises of (PID) control valves. Potential DPM applications may involve usage of soft-sensor for estimation of difficult-to-measure controlled variables.

⬇

- The multivariable control layer usually consists of MPC and RTO modules. Here, dynamic models representing multivariable relationships between plant variables are used to ensure optimal operations of the plant. Another interesting application is development of operator training simulators (OTSs) for training plant operators.

⬇

- The process diagnostics layer ensures timely fault detection and diagnostics. Here, process measurements can be compared against predictions from dynamic models to check for presence of process abnormalities.

⬇

- The production scheduling layer has dynamic models to determine short-term optimal production schedules using forecasts of product demand and/or raw-material cost.

This concludes our quick attempt to establish the connection between process industry, dynamic process modeling, and machine learning. It must now be obvious to you that modern process industry relies heavily on dynamic process modeling to achieve its objectives of reducing maintenance costs, and increasing productivity, reliability, safety, and product quality. ML-based DPM helps build tools quickly to facilitate achieving these objectives.

This introductory chapter has also cautioned against blind application of ‘convenient’ ML models for dynamic modeling. Your process data will throw several questions at you at each stage of system identification. No straightforward answers exist to these questions and only some time-tested guiding principles are available. While the rest of the book will familiarize you with these principles, the onus still lies on you to use your process insights and SysID

`

understanding to make the right modeling choices. And yes, remember the age-old advice[6], "All models are wrong, but some are useful."

Summary

This chapter impressed upon you the importance of DPM in process industry and the role machine learning plays in it. We familiarized ourselves with the typical SysID workflow, explored its different tasks, and looked at different ML models available at our disposal for model identification. We also explored the application areas in process industry where ML has proved useful. In the next chapter we will take the first step and learn about the environment we will use to execute our Python scripts containing SysID code.

[6] Attributed to the famous statistician George E. P. Box. It basically implies that your (SysID) model will seldom exactly represent the real process. However, it can be close enough to be useful for practical purposes.

`

Chapter 2

The Scripting Environment

In the previous chapter we studied the various aspects of system identification and learned about its different uses in process industry. In this chapter we will quickly familiarize ourselves with the Python language and the scripting environment that we will use to write ML codes, execute them, and see results. This chapter won't make you an expert in Python but will give you enough understanding of the language to get you started and help understand the several in-chapter code implementations in the upcoming chapters. If you already know the basics of Python, have a preferred code editor, and know the general structure of a typical SysID script, then you can skip to Chapter 3.

If you skim through the system identification literature, you will find almost exclusive usage of MATLAB software as the computing environment. This is mostly attributed to the System Identification toolbox, a very powerful MATLAB toolbox developed by Prof. Lennart Jung (a legend in system identification). Unfortunately, this tool not freely available, and MATLAB is not yet as popular as Python among the ML community. Luckily, several good souls in the Python community have developed specialized libraries for all aspects of SysID. Most of the popular SysID models can now be generated using off-the-shelf Python libraries. Considering the dominance of Python for deep learning, Python becomes an excellent choice for SysID scripting.

In the above context, we will cover the following topics to familiarize you to Python

- Introduction to Python language
- Introduction to Spyder and Jupyter, two popular code editors
- Overview of Python data structures and scientific computing libraries
- Python libraries for system identification
- Overview of a typical ML-DPM/SysID script

`

2.1 Introduction to Python

Python is a high-level general-purpose computer programming language that can be used for application development and scientific computing. If you have used other computer languages like Visual Basic, C#, C++, Java, then you would understand the fact that Python is an interpreted and dynamic language. If not, then think of Python as just another name in the list of computer languages. What is more important is that Python offers several features that sets it apart from the rest of the pack making it the most preferred language for machine learning. Figure 2.1 lists some of these features. Python provides all tools to conveniently carry out all steps of an ML-DPM project, namely, data collection, data exploration, data pre-processing, model ID, visualization, and solution deployment to end-users. In addition, freely available tools make writing Python code very easy[7].

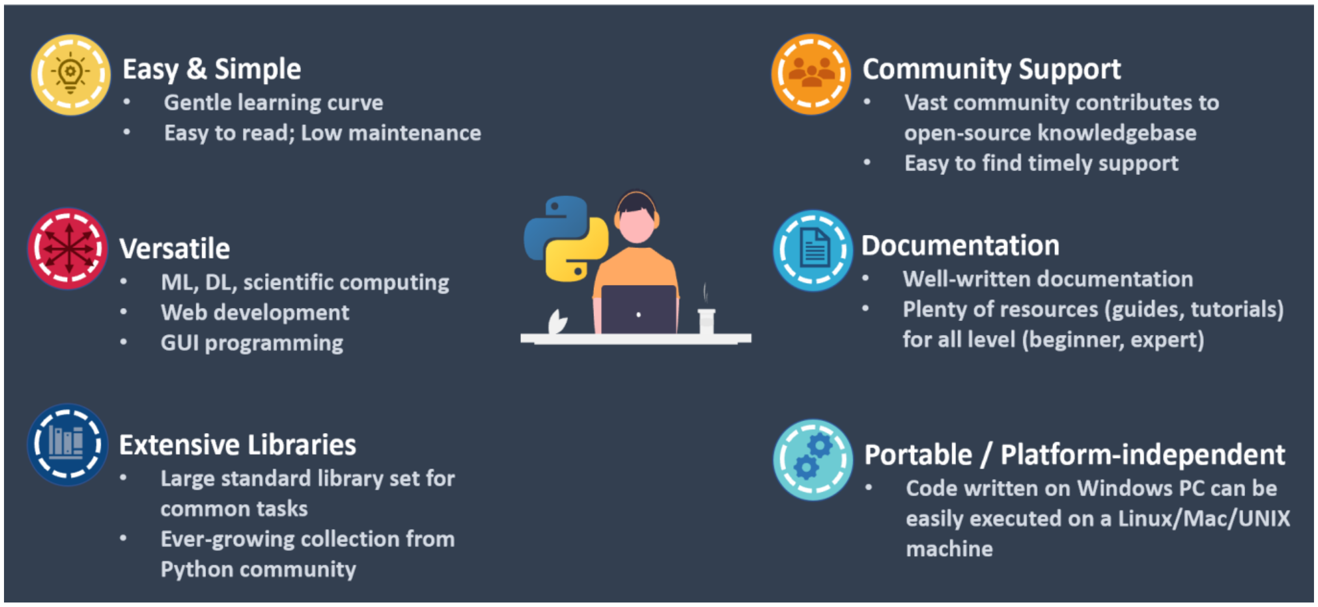

Easy & Simple
• Gentle learning curve
• Easy to read; Low maintenance
Community Support
• Vast community contributes to open-source knowledgebase
• Easy to find timely support
Versatile
• ML, DL, scientific computing
• Web development
• GUI programming
Documentation
• Well-written documentation
• Plenty of resources (guides, tutorials) for all level (beginner, expert)
Extensive Libraries
• Large standard library set for common tasks
• Ever-growing collection from Python community
Portable / Platform-independent
• Code written on Windows PC can be easily executed on a Linux/Mac/UNIX machine

Figure 2.1: Features contributing to Python language's popularity

Installing Python

One can download official and latest version of Python from the *python.com* website. However, the most convenient way to install and use Python is to install Anaconda (*www.anaconda.com*) which is an open-source distribution of Python. Along with the core Python, Anaconda installs a lot of other useful packages. Anaconda comes with a GUI called Anaconda Navigator (Figure 2.2) from where you can launch several other tools.

[7] Most of the content of this chapter is similar to that in Chapter 2 of the book 'Machine Learning in Python for Process Systems Engineering' and have been re-produced with appropriate changes to maintain the standalone nature of this book.

`

`

Chapter 3

Exploratory Analysis and Visualization of Dynamic Dataset: Graphical Tools

In Chapter 1 we had remarked that system identification is an art. A part of the art lies in making right inferences from exploratory analysis of data and model residuals via visual plots. Yeah, you read that right: even with all the fancy algorithms at our disposal, sometimes visual inspection of data is still the best tool for a quick assessment of dataset and model validity. Visual plotting can often provide crucial clues about model structure and model troubleshooting, and forms an important component of deductive phase of SysID. These plots will be used repeatedly in the in-chapter illustrations in this book and therefore we thought it would be best to introduce the concepts behind these graphical tools right away.

Visual plots can range from simple time plots to advanced spectral density plots. These may help us make some quick educated judgement about data stationarity, deterministic vs stochastic trends, presence of colored noise, whiteness of model residuals, etc. Don't worry if you don't understand these dynamic modeling-specific jargons for now. We will focus on generation and conceptual understanding of these plots in this chapter and learn inference-making using these plots in the later chapters.

