State estimation and model-based fault detection in a submerged arc furnace

by
Isabella Kristensen

Thesis presented in fulfilment of the requirements for the degree of Master of Engineering (Chemical Engineering) in the Faculty of Engineering at Stellenbosch University

Supervisor: Prof Tobi Louw
Co-supervisor: Prof Steven Bradshaw

December 2023

DECLARATION

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

December 2023

ABSTRACT

Model-based state estimators use noisy plant measurements and a process model to calculate accurate and timely estimates of the state variables for process monitoring, model-based fault detection, and model predictive control. The aim of this project was to perform model-based fault detection using state estimation in a complex chemical unit operation and compare the model-based fault detection to a data-driven technique under plant-model mismatch. A system observability analysis and fault detectability analysis was first conducted. The performance of the various nonlinear state estimation techniques, namely the extended Kalman filter (EKF), the unscented Kalman filter (UKF), the particle filter (PF), and the moving horizon estimator (MHE), was then assessed, enabling the selection of appropriate state estimation techniques for model-based fault detection. Model-based fault detection was employed using the residuals generated from the state estimators followed by residual evaluation using PCA. The model-based fault detection was compared to data-driven fault detection using PCA on the measurements and the effect of plant-model mismatch on the performance of model-based fault detection was investigated. A submerged arc furnace (SAF) for platinum group metal smelting was used as a case study to apply these techniques.

The state observability analysis found the SAF system to be locally observable and the measured states to have a higher degree of observability than the unmeasured states. Upon implementation of the state estimation algorithms, the least observable states corresponded to states estimates with the largest estimation error. The fault detectability analysis identified all faults investigated to be structurally detectable. Upon implementation of model-based fault detection, it was concluded that the more structurally detectable a fault is, the better the fault detection performance.

The investigation into state estimation in the SAF showed that the EKF, UKF, and PF display good estimation accuracy and fast computation times. The PF showed superior estimation accuracy under low process noise conditions and was selected for model-based fault detection. The EKF, being the most popular algorithm in literature and displaying fairly good estimation accuracy, was selected as the second method. The computational requirements of the MHE proved to be its greatest limitation. Investigations were carried out into reducing the computational load of the method using alternative singular perturbation SAF model with larger integration steps which halved the computational requirements. However, the computation times remained inappropriate for application in model-based fault detection.

Lastly, this study found that the model-based fault detection using the PF residuals outperformed the model-based fault detection using the EKF residuals and the data-driven PCA method for detection of faulty conditions within the SAF process. Due to the sensitivity of the PF residuals resulting from the nature of the algorithm, this method showed exceptionally poor robustness to plant-model mismatch. The investigation then demonstrated that residual evaluation of the PF and EKF residuals in a reduced-dimensional space using PCA improved the classification performance of the method when plant-model mismatch was present. However, when no modelling error is present, the classification of PF and EKF residuals showed the best performance in the original dimension space.

OPSOMMING

Model-gebaseerde toestandberamers gebruik raserige aanlegmetings en ’n prosesmodel om akkurate en tydige beramings van die toestandveranderlikes vir prosesmonitering, model-gebaseerde foutopsporing, en modelvoorspellingsbeheer, te bereken. Die doel van hierdie projek was om model-gebaseerde foutopsporing uit te voer deur toestandberaming in ’n komplekse chemiese eenheidbedryf te gebruik en die model-gebaseerde foutopsporing te vergelyk met ’n data-gedrewe tegniek onder aanleg-model wanverhouding. ’n Sisteem waarnemingsanalise en foutopsporingsanalise is eerste uitgevoer. Die doeltreffendheid van die verskeie nie-liniêre toestand beramingstegnieke, naamlik die uitgebreide Kalman-filer (EKF), die geurlose Kalman-filter (UKF), die partikelfilter (PF), en die bewegende horisonberamer (MHE), is toe geassesseer, wat die keuse van gepaste toestandberamingstegnieke vir model-gebaseerde foutopsporing moontlik gemaak het. Model-gebaseerde foutopsporing is toegepas deur die residu’s gegeneer deur die toestandberamers te gebruik, gevolg deur residu-evaluasie deur PCA. Die model-gebaseerde foutopsporing is vergelyk met data-gedrewe foutopsporing deur PCA te gebruik op mates en die effek van aanleg-model wanverhouding op die doeltreffendheid van model-gebaseerde foutopsporing is ondersoek. ’n Onderdompelde boogoond (SAF) vir platinum-groepmetaalsmelting is gebruik as ’n gevalle studie om hierdie tegnieke toe te pas.

Die toestand waarneembaarheidanalise het gevind dat die SAF-sisteem lokaal waarneembaar is en die gemete toestande ’n hoër graad van waarneembaarheid het as die ongemete toestande. Met implementasie van die toestandberamingsalgoritmes, het die minste waarneembare toestande met die toestandberamings met die grootste beramingsfout ooreengestem. Die foutopsporingsanalise het alle foute ondersoek geïdentifiseer as struktureel opspoorbaar. Met implementasie van model-gebaseerde foutopsporing is dit beslis dat hoe meer struktureel opspoorbaar ’n fout is, hoe beter die foutopspoorbaarheidsdoeltreffendheid.

Die ondersoek in toestandberaming in die SAF het getoon dat die EKF, UKF, en PF goeie beramingakkuraatheid vertoon asook vinnige berekeningstye. Die PF het superior beramingakkuraatheid getoon onder lae geraaskondisies en is gekies vir model-gebaseerde foutopsporing. Die EKF, wat die populêrste algoritme in literatuur is en redelike goeie beramingsakkuraatheid toon, is gekies as die tweede metode. Die berekeningsvereistes van die MHE is bewys as die grootste beperking. Ondersoeke is uitgevoer om die berekeningslading van die metode te verminder deur alternatiewe verspreidingsdinamiek SAF-model met groter integrasie stappe te gebruik, om die berekeningsvereistes te halveer. Die berekeningstye het egter ongepas gebly vir toepassing in model-gebaseerde foutopsporing.

Laastens, hierdie studie het gevind dat die model-gebaseerde foutopsporing wat die PF-residu’s gebruik, beter presteer het as die model-gebaseerde foutopsporing wat EKF-residu’s gebruik en die data-gedrewe PCA-metode vir opsporing van foutiewe kondisies binne die SAF-prosesse. As gevolg van die sensitiwiteit van die PF-residu’s as gevolg van die natuur van die algoritme, het hierdie metode besonder swak robuustheid teenoor aanleg-model wanverhouding. Die ondersoek het dan gedemonstreer dat residuele evaluasie van die PF- en EKF-residu’s in ’n verminderde-dimensionele spasie wat PCA gebruik, die

klassifikasiedoeltreffendheid van die metode verbeter het toe aanleg-model wanverhouding teenwoordig was. Wanneer geen modelleringsfout teenwoordig is nie, het die klassifikasie van PF- en EKF-residu's die beste doeltreffendheid in die oorspronklike dimensie spasie.

ACKNOWLEDGEMENTS

I wish to thank the following people who contributed to the success of this project:

- Professor Tobi Louw for providing invaluable knowledge and always having an open door.
- Professor Steven Bradshaw for your guidance and constant moral support.
- My family, for being my support system throughout this process.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

Symbols		
a	Number of retained principal components	-
A	State transition matrix for continuous-time system	-
B	Control input matrix for continuous-time system	-
C	Measurement matrix for continuous-time system	-
h	Vector of nonlinear state-transition equations	-
e	Reconstruction error	-
F	State transition matrix for discrete-time system	-
G	Control input matrix for discrete-time system	-
h	Vector of nonlinear measurement equations	-
H	Measurement matrix for discrete-time system	-
K	Kalman gain	-
L	Process noise matrix	-
M	Measurement noise matrix	-
n	Number of state variables	-
N	Number of particles in the PF algorithm	-
O	Observability matrix	-
P	State estimation error covariance matrix	-
q	Relative likelihood	-
Q	Process noise covariance matrix	-
R	Measurement noise covariance matrix	-
t	Time point	s
u	Vector of control inputs	-
W	Weighting	-

x	State of a system	-
$\hat{x}$	State estimate	-
y	Output of a system	-
$\hat{y}$	Predicted output	-
Z	Arrival cost approximation	-
Greek symbols		
γ	Innovation	-
δ	Specificity	-
η	Innovation covariance	-
θ	Vector of system parameters	-
μ	Mean	-
σ	Standard deviation	-
τ	Time constant	s
υ	Measurement noise	-
φ	Sensitivity	-
ψ	Precision	-
ω	Process noise	-
Subscripts		
k	Timestep	
i	Particle number	
Superscripts		
i	Sigma point number	
k	Variable condition at timestep k	
$+$	Posterior condition	
$-$	Prior condition	

Acronyms	
AUC	Area under curve
CUMSUM	Cumulative sum
EKF	Extended Kalman filter
FDI	Fault detection and isolation
KF	Kalman filter
KLD	Kullback-Liebler Divergence
MAPE	Mean absolute percentage error
MHE	Moving horizon estimator
MSE	Mean squared error
NOC	Nominal operating conditions
NRMSE	Normalized root mean squared error
ODE	Ordinary differential equation
pAUC	Partial area under the curve
PC	Principal component
PCA	Principal component analysis
pdf	Probability density function
PF	Particle filter
ROC	Receiver operating characteristic
RMSE	Root mean squared error
SAF	Submerged arc furnace
SNR	Signal to noise ratio
SVD	Singular value decomposition
UKF	Unscented Kalman filter
WSSR	Weighted sum of squared residuals

1 INTRODUCTION

This chapter provides a brief introduction to state estimation, fault detection, and the case study of the submerged arc furnace used to employ state estimation and fault detection techniques for this study. The chapter then goes on to present the project motivation, highlighting the importance of state estimation and fault detection within the smelting process and provides the reader with gaps in the literature that parts of this study contribute to. The aims and objectives are then presented, followed by the project scope. Lastly, the layout of the thesis is provided.

1.1 Background

1.1.1 State estimation

The state of any physical system refers to the process variables which completely define the internal conditions of the system (Simon, 2006). In the state-space representation of a system, the states are the minimum set of process variables required to fully define the response of a system to a set of inputs (Rowell, 2004). These state variables appear as time-dependent variables with dynamics explained by differential equations in the system model (Villaverde et al., 2019). State information is used to make judgements on the condition of the system and to enforce control over the states via state-feedback control (Simon, 2006). Chemical and bioprocesses present with their own unique requirements necessitating rapid and accurate information on the state variables.

In chemical processes, accurate state information is essential for implementation of model-based control which predicts future control actions based on information on the current full state of the system (Kumar & Ahmad, 2012). Knowledge of the states can also be used to improve process monitoring and real-time optimization or for model-based fault detection which relies on state information to identify abnormal system behaviour.

When accurate measurements for the full set of state variables are readily available and uncorrupted by noise, state estimation techniques become redundant. However, in industrial processes there often exist unmeasurable states due to the cost of sensors and complexity of measurements. Additionally, the measurements that are available tend to be corrupted by measurement noise or are intermittently accessible due to sensor delay. To enable the implementation of model-based control, process monitoring, or model-based fault detection, access to the full state information is desirable. A group of mathematical techniques known as state estimators can be used to estimate values for unmeasured state variables or used to improve upon state information given by noisy measurements of the state variables.

Model-based state estimation, specifically model-based Bayesian state estimation, is the focus of this study as it is the most widely used method of state estimation for chemical and bioprocesses (Alexander et al., 2020). Model-based methods combine knowledge of the dynamic model with available plant measurements to inform the estimates of the states. Model-based Bayesian state estimation methods assumes that the process is stochastic, where the states and measurements are represented by random variables that change with time (Alexander et al., 2020). The other category of model-based state estimation is deterministic, such as the Luenberger observer and the asymptotic observer, which involve

no randomness. Non-linear state estimation methods are specifically focused on as complex industrial processes most often display nonlinear dynamics. The model-based non-linear Bayesian state estimators that will be the focus of this study are the extended Kalman filter (EKF), the unscented Kalman filter (UKF), the particle filter (PF), and moving horizon estimator (MHE).

The standard Kalman filter (KF) is a model-based Bayesian state estimation technique that is the optimal linear state estimator (Simon, 2006). The KF was first introduced by Rudolf E. Kalman in 1960, providing the first state estimation algorithm with real-time application (Patwardhan et al., 2012). The popularity of the KF is attributed to its simplicity, reduced storage, and reduced computational effort arising from its recursive nature (Jin et al., 2021). As it is the optimal linear filter, application of the standard KF algorithm requires the state transition and measurement models in linearized form. In addition, the algorithm is developed based on the assumption that the state estimate distribution is Gaussian. Lastly, the standard KF algorithm does not enable explicit incorporation of physical constraints on the states (Patwardhan et al., 2012). When the system in question is approximately linear with approximately Gaussian state distributions, the KF is the best performing filter in terms of estimation accuracy (Simon, 2006). In practice, complex industrial processes often exhibit severe nonlinearities and non-Gaussian distributions where the KF accuracy and performance significantly deteriorates. Therefore, alternative filters were developed to handle nonlinearities and non-Gaussian distributions.

The EKF and UKF are nonlinear extensions of the standard KF. The EKF and UKF apply the standard KF algorithm to nonlinear systems by approximating the nonlinear state transition and measurement models. The EKF performs local linearization around the previous state estimate to obtain a linearized model. The UKF approximates the multivariable integrals via an unscented transformation (Daum, 2005). The unscented transformation shows improved estimation accuracy as it has a reduced mean and covariance approximation error compared to the linearization of the EKF (Simon, 2006).

The PF propagates the states through the nonlinear model by random sampling of state vectors known as particles and Monte Carlo integration. The PF then corrects the model prediction with the measurement via a method known as importance sampling coupled with a resampling strategy (Daum, 2005). It does this in such a way that enables the handling of nonlinear process dynamics and measurements as well as non-Gaussian state distributions (Elfring et al., 2021).

The MHE is the most recent of the four state estimation techniques, proposed by Robertson *et al.* in 1996. The MHE achieves state estimation by solving an optimization problem to obtain a horizon of state estimates. The advantage of the MHE over other state estimation techniques arises from its ability to explicitly handle constraints and nonlinear dynamics (Mesbah et al., 2011). However, these advantages come at the cost of significant computational effort, limiting its widespread application (Alexander et al., 2020).

Rapid and accurate estimates of important state variables become accessible upon implementation of these state estimation techniques. These state estimates can then be used for process monitoring, model-based process control or model-based fault detection. Model-based fault detection using state estimators will be investigated further in this study.

1.1.2 Fault detection

Process monitoring and fault detection are essential in any industrial process. Fault detection ensures safe and profitable operation by rapidly alerting operators to adverse process conditions to allow for correction and avoid prolonged operation under sub-optimal conditions. Fault detection methods are often categorized as either model-based methods or process history-based methods.

Process history-based methods construct statistical models from a set of past measurements. Data-driven process monitoring involves assessing the system condition on-line by calculating various monitoring statistics for real-time measurements. The monitoring statistics measure the statistical probability of the new measurement belonging to the distribution defined by nominal or faulty past measurements. These monitoring methods can either involve calculating monitoring statistics for single process variables, univariate methods, or calculating monitoring statistics for multiple process variables, multivariate methods. Principal component analysis (PCA) is one example of a multivariate data-driven method that has been widely used for process monitoring applications. PCA performs feature extraction by constructing latent variables that account for the maximum variance in the original measurements (Abdi & Williams, 2010). This latent variable model can then be used in conjunction with monitoring statistics to assess the health of a system (Kourti, 2002). Data-driven methods require no prior knowledge of the system dynamics, however, they require large sets of past process variable time-series data to adequately train the data-driven models (Qu, 2009).

Model-based fault detection methods also use the current measurements from the process, however, they additionally require fundamental process knowledge in the form of a mathematical model of the system. The process model is used to make predictions for the state variables under nominal conditions and the difference between the predicted values for the state and the current state measurement is known as the residual. Residual generation can be achieved via several methods. One method involves using state estimation where the residual is the difference between the current measurement and the current state estimate. The use of state estimators for the residual generation is common in literature with the EKF being the most popular technique for residual generation (Chang & Chen, 1995; Li & Olson, 1991; Sunil Nag et al., 2015). Following generation of the residual is a residual evaluation procedure involving a decision-making process that classifies residuals as faulty or nominal. The model-based fault detection does not require past measurements as there is no training step in the procedure, however, the need for an accurate model of complex processes limits its applications in industry.

1.1.3 Case study: submerged arc furnace

The chosen case study for application of state estimation and fault detection techniques is a submerged arc furnace (SAF) used for platinum group metal (PGM) smelting. According to Nell (2004), the majority of the worlds PGMs are located in South Africa. PGMs are extracted from sulphide ores mainly by the use of six-electrode SAFs (Nell, 2004). Smelting occurs at extreme operating temperatures to enable the formation and subsequent separation of liquid phases. This separation collects PGMs for further

downstream processing. PGM smelting is an important industrial process in South Africa and functions as a suitable case study for application of state estimation and fault detection techniques.

Implementation of model-based state estimation techniques in practice relies upon having access to an accurate mathematical model of the process. In general, SAF models are complex models involving a large set of nonlinearly interacting state variables. There exist several process models of a SAF in literature. The model developed by Theunissen (2021) provides a simplified dynamic model of a SAF in the form of a system of ordinary differential equations (ODEs) developed specifically for investigating fault detection algorithms. This form of an explicit set of ODEs is preferred for application of state estimation techniques. In addition, the process model is computationally affordable, facilitating rapid generation of synthetic measurement data for state estimation and fault detection. Like any commercial process, the SAF is susceptible to faulty conditions. These include major events like furnace blowback and other abnormal process conditions that ultimately impact the integrity of the process. Timely and accurate fault detection is therefore essential for safe and profitable operation of the SAF.

1.2 Motivation

The need for state estimation in the smelting process is apparent. The process is highly energy intensive with the majority of the operating costs arising from the energy requirements (Shyamal, 2018). This drives the need for process control. Furthermore, the process is susceptible to disturbances and faults, which can be detrimental to safe and profitable operation. These conditions motivate the need for process monitoring and fault detection. However, there is a lack of available measurements due to the extreme operating conditions of the furnace (Ghobara, 2013). SAFs typically operate at a slag temperature of 1900 K, making frequent sampling impractical. By implementing state estimation techniques, accurate and rapid state information becomes available, allowing for improved process monitoring and implementation of model-based control and model-based fault detection.

There are limited studies in literature which compare the performance of the EKF, UKF, PF and MHE on a moderately complex chemical process. There have been a number of papers comparing the MHE and EKF (Lima & Rawlings, 2011) (Alexander et al., 2020) (Gmehllng et al., 1986) (Haseltine & Rawlings, 2003b), the EKF and UKF (Geetha et al., 2013), and the EKF and PF (Mansouri et al., 2013), for state estimation in a simple CSTR. One study compared the EKF and UKF for state estimation in the complex Tennessee Eastman process (Upendra & Prakash, 2013). It is evident that there is potential to gain valuable insight from a comparative study investigating various state estimation techniques in a complex industrial process. This project would entail comparison of the EKF, UKF, PF and MHE in a complex smelting process involving a large set of nonlinearly interacting state variables.

Furthermore, the MHE is a fairly modern state estimation technique and remains a challenging filter to implement due to the lack of a general set of heuristics and large computational burden (Alexander et al., 2020). Comparison of the MHE to simple popular techniques such as the EKF for state estimation in a

complex chemical process could highlight its advantages and flaws and give insight for future investigations and real-world practical implementation.

Lastly, there are limited literature studies comparing model-based and data-driven methods of fault detection. Yang & Rizzoni (2016) compared model-based fault detection using an unknown-input observer with data-driven fault detection via linear discriminant analysis. There currently exists no literature that directly compares process history-based fault detection methods with model-based fault detection using the state estimators for fault detection in a complex chemical process. In addition, this study provides further insights to the study done by Yang (2004), highlighting the potential benefits of performing residual evaluation using PCA in an attempt to control the bias-variance trade-off.

1.3 Project aim and objectives

The aim of this project was to investigate the use of state estimation techniques, namely the EKF, UKF, PF and MHE, to enable model-based fault detection and compare the model-based technique to data-driven fault detection under plant-model mismatch. This is done for a typical and moderately complex unit operation by means of a suitable case study, in this case a submerged arc smelting furnace. To satisfy this aim, the following objectives have been identified:

1. Assess the state observability of the SAF system to validate the state estimation in objective 2 and conduct a fault detectability analysis by assessing structural fault parameter observability and performance-based fault detectability to inform the fault detection in objective 3.
2. Estimate the observable states of the SAF system identified in objective 1 using the EKF, UKF, PF, and MHE, and select one or more of these techniques for model-based fault detection for objective 3.
3. Assess the performance of model-based fault detection using state estimation under plant-model mismatch and compare the performance of the model-based method with a data-driven method.

1.4 Project scope

1. The model of the SAF developed by Theunissen (2021) is used to generate synthetic plant measurements and used as the process model in the state estimation algorithms. The validity and accuracy of the model are not investigated as model development does not fall within the scope of the project.
2. State observability of the SAF system model is assessed to validate the application of the state estimation algorithms and to inform the results of the state estimation based on the degree of observability of the various states.
3. The detectability of the fault parameters is assessed via a structural parameter observability analysis as well as a performance-based detectability analysis to inform the results of the fault detection.
4. State estimation algorithms are employed using synthetic plant measurement data simulated using the SAF model. Data-driven and model-based fault detection of various simulated faults is

also carried out on synthetically generated data. Extending this application to industrial smelting data falls outside the scope of this project due to the practical limitations of obtaining real-world plant data.

5. The performance of the state estimators for model-based fault detection are compared to a data-driven method of PCA. The classification performance of the fault detection techniques is assessed. Subsequent fault isolation and diagnosis falls out of the scope of this project.

1.5 Thesis layout

Chapter 2 provides the reader with the necessary theory required to understand the concepts investigated in this study. This chapter begins with an overview of state estimation and presents the theoretical framework of the standard KF algorithm, which forms the basis for all the algorithms investigated in this study. This is followed by development of the EKF, UKF, PF and MHE algorithms. The chapter then goes on to discuss the concept of system observability and how to conduct an observability analysis for linear and nonlinear systems. The chapter then presents an in-depth review on the current literature pertaining to model-based fault detection methods, with focus on residual generation using state estimators, and data-driven fault detection, specifically exploring principal component analysis. The chapter concludes by introducing the concept of fault detectability, with reference to both structural fault detectability and its relation to parameter observability, as well as performance-based fault detectability.

Chapter 3 presents the case study of the SAF used in objectives 1, 2, and 3 of this study. The chapter begins with a brief overview of the smelting process for PGM smelting. The reader is then provided with a description of the process model and measurement model of the SAF. The chapter also presents the typical faults that occur within the smelting process along with the categorization of the faults and the fault models for simulating these faulty conditions in the SAF.

Chapter 4 details the approach used to conduct the state observability analysis and the fault detectability analysis. The chapter then presents the results of the observability and detectability analyses to address objective 1 of the study.

Chapter 5 outlines the procedure for numerically implementing the EKF, UKF, PF, and MHE algorithm and derives appropriate tuning parameters required for implementation of the state estimation algorithms. The chapter then presents the results from implementation and comparison of the performance of the state estimation techniques as well as selection of state estimators to use in model-based fault detection.

Chapter 6 presents the methodology for model-based fault detection using state estimation and data-driven fault detection using PCA used in this study. The chapter then presents the findings of the comparison between model-based fault detection and a data-driven method and evaluates the performance of the model-based method under plant-model mismatch.

Chapter 7 summarizes the findings of each of the objectives of the study and concludes the thesis by presenting recommendations for practical implementations of these techniques and recommendations for future studies based on the findings drawn from this study.

2 THEORY AND LITERATURE REVIEW

2.1 State estimation theory and literature review

The following sections summarise the theoretical basis and relevant literature on the state estimation techniques investigated in this study. Section 2.1.1 provides a brief introduction to state estimation and outlines the broad categories of state estimators. Section 2.1.2 derives the standard KF, which forms the foundation for the state estimation techniques used in the remainder of this study. Sections 2.1.3 through 2.1.6 provide a theoretical background on each of the nonlinear state estimation techniques investigated in this study. Section 2.1.3 details the EKF theory and relevant literature, section 2.1.4 the UKF, section 2.1.5 the PF, and section 2.1.6 the MHE. Section 2.1.7 summarises the literature on the selection of state estimator tuning parameters. Section 2.1.8 briefly outlines various applications of state estimation techniques. Section 2.1.9 summarizes the typical methods for assessing the performance of state estimation techniques used in literature. Lastly, section 2.1.10 explains the negative impact of plant-model mismatch on the performance of model-based state estimators.

2.1.1 Introduction to state estimation

The states of a system are the important process variables that provide critical information about the system conditions (Simon, 2006). In industrial processes, access to rapid and accurate state information is not always readily available due to limited measurements that are often corrupted by noise. State estimators have been widely applied to a variety of systems to obtain real-time, accurate estimates for the states of the system. These state estimation techniques can be categorized into data-driven methods, hybrid methods, and model-based methods.

2.1.1.1 Data-driven state estimation

Data-driven methods of state estimation develop an understanding of the relationships between process variables by training on large sets of past data to develop a model that can be used to make reliable estimates for the states based on the current process inputs and current measurement data. Examples of data-driven state estimation techniques include using fuzzy logic or artificial neural networks to build the model of a process. In recent years, these methods have become increasingly popular in the state estimation space due to the increase in computational power and advancements in sensor technology resulting in availability of extensive sets of historical data (Jin et al., 2021). Data-driven methods of state estimation are ideal for systems with ill-defined process models, as variable relationships are formulated independently from model knowledge (Ali et al., 2015). However, the data-driven methods require significant computational time, require large sets of historical process data under various operating conditions for training, show poor performance when tested under adverse process conditions not included in training data, and provide inaccurate state estimates when trained on limited and noisy measurements (Alexander et al., 2020).

2.1.1.2 Hybrid state estimation

A new type of state estimation technique has arisen in recent years, termed hybrid state estimation. A hybrid state estimator refers to any state estimation technique that combines the algorithms of two or more techniques. This can be any combination of two model-based algorithms, two data-driven methods, or a combination of both model-based and data-driven techniques. Hybrid estimators are used to augment the performance of a single estimator by improving the accuracy, robustness, or convergence, when the performance of that single estimator is unsatisfactory in its application. This is done by careful selection and incorporation of an additional estimator which resolves the limitations associated with the single estimator (Ali et al., 2015). Ali *et al.* (2015) explain in a review paper that since the introduction of hybrid observers to the state estimation space in the 2000s, the use of hybrid observers over single observers is prevalent and growing. More specifically, hybrid estimators which combine model-based and data-driven state estimation techniques have gained particular attention (Jin et al., 2021).

2.1.1.3 Model-based state estimation

Model-based methods use a dynamic model developed from first principles and noisy measurements from the process to make accurate estimates for states of the system. Therefore, model-based methods require complete and precise knowledge of the dynamic model of a system (Kravaris & Hasan, 2016). Model-based methods are often categorized as deterministic or Bayesian (Alexander et al., 2020). Deterministic methods assume that the system involves no randomness, and the output will always be the same for the same input and parameters. Bayesian methods assume that the process is stochastic, whereby the states and measurements are random variables that change with time and the final state estimate is solved for by calculating the conditional probability distribution using Bayes theorem (Jin et al., 2021).

The two most popular deterministic state estimators are the Luenberger observer (LO) and the asymptotic observer (AO). The LO is limited to first-order linear systems of differential equations and requires highly accurate process models and initial guesses for the states (Alexander et al., 2020). The AO is advantageous in biochemical processes with complex process kinetics as the model simplifies the differential equation system to exclude the process kinetics terms. However, in application with chemical and biochemical processes, the AO only converges for fed-batch and continuous systems (Alexander et al., 2020), tends to display slow rates of convergence, and requires a large number of available measured variables (Dochain, 2003).

The other category of model-based state estimators are Bayesian state estimation techniques. Within Bayesian state estimators, there exist two sub-categories: recursive filters and optimization-based filters. Recursive filters use state information from the previous estimate to estimate the current state and do not require all the past information. Examples of recursive Bayesian state estimators include the KF, EKF, UKF, and PF. The second type of Bayesian state estimator is optimization-based, whereby the state estimation problem is reformulated as an optimization problem using a past horizon of measurements. The most popular optimization-based Bayesian state estimator is the MHE (Alexander et al., 2020).

To develop the basic understanding of model-based state estimation theory, the standard KF algorithm is derived. The KF can be derived from the Bayesian state estimator or from the least squares approach. The KF is the optimal linear filter when the measurement and process noise is Gaussian in the Bayesian derivation or it is the optimal linear filter, regardless of if the noise is Gaussian or not, when derived from the least-squares approach (Simon, 2006). Derivation of the standard KF firstly requires an understanding of a state observer and state space notation.

2.1.2 Standard Kalman Filter

A linear discrete-time system can be represented in state space form as:

$$x_k = Fx_{k-1} + Gu_{k-1} + w_{k-1} \quad [1]$$

$$y_k = Hx_k + v_k \quad [2]$$

x_k represents the state vector at timestep k, this is a vector containing each of the state variables of the system. u is the input vector containing the exogeneous input variables that arise from outside of the system boundaries and are therefore independent of the system itself. y_k is the output vector at timestep k containing the observable variables which are the measurements provided by sensors on the plant. F is the state transition matrix that contains the differential equations representing how the current state, x_k, affects the rate of change of the state. G is the input matrix containing the equations which explain the relationship between the inputs and the rate of change of the states. H is the measurement matrix containing the measurement equations relating the system state variables to the measurements. w is the process noise or the unknown and unmeasured plant disturbances that cause unmodelled variations in the states of the system. w is a stochastic process that is partly responsible for driving the true underlying states of the system (Simon, 2006). v is a stochastic process that represents the measurement noise that arises due to sensor inaccuracies or thermal noise within the sensor itself.

Figure 1 shows a visual depiction of a state observer. The state observer uses the measurements from the real plant process and a process model to make estimates for the state variables, $\hat{x}_k$.

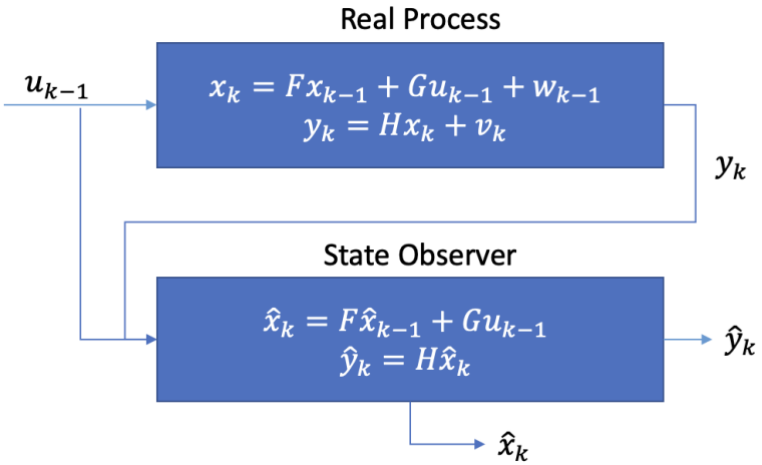

Figure 1: Visual depiction of a state observer.

The equations below represent the mathematical model of the linear discrete-time process used in the state observer.

$$\hat{x}_k = F\hat{x}_{k-1} + Gu_{k-1} \quad [3]$$

$$\hat{y}_k = H\hat{x}_k \quad [4]$$

It is assumed that the control inputs, u, and the measurements, y_k, are exactly known. It is also assumed that the plant dynamics are perfectly known, therefore, the system matrices F, G and H are used in the mathematical model of the state observer. $\hat{x}$ represents the vector of state estimates and $\hat{y}$ represent the vector of estimated measured variables obtained from the state observer.

The KF is a type of state observer derived under specific system conditions:

1) The system must represent a stochastic process. The states, x_k, and measurements, y_k, are time-variant random variables.
2) The system must be linear.
3) The process and measurement noise are zero-mean white noise with known covariances Q and R, respectively. $w_k \sim (0, Q)$ and $v_k \sim (0, R)$.

The state estimates are random variables with probability density functions (pdf) that are fully defined by their mean, $\hat{x}$, and covariance, P. The standard KF splits the state estimation procedure that occurs within the state observer into two steps: the prediction step and the update step (Becker, 2023). Figure 2 is a visual representation of this procedure.

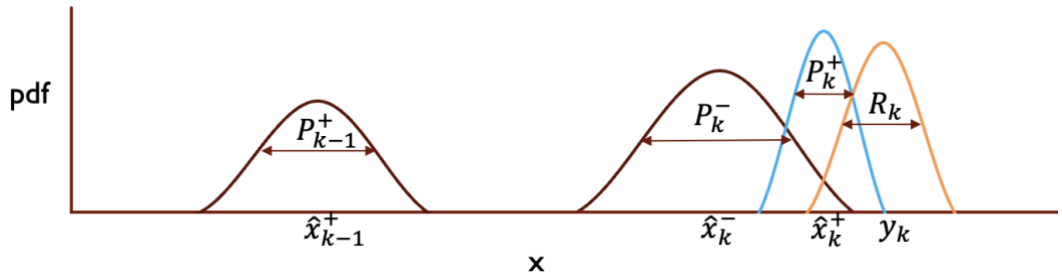

Figure 2: Visual representation of the standard KF procedure, adapted from Becker (2023).

The prediction step involves propagating the mean and covariance of the state estimates from the previous timestep to the current timestep. The left maroon distribution representing the state estimate at the previous timestep is defined by mean $\hat{x}_{k-1}^+$ and covariance P_{k-1}^+. This distribution is propagated to timestep k to obtain the right maroon distribution fully defined by $\hat{x}_k^-$ and P_k^-. The update step involves updating the model prediction with the available measurement, the orange distribution with mean y_k and covariance R_k. This update step results in the final state estimate distribution, represented by the blue distribution with mean and covariance, $\hat{x}_k^+$ and P_k^+, respectively.

The derivation of the standard KF equations can be found in appendix A.1 by the least-squares derivation and appendix A.2 by the Bayesian derivation. These equations are summarized below.

The prediction step propagates the previous state estimate through time using the linear process model presented in Equation 3. Equation 5 describes the propagation of the mean, $\hat{x}_{k-1}^{+}$, through the linear process model.

$$\hat{x}_k^- = F_{k-1}\hat{x}_{k-1}^+ + G_{k-1}u_{k-1} \quad [5]$$

The covariance is propagated using Equation 6 by propagating the covariance through the linear model and adding the process noise, Q. Although adding additional noise to the estimate error covariance seems counterintuitive, given that the goal of the filter is to filter out noise, the process noise represents the inherent stochastic process that drives the states of the system and must be considered in the estimator to obtain accurate estimates (Mohan et al., 2015).

$$P_k = E[(x_k - \bar{x}_k^-)(x_k - \bar{x}_k^-)^T] = F_{k-1}P_{k-1}^+F_{k-1}^T + Q \quad [6]$$

The update step of the KF involves solving for the final state estimate, $\hat{x}_k^+$, by updating the model prediction, $\hat{x}_k^-$, with the current measurement, y_k , using the Kalman gain, K_k

$$\hat{x}_k^+ = \hat{x}_k^- + K_k(y_k - H\hat{x}_k^-) \quad [7]$$

The Kalman gain plays a crucial role in the KF algorithm as it dictates the contribution of the model prediction, $\hat{x}_k^-$, and the measurement, y_k, to the final state estimate, $\hat{x}_k^+$. The Kalman gain is calculated as:

$$K_k = P_k^- H_k^T \left(H_k P_k^- H_k^T + R_k\right)^{-1} \quad [8]$$

The covariance is updated via:

$$P_k^+ = (I - K_k H_k)P_k^-(I - K_k H_k)^T + K_k R_k K_k^T \quad [9]$$

Most real-world systems are hybrid systems with continuous-time process dynamics and discrete-time measurements. A continuous-time process with discrete-time measurements and Gaussian noise is represented as:

$$\dot{x} = Ax + Bu + w \quad [10]$$

$$y_k = H_k x_k + v_k$$

$$w \sim (0, Q_c)$$

$$v_k \sim (0, R_k)$$

Where A represents the state transition matrix and B the input matrix. $w(t)$ is now a continuous-time white noise process with continuous-time process noise covariance Q_c. The discrete-time process noise covariance, Q, can be approximated from the continuous-time process noise covariance, Q_c, by:

$$Q = Q_c T \quad [11]$$

Where T is the sampling time (Simon, 2006).

The hybrid KF algorithm uses the continuous-time KF for the prediction step and the discrete-time KF for the update step. The continuous-time KF is derived in appendix A.3. The prediction step of the

continuous-time KF, and therefore the prediction step of the hybrid KF, propagates the mean and covariance by:

$$\dot{x} = Ax + Bu \quad [12]$$

$$\dot{P} = -PC^T R_c^{-1} CP + AP + PA^T + Q_c \quad [13]$$

The update step of the hybrid KF is the same as the update step of the discrete-time KF.

For linear systems, this standard KF gives the best estimate of the state, the minimum-variance estimate, if the noise is Gaussian (Sorenson, 1970). When the noise is non-Gaussian, the standard KF is the best linear estimator (Simon, 2010). In addition, the standard KF is an unconstrained filter, limiting its application in processes that require physical constraints on the states (Patwardhan et al., 2012). These strict assumptions limit the application of the standard KF, thus, alternative state estimation techniques have been developed to explicitly handle non-linear dynamics, non-Gaussian distributions, and explicitly implement constraints.

2.1.3 Extended Kalman filter

2.1.3.1 Linearized KF

One method of applying the standard KF algorithm presented above to a nonlinear system is to linearize the model and then use the standard KF prediction and update steps using the linearized state transition and measurement matrices. The method of linearization is a Taylor series expansion around the nominal values, x_0 and u_0, based on a guess of the systems nominal trajectory.

Let the nonlinear continuous-time system be represented by nonlinear functions f and h.

$$\dot{x} = f(x, u, w, t) \quad [14]$$

$$y = h(x, v, t)$$

$$w \sim (0, Q)$$

$$v \sim (0, R)$$

Where f represents the nonlinear dynamic equations of the system and h represents the nonlinear measurement equations relating the state variables to the measurements.

The Taylor series expansion of the nonlinear system is derived in appendix A.4. A first order Taylor series approximation, which considers the first two terms in the total Taylor series expansion, is used to linearize the system model:

$$\dot{x} = f(x_0, u_0 = u, w_0 = 0, t) + A\Delta x + B\Delta u + Lw \quad [15]$$

$$y = h(x_0, v_0 = 0, t) + C\Delta x + Mv \quad [16]$$

$$A = \frac{\partial f}{\partial x}\Big|_{x_0} \qquad B = \frac{\partial f}{\partial u}\Big|_{u_0 = u} \qquad L = \frac{\partial f}{\partial w}\Big|_{w_0 = 0} \qquad C = \frac{\partial h}{\partial x}\Big|_{x_0} \qquad M = \frac{\partial h}{\partial v}\Big|_{v_0 = 0}$$

The standard KF equations can then be applied to this linearized system to estimate the states of the system.

The Taylor series linear approximation is only accurate if the actual system is close to the nominal values, x_0 and u_0, used in the approximation. As the system is subjected to unknown disturbances, changes in the input conditions, and potential modelling error, the system diverges from these nominal conditions, inducing significant linearization error in the estimator (Simon, 2006). The linearization error causes poor state estimation and potential filter divergence.

2.1.3.2 EKF algorithm

In the 1970s, the EKF was proposed by Bucy and Sunahara to handle state estimation in nonlinear systems (Bucy & Senne, 1971) (Sunahara, 1970). Nonlinear state estimation applications of the EKF quickly became widespread and it remains one of the most popular state estimation techniques today (Nørgaard et al., 2000). The EKF is based on the idea of applying the standard KF to a linearized system of equations, however, the EKF performs local linearization of the dynamic model and measurement equations at each timestep around the state estimate obtained at the previous timestep.

The hybrid EKF algorithm is incorporated using the same equations as the hybrid KF algorithm. The prediction step of EKF integrates the nonlinear process model, Equation 14, to calculate the prior state estimate and integrates the standard continuous-time KF rate of change of covariance, Equation 13, to propagate the estimation error covariance. The update step of the EKF uses the standard discrete-time KF equations, Equations 7 through 9, to calculate the posterior state estimate and posterior estimation error covariance. However, the system matrices, A and C, in these equations are obtained from the Jacobian of the nonlinear equations, f and h, evaluated around the previous state estimate. This algorithm is presented in appendix B.1.

2.1.3.3 Advantages of the EKF

The EKF is the simplest nonlinear state estimator, is easily implementable, and has minimal computational requirements. This simplicity, ease of application, and ability to rapidly estimate states, all contribute to the success and popularity of the EKF. However, the EKF comes with disadvantages mainly stemming from the crude approximation of the nonlinear model.

2.1.3.4 Disadvantages of the EKF

The EKF employs a first-order Taylor series approximation to linearize the model. First-order linearization utilizes only the first two term of the Taylor series expansion. Appendix A.4 proves that this approximation only follows the true mean and covariance up until the first order (Simon, 2010). If there exist severe nonlinearities in the process model, first-order linearization induces significant linearization error, potentially causing inaccurate state estimates or filter divergence (Castellanos et al., 2004). Linearization error is reduced by using a higher order Taylor series approximation or using an alternative method to propagate the mean and covariance through the nonlinear functions.

Another disadvantage of the EKF that has been highlighted in literature is the requirement for computation of Jacobian matrices for the linearization procedure (Julier & Uhlmann, 2004). However, for many systems this calculation of Jacobians can be done fairly easily (Laviola, 2003). Moreover, recent papers present techniques such as automatic differentiation or symbolic calculation of Jacobians prior to implementation of the filter that mitigate this aforementioned disadvantage of the EKF (Duerinckx et al., 2015)(Tian et al., 2023).

2.1.4 Unscented Kalman filter

The UKF was proposed by Julier and Uhlmann in 1997 (Julier & Uhlmann, 1997). The UKF provides an alternative method for propagating the mean and covariance through the nonlinear model by means of the unscented transformation. The unscented transformation involves representing a distribution through deterministic sampling of samples known as sigma points and individually transforming the sigma points through the nonlinear function.

2.1.4.1 Unscented transformation

When independent Gaussian random variables are transformed through a linear function, the resulting random variables are also Gaussian. It is much more difficult to transform a random variable through a nonlinear function as the shape of the distribution changes (Ross, 2021). One method of approximating the mean and covariance of a nonlinearly transformed random variable is by the unscented transformation. The unscented transformation makes use of the principle that it is easier to transform single points through a nonlinear function than it is to transform an entire distribution (Li et al., 2009). To exploit this principle, one needs a set of points with a statistical distribution that accurately approximates the original distribution. The unscented transformation makes use of deterministic sampling to obtain this set of points.

Deterministic sampling involves taking a small, non-random set of weighted points (Sahlberg, 2016). Deterministic sampling is significantly less computationally intensive than random sampling. The weighted mean and covariance of these deterministically sampled points is equivalent to the mean and covariance of the original pdf. The points are then individually transformed through the nonlinear function to obtain a set of transformed points with a weighted average and covariance representing the approximate mean and approximate covariance of the transformed distribution.

There exist several methods for performing these unscented transformations (Simon, 2006). One of these methods, known as the general unscented transformation, is presented below.

2.1.4.2 Generation of the sigma points

Let x represent the original state vector and x^i are the sigma points, which represent a set of vectors. Let the mean and covariance of x be exactly known as $\bar{x}$ and P, respectively. The sigma points are generated as:

$$x^0 = \bar{x} \qquad [17]$$

$$x^i = \bar{x} + \tilde{x}^i \qquad i = 1, \cdots, 2n$$

$$\tilde{x}^i = \left(\sqrt{(n+\kappa)P}\right)_i^T \qquad i = 1, \cdots, n$$

$$\tilde{x}^{n+i} = -\left(\sqrt{(n+\kappa)P}\right)_i^T \qquad i = 1, \cdots, n$$

The number of sigma points generated is $2n + 1$, where n is the number of states, or the number of rows in state vector x. One of the sigma points is exactly equal to the mean. The other $2n$ sigma points are each one standard deviation away from the mean in both directions for all n dimensions. The sigma points are symmetric, where half of them distributed on the positive side of the mean and the other half are located on the negative side of the mean.