Specifically, we will cover these topics

- Autocorrelation and autocovariance plots
- Partial autocorrelation plots
- Cross-correlation and cross-covariance plots
- Spectrum, spectral density plots, periodogram

`

3.1 Visual Plots: Simple Yet Powerful Tools

In system identification, careful scrutiny of data plots forms a critical component right from the initial step of exploratory data analysis (EDA) to the last step of model validation. Consider plots *a* and *b* in Figures 3.1 which are time plots of output signals in two different scenarios. Just a visual inspection makes it apparent that the raw output signals contain trends. One can further infer that while plot *a* exhibits a deterministic trend, plot *b* exhibits a stochastic trend[8]. Such assessment during EDA can help a process modeler select appropriate class of models for their systems.

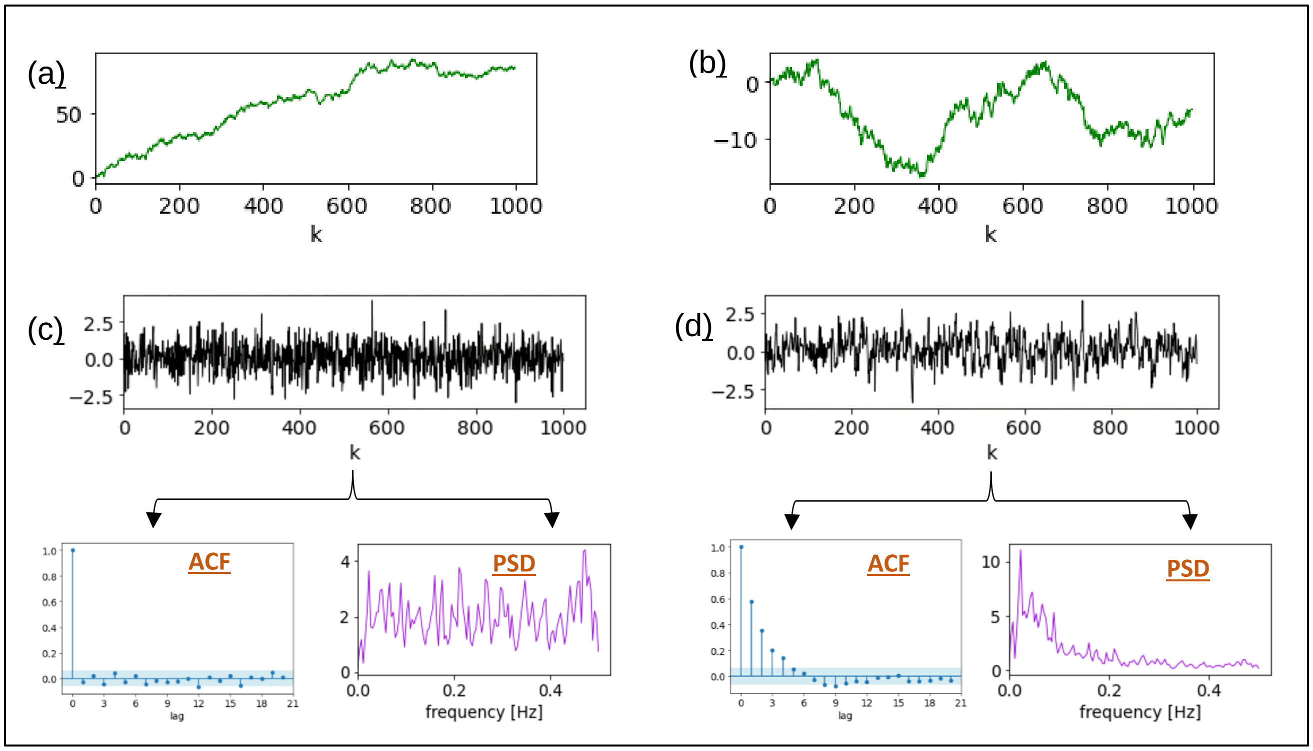


Figure 3.1: Visual plots for time series data

Consider further the plots 3.1*c* and *d*. Here, the shown noisy signals could correspond to the disturbance signals or the model residuals (difference between observations and model predictions). Here, unlike the previous example, the time plots don't provide any quick obvious assessment. However, the derived visualizations, autocorrelation and spectral density plots make the distinction between the two scenarios very evident. These plots confirm that the shown signals have white noise and colored noise properties, respectively. If these were model residuals, then non-zero autocorrelations in plot 3.1*d* immediately suggest inadequate modeling. Moreover, the spectral density plot looks qualitatively similar to that of an autoregressive (AR) process and therefore, an AR model could be attempted for the colored

[8] We will see the details on the distinction between these two types of trends in Chapter 4.

`

.

`

Chapter 4

Machine Learning-based Dynamic Modeling: Workflow and Best Practices

In Figure 1.4 in Chapter 1 we looked at a typical workflow for system identification. What is noteworthy is that while model ID is only a part of this workflow, very often too much focus goes into application of model ID to estimate model parameters at the expense of other aspects of SysID. Dumping raw data into model ID module will invariably generate unsatisfactory model if data is not pre-treated appropriately. Additionally, you will seldom be right in your modeling choices in the very first attempt. The take-home message is that knowledge about how to prepare raw data to enhance its information content about the system, how to assess validity of obtained model, how to analyze modeling results critically, and how to iterate judiciously is absolutely critical for successful SysID. Fortunately, several guidelines and best practices have been devised to guide a process modeler at each step of the SysID workflow. We will learn these guidelines and best practices in this chapter.

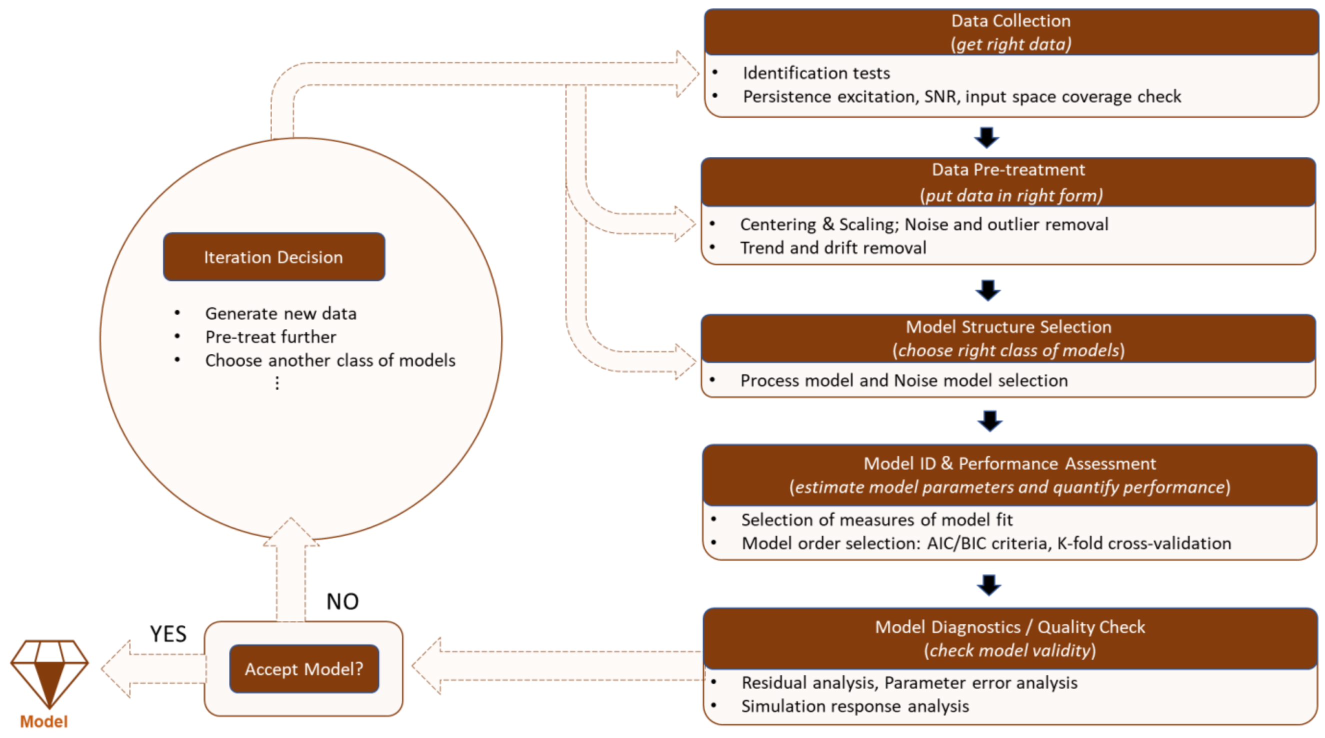
We will not cover the best practices associated with generic machine learning workflow. Concepts like feature extraction, feature engineering, cross-validation, regularization, etc. have already been covered in detail in our first book of the series. In this chapter our focus will be on the unique challenges presented by dynamic processes and the specific best practices to deal with them. Specifically, the following topics are covered

- Identification test design using PRBS and GBN signals
- Pre-treatment of raw data for removal of measurement noise, offsets, trends, and drifts
- Guidelines around selection of model structure
- Model order selection via AIC and cross-validation
- Model quality assessment via residual analysis, simulation response analysis, etc.