The matrix square root, $\sqrt{(n+\kappa)P}$, is obtained by Cholesky decomposition whereby $\sqrt{(n+\kappa)P}$ is the matrix square root of $(n+\kappa)P$ such that $(n+\kappa)P = \sqrt{(n+\kappa)P}^T \sqrt{(n+\kappa)P}$. The matrix square root is a matrix itself of size $n \times n$. The subscript i on $\left(\sqrt{(n+\kappa)P}\right)_i^T$ extracts the i'th row of the matrix square root and is transposed because the Cholesky decomposition of the matrix square root results in the transpose of the desired result. (Simon, 2006)

2.1.4.3 Mean approximation via the unscented transformation

Let an arbitrary non-linear function be defined as:

$$z = g(x) \qquad [18]$$

The sigma points, x^i, are transformed through the non-linear function, g.

$$z^i = g\left(x^i\right) \qquad [19]$$

The approximate mean is found by weighted linear regression of the transformed sigma points (Martinez-Cantin & Castellanos, 2005).

$$\bar{z}_{approx} = \sum_{i=1}^{2n+1} W^i z^i \qquad [20]$$

Where the weighting coefficient is defined as:

$$W^0 = \frac{\kappa}{n+\kappa} \qquad [21]$$

$$W^i = \frac{1}{2(n+\kappa)} \qquad i = 1, \cdots, 2n$$

κ is a scaling factor that influences the higher-order moments of the mean and covariance approximation (Simon, 2006). For Gaussian distributions, $\kappa = 3 - n$ is appropriate as this reduces the error on the fourth-order term of the approximation of the mean and covariance (Ebeigbe et al., 2021) (Julier & Uhlmann, 2004).

2.1.4.4 Covariance approximation via the unscented transformation

Similarly, the approximated covariance is calculated from linear weighted regression as:

$$P_{approx} = \sum_{i=1}^{2n+1} W^i \left(z^i - \bar{z}_{approx}\right)\left(z^i - \bar{z}_{approx}\right)^T \quad [22]$$

2.1.4.5 Unscented Kalman filter algorithm

The hybrid UKF algorithm makes use of deterministic sampling and the unscented transformation during the prediction step and the update step of the filter. In the prediction step, sigma points are generated using Equation 17 based on the state estimate and covariance from the previous timestep. The unscented transformation is used to transform the sigma points from the previous timestep to the current timestep via numerical integration of the nonlinear function representing the state dynamics, f. The prior state estimate and estimation error covariance after the prediction step are the weighted mean, calculated using Equation 21, and the weighted covariance, calculated using Equation 22, of these transformed sigma points.

The update step involves deterministically sampling sigma points from the prior state estimate and prior estimation error covariance. The update step also involves using the unscented transformation to transform the sigma points through the nonlinear measurement equation, h. The hybrid UKF then uses the standard KF update step, Equations 7 through 9, using the approximate mean and covariance from the unscented transformation to calculate the posterior state estimate and estimation error covariance. This algorithm is presented in appendix B.2.

2.1.4.6 Advantages of the UKF

The UKF has some advantages over other nonlinear estimation algorithms. Firstly, the UKF provides a more accurate method of approximation of the mean and covariance of nonlinearly transformed distributions compared to the EKF when the system has severe nonlinearities (Simon, 2006). Appendix A.5 shows that the unscented transformation approximations for the mean and covariance follow the true mean and covariance up until the third order (Simon, 2006). This is improvement over the first order approximation of the EKF and reduces the approximation error introduced in the prediction step, allowing for more accurate state estimates in the presence of nonlinearities. It should be noted that the superior approximation accuracy of the UKF is only fully exploited for highly nonlinear system models and when the measurement noise is significantly larger than the process noise. When the measurement noise is larger than the process noise, the model prediction has a significant contribution to the final state estimate and, thus, model approximation accuracy significantly affects the final state estimate (Qu, 2009). The UKF also achieves this approximation without the need for the computation of Jacobians, which can become a complex task when system equations are not in analytical form (Simon, 2006). The last advantage of the UKF is its ease of parallelization as the propagation of each of the sigma points through the nonlinear transformation occurs independently (Mangold et al., 2009).

2.1.4.7 Disadvantages of the UKF

The superior estimation accuracy of the UKF comes at the cost of a slightly larger computational burden due to the computation of n state estimates with the $2n + 1$ sigma points. However, the computational

burden of the UKF is not significantly greater than the EKF due to the use of deterministic sampling of sigma points rather than random sampling.

2.1.5 Particle filter

Another popular state estimation technique developed to handle non-linear systems with non-Gaussian distributions is the PF. The first developments in the PF methodology were proposed in 1956 by Norbert Weiner, however, implementation was limited due to the large computational effort required (Wiener, 1956). The modern PF gained attention in 1980 due to the availability of enhanced computational power.

The PF is derived from the recursive Bayesian estimator, formulated in appendix A.2. The pdf of the state estimate at time k, x_k, given all the measurements prior to time k, $Y_{k-1} = y_1 \dots y_{k-1}$, is represented as $p(x_k|Y_{k-1})$. $p(x_k|Y_{k-1})$ is the *a priori* pdf obtained after the prediction step of the standard KF. The *a priori* pdf is calculated as:

$$p(x_k|Y_{k-1}) = \int p(x_k|x_{k-1})p(x_{k-1}|Y_{k-1})dx_{k-1} \quad [23]$$

The *a posteriori* conditional pdf, $p(x_k|Y_k)$, obtained after the update step of the KF, is calculated from:

$$p(x_k|Y_k) = \frac{p(y_k|x_k)p(x_k|Y_{k-1})}{\int p(y_k|x_k)p(x_k|Y_{k-1})dx_k} \quad [24]$$

When the state transition and measurement equations are linear and the state and measurement distributions are assumed to be Gaussian, the analytical solutions to these equations give rise to the standard KF equations, as derived in appendix A.2. However, when the system does not obey these assumptions, the analytical solutions are often difficult to calculate (Simon, 2006).

The PF approximates the *a priori* and the *a posteriori* state estimate distributions by approximating the solutions to Equations 23 and 24. The PF algorithm involves a prediction step, for approximating the *a priori* state estimate distribution, and an update step, for approximating the *a posteriori* state estimate distribution. The PF also contains a crucial additional step called resampling.

2.1.5.1 Prediction step

The prediction step of the PF aims to approximate the *a priori* pdf via Monte Carlo integration. Monte Carlo integration approximates the integral of a function $f(x)$ over the density $p(x)$ using random samples drawn from the density (Li et al., 2014). From Equation 23, the function $f(x)$ is the model prediction $p(x_k|x_{k-1})$,. The density $p(x)$ is the *a posteriori* state estimate from the previous timestep $p(x_{k-1}|Y_{k-1})$.

N random samples are drawn from the distribution, $p(x)$. The random samples are state vectors called particles, x_i with $i = 1, \dots, N$. The particles are distributed across the state space and the frequency of particles within a region approximately corresponds to the probability of the state existing within that region.

Each individual particle undergoes a nonlinear transformation using the nonlinear process model, f. Instead of transforming the entire distribution using the nonlinear function, which is numerically

complex, the individual particles are transformed. This results in a distribution of particles at the current timestep representing the *a priori* distribution.

The *a priori* state estimate at timestep k, $\hat{x}_k^-$, is equivalent to the mean of the set of transformed particles:

$$\hat{x}_k{}^- = \int f(x)p(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_{k,i}) \quad [25]$$

The distribution of the initial state estimate at timestep zero is assumed to be a Gaussian distribution or uniform distribution for example, in order to generate the initial particles for the PF algorithm. For a Gaussian distribution, randomly sampling is achieved by randomly generating samples around the mean of the state estimate with covariance equal to the state estimation error covariance. For the prediction step of the successive timesteps, the *a posteriori* particles from the previous timestep are individually transformed through the model prediction. Therefore, no further assumptions are made about the shape of the underlying distribution besides during the initialization step.

2.1.5.2 Update step

The next step in the PF algorithm involves approximating the *a posteriori* state estimate using the measurement at the current timestep to improve upon the *a priori* state estimate obtained in the prediction step.

Monte Carlo integration cannot be used to approximate the *a posteriori* pdf, $p(x_k|Y_k)$, because the underlying distribution of this pdf is unknown. Therefore, sequential importance sampling is used to approximate the pdf. Sequential importance sampling is the key principle driving the PF algorithm (Li et al., 2014). The idea of sequential importance sampling is that the underlying distribution of the *a posteriori* pdf is assumed. This assumed distribution is referred to as the importance density. Random sampling using particles is used to approximate this importance density. To account for the difference between the true underlying pdf and the importance density, weightings are assigned to each particle representing the importance density.

It is common practice in literature to choose the importance density as the *a priori* pdf from the prediction step, also known as the transition density $p(x_{k,i}|x_{k-1,i})$ (Elfring et al., 2021) (Patwardhan et al., 2012). Using the transition density as the importance density function enables simple calculation of the weightings assigned to each particle using the likelihood of the observation.

For each of the *a priori* particles, their corresponding predicted measurement can be calculated by transforming each particle using the measurement equation, $\hat{y}_{k,i} = h(\hat{x}_{k,i})$. There exists the true measurement at the current time step, y_k, which is assumed to have a Gaussian distribution. For each particle, the relative likelihood, q_i, of obtaining the true measurement, y_k, from the predicted distribution, $h(\hat{x}_{k,i})$, can be calculated. This is calculated by evaluating the pdf for the conditional probability of y_k given $\hat{x}_{k,i}$. The relative likelihood is easy to calculate for Gaussian distributions using the definition of a Gaussian pdf: $X \sim N(\bar{X}, C)$.

$$pdf(X) = \frac{1}{(2\pi)^{n/2}|C|^{1/2}} \exp\left(-\frac{1}{2}(X-\bar{X})^T C^{-1}(X-\bar{X})\right. \quad [26]$$

Where n is the size of vector X.

The relative likelihood, q_i, is directly proportional to the conditional pdf, $p(y_k|x_{k,i}^-)$. Therefore, $p(y_k|x_{k,i}^-)$ can be calculated as:

$$q_i = p(y_k|x_{k,i}^-) \sim \frac{1}{(2\pi)^{\frac{n_y}{2}}|R|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(y_k - \hat{y}_{k,i}^-)^T R^{-1}(y_k - \hat{y}_{k,i}^-)\right) \quad [27]$$

Where n_y is the number of measurements. (Simon, 2006)

This method of using the transition density as the importance density and subsequent likelihood calculation is common practice. However, since the transition density fails to account for the most recent observation, y_k, the importance density may be far from the true posterior distribution and thus result in low likelihoods (Patwardhan et al., 2012). The transition density is an exceptionally poor assumption for the importance density when there exists plant model mismatch or inaccurate initial guesses (Shao et al., 2009). Furthermore, the transition density fails to approximate the posterior pdf when there are sudden changes to the system, such as disturbances or changes to the inputs (Bolić et al., 2002).

This relative likelihood obtained from Equation 27 is the weighting assigned to each of the particles in the sequential importance sampling step. The weighting quantifies the accuracy of the *a priori* state estimate given the measurement. In other words, the measurement dictates how 'good' each of the prior estimates are. This method of calculating the relative likelihood makes no assumptions about the shape of the *a priori* distribution, only that the measurement distribution is Gaussian.

The *a posteriori* state estimate is then calculated using these weightings. The update step of the PF approximates the continuous *a posteriori* pdf by discretizing the pdf using N samples of $x_{k,i}$ *a priori* particles each with a weighting $w_{k,i}$. $\delta_{x_{k,i}}$ represents the Dirac delta function, which has a value of zero everywhere except for at $x_{k,i}$, and a function integral of 1.

$$p(x_k|Y_k) = \frac{p(y_k|x_k)p(x_k|Y_{k-1})}{\int p(y_k|x_k)p(x_k|Y_{k-1})dx_k} \approx \sum_{i=1}^{N} w_{k,i}\delta_{x_{k,i}}(x_k - x_{k,i}) \quad [28]$$

(Elfring et al., 2021)

The Dirac delta function, $\delta_{x_{k,i}}(x_k - x_{k,i})$, is the limit as the covariance matrix associated with a Gaussian distribution goes to zero. The Gaussian distribution has a mean $\bar{x}_k$, $N \sim (\bar{x}_k, 0)$.

$$\delta_{x_{k,i}}(x_k - x_{k,i}) = n(x_k; \bar{x}_k, 0) \quad [29]$$

2.1.5.3 Resampling

The same set of particles are used in successive timesteps for the entire estimation period. After multiple iterations of the prediction and update steps, a phenomenon known as particle degeneracy occurs. Particle degeneracy occurs when the majority of the weighting is associated with only a few particles (Li et al., 2014). This occurs as the measurement, or evidence term of the importance sampling, only affects the particles in the weighting term of the update (Abbeel, 2020). Particle degeneracy is prevalent when working with high-dimensional systems, when measurement noise is low, or when measurements

contain outliers (Wigren et al., 2018). This results in a large computational effort being used on particles with low likelihoods and may cause divergence of the PF (Elfring et al., 2021). This phenomenon can be avoided by introducing a resampling step.

Resampling can be performed at any timestep. Resampling aims to reduce the computational effort of the PF by preferentially allocating memory to 'good' estimates with high likelihoods and discarding the 'bad' estimates with low likelihoods. Resampling occurs by drawing samples from the weighted set of particles with replacement. This is done in a way such that particles are resampled with probability proportional to their weighting, resulting in preferential selection of high likelihood particles (Doucet & Johansen, 2008). Thus, converting weighting into frequency of particles and tightening the distribution.

There are many resampling methods available. The most basic method of resampling is to simply draw N particles with replacement with the probability of a particle being drawn being equal to its likelihood. This is done by independently generating N random numbers and then selecting the particle corresponding to this random number (Simon, 2006). This changes the distribution of the particles and can cause significant repetition of particles with high likelihoods in the resampled set. If this occurs and the new set of particles all exist far from the true state, this results in approximation error, known as variance.

One method of reducing variance during resampling is by using an alternative resampling method known as low variance resampling or stochastic universal sampling. This is one of the preferred methods of resampling for the PF as it reduces the variance, is efficient, and is easy to implement (Doucet & Johansen, 2008). A basic explanation of this method is selecting all N particles in a sequential stochastic manner. If all the particles have the same weight, then resampling will result in the same particles as the original sample. This is achieved using the cumulative normalized likelihoods. A random number, r, is initially selected between 0 and $\frac{1}{N}$. The low variance resampler then systematically sorts through the particles by adding a fixed amount, $\frac{1}{N}$, to r using $U = r + \frac{1}{N}$. The resampling algorithm does this until it finds the first particle with its cumulative likelihood greater than or equal to U and adds this particle to the resampled set. This is done for all N resampled particles.

The low variance resampling algorithm can be written as:

1) Generate a random number between 0 and $\frac{1}{N}$.
$$r = rand(0; N^{-1})$$
2) Generate a variable representing the likelihood of the particles.
$$c = q_1$$
3) Generate a variable to keep track of the particle number.
$$i = 1$$
4) Create a loop that selects particles based on their cumulative likelihood.
$for\ j = 1\ to\ N$
$U = r + (j - 1)\frac{1}{N}$
$while\ U > c$

$$i = i + 1$$
$$c = c + q_i$$
$$end\ while$$
$$x_i^+ = x_i^-$$
$$end\ for$$

(Thrun et al., 2005)

The mean and covariance of this resampled distribution represents the updated state estimate mean and its covariance.

2.1.5.4 Particle filter algorithm

The general term 'particle filter' encompasses a number of state estimation methods based on importance sampling and Monte Carlo integration. The PF algorithm presented in appendix B.3 summarizes the prediction, update, and resampling steps presented above. This algorithm uses the transition density as the proposal distribution and resampling is performed at every step. This method is the most common particle filtering method and is known as the bootstrap PF (Wigren et al., 2018).

2.1.5.5 Advantages

The PF is advantageous in its ability to handle complex, highly non-linear systems without the need for linear approximations (Simon, 2006). The PF also provides a unique advantage over the Kalman-based methods as it is a non-parametric filter, therefore, it can accurately estimate states with non-Gaussian and potentially multi-model distributions (Shao et al., 2009). Evidently, these advantages come at the cost of larger computational effort required to compute accurate state estimates. However, with modern computational power and ease of parallel implementation of the PF, the computational limitations of the filter are mostly overcome (Straka & Miroslavšimandl, 2006).

2.1.5.6 Disadvantages

Challenges to the practical implementation of the PF algorithm include particle degeneracy, which is alleviated using resampling, and sample impoverishment, which occurs as a result of the resampling. Other challenges of the PF include the sensitivity of the algorithm to potential noise and plant-model mismatch as well as the computational burden of the filter, especially for high-dimensional systems using a large number of particles (Elfring et al., 2021).

2.1.5.7 Sample impoverishment

A common issue that arises due to the resampling step of the PF is sample impoverishment. In the update step, the relative likelihood of each *a priori* particle conditioned on the measurement, y_k, is calculated based on the conditional pdf $p(y_k|x_{k,i}^-)$ (Simon, 2006). Sample impoverishment occurs when the distribution of *a priori* particles does not significantly overlap with the measurement distribution. When there is not significant overlap between the pdfs, only a few particles have large likelihoods and are resampled. Thus, the *a posteriori* particles are represented by a few particles with the same value (Elfring et al., 2021). This is especially prevalent in the bootstrap PF when the transition density is used as the

importance density since the model prediction typically does not have good overlap with the measurements (Pardal et al., 2015).

This becomes more of a serious issue when there are modelling errors or in the case of a poor initialization, causing model predictions to differ significantly from the measurements. Sample impoverishment is also exacerbated when the measurement noise is small as this causes a peaked likelihood function and inaccurate assignment of weightings (Chatzi & Smyth, 2002). Inappropriately small measurement noise causes fairly accurate *a priori* particles have small likelihoods and be excluded during resampling. There can also be insufficient overlap due to the presence of outlier measurements (Wigren et al., 2018).

Sample impoverishment can also result from a small process noise covariance matrix, Q. In the prediction step, *a posteriori* particles from the previous timestep are propagated through the nonlinear state transition function to obtain the *a priori* particles at the current timestep, where uncertainty is introduced by adding random process noise to the particles. When there is small value for the process noise, the distribution does not diversify. This results in a tight distribution of the particles and less chance of the particle distribution overlapping with the measurement distribution (Elfring et al., 2021). This is especially prevalent in time-invariant states which remain unaltered during the prediction step and rely on process noise for diversification (Chatzi & Smyth, 2002). By selecting a larger value for Q, also known as adding artificial process noise, this induces random variations in the prediction, thus, diversifying the particles. Essentially, this changes the importance density and has been shown to reduce the variance of the posterior distribution (Snyder et al., 2015).

As mentioned in sub-section 2.1.5.3, particle degeneracy occurs when the variation among particles is too large. Resampling prevents particle degeneracy by ensuring low variance in the distribution of particles. However, this low variance induces sample impoverishment (Shao et al., 2009).Therefore, an important consideration when implementing the PF is to strike a balance between accuracy and variance.

Methods for overcoming sample impoverishment include roughening, priori editing, regularized particle filtering, Markov chain Monte Carlo resampling, and auxiliary particle filtering (Simon, 2006). These methods for overcoming impoverishment inject variance into the filter and thus decrease the accuracy of the state estimates but ensure particle diversity.

2.1.5.8 Robustness of the PF

Due to the nature of the update-step of the PF algorithm and the resulting phenomenon of sample impoverishment, the PF has been proven to show exceptionally poor robustness to plant-model mismatch. When the transition density is used as the importance density, the filter becomes increasingly sensitive to plant-model mismatch as the importance density is solely dependent on the process model and not the current measurement (Chen et al., 2004). In the presence of plant-model mismatch, the samples from the transition density lie far from the true state. Once these samples are incorrectly located in the state space, the weights cannot always recover the samples as the weights of most particles tend to zero and resampling does not correct for the erroneous samples (Jagadeesan et al., 2011). This induces

sample impoverishment, resulting in lack of diversification of particles and limited expressiveness of the filter.

In addition, the PF shows poor robustness to inaccurate initialization compared to other filters (Chen et al., 2004). The recursive calculation of particle weights and subsequent resampling step prevents recovery of the filter when poor initial conditions are used. These inaccurate states caused by poor initialization are assigned higher weights and are preferentially resampled causing potential divergence of the filter (Chatzi & Smyth, 2002). Once the state estimates sufficiently deviate from the true state values, the filter cannot always recover as the likelihood will be small for true state values and resampling will not move the distribution of the particles to the correct location (Rawlings et al., 2018).

2.1.5.9 Particle number selection

The large number of particles required for accurate state estimation in the PF is the major limitation in real-world application due to the excessive computational cost incurred (Alexander et al., 2020). Particle number selection is therefore an important consideration in the design of an efficient PF to accomplish good estimation accuracy without incurring an inappropriate computational burden.

By increasing the number of particles infinitely, the posterior pdf approximation converges to the exact pdf (Elfring et al., 2021). Thus, the number of particles selected dictates the accuracy of the estimation method. A general guideline when selecting the number of particles is to consider the number of states being estimated. For a system with 17 state variables, using 1000 particles results in $\sqrt[17]{1000} = 1.5$ particles per independent axis (Chen et al., 2005). However, due to the dynamic relationships between the states, using a lower number of particles is still viable. Problems arise when a large number of particles is required to accurately represent the distributions. This is exacerbated in high dimensional state spaces, known as the curse of dimensionality.

The curse of dimensionality is a major drawback of the PF and limits its application in high dimensional estimation (Patwardhan et al., 2012). As the number of states being estimated increases, the number of particles required to accurately approximate the distribution increases exponentially (Shao et al., 2009). The curse of dimensionality is explained using the likelihood function, given by Equation 26, where n is the number of state variables. As n increases, the distribution of the likelihood narrows (Smith, 2019). This results in some particles with extremely large weights compared to others, inducing sample impoverishment during the resampling step.

An important consideration that has been given widespread attention in literature is how to enhance the efficiency of PFs and ensure that the computational burden incurred is justified. There have been several studies which investigate on-line adaptive selection of the number of particles to improve the filters efficiency. A common method in literature is to monitor the sum of the non-normalized likelihoods, known as the likelihood approach to adaptive particle filtering (Straka & Miroslavšimandl, 2006) (Shao et al., 2009) (Abbeel, 2020). This method is based on the idea that the likelihoods indicate mismatch between the importance density and the posterior distribution (Fox, 2003). A large sum of likelihoods indicates a sufficient overlap between the importance density and the measurement distribution.

Likelihood methods aim to achieve a constant high likelihood by adaptively increasing the number of particles used in the algorithm when the likelihood drops below a pre-defined threshold. The goal is to cause sufficient overlap of the distribution by increasing the number of particles rather than increasing the variance of the system by injecting artificial process noise (Bolić et al., 2002).

2.1.6 Moving horizon estimator

In many systems, states and parameters are constrained by physical laws. This is especially prevalent in chemical and biochemical processes when pressures, concentrations, and mole fractions are constrained to non-negative values (Ungarala et al., 2007). The standard EKF, UKF and PF methodologies do not explicitly incorporate bounds, however, there have been methods proposed, such as truncating the Gaussian distribution, which constrain the states but result in poorer estimation accuracy (Patwardhan et al., 2012). One of the proposed solutions is to explicitly incorporate constraints using the MHE.

The original idea of an optimization-based state estimator was proposed by Thomas (1975), however, the modern MHE algorithm was formally proposed by Robertson *et al*. (1996). The MHE quickly gained interest after its proposal in 1996 and research was done into investigating its applicability and performance relative to other established state estimation techniques.

The MHE is derived from the full information estimator. The optimization problem for the full information estimate, as derived by Haseltine & Rawlings (2003) and Larsson (2015), is available in appendix A.6. The full information estimate is derived from the maximum *a posteriori* estimate as:

$$\min_{x_0,\dots,x_k} \left|\left|x_0 - \bar{x}_0\right|\right|^2_{P_0^{-1}} + \sum_{j=1}^{k} \left|\left|y_j - h(x_j)\right|\right|^2_{R^{-1}} + \sum_{j=0}^{k-1} \left|\left|x_{j+1} - f(x_j)\right|\right|^2_{Q^{-1}} \qquad [30]$$

The optimal full trajectory of states, $\{\hat{x}_0, \dots, \hat{x}_k\}$, from timestep 0 to the current timestep k is solved for by minimizing this objective function in an optimization routine. The objective function consists of three terms: the difference between the initial state estimate and the true value of the initial state, the difference between the measurement prediction using the state estimate and the actual value of the measurement, and the difference between next state estimate and the model prediction for the next state estimate. Each of the three terms are weighted according to their uncertainties. The full information estimate uses all available past measurements to estimate the full state trajectory $\{\hat{x}_0, \dots, \hat{x}_k\}$. Evidently, the dimensionality of the problem quickly becomes intractable as the time increases. The MHE reduces this computational effort of the full information estimate by solving the optimization over a sliding window.

The MHE reformulates the state estimation problem as a least-squares optimization problem by solving for a horizon of state estimates, $\{\hat{x}_{k-H}, \dots, \hat{x}_k\}$, using a horizon of past measurements (Diaz et al., 2017). The objective function of the optimization problem is given by:

$$\min_{x_{k-N},\dots,x_k} Z(\hat{x}_{k-N,k}) + \sum_{j=k-H+1}^{k} \left|\left|y_j - h(x_j)\right|\right|^2_{R^{-1}} + \sum_{j=k-H}^{k-1} \left|\left|x_{j+1} - f(x_j)\right|\right|^2_{Q^{-1}} \qquad [31]$$

(Elsheikh et al., 2021)

The horizon length is H. The smaller the value of H, the smaller the horizon and less computational effort is required. A larger the value of H means a longer horizon length, which may improve the estimation accuracy at the expense of increased computational effort. The remainder of the past measurements and predictions not included in the window are incorporated in an arrival cost, $Z(\hat{x}_{k-N,k})$ (Larsson, 2015). The MHE collapses to the standard KF when applied to an unconstrained linear model with an infinite horizon length (Alexander et al., 2020).

2.1.6.1 Arrival cost

The arrival cost summarizes all the past information that is not included in the horizon. The arrival cost can be assumed zero, however, there is not guaranteed convergence and a large horizon length is required for accurate state estimation (Larsson, 2015). Alexander *et al.* (2020) used this method of excluding the arrival cost from the MHE cost function in a comparative analysis of the MHE. Although the arrival cost was ignored, the MHE still performed well, and showed superior estimation accuracy compared to the EKF.

Alternatively, the arrival cost can be approximated using filtering or smoothing methods. The accuracy of the MHE is severely impacted by crude approximation of the arrival cost (Al-Matouq & Vincent, 2015). Inaccurate approximations of the arrival cost result in a biased state estimate and possibly lead to divergence of the estimator. This is especially prevalent when short horizon lengths are used as the measurement data in the horizon cannot compete with the prior information given by the arrival cost (Haseltine & Rawlings, 2003). The simplest and most common arrival cost approximations are filtering methods of approximation, which are known to be erroneous.

Both filtering and smoothing methods define the arrival cost as:

$$Z(\hat{x}_{k-H,k}) = (\hat{x}_{k-H,k} - \bar{x}_{k-H})^T P_{k-N}^{-1} (\hat{x}_{k-H,k} - \bar{x}_{k-H}) + \rho_{k-H} \quad [32]$$

$\hat{x}_{k-H,k}$ is the horizons initial state estimate in the optimization routine. $\bar{x}_{k-H}$ is the approximated estimate for the initial state in the horizon and P_{k-H} is the associated covariance. ρ_{k-H} is an additional term which is used to account for overlap of measurements between the horizon and the window, used in the smoothing method. Therefore, $\rho_{k-H} = 0$ for all filtering methods. (Elsheikh et al., 2021)

a) Filtering method

The filtering method involves minimizing the difference between the initial state estimate in the horizon and a single approximated state estimate (Haseltine & Rawlings, 2003). The single state estimate, $\bar{x}_{k-H}$, and estimation error covariance, P_{k-H}^{-1}, are commonly estimated using the EKF approximation, the smoothed EKF approximation, or QR approximation. The EKF approximation is one of the simpler methods of arrival cost approximation and is used extensively in real-world applications. Other filters, such as the UKF, can also be used to approximate the arrival cost. More accurate approximation in nonlinear systems is achieved via the UKF, however, this comes at the cost of increased computational requirements (Qu & Hahn, 2009). Furthermore, for long horizon lengths the benefits of more accurate

approximations are reduced as error in the arrival cost becomes less significant as the horizon length increases.

b) Smoothing method

The smoothing method involves minimizing the difference between a trajectory of state estimates in the horizon and a single state estimate (Haseltine & Rawlings, 2003). Elsheikh *et al.* (2021) found that smoothing methods consistently outperform filtering methods in the presence of poor initial guesses. However, smoothing schemes typically require up to double the computational time compared to filtering schemes (Elsheikh et al., 2021).

2.1.6.2 MHE optimization routine

The optimization problem solves for a horizon of state estimates, $\{\hat{x}_{k-H}, \ldots, \hat{x}_k\}$. Therefore, at each point in time, the optimization routine must be initialized with a guess for each of the state estimates in the current horizon. Initialization for solving the first estimate requires a guess of the state estimates in the initial horizon $\{\hat{x}_1, \ldots, \hat{x}_H\}$. These guesses are typically calculated using one of the other nonlinear state estimators such as the EKF, UKF or PF (Ramalingam, 2013). The accuracy of this initialization has a significant impact on the efficiency and accuracy of the optimization routine. Ramalingam (2013) found that initialization of the MHE algorithm with the PF outperforms initialization with the EKF in a system displaying a high degree of nonlinearities.

Furthermore, several optimization algorithms can be used in the optimization routine of the MHE. Alexander *et al.* (2020) highlight the dependency of the MHE on an appropriate optimization algorithm. The study found that different optimization algorithms have a significant impact on both the estimation accuracy and, most notably, the computational time of the MHE. The study found that the MHE using the *fmincon* optimization routine achieved an average computational time of $46.2\ s$ and the MHE using *modSQP* yielded an average computational time of $2.92\ s$.

2.1.6.3 Horizon length selection

The horizon length is a critical tuning parameter in the MHE algorithm. Longer horizon lengths typically result in more accurate state estimates as they overcome erroneous arrival cost approximations (Al-Matouq & Vincent, 2015). However, the computational effort of the MHE algorithm increases linearly with the horizon length. Therefore, the MHE can quickly become intractable for longer horizon lengths. Inappropriate horizon length selection results in large estimation errors, intractable computational times, inactive constraints on the states resulting in wasted computational effort (Al-Matouq & Vincent, 2015), or potential instability and filter divergence. Literature studies typically investigate the impact of increasing horizon length on the estimation error of the MHE. Another consideration when selecting the horizon length is the time constants of the states in the process model (Qu, 2009).

a) Horizon length vs estimation error

An optimal horizon length should be selected to achieve an appropriate balance between the estimation accuracy and the computational effort. Table 1 summarizes the findings from several literature studies

investigating the relationship between horizon length selection in the MHE and the accuracy of the estimated states.

Table 1: A summary of the results from various literature studies investigating the effect of horizon length on the estimation accuracy of the MHE.

Study reference	Physical process	Horizon length	Estimation accuracy metric	Findings
(Mesbah et al., 2011)	Batch crystallization (Simulated process data)	$3-7$ $(300-700s)$	NRMSE	There was a small decrease of 0.04 in the NRMSE of the estimated state as the horizon length was increased from $300-700s$.
(Larsson, 2015)	CSTR (Simulated process data)	$2-20$ $(0.4-4s)$	MSE	The study showed surprising results of an initial increase in the MSE as the horizon length increased, attributed to the cost function approximation error inducing a large MSE. The MSE then decreased after this initial increase. The MSE under short horizon lengths of 2 was already small, 0.0043 and 0.00428 for concentration and temperature state estimates, respectively. A horizon length of 20 only slightly decreased this MSE to 0.0039 and 0.00418.
(Larsson, 2015)	Thermal power plant (Simulated process data)	$1-7$	MSE	The solver failed under a horizon length greater than 7. The majority of the states MSEs decreased with an increase in horizon length. However, this did not occur for all states and is hypothesized to be a result of the complexity of the dynamic model and sensitivity of the solver algorithm.
(Qu & Hahn, 2009)	Nonisothermal CSTR (Simulated process data)	$3-1$ $(4.2-14s)$	MSE	MSE decreased as horizon length increased.
(Diaz et al., 2017)	Estimation of density and viscosity of mineral slurry (Real plant data)	$5-10$ $(0.5-1s)$	RMS, RSD, IA	All metrics remained relatively constant with an increase in the horizon length. Some metrics showed an increase, therefore, the shortest horizon length was selected.
(Ramalingam, 2013)	Benchmark nonlinear estimation problem (Simulated process data)	$2-10$	MSE	The MSE of the estimated state decreased from 1.95×10^{4} to 1.23×10^{4} as the horizon length increased from $2-10$.

The findings from Table 1 show that typically as the horizon length increases, the estimation accuracy of the MHE improves. However, in more complex systems with a large set of interacting state variables, this is not always the case due to the complex dynamics of the system. It is evident from the studies that long horizon lengths are not commonly used in practice, even in simple systems estimating a small number of state variables. This is a result of the large computational burden incurred by long horizon lengths, that quickly becomes impractical for real-world application. Based on these results, the computational requirements of the MHE for estimation in a complex system involving a large set of state variables are expected to be intractable even for relatively short horizon lengths. This ultimately leads to the horizon length being selected purely based on the computational limitations (Al-Matouq & Vincent, 2015) (Dubois et al., 2018). Based on the literature findings, the horizon length should be selected as the maximum horizon length achievable whilst maintaining an appropriate computation time.

b) Time constants of the dynamic model

For a linear continuous-time system with dynamics defined by:

$$\dot{x} = Ax \qquad [\,33\,]$$

The solution to Equation 33 is given by:

$$x(t) = \exp(At)\, x(0) \qquad [\,34\,]$$

The eigenvalues of the state transition matrix, A, give an indication of the boundness of the system solution and therefore explain the stability of the system (Simon, 2006). A negative eigenvalue indicates the term on the RHS of Equation 33 becomes smaller with time and decreases to zero, indicating the system reaches a new steady state after a disturbance. A positive eigenvalue indicates the term on the RHS will grow exponentially. A zero eigenvalue indicates that the accumulation term on the RHS is independent of the state, x. Thus, the states associated with zero eigenvalues are known as integrators.

The time constants, τ, are calculated from the eigenvalues, λ, of the system matrix A. Whereby:

$$\tau = -\frac{1}{\lambda} \qquad [\,35\,]$$

The time constants represent the speed of the response of a system to reaching a new steady state after a disturbance. The smallest negative eigenvalue corresponds with the slowest time constant. Thus, the state variables associated with the smallest negative eigenvalue have the slowest dynamics. Conversely, the largest negative eigenvalue corresponds with the shortest time constant, and the associated states have the fastest dynamics. When solving for the exact solution of the states, both the longest time constant and shortest time constant should be considered. The largest time constant dictates an appropriate integration interval and the smallest time constant indicates an appropriate step size for integration.

A system is considered to be stiff when the difference between the fastest and longest time constants is large. A stiff system of equations requires long integration intervals using short timesteps, making integration numerically sensitive and resulting in failure of the solver or the solver requiring long computation times. Stiff systems of equations, also known as systems with time-scale multiplicity, are a

common phenomenon in chemical processes (Chen et al., 2011). Stiff systems of equations not only affect the stability of the numerical integration method, but also have a significant impact on optimization routines when the cost function involves computation of the solution of stiff equations, such as in model-predictive control and optimization-based state estimation. When the stiffness of a system is not considered, MPC and the MHE become ill-conditioned and potentially unstable (Yin & Liu, 2017). When the horizon length selected is too short, the system no longer converges (Debnath et al., 2021). Long horizon lengths are required to ensure the MHE algorithm stabilizes in the presence of time-scale multiplicity, which make the optimization problem impractical in real-world application due to the increased computational burden.

Solutions have been proposed for handling stiff systems in the MHE problem, such as a distributed MHE proposed by Yin & Liu (2017) or a modified MHE technique using selective measurements proposed by Wang *et al.* (2017). Debnath *et al.* (2021) proposed a unique technique for handling time-scale multiplicity by splitting the system into slow and fast dynamics and using different state estimation techniques to handle the two subsystems. An EKF was used to handle state estimation of the states with fast dynamics. The EKF rapidly computes these state estimates at the expense of reduced accuracy. Whilst the second subsystem has slow dynamics but requires high accuracy, therefore, the MHE is used for state estimation to accurately handle nonlinearities at the expensive of a larger computational effort. Another technique known as singular perturbation theory is a well-known technique for handling stiff systems in process control applications that involves reduced order modelling based on separating dynamics with different time-scales.

c) Singular perturbation theory

Singularly perturbed systems are characterized as being two-time-scale systems, where the states of the system can easily be separated as fast or slow based on the speed of their dynamics. A nonlinear singularly perturbed system can be modelled as:

$$\dot{x}_s = f_s(x_s, x_f, u, t, \varepsilon) \quad [36]$$

$$\varepsilon\dot{x}_f = f_f(x_s, x_f, u, t, \varepsilon) \quad [37]$$

Where x_s represent the slow states with dynamics f_s. x_f represent the fast states with dynamics f_f. The scalar value ε represents a small value perturbation parameter. The singular perturbation theory first neglects the fast dynamics by setting $\varepsilon = 0$. Thus, the dynamics of the fast variables are instantaneous and this forces the fast state variables to converge to their quasi-steady state. (Subbararn & Calise, 2001)

$$0 = f_f(x_s, x_f, u, t, 0) \quad [38]$$

The singular perturbation theory then introduces the effects of the fast states within the model as boundary layer corrections (Garg et al., 2016).

2.1.6.4 Advantages of the MHE

The major advantage of the MHE lies in its ability to implicitly incorporate constraints on the states and handle the nonlinear equations without approximations. Haseltine & Rawlings (2003) demonstrate the

superior state estimation performance achieved with the MHE over the EKF for estimation of the concentrations in a CSTR. The study highlights that in the case where multiple steady states satisfy the steady-state measurement, the incorporation of constraints in the MHE prevents convergence to physically unrealizable states. Qu (2009) also highlight the benefits of the constrained MHE, as it converges to accurate estimates for state estimation in a batch reactor, whilst the EKF and UKF converge to physically unrealizable states.

Furthermore, the MHE has an intrinsic robustness compared to other state estimation techniques (Alexander et al., 2020). The MHE has been proven to show superior robustness to erroneous initial conditions (Wan & Keviczky, 2010) (Haseltine & Rawlings, 2003).

An additional advantage of the MHE relates to the system observability. The EKF is the standard KF applied to a linearized system, and thus, requires the system to display full linear observability to carry out successful state estimation. Some systems are not linearly observable as they are not observable based on a steady-state measurement (Haseltine & Rawlings, 2003). For these systems, process dynamics are required for observability. These types of systems cause the EKF to fail whilst the MHE allows for sufficient excitation of the states over the horizon period to induce observability in the system. In other words, the MHE can handle non-uniform observability when input excitation over a horizon is required for system observability (Wan & Keviczky, 2010).

2.1.6.5 Disadvantages of the MHE

The MHEs' advantages over other filters lie in its formulation of the state estimation problem as an optimization problem, however, this advantage also proves to be the greatest limitation of the method. The computation burden of solving an optimization problem at each time step, even for short horizon lengths, often proves to be impractical for real-world application (Rao et al., 2001). The computational requirements of the MHE are typically three orders of magnitude larger than the requirements of the EKF (Alexander et al., 2020). For systems that require rapid state estimation, the computational time required for the MHE is often inappropriate. Additionally, for systems with fast process dynamics and short sampling times, solving an optimization problem at every time step is impractical (Campani et al., 2019).

With the major limitation of the MHE being the computational burden, there exist many studies aimed at augmenting the algorithm to reduce the computational effort. Some of these methods include incorporating nonlinear programming algorithms (Zavala et al., 2008), careful selection of the optimization strategy, and reduced-order modelling (Alexander et al., 2020). The simplest solution to reducing computational effort of the MHE is to simplify the process model, commonly done by reduced-order modelling. A lower-order approximation of the original process model can be used for any of the aforementioned filters to estimate a smaller subset of states to reduce the computational requirements of the filter (Simon, 2006). However, by simplifying the original model and reducing the number of state estimates, this can compromise the accuracy of estimates and only obtains a small subset of state estimates and not the full state information. An alternative method, mentioned in sub-section 2.1.6.3, involves separating the system by exploiting time-scale multiplicities of the states and implementing a

distributed MHE (Alexander et al., 2020). This can also be used to significantly reduce the computational requirements of the MHE.

Another limitation of the MHE is the lack of a general guideline on how to implement the algorithm (Alexander et al., 2020). The performance of the MHE in terms of estimation accuracy and computational effort heavily depends on the choice of an optimization method, horizon length, arrival cost approximation, initialization values, and constraint selection. However, the selection of these parameters is complex and, without a general set of heuristics, can require copious time and effort and often ultimately becomes a trial-and-error approach to selection.

2.1.7 Tuning of state estimators

Implementation of the EKF, UKF, PF, and MHE algorithms all require a common set of user-defined tuning parameters to be supplied to the state estimators prior to implementation. The only independent engineering required for implementation of the state estimation algorithms is the engineering design choice for the tuning parameters. These common tuning parameters are the initial state estimate, $\hat{x}_0$, the initial state estimation error covariance, P_0, the measurement noise covariance matrix, R, and the process noise covariance matrix, Q. The PF and MHE have additional tuning parameters unique to the algorithms, namely the number of particles in the PF and the horizon length in the MHE. These parameters are known as tuning parameters of the state estimator as they dictate the convergence of the state estimate, $\hat{x}_k$, to the true value of the state, x_k, thus, dictating the state estimation performance. Selection of these tuning parameters should be carried out using engineering knowledge of the specific system and should consider the desirable characteristics of the state estimator for the specific application.

2.1.7.1 Initial state estimate

For chemical and bioprocesses, the initial state estimate, $\hat{x}_0$, is widely considered a non-critical tuning parameter as poor initialization typically only impacts the initial stages of state estimation and rarely shows long-term effects on the performance (Saha et al., 2011). In short time applications, initialization error can have more serious consequences. Although accurate initialization is desirable, the standard KF algorithm does not necessitate an accurate initial state estimate and convergence of the filter is guaranteed, irrespective of the accuracy of initial conditions (Reid, 2001). Convergence in the presence of initialization error is not guaranteed for nonlinear filters, thus, the accuracy of $\hat{x}_0$ should be more carefully considered in nonlinear state estimation.

The EKFs convergence is impacted by the linearization error. Initial linearization error occurs when $\hat{x}_0$ is far from the actual initial state and the state transition matrix and measurement matrix linearized around $\hat{x}_0$ do not accurately approximate the true system. For the EKF, it has been shown that as the initialization error increases, the state estimation error increases (Schneider & Georgakis, 2013). When the initialization error becomes very large, the state estimate can diverge if the EKF converges to another state far from the true solution (Louédec & Jaulin, 2021). For both the EKF and UKF, when poor initial guesses are made but remain within an acceptable range of the true state, the filters will simply take

longer to converge to the true state. However, when the initial guesses are exceptionally poor, both filters tend to diverge. Convergence can be proven for the PF in the presence of estimation error. However, when poor initial state estimates are made, the PF takes many iterations to reach the true state (Chen et al., 2004). When very poor initial estimates are made, the PF may diverge and never reach the true state (Elfring et al., 2021). In a study done by Haseltine & Rawlings (2003), several examples involving state estimation in batch reactors showed failure of the EKF due to poor initial guesses resulting in convergence to physically unrealizable states. The MHE was tested on all several examples, where it displayed more accurate state estimation and improved robustness against poor initial conditions resulting from the optimization-based nature of the method. Optimization-based algorithms show improved robustness to poor initialization over recursive-based methods as constraints on the states avoid divergence of the filter to physically unrealizable states.

The initial state estimate should be chosen within an acceptable range of the true state to avoid poor estimation performance and possible divergence. The initial state estimate is usually accurate as it is can be obtained from the measurement at time zero or an average of off-line measurements. Additionally, special care should be taken when initializing unmeasured states as they are not corrected in the update step and are more susceptible to long-term effects of initialization error.

2.1.7.2 Initial state estimation error covariance matrix

In theory, the initial state estimation error covariance, P_0, is calculated using Equation 39 (Schneider & Georgakis, 2013).

$$P_0 = E[(x_0 - \hat{x}_0)(x_0 - \hat{x}_0)^T] \quad [39]$$

In reality, the ground truth value of the initial state, x_0, is unknown. It is typically assumed that the model states are uncorrelated, therefore, P_0 is a diagonal matrix and the diagonal elements of P_0 are guessed based on the confidence of the guess made for $\hat{x}_0$. Ensuring P_0 and $\hat{x}_0$ are consistent with one another is regarded as the most important consideration when selecting P_0. If $\hat{x}_0$ is randomly guessed, then P_0 should be large. Conversely, if $\hat{x}_0$ is considered accurate, then the value of P_0 should be small. An inconsistent pair of initial conditions can result in divergence of the filters (Schneider & Georgakis, 2013).