`

4.1 System Identification Workflow

So far in this book we have been consciously emphasizing the difference between model identification and system identification. While model ID is all about estimating model parameters, SysID is about ensuring that the model truly represents the underlying system and stands consistent w.r.t. any prior available knowledge and any modeling assumptions. Figure 4.1 reproduces the SysID workflow we had seen in Chapter 1 and shows the several steps that serve to achieve the stated goals. The rest of the chapter will sensitize you about these varied aspects of SysID and help you gain working-level understanding about how to successfully execute a SysID project.



Data Collection
(get right data)
• Identification tests
• Persistence excitation, SNR, input space coverage check
Data Pre-treatment
(put data in right form)
• Centering & Scaling; Noise and outlier removal
• Trend and drift removal
Model Structure Selection
(choose right class of models)
• Process model and Noise model selection
Model ID & Performance Assessment
(estimate model parameters and quantify performance)
• Selection of measures of model fit
• Model order selection: AIC/BIC criteria, K-fold cross-validation
Model Diagnostics / Quality Check
(check model validity)
• Residual analysis, Parameter error analysis
• Simulation response analysis
Iteration Decision
• Generate new data
• Pre-treat further
• Choose another class of models
⋮
NO
YES
Accept Model?
Model

Figure 4.1: Typical steps in a SysID workflow that we will cover in this chapter

The first step of the workflow is data collection. Whether you are using pre-existing historical data or conducting fresh identification tests, you must ensure that the dataset has enough ‘juice’ in it to enable identification of the desired model. We will soon see some guidelines around this task. The next step corresponds to sanitizing the data to remove the unwanted components that can negatively impact model ID. For example, elimination of any slow drift in the output signals is essential to avoid incorrect model parameters. After the first two steps, our dataset is ready and we come to model structure selection where the onus is on the process modeler to choose the right class of process and noise models. We saw in Chapter 1 that an incorrect choice can lead to biased parameters estimates. Your expert domain knowledge and any prior information on process disturbances can come in quite handy at this

`

`

Part 2

Classical Machine Learning Methods for Dynamic Modeling

`

Chapter 5

Time Series Analysis: Concepts and Applications

As alluded to before, time series analysis (TSA) refers to study of time series signals from processes without exogenous inputs. Before we dive into the relatively more complex world of input-output modeling, it would be prudent to first get acquainted with how to model stochastic variations in signals and compactly describe the dependencies among adjacent observations in a time series. The study of time series is not just for pedagogical convenience. SysID derives many concepts from time series analysis and therefore, a strong foundation in modeling time series is important for mastering the art of SysID.

Time series analysis can help us to characterize I/O model residuals and get clues regarding further model refinement. If your process has measured disturbance signals (inputs that can't be manipulated; for example, ambient temperature), then they can be modeled via TSA to provide better forecasts and control. Furthermore, TSA can be used to model controlled process variables and build process monitoring tools. In this chapter, we will work through case-studies illustrating these applications.

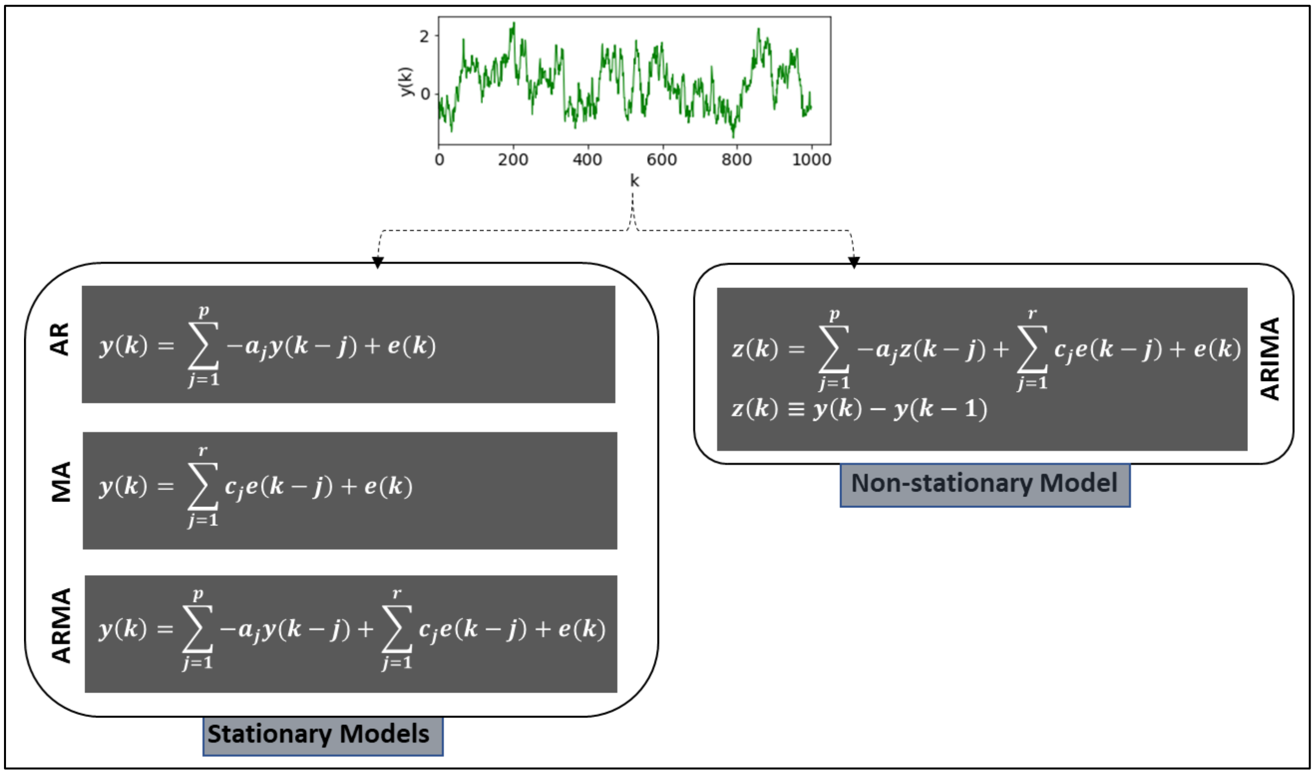
Apart from setting a strong foundation of digital signal processing, this chapter will also introduce the backshift notation and transfer function operator. These constructs are quite useful for compact description and algebraic manipulation of the difference equations. Specifically, the following topics are covered

- Introduction to AR, MA, and ARMA models for stationary signals
- Using ACF and PACF for model structure selection
- Monitoring controlled process variable in a CSTR using ARMA models
- Introduction to ARIMA models for nonstationary signals
- Forecasting measured signals using ARIMA models

`

5.1 Time Series Analysis: An Introduction

Time series analysis refers to the study and modeling of dependencies among sequential data points in time series signals. Several approaches exist for modeling time series. In this chapter, we will look at four of the most common models as shown in Figure 5.1. While AR, MA, and ARMA models are used to describe stationary signals, ARIMA is used for (homogeneous) nonstationary signals. In this chapter, we will understand the differences, similarities, and the inherent connections between these models.



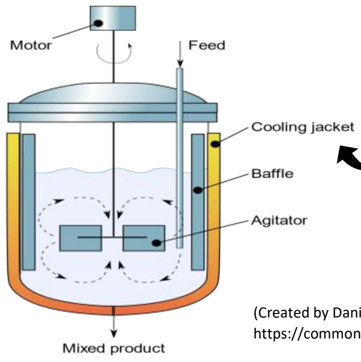
y(k)
k
AR
$y(k) = \sum_{j=1}^{p} -a_j y(k-j) + e(k)$
MA
$y(k) = \sum_{j=1}^{r} c_j e(k-j) + e(k)$
ARMA
$y(k) = \sum_{j=1}^{p} -a_j y(k-j) + \sum_{j=1}^{r} c_j e(k-j) + e(k)$
Stationary Models
ARIMA
$z(k) = \sum_{j=1}^{p} -a_j z(k-j) + \sum_{j=1}^{r} c_j e(k-j) + e(k)$
$z(k) \equiv y(k) - y(k-1)$
Non-stationary Model

Figure 5.1: Commonly employed univariate time-series models

To further motivate the study of time series analysis, let's briefly look at two of the use cases that were mentioned previously. The illustration below shows a CSTR unit that has measured disturbance signals (feed concentration and temperature). An accurate model for these signals can help generate accurate forecasts and control the vessel temperature more effectively. Also, it is easy to show that if the setpoint of the controllers do not change, then the controlled variables (here, output product temperature and concentration) depend on process disturbances alone. Therefore, a TSA model for these variables can be built to eventually look for significant mismatch between measurements and model predictions as an indicator of process faults.

`

Continuous Stirred Tank Reactor



Cooling jacket temperature is used to control the reactor/product temperature at optimal set-point

(Created by Daniele Pugliesi under Creative Commons Attribution-Share Alike 3.0, https://commons.wikimedia.org/wiki/File:Agitated_vessel.svg)

Rest of the Chapter 5 not shown in this preview

`

Chapter 6

Input-Output Modeling - Part 1: Simple Yet Popular Classical Linear Models

There is beauty is simplicity - this age-old truth perfectly sums up the reason behind the popularity of two simple input-output models, FIR and ARX models, that we will learn about in this chapter. In the age of artificial neural networks, these models still hold on their own and are an essential part of any PDS's toolkit. As alluded to before, I/O processes have inputs signals that can be manipulated to control the output signals. The objective, therefore, of I/O modeling is to find the deterministic relationship between the input and output signals, and, optionally, find a suitable model for the stochastic disturbances affecting the outputs.

FIR model is a non-parametric model and is used in thousands of MPC solutions in process industry globally. The remarkable success of the FIR-based MPCs is testament to the fact that simple models can be quite useful representation of complex industrial processes. FIR models are intuitive and simple enough to be understood by plant operators. However, nothing is all rosy with anything. FIR models, owing to their flexibility, often lead to overfitting. ARX models, on the other hand, are parametric models and require much lower number of parameters. Consequently, the resulting models are 'smoother'. ARX models are often the initial choice in any SysID exercise. Nonetheless, ARX models can easily suffer from high bias errors. Since these two models are so critical, this chapter is devoted to understanding their various aspects, along with their pros and cons, in detail.

Specifically, the following are covered in the chapter.

- Introduction to FIR and ARX models
- SysID of industrial furnaces using FIR and ARX models
- Hyperparameter selection for FIR and ARX models
- Stochastic component of ARX models
- Model bias in FIR and ARX models

`

6.1 FIR Models: An Introduction

Impulse response of a system is its response to a special impulse input as shown in Figure 6.1. The impulse response model approximates the system output as a linear combination of several impulse responses due to past inputs. Since the impulse responses usually decay to zero, it suffices to include only the past few values (say *M*) of *u(k)* to approximate *y(k)*. The value '*M*' is called the model order and the resulting model is called the finite impulse response (FIR) model. You will notice that there is no attempt to model stochastic disturbances and a non-parametric modeling form (there is no compact mathematical structure) is adopted. Consequently, FIR model is among the simplest SysID model.

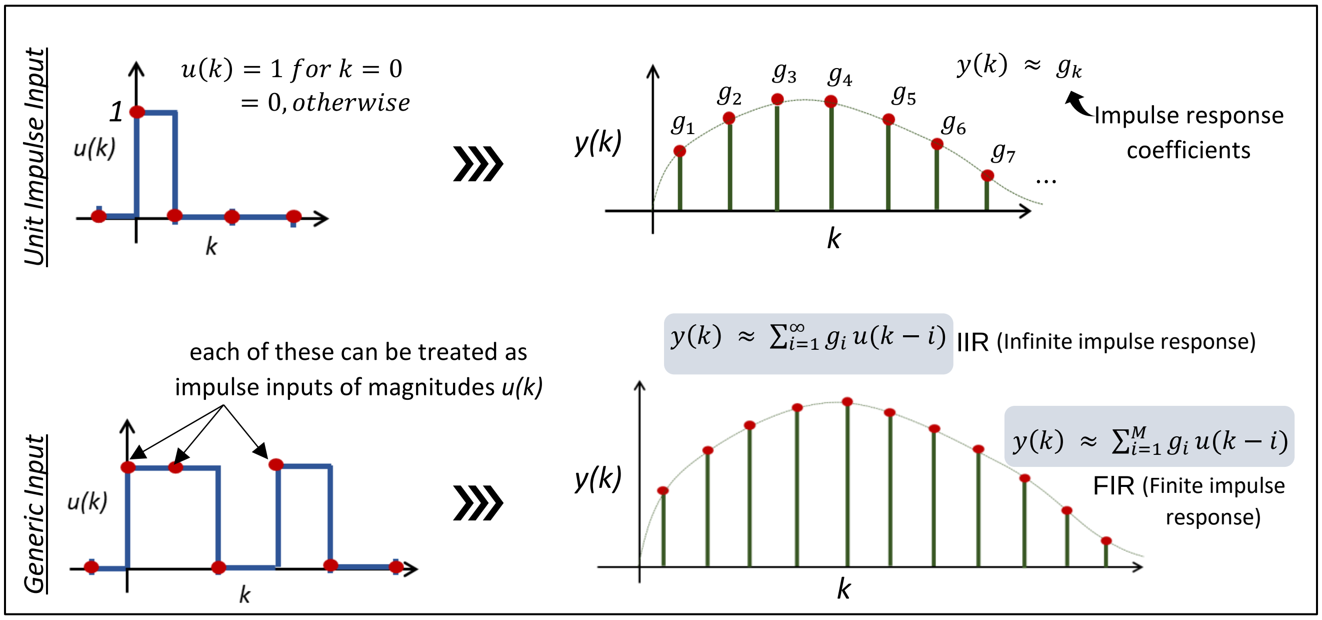

Unit Impulse Input
$u(k) = 1\ for\ k = 0$
$= 0, otherwise$
1
u(k)
k
y(k)
g_1 g_2 g_3 g_4 g_5 g_6 g_7 …
$y(k) \approx g_k$
Impulse response coefficients
k
Generic Input
each of these can be treated as impulse inputs of magnitudes u(k)
u(k)
k
$y(k) \approx \sum_{i=1}^{\infty} g_i\, u(k-i)$ IIR (Infinite impulse response)
y(k)
$y(k) \approx \sum_{i=1}^{M} g_i\, u(k-i)$
FIR (Finite impulse response)
k

Figure 6.1: Impulse response model for a SISO system

Fitting a FIR model entails estimation of the impulse response coefficients. Although a FIR model's structure is quite simple, it is very flexible and given sufficiently large *M*, any type of complex impulse response can be fitted. It is common to employ model orders of 90 to 120 to get all the impulse coefficients. However, such large parameters dimensionality often leads to overfitting (impulse response curves show wiggles or spikes due to output signal corruption by process disturbances). Apart from model flexibility, another advantage of FIR model is that an estimate of I/O delay is automatically obtained after model fitting. For a system with delay *d*, the first *d* impulse coefficients will be zero or close to zero. This is in fact a classical method for delay estimation. FIR models also provide unbiased parameter estimates as the deterministic part of the model is independently parametrized (recall our discussion from Chapter 4). These favorable properties of FIR models make them the dominant SysID model deployed in industrial MPCs[9].

[9] Darby and Nikolaou, MPC: Current practice and challenges. *Control Engineering Practice*, 2012

`

`

6.2 FIR Modeling of Industrial Furnaces

To showcase an industrial application of FIR models, we will consider fired heaters used in petroleum refineries wherein, as shown in the figure below, fuel is combusted to heat feed oil and vaporize it for subsequent fractionation[10]. In the SISO version of the system, we will consider the task of keeping the outlet temperature of the heated oil (*TO*) in control. Changes in feed oil temperature, *TI*, or the feed flow will impact *TO*. The controller adjusts the flow of the fuel gas (although indirectly, by manipulating the set-point of the flue-gas PID controller. The PID controller, not shown in the schematic, adjusts the fuel-gas stream's valve opening to meet the specified flow)[11]. The process controller, however, needs a model to understand the relationship between flue gas flow set-point (FGSP) and *TO*.

To obtain the model, we will use data provided in the file *IndustrialFiredHeater_SISO.csv* which contains 1000 samples of TO and FGSP obtained from an open-loop simulation of the system wherein FGSP was excited using a GBN signal and the system was subjected to disturbances in *TI*.[12] Note that the time interval here is in minutes and not seconds.