In a study done by Schneider & Georgakis (2013), the EKF was able to converge to the true state when the initial guesses were poor but consistent. When P_0 and $\hat{x}_0$ were inconsistent, the EKF did not converge and the filter displayed large estimation errors. The study also found that the UKF showed marginally better performance than the EKF under inconsistent initialization. Studies done by Geetha *et al.* (2014) and Mangold *et al.* (2007) both show that the UKF displays superior estimation performance than the EKF under erroneous but consistent initial conditions.

2.1.7.3 Measurement noise covariance matrix

The measurement noise covariance matrix, R, is typically acquired prior to filter implementation via off-line sensor calibration information (Mohan et al., 2015). A set of measurement samples are taken from sensors and analyzed to determine the nominal standard deviation on each measurement, σ_{meas}. R is assumed to be a diagonal matrix with diagonal entries equal to σ_{meas}. It is assumed that σ_{meas} is time-

invariant, therefore, a constant R matrix is used throughout the entire state estimation period (Schneider & Georgakis, 2013). R is calculated as:

$$R = diag(\sigma_{meas}^2) \quad [40]$$

Underestimating the value of R can result in the filter having false confidence in the measurements and if the measurements are noisy, the filter may diverge. A smaller value of R results in faster convergence of the KF, however, the estimation error covariance remains large. The PF suffers from exacerbated sample impoverishment when R is small as there is less chance of overlap between *a priori* particles and the measurements (Arulampalam & Ristic, 2000). Imtiaz *et al*. (2006) investigated the effects of measurement noise on the PF and found that there exists a conservative value for R that maintains the integrity of the filter whilst avoiding excessive sample impoverishment.

Overestimating the value of R results in a filter that has false confidence in the model prediction and, in the presence of plant-model mismatch, the filter has a higher chance of divergence (Schneider & Georgakis, 2013). However, this makes the filter more robust to noisy measurements. In the standard KF, convergence is slower and the filter has a decreased bandwidth, but the estimation error covariance is smaller, indicating more confidence in the final state estimates (Jumaa et al., 2010). For a larger R, the PF is more robust to sample impoverishment as there is greater chance of overlap between the prior particles and the measurement.

2.1.7.4 Process noise covariance matrix

Careful selection of R and Q is critical as the ratio of these tuning parameters in the gain term of Equation 8 dictates the state estimators' performance. As mentioned above, a fairly accurate value for R can be easily selected. However, an accurate choice for Q is not as easily defined. Therefore, the process noise covariance matrix, Q, is often considered the most critical tuning parameter (Saha et al., 2011).

As explained in sub-section 2.1.2, the process noise represents the stochastic process partly driving the behaviour of the state variables. At every point in time, there is a model prediction obtained by propagating the previous state estimate over a specified timestep. If the process noise was not considered, the state behaviour over the timestep is driven by the deterministic model. The process noise informs the state estimator about the level of unmodelled change in the state value over the course of that timestep and represents the true random stochastic nature of the state (Mohan et al., 2015). Q should account for any unmodelled disturbances and dynamics of the system (Imtiaz et al., 2006). By artificially injecting additional process noise, in the form of a fictitious large Q, this can be used to account for any uncertainties in the process model that would reflect as modelling error in practice (Simon, 2006)(Reid, 2001).

Q is usually assumed to be a constant throughout the state estimation period. The process noise between the states is assumed to be uncorrelated, therefore, Q is a diagonal matrix. It is common practice that the diagonal elements are initially chosen to be small and by trial-and-error are tuned to ensure good performance of the filter (Schneider & Georgakis, 2013). There exist other methods for estimating Q, such as the method presented by Valappil & Georgakis (2004). The proposed methodology involves on-

line computation of a time-varying, non-diagonal Q based on the covariance matrix associated with the parameters. The time-varying value of Q enhancing the filters robustness to modelling error and improves state estimation accuracy. However, an additional computational burden is expended for this calculation.

Underestimating the value of Q leads to the filter having a false confidence in the model, resulting in divergence of the filter in the presence of plant-model mismatch. The filter has a lower bandwidth and tracks the model closely (Simon, 2006).Overestimating Q gives the filter an improved robustness in the presence of modelling errors (Reid, 2001). For large values of Q, the filter has a high bandwidth and is more responsive filter with faster convergence (Simon, 2006)(Jumaa et al., 2010). A falsely assumed large Q leads to false confidence in the measurements, thus, in the presence of noisy measurements, the state estimates display undesirable oscillatory behavior (Imtiaz et al., 2006). In the PF, a large Q results in more diverse particles after the prediction step. The prior density, used as the importance density in the update step, has a larger variance and sample impoverishment is reduced making the PF more robust to modelling error. Furthermore, adding artificial process noise has been proven to decreases the variance of the posterior distribution, therefore, the final state estimate distribution has a smaller variance (Bolić et al., 2002). However, there exists a bias-variance trade-off as increasing the process noise resulting in less accurate state-estimates by introducing off-model predictions causing bias (Wigren et al., 2018) (Bolić et al., 2002).

2.1.8 Application of state estimators

Once a state estimation algorithm has been implemented, accurate state estimates are readily available to the user. These state estimates can be used directly for process monitoring or in other applications such as model-based control and model-based fault detection.

2.1.8.1 Model-based control

Model predictive control is a feedback control algorithm that uses the dynamic model of a process, plant measurements, known inputs to the process, and knowledge of past control moves to make a prediction for future control actions (Kumar & Ahmad, 2012). This control algorithm requires measurements for the complete set of state variables, which are typically not available. This limitation is overcome using the state estimate to provide MPC will the full state information. The use of a state estimate in the MPC algorithm has been proven to show increased robustness of the controller to measurement noise, whilst retaining good control performance (Ricker, 1990). However, control performance can degrade under high process noise and plant-model mismatch (Mesbah et al., 2011).

2.1.8.2 Model-based fault detection

Model-based fault detection involves two steps: residual generation and residual evaluation (Rahoma, 2021). Residuals are calculated as the difference between state predictions, made by a process model, and the actual values of the state, obtained from the measurements. Residuals can be generated using a parity space approach, observer-based approach, filter-based approach, or a parameter identification approach (Frank, 1990). In the filter-based approach, the residual is equivalent to the innovation term in

the state estimation algorithm. The innovation, γ, is defined as the difference between the measurement at time k and the state estimator prediction for the measurement at time k.

$$\gamma_k = y_k - h(x_k^-) \quad [41]$$

Based on the derivation of the EKF, the innovation sequence is zero-mean white noise with a covariance η_k:

$$\eta_k = H_k P_k H_k^T + R \quad [42]$$

(Chetouani, 2008a)

Residual evaluation then classifies this residual signal as either nominal or faulty. Residual evaluation is typically achieved via statistical tests on the residuals such as CUMSUM, sequential probability ratio testing (SPRT), generalized likelihood ratio (GLR) hypothesis tests, weighted-sum-squared residual (WSSR), χ^2 test, and multiple hypothesis testing (Yang, 2004).

2.1.9 State estimation performance

The performance of a state estimation technique is typically quantified using two metrics; the average computation time per iteration and the estimation error (Alexander et al., 2020).

The computation time per iteration quantifies the computational burden of the filter. The computational time required to obtain a state estimate is calculated as the elapsed time taken to run the state estimation algorithm over one timestep.

The most popular method of quantifying the estimation accuracy in state estimation literature studies is with the mean squared error (MSE) (Schneider & Georgakis, 2013). The MSE represents the average squared error of the state estimates over the entire estimation period. Therefore, the larger the MSE, the poorer the filters estimation accuracy.

The MSE is calculated as:

$$MSE = \frac{1}{T}\sum_{k=1}^{T}(x_k - \hat{x}_k)^T(x_k - \hat{x}_k) \quad [43]$$

Where x_k is the ground truth value for the state at time k and $\hat{x}_k$ represents the state estimate at time k. T is the estimation period over which the MSE is assessed. In appendix A.1 the standard KF is derived from the recursive least squares estimator. The optimal gain, the Kalman gain, is derived by defining optimality as minimizing the sum of squared estimation errors. Therefore, the optimization objective matches the performance measure, the MSE.

The MSE is scale-dependent, therefore, state variables with vastly different scales of units cannot be fairly compared. The MSE can be normalized to account for this. Alternatively, the mean absolute percentage error (MAPE) can be calculated. The MAPE is scale-free metric and is more easily interpretable than the MSE. The MAPE is calculated as:

$$MAPE(\%) = \frac{1}{T}\sum_{k=1}^{T}\frac{|x_k - \hat{x}_k|}{x_k} \times 100 \quad [44]$$

The absolute percentage error (APE) gives the estimation error for each of the states, $x_k - \hat{x}_k$, as a percentage of the ground truth value for the states. The MAPE represents the mean of the APEs over an estimation period, T. Similar to the MSE, the larger the MAPE, the larger the estimation error associated with the particular state estimate and the poorer the estimation accuracy.

Evidently, both the MSE and the MAPE can only be calculated when the ground truth values of the states, x_k, are exactly known. Therefore, the MAPE and MSE metrics cannot be calculated in industrial applications and are used in theoretical studies to assess and compare the performance of state estimation techniques.

2.1.10 Plant-model mismatch

Plant-model mismatch occurs when the process model differs from the real-world plant. This has a significant impact on any method that relies on accurate knowledge of a process model. Model-based state estimation exhibits significant performance deterioration in the presence of plant-model mismatch.

Plant-model mismatch increases the estimation error associated with the prediction step in the state estimation algorithm as the model prediction is less accurate (Bavdekar et al., 2013). This prediction error negatively impacts the accuracy of the state estimates, deteriorating the performance of state estimators. Plant-model mismatch is a common cause of divergence in EKFs in industrial applications (Schneider & Georgakis, 2013). A number of studies have shown that careful tuning of the UKF enhances the robustness of the filter in the presence of plant-model mismatch (Geetha et al., 2014) (Mangold et al., 2009). A study done by Salau *et al.* (2014) compared a MHE and constrained EKF in the presence of unexpected disturbances. The study found that the MHE is more robust to plant-model mismatch as it solves for the optimal state estimate using a horizon of past measurements and not just the most recent measurement as in the recursive estimator. In literature, it has been noted that the PF shows particularly poor robustness to modelling error whilst the EKF and UKF maintain their performance due to the nature of the Kalman-update (Jagadeesan et al., 2011). The PFs lack of robustness against modelling error arises from the sampling step in the algorithm (Rawlings et al., 2018).

Plant-model mismatch reflects as modelling error within the prediction step of the state estimators. Therefore, tuning the value of the process noise covariance supplied to the state estimator to account for potential modelling uncertainties has been shown to improve the robustness of filters to plant-model mismatch (Bavdekar et al., 2013). By inflating the value of the process noise covariance matrix or decreasing the magnitude of the measurement noise covariance, this induces less trust in the process model and more trust in the measurements and reduces the effects of plant-model mismatch on the estimation accuracy (Larsson, 2015).

2.1.10.1 Simulation of plant-model mismatch

Plant-model mismatch occurs naturally in industrial processes. For research studies that involve synthetic generation of plant data, plant-model mismatch can be simulated via structural uncertainty or parametric uncertainty (Bavdekar et al., 2013)(Shyamal, 2018)(Mesbah et al., 2011). The true plant dynamics are represented by some equation:

$$x_{k+1} = f_{plant}(x_k, u_k, p_{plant,k}) \quad [45]$$

The nonlinear model of the process used in the state estimation algorithm is:

$$x_{k+1} = f_{model}(x_k, u_k, p_{model,k}) \quad [46]$$

Structural mismatch occurs when the model is an oversimplified representation of the real-world process (Jagadeesan et al., 2011). Therefore, f_{plant} is structurally different from f_{model}. The simulation of plant-model mismatch via structural uncertainty is not well-documented in literature and has been shown to cause very large deviations in state predictions and excessive process noise.

The most common method of simulating plant-model mismatch in state estimation literature is via parametric uncertainty. Parametric mismatch is caused by incorrect values for the model parameters or time-variant plant parameters that are falsely assumed constant in the model (Schneider & Georgakis, 2013). Parameters used in process models are calculated based on offline measurements from the process. Therefore, noisy measurements or varying process conditions result in inaccurate parameter values (Geetha et al., 2014). Parameters that typically vary due to changing process conditions are heat transfer coefficients (Ferhatbegovic et al., 2012) and reaction kinetic parameters, such as the reaction rate constants and the activation energies (Alexander et al., 2020) (Zavala et al., 2008).

For studies involving simulation of synthetic plant data and processes, plant-model mismatch via parametric uncertainty can be modelled as fixed parametric uncertainty or random variation within the parameters (Valappil & Georgakis, 2004).

Fixed parametric uncertainty is modelled as:

$$p_{plant} = p_{model} \pm \sigma \quad [47]$$

Where p_{plant} are the parameters used to generate synthetic plant data in Equation 45 and p_{model} are the parameters used in the process model supplied to the state estimator in Equation 46. σ represents a fixed standard deviation added or subtracted from the nominal plant parameters, p_{model}, supplied to the state estimator (Valappil & Georgakis, 2004).

Alternatively, the plant parameters can vary with time. This can be modelled as:

$$p_{plant,k+1} = p_{plant,k} + w_{p,k} \quad [48]$$

Where $w_{p,k}$ is a white noise process such that $w_{p,k} \sim N(0, \sigma_p^2)$ (Bavdekar et al., 2013).

A number of state estimation studies have implemented plant-model mismatch via parametric uncertainty. Valappil & Georgakis (2004) and Hsoumi *et al.* (2009) add an uncertainty of 5% to the rate constant, activation energy, and heat transfer coefficient in a CSTR process. Mesbah *et al.* (2011) added parametric uncertainty of 35% to the nucleation rate and crystal growth rate parameters for state estimation of a seeded batch crystallization process. Shyamal (2018) added a fixed value of $\pm 5\%$ to the power factor, base mass transfer coefficient, and oxygen injection factor in an electric arc furnace model.

2.1.10.2 Simulation of process noise

As explained in sub-section 2.1.7.4, in the design of the state estimator careful selection of the process noise covariance, Q, is critical. Q should account for the expected modelling errors associated with the specific system. Q should also reflect the true underlying process noise that represents the stochastic process partly driving the state dynamics. This process noise can be attributed to unmeasured disturbances that are not modelled in the deterministic model of the process (Bavdekar et al., 2013).

Bavdekar (2013) presents three methods for simulating process noise within the system. The process noise can be modelled as unstructured noise affecting the states. Equation 45 now becomes:

$$x_{k+1} = f_{plant}(x_k, u_k, p_{plant,k}) + w_{s,k} \quad [49]$$

Where $w_{s,k}$ is a zero-mean white noise process such that $w_{s,k} \sim N(0, \sigma_s^2)$.

Process noise can also enter the system through unmeasured disturbances with known sources, d.

$$x_{k+1} = f_{plant}(x_k, u_k, p_{plant,k}, d_k) \quad [50]$$

$$d_{k+1} = d_k + w_{d,k}$$

Where $w_{d,k}$ is a zero-mean white noise process such that $w_{d,k} \sim N(0, \sigma_d^2)$.

Lastly, the process noise can enter the system through the manipulated inputs, u_k. Where u_k is the known or computed value of the manipulated inputs that is supplied to the state estimator model of the process. The true value of the manipulated inputs, m_k, is calculated as:

$$x_{k+1} = f_{plant}(x_k, m_k, p_{plant,k}) \quad [51]$$

$$m_k = u_k + w_{u,k}$$

Where $w_{u,k}$ is a zero-mean white noise process such that $w_{u,k} \sim N(0, \sigma_u^2)$.

Bavdekar *et al.* (2013) simulated plant-model mismatch whilst estimating the states of a CSTR using an EKF by corrupting the inputs with zero-mean Gaussian noise. This was only done to the inputs used to generate synthetic measurements, whilst the inputs in the EKF model of the process are assumed to be constant.

2.2 Observability theory and literature review

The following sub-section provides the reader with a comprehensive explanation of system observability. Section 2.2.1 highlights the importance of system observability for state estimation. Section 2.2.2 explains the steps involved in an observability analysis for both linear and nonlinear systems. Lastly, section 2.2.3 describes how singular value decomposition of the observability matrix can be used to provide further insight into the observable and unobservable subspaces of the system.

2.2.1 Observability and state estimation

A system is observable if, given a time series of the outputs, the states of a system can be fully constructed (Ghobara, 2013). More formally defined, a system is globally observable at x_0 if every x in the open

neighborhood surrounding x_0 is distinguishable. Whereby states x_0 and x_1 are distinguishable from one another if there exist unique outputs, $y(x_0) \neq y(x_1)$ (James, 1987).

For an uncontrolled linear continuous-time system:

$$\dot{x} = Ax \quad [52]$$

$$y = Cx \quad [53]$$

The state observer dynamics are represented as:

$$\hat{\dot{x}} = A\hat{x} + Bu + K(y - C\hat{x}) \quad [54]$$

The state estimation problem is formulated by solving for a unique solution for each of the states, x, given the limited outputs of the system, y. Where the goal of the observer is to minimize the difference between y and $\hat{y} = C\hat{x}$, using some controller with gain K. The estimation error, e, is the difference between the state estimate and the true state, $(x - \hat{x})$. The estimation error dynamics are calculated as the difference between $\dot{x}$ in Equation 52 and $\hat{\dot{x}}$ in Equation 54 and substituting Equation 53 for the value of y:

$$\dot{x} - \hat{\dot{x}} = A(x - \hat{x}) + w + K(Cx + v - C\hat{x}) = (A - KC)(x - \hat{x}) \quad [55]$$

The rate of error decay is dictated by the eigenvalues of $(A - KC)$. If the system is observable, we can arbitrarily place the eigenvalues of $(A - KC)$, meaning we can control the rate of convergence of the estimator using K (Bernard et al., 2022). Any unobservable eigenvalues of $(A - KC)$ are not updated in the update step of the state estimator. From Equation 55, it can be deduced that the estimator can only converge to a solution if the system is observable or in the case of unobservable modes, if all of the unobservable modes are stable (detectable) (Bernard et al., 2022). System observability is therefore an essential quality for successful state estimation (Wu et al., 2012).

Unobservable and unstable modes results in filter instability and failure to converge to a solution (Southall et al., 2013). The unobservable and unstable states induce instability during the update step of the filter as the estimation error covariance for unobservable states is unbounded, preventing stable convergence to a solution (Devos et al., 2021).

Furthermore, the degree of observability of states has implications on the performance of state estimators. Firstly, the observability impacts the accuracy of the state estimates as the estimation error covariance matrix is directly linked to the observability of the system. The largest estimation error covariance is associated with the least observable states (Ham et al., 1983) (Chen, 1991). Secondly, when a system contains states with vastly different degrees of observability, the estimation error covariance matrix may become ill-conditioned and no longer positive semidefinite (Simon, 2006). Ill-conditioned state estimation problems result in the state estimator being sensitive to measurement noise and to numerical inaccuracies such as round-off error (Sui & Johansen, 2011).

2.2.2 Observability analysis

System observability can be tested via an observability analysis. The first step in the observability analysis is to construct the observability matrix. System observability is then assessed by performing a rank test on the observability matrix.

2.2.2.1 Linear observability

The observability matrix for a linear discrete-time system is derived in appendix A.7. Where O is the discrete-time linear observability matrix, constructed by analyzing the measurement matrix, H, at different discrete time points in the system dynamics.

$$O = \begin{bmatrix} H \\ HF \\ \vdots \\ HF^{n-1} \end{bmatrix} \quad [56]$$

Appendix A.8 presents the derivation for the linear continuous-time observability matrix in terms of the continuous-time state matrix, A, and continuous measurement matrix, C.

$$O = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix} \quad [57]$$

2.2.2.2 Nonlinear observability

For non-linear systems, deriving the observability matrix proves more difficult. In fact, there is no general test for observability for non-linear systems (Southall et al., 2013). The most common method of assessing the observability of non-linear systems is by linearizing the model around a nominal trajectory and performing a linear observability analysis (Huang et al., 2020)(Wu et al., 2012) (Alaeddini & Morgansen, 2013). Alternatively, a nonlinear observability analysis can be conducted on the observability matrix generated using the Lie derivatives of the continuous non-linear function.

Sub-section 2.2.2.3 derives the linearized observability matrix and sub-section 2.2.2.4 derives the nonlinear observability matrix from the Lie derivatives.

2.2.2.3 Linearized observability matrix

For a nonlinear continuous-time system:

$$\dot{x} = f(x, u) \quad [58]$$

$$y = h(x) \quad [59]$$

The linearized model is given by a first order Taylor series approximation around x_0.

$$\dot{x} = Ax + Bu + Lw \quad [60]$$

$$y = Cx + Mv \quad [61]$$

$$A = \frac{\partial f}{\partial x}\Big|_{x_0} \qquad B = \frac{\partial f}{\partial u}\Big|_{u_0} \qquad L = \frac{\partial f}{\partial w}\Big|_{w_0} \qquad C = \frac{\partial h}{\partial x}\Big|_{x_0} \qquad M = \frac{\partial h}{\partial v}\Big|_{v_0}$$

The linearized model is then used to obtain the local observability matrix evaluated at x_0.

$$O(x_0, u) = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix} \quad [62]$$

Wu *et al.* (2012) explain the implications of working with a linearized model when conducting an observability analysis. If the linearized observability matrix is full rank, this proves local observability of the system and does not prove global observability. James (1987) state that a system is locally observable at x_0 if there exists a neighborhood of x_0 such that every x in that neighbourhood has a measurable control, u, that result in a unique output. Thus, local observability does not prove global observability of the system, but rather guaranteed observability at a specific point which the linearized matrices were evaluated at (Chen, 1991).

2.2.2.4 Nonlinear observability matrix

Hermann & Krener (1977) developed a test for nonlinear observability using Lie algebra, presented in appendix A.9. This test is known as the *observability rank condition*.

The local nonlinear observability matrix at x_0 for a system with n_y measurements is defined as:

$$O(x_0, u) = \begin{bmatrix} dL_f^0 h_1 \\ \vdots \\ dL_f^0 h_{n_y} \\ \vdots \\ dL_f^{n-1} h_1 \\ \vdots \\ dL_f^{n-1} h_{n_y} \end{bmatrix} \quad [63]$$

The *observability rank condition* states that if the observability matrix formed from the Lie derivatives evaluated at x_0 is full column rank, then the system is locally weakly observable at x_0 (Martinelli, 2011).

Hermann & Krener (1977) explain the relationship between the various forms of observability. Local observability implies local weak observability, however, the converse is not guaranteed. This means that a system that is found to be locally observable is always locally weakly observable. Conversely, a system that is locally weakly observable is not always locally observable.

This means that if a system is found to be locally observable via the linearization method, this is a sufficient claim for observability. If a system is found to be locally unobservable via the linearization method, it should then be checked for local weak observability via the Lie derivatives.

Aleddini & Morgansen (2013) prove this concept through multiple tests that show the nonlinear observability matrix constructed via the Lie derivatives tends to show higher degrees of observability than the linearized observability matrix. Thus, the linearized observability matrix can sometimes give misleading results. Another study done by Martinelli (2011) found a system to be unobservable based on the local definition obtained via the linearized observability matrix, however, the system was found to be locally weakly observable using the observability matrix constructed from the Lie derivatives.

The computation of Lie derivatives proves impractical for high-dimensional systems as algebraic computation of higher order Lie derivatives requires long computational times and sufficient memory (Stigter et al., 2017). This expensive computation arises when symbolic differentiation is used to calculate the Lie derivatives (Röbenack & Reinschke, 2000).

2.2.2.5 Rank test

The second step in the observability analysis involves performing a rank test on the observability matrix. The observability matrices for linear discrete and continuous-time systems are defined in Equations 56 and 57, respectively. For nonlinear systems, the observability matrices are derived by linearization or the Lie derivatives in Equations 62 and 63, respectively. The observability matrices are defined according to the equation:

$$\begin{bmatrix} y(0) \\ y(1) \\ \vdots \\ y(n-1) \end{bmatrix} = Ox_0 \qquad [\,64\,]$$

The left hand side of the equation is a vector of $n_y \times n$ linear equations and x_0 is vector with size $n \times 1$, containing the n state variables. The observability matrix must have n columns to give n unique solutions for x_0 in Equation 64 (Rutgers Electrical & Computer Engineering, 2007). This results in an observability matrix with $n_y \times n$ rows and n columns. A system is considered observable if the rank of the observability matrix, the number of linearly independent columns, is equal to n.

For the nonlinear observability matrix defined by the gradient of the Lie derivatives, the system is considered observable if the rank of the observability matrix is equal to n. Often the nonlinear observability matrix becomes full rank prior to inclusion of the $(n-1)^{th}$ order Lie derivative. Therefore, it is common practice to recursively construct the observability matrix and test the rank after each higher-order Lie derivative has been added (Villaverde et al., 2019).

A system is considered unobservable if the number of linearly independent columns of the observability matrix is less than n. Unobservable states can arise when one of the columns consists of only zeros, indicating that the observation data has no measurements for the corresponding state and the dynamic relationships with measured states are not sufficient to achieve observability (Nakhaeinejad & Bryant, 2011). Unobservable states can also result from the columns of the observability matrix being linearly dependent, meaning that column vectors can be made up of linear combinations of other column vectors and states are indistinguishable from one another. In practice there also exists measurement noise, missing measurements, and plant-model mismatch which can cause a system to become unobservable, especially if a system is close to unobservability (Nakhaeinejad & Bryant, 2011).

2.2.3 Singular value decomposition

The rank test enables classification of a system as observable or unobservable. However, this gives no indication of the degree of observability of the states of the system and which states may be causing

potential lack of observability, or if the system may be close to unobservability (Nakhaeinejad & Bryant, 2011).

The observability matrix can be analyzed through a matrix decomposition method known as singular value decomposition (SVD). SVD is performed on the discrete observability matrix. The discrete observability matrix contains information on how perturbations in the states manifest within the discrete measurements (De Santis & Di Benedetto, 2017).

The rank of the observability matrix is assessed by analyzing the singular values obtained from SVD. Furthermore, Stigter & Molenaar (2015) explain that SVD acts as a 'magnifying glass' to further assess the observability by highlighting the state variables associated with the observable and unobservable subspaces.

2.2.3.1 SVD procedure

Eigenvalue decomposition is a well-known method of matrix decomposition that decomposes a square matrix, X, of size $n \times n$ into a $n \times n$ matrix Q with the columns corresponding to the eigenvectors, u_i, of X and a diagonal matrix Λ containing the eigenvalues, λ_i. SVD is a method of matrix decomposition that is not limited to square and symmetric matrices.

If X is a $m \times n$ non-symmetric matrix, with n column vectors.

$$X = \begin{bmatrix} | & | & | \\ x_1 & \dots & x_n \\ | & | & | \end{bmatrix} \quad [65]$$

$X^T X$ is a $n \times n$ symmetric matrix that can be decomposed using eigenvalue decomposition. Eigenvalue decomposition on $X^T X$ gives the eigenvectors, v_i, and eigenvalues, λ_i, where $i = 1, \dots, n$. It is assumed that the eigenvectors are normalized so that $v_i^T v_i = 1$. The vectors Xv_i give the n directions of stretching or shrinking. The set of vectors Xv_i form an orthogonal basis for $col(X)$, the set of all linear combinations of columns in X. The 2-norms of the Xv_i vectors are given by:

$$\left||Xv_i|\right|^2 = v_i^T X^T X v_i = v_i^T \lambda_i v_i = \lambda_i \quad [66]$$

The singular values represent the lengths of the vectors Xv_i, given by:

$$\sigma_i = \sqrt{\left||Xv_i|\right|^2} = \left||Xv_i|\right| \quad [67]$$

The singular value decomposition of $m \times n$ matrix X is given by:

$$X = U\Sigma V^T \quad [68]$$

V is a $n \times n$ orthonormal matrix of the eigenvectors of $X^T X$. The columns of V are v_i, where $i = 1, \dots, n$, and are called the right singular vectors.

$$V = \begin{bmatrix} | & | & | \\ v_1 & \dots & v_n \\ | & | & | \end{bmatrix} \quad [69]$$

Σ is a $m \times n$ diagonal matrix of the singular values of $X^T X$ with the rest of the rows filled with zeros to make it have m rows. The singular values are ordered in descending order, $\sigma_1 > \cdots > \sigma_n$.

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_n \\ 0 & 0 & 0 \end{bmatrix} \quad [70]$$

U is a $m \times m$ orthonormal matrix of the vectors u_i, called the left singular vectors.

$$U = \begin{bmatrix} | & | & | \\ u_1 & \dots & u_m \\ | & | & | \end{bmatrix} \quad [71]$$

The left singular vectors u_i are formed by normalizing the vectors Xv_i by diving by the length of the vectors, or diving by the singular value.

$$u_i = \frac{Xv_i}{\sigma_i} \quad [72]$$

By expanding Equation 68, the original matrix X can be written as a weighted linear combination of n $m \times n$ matrices. These matrices, $u_i v_i^T$, have a rank of 1 and are weighted according to their corresponding singular value.

$$X = \sigma_1 u_1 v_1^T + \cdots + \sigma_n u_n v_n^T \quad [73]$$

Equation 73 is true when the rank of the matrix is equal to n.

The columns of U have the same shape and physical interpretation as the columns of X and are arranged in descending order in terms of their ability to describe the variance within the columns of X. The underlying patterns in X are extracted in the matrix V. The magnitude of the singular value, σ_i, indicates the corresponding left and right singular vectors, u_i and v_i, ability to describe the information in the original matrix X. (Berkeley Math, 2017)

2.2.3.2 Pre-processing data prior to SVD

Before the observability matrix can be decomposed by SVD, the data in the matrix must first be pre-processed. The observabilities of the states, or the row entries, need to be normalized to prevent the SVD being skewed towards states with larger observability entries (The Pennsylvania State University-Department of Statistics Online Programs, 2018). Entries in the measurement matrix, C, may falsely result in higher observability values due to the units of measurements. Similarly, fast dynamics can result in large values in the state matrix, A, that skew observabilities for rapidly changing states. Therefore, the system model should be scaled prior to formation of the observability matrix and subsequent SVD.

2.2.3.3 Rank test via SVD

SVD can be used to calculate the rank of a matrix. The rank is equal to the number of non-zero singular values in Σ. A singular value with a value of zero, $\sigma_i = 0$, has a corresponding eigenvalue of zero, $\lambda_i = 0$, and the rank of a matrix is equal to the number of non-zero eigenvalues. Therefore, the rank of $X^T X$ is equal to the number of non-zero singular values. By the rank-nullity theorem, the rank of $X^T X$ is equal

to the rank of X (Bagheri, 2020). This rank test can be used to determine if a system is observable. The number of non-zero singular values of the observability matrix should be equal to the number of columns in the matrix, or number of state variables, n, to be considered full rank and therefore observable. When the rank is less than n, or the last few singular values are close to zero, this indicates unobservability or nearly unobservable directions.

The singular values also give insight into the expected estimation error associated with the state variables. The smallest singular value, σ_{min}, indicates how close the observability matrix is to being singular and corresponds with the maximum estimation error covariance. Singularity indicates unobservability. The largest singular value, σ_{max}, is associated with the most observable direction and the minimum estimation error covariance. The ratio of the largest to the smallest singular values is known as the condition number of the matrix.

The condition number of a matrix is defined as:

$$\kappa = \frac{\sigma_{max}}{\sigma_{min}} \qquad [\ 74\ ]$$

The condition number is always ≥ 1 since the singular values are always real and nonnegative. The condition number indicates how close the observability matrix is to singularity, or, in other words how ill-conditioned the matrix is. As $\kappa \to \infty$, the matrix becomes closer to singularity. A smaller number close to 1 indicates a better conditioned matrix. (Simon, 2006)

2.2.3.4 Multivariate data analysis

SVD is also used as a tool for multivariate data analysis by extracting underlying patterns in large sets of data (Wall et al., 2003). The results of SVD can have physical interpretations that can be used to analyze and understand patterns in the original data.

Performing SVD on the observability matrix results in:

$$O = U\Sigma V^T \qquad [\ 75\ ]$$

The observability matrix has m rows, where $m = n_y \times n$. Progressing from row 1 to row m represents the observability changing 'in time' as the dynamics of the system progress. This can be interpreted as a direction of observability. The left singular vectors, u_i, have the same physical interpretation and therefore are considered the different directions of observability ordered in descending order in terms of their degree of observability.

The observability matrix has n columns representing the n states of the system. Each right singular vector, v_i, contains n rows which correspond to the n states in the original matrix O (Sui & Johansen, 2011).

The matrix Σ is a $m \times n$ matrix containing the singular values, σ_i, ordered in descending magnitude. The magnitude of the singular value, σ_i, indicates to the degree of observability of the corresponding observable direction, u_i (Nakhaeinejad & Bryant, 2011). The most observable direction is associated with the first and largest singular value, σ_1, and the least observable direction with the last and smallest singular value, σ_n.

The observable, unobservable, and nearly unobservable directions can be analyzed in detail to understand which states are associated with to those directions. This is done by analyzing the column vector v_i corresponding to the singular value σ_i. The non-zero elements of v_i indicate which columns in O that are linearly dependent. The columns in O represent the states of the system. The non-zero entries in v_i are involved in a total correlation that contribute to the observable direction u_i (Stigter et al., 2017). Thus, the magnitude of the entries in v_i dictate the magnitude of a specific states' contribution to the $i'th$ observable direction.

The non-zero elements of v_n are the correlated states associated with the least observable direction. The observability of the states associated with the least observable direction can be improved upon by adding additional sensors that directly measure these states, thus, improving observability of the system (Nakhaeinejad & Bryant, 2011).

2.3 Fault detection theory and literature review

This sub-section provides a summary of the relevant literature and theory pertaining to fault detection in chemical processes. Section 2.3.1 gives a brief introduction to fault detection in general. Section 2.3.2 introduces model-based fault detection, with specific focus on literature studies investigating model-based fault detection using state estimators. Section 2.3.3 explains data-driven fault detection and section 2.3.4 describes the concept of combining model-based and data-driven approaches for fault detection. Section 2.3.5 outlines the procedure for performing principal component analysis and calculating various monitoring statistics. Section 2.3.6 details how fault detection performance is typically assessed in literature. Lastly, section 2.3.7 discusses fault detectability, with reference to both structural detectability and performance-based detectability.

2.3.1 Introduction to fault detection

Rapid and accurate detection of abnormal process conditions is essential in any industrial process to maintain the integrity of the process. Process variables should ideally operate between a range of acceptable nominal operating conditions (NOCs). However, deviation from normality is a regular occurrence in real-world processes and can potentially cause poor product quality, equipment degradation, or dangerous operating conditions that pose a risk to plant personnel. Fault detection methods can be categorized into two broad groupings: model-based fault detection and data-driven fault detection.

2.3.2 Model-based fault detection

Model-based approaches rely on a mathematical model of the process and include qualitative approaches such as causal-models and fault trees, or quantitative approaches, which involve generating residuals. Quantitative approaches include observers, such as the unknown-input observer, filters, such as the Kalman filter, parity space approaches involving consistency checks, or frequency domain approaches (Yang, 2004). Quantitative model-based fault detection involving residual generation via state estimation will be the focus of this study.

Li *et al.* (2020) provide an in-depth review of model-based fault detection techniques. The review highlights the availability of extensive literature studies on model-based fault detection techniques and laboratory-scale tests employing these methods. However, the review notes that industrial applications are limited. Model-based fault detection faces many challenges, mainly attributed to the difficulty of modelling complex dynamic processes (Alrowaie et al., 2012). Modelling error arises when the model driving the actual plant process differs from the model used in the model-based method. Modelling error can arise due to structural differences or parametric uncertainties. This mismatch between the plant and the process model induces large residuals under nominal conditions causing high false alarm rates and degrading the performance of model-based fault detection. In real-world applications, the presence of minor modelling uncertainties can severely degrade the fault detection performance (Tyler et al., 2000). Therefore, robustness to model uncertainty is one of the most important considerations in model-based fault detection (Gautam et al., 2019).

2.3.2.1 Model-based fault detection using state estimation

Within the model-based fault detection literature space, there exist several studies investigating the use of state estimators. Table 2 presents a summary of these studies, with specific focus on studies using the EKF, UKF, PF and MHE.

Table 2: Several studies involving model-based fault detection using state estimation.

Study reference	Study application	Residual generation	Residual evaluation method
(Gautam et al., 2019)	Sensor incipient fault detection and isolation of a simulated nuclear power plant	EKF	The Kullback-Liebler Divergence (KLD) between the residual pdf from the current operation and a reference residual pdf representing NOCs is the fault detection statistic. A KLD value greater than a pre-defined threshold indicates faulty behaviour.
(Chetouani, 2004) (Chetouani, 2008a) (Chetouani, 2008b)	Fault detection and isolation in an experimental CSTR	Bank of EKFs; a single EKF representing the fault-free model, and an EKF for each faulty condition.	Statistical test on the mean of the normal law of the standardized innovation sequence, $\widehat{\widehat{\eta s}}$. $\widehat{\widehat{\eta s}} = \frac{1}{N}\sum_{j=1}^{N} \eta s_j$ Where ηs_k is the standardized residual. $\eta s_k = \left(H_k(\hat{x}_k^-)P_k^- H_k^T(\hat{x}_k^-) + R\right)^{-\frac{1}{2}}\left(y_k - h_k(\hat{x}_k^-)\right)$ $= \eta_k^{-1/2}\gamma_k$ N is the size of the horizon of past residuals.

			Under nominal conditions, $\overline{\eta s}$ has a gaussian distribution with zero mean and covariance $\frac{I}{N}$. Abnormal process behaviour is identified when $\widehat{\overline{\eta s}} \succ \alpha$, where α is a pre-defined threshold.
(Mehra & Peschon, 1971)	Theoretical development of innovations approach to fault detection and diagnosis	KF	The innovation sequence from the KF is tested by: 1) Tests of whiteness: innovation sequence is assumed to be a white noise sequence. The autocorrelation function at time k, c_k, can be estimated as: $\hat{c}_k = \frac{1}{N}\sum_{j=k}^{N}(\eta_j - \hat{\bar{\eta}})(\eta_{j-k} - \hat{\bar{\eta}})^T$ Where $\hat{\bar{\eta}}$ is the sample mean: $\hat{\bar{\eta}} = \frac{1}{N}\sum_{j=1}^{N}\eta$ For the sequence to be white noise, the autocorrelation function $\hat{c}_k$ for k = 1,2, … must be normal with zero mean and covariance $1/N$. 2) Tests on the mean of the normal law on the innovation sequence: $\hat{\bar{\eta}} = \frac{1}{N}\sum_{j=1}^{N}\eta$ $\hat{\bar{\eta}}$ must be zero-mean and covariance $1/N$. 3) Tests on the covariance: covariance of the innovation sequence can be estimated by: $\hat{c}_o = \frac{1}{N}\sum_{j=1}^{N}(\eta_j - \hat{\bar{\eta}})(\eta_j - \hat{\bar{\eta}})^T$ This must be equal to the identity matrix. Abnormal behaviour is identified when these tests fail. $\widehat{\overline{\eta}} \succ 1 \cdot 96I/\sqrt{N}$. This is done at a 5% significance level to avoid false alarms.
(Li & Olson, 1991)	Fault detection and diagnosis in	EKF for combined	Faulty behaviour is identified when the difference between the estimated

	a simulated nonlinear distillation process	state and parameter estimation	parameter and its nominal value exceeds a pre-defined threshold.
(Yang, 2004)	Fault detection and diagnosis in a simulated CSTR	Bank of EKFs	Hotellings' T^2 statistic and Q statistic obtained from performing PCA on the EKF residuals. Faulty behaviour is identified when these statistics exceed a pre-defined threshold.
(Li & Kadirkamanathan, 2001)	Theoretical development of fault detection and isolation procedures	EKF	The weighted sum squared residual (WSSR) is used as the fault detection statistic. The residual from the EKF, $\gamma_k = y_k - h(x_k^-)$, has covariance $\eta_k = H_k P_k H_k^T + R$. The weighted squared residual at time k, l_k, is: $l_k = \gamma_k^T \eta_k^{-1} \gamma_k$ The WSSR over a sliding window of length M at time k, d_k, is calculated as: $d_k = \sum_{j=k-M+1}^{k} l_j$ $d_k > threshold$ indicates faulty behaviour.
(Xiong et al., 2007)	Fault detection of satellite attitude sensor fault based on simulated process data	UKF	The standardized innovation sequence from the UKF is assumed to be from a χ^2 distribution. Thus, faulty conditions are identified when the standardized innovation sequence $\widehat{\eta s}$ exceeds a threshold λ selected from the statistics table at a specific confidence level.
(Cornejo et al., 2010)	Fault detection for DELFI-N3XT attitude determination system based on simulated process data	UKF	Faulty behaviour is identified when the CUMSUM of the residuals from the UKF exceeds a pre-defined threshold.
(Das et al., 2014)	Fault detection and isolation in a simulated LEO	UKF	The student t-statistic of each innovation sequence is used to detect a change in the mean of the innovations. Faulty behaviour is identified when the t-statistic exceeds a threshold.

	satellite planar model		
(Kadirkamanathan et al., 2001) (P. Li & Kadirkamanathan, 2001) (Kadirkamanathan et al., 2002) (Souibgui et al., 2011) (Yin & Zhu, 2015)	Theoretical development of particle-based fault detection methodology and implementation for a simulated nonlinear process	PF	Likelihood approach: In the update step of the PF algorithm, at each timepoint, k, a relative likelihood, $q_{i,k}$, is calculated for each particle. Where $i = 1, \dots, N$ for N particles. The likelihood at timepoint k is calculated as: $$L_k = \frac{1}{N}\sum_{i=1}^{N} l_k(i)$$ The decision function is the sum of log-likelihoods over a window of width M. $$d_k = \sum_{j=k-M+1}^{k} \ln(L_j)$$ Once the decision function, d_k, drops below some threshold, the observation at time k is classified as faulty.
(Alrowaie et al., 2012)	Fault detection and isolation for a simulated multi-unit chemical reactor and polyethene reactor system	Bank of PFs	Likelihood approach presented above.
(Gatzke & Doyle, 2001)	Fault detection and isolation for a simulated chemical reactor system	MHE for combined state and parameter estimation	Faulty behaviour is identified when the difference between the estimated parameter and its nominal value exceeds a pre-defined threshold.

The EKF is the most popular choice of state estimator for model-based fault detection. Table 2 highlights a few of these studies employing the EKF for model-based fault detection. Kadirkamanathan *et al.* (2001) first proposed the idea of using the PF as opposed to the EKF for residual generation in model-based fault detection and replaced the WSSR residual evaluation with the likelihood from the PF algorithm. The study compared the EKF and PF for fault detection in a highly nonlinear system where the PF showed superior fault detection performance with a lower false alarm rate. This is attributed to the residual evaluation technique, whereby the likelihood-approach of the PF uses the entire state estimate distribution whilst

the EKF uses the residual generated from the approximate mean (Kadirkamanathan et al., 2001). Studies by Souibgui *et al.* (2011), Chen *et. al.* (2008), and Alrowaie *et al.* (2012) found similar results when comparing fault detection using the PF and the EKF. The likelihood statistic of the PF proved to be more sensitive to faults than the WSSR from the EKF. Furthermore, the studies show that the EKF has a high false alarm rate when compared to the PF as the EKF fails to accurately estimate the highly nonlinear states of the process.

Within literature, a common pitfall of the PF for residual generation in model-based fault detection has been identified. The standard bootstrap PF uses the transition density as the importance density and therefore does not account for the most recent observation. This can make the PF particularly sensitive to outliers and may cause divergence of the filter which compromises the fault detection (Jayaprasanth & Kanthalakshmi, 2018).

2.3.2.2 Process noise covariance and fault detection performance

One of the drawbacks of model-based fault detection using state estimators is the requirement for a priori tuning of the noise covariance matrices to achieve appropriate sensitivity and selectivity (Alrowaie et al., 2012). The measurement covariance matrix, R, is fairly easy to obtain from sensor calibration. Therefore, the critical tuning parameter is the process noise covariance matrix. Careful selection of Q is essential for fault detection performance.

As explained in sub-section 2.1.7.4, the process noise should incorporate any expected potential plant-model mismatch and unknown disturbances. Thus, the selection of Q should be based on engineering knowledge of the specific system as well as considering the specific application of the state estimator. For state estimation applications in model-based fault detection, the choice of Q has a direct impact on the fault detection performance. When Q is overestimated, this can significantly degrade fault detection performance when plant-model mismatch and process noise are not significant. Figure 3 shows how the choice of Q impacts the residual generation and thus the performance of model-based fault detection using state estimation when no modelling error is present.