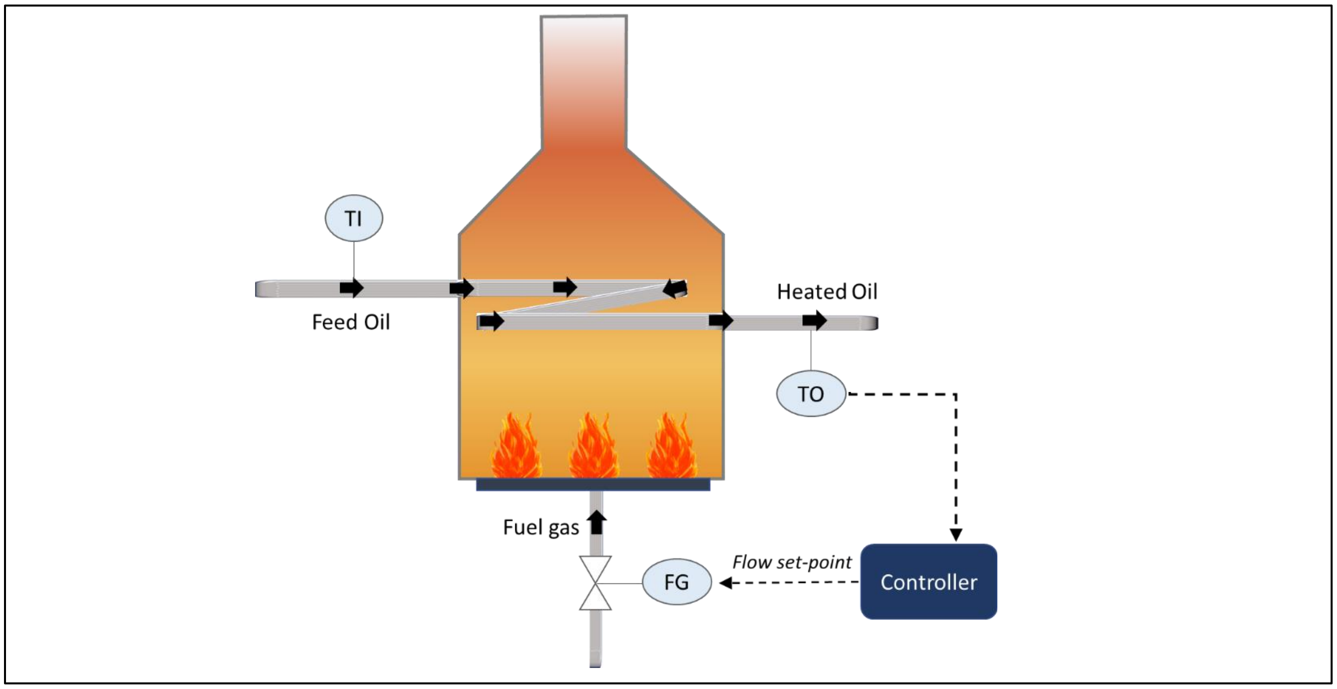


Figure 6.2: Representative schematic of a fired heater used in petroleum refineries

[10] The reader is referred to https://apmonitor.com/dde/index.php/Main/FiredHeaterSimulation for more details on the system. At this link, readers can also find the system model that is used to generate the simulation dataset used in this case-study.
[11] Using PID controller's setpoints as manipulated variables is a very standard approach in modern (MPC) controllers.
[12] Online code repository shows how the identification test data was generated

`

`

Chapter 7

Input-Output Modeling - Part 2: Handling Process Noise the Right Way

In the previous chapter we saw that the FIR and high-order models can provide unbiased and linear-in-parameter models, but, unfortunately, suffer from high variance errors due to large number of parameters. Therefore, in this chapter, we will look at other popular model structures, namely, ARMAX, OE, and Box-Jenkins, that can provide more parsimonious description of the underlying process and generate unbiased models. These models allow greater flexibility in noise dynamics and capture the fact that noise may enter (or originate in) the system at different points which, in turn, determines the structure of the noise transfer operator. However, there is no free lunch! These models generate non-linear-in-parameter predictors and therefore the parameter estimation procedures are computationally more complex. Nonetheless, addition of these models to your ML-DPM toolkit will allow you to choose model structure specific to the problem at hand.

In the later part of this chapter, we will relax the stationarity conditions on disturbance signals and look at how to model processes with non-stationary disturbances. We will see how differencing input and output signals helps in removing disturbance non-stationarities. Using simulated illustrations, we will try to understand the implications of choosing these model structures. Specifically, we will study the following topics.

- Introduction to ARMAX and Box-Jenkins models
- Modeling distillation columns using ARMAX models
- Introduction to OE model structure and its comparison to ARX model structure
- Introduction to ARIMAX models
- Modeling gas furnace systems using Box-Jenkins models

`

7.1 PEM Models

In Chapter 6, we saw how the ARX model provides a very restrictive treatment of the process disturbances. We also saw how this shortcoming can lead to significant model bias. Accordingly, the focus in this chapter is to overcome this shortcoming and look at other popular alternatives to ARX model, namely, the output error (OE) model, the autoregressive moving average with eXogenous (ARMAX) inputs model, and the Box-Jenkins (BJ) model. As shown in Figure 7.1, these parametric models provide a broad spectrum of options regarding the parametrization of the noise transfer operator (*H*). For example, while disturbances are not modeled and only input transfer operator is sought in OE framework, completely independent parametrization of both *G* and *H* operators are allowed in the BJ model. In the next few sections, we will explore these models' properties and learn how to use them judiciously.