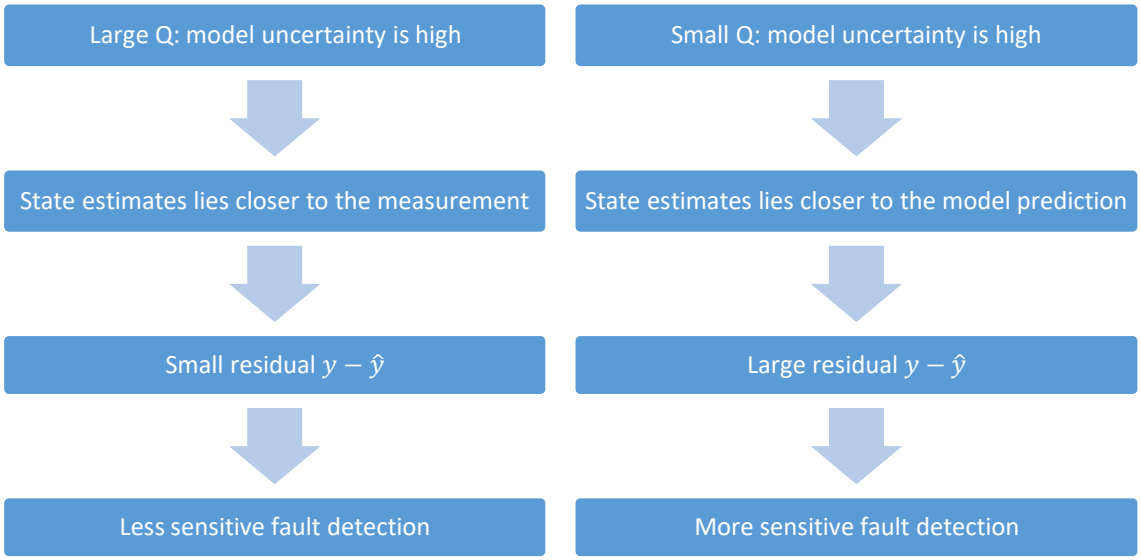

Figure 3: Flowchart showing the effect of using a large Q and small Q for state estimator-based residual generation for fault detection.

It has been found in literature that selecting a smaller Q enables more sensitive and faster fault detection as the state estimator places more trust in the model (Cornejo et al., 2010). For a small Q, the state estimate tracks the model closely and generates large residuals in response to any off-model movement. Conversely, a larger value for Q results in longer detection delays as residuals are more insensitive to faults (Alrowaie et al., 2012). In practice modelling error is a common phenomenon. When a smaller Q is selected, the filter is more sensitive to faults, however, the filter shows poor robustness to modelling error. Thus, the false alarm rates are high in the presence of plant-model mismatch and the filter is less robust. An important consideration is to strike a balance between the state estimators' sensitivity and the robustness (Saha et al., 2014). This is done by selecting realistic values for Q, which requires knowledge of potential parametric and structural uncertainties within the system.

2.3.3 Data-driven FDI

Data-driven fault detection is a form of statistical process monitoring whereby statistical techniques are employed to analyse and monitor process data. Multivariate statistical process monitoring involves calculating statistical metrics that allow for simultaneous monitoring of many process variables. The statistical metrics calculated from process data enable classification of data as either nominal or faulty. The classifier requires training on large sets of historical data to learn nominal or faulty conditions (Khorasgani et al., 2018). One method of training classifiers is using past measurements under nominal conditions. The classifier then labels monitoring statistics deviating from this learned nominal behavior as faulty.

Due to the complexity and scale of industrial operations, multivariate process monitoring typically involves monitoring many process variables. When fitting statistical models to this high dimensional data, a phenomenon known as the 'curse of dimensionality' occurs. The curse of dimensionality was first introduced by Bellman (1975). When a data set is described by many features, more parameters are

required to accurately define the distribution of the data. This is because the data becomes sparser with increasing dimension. Each of these parameters used to define the distribution has an uncertainty associated with them resulting in high dimensional models with high variance. These large sets of process variables are often correlated, resulting in analytical redundancy amongst the process data (Peng et al., 2017).

A method known as feature extraction is often employed prior to calculation of monitoring statistics and subsequent classification. Feature extraction involves generating statistically significant features containing meaningful process information from the measurement data to reduce the computational complexity, reduce redundancy, reduce the effects of noise, and avoid model overfitting (Jung & Sundstrom, 2019). These features are then sent to a statistical classifier which classifies the data as nominal or faulty. Methods such as principal component analysis (PCA), partial least squares (PLS), independent component analysis (ICA), and Fishers discriminant analysis (FDA) are examples of statistical classifiers (Rahoma, 2021). PCA is one of the most popular methods for performing feature extraction in multivariate process monitoring (Tong et al., 2013).

PCA performs feature extraction by constructing uncorrelated principal components (PCs), which represent a multivariable linear correlation structure extracted from the original observations (Theunissen, 2021). The PCs are constructed in such a way to maximize the variance in the original data whilst minimizing the redundancy (Owen & Demirkiran, 2014). The benefits of performing PCA are widely acknowledged in literature. PCA achieves dimensionality reduction by reducing redundancies in the data. This reduces the complexity of the dataset and potentially exposes underlying simple relationships between variables (Owen & Demirkiran, 2014). PCA has also been widely used as a denoising technique and has shown particular robustness against white noise (Spiegelberg & Rusz, 2017). These two benefits of PCA highlight the bias-variance trade-off that PCA aims to control with careful selection of the number of PCs. When performing dimensionality reduction or denoising data via PCA, the high variance of the model incurred due to the curse of dimensionality is reduced by selecting fewer PCs than there are measurements. By decreasing the variance of a model, this reduces the risk of a model overfitting and lacking robustness to new testing data. However, when the variance is reduced via PCA, bias is introduced by assuming the original observations can be accurately represented in a reduced-dimension space. This simplified model may be an incomplete representation of the true process and runs the risk of underfitting new data (Rahoma, 2021)(Kopper et al., 2020). Alternatively, when too many PCs are selected, the PCA model has high variance and may be modelling the noise in the process (Rahoma, 2021).This phenomenon is known as the bias-variance trade-off.

Further limitations of the standard PCA procedure arise from the assumptions of the method. PCA assumes the process variables are independent and uncorrelated in time and models the process as a linear, time-invariant process (Yang, 2004). Static linear PCA can be applied to nonlinear dynamic systems, however, the PCA model is a linear static approximation of the process (Ku et al., 1995). An additional limitation of PCA lies in the interpretability of the results. The PCs of the PCA model are linear combinations of the original process variables, thus, interpreting real-world meaning from the latent variables can be difficult (Rahoma, 2021).

Data-driven methods of fault detection have some advantages over model-based methods in that they require no prior knowledge or understanding of the underlying process and therefore remain unaffected by modelling error. The main limitation of data-driven methods stems from the reliance on having access to large historical datasets. Data-driven methods require sufficient training data under various nominal operating conditions or correctly labelled faulty conditions (Khorasgani et al., 2018).

2.3.4 Combining model-based fault detection and data-driven fault detection

Both model-based fault detection and data-driven fault detection each suffer from their own unique shortcomings. There exists potential for hybrid techniques which combine these two approaches to fault detection that exploit the unique benefits of each approach and overcome the limitations.

Khorasgani *et al.* (2018) define a framework for unifying model-based and data-driven methods for fault detection and diagnosis. This framework forms the basis of the fault detection pipeline presented in Figure 4. The framework divides the fault detection procedure into feature extraction followed by a fault decision-making step. Feature extraction takes place using data-driven techniques or model-based techniques. Fault decision making can be supervised, whereby classification methods use training data to learn fault-free and faulty classes, or unsupervised, by employing hypothesis tests, clustering, or PCA.

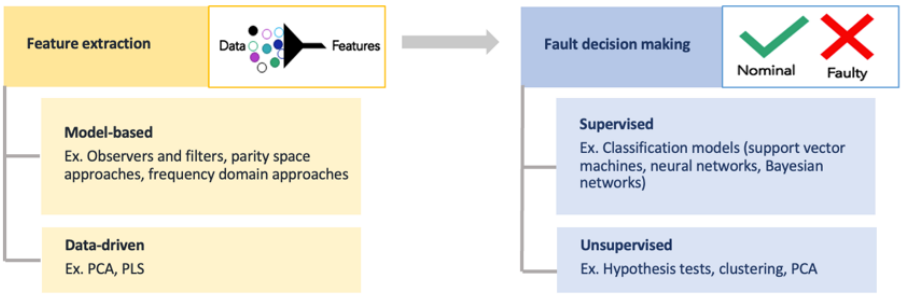

Figure 4: Fault detection pipeline.

There are limited literature studies which combine both model-based and data-driven method of FDI. Studies done by both Sheibat *et al.* (2014) and Khorasgani *et al.* (2018) use an observer for residual generation with a support vector machine for fault classification. Benkouider *et al.* (2012) use an EKF to generate residuals and a neural network for classification.

Yang (2004) carry out model-based fault detection via residual generation using a state estimator, the EKF, and residual evaluation via PCA and monitoring of Hotelling's T^2 statistic and the Q statistic in the reduced space. The study explains that PCA and calculation of the monitoring statistic thresholds assume that observations are independent, Gaussian random variables. Measurements from a process often violate these conditions. Under no plant-model mismatch, the residuals from the EKF obey these conditions. Therefore, it is more appropriate to perform PCA on the residuals than the measurements. Furthermore, the study found this technique of residual evaluation provides enhanced robustness over other statistical tests such as tests of whiteness, mean, and covariance, as these tests are particularly sensitive to unknown disturbances, input variation, noise, and plant-model mismatch.

From Table 2, model-based fault detection via state estimation typically involves residual evaluation using univariate statistical analyses of the residual such as individual t-tests or CUMSUM of each of the residuals. From the state estimator, there are n_y innovation sequences for each of the n_y measurements. In the univariate statistical analyses, each innovation sequence is assessed independent of the other. Therefore, bias is introduced by assuming there is no cross-covariance between the residuals. Multivariate statistical analyses such as Hotelling's T^2 statistic account for cross-covariance, however, this increases the variance of the model as more parameters in the covariance matrix must be assumed. Performing PCA and calculating multivariate monitoring statistics, such as the T^2 statistic, in a reduced dimensional space reduces this variance as the covariance matrix is smaller. However, bias is introduced by assuming the model can be accurately represented in a reduced dimension space. PCA attempts to control this bias-variance tradeoff with careful selection of the number of PCs retained. This may be a more appropriate way of introducing bias into the residual evaluation step, opposed to introducing bias in univariate statistical analyses by assuming there is no cross-covariance between residuals.

2.3.5 PCA

A PCA model is constructed by training the model on a set of historical measurements from the process known as training data. This training dataset often represents observations from the process under nominal operating conditions. The training dataset is represented by the matrix $X \in \mathcal{R}^{n_y \times m}$. Where n_y represents the number of measured process variable available at any point in time and m is the number of timepoints that the training data has been collected for. The principal components (PCs), also known as the loadings vectors p_i, are computed by eigenvalue decomposition of the training data matrix.

$$X^T X p_i = \lambda_i p_i \qquad [\ 76\ ]$$

These directional loadings vectors create a hyperplane in the original n_y-dimension space. When dimensionality reduction is desired, the A most relevant PCs are selected, where when $A < n_y$. A typical rule-of-thumb in deciding an appropriate number of PCs to retain is to ensure that more than 70% of the total variance is explained by the PCs (Tong et al., 2013). Other methods used to select an appropriate number of PCs include utilizing a scree plot, cross validation, or an eigenvalue approach (Tong et al., 2013).

The loadings matrix, P, contains the loadings vectors p_1 through p_A. The loadings matrix is used to project the original observations into this reduced-dimensional space. The projection of observations into the A-dimensional space is given by the scores matrix, T.

$$T = XP \qquad [\ 77\ ]$$

Process monitoring and detection of faulty or abnormal process behavior then takes place on unseen data. Using the trained PCA model, each new observation vector, $x_{new} \in \mathcal{R}^{n_y}$, is projected into the PCA model hyperplane. This projection is represented as the score of the new observation:

$$t_{new}{}' = x_{new}{}' P \qquad [\ 78\ ]$$

Each new observation is monitored via calculation of various monitoring statistics. For a new observation, the monitoring statistic should represent the probability that this new sample is drawn from a distribution generated under nominal conditions. A threshold is selected for the monitoring statistic based on typical statistic values obtained for nominal observations to avoid a high false alarm rate. When the monitoring statistic of a new observation exceeds this threshold, the new observation is classified as faulty. The most popular monitoring statistics used for process monitoring in conjunction with PCA are Hotelling's T^2 statistic and the Q-statistic or squared prediction error (SPE).

2.3.5.1 Hotelling's T^2 statistic

For a univariate distribution, the Student's t-statistic can be used for hypothesis testing. The Student's t-distribution is a normal distribution fit to the training data generated under nominal operating conditions. The t-statistic is calculated for each new observation, which represents the probability of that new observation being drawn from the t-distribution. However, the data available is often a multivariate distribution. Thus, an adaption of the Student's t-statistic known as Hotelling's T^2 statistic is used for multivariate hypothesis testing.

The T^2 statistic is calculated using the scores, t_{new}, of the new observations projected into the trained PCA model:

$$T^2 = \sum_{a=1}^{a=A} \left(\frac{t_a}{s_a}\right)^2 \quad [79]$$

Where A is the number of retained PCs, or the dimension of the hyperplane. s_a is the standard deviation of each of the PC scores and is calculated when the model is being trained. (Dunn, 2020)

The T^2 statistic is the scaled multivariate distance from the center of the PCA model hyperplane to the projection of the new observation onto the hyperplane (Dunn, 2020). The T^2 statistic explains the within-plane error of the new observations or the variation of each new observation within the reduced-dimension hyperplane (Mujica et al., 2011). A large T^2 statistic indicates unusual variability, and thus, abnormal process conditions (Addo, 2019).

It is assumed that all observations are from a Gaussian distribution, thus, under nominal operating conditions the T^2 statistics for each new observation should fall within a certain distance from the mean of the original PCA model hyperplane (Addo, 2019). Faulty conditions are therefore identified as T^2 values exceeding this defined threshold distance.

2.3.5.2 Reconstruction error

The reconstruction error is a vector perpendicular to the plane that represents the out-of-plane error of a new observation (Dunn, 2020). The reconstruction error is the distance between the new observation and the new observations perpendicular projection into the trained PCA model hyperplane. The perpendicular projection of the observation is known as the PCA model predicted value of the observation. The predicted value for each new observation is calculated as:

$$\hat{x}' = t'P' \quad [80]$$

The reconstruction error is defined as the difference between the actual observation and its predicted value:

$$e = x' - \hat{x}' \quad [81]$$

The monitoring statistic is the sum of squared errors for each observation (Dunn, 2020).

This monitoring statistic is responsible for tracking any system conditions not modelled by the trained PCA model (Mujica et al., 2011). Zero reconstruction error indicates that the new observation lies exactly on the PCA model hyperplane. An observation with a large reconstruction error is said to be inconsistent with the model as it lies far from the model hyperplane (Dunn, 2020). A large reconstruction error indicates that the observation violates the linear correlation structure of the PCA model defining nominal conditions (Addo, 2019). Defining the reconstruction error and its thresholds makes no assumptions about the shape of the underlying distribution.

The reconstruction error tends to be a more sensitive monitoring statistic than Hotelling's T^2 statistic as the variance of the T^2 statistic tends to be large (Mujica et al., 2011). High sensitivity of the monitoring statistics enables rapid detection of abnormal behavior.

2.3.6 Fault detection performance evaluation

2.3.6.1 Fault detection metrics

To assess and compare the performance of various fault detection techniques, metrics must be defined that quantify the performance. The goal of fault detection is to correctly flag abnormal process conditions as faulty and NOCs as nominal. The classifier has four outcomes, a true positive (TP) which correctly flags faulty behaviour, a true negative (TN) where nominal behaviour is correctly identified and not flagged, a false positive (FP) where a false warning is given to nominal behaviour, and a false negative (FN) where a fault is missed. Common metrics used to assess the classification performance of a fault detection technique are the sensitivity, specificity, and precision(Salfner et al., 2010).

Precision, ψ, is the probability of an observation correctly being classified as faulty. This is calculated as the number of true positive observations over the sum of true positive and false positive observations.

$$\psi = \frac{TP}{TP+FP} \quad [82]$$

The sensitivity, φ, of a classification model is the true positive rate, calculated as the number of true positive observations over the total number of faulty observations. Sensitivity explains the proportion of the truly faulty observations that were correctly identified as faulty.

$$\varphi = \frac{TP}{TP+FN} \quad [83]$$

The specificity, δ, is the true negative rate, calculated as the number of true negative observations over the total number of nominal observations. This explains the proportion of the true nominal observations that were correctly identified as nominal.

$$\delta = \frac{TN}{TN+FP} \quad [84]$$

2.3.6.2 Receiver operating characteristic curve

In practice, a classification model should have a balance between the sensitivity and the specificity. This balance is dictated by a threshold set on the monitoring statistic, such as the Hotelling's T^2 statistic or the reconstruction error. The threshold on the monitoring statistic defines the point at which an alarm is raised.

For visualization purposes, a receiver operating characteristic, ROC, curve can be generated. A ROC curve is a plot of the sensitivity versus the specificity over a window of threshold values for the monitoring statistic. A perfect classifier would have no false positives and no false negatives, shown by the blue curve in Figure 5. A random classifier would be the orange curve in Figure 5. A classifier shows superior fault detection performance the closer it is to the perfect classifier.

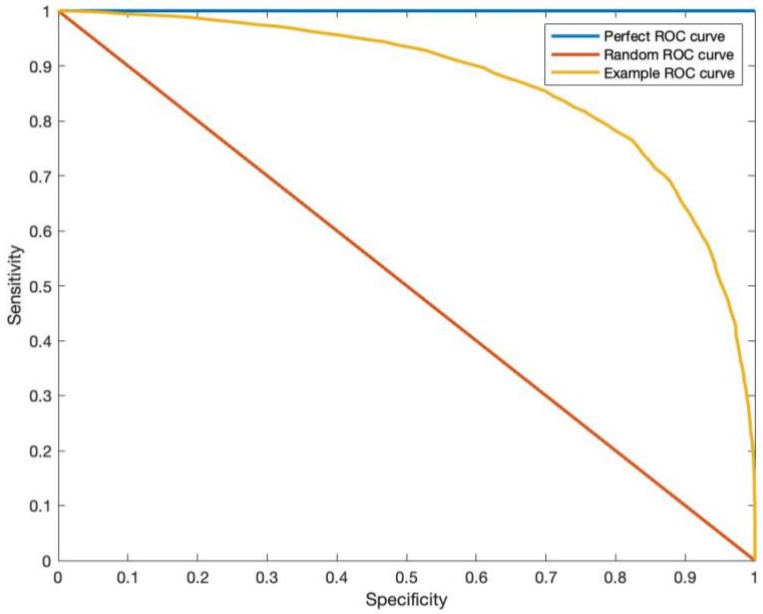

Figure 5: ROC curve for a perfect classification model, a random classification model, and an example of a real classification model.

The area under the ROC curve, AUC, is a metric that incorporates both the sensitivity and specificity of the model over a number of threshold values and can be used to quantify the performance of a classification model. The random classifier has an $AUC = 0.5$ and the perfect classifier has an $AUC = 1$. Good classification performance is inferred by an AUC closer to 1. The parts of the curve on the left occur at low thresholds for the monitoring statistic. At low thresholds, the sensitivity of the classifier is high, however, this can result in false alarms resulting in low specificity (Salfner et al., 2010). At high thresholds, the specificity is high, however, there may be a high rate of missed alarms resulting in low sensitivity.

The selection of this threshold is specific to the real-world application of the classifier. In fault detection applications, the severity of the consequences of the fault being detected dictate an appropriate threshold. In chemical processes, faulty process conditions can have severe consequences such as damage to equipment, plant shutdown, environmental pollution, or dangerous operating conditions threatening the safety of plant personnel. Under these circumstances, thresholds should be low to ensure

the probability of missing a fault is low. However, this results in a common phenomenon known as alarm overloading, where inconsequential or false alarms overwhelm operators (Kaced et al., 2021).

Typically for fault detection applications, low threshold values for the test statistics are not appropriate due to the high false alarm rates resulting in alarm overloading. It is more appropriate to exclude this area at low specificities from the AUC metric to ensure that the models' performance is not inflated (Dodd & Pepe, 2003). Dodd & Pepe (2003) proposed the partial area under the curve, $pAUC$. This metric that excludes the low specificity area by only calculating the AUC for a specific region under the ROC curve between defined specificities δ_1 and δ_2.

$$pAUC = \int_{\delta_1}^{\delta_2} \varphi d\delta \quad [85]$$

The specificities are calculated as to achieve a minimum precision, ψ_{min}. This minimum precision is defined by the user and specific to the application in question.

The lowest specificity at which ψ_{min} can be achieved is calculated as:

$$\delta_1 = 1 + \frac{TP+FN}{FP+TN}\left(\frac{\psi_{min}-1}{\psi_{min}}\right) \quad [86]$$

δ_2 is always 1 as this is the maximum specificity achievable.

For this study, the $pAUC$ is used a performance metric. A higher $pAUC$ is indicative of good fault detection performance, achieving a balance between high sensitivity and high specificity by minimizing false negative and false positive outcomes.

2.3.7 Fault detectability

Fault detectability is an essential criterion for successful detection of faulty process conditions. In literature, fault detectability is typically assessed in two ways: by evaluating the structural detectability or the performance-based detectability (Roy & Dey, 2019). Performance-based detectability investigates fault detectability by considering the performance metrics achieved by a specific fault detection method (Basseville, 2001). Structural detectability refers to the intrinsic detectability of a fault within a specific model of a system, similar to observability or controllability of a system. Structural detectability is independent of the method of fault detection, type and size of the fault, and any disturbances or modelling uncertainties the system may be subject to in practice (Ding, 2008). Developing the procedure for conducting a structural detectability analysis requires an understanding of how the different types of faults manifest within systems and how these faults can be categorized.

2.3.7.1 Fault categorization

In literature, faults are categorized in number of ways. One common method of categorization is to define faults as either additive or multiplicative. This categorization of faults is based upon the manner in which a fault interacts with and affects the states of a system.

The observations obtained from a physical process are the measurements via sensors. Potential faulty conditions reflect within these measurements. When the fault signal is obtained by adding the fault to

the observation, this is known as an additive fault. When the fault is multiplied by the observation signal, this is a multiplicative fault (Chang & Chen, 1995; Sun, 2013). An additive fault will change the expected value of the signal, whilst a multiplicative fault changes the covariance structure of the signal (Zhang et al., 2017). In the case of an additive fault, the influence the fault has on the process variables is independent of the value of the process variables themselves. A multiplicative fault is dependent on the values of the states themselves (Patton et al., 2000).

An additive fault is an unobservable variable that influences the observed variables of the system (Patton et al., 2000). Additive faults can occur at the inputs, outputs, or states. An additive fault can occur at the inputs, via actuator drift for example, and can be modelled for a linear continuous-time system as:

$$\dot{x} = Ax + B(u + f_A) \quad [87]$$

Where f_A represents the fault variable causing a change in the actuator.

Additive faults can also occur at the outputs caused by sensor bias or offset. This is modelled as:

$$y = Cx + f_S \quad [88]$$

Where f_S represents the fault variable causing a change to the sensor output.

Lastly, additive faults can also occur as a result of abnormal disturbances to the process. A fault variable causing abnormal system behaviour in the process, f_P, is modelled as:

$$\dot{x} = Ax + Bu + f_P \quad [89]$$

$$y = Cx + f_P \quad [90]$$

A multiplicative fault results in changes to the system matrices:

$$\dot{x} = (A + \Delta A)x + (B + \Delta B)u \quad [91]$$

$$y = (C + \Delta C)x$$

(Ding, 2008)

Multiplicative faults are caused by changes to the parameters of the system. Parameter values vary due to a change in the process operating conditions, causing changes in parameters such as reaction rates or activation energies (Khorasgani et al., 2014). Significant parameter deviations can also occur as a result of physical damage to plant equipment, causing changes to heat transfer coefficients, resistance coefficients, or mass transfer coefficients (Hsoumi et al., 2009). Multiplicative faults reflect within the system in the same manner as modelling error (Patton et al., 2000). These parameter changes can reflect in parameters within the state matrix, A, affecting the dynamic behavior of state variables. Parameter change in the control input matrix, B, affecting the state-input relationships. Lastly, parameter change can present within the measurement matrix, C, affecting the state-measurement relationships.

2.3.7.2 Structural detectability

For a specific model of a system, structural detectability investigates whether the system observables are able to capture the signature of the fault (Basseville, 2001). Basseville (2001) presents a review on fault

detectability, detailing the various methods for assessing structural detectability of faults in linear models. These methods include analyzing the fault-to-output incidence matrices or transfer functions, calculating the fault-to-noise ratio within the observed output, calculating the distance between the fault-free system model and the faulty system model, or by analyzing the observability or controllability of the state-space.

From sub-section 2.3.7.1, the faults are categorized as an additive or multiplicative fault. Equation 87 represents how an additive fault manifests within a linear system. This additive fault is structurally detectable if the fault-to-output transfer function is non-zero (Roy & Dey, 2019). An additive fault is said to be detectable if the system is completely observable (Basseville, 2001) (Frank, 1990).

From Equation 91, a multiplicative fault can exist as a parameter deviation within the state matrix, A, the control matrix, B, or measurement matrix, C. A multiplicative fault in B is structurally detectable if the input is observable and the multiplicative fault in C is structurally detectable if the output is controllable (Ding, 2008).

The multiplicative fault ΔA results from deviation of model parameters and, thus, causes changes to the state transition structure. The detectability of this multiplicative fault is related to the parameter identifiability of the fault parameters that deviate in ΔA. Parameter identifiability refers to the sensitivity of the observable outputs of a system to potential variation in the model parameter values (Gábor et al., 2017). Various approaches exist for investigating parameter identifiability and detectability of multiplicative faults (Stigter & Molenaar, 2015).

Conducting a sensitivity analysis is a common method of assessing parameter identifiability. A parameter is considered unidentifiable when the sensitivity drops below a pre-defined threshold (Stigter & Molenaar, 2015)(Ghobara, 2013). Another method of assessing structural identifiability of a system was proposed by Tunali & Tarn (1987) whereby system parameters are considered identifiable if the augmented system of states and parameters satisfies the observability rank criterion (Stigter & Molenaar, 2015). This means parameter identifiability can be assessed via an observability analysis by reformulating the state estimation problem to include unknown parameters (Martínez & Villaverde, 2020) (Anguelova, 2004). The structural identifiability, or observability, of the parameters gives an indication of the extent to which specific model parameters reflect within the observables of the model. If the parameter reflects fault conditions, then the structural observability of the fault parameter gives an indication of the expected fault detectability within the measurements.

2.3.7.3 Parameter observability

Parameter observability refers to the ability to infer model parameters from the system outputs. The parameters of a model can be viewed as constant states of the system with dynamic equations of zero as the parameters remain constant in time, $\dot{\theta} = 0$. The parameters are incorporated as additional unknown states of the system. The augmented state-parameter vector, x_{aug}, combines the vector of unknown states, x, and unknown parameters, θ.

$$x_{aug} = \begin{bmatrix} x \\ \theta \end{bmatrix} \quad [92]$$

The state-space representation of the system is given by:

$$\dot{x}_{aug} = \begin{bmatrix} \dot{x} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} f(x,\theta) \\ 0 \end{bmatrix} \quad [93]$$

$$y = f(x_{aug})$$

Parameter identifiability is equivalent to state observability when the parameters are viewed as constant states of the system (Villaverde et al., 2019). A nonlinear observability-identifiability matrix is constructed in the same way as the observability matrix constructed in equation 63.

$$O_I(x_{aug}) = \begin{bmatrix} dL_f^0 h \\ dL_f^1 h \\ \vdots \\ dL_f^{n+q-1} h \end{bmatrix} \quad [94]$$

Where q is the number of parameters incorporated as states.

2.3.7.4 Input observability

Input observability relates to the ability to fully construct the inputs of the system from the outputs. The state vector can be augmented to include the time-varying inputs, $\dot{u}$.

$$\dot{\tilde{x}} = \begin{bmatrix} \dot{x} \\ \dot{\theta} \\ \dot{u} \end{bmatrix} = \begin{bmatrix} f(x,\theta) \\ 0 \\ \dot{u} \end{bmatrix} \quad [95]$$

The observability-identifiability matrix includes the inputs as additional state variables and is constructed in the same manner as in Equation 94. When the inputs are constant, the inputs are viewed as additional unknown constant parameters with zero dynamics. Therefore, the observability matrix is equivalent to the observability-identifiability matrix constructed in equation 94.

2.3.7.5 Performance-based detectability

When an accurate process model is not available, fault detectability cannot be assessed from a structural point of view. The alternative is to assess the detectability of faults using a performance-based methodology. Structural detectability is a necessary but not sufficient condition for practical detectability (Villaverde et al., 2019). This is due to structural detectability being independent of fault size, type, and system noise. For real-world application, conditions such as modelling uncertainties, unknown disturbances, or poor data quality due to measurement noise or missing data, all have a significant impact on the performance of fault detection procedures. Performance-based fault detectability takes these conditions into account as they directly assess a particular fault detection algorithms' ability to capture faulty information within the observations (Basseville, 2001) (Roy & Dey, 2019). One method of performance-based fault detectability is to calculate the fault signal-to-noise ratio (SNR) of the observations.

In a study done by Khorasgani *et al.* (2014), fault detectability for model-based fault detection is assessed using the SNR of each residual. A fault is defined as detectable if at least one of the available residuals is sensitive to that fault, where sensitivity is quantified by the SNR. Basseville (2001) also presents a method

of assessing performance-based detectability based on the fault SNR within the residuals, where high detectability of a fault is inferred by a large fault SNR.

During faulty operation, the observations from a process are made up of a fault signal and noise. The fault signal is the true underlying deviation from nominal behavior and the noise is the random noise associated with the observation that arises due to both process and measurement noise. The signal-to-noise ratio, SNR, is the ratio of the underlying signal to the noise. The fault SNR can be calculated as:

$$Fault\ SNR = \frac{signal}{\sigma_{meas}} = \frac{\mu_{faulty} - \mu_{nom}}{\sigma_{meas}} \quad [96]$$

Where σ_{meas} represents the standard deviation of the measurement noise. The signal is the magnitude of the fault reflected within the measurements, calculated as the difference between the mean of the faulty observations, μ_{faulty}, and the mean of the nominal observations, μ_{nom}. A fault $SNR > 1$ indicates that the fault signal is detectable. A SNR much greater than one indicates that the underlying fault signature within an observation is large compared to the respective noise of that observation, implying high detectability. A fault $SNR \leq 1$ indicates that the fault cannot be reliably detected as the effect of noise is greater than or equal to the effect of the fault within the measurements (Khorasgani et al., 2014). A low SNR is indicative of either a small fault signal, whereby the fault does not significantly reflect in the available observations, or large noise on the observation that masks the fault signal.

3 CASE STUDY: SUBMERGED ARC FURNACE

For application of state estimation techniques, a physical process needs to be selected for which there exists a process model and a method for obtaining historical plant data, either by simulation or from industrial plant measurements. For this study, a submerged arc furnace (SAF) for smelting of platinum group metals (PGMs) is the case study. Section 3.1 provides a brief overview of the PGM smelting process. Section 3.2 presents the process model of the SAF. Section 3.3 provides the measurement model used to generate synthetic noisy process data used for subsequent state estimation and fault detection. Lastly, section 3.4 details various faulty conditions that can potentially arise within the SAF, fault modelling, and fault categorization.

3.1 PGM smelting process

PGMs exist naturally in ores, primarily in the form of sulphide ores. PGMs are extracted from these ores in a complex series of steps. A crucial step in the extraction process is smelting. The smelting phase typically occurs in six-electrode submerged arc furnaces (SAFs) following comminution and flotation (Nell, 2004).

Incoming ore concentrate enters the SAF into a top layer of unsmelted concentrate. SAFs carry out smelting at high temperatures to melt and separate this ore concentrate into two liquid phases; a less dense oxide slag layer and a denser sulphide matte layer. In the liquid concentrate layer, the ores undergo desulphurization where the PGMs and base metal sulphides are converted into matte components. The PGM minerals sink through the slag into the matte layer (Crundwell et al., 2011). The PGM-rich matte phase is tapped from the SAF and undergoes conversion in a Pierce-Smith converter to further concentrate the PGMs. Leaching is then carried out to obtain a leach residue of concentrated PGMs (Jones, 2005).

3.2 SAF process model

There exist several process models of a SAF in literature. Theunissen (2021) explains that the available models are limited to steady-state models (Sheng et al., 1998) (Pan et al., 2011), models which do not describe the entire furnace (Eksteen, 2011), or models that are overly complex with high computational burden (Ritchie & Eksteen, 2011) (Bezuidenhout et al., 2009). Mesbah *et al.* (2011) explains that for the application of state estimation algorithms, a simple system of explicit ODEs is preferred to ease the computational burden of the state estimation problem. A simple dynamic model that consists of explicit equations is desirable, especially in the case of an EKF to facilitate the computation of Jacobians. The model that was chosen for this study is a dynamic model of a six-electrode sulphide smelting SAF developed by Theunissen (2021). For a derivation of the model and model assumptions the reader is referred to Theunissen (2021). The model is a dynamic model of a SAF represented as a system of ODEs.

Theunissen (2021) derived the model based on the methodology presented by Logar *et al.* (2012a, 2012b) by performing mass and energy balances over furnace zones, depicted in Figure 6. The SAF process is a

continuous process whereby the ore concentrate is charged into the furnace and the slag and matte liquid phases are tapped from opposite sides of the furnace (Logar et al., 2012).

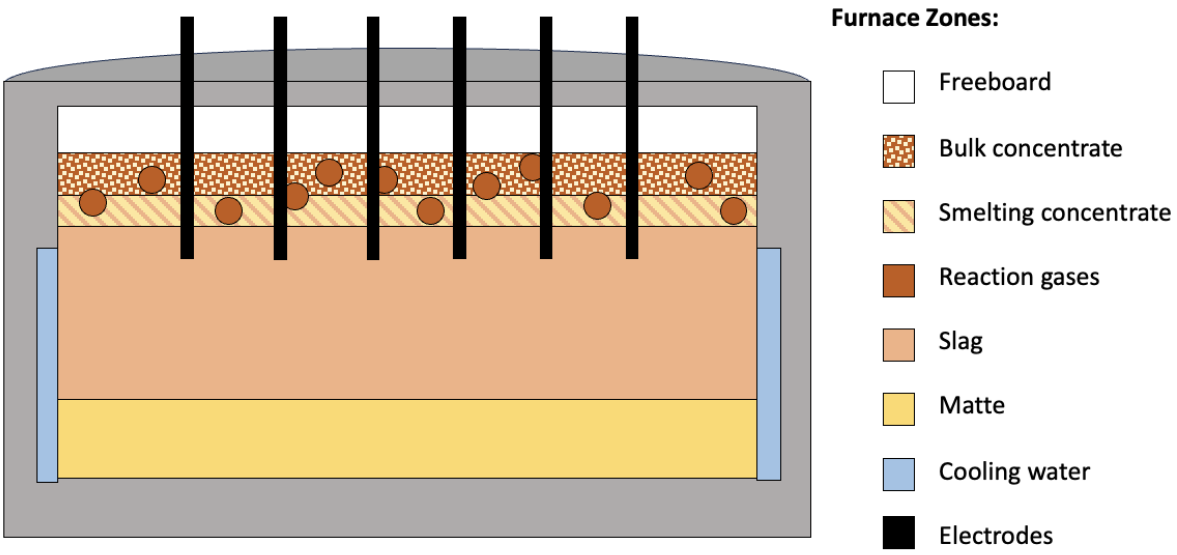

Figure 6: Depiction of distinct zones in the SAF model, adapted from Theunissen (2021).

The model that describes this SAF smelting process consists of a nonlinear system of equations. In state-space representation, the model is represented as:

$$\dot{x} = f(x, u) \qquad [97]$$

With discrete-time measurements:

$$y_k = h(x_k) \qquad [98]$$

Equation 97 represents the nonlinear dynamics, or how the states, x, change with respect to time. The state dynamics are influenced by the states themselves and the inputs of the system, u. Equation 98 describes the nonlinear measurement equations, which describe how the measurements, y, relate to the state variables.

The SAF system consists of 17 state variables that are either molar amounts or temperatures within the different zones of the furnace. The states of the system, x, are defined in Table 3.

Table 3: SAF model state variables.

	Bulk concentrate zone $C(B)$	**Symbol**	**Units**	**State**
1	Moles of slag component in $C(B)$	$N_{C(B),X0}$	mol	x_1
2	Moles of matte component in $C(B)$	$N_{C(B),XS}$	mol	x_2
3	Moles of sulfurized matte component in $C(B)$	$N_{C(B),XS_2}$	mol	x_3
4	Temperature of $C(B)$	$T_{C(B)}$	K	x_4

Smelting concentrate zone $C(S)$				
5	Moles of slag component in $C(S)$	$N_{C(S),X0}$	mol	x_5
6	Moles of matte component in $C(S)$	$N_{C(S),XS}$	mol	x_6
7	Moles of sulfurized matte component in $C(S)$	$N_{C(S),XS_2}$	mol	x_7
8	Temperature of $C(S)$	$T_{C(S)}$	K	x_8
Reaction gases in concentrate zone $C(R)$				
9	Moles of reaction gases trapped in $C(R)$	$N_{C(R)}$	mol	x_9
Slag zone S				
10	Moles of slag in S	N_S	mol	x_{10}
11	Temperature of S	T_S	K	x_{11}
Matte zone M				
12	Moles of matte in M	N_M	mol	x_{12}
13	Temperature of M	T_M	K	x_{13}
Furnace freeboard zone G				
14	Moles of reaction gases in G	$N_{G,R}$	mol	x_{14}
15	Moles of air in G	$N_{G,A}$	mol	x_{15}
16	Temperature of G	T_G	K	x_{16}
Cooling unit zone W				
17	Cooling water temperature	T_W	K	x_{17}

The nonlinear function $f(x,u)$ of equation 97 represents the vector of 17 nonlinear ordinary differential equations that describe how each of the states change over time with respect to the states themselves, x, and the inputs to the process, u. The 17 ODEs which describe this dynamic behavior are presented in Table 4 along with the supporting equations.

Table 4: SAF model ODEs.

	ODEs
1	$\frac{dN_{C(B),XO}}{dt} = F_{charge}x_{charge,XO} - \boldsymbol{F_{mix,XO}}$
2	$\frac{dN_{C(B),XS}}{dt} = F_{charge}x_{charge,XS} - \boldsymbol{F_{mix,XS}} + \boldsymbol{r_{F,C(B)}V_{C(B)}}$

3	$$\frac{dN_{C(B),XS_2}}{dt} = F_{charge}x_{charge,XS_2} - \boldsymbol{F_{mix,XS_2}} - \boldsymbol{r_{F,C(B)}V_{C(B)}}$$
4	$$\frac{dT_{C(B)}}{dt} = \frac{\boldsymbol{Q_{C(S):C(B)}} + \boldsymbol{Q_{G:C(B)}} + c_{P,C}\left(T_{charge} - T_{C(B)}\right)F_{charge}}{\boldsymbol{N_{C(B)}}c_{P,C}}$$
5	$$\frac{dN_{C(S),XO}}{dt} = \boldsymbol{F_{mix,XO}} - \boldsymbol{F_{melt,XO}}$$
6	$$\frac{dN_{C(S),XS}}{dt} = \boldsymbol{F_{mix,XS}} - \boldsymbol{F_{melt,XS}} + \boldsymbol{r_{F,C(S)}V_{C(S)}}$$
7	$$\frac{dN_{C(B),XS_2}}{dt} = \boldsymbol{F_{mix,XS_2}} - \boldsymbol{r_{F,C(S)}V_{C(S)}}$$
8	$$\frac{dT_{C(S)}}{dt} = \frac{\boldsymbol{Q_{S:C(S)}}\left(1 - \frac{T_{C(S)}}{T_{C,melt}}\right) - \boldsymbol{Q_{C(S):C(B)}} + \left(T_{C(B)} - T_{C(S)}\right)\left(\boldsymbol{F_{mix}}c_{P,C} + \boldsymbol{r_{F,C(B)}V_{C(B)}}c_{P,G}\right)}{\boldsymbol{N_{C(S)}}c_{P,C}}$$
9	$$\frac{dN_{C(R)}}{dt} = \boldsymbol{r_{F,C(B)}V_{C(B)}} + \boldsymbol{r_{F,C(S)}V_{C(S)}} + \boldsymbol{r_C} - \boldsymbol{J_R}A$$
10	$$\frac{dN_S}{dt} = \boldsymbol{F_{melt,XO}} - F_{tap,S}$$
11	$$\frac{dT_S}{dt} = \frac{\boldsymbol{Q_J} + \boldsymbol{Q_{M:S}} + \boldsymbol{Q_{W:S}} - \boldsymbol{Q_{S:C(S)}} + \left(T_{C,melt} - T_S\right)\left(\boldsymbol{F_{melt,XO}}c_{P,S} + \boldsymbol{F_{melt,XS}}c_{P,M}\right)}{N_S c_{P,S}}$$
12	$$\frac{dN_M}{dt} = \boldsymbol{F_{melt,XS}} - F_{tap,M}$$
13	$$\frac{dT_M}{dt} = \frac{\boldsymbol{Q_{W:M}} - \boldsymbol{Q_{M:S}} + (T_S - T_M)\left(\boldsymbol{F_{melt,XS}}c_{P,M}\right)}{N_M c_{P,M}}$$
14	$$\frac{dN_{G,A}}{dt} = \boldsymbol{F_{neg.P}} - \boldsymbol{F_{pos.P,A}} - \boldsymbol{F_{ext,A}}$$
15	$$\frac{dN_{G,R}}{dt} = \boldsymbol{J_R}A - \boldsymbol{F_{pos.P,R}} - \boldsymbol{F_{ext,R}}$$
16	$$\frac{dT_G}{dt} = \frac{c_{P,G}\left[\left(T_{C(S)} - T_G\right)\boldsymbol{J_R}A + (T_{atm} - T_G)\boldsymbol{F_{neg.P}}\right] - \boldsymbol{Q_{G:C(B)}}}{\boldsymbol{N_G}c_{P,G}}$$
17	$$\frac{dT_W}{dt} = \frac{c_{P,W}\left(T_{W,0} - T_W\right)F_W - \boldsymbol{Q_{W:M}} - \boldsymbol{Q_{W:S}}}{N_W c_{P,W}}$$
	Bulk concentrate mixing rates
	$$\boldsymbol{F_{mix,XO}} = k_v\boldsymbol{V_{C(B)}}\boldsymbol{C_{C(B),XO}}$$
	$$\boldsymbol{F_{mix,XS}} = k_v\boldsymbol{V_{C(B)}}\boldsymbol{C_{C(B),XS}}$$
	$$\boldsymbol{F_{mix,XS_2}} = k_v\boldsymbol{V_{C(B)}}\boldsymbol{C_{C(B),XS_2}}$$

	Desulphurization and oxidation reaction rates
	$$\boldsymbol{r_{F,C(B)}} = k_F e^{-\left(\frac{E_{A,F}}{RT_{C(B)}}\right)} \boldsymbol{C_{C(B),XS_2}}$$
	$$\boldsymbol{r_{F,C(S)}} = k_F e^{-\left(\frac{E_{A,F}}{RT_{C(S)}}\right)} \boldsymbol{C_{C(S),XS_2}}$$
	$$\boldsymbol{r_C} = k_C e^{-\left(\frac{E_{A,C}}{RT_S}\right)}$$
	Smelting equations
	$$\boldsymbol{F_{melt,XO}} = \frac{\boldsymbol{Q_{S:C(S)}} \frac{T_{C(S)}}{T_{C,melt}}}{\lambda_C + c_{P,C}\left(T_{C,melt} - T_{C(S)}\right)} \frac{N_{C(S),XO}}{N_{C(S),XO} + N_{C(S),XS}}$$
	$$\boldsymbol{F_{melt,XS}} = \frac{\boldsymbol{Q_{S:C(S)}} \frac{T_{C(S)}}{T_{C,melt}}}{\lambda_C + c_{P,C}\left(T_{C,melt} - T_{C(S)}\right)} \frac{N_{C(S),XS}}{N_{C(S),XO} + N_{C(S),XS}}$$
	Molar flow in and out of the freeboard
	$$\boldsymbol{F_{neg.P}} = \begin{cases} k_{PR}(P_{atm} - \boldsymbol{P_G}) & if\ P_{atm} \geq P_G \\ 0 & if\ P_{atm} < P_G \end{cases}$$
	$$\boldsymbol{F_{pos.P,R}} = \begin{cases} \frac{k_{PR} N_{G,R}(\boldsymbol{P_G} - P_{atm})}{N_{G,A} + N_{G,R}} & if\ P_G \geq P_{atm} \\ 0 & if\ P_G < P_{atm} \end{cases}$$
	$$\boldsymbol{F_{pos.P,A}} = \begin{cases} \frac{k_{PR} N_{G,A}(\boldsymbol{P_G} - P_{atm})}{N_{G,A} + N_{G,R}} & if\ P_G \geq P_{atm} \\ 0 & if\ P_G < P_{atm} \end{cases}$$
	$$\boldsymbol{F_{ext,R}} = \frac{k_{PE} N_{G,R}(\boldsymbol{P_G} - P_{ext})}{N_{G,A} + N_{G,R}}$$
	$$\boldsymbol{F_{ext,A}} = \frac{k_{PE} N_{G,A}(\boldsymbol{P_G} - P_{ext})}{N_{G,A} + N_{G,R}}$$
	Heat generation and transfer equations
	$$\boldsymbol{Q_{C(S):C(B)}} = h_{C(S):C(B)} A_{SAF}\left(T_{C(S)} - T_{C(B)}\right)$$
	$$\boldsymbol{Q_{G:C(B)}} = h_{G:C(B)} A_{SAF}\left(T_G - T_{C(B)}\right)$$
	$$\boldsymbol{Q_{S:C(S)}} = h_{S:C(S)} A_{SAF}\left(T_S - T_{C(S)}\right)$$
	$$\boldsymbol{Q_J} = \frac{V_{electrodes}^2}{R_0\left(1 + \alpha\left[T_S - T_{S,0}\right]\right)}$$
	$$\boldsymbol{Q_{M:S}} = h_{M:S} A_{SAF}(T_M - T_S)$$
	$$\boldsymbol{Q_{W:S}} = h_{W:S} \rho_{SAF} L_S (T_W - T_S)$$

	$$\boldsymbol{Q_{W:M}} = h_{W:M}\rho_{SAF}L_M(T_W - T_M)$$
	Expressions used to emulate furnace blowback
	$$\boldsymbol{J_R} = \begin{cases} \left.\begin{matrix}\boldsymbol{J_{R,PBR}}\ if\ \Delta\boldsymbol{P_{C(R):G}} \leq c\Delta\boldsymbol{P_{crit}} \\ \boldsymbol{J_{R,Ch}}\quad if\ \Delta\boldsymbol{P_{C(R):G}} > c\Delta\boldsymbol{P_{crit}}\end{matrix}\right\}\ if\ J_R = J_{R,PBR} \\ \left.\begin{matrix}\boldsymbol{J_{R,PBR}}\ if\ \Delta\boldsymbol{P_{C(R):G}} \leq \Delta\boldsymbol{P_{crit}} \\ \boldsymbol{J_{R,Ch}}\quad if\ \Delta\boldsymbol{P_{C(R):G}} > \Delta\boldsymbol{P_{crit}}\end{matrix}\right\}\ if\ J_R = J_{R,Ch} \end{cases}$$
	$$\boldsymbol{J_{R,PBR}} = \frac{k_{PBR}}{\boldsymbol{L_C}}\Delta\boldsymbol{P_{C(R):G}}$$
	$$\boldsymbol{J_{R,Ch}} = k_{Ch}\Delta\boldsymbol{P_{C(R):G}}$$
	$$\Delta\boldsymbol{P_{crit}} = \rho_{C,bulk}g\boldsymbol{L_C}$$
	Total moles in the bulk concentrate, smelting concentrate, and freeboard equations
	$$\boldsymbol{N_{C(B)}} = N_{C(B),XO} + N_{C(B),XS} + N_{C(B),XS_2}$$
	$$\boldsymbol{N_{C(S)}} = N_{C(S),XO} + N_{C(S),XS} + N_{C(S),XS_2}$$
	$$\boldsymbol{N_G} = N_{G,R} + N_{G,A}$$
	Volumes of the bulk and smelting concentrate
	$$\boldsymbol{V_{C(B)}} = \left(\frac{M_{XO}N_{C(B),XO} + M_{XS}N_{C(B),XS} + M_{XS_2}N_{C(B),XS_2}}{\rho_{C,bulk}}\right)$$
	$$\boldsymbol{V_{C(S)}} = \left(\frac{M_{XO}N_{C(S),XO} + M_{XS}N_{C(S),XS} + M_{XS_2}N_{C(S),XS_2}}{\rho_{C,bulk}}\right)$$
	Concentrations in the bulk and smelting concentrate
	$$\boldsymbol{C_{C(B),XO}} = \frac{N_{C(B),XO}}{V_{C(B)}}$$
	$$\boldsymbol{C_{C(B),XS}} = \frac{N_{C(B),XS}}{V_{C(B)}}$$
	$$\boldsymbol{C_{C(B),XS_2}} = \frac{N_{C(B),XS_2}}{V_{C(B)}}$$
	$$\boldsymbol{C_{C(S),XS_2}} = \frac{N_{C(S),XS_2}}{V_{C(S)}}$$
	Bed height equations
	$$\boldsymbol{L_S} = \frac{M_{XO}N_S}{\rho_S A_{SAF}}$$
	$$\boldsymbol{L_M} = \frac{M_{XS}N_M}{\rho_M A_{SAF}}$$

	$$\boldsymbol{L_C} = \frac{V_{C(B)} + V_{C(S)}}{A_{SAF}}$$
	Bed height control
	$$F_{charge} = \begin{cases} F_{charge,const.} & \text{for } 20\,000s \\ 0 & \text{for } 4\,000s \end{cases}$$ Concentrate bed height is controlled through charging cycles. The concentrate bed is charged by for 20 000s followed by halting of charge for 4000s, followed by repeated charging and halting of charge.
	$$F_{tap,S} = \begin{cases} \left.\begin{matrix} F_{tap,S,const.}\ if\ L_S < L_{S,upper\ lim} \\ 0\ if\ L_S \geq L_{S,upper\ lim} \end{matrix}\right\} if\ F_{tap,S} = F_{tap,S,const.} \\ \left.\begin{matrix} F_{tap,S,const.}\ if\ L_S < L_{S,lower\ lim} \\ 0\ if\ L_S \geq L_{S,lower\ lim} \end{matrix}\right\} if\ F_{tap,S} = 0 \end{cases}$$ Slag bed height is controlled by the slag zone tapping rate. This is done on a switch basis, whereby the slag bed height is controlled between and upper and lower limit.
	$$F_{tap,M} = \begin{cases} \left.\begin{matrix} F_{tap,M,const.}\ if\ L_M < L_{M,upper\ lim} \\ 0\ if\ L_M \geq L_{M,upper\ lim} \end{matrix}\right\} if\ F_{tap,M} = F_{tap,M,const.} \\ \left.\begin{matrix} F_{tap,M,const.}\ if\ L_M < L_{M,lower\ lim} \\ 0\ if\ L_M \geq L_{M,lower\ lim} \end{matrix}\right\} if\ F_{tap,M} = 0 \end{cases}$$ Matte bed height is controlled by the matte zone tapping rate. This is done on a switch basis, whereby the matte bed height is controlled between an upper and lower limit.
	Pressure equations
	$$\boldsymbol{P_G} = \frac{N_G R T_G}{V_G}$$
	$$\Delta \boldsymbol{P_{C(R):G}} = \boldsymbol{P_{C(R)}} - P_G$$
	$$\boldsymbol{P_{C(R)}} = \frac{\boldsymbol{N_{C(R)}} \boldsymbol{R} T_{C(S)}}{\varepsilon_C V_C}$$
	$$\boldsymbol{V_C} = V_{C(B)} + V_{C(S)}$$

The input vector, u, consists of exogeneous model variables that arise from outside the system and are therefore unaffected by the system itself. The input variables of the smelting process are the composition of the incoming concentrate (x_{charge}), the temperature of the incoming concentrate (T_{charge}), the flowrate of the cooling water (F_W), and the temperature of the cooling water supplied to the furnace ($T_{W,0}$). For the process model used in the state estimation algorithms, these inputs are kept constant. Other inputs to the smelting process are the concentrate charging rate (F_{charge}), the matte tapping rate ($F_{tap,M}$) and slag tapping rate ($F_{tap,S}$). From Table 4, it can be seen that the flowrate of the charged concentrate and the flowrate of the tapped slag and matte are controlled on a switch basis based on the

limits placed on each bed height. Their values are either a constant flowrate, when charging/tapping is switched on, or zero, when charging/tapping is switched off.

Using these ODEs, the process is simulated in MATLAB using a built-in ODE solver *ode15s*, chosen due to the stiff nature of this system of equations. The ODE solver requires initial values for the states, presented in Table C.2.

It should be noted that the process model of the SAF developed by Theunissen (2021) presented in Table 4 displays hysteresis. There exist model equations for a 'pre-failure' model, before furnace blowback has occurred, and a 'post-failure' model, after blowback. The process model used within the state estimators will use the pre-failure model of the process where gas flux out of the furnace behaves as a packed bed reactor (PBR), thus $J_R = J_{R,PBR}$ within the differential equations in Table 4. The post-failure model assumes gas flux occurs via channelling $J_R = J_{R,Ch}$, and is used to generate synthetic measurement data under faulty blowback conditions.

3.3 Measurement model

For application of state estimation algorithms, in addition to a process model, measurements from the process are required. In real-world application of state estimation, these measurements will be obtained from the sensors on the plant. For a theoretical study, when industrial data from a real-world process is unavailable, the alternative is to generate synthetic measurement data via simulation using the process model. To generate the measurements via simulation, the measurement variables are sampled from the ODE solver solution at regular intervals and random noise is added to the sample to simulate measurement noise.

There are 9 measured variables in the SAF smelting process. These measured variables represent typically available measurements in industrial SAFs. Table 5 summarizes the measured variables and presents the discrete measurement equations, $h(x_k)$ from equation 98, which describe the relationship between each of the measurements and the state variables.

Table 5: Measured variables from the SAF model.

	Measurement	**Symbol**	**State relationship**	**Standard deviation of the measurement noise**	**Unit**
y_1	Bulk concentrate temperature	$T_{C(B)}$	$\boldsymbol{T_{C(B)}}$	10	K
y_2	Slag zone height	L_S	$\frac{M_{XO}\boldsymbol{N_S}}{\rho_S A_{SAF}}$	0.05	m
y_3	Slag zone temperature	T_S	$\boldsymbol{T_S}$	19	K

y_4	Matte zone height	L_M	$\frac{M_{XS}\boldsymbol{N_M}}{\rho_M A_{SAF}}$	0.05	m
y_5	Matte zone temperature	T_M	$\boldsymbol{T_M}$	18	K
y_6	Reaction gas concentration in freeboard	$C_{G,R}$	$\frac{\boldsymbol{N_{G,R}}}{V_G}$	0.1	$\frac{mol}{m^3}$
y_7	Freeboard pressure	P_G	$\frac{(\boldsymbol{N_{G,A}} + \boldsymbol{N_{G,R}})R\boldsymbol{T_G}}{V_G}$	2	Pa
y_8	Freeboard temperature	T_G	$\boldsymbol{T_G}$	2.2	K
y_9	Cooling water temperature	T_W	$\boldsymbol{T_W}$	1	K

Table 5 shows that of these 9 measurements, 5 are state variables themselves. The remaining 4 measurements are functions of the state variables and constant parameters from the process model M_{XO}, M_{XS}, ρ_S, ρ_M, A_{SAF}, and V_G. The values for these constant parameters are presented in Table C.3.

The measurements are sampled at specific points in the simulation time to simulate realistic measurement frequency. Each of the measurements have equivalent sampling rates of 10 seconds, corresponding to the sampling rate of the sensors at Anglo Platinum's Polokwane smelter (Groenewald et al., 2018).

The magnitude of the standard deviation of the measurement noise is estimated based on sensor accuracy information or by calculating the standard deviation on past measurements. Based on real-world SAF instrumentation information obtained via confidential communication with industrial partners, realistic measurement noise was chosen. Table 5 presents the standard deviation associated with each of the measurements. Table C.4 presents these standard deviation values along with justification for the choice of these values.

3.4 SAF faults

A fault is an unobserved occurrence that manifests within or outside of a process causing abnormal behaviour in system variables and the dynamic relationships between variables (Patton et al., 2000). Faults can manifest from a variety of sources including mechanical or electrical failure of plant equipment such as sensors and actuators, or abnormal disturbances causing changes to the process conditions. The SAF is subject to numerous potential faults. The four faults that are investigated in this study are a fault in the flowrate of the cooling water, an extraction pressure fault, a change in the charged concentrate composition, and furnace blowback.

3.4.1 Fault modelling

3.4.1.1 Cooling water fault

The flowrate of cooling water is a constant input to the process. The cooling water fault is simulated as a sudden decrease in this input. Sidewall cooling is important in the operation of a SAF to maintain the integrity of the furnace lining (Jones, 2005). This abnormal disturbance in the flowrate of cooling water reflects within the measurement of the temperature of the cooling water, T_W, disrupting the process of cooling the sidewalls and can potentially cause structural damage to the refractory shell of the furnace.

3.4.1.2 Extraction pressure fault

The desulphurization and oxidation reactions occurring in the smelting process release sulphur, sulphur dioxide, hydrogen sulphide, and hydrogen chloride. These gases are contained in the freeboard of the SAF (Jones, 2005). The gauge pressure within the furnace freeboard is maintained at a slightly negative pressure to prevent reaction gases from escaping to the atmosphere (Thethwayo, 2010). This is achieved by extracting the reaction gases from the furnace and replacing the gases with air. Controlling the off-gas removal is necessary in smelting processes (Crundwell et al., 2011). This process is controlled by forced draught or forcing air flow inside the furnace via fans. The molar flowrate of the reaction gases being extracted from the furnace is calculated as:

$$\boldsymbol{F_{ext,R}} = \frac{k_{PE} N_{G,R} (\boldsymbol{P_G} - P_{ext})}{N_{G,A} + N_{G,R}} \qquad [\,99\,]$$

Where P_{ext} is a constant parameter of the furnace model, representing the forced draught. The parameter P_{ext} dictates the flowrate of reaction gases and air being extracted from the furnace due to the pressure difference created between P_{ext} and the pressure in the freeboard, P_G. The extraction pressure fault is initiated via a sudden decrease in the extraction draught. In practice, a failure in the forced draught mechanism causes this type of fault. A sudden decrease in the extraction draught results in a smaller pressure difference, thus, causing an increase in the amount of reaction gases within the freeboard as less reaction gases are being extracted. This disrupts the pressure conditions within the furnace, disrupting the integrity of the smelting process and can potentially lead to excess release of reaction gases to the atmosphere.

3.4.1.3 Charge composition fault

The charge composition fault is simulated by a step change in the composition of the charging concentrate feed to the furnace. There is a decrease in the slag component composition in the charging concentrate and an increase in the matte component composition. This causes an increase in the number of moles of matte component and decrease in the number of moles of slag component within the furnace. The furnace operates at set temperature intensities to ensure optimal operating conditions for the specific composition of ore being processed. A decrease in the slag component favours a less energy intensive process as chromite formation is not as prevalent (Nell, 2004). Thus, an unmonitored change in the ore composition disrupts the process conditions, resulting in sub-optimal performance of the smelting process as energy is being wasted.

3.4.1.4 Furnace blowback

When the freeboard pressure becomes greater than the atmospheric pressure outside the furnace, blowback occurs. A blowback, or furnace eruption, is the sudden release of the gases from the freeboard into the surrounding atmosphere. The hazardous gases can be harmful to nearby operators and the sudden release of gas and subsequent temperature drop can disrupt the process conditions within the furnace (Theunissen, 2021).

The reaction gases that are generated from desulphurization and oxidation accumulate within the voids of the concentrate layer. Under nominal operating conditions, these reaction gases flux to the freeboard due to the pressure difference between the concentrate and the freeboard. Furnace blowback occurs when there is a buildup of pressure in the concentrate voids caused by excess accumulation of reaction gases within the concentrate. When the pressure in the concentrate exceeds a critical pressure, reaction gases from the concentrate rapidly channel into the freeboard and cause an increase in the freeboard pressure.

This pressure buildup in the concentrate is caused by an increase in the number of moles of reaction gases in the concentrate voids, $N_{C(R)}$. Thus, an increase in $N_{C(R)}$ causes furnace blowbacks. Based on the dynamic equation of $N_{C(R)}$, in this study, furnace blowback is simulated via a sudden change in the parameters $h_{S:C(S)}$ and k_{PBR}.

A sudden decrease in the heat transfer coefficient, $h_{S:C(S)}$, causes a decrease in F_{melt}, the flowrate of melted concentrate into the slag phase. The concentrate charging rate remains constant, however, less concentrate melts into the slag, causing a buildup of smelting concentrate, $N_{C(S)}$. More smelting concentrate results in more reaction gases, $N_{C(R)}$.

k_{PBR} is the flux coefficient which dictates the reaction gas flux from the concentrate to the freeboard. A sudden decrease in the parameter k_{PBR} causes slower movement of reaction gases to the freeboard, and thus, a build-up of reaction gases in the voids within the concentrate.

Both these changes result in blowback-preceding conditions and eventual furnace blowback once $N_{C(R)}$ exceeds a certain level.

The original model presented by Theunissen (2021) modelled concentrate charging on a switch basis based on limits placed on the concentrate bed height and modelled furnace blowback by suddenly increasing the concentrate bed height limits, allowing for more ore to be charged to the furnace and thus more gases to build up.

The new method of modelling furnace blowback induces concentrate buildup via a sudden decrease in $h_{S:C(S)}$. When bed height limits are used to initiate concentrate charging, the concentrate will not build up as a result of a decrease in $h_{S:C(S)}$. Thus, the process model of the SAF developed by Theunissen (2021) is altered to facilitate this modelling of the furnace blowback. The model used in this study models the concentrate charging based a timed switch as opposed to a switch based on the bed height limits.

The values for each of the parameters under nominal operating conditions and altered faulty conditions are summarized in Table 6.

Table 6: Nominal operating conditions and faulty conditions for fault simulation.

	Fault	Nominal operation	Faulty operation
1	Cooling water fault	$F_W = 2400 \frac{mol}{s}$	$F_W = 1900 \frac{mol}{s}$
2	Extraction pressure fault	$P_{ext,gauge} = -10\ Pa$	$P_{ext,gauge} = -9\ Pa$
3	Composition switch	$x_{charge,XO} = 0.90$ $x_{charge,XS} = 0.09$ $x_{charge,XS_2} = 0.01$	$x_{charge,XO} = 0.78$ $x_{charge,XS} = 0.21$ $x_{charge,XS_2} = 0.01$
4	Furnace blowback	$h_{S:C(S)} = 0.31 \frac{kW}{m^2.K}$ $k_{PBR} = 1 \times 10^{-8} \frac{mol}{m.Pa.s}$	$h_{S:C(S)} = 0.248 \frac{kW}{m^2.K}$ $k_{PBR} = 0.7 \times 10^{-8} \frac{mol}{m.Pa.s}$

3.4.2 Fault categorization

Each of the four faults simulated in this study are categorized as additive or multiplicative faults by analyzing how the faults, presented in Table 6, reflect within the state dynamics of the system. The ODEs, presented in Table 4, are used to ascertain the dynamic relationship between the faults and the state variables. The results are presented in Table 7.

Table 7: Relationship between fault parameters and state ODEs and subsequent fault categorization.

	Fault	Dynamic equation	Categorization
1	Cooling water fault	1. $\frac{dT_W}{dt} = \frac{c_{P,W}(T_{W,0}-T_W)F_W - Q_{W:M} - Q_{W:S}}{N_W c_{P,W}}$	Multiplicative in the state equation
2	Extraction pressure fault	1. $\frac{dN_{G,A}}{dt} = F_{neg.P} - F_{pos.P,A} - \boldsymbol{F_{ext,A}}$ $F_{ext,A} = \frac{k_{PE}N_{G,A}(\frac{(N_{G,A}+N_{G,R})RT_G}{V_G} - P_{ext})}{N_{G,A}+N_{G,R}}$ 2. $\frac{dN_{G,R}}{dt} = J_R A - F_{pos.P,R} - \boldsymbol{F_{ext,R}}$ $F_{ext,R} = \frac{k_{PE}N_{G,R}(\frac{(N_{G,A}+N_{G,R})RT_G}{V_G} - P_{ext})}{N_{G,A}+N_{G,R}}$	Multiplicative in the state equation
3	Composition switch	1. $\frac{dN_{C(B),XO}}{dt} = F_{charge}x_{charge,XO} - F_{mix,XO}$ 2. $\frac{dN_{C(B),XS}}{dt} = F_{charge}x_{charge,XS} - F_{mix,XS} + r_{F,C(B)}V_{C(B)}$	Multiplicative in the input equation
4	Furnace blowback	1. $\frac{dN_{C(S),XO}}{dt} = \boldsymbol{F_{mix,XO}} - F_{melt,XO}$ $\boldsymbol{F_{melt,XO}} = \frac{h_{S:C(S)}A_{SAF}(T_S - T_{C(S)})\frac{T_{C(S)}}{T_{C,melt}}}{\lambda_C + c_{P,C}(T_{C,melt} - T_{C(S)})} \frac{N_{C(S),XO}}{N_{C(S),XO}+N_{C(S),XS}}$	Multiplicative in the state equation

	2. $\frac{dN_{G,R}}{dt} = \boldsymbol{J_R}A - F_{pos.P,R} - F_{ext,R}$ $\boldsymbol{J_R} = \frac{k_{PBR}}{\frac{V_{C(B)}+V_{C(S)}}{A_{SAF}}}\left(\frac{N_{C(R)}RT_{C(S)}}{\varepsilon_C(V_{C(B)}+V_{C(S)})} - \frac{(N_{G,A}+N_{G,R})RT_G}{V_G}\right)$	

All four faults investigated in this study are multiplicative faults. Faults 1, 2 and 4 are multiplicative in the state equations, whereby the magnitude of the effect of the fault parameters is dependent on the values of the states themselves. The magnitude of the effect of fault 3 does not depend on the value of the states, however, it does depend on the value of the input, F_{charge}.

4 OBSERVABILITY AND DETECTABILITY ANALYSIS APPROACH AND RESULTS

The following chapter presents the methodology and results for the observability analysis of the SAF system and fault detectability analysis to address objective 1 of the study. Section 4.1 outlines the procedure used to conduct the state observability analysis and the fault detectability analysis via structural detectability and performance-based detectability. Section 4.2 presents the results of the state observability analysis to ascertain system observability and the degree of observability of the state variables. Section 4.3 provides the results for the fault detectability, addressing both structural detectability via parameter observability and the performance-based detectability using the SNR.

4.1 Observability analysis approach

4.1.1 Observability matrix of the SAF model

The process model and measurement equations which define the SAF system were presented in Chapter 3 in sections 3.2 and 3.3. The method of constructing the observability matrix for a nonlinear system by linearization is presented in sub-section 2.2.2.3. For this study, the linearized observability matrix is constructed as opposed to the nonlinear observability matrix using Lie derivatives due to the impractical computational expense incurred when calculating higher order Lie derivatives. Furthermore, section 2.2.2.4 explains that local observability via linearization is a sufficient condition for observability. Under the circumstances that the system is found to be locally unobservable via the linearization method, it should then be checked for local weak observability using the nonlinear observability matrix constructed from the Lie derivatives.

The SAF nonlinear system of equations is linearized around state values chosen at an arbitrary point representing NOCs, x_{nom}.The linearization is achieved via the symbolic math toolbox in MATLAB to calculate the Jacobian matrices $A = \frac{\partial f}{\partial x}\Big|_{x_{nom}}$ and $C = \frac{\partial h}{\partial x}\Big|_{x_{nom}}$.

The linearized system of equations is then scaled using the range of NOCs as part of the data pre-processing described in sub-section 2.2.3.2. The linearized and scaled state matrix is given by $A_{scaled} = S_x A S_x^{-1}$ and the linearized and scaled measurement matrix is $C_{scaled} = S_y C S_x^{-1}$. The derivation of the scaled matrices is presented in appendix A.10.

This SAF system model is as a hybrid system of equations, whereby the dynamics of the system model are continuous, and the measurement model is discrete. As explained in sub-section 2.2.3, the discrete system model is used to analyze the degree of observability. The scaled continuous-time state matrix, A_{scaled}, is converted to the discrete-time state matrix, F, by:

$$F = e^{A_{scaled}\Delta t} \qquad [\ 100\ ]$$

Where Δt is the discretization step size.

The linearized observability matrix for the SAF model is generated using the linearized and scaled measurement matrix, C_{scaled}, and the discretized, linearized, and scaled state transition matrix, F. This is done in MATLAB using the function *obsv*.

$$O_{SAF} = \begin{bmatrix} C_{scaled} \\ C_{scaled}F \\ \vdots \\ C_{scaled}F^{n-1} \end{bmatrix} \quad [\ 101\ ]$$

The SAF linearized observability matrix, O_{SAF}, is an $m \times n$ matrix. The number of rows is equal to the number of state variables multiplied by the number of measurement variables $m = n_y \times n$. The number of columns is equal to the number of state variables, n. The SAF system has $n = 17$ state variables and $n_y = 9$ measured variables.

4.1.2 SVD of the observability matrix

To assess the observability of the SAF system, SVD is performed on the observability matrix. The theory of SVD is presented in sub-section 2.2.3. The observability matrix is decomposed via SVD into the matrix of left singular vectors, U, the matrix of right singular vectors, V, and the diagonal matrix containing the singular values, Σ.

$$O_{SAF} = U\Sigma V^T \quad [\ 102\ ]$$

4.1.2.1 Rank test

The rank of the observability matrix is assessed following the theory presented in sub-section 2.2.3.3. The rank of the observability matrix is indicated by the number of non-zero singular values in Σ. An observable system has a full rank observability matrix. Therefore, the number of non-zero singular values should be equal to 17 for the SAF system to be considered observable.

4.1.2.2 Analysis of V

According to the theory presented in sub-section 2.2.3.4, the states associated with each direction of observability can be assessed by analysing the elements of the right singular vectors of V. For the SAF system, V is a 17×17 matrix composed of 17 right singular vectors, v_1 through v_{17}.

$$V = \begin{bmatrix} | & | & | \\ v_1 & \dots & v_{17} \\ | & | & | \end{bmatrix}$$

Each right singular vector represents an observable direction. The entries in each right singular vector, v_i, indicate the corresponding states contribution to the i^{th} observable direction.

4.1.2.3 Generating an observability index

The observability of the SAF system can be assessed according to the rank test outlined in sub-section 4.1.2.1. In addition, the states that contribute to the various observable directions can be analyzed following the procedure outlined in sub-section 4.1.2.2. Both these analyses do not summarize the degree of observability of the individual states. Therefore, a single metric, which will be referred to as the observability index of a state, is defined for this study. This observability index is used as a qualitative metric to investigate the level of observability of the individual state variables for this study.

The entry V_{ji} represents the j^{th} states' contribution to the i^{th} direction of observability. For each state, j, an observability metric, o_j, can be calculated as the sum of the j^{th} row of V, where each entry in the row is weighted according to a weighting W_i.

$$o_j = \sum_{i=1}^{i=n} W_i V_{ji} \qquad j = 1, \dots, n \qquad [\ 103\ ]$$

The weightings, W_i, are calculated using the singular value, σ_i, corresponding to the observable direction i. The singular values are normalized to ensure the sum of all the weightings is equal to 1. The smallest weighting will be closest to zero and corresponds to the smallest singular value associated with the least observable direction. The largest weighting will be closest to 1 and will be associated with the largest singular value and thus most observable direction.

$$W_i = \frac{\sigma_i}{\sum_1^n \sigma_i} \qquad i = 1, \dots, n \qquad [\ 104\ ]$$

The largest o_j is associated with the most observable state and the smallest o_j with the least observable state.

4.1.3 Observability-identifiability matrix of the SAF

Based on the categorization of faults in this study, as presented in Section 3.4.2, all four faults investigated are categorized as multiplicative faults. Faults 1, 2, and 4 are multiplicative in the states and structural fault detectability is assessed via parameter observability. Fault 3 is multiplicative in the input and structural fault detectability is assessed via input observability. As described in section 2.3.7.2, structural fault detectability is assessed by parameter and input observability by constructing the observability-identifiability matrix. For this study, as described in section 3.2, the inputs F_W and x_{conc} are constant variables, therefore, time-varying inputs do not need to be accounted for in the observability-identifiability analysis.

The state vector was augmented to include all 17 state variables and the 5 fault parameters and inputs, presented in Table 6, as additional unknown states.

$$\tilde{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_{17} \\ h_{S:C(S)} \\ k_{PBR} \\ F_W \\ P_{ext} \\ x_{conc} \end{bmatrix}$$

An observability analysis is carried out on this augmented state system. The same procedure is followed as presented in section 4.1.1. The linearized observability-identifiability matrix for the SAF model is generated using the linearized and scaled measurement matrix, C_{scaled}, and the discretized, linearized, and scaled state transition matrix, F. Where the additional states, the 5 fault parameters and inputs, have zero dynamics in F.

$$\dot{\tilde{x}} = \begin{bmatrix} \dot{x} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} F(x, \theta) \\ 0 \end{bmatrix}$$

$$O_{I,SAF} = \begin{bmatrix} C_{scaled} \\ C_{scaled}F \\ \vdots \\ C_{scaled}F^{n-1} \end{bmatrix} \quad [\ 105\ ]$$

The procedures followed in sub-sections 4.1.2.2 and 4.1.2.3 will be repeated for the observability-identifiability matrix to evaluate each of the fault parameters level of observability. This level of observability corresponds to the level of structural detectability of the fault parameters.

4.2 State observability results

The linearized observability matrix is decomposed via SVD. The singular values obtained from SVD of the observability matrix are plotted on a logarithmic plot in Figure 7.

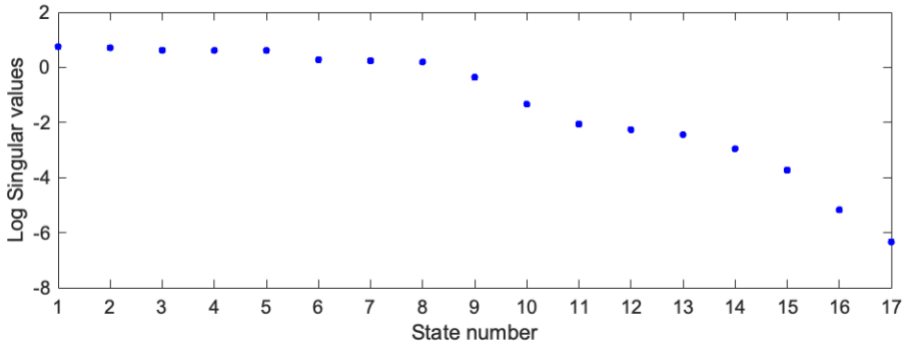

Figure 7: Logarithmic plot of singular values from SVD of the linearized observability matrix.

The results show that there exist 17 non-zero singular values. Therefore, the system is considered fully observable as the number of non-zero singular values is equal to the number of states in the system. It should be noted that the last few singular values are particularly small, indicating poor observability of the last few observable directions.

When the singular values are plotted on a logarithmic scale, system unobservability is visualized by a sudden gap in the plot when the singular values differ by orders of magnitude (Stigter et al., 2017). Figure 7 shows relatively similar magnitudes for the first 8 singular values, $\sigma_1 - \sigma_8$. There is a sudden decrease in the magnitude at σ_9 and the magnitude of the singular values steadily decrease thereafter. Although there is a small gap between σ_8 and σ_9 and the subsequent singular values, Figure 7 does not display the characteristic large gap indicating system unobservability. Therefore, although the last few singular values are close to zero, the system is classified as observable.

These results are further corroborated by analysing the groups of correlated states associated with the directions of observability with small singular values. This is done by analysis of the right singular vectors. Figure 8 represents a heatmap of the right singular vectors obtained from SVD of the linearized observability matrix.

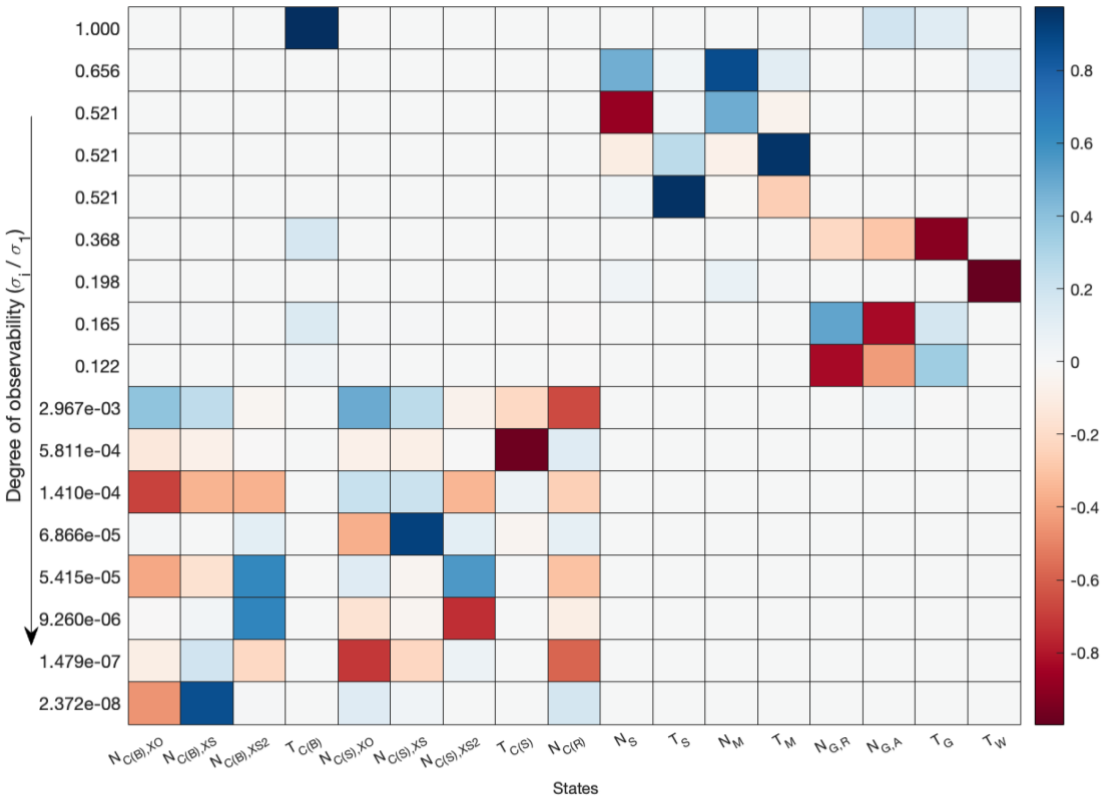

Figure 8: Heat map of right singular vectors of the linearized observability matrix.

A heatmap of the right singular vectors, v_1 through v_{17}, is used to visualize the states correlated with the various directions of observability. The rows of the heatmap represent the 17 observable directions, with the first row being the most observable direction and the last row being the least observable direction. The columns of the heatmap represent the 17 state variables. Each block ij in the heatmap is coloured according to the magnitude of the correlation of the state/column j to the observable direction/row i.

From Figure 8, it can be seen that state variables $N_{C(B),XO}$, the number of moles of slag component in the bulk concentrate, and $N_{C(B),XS}$, the number of moles of matte component in the bulk concentrate, are strongly correlated with the smallest singular value and the least observable direction. From Figure 8, states $N_{C(S),XO}$, the number of moles of slag component in the smelting concentrate, and $N_{C(R)}$, the number of moles of reaction gases trapped in the concentrate, are strongly associated with the second smallest singular value. Therefore, the states $N_{C(B),XO}$, $N_{C(B),XS}$, $N_{C(S),XO}$, and $N_{C(R)}$ are the states that cause the system to exist close to unobservability. However, Figure 8 indicates these states are also associated with other, more observable, directions. This further confirms the system is indeed observable.

From Figure 8, the states contributing to the most observable direction can also be inferred. It is expected that states that are directly or indirectly measured should be associated with the most observable directions. State $T_{C(B)}$, the temperature of the bulk concentrate, is associated with the most observable direction, and states N_S, the number of moles of slag in the slag zone, and N_M, the number of moles of matte in the matte zone, are associated with the second most observable direction. From the

measurement equations presented in Table 5, it can be seen that these states are directly or indirectly measured. The other measured states, T_S, T_M,$N_{G,R}$, $N_{G,A}$ and T_G are all associated with more observable directions. State T_W, the temperature of the cooling water, is one of the less observable measured variables. This is due to the fact that although T_W is directly measured, the state is not involved in dynamic equations of the other states. The observability matrix considers both the measurement matrix, but also the state transition matrix which explains dynamic relationships amongst the state variables. More observable measured states, such as N_S and N_M, are measured and involved in a number of dynamic equations with other measured variables, thus, making them more observable.

The observability index described in section 4.1.2.3 is calculated for each of the 17 state variables from the SVD of the linearized observability. The results are presented in Table 8, with the states ordered from most observable to least observable. Additionally, the ratio of the observability index of the current state to previous state in the table is calculated.

Table 8: Observability index for each of the state variables obtained from the linearized observability matrix.

Ranking (i)	**State**	o_i	**Ratio $\frac{o_i}{o_{i-1}}$**	
1	$T_{C(B)}$	0.2608	-	Measured states
2	N_M	0.2176	0.8342	
3	N_S	0.2059	0.9464	
4	T_S	0.1813	0.8803	
5	T_M	0.1646	0.9080	
6	T_G	0.1279	0.7772	
7	$N_{G,A}$	0.1186	0.9275	
8	$N_{G,R}$	0.0664	0.5594	
9	T_W	0.0610	0.9195	
10	$N_{C(R)}$	0.0024	**0.0393**	Unmeasured states
11	$N_{C(S),XO}$	0.0018	0.7321	
12	$N_{C(B),XO}$	0.0014	0.8033	
13	$T_{C(S)}$	0.0012	0.8264	
14	$N_{C(S),XS}$	0.0010	0.8449	
15	$N_{C(B),XS}$	0.0009	0.9220	
16	$N_{C(S),XS_2}$	0.0002	**0.2315**	
17	$N_{C(B),XS_2}$	0.0001	**0.4728**	

The results presented in Table 8 provide a summary of the results obtained in section **Error! Reference source not found.** by analysis of Figure 8. This confirms that the measured state variables are more observable than the unmeasured states. The sudden jump between the eighth and ninth singular values seen in Figure 7 is also reflected in Table 8. The ratio, $\frac{o_i}{o_{i-1}}$, suddenly drops at the tenth ranking. This sudden drop in the ratio is caused by a sudden significant decrease in the observability index between the measured states and the unmeasured states. Table 8 also highlights the significant drop in the ratio at the 16th ranking. This indicates that the level of observability significantly decreases for states

$N_{C(B),XS_2}$ and $N_{C(S),XS_2}$, the sulphurized matte compoents in the bulk concentrate and the smelting concentrate, respectively.

4.3 Fault detectability results

4.3.1 Structural fault detectability

Figure 9 shows the logarithmic plot of the singular values obtained from SVD of the observability-identifiability matrix. Figure 9 displays the characteristic large gap in the singular values between the 21st and 22nd singular values indicating that the observability-identifiability matrix is rank deficient.

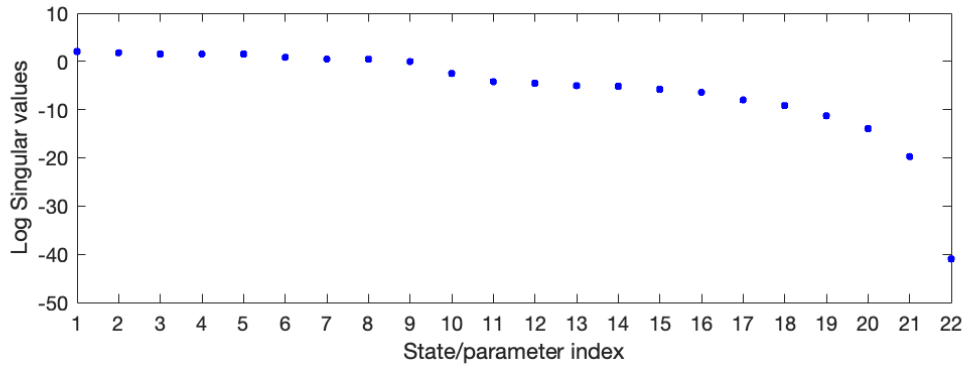

Figure 9: Logarithmic plot of singular values from SVD of the observability-identifiability matrix.

Figure 10 shows the heatmap of the right singular vectors obtained from SVD of the observability-identifiability matrix. For simplicity, the figure excludes the contributions of the state variables x_1 through x_{17} and only shows the fault parameter contributions to the observable directions.

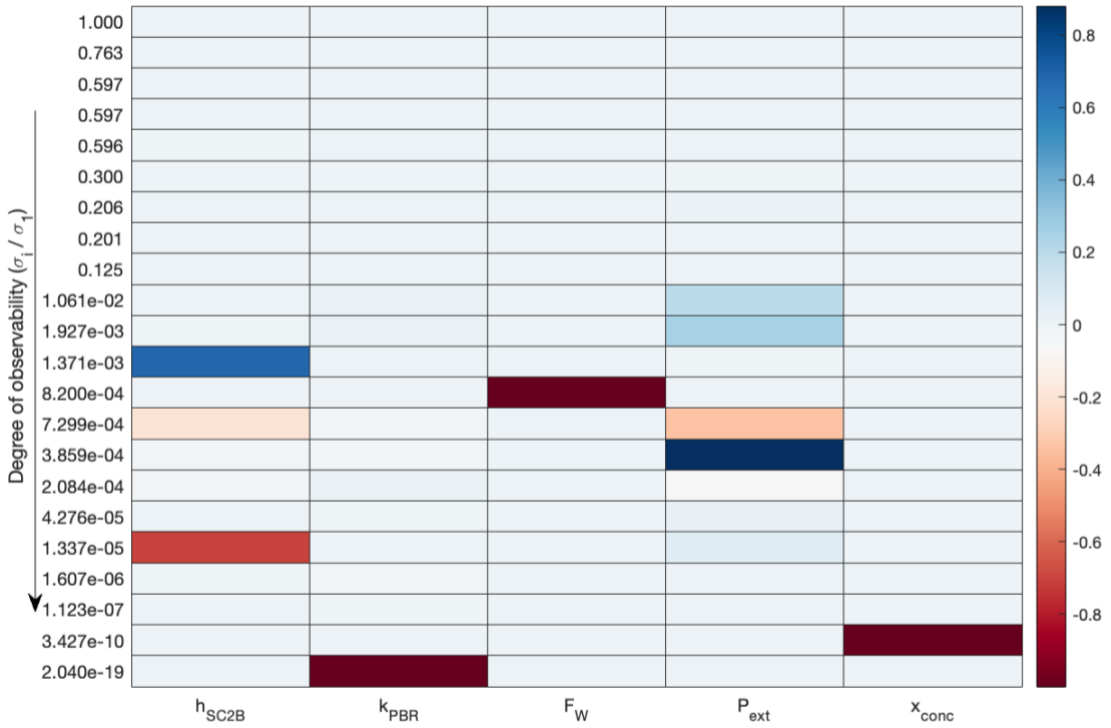

Figure 10: Heatmap of the right singular vectors of the observability-identifiability matrix.

Figure 10 shows that parameter P_{ext} contributes to the tenth, eleventh, fourteenth and fifteenth observable directions. Fault parameter $h_{S:C(S)}$ is associated with the twelfth, fourteenth, and eighteenth observable directions. Fault F_W is associated with the thirteenth observable direction. Fault x_{conc} is associated with the second to least observable direction. Lastly, fault parameter k_{PBR} is associated with the least observable direction. From Figure 9, the last singular vector can be assumed to have a value of zero due to the gap in the singular values, thus, indicating system unidentifiability. Upon analysis of the null space basis vector, the parameter involved in a total correlation causing system unidentifiability is only parameter k_{PBR}.

The results in Figure 10 are supported by calculating the observability index for each of the fault parameters, presented in

Table 9.

Table 9: Fault parameter scaled observability indices.