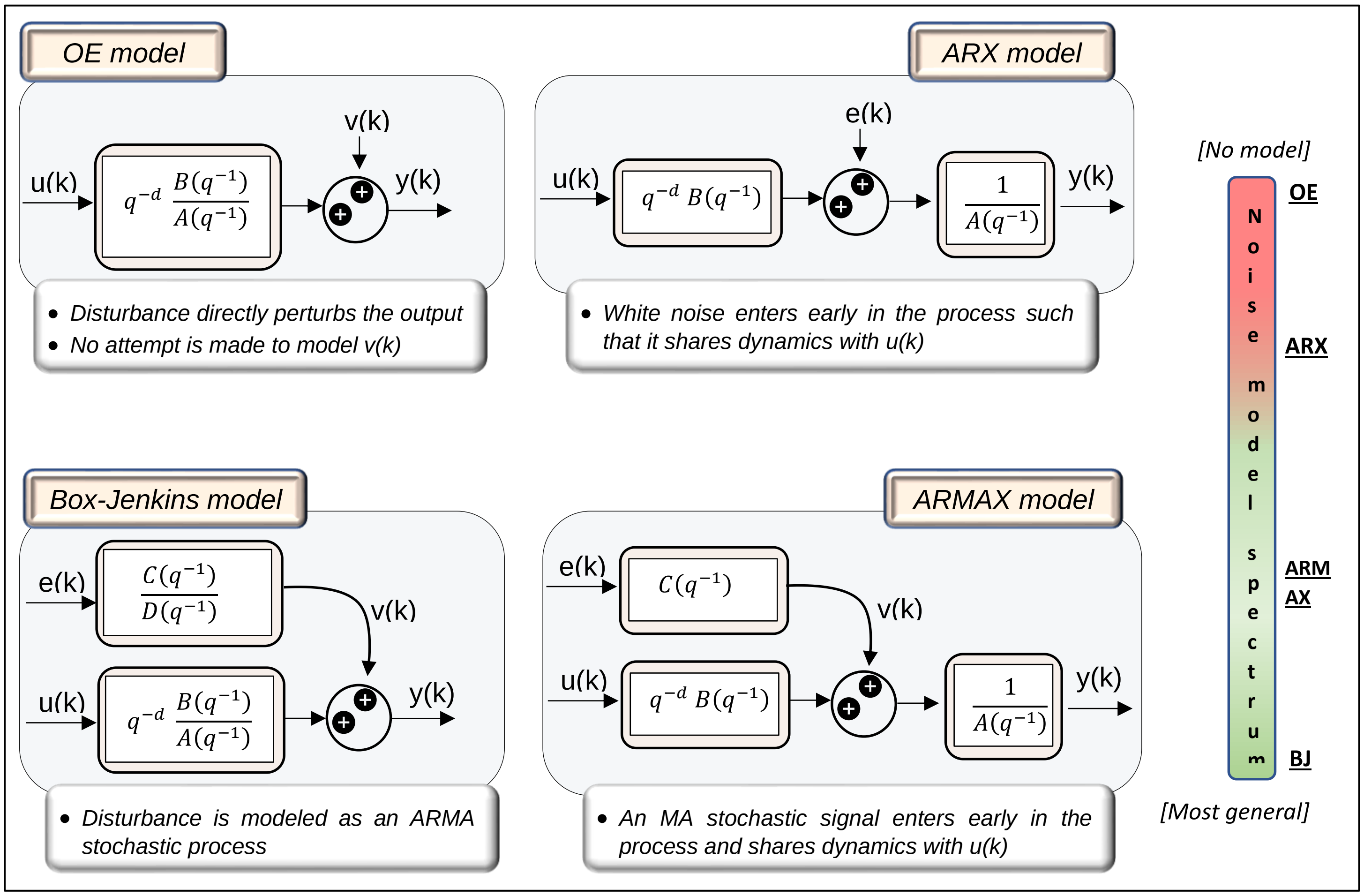

OE model
u(k)
$q^{-d}\frac{B(q^{-1})}{A(q^{-1})}$
v(k)
y(k)
Disturbance directly perturbs the output
No attempt is made to model v(k)
ARX model
e(k)
$q^{-d}B(q^{-1})$
$\frac{1}{A(q^{-1})}$
White noise enters early in the process such that it shares dynamics with u(k)
Box-Jenkins model
$\frac{C(q^{-1})}{D(q^{-1})}$
Disturbance is modeled as an ARMA stochastic process
ARMAX model
$C(q^{-1})$
An MA stochastic signal enters early in the process and shares dynamics with u(k)
[No model]
Noise model spectrum
OE
ARX
ARMAX
BJ
[Most general]

Figure 7.1: Model structure of PEM models

All the models in Figure 7.1 belong to the family of prediction error methods wherein the model parameters are estimated by minimizing the one-step ahead prediction errors (*ε(k))* where,

`

`

7.3 ARMAX Modeling of Distillation Columns

To showcase an industrial application of ARMAX models, we will use a simulated dataset from a distillation column. The ubiquity and importance of distillation columns in process industry cannot be overstated. Figure 7.3 shows a schematic of one such column wherein the incoming feed is separated into a top product (distillate) and a bottoms product. Strict control of the product flows and concentrations is critical for optimal column operations and the heating power (*QB*) supplied to the reboiler which drives the vapor flow in the column is a commonly used manipulated variable. Therefore, we will attempt to build a dynamic model between the distillate concentration (X_D) and *QB* to help the column controller operate the unit efficiently. To obtain the model, we will use data provided in the file *DistillationColumn_SNR10.csv* which contains 1000 samples of *QB* and X_D obtained from an open-loop simulation of the system wherein *QB* was excited using a GBN signal.[13]

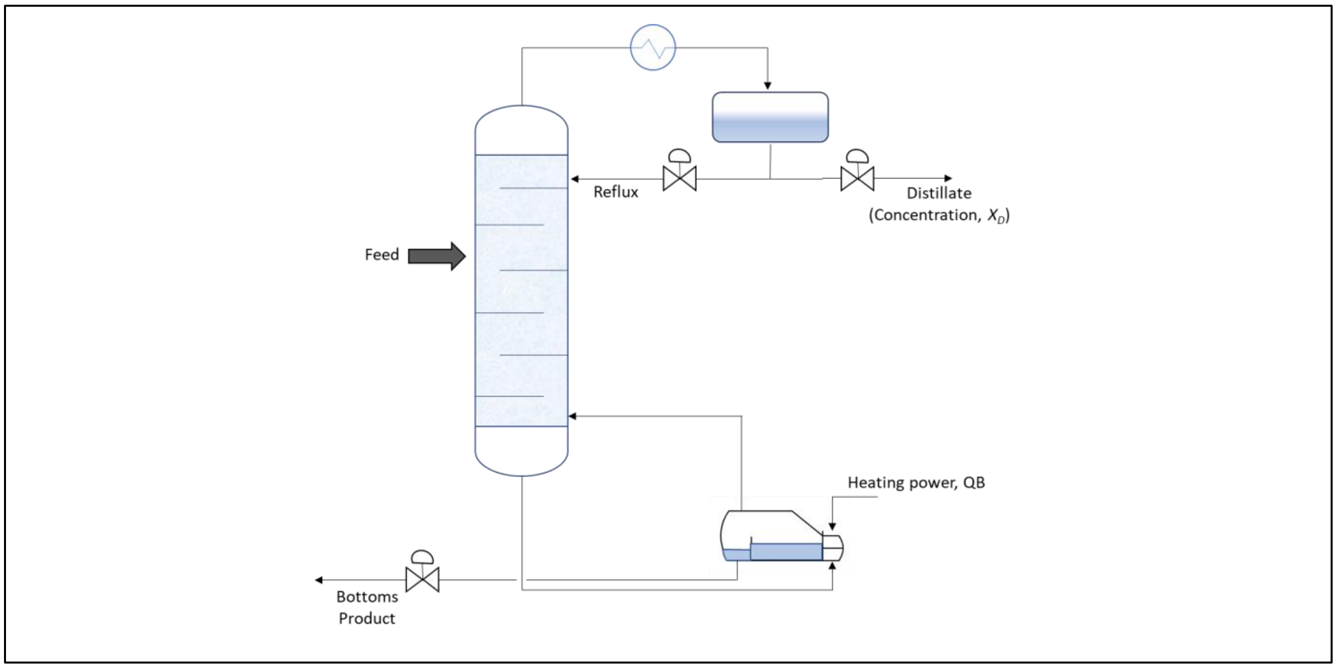


Figure 7.3: Representative schematic of a distillation column used in process industry

Let's start by importing the required packages and exploring the provided I/O dataset.

[13] The model for the distillation column was taken from '*Digital control systems: design, identification and implementation. Springer, 2006*'.

`

`

Chapter 8

State-Space Models: Efficient Modeling of MIMO Systems

In the previous chapter we alluded to the problems related to the classical methodology of modeling a MIMO system through separate MISO models for each output. In a physical system, the different output variables seldom behave independently; they often share common parameters and/or correlated noise. Therefore, identifying all the output models simultaneously while being cognizant of shared dynamics between them leads to better and more robust models. This alternative approach to SysID is provided by state-space (SS) models. State-space models add an intermediate layer of ‘state variables’ between the input and output variables, wherein, the inputs determine the states, and the outputs are derived as linear combination of the states. This modeling paradigm leads to parsimonious models capable of fitting complex processes with both fast and slow dynamics.

In this chapter, we will cover in detail the subspace identification techniques which are utilized for fitting the SS models. While several popular subspace identification methods (SIMs) such as N4SID, CVA, MOESP exist, CVA will be our primary focus. Subspace identification offers several additional advantages over I/O models. SIMs involve only linear algebra for model ID (no iterative nonlinear optimization schemes are needed). SIMs also provide in-situ mechanism for automated optimal model order selection. Subspace models are more robust to noise and provide smooth (step and impulse) response curves. Infact, it is not uncommon to fit SIM model to I/O data and then derive FIR model using the SIM model for usage in MPC applications. Due to such superior properties of SIM, they are among the ‘mainstream’ models provided by advanced process control (APC) solution vendors.

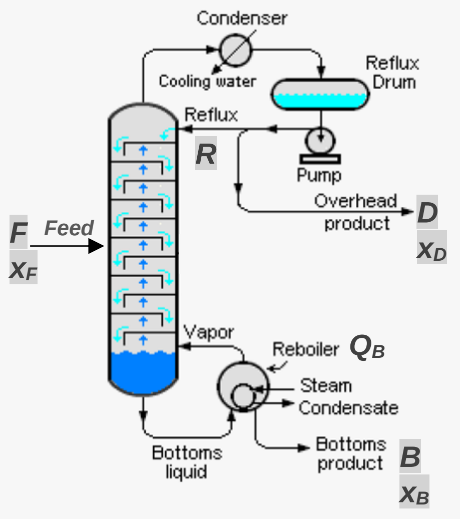
CVA models are also inherently suitable for computing fault detection indices, performing fault diagnosis, and therefore building process monitoring tools. We will build one such application in this chapter. Specifically, the following topics will be covered

- Introduction to state-space models and CVA modeling
- Modeling MIMO glass furnace via CVA
- Process monitoring of Industrial Chemical Plants using CVA

`

8.1 State-Space Models: An Introduction

Consider the distillation column in Figure 8.1 where we have four inputs and four outputs. Let's focus on the relationship between the reboiler duty Q_B and distillate composition x_D. We can build an I/O model between Q_B and x_D, but it is clear that Q_B doesn't affect x_D directly. There are internal dynamics or states (e.g., arising from the liquid holdup at each column tray) that changes in Q_B have to pass through before impacting x_D. Similar argument could be made for Q_B vs D relationship. The unobserved intermediate variables which characterize the internal dynamics or 'states' of the column are termed state variables and, as argued before, the outputs share these states. The modeling framework that reflects such interrelationship between the inputs and outputs is the state-space model whose structure[14] is shown in Figure 8.1. In a generic modeling exercise, the estimated state variables may not always have any physical meanings and even the choice of the states is not unique.