Ranking (i)	**State**	o_i	**Ratio** $\frac{o_i}{o_{i-1}}$
1	x_{10}	0.2412	
2	x_4	0.2382	0.9873
3	x_{12}	0.2276	0.9557
4	x_{13}	0.2144	0.9422
5	x_{11}	0.1842	0.8589
6	x_{15}	0.1158	0.6288
7	x_{16}	0.0968	0.8356
8	x_{14}	0.0680	0.7027
9	x_{17}	0.0556	0.8183
10	x_3	0.0058	**0.1035**
11	x_9	0.0044	0.7640
12	x_5	0.0032	0.7284
13	P_{ext}	0.0024	0.7484
14	x_1	0.0023	0.9723
15	x_6	0.0017	0.7261
16	x_8	0.0015	0.8985
17	x_2	0.0015	0.9956
18	$h_{S:C(S)}$	0.0007	0.4847
19	x_7	0.0006	0.7955
20	F_W	0.0004	0.6625
21	x_{conc}	0.0002	0.5565
22	k_{PBR}	4.34E-07	**0.0020**

The characteristic gap seen in Figure 9 is reflected as a small ratio, $\frac{o_i}{o_{i-1}}$, at the 22nd ranking in Table 9 indicating that the observability index suddenly decreased. The fault parameter k_{PBR} is unidentifiable and therefore structurally undetectable. However, due to the identifiability of parameter $h_{S:C(S)}$, the blowback fault is still considered structurally detectable.

According to the observability analysis, it can be concluded that the P_{ext} fault is the most structurally detectable. The blowback fault is the second most structurally detectable due to the change in parameter $h_{S:C(S)}$. The fault in F_W is the third most structurally detectable. Lastly, the least structurally detectable fault is the fault in x_{conc}.

4.3.2 Performance-based fault detectability

A fault SNR is calculated for each observation for each of the four faults investigated using equation 96 presented in sub-section 2.3.7.5. In data-driven fault detection, the observation is the measurement from the process. In model-based fault detection, the observation is the residual generated by the EKF or the PF. The SNR for each observation for each of the four faults is presented in Figure 11. From these results, the performance-based detectability for each fault detection method for each respective fault can be ascertained.

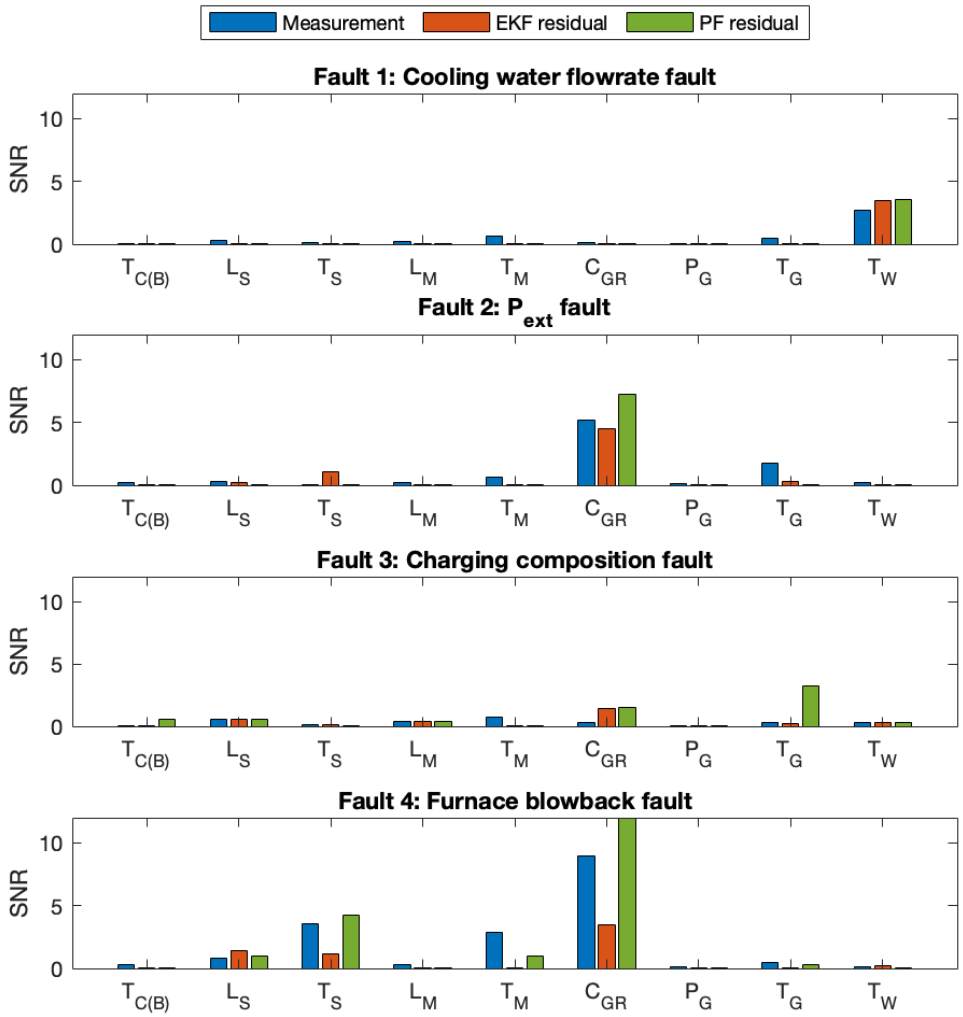

Figure 11: Fault SNR for each residual/measurement for each of the faults.

The faults are first compared to one another. Figure 11 shows that blowback fault has the highest SNR in the C_{GR} observation. The blowback fault also reflects as large signals within a number of different observations including L_s, T_s, and T_M. The extraction pressure fault has the second largest SNR in the C_{GR}

observation. The cooling water flowrate fault shows a smaller SNR and the fault mainly reflects within the T_w observation. The charging composition fault reflects in a number of observations, each with relatively small SNRs.

From this analysis, the blowback fault is the most detectable, followed by the extraction pressure fault and the cooling water fault. The charge composition fault is the least detectable based on the performance-based detectability. These results are similar to those obtained by the structural detectability analysis in section 4.3.1. However, structural detectability indicates the extraction pressure fault is the most structurally detectable and the blowback fault the second most.

From Figure 11, the fault detection methods can also be compared. The PF residual shows consistently good performance-based detectability for all four faults with consistently higher SNRs than the EKF and measurement observations. The EKF residual shows comparable detectability to the PF residual for the cooling water fault and better performance than the measurement for the charge composition fault. The measurement used for data-driven fault detection shows superior performance-based fault detectability than the EKF residual for the extraction pressure fault and the blowback fault.

This performance-based assessment of the fault detectability is dependent on the type and size of the fault, as well as the specific method of obtaining the faulty signal, either via the measurements or residuals from state estimators. This method also considers the effect of measurement noise on the detectability of the faults within the observations. It should be noted that in real-world applications there may exist unknown fault modes or disturbances that manifest within the measurements. Furthermore, labelled historical data may not be available for each individual fault mode. Therefore, the SNR cannot always be calculated for each individual fault mode in practice.

5 STATE ESTIMATION APPROACH AND RESULTS

The following chapter outlines the approach and results in fulfillment of objective 2 of the study: implement the EKF, UKF, PF, and MHE for estimation of the states in the SAF smelting process to enable selection of one or more state estimation methods to be used in model-based fault detection. Section 5.1 outlines the methodology for implementation of the four state estimation techniques. Section 5.2 presents the results obtained from using the state estimation algorithms to estimate the states of the SAF. This section highlights and compares the performance of each of the state estimation methods investigated, allowing for selection of the most appropriate technique/s for application in model-based fault detection.

5.1 State estimation approach

The following section describes the approach used to meet the second objective of this study. Sub-section 5.1.1 describes the numerical implementation of the state estimation techniques in MATLAB. Sub-section 5.1.2 outlines the metrics that are used to evaluate the performance of the state estimation techniques. Sub-section 5.1.3, 5.1.4, and 5.1.5 derive appropriate values for the tuning parameters required for implementation of the state estimation algorithms for this study.

5.1.1 State estimation algorithms

The four state estimation algorithms investigated in this study, the EKF, UKF, PF, and MHE, are employed for estimation of the 17 state variables of the SAF system, presented in Table 3. From section 3.3, a new measurement is obtained from the SAF process every $10\ s$. Therefore, a new state estimate for each of the state variables is calculated every $10\ s$.

The EKF algorithm explained in sub-section 2.1.3.2 and presented in appendix B.1, the UKF algorithm detailed in sub-section 2.1.4.5 and presented in appendix B.2, and the standard bootstrap PF algorithm in sub-section 2.1.5.4 and appendix B.3, are implemented in MATLAB using the code presented in

Appendix D – MATLAB code. The MHE is implemented in MATLAB using the nonlinear programming solver *fminunc* with the optimization algorithm *sqp* to solve for a horizon of state estimates based on the objective function presented in sub-section 2.1.6.

The prediction step of the algorithms involves propagating the state estimate from the previous timestep to the next timestep using the nonlinear model representing the dynamics of the system. For this study, the nonlinear dynamics are represented by the model equations presented in Table 4. The solution for the *a priori* state estimate is calculated in MATLAB by propagating the previous state estimate 10 seconds ahead to the next timestep using *ode15s.*

The update step of the algorithms involves updating the *a priori* state estimate with the current measurement. This involves transforming the *a priori* state estimate through the measurement equations presented in Table 5.

Each algorithm requires tuning parameters that are defined prior to implementation. The common tuning parameters required for each algorithm are the initial state estimate, initial state estimate error covariance, measurement noise covariance matrix, and process noise covariance matrix. An additional parameter is required for the PF, the number of particles. Additional parameters for the MHE are the horizon length, as well as any constraints on the state variables. Placing constraints on the state estimates is important when the system is operating close to physically unrealizable values of the states. For the SAF model, the 17 state variables being estimated are molar amounts and temperatures. Nonnegative molar amounts would be an example of an appropriate physically motivated constraint. However, the range of NOCs for of the SAF do not lie close to this physical constraint. Furthermore, upon implementation of the unconstrained nonlinear filters, the EKF, UKF, and PF, the state estimates do not converge to any physically unrealizable solutions. Therefore, for this study the constraints are omitted from the MHE optimization algorithm.

5.1.2 Performance evaluation

Based on the theory presented in sub-section 2.1.9, for this study the metric used to quantify the estimation accuracy of each state estimator is the MAPE, calculated as:

$$MAPE(\%) = \frac{1}{N}\sum_{k=1}^{N}\frac{|x_k - \hat{x}_k|}{x_k} \times 100 \qquad [\ 106\ ]$$

The state estimates, $\hat{x}_k$, are obtained from each state estimation algorithm. The ground truth values, x_k, are the underlying true values for the states obtained during the generation of synthetic measurement data.

The MAPE provides a scale-free and more interpretable value for the estimation accuracy. However, it should be noted that the objective function of the state estimators (the squared error) does not match the performance metric (the MAPE). Moreover, the MAPE calculated when the true value of the state, x_k, goes to zero is a meaningless value and should be ignored. This is not relevant for this study as the furnace does not operate under conditions resulting in a value of zero for any of the state variables.

The computational burden of the filters is quantified by the average elapsed time taken to obtain each state estimate. This is obtained directly from MATLAB using the *tic* and *toc* functions. Evidently, the computational times are specific to this study and the hardware used. The hardware used in this study is Apple MacBook Pro M1 8 GB.

5.1.3 Tuning parameters

5.1.3.1 Filter initialization

As per the theory described in sub-sections 2.1.7.1 and 2.1.7.2, each filter is initialized with a guess for the initial state estimate and the initial estimation error covariance matrix. The most important consideration is to ensure that the initial state estimate and estimation error covariance are consistent with one another.

For this study, the ground truth initial conditions, x_0, of the system are exactly known and presented in Table C.2. Therefore, the initial state estimate tuning parameter, $\hat{x}_0$, is based on this ground truth value. Realistic initialization error is simulated by adding some error to the initial ground truth values of the states.

$$\hat{x}_0 = (1.01)x_0 \qquad [\ 107\ ]$$

Where the initial state estimates differ from the ground truth initial conditions by 1% of the initial ground truth values.

The initial estimation error covariance matrix is selected to be consistent with the initial state estimate using the equation:

$$P_0 = E[(x_0 - \hat{x}_0)(x_0 - \hat{x}_0)^T] \qquad [\ 108\ ]$$

Where the initial estimation error covariance is a diagonal matrix with elements on the diagonal equal to the variance of the initial state estimate error.

As explained in sub-section 2.1.6.2, the MHE optimization routine requires an initial guess for the state estimates over the entire initial horizon length. This is often achieved via another nonlinear state estimator. For this study, the EKF is used to supply the MHE optimization routine with the guesses for the initial horizon of state estimates. This EKF is initialized with $\hat{x}_0$ and P_0 presented above.

5.1.3.2 Measurement noise covariance matrix

The measurement noise covariance matrix supplied to the state estimators in this study is calculated from the standard deviation of the measurements as described in sub-section 2.1.7.3.

$$R = diag(\sigma_{meas}^2) \qquad [\ 109\ ]$$

Where σ_{meas} is the standard deviation used to generate synthetic measurement data found in Table 5. As explained in sub-section 2.1.7.3, in practice the measurement noise is fairly easy to obtain from sensor information or data. Thus, for this study the measurement noise covariance supplied to the state

estimator is assumed exactly equal to the measurement noise covariance on the synthetically generated measurements.

5.1.3.3 Process noise covariance matrix

For this study, the process noise covariance matrix, Q, is assumed to be a constant diagonal matrix. The diagonal elements are selected based on knowledge of the process model, considering potential variation in the parameters and inputs that impact model accuracy.

Table 4 summarizes the dynamic equations used to make the model predictions for the SAF model. The parameters and inputs involved in the dynamics of each state variable should be considered when choosing an appropriate process noise associated with each state. Another important consideration is the presence of unmeasured state variables. The variances associated with unmeasured states need to be appropriately selected, as too large values can cause divergence of the state estimator. Lastly, each of the state estimation algorithms calculate the final state estimate based on the ratio of process noise to measurement noise. Thus, when selecting the process noise, the magnitude of the measurement noise should also be considered.

Table B.1 summarizes the process model uncertainties and measurement uncertainties for each of the 17 state variables. The state variables are all molar amounts or temperatures. From Table B.1, most of the molar amounts have process noise with a standard deviation of $100\ mols$. This is chosen to reflect the parametric uncertainty of the kinetic parameters and heat transfer coefficients. Based on literature studies presented in sub-section 2.1.10.1, expected parameter deviation ranges between $5-35\%$ of the parameter original value. This level of parametric uncertainty in the model equations reflects as a standard deviation of about $100\ mols$ within the molar amount state variables over the timestep of $10\ s$.

The molar amount state variables with a larger standard deviation of the process noise of $1000\ mols$ are $N_{C(B),X0}$, N_S and N_M. This added uncertainty is attributed to the potential variation in the inputs, whereby small variation in the charging and tapping flowrates or the charging composition causes a large variation in the model prediction. $N_{C(B),XS}$ and $N_{C(B),XS_2}$ are also affected by potential variation in the charging rate and composition, however, these molar mounts are significantly smaller than $N_{C(B),X0}$ and are not as greatly affected by input variation.

All temperature state variables have process noise with standard deviations of $1\ K$. This uncertainty is attributed to the parametric uncertainty of the heat transfer coefficients. Relatively large variation in the heat transfer coefficients does not cause significant variation in the model prediction of the temperatures over a timestep of $10\ s$. The model prediction of $T_{C(B)}$ is further affected by inputs to the process, however, variation in these inputs also does not cause significant variation in the state. T_W is affected by the input of the flowrate of cooling water, however, input variation is not expected to cause significant variation, thus, a standard deviation of the process noise of $1\ K$ is selected.

Additionally, Table B.1 compares the process noise with the measurement noise for measured state variables. The level measurements have a large degree of uncertainty associated with them due to the inaccuracies of the measurement method. The concentration of reactions gases has a much smaller

measurement uncertainty compared to the process model uncertainty arising from parametric uncertainty.

The last measurement that must be considered is the freeboard pressure measurement. The pressure in the freeboard is calculated from $N_{G,A}$, $N_{G,R}$, and T_G. The standard deviation of the pressure measurement noise is $2\ Pa$ and during NOCs the pressure remains relatively constant. The pressure in the freeboard is very sensitive to small changes in the state variables, $N_{G,A}$, $N_{G,R}$, and T_G. Thus, it is important that the process noise associated with these parameters is not too large as this reflects as unrealistic model predictions for P_G. This is especially important in the PF algorithm during the likelihood calculation. Upon implementation of the PF algorithm using the process noise in Table B.1, unacceptably small likelihoods are calculated due to the large differences between the measurement P_G and the model predicted value for P_G, causing fairly accurate particles to be assigned low weightings. Thus, smaller process model uncertainty values are selected for $N_{G,A}$, $N_{G,R}$, and T_G to ensure appropriate performance of the PF. Table 10 summarizes the final values for the process model uncertainties used in the process noise covariance matrix supplied to the state estimators.

The standard deviations associated with the process noise as discussed above refer to the continuous-time process noise, or the expected random variation in the state variables over a specified timestep of $\Delta t = 10s$. Table 10 converts these values to the standard deviation of the variation of the state variable per second.

Table 10: Final process model uncertainty for each of the state variables.

State	Standard deviation of the process noise	Unit
$N_{C(B),X0}$	100	mol/s
$N_{C(B),XS}$	10	mol/s
$N_{C(B),XS_2}$	10	mol/s
$T_{C(B)}$	0.1	K/s
$N_{C(S),X0}$	10	mol/s
$N_{C(S),XS}$	10	mol/s
$N_{C(S),XS_2}$	0.1	mol/s
$T_{C(S)}$	0.1	K/s
$N_{C(R)}$	10	mol/s
N_S	100	mol/s
T_S	0.1	K/s
N_M	100	mol/s
T_M	0.1	K/s
$N_{G,A}$	0.01	mol/s
$N_{G,R}$	0.01	mol/s
T_G	0.01	K/s
T_W	0.1	K/s

The standard deviations of the process noise associated with each state variable, presented in Table 10, represent the realistic magnitudes of the noise incurred when significant plant-model mismatch is expected in the form of parametric uncertainty and unknown disturbances in the inputs. As highlighted in sub-section 2.3.2.2, when no or modelling errors occur in reality, an overestimated value for Q significantly degrades the accuracy of the state estimates. For this study, two scenarios will be assessed; the performance of state estimators when supplied a large process noise, presented in Table 10, to assess the impact that an overestimated value for Q has on the performance of the state estimators. The second scenario assessed is the performance of state estimators using a smaller process noise covariance matrix constructed using standard deviations of the process noise of 1% of the values presented in Table 10. This small value of Q is a more accurate representation of the system simulated in this study when no plant-model mismatch is simulated.

5.1.4 Number of particles in the PF

The method used to select the number of particles for this study is based on the likelihood approach to adaptive particle filtering described in sub-section 2.1.5.9. For this study, the method presented for on-line adaption of the number of particles is employed for particle number selection purposes prior to implementation of the filter, rather than on-line adaption during filter operation. The number of particles in the final PF algorithm remains constant during operation and is chosen *a* priori using the following selection procedure:

1) Initiate the PF with $N = 10$ particles.
2) At each timestep calculate the sum of non-normalized loglikelihoods. The likelihood is calculated in the PF algorithm in the update step using Equation 27.

$$q_i = \frac{1}{(2\pi)^{\frac{n_y}{2}}|R|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}\left(y_k - \hat{y}_{k,i}^{-}\right)^T R^{-1}\left(y_k - \hat{y}_{k,i}^{-}\right)\right)$$

The log-likelihood is $\log(q_i)$ and the sum of log-likelihoods is calculated as the sum of all the log-likelihoods of each of the N particles.

$$\sum_{i=1}^{n} \log(q_i)$$

This is done over a specified time period on a set of test measurements.

3) Repeat the procedure with double, $2N$, the number of particles on the same set of test measurements.
4) Repeat this entire procedure on an additional 9 set of test measurements to smooth the results.

As the number of particles increases, the log-likelihood should increase, indicating improved filter performance as the distribution can be more accurately approximated and there exist more significant overlap between the measurement distribution and the prior distribution. As the particle number continues to increase, the log-likelihood increase begins to flatten. This creates a scenario whereby additional computational effort, incurred by increasing the number of particles, yields progressively

smaller benefits. Therefore, the 'best performing' PF is selected as the number of particles that results in a mean log-likelihood within 10% of the mean log-likelihood obtained using the previous number of particles. This threshold of 10% dictates the point whereafter there is no significant benefit in increasing the number of particles further and the increased computational burden is no longer justified. The final particle number selection is the number of particles achieving a mean log-likelihood within one standard deviation of the 'best performing' number of particles.

Figure 12 depicts the results from the implementation of this procedure in a boxplot of the sum of the log-likelihoods obtained using various numbers of particles in the PF algorithm.

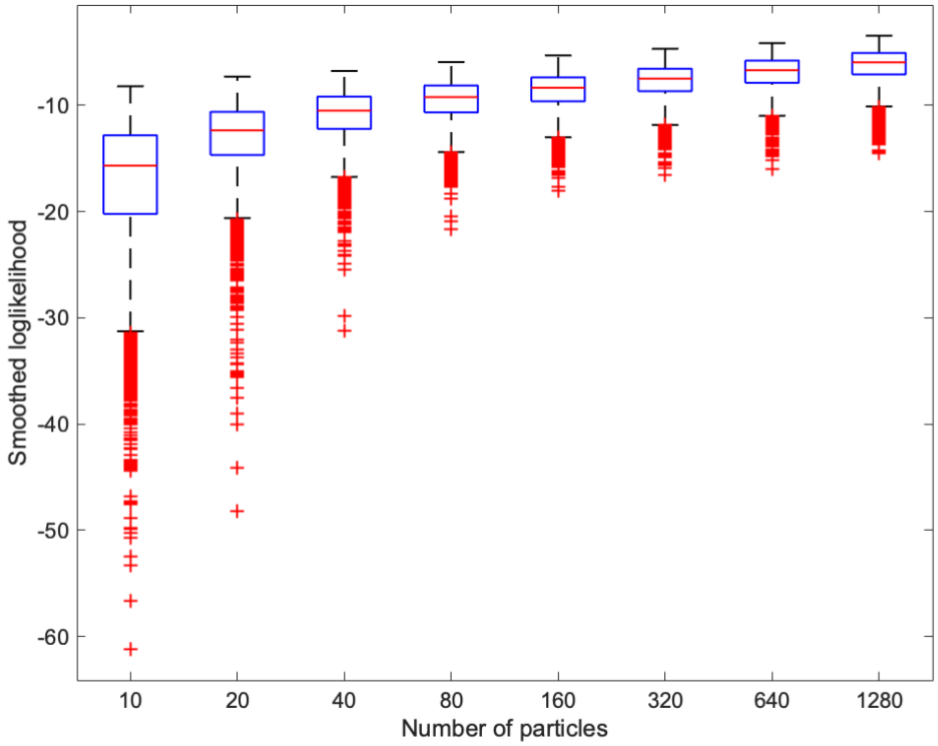

Figure 12: Smoothed boxplot of the sum of the log-likelihood versus the number of particles.

Figure 12 shows that as the number of particles doubles, the log-likelihood increases. As the number of particles continues to double, the log-likelihood begins to flatten. The mean log-likelihood using 160 particles is within 10% of the mean log-likelihood using 80 particles. Thus, 160 particles is considered the 'best' performing PF. The PF model that achieves a mean log-likelihood that falls within one standard deviation of the 160 particle model is the 80 particle model. It can be concluded that the appropriate number of particles for this system is 80 particles. In the PF algorithm presented in sub-section 2.1.5.4, $N = 80$.

5.1.5 MHE horizon length

As per the literature review on the MHE horizon length selection presented in sub-section 2.1.6.3, the two methods for selecting an appropriate horizon length are to evaluate the estimation error at different horizon lengths and to consider the time constants of the dynamic model. Both these methods are employed.

5.1.5.1 Time constants of the SAF model

The SAF system is linearized around the initial values presented in Table C.2 to obtain the linearized state transition matrix, A. Eigen decomposition is performed on A to obtain the eigenvalues of the system. Table 11 presents the eigenvalues ordered from the largest negative value to the smallest negative value with the corresponding time constants ordered from shortest time constant to longest time constant.

Table 11: Eigenvalues and corresponding time constants of the linearized SAF model.

Eigenvalue	Time constant (s)
-286.32	3.49E-03
-0.63	1.59
-0.24	4.14
-0.21	4.81
-0.01	73.76
-2.96E-03	337.77
-1.38E-03	724.26
-1.17E-03	854.64
-5.84E-04	1.71E+03
-3.67E-04	2.73E+03
-2.51E-04	3.99E+03
-2.16E-04	4.64E+03
-1.63E-04	6.13E+03
-4.88E-05	2.05E+04
-3.99E-06	2.50E+05
0.00	-
0.00	-

From Table 11, all the eigenvalues of the system are negative, with the exception of the last two eigenvalues being calculated as very small negative numbers and approximated as zero eigenvalues. The largest negative eigenvalue is exceptionally large compared to the other eigenvalues, with a corresponding time constant of $3.49 \times 10^{-3}\ s$, indicating very fast state dynamics. The first four eigenvalues have relatively fast dynamics of less than $5\ s$. Thereafter, the eigenvalues are very small negative numbers, resulting in long time constants indicating slow state dynamics.

The states associated with each eigenvalue are analyzed using the entries in the corresponding eigenvectors. Figure 13 shows the state variables that contribute to the last two eigenvalues, the zero eigenvalues.

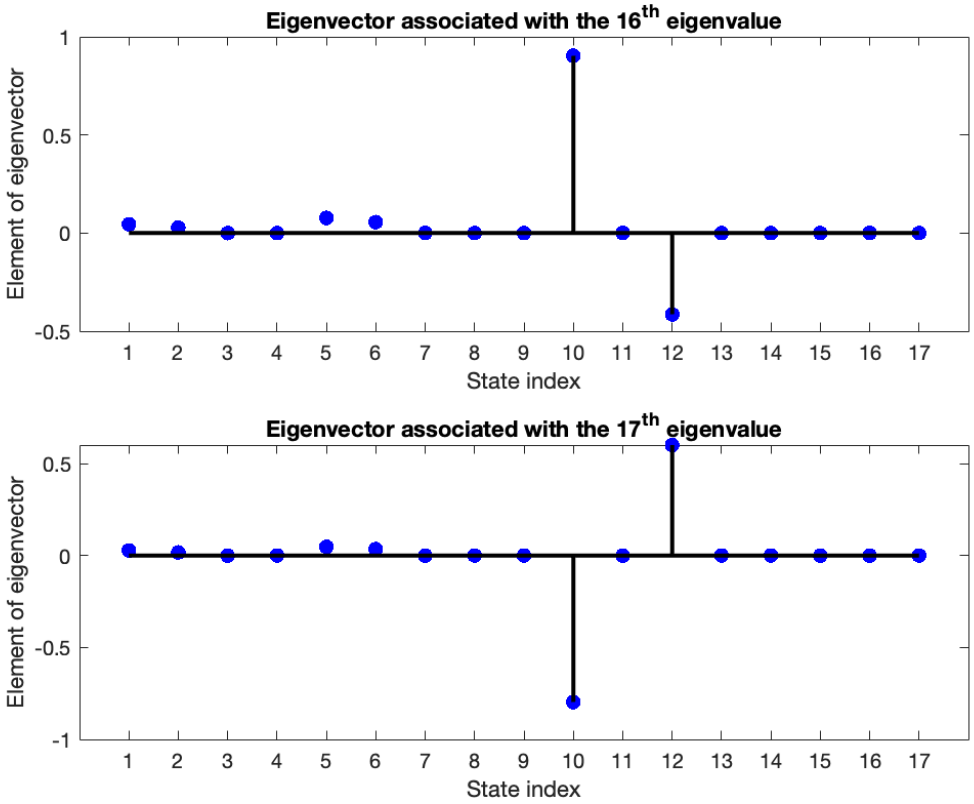

Figure 13: Elements of the eigenvectors associated with the last two eigenvalues of A.

From Figure 13, the states associated with the zero eigenvalues are x_{10} and x_{12}, the number of moles in the slag zone and the matte zone, respectively. These are the integrator states, whose dynamics do not depend on the states themselves and are therefore not self-regulating. The rate of change of x_{10} is dependent on the smelting rate of the slag component, which is dictated by the state variables in the smelting concentrate zone and the slag tapping rate, which is an input to the process that is externally controlled. Similarly, the rate of change of x_{12} is dependent on the smelting rate of matte component and the matte tapping rate. The rate of change of these states do not directly depend on the states themselves.

Figure 14 is a plot of the entries in the eigenvectors associated with the first and second eigenvalues.

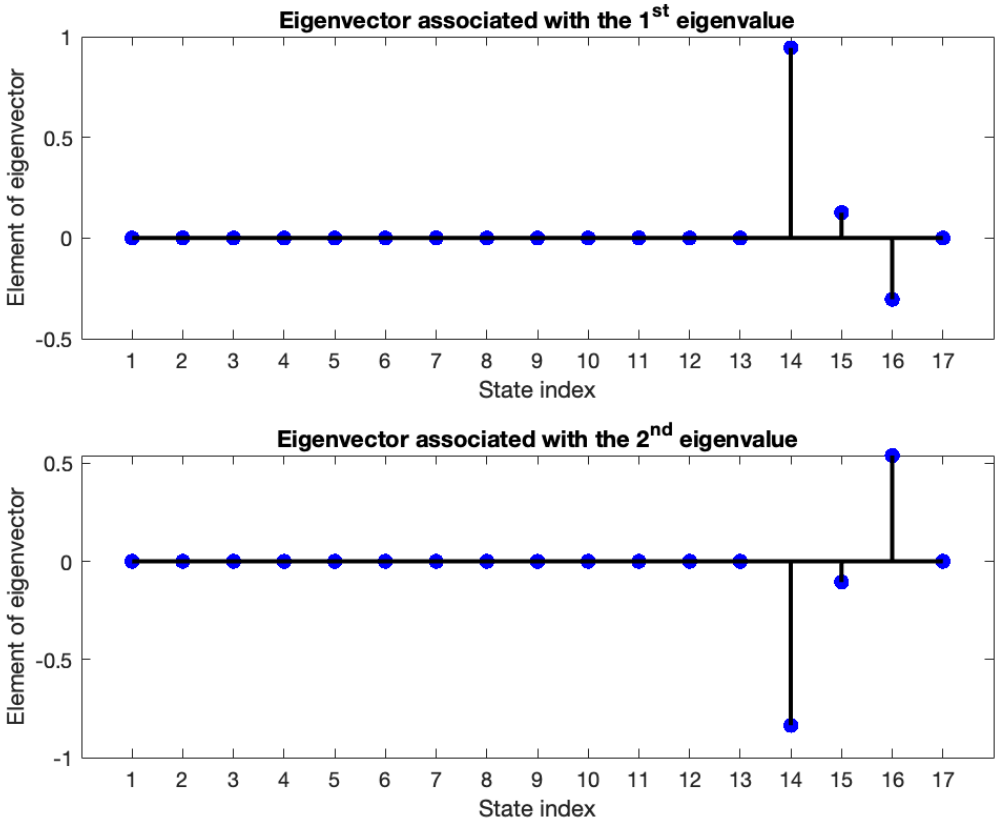

Figure 14: Elements of the eigenvectors associated with the first and second eigenvalues of A.

From Figure 14, states x_{14}, x_{15}, and x_{16} are associated with the first and second eigenvalues and therefore have the fastest dynamics. These states are the number of moles of reaction gases in the freeboard $N_{G,R}$, the number of moles of air in the freeboard $N_{G,A}$,, and the temperature in the furnace freeboard T_G, respectively. These exceptionally fast gas dynamics can be explained by analysis of the SAF model ODEs in Table 4 and the initial model conditions of Table C.2. The exceptionally fast dynamics associated with the gases compared to the slow dynamics within the concentrate, matte, and slag regions of the furnace are due to the significantly smaller molar amounts of gases within the freeboard compared to the large molar amounts of liquid within the other zones.

Upon analysis of the time constants of the SAF process model, the first eigenvalue in Table 11 is a large negative eigenvalue with a corresponding exceptionally small time constant. The last few eigenvalues of Table 11 are small with corresponding long time constants. It is evident that there is an inherent stiffness in the system equations. The system stiffness is exacerbated by the exceptionally fast gas dynamics. This SAF system displays characteristics of a singularly perturbed system due to this presence of time-scale multiplicity. Handling of time-scale multiplicity using singular perturbation theory is explained in sub-section 2.1.6.3 c.

Sub-section 2.1.6.3 b explains the implications of working with a stiff system of equations. Longer horizon lengths are required in the MHE algorithm to ensure stability and convergence, thus, resulting in a high computational burden. Several literature studies, presented in sub-section 2.1.6.3 b, propose modified

MHE algorithms for handling stiff systems of equations in an attempt to reduce the computational burden of the MHE.

Analysis of the computational requirements of the MHE algorithm employed for state estimation of the SAF states highlights that the greatest computational load stems from the integration in the prediction step. During the prediction step, a trajectory of predicted state estimates is obtained by integrating the system of ODEs over $10\ s$. This integration is performed multiple times in the optimization routine. The built-in MATLAB solver *ode15s* used in the prediction step is designed to handle stiff ODEs using an adaptive step-size. During this relatively short integration interval of $10\ s$, the solver does not have sufficient time to adapt the step size and a large computational effort is incurred due to the small integration time steps.

The proposed method for reducing the computational requirements of the MHE for this study involves targeting the greatest computational load in the optimization procedure, the integration. An alternative integration method, Runge-Kutta RK4 step, is used between timepoints with a larger specified timestep of $5\ s$. This significantly reduces the computational time spent in the prediction step. However, the solution does not converge for states with dynamics that are significantly faster than this specified timestep of $5\ s$. Therefore, a reduced order model of the system needs to be identified based on the different time-scale multiplicities of the states. For this study, an alternative model based on singular perturbation theory is developed.

5.1.5.2 MHE singular perturbation model

The process model of the SAF supplied to the MHE is divided into two subsystems based on the time-scale multiplicity of the system and singular perturbation theory presented in sub-section 2.1.6.3 c. The system is split by the speed of the dynamics into states with fast dynamics and states with slow dynamics. The speed of the dynamics is classified according to the step size taken by the solver. If the time constant is shorter than the step size, the dynamics are considered fast. Conversely, if the time constant is longer than the step size, the dynamics are considered slow. The states corresponding to short time constants are identified from the eigenvectors corresponding to the largest eigenvalues, these are the fast states, x_f. The remaining states are the slow states, x_s. Based on the analysis of the time constants of the dynamic model presented in sub-section 5.1.5.1, the fast states of the singular perturbation model are:

$$x_f = x_{14}, x_{15}, x_{16}$$

The remaining states are the slow states:

$$x_s = x_1 \rightarrow x_{13}, x_{17}$$

The original nonlinear SAF system dynamics are represented by:

$$\frac{dx}{dt} = f(x) + w \qquad [\ 110\ ]$$

The system dynamics are split into fast and slow dynamics according to:

$$\frac{dx}{dt} = \frac{d}{dt}\begin{bmatrix} x_s \\ \varepsilon x_f \end{bmatrix} = \begin{bmatrix} f_s(x_s, x_f) \\ f_f(x_s, x_f) \end{bmatrix} + \begin{bmatrix} w_s \\ w_f \end{bmatrix} \quad [111]$$

$$w_s \sim (0, \sigma_s^2)$$

$$w_f \sim (0, \sigma_f^2)$$

Whereby the dynamics and the process noise are separated based on the speed of the dynamics of the states. The slow states have dynamics from the original SAF model. The perturbation parameter, ε, is set to zero as the fast dynamics are assumed to run in quasi-steady state. This enables the use of the RK4 method with larger step sizes to be used in the integration step. Thus:

$$f_f(x_s, x_f) = 0 \quad [112]$$

During the prediction step, the fast states remain constant, $x_{f,k+1}^{-} = x_{f,k}^{+}$. The slow states have their original dynamics and original process noise variance. To account for the inaccurate prediction of fast states, the variance of w_f is chosen to be exceptionally large. This induces uncertainty in the process model for the fast states, and the state estimate is biased towards the measurement.

$$\sigma_{f,original} = [0.01 \quad 0.01 \quad 0.01]$$

$$\sigma_{f,modified} = [10 \quad 10 \quad 0.1]$$

The singular perturbation model using a *RK4* step in the prediction step is employed to reduce the computational effort of the MHE. This singular perturbation model makes accurate predictions for the slow states but poor predictions for the fast states. However, these poor predictions of the fast states are rectified for in the update step. Alternative approaches such as reduced-order modelling degrade the prediction accuracy for all states and/or reduce the number of states estimated.

State estimation using the MHE to estimate the 17 state variables of the SAF is performed using the original SAF model and *ode15s* for the prediction step and the singular perturbation model with *RK4* for the prediction step. State estimation is performed using varied horizon lengths in the MHE algorithm. Table 12 shows the average computational times achieved by the MHE using both models under various horizon lengths.

Table 12: Average computational times for the MHE using the original model and the singular perturbation model at various horizon lengths.

Horizon length	Average computational time per estimate (s)		Ratio of average computational time (original/modified)
	Original model	Singular perturbation model	
2	7.70	4.43	1.7
3	27.15	18.60	1.5
4	62.01	36.74	1.7
5	93.83	76.11	1.2

6	126.49	105.54	1.2
7	173.19	170.92	1.0
8	233.22	204.43	1.1
9	297.02	318.06	0.9
10	385.63	413.99	0.9

Table 12 shows the average computational time per estimate is significantly reduced when using the singular perturbation model with the RK4 step at short horizon lengths, with the singular perturbation model being $1.5 - 1.7$ times faster than the original model using *ode15s*. However, for horizon lengths longer than 7, the singular perturbation model requires comparable computational times to the original model. The average computational time per estimate at longer horizon lengths becomes intractable for both models. Therefore, the horizon length selection is ultimately limited by the computational effort.

The estimation accuracy of the models is also compared by calculating the MAPE of the 17 state estimates. Figure 15 illustrates the MAPE for each of the state estimates across different horizon lengths achieved by the MHE using both the singular perturbation model and the original model.

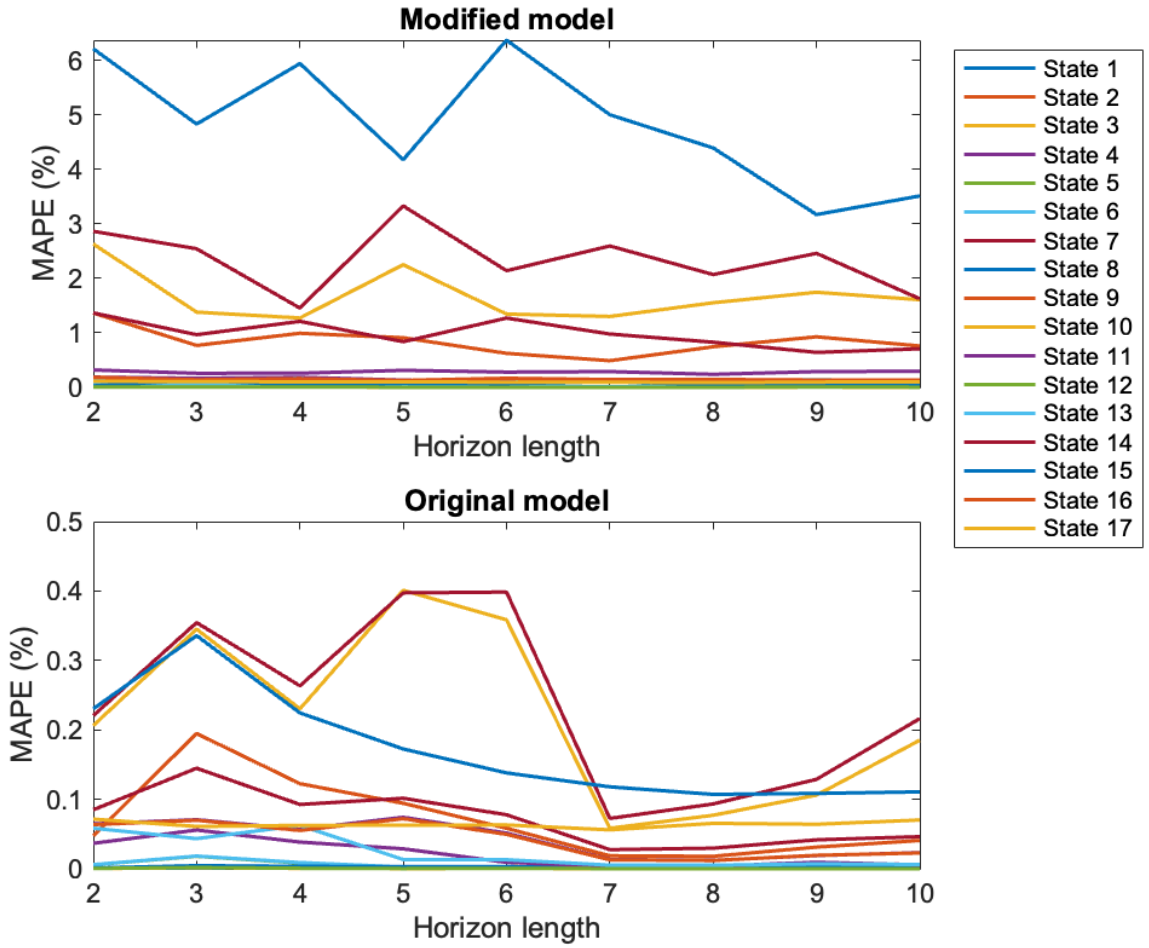

Figure 15: MAPE of the state estimates under different horizon lengths achieved by the MHE using a singular perturbation model and RK4 prediction step and the MHE using the original model with *ode15s* prediction step.

For state estimates $\hat{x}_3, \hat{x}_7, \hat{x}_9, \hat{x}_{14}$, and $\hat{x}_{15}$, the singular perturbation model results in notably larger MAPEs than the original model over all horizon lengths. Most of the state estimates show an overall decrease in MAPE for an increase in horizon length for both models. However, this is not always a consistent decrease between each horizon length increase. As explained by Larsson (2015), this is attributed to the complex dynamic relationships between variables and cost function approximation error. For the state estimates that do show a decrease in MAPE, increasing the horizon length results in marginally better estimation accuracy at the cost of a significant increase in computational effort. Therefore, the horizon length is selected based on achieving an acceptable computational time, rather than striving for the highest estimation accuracy.

A horizon length selection of 4, corresponding to a horizon time of $40\ s$, is ultimately selected. This is based on the significant improvement in computation time achieved using the singular perturbation model at this horizon length, seen in Table 12. This reduced computational time justifies the loss of estimation accuracy seen in Figure 15.

5.2 State estimation results

This section presents the results obtained to address objective 2 of the study. Sub-section 5.2.1 presents the results from investigating the state estimation performance using a larger process noise appropriate for robust state estimation. Section 5.2.2 presents the results from evaluating state estimation performance using a smaller process noise covariance that is more appropriate for model-based fault detection applications. These results facilitate selection of state estimation techniques for performing model-based fault detection to facilitate objective 3 of the study.

5.2.1 State estimation performance under large process noise

The EKF, UKF, PF, MHE using the original process model with *ode15s* prediction step and the MHE using the singular perturbation model with *RK4* prediction step, were used to estimate the 17 states of the SAF. The state estimation algorithms use the large process noise covariance presented in Table 10 of sub-section 5.1.3.3. Table 13 presents the MAPE for the state estimates generated by the EKF, UKF, PF, and MHE.

Table 13: MAPEs of the state estimates generated for EKF, UKF, PF and MHE.

	MAPE (%)				
State	**EKF**	**UKF**	**PF**	**MHE (original model)**	**MHE (singular perturbation model)**
$N_{C(B),X0}$	0.599	0.003	0.814	0.001	0.003
$N_{C(B),XS}$	0.094	0.004	0.190	0.002	0.004
$N_{C(B),XS_2}$	4.872	1.242	20.504	0.127	2.645
$T_{C(B)}$	0.147	0.077	0.094	0.055	0.381

$N_{C(S),X0}$	0.148	0.032	0.044	0.016	0.054
$N_{C(S),XS}$	0.085	0.033	0.084	0.016	0.057
$N_{C(S),XS_2}$	5.583	1.295	23.551	0.250	2.795
$T_{C(S)}$	0.078	0.011	0.165	0.013	0.073
$N_{C(R)}$	7.219	2.084	4.951	0.181	1.331
N_S	0.049	0.002	0.129	0.001	0.004
T_S	0.184	0.124	0.085	0.113	0.246
N_M	0.061	0.004	0.053	0.002	0.005
T_M	0.266	0.116	0.227	0.111	0.212
$N_{G,A}$	0.763	0.342	0.470	0.062	1.141
$N_{G,R}$	3.450	1.402	2.275	0.193	5.845
T_G	0.137	0.077	0.087	0.054	0.162
T_W	0.204	0.094	0.134	0.082	0.103

Table 13 shows that for each respective state estimation technique, the largest MAPE is associated with the state estimates $N_{C(B),XS_2}$, $N_{C(S),XS_2}$, $N_{C(R)}$, and $N_{G,R}$. Based on the state observability analysis results presented in chapter 4 sub-section 4.2, states $N_{C(B),XS_2}$, $N_{C(S),XS_2}$, $N_{C(R)}$ are associated with directions with poor observability. Therefore, it can be ascertained that there is a link between the degree of observability of the states and the expected estimation accuracy associated with the state estimates.