State-space model (or order *n*)

$$x(k+1) = Ax(k) + Bu(k) + w(k)$$

$$y(k) = Cx(k) + Du(k) + v(k)$$

$$u = [Q_B, x_F, F, R]$$

$$y = [D, x_D, B, x_B]$$

$x(k) \in \mathbb{R}^n$: State vector

$u(k) \in \mathbb{R}^m$: Input vector

$y(k) \in \mathbb{R}^r$: Output vector

$w(k) \in \mathbb{R}^n$: Process noise

$v(k) \in \mathbb{R}^r$: Measurement noise

State-space ID objective

Given the I/O data sequences $\{u(k)\} = [u(0), u(1), \cdots, u(N-1)]$ and $\{y(k)\} = [y(0), y(1), \cdots, y(N-1)]$, estimate the system order *n*, the system matrices (*A, B, C, D*), the initial state *x(0)*, and the statistical properties (covariance matrices) of the noises $(\Sigma_{ww}, \Sigma_{vv})$

Figure 8.1: State-space representation of a process[15]

Our familiar PEM methodology could be used to estimate SS models but that would involve solving a nonlinear optimization problem. The better alternative is the subspace identification method.

[14] The matrix *D* is often kept equal to zero due to no immediate direct impact of *u(k)* on *y(k)*

[15] Column diagram created by Milton Beychok under Creative Commons Attribution-Share Alike 3.0 [http://commons.wikimedia.org/wiki/File:Continuous_Binary_Fractional_Distillation.PNG]

`

`

Chapter 9

Nonlinear System Identification: Going Beyond Linear Models

In the previous chapters we restricted our attention to only linear processes. However, most industrial processes exhibit linear behavior only in a limited range of operating conditions. If a global model of complex processes (such as high-purity distillation, pH neutralization, polymerization, etc.) is sought, then the demon of nonlinearity would inadvertently surface. It won't be an exaggeration to state that nonlinearity is not an exception, rather the rule in process industry. We won't be surprised if you are tempted to always resort to artificial neural networks-based structures which have become synonymous with nonlinear modeling now-a-days. However, there exist simple classical nonlinear model structures which are more interpretable and equally powerful. We will explore these models in this chapter.

Nonlinear models can be powerful tools but there are certain costs to their usage. These include greater demand for variability in training data (which means more laborious experiment design), more complex model selection procedure, and more computationally intensive parameter estimation. Therefore, the decision to use nonlinear models should be judiciously made. If you do decide to go ahead, then the concepts covered in this chapter may help you greatly. Specifically, the following topics will be covered

- Introduction to NARX models
- Modeling heat exchanger process using NARX models
- Introduction to block-structured models, specifically, Hammerstein models and Wiener models

`

9.1 Nonlinear System Identification: An Introduction

Nonlinear SysID becomes a necessary evil when accurate predictions are sought over wide range of operating conditions. Although nonlinear SysID literature is not as mature as linear SysID, SysID community has developed several useful approaches for nonlinear modeling. Figure 9.1 shows a few popular approaches. In the 1st approach, a nonlinear relationship is directly fitted between *y(k)* and past data The nonlinear form could be polynomial-based or more complex such as neural network-based. We will focus on the polynomial form in this chapter. Just like linear SysID, different model structures can be postulated, namely, nonlinear ARX (NARX), NOE, NARMAX, etc. In the 2nd approach, the linear and nonlinear parts are defined in separate blocks, leading to block-structured models. Depending upon whether the nonlinear block comes before or after the linear block, one could have the Hammerstein model or the Wiener model (the two most popular models from this class). In the 3rd approach, several linear models are built for different regions of the operating space and then they are combined to give one global model. Numerous industrial applications of these approaches have been reported for predictive modeling, process control, and process monitoring.

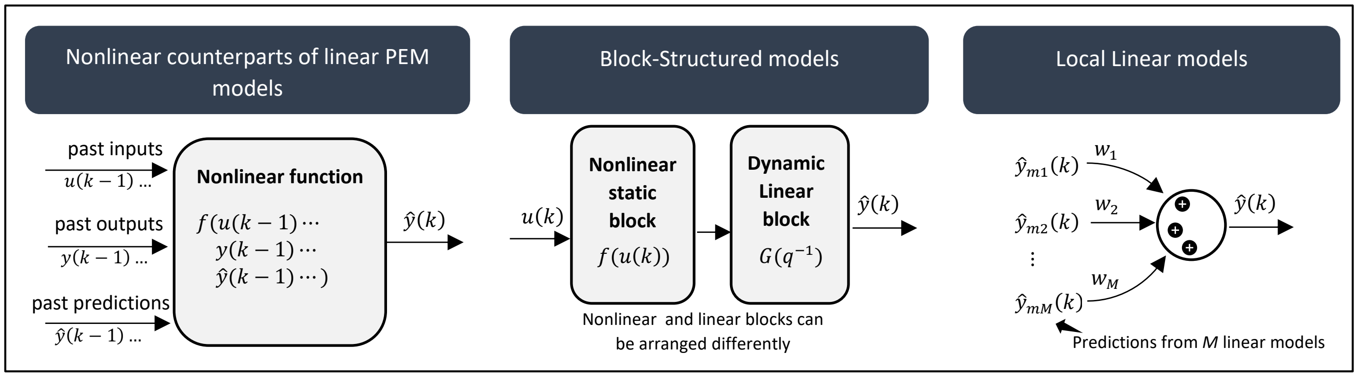
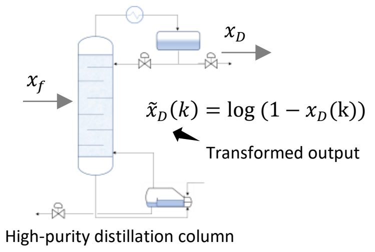
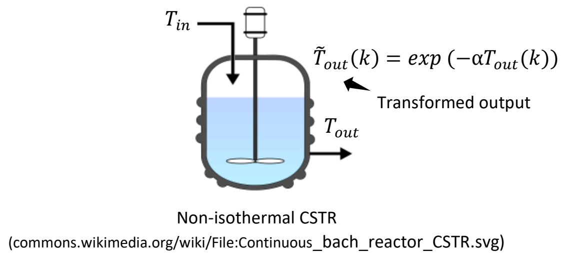


Figure 9.1: Popular classical approaches for nonlinear system identification

Nonlinear transformation

Considering high computational burden of nonlinear SysID, if the nature of nonlinearity is known beforehand, then a nonlinear transformation of the input or output variables may be utilized to avoid explicit nonlinear fitting. Illustrations below provide some example scenarios.


x_D
x_f
$\tilde{x}_D(k) = \log(1 - x_D(k))$
Transformed output
High-purity distillation column


T_{in}
$\tilde{T}_{out}(k) = exp(-\alpha T_{out}(k))$
Transformed output
T_{out}
Non-isothermal CSTR
(commons.wikimedia.org/wiki/File:Continuous_bach_reactor_CSTR.svg)

`

`

9.3 Nonlinear Identification of a Heat Exchanger Process Using NARX

To demonstrate the utility of nonlinear SysID, we will use data from a liquid-saturated steam heat exchanger[16] where hot saturated steam is used to heat cold liquid. The provided dataset contains 4000 samples (sampling time 1s) of input (liquid flow rate) and output (outlet liquid temperature) variables. For this benchmark nonlinear process, our objective will be to build a dynamic model to predict the outlet liquid temperature as a function of incoming liquid flowrate. The dataset can be downloaded from the DaISy datasets repository.