The MHE using the original model displays the best state estimation performance in terms of the estimation accuracy, with all 17 state estimates achieving the lowest MAPEs compared to the other five estimators investigated. The MHE using the singular perturbation model has an elevated MAPE for some state estimates, resulting in larger MAPEs than the original MHE and the UKF. The UKF displays good estimation accuracy, having relatively small MAPEs for all estimated states. The EKF displays larger MAPEs than the UKF and MHE for all estimates state variables. The PF exhibits the poorest estimation accuracy of the five methods investigated.

Similar results have been found in literature, namely Chatzi & Smyth (2002) found that the UKF outperforms the PF. This is explained by the selection of relatively large values for the process noise covariance to account for any potential plant-model mismatch. The PF algorithm is sensitive to high values for the process noise covariance supplied to the estimator. The optimal importance density can be accurately approximated by the transition density only under small process noise (Arulampalam & Ristic, 2000). When high process noise is selected for the PF, particle degeneracy is reduced, however, this has been proven to come at the cost of increased bias (Wigren et al., 2018). Therefore, the higher MAPEs associated with the PF estimates are explained by this increased bias.

Table 14 presents the results for the average computational time per estimate for each of the state estimation techniques.

Table 14: Average computational time per estimate achieved by each state estimation technique.

Average computational time (s)	EKF	UKF	PF	MHE (original model)	MHE (singular perturbation model)
	0.21	0.36	0.87	75.51	41.00

According to Table 14, the EKF, UKF, and PF require short average computation times per estimate, whilst both MHE techniques have considerably larger computational requirements. These computational times for the MHE are comparable to other literature studies. For example, a study done by Alexander *et. al.* (2020) used the MHE with a horizon length of 30 to estimate the concentrations, 3 state variables, in a simple batch reactor and required an average computational time per estimate of $42s$.

In fulfillment of objective 3 of the study, it is necessary to select one or more state estimators for incorporation in the model-based fault detection algorithm. This selection is based on the above performance evaluation of the four techniques. It is evident that the computational requirements of the MHE, using both the original model and the singular perturbation model, are inappropriate for fault detection purposes. Therefore, the choice lies between the EKF, UKF, and PF. The computational times of these three methods are comparable. The UKF achieves the most accurate state estimates, whilst the PF displays the highest estimation error. Further investigation of the EKF, UKF, and PF, using a smaller process noise covariance matrix is conducted.

5.2.2 State estimation performance under small process noise

For the purposes of objective 2 of this study, the performance of the various state estimators is assessed under no plant-model mismatch. Therefore, the larger process noise covariance matrix is an over estimation of the true process noise covariance, thus, the performance of the state estimators is significantly degraded. The EKF, UKF, and PF algorithms using a smaller process noise covariance were used to estimate the 17 states of the SAF. This smaller process noise covariance is equivalent to 1% of the large process noise covariance presented in Table 10 of sub-section 5.1.3.3. This value for Q is more appropriate for this application, as no plant-model mismatch is simulated.

Figure 16 summarizes the findings of the performance evaluation in a boxplot of the MAPEs of the state estimates generated by the EKF, UKF, and PF under the smaller process noise versus the computational times.

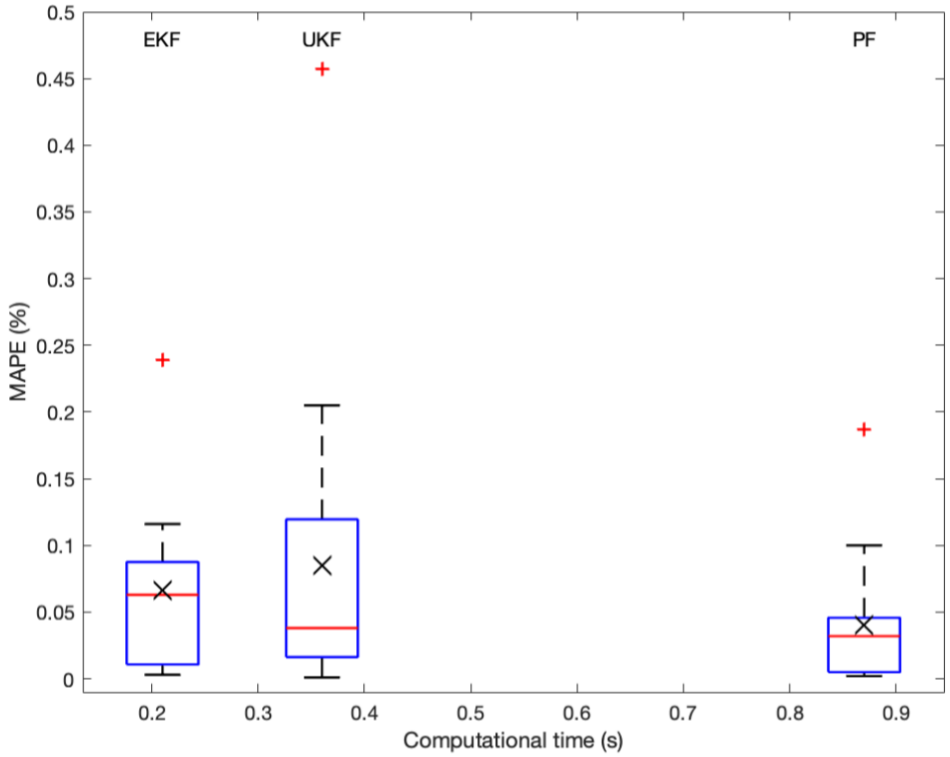

Figure 16: Boxplot of the MAPEs obtained for the state estimates versus the average computational time required for the EKF, UKF, and PF under low process noise.

The MAPEs achieved by all three filters are significantly lower than the MAPEs obtained under high process noise conditions presented in Table 13. This is expected, as the investigations are carried out under zero modelling error. Thus, the overestimated large process noise degrades estimation accuracy of state estimators in the presence of no plant-model mismatch. The results presented in Figure 16 show that all three filters display comparable performance obtaining relatively low MAPEs for all state estimates. This shows that the state estimates converge to close to the same value for all three algorithms under low process noise. Thus, all three filters are able to accurately track the model. The PF obtains the lowest average MAPE (black cross), displaying slightly superior estimation accuracy under low process noise. The slightly superior performance of the PF can be attributed to the PFs ability to approximate the severe nonlinearities of the system more accurately.

The EKF is selected as one of the state estimation techniques for model-based fault detection. This choice is based on the literature review on model-based fault detection presented in sub-section 2.3.2.1 and the performance evaluation results presented above. The literature consensus is that the EKF is the most widely used state estimator in general and in model-based fault detection using state estimators. The results above highlight the simplicity of EKF algorithm as it has the lowest computational times. Furthermore, the results illustrate the comparative, and sometimes superior, estimation accuracy achieved by the EKF compared to the more complex nonlinear filters.

The PF is the second state estimation technique selected for model-based fault detection. The PF is selected over the UKF as it achieves superior estimation accuracy under small process noise, shown by Figure 16. The PF is also unique in its ability to handle non-Gaussian state distributions.

6 FAULT DETECTION APPROACH AND RESULTS

The following chapter addresses objective 3 of this study: comparing model-based fault detection using the state estimation techniques identified in objective 2 to a data-driven method of fault detection and assessing the impact of plant-model mismatch on the performance of the model-based method. Section 6.1 outlines the approach used to complete this objective and section presents 6.2 the results in fulfilment of objective 3 of this study.

6.1 Fault detection approach

This section outlines the procedure used in this study to conduct model-based fault detection using state estimation and data-driven fault detection in order to complete objective 3. Sub-section 6.1.1 details the procedure for conducting model-based fault detection. Sub-section 6.1.2 presents the methodology used to implement data-driven fault detection. Lastly, sub-section 6.1.3 details how plant-model mismatch is simulated for this study.

6.1.1 Model-based fault detection methodology

Figure 17 summarizes the procedure for performing model-based fault detection used in this study.

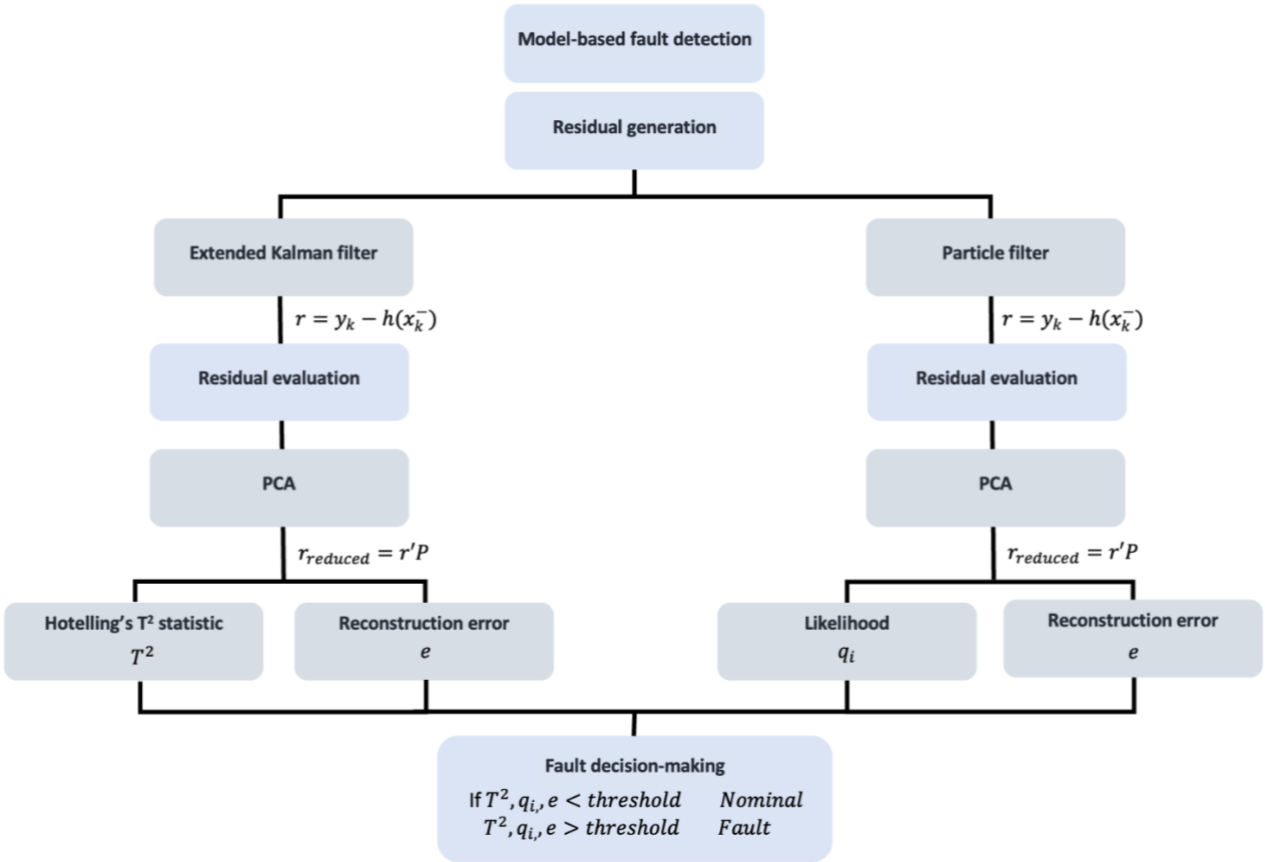

Figure 17: Model-based fault detection methodology.

Based on the state estimation results, presented in sub-section 5.2.2, the EKF and the PF are identified as appropriate state estimation techniques for model-based fault detection. The literature review

presented in sub-section 2.3.2 explains that model-based fault detection involves a residual generation step and a residual evaluation step. From Figure 17, residual generation in this study uses the EKF and PF to generate residual signals. The EKF and PF are supplied a smaller process noise covariance matrix constructed using process noise uncertainties of 1% of the values presented in Table 10. This residual generation step is followed by a residual evaluation step.

The residual evaluation procedure adopted in this study is based on the idea proposed by Yang (2004) presented in sub-section 2.3.4. Figure 17 shows that the residuals from the EKF and PF are evaluated by projecting the residuals into the reduced dimension-space defined by a PCA model. Following the theory presented in sub-section 2.3.4, PCA attempts to control the bias-variance tradeoff in the residual evaluation step.

The PCA models are trained on residuals generated by the EKF and PF under NOCs, following the PCA procedure outlined in sub-section 2.3.5. The trained PCA models can be constructed using various numbers of retained PCs, ranging from 1 to the maximum 9. The maximum number of PCs is equal to the number of residuals in the original feature space, which is 9 for the SAF process as there are 9 measured variables. When all 9 PCs are retained, the reduced PCA model is simply equivalent to the original model and no dimensionality reduction has taken place.

The PCA model is tested on new residuals. New residuals are generated by the EKF and PF under both nominal and faulty conditions. The EKF and PF process models are the nominal models of the SAF, therefore, under faulty measurements the residuals should be large. The four faulty conditions outlined in sub-section 3.4.1, namely a fault in the flowrate of the cooling water, a fault in the extraction pressure, a fault in the composition of the furnace charge, and furnace blowback, are simulated in the synthetically generated measurements supplied to the state estimators. These new residuals from the EKF and PF are then projected into the reduced-dimensional space defined by their respective trained PCA models. Monitoring statistics are calculated for each residual projected in the reduced dimensional space and the monitoring statistics are used in the fault decision-making step, whereby statistic values exceeding a pre-defined threshold are classified as faulty.

The EKF is formulated based on an assumption that the states and measurements, and therefore the residuals, have approximately Gaussian distributions. Based on the theory presented in sub-section 2.3.5.1, the T^2 statistic thresholds are defined based on the assumption that observation data is approximately Gaussian. The EKF residuals are evaluated using Hotelling's T^2 statistic, presented in sub-section 2.3.5.1, to capture the within-plane error of the PCA model and the reconstruction error, as presented in sub-section 2.3.5.1, to capture the out-of-plane error of the PCA model. If the underlying distribution is not approximately Gaussian, the EKF may display poor estimation accuracy and the T^2 statistic thresholds of the distribution may be inaccurate.

The PF is a non-parametric state estimator that makes no assumptions about the underlying distribution and is therefore able to approximate non-Gaussian state distributions more accurately than the EKF. An alternative monitoring statistic to the T^2 statistic is defined to measure the within-plane error of PF residuals in the PCA model that does not make assumptions about the shape of the distribution of

residuals. This alternative statistic is the relative likelihood of the particles, q_i. The reconstruction error captures the out-of-plane error of the PF residuals in the PCA model as it makes no assumptions about the shape of the underlying distribution and the likelihood captures the within-plane error of the PCA model.

6.1.1.1 Relative likelihood monitoring statistic

a) Relative likelihood in the original feature space:

Following the PF algorithm presented in sub-section 2.1.5.4, at each timestep k, there exist N particles representing the *a priori* state estimates, $x_{k,i}^-$. Each sample is transformed with the nonlinear measurement equation h and the transformed particles are represented as $h(x_{k,i}^-)$. Each transformed particle within measurement space represents a Gaussian distribution with covariance R. The sum of the Gaussians represents the prior distribution in the measurement space.

At each timestep there is also a measurement, y_k. The relative likelihood that the measurement is drawn from the Gaussian distribution defined by the predicted measurement, $h(x_{k,i}^-)$, is calculated as:

$$q_i \sim \frac{1}{(2\pi)^{ny/2}|R|^{1/2}} \exp\left(\frac{-\left(y_k - h(x_{k,i}^-)\right)^T R^{-1}\left(y_k - h(x_{k,i}^-)\right)}{2}\right) \quad [\ 113\ ]$$

(Pardal et al., 2015)

Since each of the particles represents a sample from a distribution, the likelihood at timestep k is solved for using Monte Carlo integration:

$$L_k = \frac{1}{N}\sum_{i=1}^{N} q_i \quad [\ 114\ ]$$

The likelihood L_k, is calculated for each new observation, y_k, for each timestep k. The monitoring statistic is the log-likelihood.

b) Relative likelihood in the reduced-dimension feature space:

At each timestep, the residual from the PF can be obtained for each particle as $\bar{r}_{k,i} = \bar{y}_k - h\left(\bar{x}_{k,i}^-\right)$. The residual can be transformed into the reduced-dimension space defined by trained PCA model using the loadings vector, P:

$$r_{k,i,reduced} = P' r_{k,i} = P'\left(\bar{y}_k - h\left(\bar{x}_{k,i}^-\right)\right) \quad [\ 115\ ]$$

The original measurement noise covariance matrix used in the PF algorithm, R, can be transformed into the measurement noise covariance matrix in the reduced space:

$$R_{reduced} = P^T R P \quad [\ 116\ ]$$

The relative likelihood in a reduced-dimension space is calculated as:

$$q_{i,reduced} \sim \frac{1}{(2\pi)^{A/2}|R_{reduced}|^{1/2}} \exp\left(\frac{-\left(r_{k,i,reduced}\right)^T R_{reduced}^{-1}\left(r_{k,i,reduced}\right)}{2}\right) \quad [\ 117\ ]$$

Where A is the number of PCs retained.

The likelihood in the reduced-dimension space is calculated as:

$$L_{k,reduced} = \frac{1}{N}\sum_{i=1}^{N} q_{i,reduced} \quad [\ 118\ ]$$

The monitoring statistic is the log of this likelihood in the reduced space.

6.1.2 Data-driven fault detection methodology

The data-driven fault detection methodology used in this study is summarized in Figure 18.

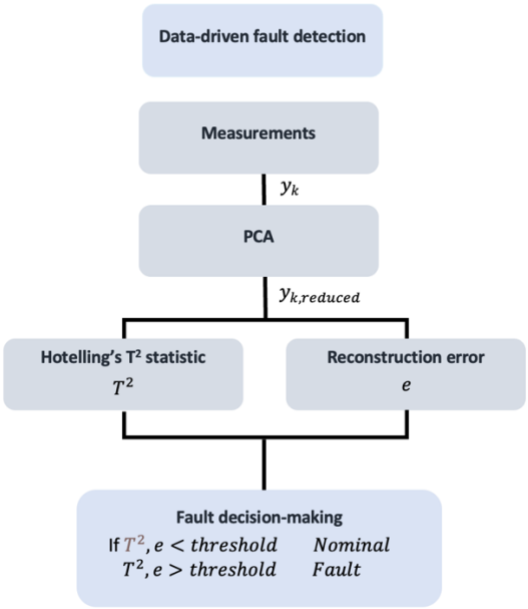

Figure 18: Data-driven fault detection methodology.

Synthetically generated measurements generated under NOCs are used to train the PCA model. New synthetically generated measurements at each timepoint k, y_k, are projected into the reduced dimension space defined by the trained PCA model. The T^2 statistic and reconstruction error are calculated for each new measurement and used in the fault decision-making step to classify measurements as nominal or faulty.

6.1.3 Plant-model mismatch

In fulfillment of objective 3 of this study, the effect of potential plant-model mismatch on the performance of the model-based fault detection using state estimation is investigated. Figure 19 depicts how plant-model mismatch is simulated in this study:

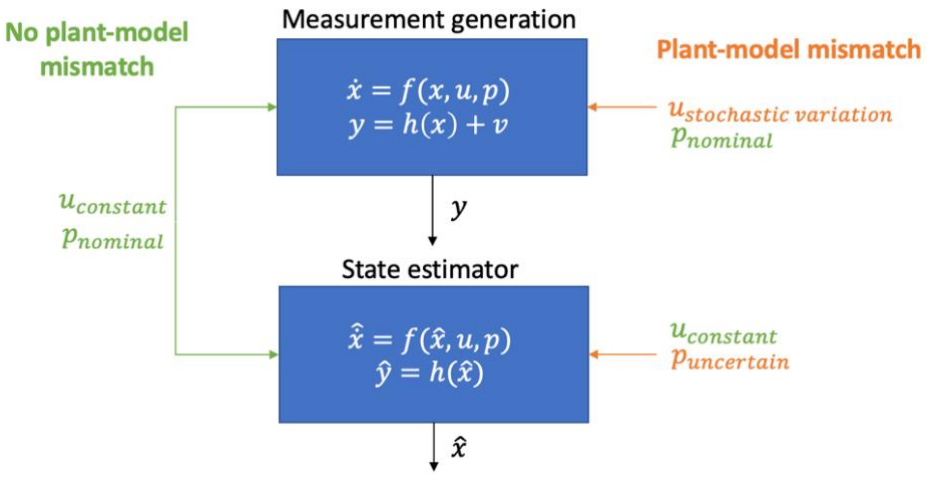

Figure 19: Depiction of the physical implementation of plant-model mismatch used in this study.

For this study, the SAF process dynamics presented in sub-section 3.2 represent the nonlinear function f and the measurement equations presented in sub-section 3.3 represent the nonlinear function h. The SAF process model using f and h is used to generate synthetic measurements with the addition of synthetic measurement noise v. The generated measurements are sent to the state estimation algorithm along with the same SAF process model functions f and h. When there is no plant-model mismatch, both the measurement generator and the state estimator are supplied with the same constant inputs, $u_{constant}$, and nominal model parameters, $p_{nominal}$.

One method of simulating plant-model mismatch used in this study is where uncertainty enters the system via unknown disturbances within the measured inputs. Similar to the methodology outlined in Bavdekar (2013) and presented in sub-section 2.1.10.2, the state estimator uses known constant inputs, $u_{constant}$, in the process model. However, the measurements are generated with inputs that vary stochastically, $u_{stochastic\ variation}$.

The other method involves simulating plant-model mismatch via parametric uncertainty. The measurements are generated with the true nominal parameter values, $p_{nominal}$, whilst the state estimator SAF model uses alternative model parameters, $p_{uncertain}$.

6.1.3.1 Parametric uncertainty

In the SAF model, parameters such as reaction kinetic parameters, gas flux constants, and the heat transfer coefficients are possible sources of parametric uncertainty. For this study, parametric uncertainty is simulated via fixed parametric uncertainty by adding a constant offset, σ, to the original parameter value, $p_{nominal}$.

$$p_{uncertain} = p_{nominal} \pm \sigma \qquad [\ 119\ ]$$

The parameter chosen to simulate plant-model mismatch is the mixing constant, k_V, which dictates the rate of mixing between the smelting and bulk concentrate. The degree of modelling error is varied by increasing the fraction $\frac{k_{V,uncertain}}{k_{V,nom}}$, where $k_{V,uncertain}$ is the incorrect parameter supplied to the state

estimator and $k_{V,nom}$ is the true parameter value. The fraction $\frac{k_{V,uncertain}}{k_{V,nom}}$ is varied from 1, when there is no modelling error, to 5, when there is significant modelling error.

6.1.3.2 Stochastic variation in the inputs

For this study, stochastic variation in the inputs is simulated via an autoregressive (AR-1) process. The AR-1 model is given by:

$$u_t = \beta_1 + \rho u_{t-1} + \varepsilon_t, \quad t = 1, \dots, T \qquad [120]$$

The error term, ε_t, is a white noise process with zero mean and variance σ^2. When $\rho = 0$, then $cov(u_t, u_{t-1}) = 0$, and there is no time dependence between lagged values of u. When $\rho > 0$ then the covariance between u_t and u_{t-1} is positive, thus implying there is autocorrelation within the u data. When $\rho < 0$ then the covariance between u_t and u_{t-1} is negative, which is an uncommon phenomenon in AR processes. When $\rho = 1$ or $\rho = -1$, the process is no longer stable, thus $-1 < \rho < 1$.

The expected value of the u data is given by:

$$E(u_t) = \frac{\beta_1}{1-\rho} \qquad [121]$$

The variance of the u data is given by:

$$Var(u_t) = \frac{\sigma^2}{1-\rho^2} \qquad [122]$$

(Brockwell & Davis, 2016)

The input selected for simulation of stochastic variation is the flowrate of the charging concentrate, F_{charge}.The expected value of the AR-1 model is the constant input value used in the model supplied to the state estimator. The expected value of F_{charge} is $410\ mol/s$. The value of ρ is chosen to be 0.8. An appropriate standard deviation is selected as a percentage of the expected value of the input variable.

$$Var(u_t) = E(u_t) \times (user\ defined\ percentage) \qquad [123]$$

The σ value of the AR model is calculated as:

$$\sigma = \sqrt{Var(u_t)(1-\rho^2)} \qquad [124]$$

The value for β_1 in the AR model is calculated as:

$$\beta_1 = E(u_t)(1-\rho) \qquad [125]$$

The degree of modelling error is varied by increasing the size of the standard deviation used to simulate the stochastic input. The standard deviation is varied as a percentage of the expected value of the input. For this study, the user-defined percentage is varied between $1 - 20\%$.

6.2 Fault detection results

This chapter addresses objective 3 of the study. Model-based fault detection is performed using the state estimation techniques identified in objective 2, the EKF and PF, and residual evaluation using PCA

outlined in section 6.1.1. The performance of the model-based methods are compared to the PCA data-driven method outlined in section 6.1.2. The PCA models built on the EKF residuals, PF residuals, and the measurements are evaluated in section 6.2.1. Section 6.2.2 presents a comparison of the model-based fault detection performance and the data-driven method under no plant-model mismatch and under varying degrees of plant-model mismatch.

6.2.1 PCA model evaluation

Following the model-based fault detection approach presented in sub-section 6.1.1, PCA is used in the residual evaluation step. PCA is also used in the data-driven fault detection methodology, presented in sub-section 6.1.2. For both model-based and data-driven fault detection, the PCA model is trained on nominal observations.

Figure 20 is a scree plot of the cumulative variance explained by each number of retained PCs for each of the trained PCA models. The three PCA models correspond to the three fault detection methods investigated: the model-based fault detection using the EKF residuals, model-based fault detection using the PF residuals, and data-driven fault detection using the measurements.

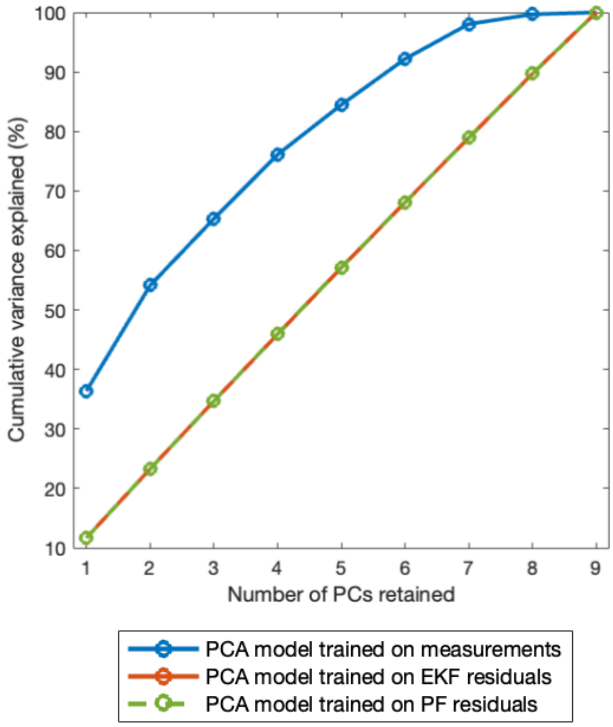

Figure 20: Scree plot of cumulative variance explained for each number of retained PCs for each trained PCA model corresponding to each method of fault detection investigated.

Figure 20 shows that the variance explained in the PCA models trained on the residuals from the PF and EKF are more spread out over the PCs than the model trained on the measurements. In fact, the PCA model trained on the residuals has PCs with equivalent variance. This indicates that all the PCs are equally important for explaining the variation in the original data, thus, all the PCs should be retained in the PCA model. This highlights an important property of the state estimators in that they perform data whitening.

The residual, or innovation term, generated from the state estimators is often referred to as the 'whitening' step of the filter (Bar-Shalom et al., 2001). The measurement sequence from the process is non-white and at each point in time the measurements are often correlated with each other. By generating the innovation term, this performs data whitening on the measurement sequence to form the zero-mean white noise residual sequence, where the residuals are uncorrelated with each other (Reid, 2001).

Figure 20 also shows that the PCA model trained on the measurements has a larger percentage of the variance explained with fewer PCs. Based on a typical desired total variance explained of 70-80%, the data-driven method trained on the measurements only required 4 PCs to capture this variance. For both model-based methods trained on the residuals, 7 PCs are required to capture this same percentage of variance.

Using these trained PCA models, the three fault detection methods are evaluated on unseen testing data consisting of both nominal and faulty operating conditions for each of the four faults investigated. Figure 21 shows the $pAUC$ obtained using each of the appropriate monitoring statistics for each method and varying number of PCs retained in the PCA model.

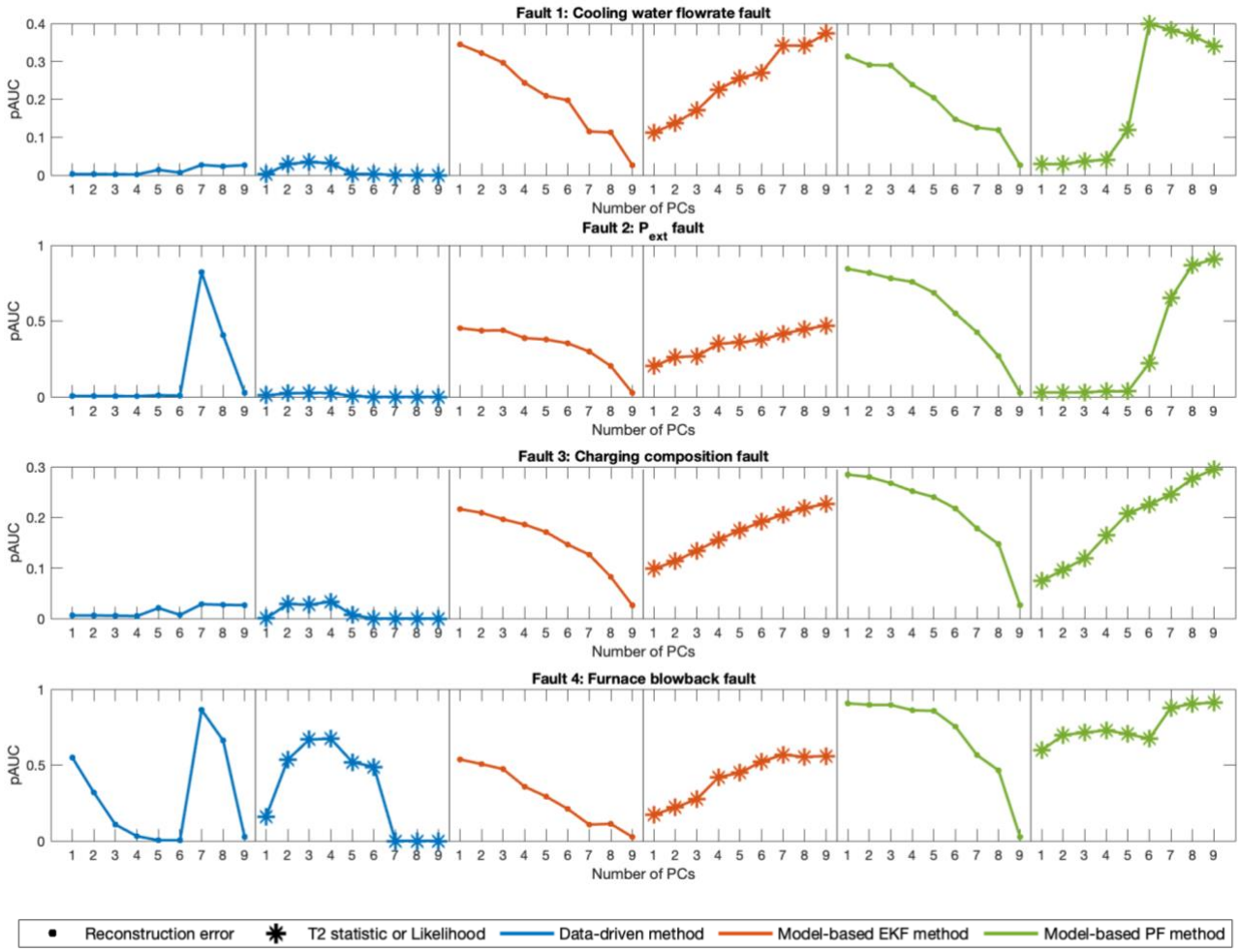

Figure 21: pAUC obtained at varying number of PCs retained for each test statistic for each fault investigated.

In blue lines in Figure 21 represent the $pAUC$ obtained by the data-driven method using the reconstruction error (points) and the T^2 statistic (asterisks). Based on Figure 21, the fault detection performance of the data-driven method is most notably poor for faults 1 and 3. For detection of faults 2 and 4, the data-driven method performs best using the reconstruction error and retaining 7 PCs. This highlights the importance of PCA as a fault detection technique as, without dimensionality reduction and retaining all 9 PCs, the performance of the data-driven technique is significantly worse than the performance whilst working in the reduced-dimensional space.

A common trend is seen in both model-based methods, where the highest $pAUC$ is obtained using the T^2 statistic of the EKF residual and the likelihood statistic of the PF residual for a maximum number of retained PCs. Apart from fault 1 likelihood statistic of the PF residual, where the highest $pAUC$ is achieved whilst retaining 6 PCs. In general, the model-based fault detection performs best using the likelihood or T^2 statistic as monitoring statistics in the original dimension space.

The goal of PCA is dimensionality reduction, whereby reconstruction of the original observations in a reduced dimension space is achieved with preferably minimal bias. PCA extracts the major sources of variation in the data. When data is subject to high noise to signal ratios, whereby noise variance is large compared to signal variance, this can cause significant bias in the PCA model (Spiegelberg & Rusz, 2017). This bias is due to the signal being spread out over several PCs.

PCA is employed as a residual evaluation technique in the model-based fault detection in an attempt to control the bias-variance trade-off. The residuals from a state estimator are independent of the process operating condition and, thus, under NOCs and no modelling error, the residuals are essentially only noise (Spiegelberg & Rusz, 2017). In the data-driven method, the measurement includes noise from the process, however, it also captures the dynamic movements of the system. Under NOC, the residual from the state estimators has a stronger noise-to-signal ratio than the measurements due to the large number of noisy directions. When the PCA model is built using the residuals, this causes high bias in the model, seen in Figure 20 as the variance being spread out over many PCs. This results in the trends seen in Figure 21, whereby the model-based methods perform best when retaining the maximum number of PCs, indicating important variation is contained in all the residuals.

With respect to the data-driven method, with the PCA model trained and tested on the measurements, the best fault detection performance occurs utilizing a PCA model retaining 7 PCs out of the maximum of 9 PCs. Therefore, the purely data-driven method also shows a tendency for better performance when retaining a relatively high number of PCs. This can be attributed to the fact that the system in question has limited measurements relative to the number of state variables with no repeated measured states. This implies that the measurements are not necessarily correlated. Therefore, retaining a high number of the PCs results in better fault detection performance.

6.2.2 Comparison of fault detection techniques

6.2.2.1 No plant-model mismatch

Figure 22 depicts the ROC curves generated for the three fault detection methods for each of the four faults investigated under no plant-model mismatch.

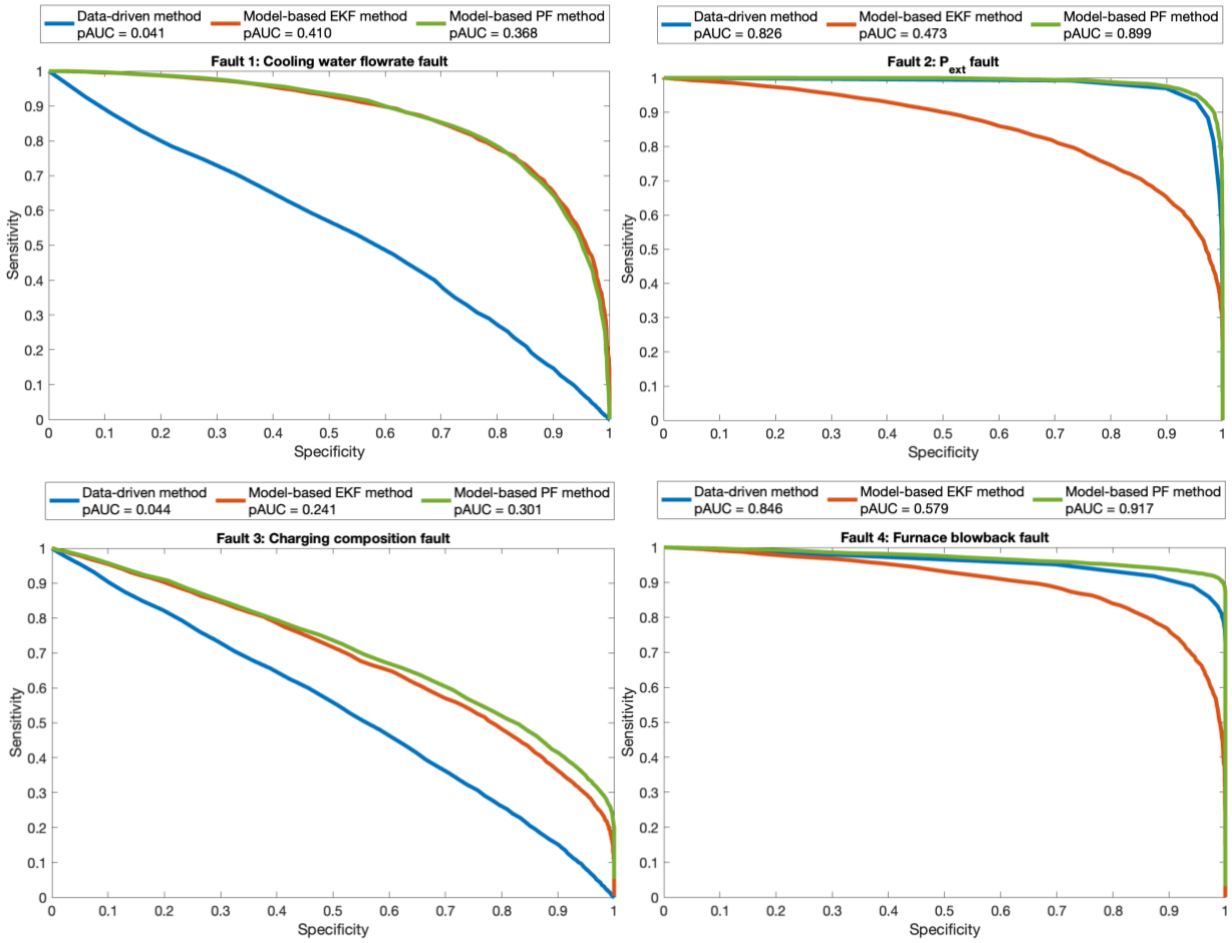

Figure 22: ROC curves generated for the data-driven method and the model-based methods using the EKF residuals and PF residuals for each of the four faults investigated.

The results show that the model-based method using the PF residuals consistently outperforms both the data-driven method and the model-based method using the EKF residuals for all four faults investigated. The model-based method using the EKF residuals outperforms the data-driven method for faults 1 and 3, where the data-driven method shows exceptionally poor performance.

In both model-based methods, the residuals only capture off-model movement of the system, heightening the sensitivity of the model-based fault detection to small changes in the process conditions. Small changes in the system caused by faults 1 and 3 clearly reflect as off-model movement within the residuals. In the measurements, the small changes in the system conditions arising from faults 1 and 3 are masked by dynamic movement of the system and the measurement noise. Thus, making the fault detection performance of the model-based methods superior to the data-driven method for detection of faults 1 and 3. When there is significant change in the process conditions, as in the case of faults 2 and 4,

the measurements deviate significantly, resulting in good fault detection performance of the data-driven technique.

Considering the performance of all three fault detection techniques, in general the highest $pAUCs$ are obtained for detection of fault 4 and the second highest for detection of fault 2. Fault 1 shows lower $pAUCs$ obtained for all three methods and fault 3 shows the lowest $pAUCs$, indicating the poorest fault detection performance. As expected, these results directly correlate with the findings of the structural detectability analysis in section 4.3.1 and the performance-based fault detectability in section 4.3.2. The more structurally detectable the fault, the better the fault detection performance. In addition, the larger the SNR of an observation, the better the fault detection performance.

Section 4.3.1 assessed the structural detectability of the SAF model used in the EKF and PF algorithms. This structural detectability is model-dependent. Thus, there may exist some other model structures of the SAF process which gives rise to higher detectability of the faults. The data-driven method of fault detection does not make use of the known process model, but instead learns a model based on historical data from the process. This model learnt by the data-driven method may have higher or lower structural fault detectability compared to the model used in the model-based methods. The results presented in Figure 22 indicate that the structural detectability of faults 1 and 3 are particularly low for the model learnt by the data driven method. Whilst faults 2 and 4 show similar structural detectability to the model used in the model-based fault detection.

6.2.2.2 Plant-model mismatch: Parametric uncertainty in k_V

The effect of parametric uncertainty in k_V on the performance of the model-based and the data-driven fault detection can be seen in Figure 23 and Figure 24 for each of the four faults investigated.

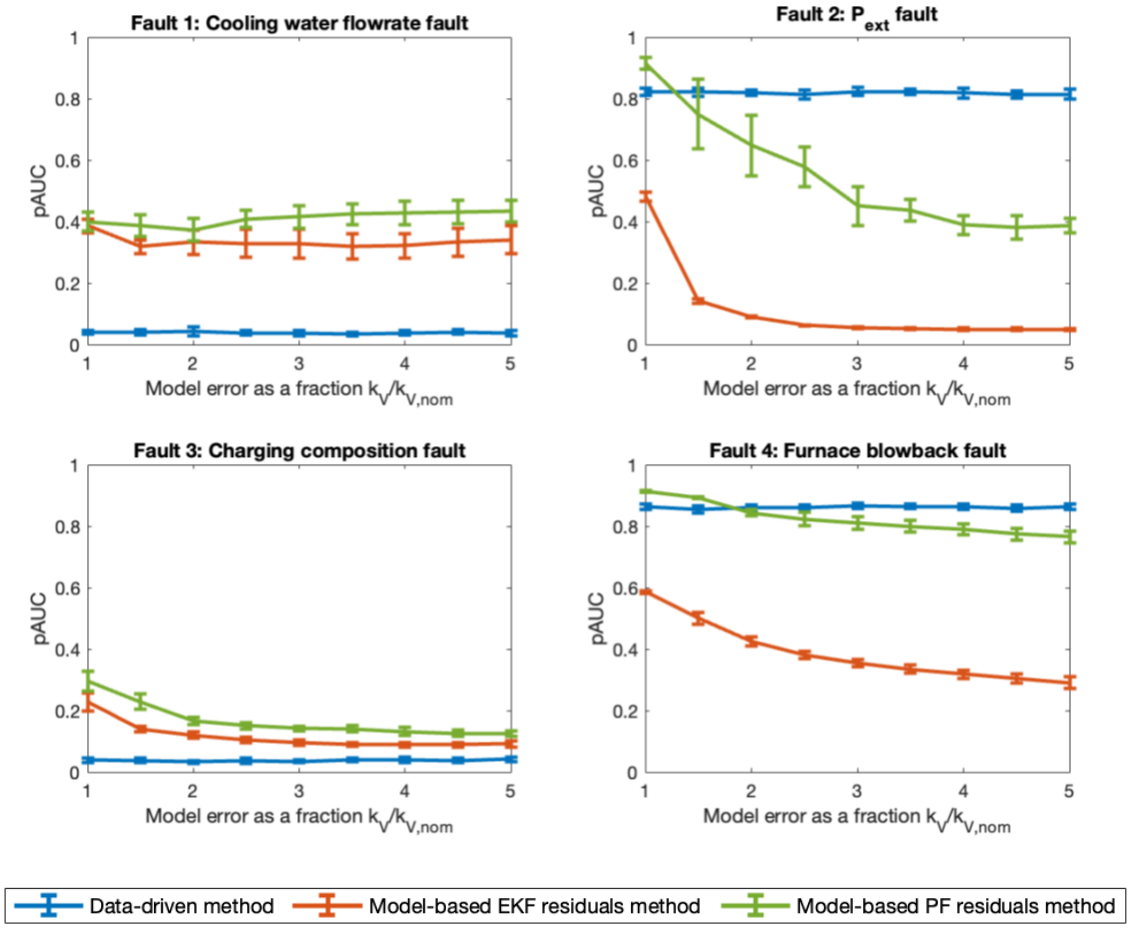

Figure 23: $pAUC$ at varied degrees of modelling error simulated via parametric uncertainty in the k_V parameter for the three fault detection methods and the four faults investigated.