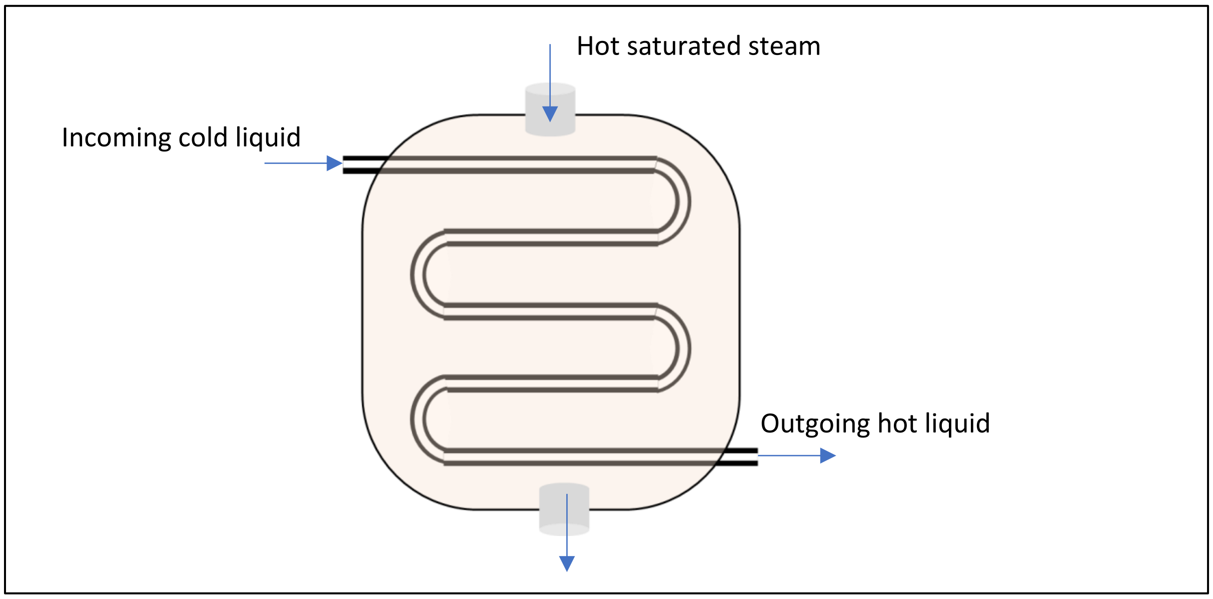


Figure 9.3: Liquid-saturated steam heat exchanger

We will use the SysIdentPy package for this nonlinear SysID exercise. Let's begin by exploring the dataset.

[16] De Moor B.L.R. (ed.), DaISy: Database for the Identification of Systems, Department of Electrical Engineering, ESAT/STADIUS, KU Leuven, Belgium, URL: http://homes.esat.kuleuven.be/~smc/daisy/

`

`

Part 3

Artificial Neural Networks & Deep Learning

`

Chapter 10

Artificial Neural Networks: Handling Complex Nonlinear Systems

It won't be an exaggeration to say that artificial neural networks (ANNs) are currently the most powerful modeling construct for describing generic nonlinear processes. ANNs can capture any kind of complex nonlinearities, don't impose any specific process characteristics, and don't demand specification of process insights prior to model fitting. Furthermore, several recent technical breakthroughs and computational advancements have enabled (deep) ANNs to provide remarkable results for a wide range of problems. Correspondingly, ANNs have re(caught) the fascination of data scientists and the process industry is witnessing a surge in successful applications of ML-based process control, predictive maintenance, and inferential modeling involving ANN-based system identification.

For SysID, ANNs are commonly used in NARX framework wherein an ANN model is fitted to find the nonlinear relationship between current output and past input/output measurements. In such applications, the FFNN (feed-forward neural networks) is invariably employed. However, for dynamic modeling, a specialized type of network called RNN (recurrent neural network) exist which is designed to be aware of the temporal nature of dynamic process data. Consequently, RNNs tend to be more parsimonious than FFNN. We will see SysID applications of both these popular architectures in this chapter.

There is no doubt that ANNs have proven to be monstrously powerful. However, it is not easy to tame this monster. If the model hyperparameters are not set judiciously, it is very easy to end up with disappointing results. The reader is referred to Part 3 of Book 1 of this series for a detailed exposition on ANN training strategies and different facets of ANN models. In this chapter, SysID relevant concepts are sufficiently described to enable an uninitiated reader to learn how to use ANNs. Specifically, the following topics are covered

- Introduction to ANN, FFNN, RNN, LSTM
- Heat exchanger SysID via FFNN-based NARX
- Heat exchanger SysID via LSTM

`

10.1 ANN: An Introduction

Artificial neural networks (ANNs) are nonlinear empirical models which can capture complex relationships between input-output variables via supervised learning or recognize data patterns via unsupervised learning. Architecturally, ANNs were inspired by human brain and are a complex network of interconnected neurons as shown in Figure 10.1. An ANN consists of an input layer, a series of hidden layers, and an output layer. The basic unit of the network, neuron, accepts a vector of inputs from the source input layer or the previous layer of the network, takes a weighted sum of the inputs, and then performs a nonlinear transformation to produce a single real-valued output. Each hidden layer can contain any number of neurons.

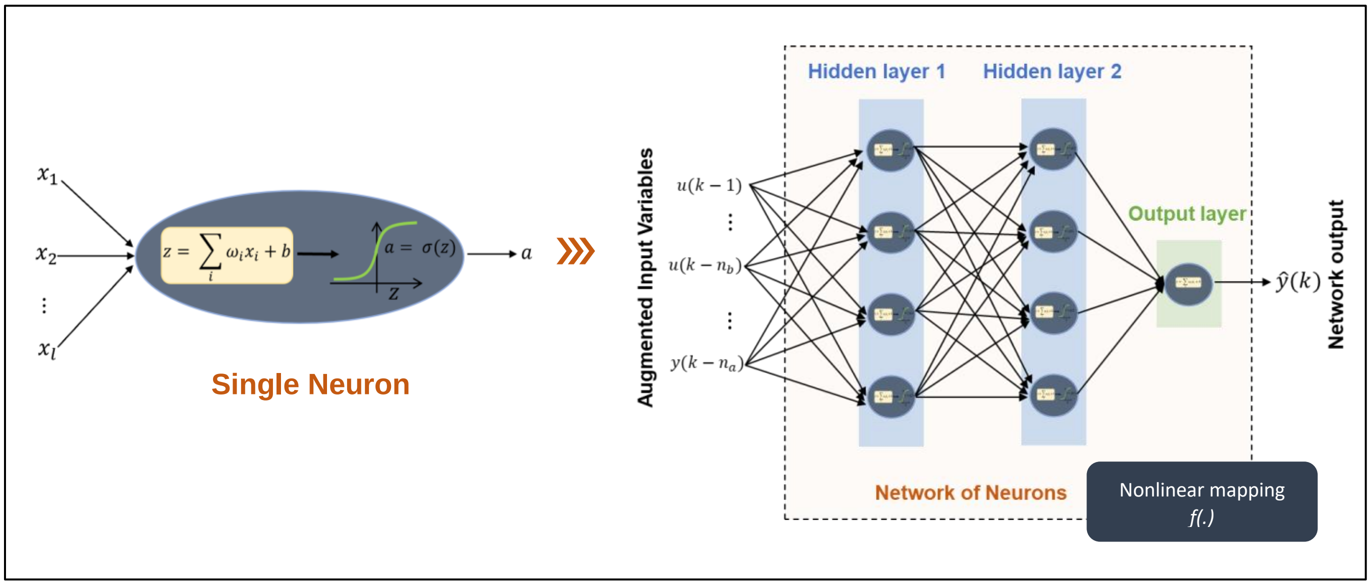


Figure 10.1: Architecture of a single neuron and feedforward neural network with 2 hidden layers for a SISO (single input single output) process within NARX framework

The network shown in Figure 10.1 is an example of a fully-connected feed-forward neural network (FFNN), the most common type of ANN. In FFNN, signals flow in only one direction, from the input layer to the output layer via hidden layers. Neurons between consecutive layers are connected fully pairwise and neurons within a layer are not connected.

What is deep learning

In a nutshell, using an ANN with a large number of hidden layers to find relationship/pattern in data is deep learning (technically, ≥ 2 *hidden layers implies a deep neural network (DNN)). Several recent algorithmic innovations have overcome the model training issues for DNNs which have resulted in the DNN-led AI revolution we are witnessing today.*

`

`

By the process data scientists, for the process data scientists, of process data science

Machine Learning in Python for Dynamic Process Systems

This book is designed to help readers gain a working-level knowledge of machine learning-based modeling techniques for dynamic processes. Readers can leverage the concepts learned to build advanced solutions for process monitoring, soft sensing, predictive maintenance, and process control for dynamic systems. The application-focused approach of the book is reader friendly and easily digestible to the practicing and aspiring process engineers, and data scientists. Upon completion, readers will be able to confidently navigate the system identification literature and make judicious selection of modeling approaches suitable for their problems.

The following topics are broadly covered:

- *Exploratory analysis of dynamic dataset*
- *Best practices for dynamic modeling*
- *Linear and discrete-time classical parametric and non-parametric models*
- *State-space models for MIMO systems*
- *Nonlinear system identification and closed-loop identification*
- *Neural networks-based dynamic process modeling*

www.MLforPSE.com