Figure 23 shows the effect of increasing modelling error on the $pAUC$ obtained for each fault detection method. This assesses the fault detection performance at high specificities as only the partial area of the ROC curve is calculated. As expected, the data-driven method of fault detection remains unaffected by the parametric modelling error. The performance of both model-based methods significantly deteriorates as the modelling error increases, except in the case of fault 1 where the performance remains relatively constant even under high modelling error. Both model-based methods show similar behavior, displaying significant performance degradation at first, however, as the degree of modelling error increases, the $pAUC$ begins to plateau.

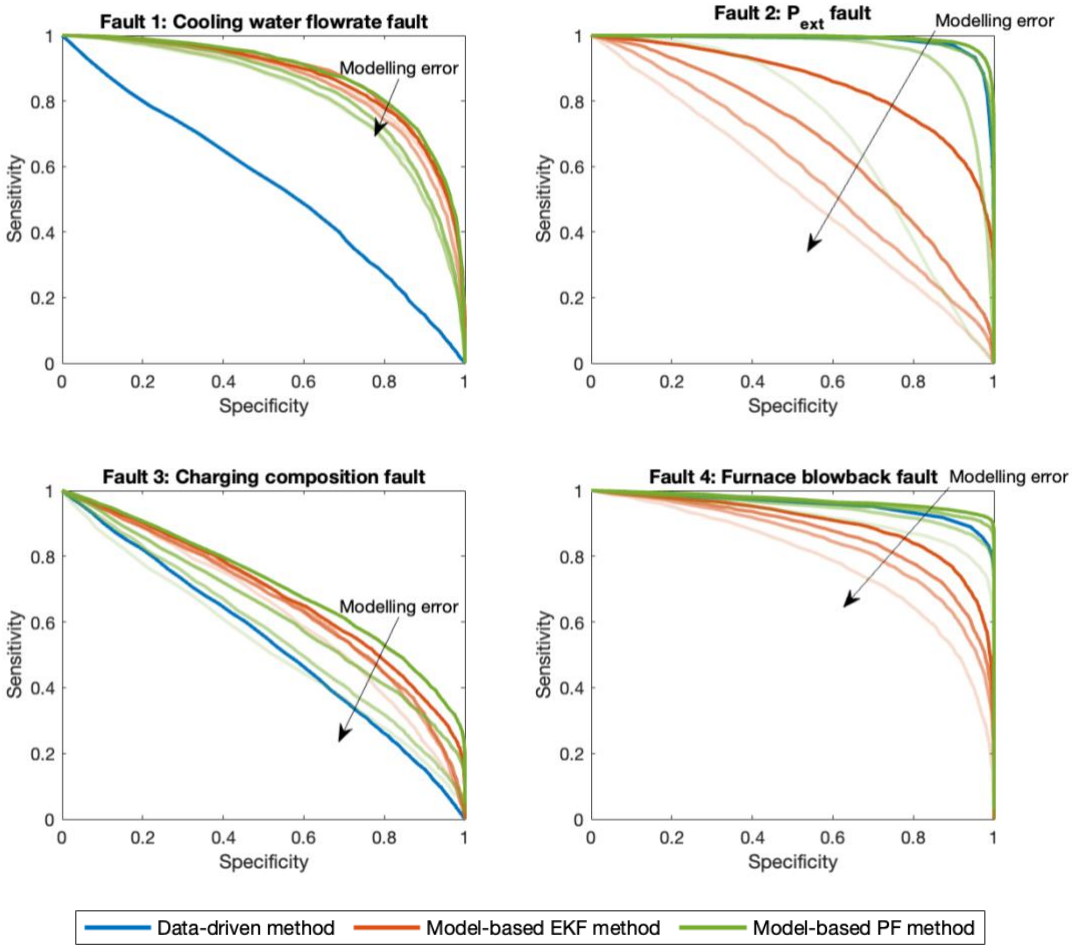

Figure 24: ROC curves generated using the EKF residuals, PF residuals, and measurements under varying degrees of modelling error.

Figure 24 shows the effect of modelling error on the entire ROC curve. Contrary to the results seen in Figure 23, the performance of the model-based techniques in detection of fault 1 is in fact degraded by modelling error. This cannot be observed from Figure 23 alone as the $pAUC$ only accounts for the area of high specificity in the ROC curve. Fault detection of fault 1 is most significantly affected at high sensitivities and low specificities, where the test statistic threshold is low. This is because the modelling error induces large residuals under fault-free conditions. The effects of fault 1 are mainly reflected in the T_w residual. This residual remains mostly unaffected by the modelling error, thus, good performance at high specificities is maintained even under large parametric uncertainty, resulting in the fairly unaffected $pAUC$ seen in Figure 23.

Model-based fault detection performance for the detection of faults 2, 3 and 4 is significantly impacted by modeling error as the residuals which reflect these faults are directly impacted by the error in k_V. For faults 3 and 4, the effect of modelling error can be seen at the flattening of the ROC curve in the low specificity regions caused by poor performance at low thresholds due to the large residuals generated under fault-free operation.

The performance degradation of the model-based method under modelling error is most notable for fault 2. This can be seen as the ROC curve flattening under both high and low thresholds. This is explained by analysis of Figure 11. Successful detection of fault 2 relies mostly on the $C_{G,R}$ residual signal generated by the EKF and the PF. For faults 3 and 4, the fault is reflected in a number of other residuals. Parametric uncertainty in k_V significantly affects the $C_{G,R}$ residual, thus, the faulty signal in the residual is masked by the modelling error, resulting in significant performance degradation in the detection of fault 2 seen in both Figure 23 and Figure 24.

Figure 24 further exposes another finding that cannot be deduced from Figure 23 alone. The performance degradation due to plant-model mismatch for the PF residuals is worse than the EKF residuals. This can be seen for detection of faults 1, 2, and 3, where the ROC curve generated from the PF residuals flattens more under modelling error than the ROC curve generated from the EKF residuals. Significant flattening of the curve at low thresholds is due to the PF having large residuals under fault-free conditions attributed to the filters lack of robustness to modelling error.

6.2.2.3 Effect of plant-model mismatch on the best performing number of PCs retained

Figure 25 presents the $pAUC$ obtained for each monitoring statistic for each number of PCs retained, under varying degrees of plant-model mismatch.

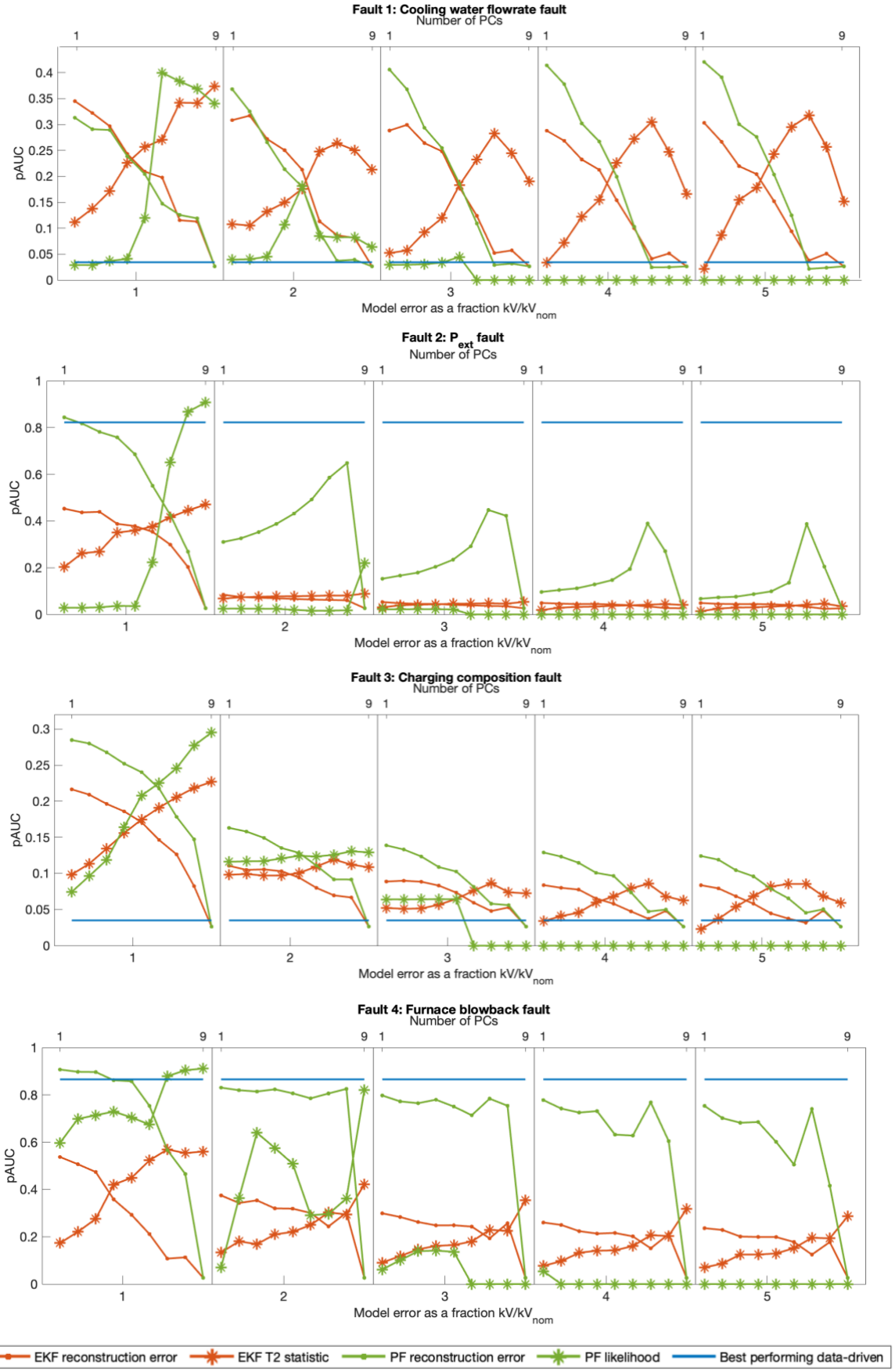

Figure 25: The pAUC obtained under varying degrees of modelling error by the test statistics for each number of retained PCs.

The blue line in Figure 25 represents the best performing data-driven method for each of the faults investigated, which remains unaffected by varying degrees of parametric uncertainty. Figure 25 shows that for detection of faults 1 and 3, the model-based methods, even under significant modelling error, consistently outperform the data-driven method. Although the model-based method using PF residuals likelihood statistic performance degrades considerably below the data-driven method under modelling error, the reconstruction error maintains good performance. For detection of faults 2 and 4, at any modelling error greater than $2\frac{k_V}{k_{V,nom}}$, the $pAUC$ obtained by both model-based methods drops below the $pAUC$ obtained by the data-driven method.

The orange line represents the model-based EKF method, showing the $pAUC$ obtained using the T^2 statistic (asterisks) and reconstruction error (points) under varying degrees of plant-model mismatch. In general, the T^2 monitoring statistic results in the highest $pAUC$ for this method. For detection of faults 1, 2, and 3, as the modelling error increases, the T^2 statistic begins to perform better whilst retaining less PCs than under no modelling error. The reconstruction error consistently performs best when retaining 1 PC under all degrees of modelling error.

The model-based fault detection using the PF residuals is represented by the green line in Figure 25, showing the $pAUC$ obtained using the likelihood statistic (asterisk) and reconstruction error (point) under varying degrees of plant-model mismatch. Under no modelling error, the likelihood statistic achieves the highest $pAUC$ in the detection of all four faults. Under modelling error, the likelihood statistic suffers the greatest performance degradation out of all the monitoring statistics. For all four faults investigated, as modelling error increases, the performance of the likelihood statistic over all possible number of retained PCs begins to flatten and the best performance occurs when less PCs are retained. As the likelihood suffers under increased modelling error, the reconstruction error becomes the best performing monitoring statistic for fault detection using the PF residuals. For detection of faults 1, 3, and 4, the highest $pAUC$ is obtained by the reconstruction error when 1 PC is retained under all degrees of modelling error. For detection of fault 2, the reconstruction error achieves the highest $pAUC$ when retaining 1 PC under no modelling error. However, under plant-model mismatch, the highest $pAUC$ is achieved whilst retaining $7-8$ PCs.

It can be concluded that, in general, when significant modelling error is expected, the T^2 statistic should be the statistical measure of choice for model-based fault detection using the EKF residuals and the number of PCs selected should explain at least 70% of the variance. For model-based fault detecting using the PF residuals, if large degrees of plant-model mismatch are expected then the reconstruction error should be used as a monitoring statistic in a reduced-dimension space retaining 1 PC.

6.2.2.4 Plant-model mismatch: stochastic variation in the inputs

Figure 26 highlights the effect of another type of modelling error, stochastic variation in the input F_{charge}, on the performance of the various fault detection algorithms.

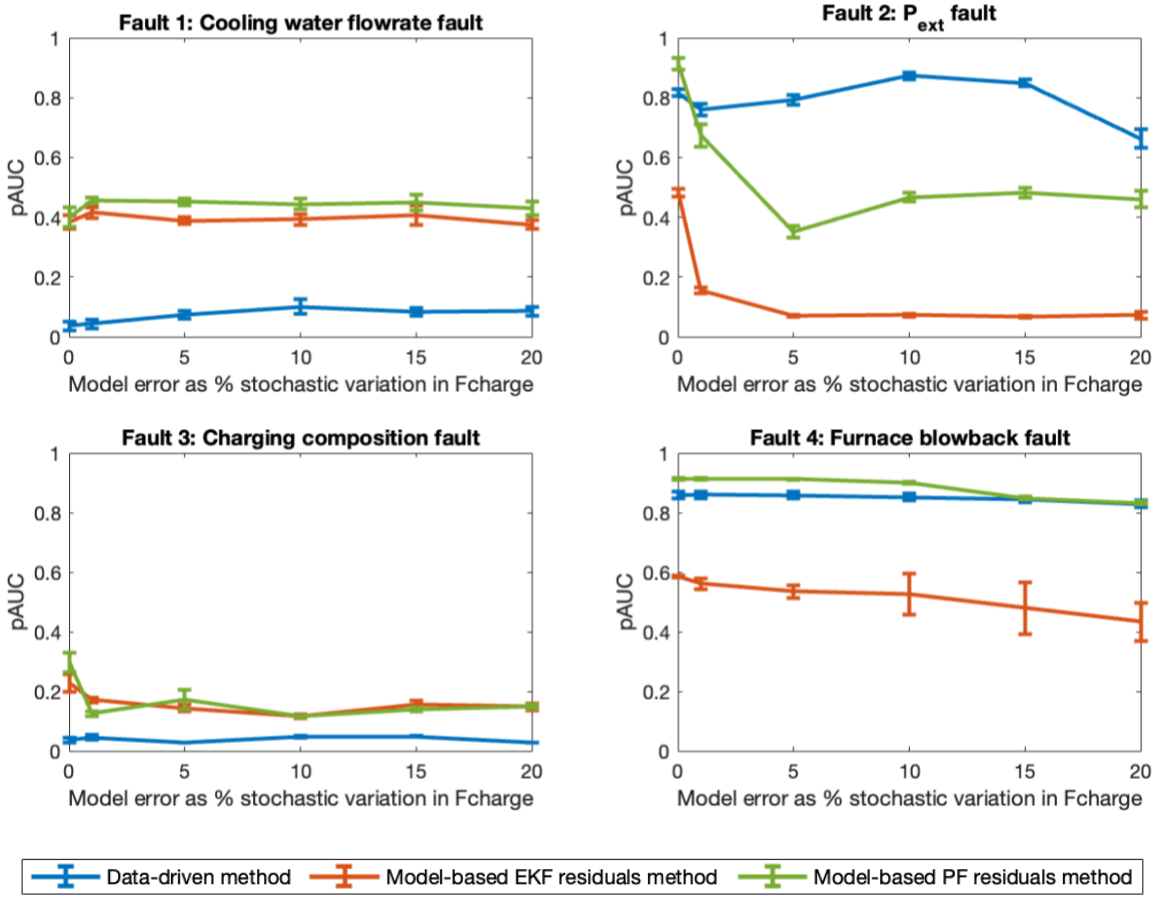

Figure 26: pAUC at varied degrees of modelling error simulated by stochastic variation in F_{charge} for the three fault detection methods and four faults investigated.

This type of plant-model mismatch not only influences the performance of model-based fault detection, but also impacts the data-driven fault detection as the measurements are now affected by stochastic variation in the inputs to the process. However, Figure 26 shows that the performance of the data-driven fault detection remains fairly unaffected in the presence of stochastic variation in the inputs, with the exception of fault 2, which shows an overall decrease in the $pAUC$ as stochastic variation increases. The model-based fault detection using both the PF and EKF residuals shows performance degradation under plant-model mismatch in the detection of fault 3, fault 4, and, most notably, fault 2.

6.2.2.5 Discussion of the results

Based on the background presented in sub-section 2.3.6, the two important metrics in the assessment of the performance of fault detection methods are the sensitivity and specificity. In model-based fault detection using state estimation, the classification performance is dictated by the ability of the residual to clearly reflect faulty process conditions as large residuals and nominal conditions as small residuals.

The accuracy of the state estimate plays an important role in achieving high specificity. The various state estimation techniques have different methods of handling the nonlinear approximation of the model prediction and measurement equations. For highly nonlinear models, the PF achieves more accurate state estimates than the EKF due to the superior accuracy of the nonlinear approximation. Accurate state

estimates ensure that the residuals remain small under nominal conditions and are not inflated due to any potential approximation error.

The sensitivity of the model-based fault detection is dictated by the sensitivity of the residual to any off-model movement caused by faulty conditions. Conversely, the residual should be insensitive to off-model movement caused by unknown disturbances and modelling error to prevent false alarms. The more robust a state estimation algorithm is to plant-model mismatch, the less sensitive the method is to faulty conditions and the poorer the fault detection performance.

Sub-sections 2.1.5.8 and 2.1.10 highlight the PFs known lack of robustness to plant-model mismatch compared to Kalman-based filters such as the EKF. The lack of robustness of the PF to modelling error is further amplified due to the small process noise covariance used to generate the residuals, exacerbating sample impoverishment in the PF. This lack of robustness of the PF algorithm to plant-model mismatch can be seen in the ROC curves in Figure 24. The model-based fault detection using the PF residuals shows worse performance at low thresholds than the EKF residuals due to the large PF residuals generated under nominal conditions.

Under no modeling error, the superior performance of the model-based fault detection using the PF residuals is attributed to this sensitivity resulting from the nature of the proposal distribution and the update-step of the PF algorithm. The bootstrap PF, used in this study, involves using the importance density as the proposal distribution in the prediction step and performing a resampling step at each iteration. The importance density is independent of the observations, thus, the bootstrap PF can often lead to loss of information in the observations (Theodoridis, 2020). Furthermore, the resampling step introduces significant sample impoverishment that is exacerbated by the small process noise covariance matrix. This leads to a loss of diversity among the particles. The culmination of these effects results in particles with observation information being assigned low weights and not resampled during the resampling step. Thus, the PF estimates track the model closely and do not carry significant information on the observations. Although this makes the PF less robust to modelling error, this ensures that large residuals are generated when there is any off-model movement caused by faulty conditions, making the PF more sensitive to faults and displaying superior fault detection performance. The Kalman-based update of the EKF ensures robustness against modelling error, however, this robustness comes at the cost of a reduced sensitivity to faulty conditions. This explains the consistently superior performance of the model-based fault detection using the PF residuals compared to the EKF residuals seen in Figure 21 through to Figure 26.

7 CONCLUSIONS AND RECOMMENDATIONS

The focus of this project was to perform model-based fault detection using state estimators and compare the performance of the model-based methods in the presence of plant-model mismatch to a data-driven method. Three objectives were defined in order to complete this aim. Objective 1 was to conduct an observability analysis to assess the state observability to validate state estimation and asses the fault detectability to inform the fault detection. Objective 2 was to employ the EKF, UKF, PF and MHE for state estimation and compare the performance of these nonlinear estimators to decide which are the most appropriate state estimators to use in model-based fault detection. Objective 3 was to carry out the model-based fault detection using the state estimators identified in objective 2, and compare the performance of the model-based fault detection under plant-model mismatch to a data-driven approach to fault detection. The case study used in this investigation is a SAF for PGM smelting.

An observability analysis on the SAF system was first conducted. State observability was assessed to validate the use of the state estimation algorithms which require a fully observable system. Fault detectability was evaluated to ascertain the structural detectability and performance-based detectability of each of the faults investigated in this study. The key findings from the observability analysis for the state observability and fault detectability are presented in sections 7.1 and 7.2, respectively.

State estimation was then employed using an EKF, UKF, PF and MHE to estimate the 17 state variables of the SAF model. The four nonlinear state estimation techniques were compared in terms of their computational requirements and estimation accuracy to determine the most appropriate techniques for application in model-based fault detection. Conclusions drawn from these results are presented in section 7.3.

Model-based fault detection was then performed using the residuals generated by the EKF and the PF and residual evaluation using PCA. The performance of these model-based methods were compared with the performance of a data-driven method using PCA on the measurements. Conclusions drawn from this comparative analysis are summarized in section 7.4. The performance of the model-based methods under plant-model mismatch was assessed and the key findings are presented in section 7.5.

Sections 7.6 and 7.7 provide recommendations for employing these nonlinear state estimation techniques in practice and recommended areas of interest for future work.

7.1 State observability

The SAF system model was found to be locally observable based on the observability analysis conducted on the nonlinear observability matrix linearized around typical operating conditions. This investigation found that the measured states of the system have a higher degree of observability than the unmeasured states of the system. However, due to the complex dynamic interactions between unmeasured state variables and measured state variables, the unmeasured states are all observable. This validated the use of the subsequent state estimation algorithms for estimating the states of the SAF system. In addition, the degree of observability of state variables informed the results of the subsequent state estimation,

whereby the less observable state variables had larger estimation errors associated with the state estimates.

7.2 Fault detectability

This investigation formalized the relationship between structural detectability of faults and parameter and input observability. Assessing the structural detectability of the faults gave an indication of the SAF models inherent ability to reflect fault parameters within the limited observables, irrespective of fault size, noise, and fault detection procedure. The four faults investigated in this study were all found to be structurally detectable by conducting a parameter observability analysis on the fault parameters used to simulate the faults of interest.

The performance-based detectability analysis used the SNR of fault observations to assess the detectability of each of the faults for each fault detection technique. The performance-based detectability further confirmed all four faults are detectable as each fault resulted in observation signals with sufficient SNR for detection. In addition, the residuals generated by the PF showed superior detectability seen by higher SNRs than the residuals generated by the EKF and the measurements from the process.

The results of the fault detectability analysis explained the results achieved upon implementing the model-based fault detection. The SAF models inherent ability to reflect faulty parameters, quantified by the structural detectability, directly corresponds to the performance results of the model-based fault detection. Superior fault detection performance resulted from more structurally detectable faults and less structurally detectable faults showed poorer fault detection performance of the methods investigated.

7.3 State estimation

By reviewing the relevant literature on nonlinear state estimation, the EKF is the preferred nonlinear state estimation algorithm due to its simplicity and ease of implementation. It is generally accepted that the MHE has limited applications due to the inappropriate computational requirements, exacerbated by long horizon lengths and the lack of a set of general heuristics to guide its implementation. This investigation corroborates the findings in literature, with the computational requirements of the MHE being orders of magnitude greater than the EKF, UKF, and PF. In an attempt to decrease the computational burden of the MHE, an alternative singular perturbation model of the SAF was supplied to the MHE to enable the use of larger step sizes in the integration step of the filter. This successfully halved the computational time per estimate at the cost of significant deterioration of the estimation accuracy. Although the computational requirements were halved, the computational effort of the MHE remains inappropriate for use in model-based fault detection.

As expected, the MHE using the original model demonstrated superior performance in terms of the estimation accuracy, followed by the UKF, the MHE using the singular perturbation model, and the EKF. The PF displayed the poorest estimation accuracy due to sensitivity of the algorithm in the presence of a relatively large process noise covariance supplied to the state estimation algorithms. In accordance with

the literature, all four state estimation techniques showed degraded estimation accuracy when the process noise covariance is large. The estimation accuracy improved significantly using a small process noise covariance matrix, where the PF showed superior estimation accuracy over both the UKF and the EKF. This is attributed to the PFs ability to handle nonlinear approximations more accurately than the EKF and UKF.

It was ultimately concluded that the EKF and the PF were the two most appropriate nonlinear state estimation techniques for application in model-based fault detection. The EKF was selected due to the popularity of the method as it was the easiest to implement, had the quickest computation time per estimate, and showed good estimation accuracy in estimating the states of the SAF. The PF was selected as it showed the best state estimation accuracy under small process noise used in model-based fault detection. Furthermore, the PF has the unique ability to handle non-gaussian distributions.

7.4 Model-based fault detection versus data-driven fault detection

This investigation found that the model-based fault detection using the PF residuals outperformed both the model-based fault detection using the EKF residuals and the data-driven fault detection using the measurements for all faults investigated. The superior performance of the PF compared to the EKF is attributed to the high sensitivity of PF residuals to faults.

The model-based methods tend to outperform the data-driven method for less detectable faults, namely the fault in the charging concentrate composition and the cooling water flowrate fault. These faults cause small deviations in the process conditions, which reflect as small changes in the measurements that are masked by measurement noise and dynamic movements of the process conditions. The residuals of the model-based fault detection only reflect off-model movement, thus, small changes in process conditions are easily detected as changes in the residuals. For the extraction pressure fault and furnace blowback, there are significant deviations in the measurements, thus, the data-driven method showed better fault detection performance. However, the model-based method using the PF residuals still outperformed the data-driven method for detection of these faults.

PCA models were developed for each method investigated. In general, the PCA model for the data-driven method should retain 7 PCs and use the reconstruction error as a test statistic for fault detection. The T^2 statistic should be used for the EKF residuals and the likelihood for the PF residuals. If there is no expected plant-model mismatch, the residuals show the best fault detection performance when all the PCs are retained, thus, indicating PCA should not be performed as part of the residual evaluation step.

7.5 Model-based fault detection performance under plant-model mismatch

This investigation showed that both model-based fault detection methods performance significantly deteriorates under plant-model mismatch simulated by both parametric uncertainty and unmodelled variation in the inputs. The PF displays poorer robustness to plant-model mismatch than the EKF, attributed to the higher sensitivity of the method to off-model movements of the system.

If plant-model mismatch is expected, the EKF and PF residuals should be evaluated in a reduced-dimensional space using PCA. The EKF residuals in the presence of modelling error showed the best fault detection performance using the T^2 statistic and retaining 7 PCs (78% of the variance explained). The PF residuals under modelling error showed the best fault detection performance using the reconstruction error and the 1 retained PC.

It was concluded that using PCA for residual evaluation in model-based fault detection enhances the performance of the model-based fault detection in the presence of plant-model mismatch. However, when there exists no modelling error, the best classification of the residuals occurs in the original-dimension space.

7.6 Recommendations for application of nonlinear state estimation techniques

The following recommendations are based on the key findings and conclusions of this investigation.

For state estimation of a nonlinear system, if the system is locally linearly observable, the EKF should be used for nonlinear state estimation due to the simplicity and reduced computational requirements of the method. If the system is not linearly observable, then the MHE must be employed to induce system observability via input excitation. The observability of the states gives insight to the estimation error covariance, or uncertainty, associated with the state estimates. Less observable states are expected to have a larger estimation error associated with the state estimates.

For practical implementation of the MHE, the computational burden of the filter will always be its greatest limitation. Thus, horizon length should be selected as the longest horizon length that maintains an appropriate computational time per estimate. For a stiff system of equations or system with time-scale multiplicity, using a singular perturbation model in the MHE significantly decreases the computational requirements of the algorithm. The singular perturbation model separates the slow and fast states, where the fast states are assumed constant in the prediction step to enable larger step sizes in the integration. This is corrected in the update step by assigning a large process noise to the fast states.

Lastly, when carrying out fault detection, when an accurate model of a process is available, model-based fault detection using state estimation should be employed over a basic data-driven fault detection technique using PCA. When significant plant-model mismatch is expected, the residual evaluation in the model-based fault detection should employ PCA, as working in a reduced-dimensional space has been proven to lessen the effects of modelling error. Prior to implementation of the fault detection algorithms, a fault detectability analysis should be carried out. Structural detectability of faults via a parameter observability analysis gives an idea of the process models inherent ability to reflect fault conditions in the outputs. Thus, the results of the structural fault detectability analysis can inform the fault detection.

7.7 Recommendations for future work

The following recommendations highlight potential future investigations based off the results of this study.

The model-based state estimation techniques and subsequent fault detection were applied to simulated data for this study. There would be great value in future work that extends these techniques to a process with an accurate model and industrial data to examine the real-world applicability of these methods.

For nonlinear state estimation, it is recommended that a future study investigate further reducing the computational requirements of the MHE to enable application of the MHE for model-based fault detection. Potential augmentations of the procedure include employing the singular perturbation model, but using the EKF to estimate the fast states and a MHE to estimate the slow states.

In this investigation, the model-based fault detection techniques were compared to a basic data-driven technique using linear PCA. It is recommended that the model-based techniques be compared to other more advanced data-driven techniques, such as nonlinear PCA via the kernel method or more advanced techniques such as using auto-encoders.

For the residual evaluation step, this study investigated projecting the residuals from the state estimators into a reduced-dimensional space defined by a PCA model. This study found that under no modelling error, the residuals are essentially white noise, thus, all PCs are important and PCA provides no benefits to the residual evaluation step. Future investigations could involve investigating other residual evaluation techniques such as independent component analysis, ICA.

The last recommendation for a future investigation would be to combine data-driven and model-based fault detection. The data-driven PCA runs on the measurements in parallel to model-based fault detection using the state estimation residuals. This ensures maintained fault detection performance in the presence of plant-model mismatch, as the data-driven method should detect faulty condition. When the model is accurate, the state estimator residuals should accurately detect faulty conditions.

8 REFERENCES

Abbeel, P. (2020). *Particle Filters*. https://people.eecs.berkeley.edu/~pabbeel/cs287-fa11/slides/particle-filters.pdf

Abdi, H., & Williams, L. J. (2010). Principal component analysis. In *Wiley Interdisciplinary Reviews: Computational Statistics* (Vol. 2, Issue 4, pp. 433–459). https://doi.org/10.1002/wics.101

Addo, P. (2019). *Adaptive Process Monitoring using Principal Component Analysis and Gaussian Mixture Models*. https://scholar.sun.ac.za

Alaeddini, A., & Morgansen, K. A. (2013). Autonomous state estimation using optic flow sensing. *Proceedings of the American Control Conference*, 585–590. https://doi.org/10.1109/acc.2013.6579900

Alexander, R., Campani, G., Dinh, S., & Lima, F. V. (2020). Challenges and opportunities on nonlinear state estimation of chemical and biochemical processes. In *Processes* (Vol. 8, Issue 11, pp. 1–27). MDPI AG. https://doi.org/10.3390/pr8111462

Al-Matouq, A. A., & Vincent, T. L. (2015). Multiple window moving horizon estimation. *Automatica*, *53*, 264–274. https://doi.org/10.1016/j.automatica.2014.12.002

Alrowaie, F., Gopaluni, R. B., & Kwok, K. E. (2012). Fault detection and isolation in stochastic non-linear state-space models using particle filters. *Control Engineering Practice*, *20*(10), 1016–1032. https://doi.org/10.1016/j.conengprac.2012.05.008

Anguelova, M. (2004). *Thesis for the degree of licentiate of engineering Nonlinear Observability and Identifiability: General Theory and a Case Study of a Kinetic Model for S. cerevisiae*. Chalmers University of technology and Göteborg University.

Arulampalam, S., & Ristic, B. (2000). Comparison of the particle filter with range-parameterized and modified polar EKFs for angle-only tracking. *Signal and Data Processing of Small Targets 2000*, *4048*, 288–299. https://doi.org/10.1117/12.391985

Bagheri, R. (2020, January 9). *Understanding Singular Value Decomposition and its Application in Data Science*. Towards Data Science. https://towardsdatascience.com/understanding-singular-value-decomposition-and-its-application-in-data-science-388a54be95d

Bar-Shalom, Y., Li, X.-R., & Kirubarajan, T. (2001). *Estimation with applications to tracking and navigation*. Wiley.

Basseville, M. (2001). On fault detectability and isolability. *European Journal of Control*, *7*(6), 625–637. https://doi.org/10.3166/ejc.7.625-637

Bavdekar, V. A., Gopaluni, R. B., & Shah, S. L. (2013). Evaluation of adaptive extended Kalman filter algorithms for state estimation in presence of model-plant mismatch. *IFAC Proceedings Volumes (IFAC-PapersOnline)*, *10*(PART 1), 184–189. https://doi.org/10.3182/20131218-3-IN-2045.00175

Becker, A. (2023). *Kalman filter from the ground up* (1st ed., Vol. 1).

Berkeley Math. (2017). *Notes on singular value decomposition for Math 54*. https://math.berkeley.edu/~hutching/teach/54-2017/svd-notes.pdf

Bernard, P., Andrieu, V., & Astolfi, D. (2022). Observer design for continuous-time dynamical systems. *Annual Reviews in Control*, *53*, 224–248. https://doi.org/10.1016/j.arcontrol.2021.11.002

Bezuidenhout, J. J., Eksteen, J. J., & Bradshaw, S. M. (2009). Computational fluid dynamic modelling of an electric furnace used in the smelting of PGM containing concentrates. *Minerals Engineering*, *22*(11), 995–1006. https://doi.org/10.1016/j.mineng.2009.03.009

Bickel, P., Diggle, P., Fienberg, S., Krickeberg, K., Oikin, I., Wermuth, N., & Zeger, S. (1996). Lecture Notes in Statistics. In *Smoothness Priors Analysis of Time Series* (Vol. 16). Springer. https://doi.org/10.1007/978-1-4612-0761-0

Bolić, M., Hong, S., & Djurić, P. M. (2002). Performance and complexity analysis of adaptive particle filtering for tracking applications. *Conference Record of the Asilomar Conference on Signals, Systems and Computers*, *1*, 853–857. https://doi.org/10.1109/acssc.2002.1197299

Brockwell, P. J., & Davis, R. A. (2016). *Stationary Processes. In: Introduction to Time Series and Forecasting*. Springer, Cham.

Brownlee, J. (2019, September 27). *A Gentle Introduction to Joint, Marginal, and Conditional Probability*. Machine Learning Mastery. https://machinelearningmastery.com/joint-marginal-and-conditional-probability-for-machine-learning/

Bucy, R. S., & Senne, K. D. (1971). Digital Synthesis of Non-linear Filters Synth. *Automatica*, *7*, 287–298.

Campani, G., Ribeiro, M. P. A., Zangirolami, T. C., & Lima, F. V. (2019). A hierarchical state estimation and control framework for monitoring and dissolved oxygen regulation in bioprocesses. *Bioprocess and Biosystems Engineering*, *42*(9), 1467–1481. https://doi.org/10.1007/s00449-019-02143-4

Castellanos, J. A., Neira, J., & Tardós, J. D. (2004). *Limits to the Consistency of EKF-based SLAM*.

Chang, C.-T., & Chen, J.-W. (1995). Implementation Issues Concerning the EKF-Based Fault Diagnosis Techniques. *Chemical Engineering Science*, *50*(18), 2861–2882.

Chatzi, E. N., & Smyth, A. W. (2002). The Unscented Kalman Filter and Particle Filter Methods for Nonlinear Structural System Identification with Non-Collocated Heterogeneous Sensing ‡. In *Control* (Vol. 00).

Chen, T., Morris, J., & Martin, E. (2004). Particle Filters for the Estimation of a State Space Model. In *Computer Aided Chemical Engineering* (Vol. 18, pp. 613–618).

Chen, T., Morris, J., & Martin, E. (2005). Particle filters for state and parameter estimation in batch processes. *Journal of Process Control*, *15*(6), 665–673. https://doi.org/10.1016/j.jprocont.2005.01.001

Chen, X., Heidarinejad, M., Liu, J., De La Peña, D. M., & Christofides, P. D. (2011). Model predictive control of nonlinear singularly perturbed systems: Application to a large-scale process network. *Journal of Process Control*, *21*(9), 1296–1305. https://doi.org/10.1016/j.jprocont.2011.07.004

Chen, Z. (1991). Local Observability and Its Application to Multiple Measurement Estimation. *IEEE transactions on industrial electronics*, *38*(6).

Chetouani, Y. (2004). Fault detection by using the innovation signal: Application to an exothermic reaction. *Chemical Engineering and Processing: Process Intensification*, *43*(12), 1579–1585. https://doi.org/10.1016/j.cep.2004.02.002

Chetouani, Y. (2008a). Design of a multi-model observer-based estimator for fault detection and isolation (FDI) strategy: application to a chemical reactor. *Brazilian Journal of Chemical Engineering*, *25*(04), 777–788. www.abeq.org.br/bjche

Chetouani, Y. (2008b). Using the Kalman Filtering for the Fault Detection and Isolation (FDI) in the Nonlinear Dynamic Processes Using the Kalman Filtering for the Fault Detection and Isolation (FDI) in the Nonlinear Dynamic Processes. *International Journal of Chemical Reactor Engineering* , *6*.

Cornejo, N. E., Amini, R., & Gaydadjiev, G. (2010). Model-based fault detection for the DELFI-N3XT attitude determination system. *IEEE Aerospace Conference Proceedings*. https://doi.org/10.1109/AERO.2010.5446801

Crundwell, F. K., Moats, M. S., Ramachandran, V., Robinson, T. G., & Davenport, W. G. (2011). Smelting and Converting of Sulfide Concentrates Containing Platinum-Group Metals. In *Extractive Metallurgy of Nickel, Cobalt and Platinum Group Metals* (pp. 437–456). Elsevier. https://doi.org/10.1016/b978-0-08-096809-4.10035-8

Dahleh, M., Dahleh, M. A., & Verghese, G. (2011). *Lectures on Dynamic Systems and Control*.

Das, M., Sadhu, S., & Ghoshal, T. K. (2014). Fault detection and isolation using an adaptive unscented Kalman filter. *IFAC Proceedings Volumes (IFAC-PapersOnline)*, *3*(PART 1), 326–332. https://doi.org/10.3182/20140313-3-IN-3024.00075

Daum, F. (2005). Nonlinear Filters: Beyond the Kalman Filter. *IEEE A&E Systems Magazine, 20*(8).

De Santis, E., & Di Benedetto, M. D. (2017). Observability and diagnosability of finite state systems: A unifying framework. *Automatica*, *81*, 115–122. https://doi.org/10.1016/j.automatica.2017.02.042

Debnath, S., Sahoo, S. R., Decardi-Nelson, B., & Liu, J. (2021). *Subsystem decomposition and state estimation of nonlinear processes with implicit time-scale multiplicity*. http://arxiv.org/abs/2110.00618

Devos, T., Kirchner, M., Croes, J., Desmet, W., & Naets, F. (2021). Sensor selection and state estimation for unobservable and non-linear system models. *Sensors*, *21*(22). https://doi.org/10.3390/s21227492

Diaz, J. L., Ocampo-Martinez, C., & Alvarez, H. (2017). Nonlinear moving horizon estimator for online estimation of the density and viscosity of a mineral slurry. *Ind. Eng. Chem. Res.*, *56*(49), 14592–14603.

Ding, S. X. (2008). Model-based fault diagnosis techniques: Design schemes, algorithms, and tools. In *Model-based Fault Diagnosis Techniques: Design Schemes, Algorithms, and Tools* (1st ed.). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-76304-8

Dodd, L. E., & Pepe, M. S. (2003). Partial AUC Estimation and Regression. In *Biometrics* (Vol. 59).

Doucet, A., & Johansen, A. M. (2008). *A Tutorial on Particle Filtering and Smoothing: Fifteen years later*.

Dubois, R., Bertrand, S., & Eudes, A. (2018). *Performance Evaluation of a Moving Horizon Estimator for Multi-Rate Sensor Fusion with Time-Delayed Measurements*. https://hal.archives-ouvertes.fr/hal-01925336

Duerinckx, A., Hamdi, R., Mahfouf, J. F., & Termonia, P. (2015). Study of the Jacobian of an extended Kalman filter for soil analysis in SURFEXv5. *Geoscientific Model Development*, *8*(3), 845–863. https://doi.org/10.5194/gmd-8-845-2015

Dunn, K. (2020). *Process Improvement Using Data Release* (76ba30 ed.). https://learnche.org/pid/.

Ebeigbe, D., Berry, T., Norton, M. M., Whalen, A. J., Simon, D., Sauer, T., & Schiff, S. J. (2021). *A Generalized Unscented Transformation for Probability Distributions*. http://arxiv.org/abs/2104.01958

Eksteen, J. J. (2011). A mechanistic model to predict matte temperatures during the smelting of UG2-rich blends of platinum group metal concentrates. *Minerals Engineering*, *24*(7), 676–687. https://doi.org/10.1016/j.mineng.2010.10.017

Elfring, J., Torta, E., & van de Molengraft, R. (2021). Particle filters: A hands-on tutorial. *Sensors (Switzerland)*, *21*(2), 1–28. https://doi.org/10.3390/s21020438

Elsheikh, M., Hille, R., Tatulea-Codrean, A., & Krämer, S. (2021). A comparative review of multi-rate moving horizon estimation schemes for bioprocess applications. In *Computers and Chemical Engineering* (Vol. 146). Elsevier Ltd. https://doi.org/10.1016/j.compchemeng.2020.107219

Ferhatbegovic, T., Zucker, G., & Palensky, P. (2012). An unscented Kalman filter approach for the plant-model mismatch reduction in HVAC system model based control. *IECON Proceedings (Industrial Electronics Conference)*, 2180–2185. https://doi.org/10.1109/IECON.2012.6388685

Fox, D. (2003). KLD-Sampling: Adaptive Particle Filters. *Advances in Neural Information Processing Systems: Natural and Synthetic.*

Frank, P. M. (1990). Fault Diagnosis in Dynamic Systems Using Analytical and Knowledge-based Redundancy A Survey and Some New Results. *Automatica*, *26*(3), 459–474.

Gábor, A., Villaverde, A. F., & Banga, J. R. (2017). Parameter identifiability analysis and visualization in large-scale kinetic models of biosystems. *BMC Systems Biology*, *11*(1). https://doi.org/10.1186/s12918-017-0428-y

Garg, M., Swarup, A., & Kanth, A. S. V. R. (2016). Application of singular perturbation theory in modeling and control of flexible robot arm. *International Journal of Advanced Technology and Engineering Exploration*, *3*(24), 176–181. https://doi.org/10.19101/ijatee.2016.324002

Gautam, S., Tamboli, P. K., Roy, K., Patankar, V. H., & Duttagupta, S. P. (2019). Sensors incipient fault detection and isolation of nuclear power plant using extended Kalman filter and Kullback–Leibler divergence. *ISA Transactions*, *92*, 180–190. https://doi.org/10.1016/j.isatra.2019.02.011

Geetha, M., Kumar, P. A., & Jerome, J. (2014). Comparative Assessment of a Chemical Reactor Using Extended Kalman Filter and Unscented Kalman Filter. *Procedia Technology*, *14*, 75–84. https://doi.org/10.1016/j.protcy.2014.08.011

Ghobara, Y. E. M. (2013a). *Modeling, Optimization and Estimation in Electric Arc Furnace (EAF) Operation*. McMaster University.

Ghobara, Y. E. M. (2013b). *Modeling, Optimization and Estimation in Electric Arc Furnace (EAF) Operation*. McMaster University.

Groenewald, J. W. D., Nelson, L. R., Hundermark, R. J., Phage, K., Sakaran, R. L., Van Rooyen, Q., & Cizek, A. (2018). Furnace integrity monitoring using principal component analysis: An industrial case study. *Journal of the Southern African Institute of Mining and Metallurgy*, *118*(4), 345–352. https://doi.org/10.17159/2411-9717/2018/v118n4a3

Gurajala, R., Choppala, P. B., Meka, J. S., & Teal, P. D. (2021). Derivation of the Kalman filter in a Bayesian filtering perspective. *2nd International Conference on Range Technology (ICORT)*.

Ham, F. M., Corporation, H., & Grover Brown, R. (1983). Observability, Eigenvalues, and Kalman Filtering. In *IEEE Transactions on Aerospace and Electronic* (Issue 2).

Haseltine, E. L., & Rawlings, J. B. (2003). A Critical Evaluation of Extended Kalman Filtering and Moving Horizon Estimation. In *TWMCC Texas-Wisconsin Modeling and Control Consortium* (Vol. 1).

Hedrick, J. K., & Girard, A. (2005). *Control of Nonlinear Dynamic Systems: Theory and Applications 6 `Controllability and Observability of Nonlinear Systems Controllability for Nonlinear Systems*.

Hsoumi, A., El Harabi, R., Bel Hadj Ali, S., & Abdelkrim, M. N. (2009). Diagnosis of a continuous stirred tank reactor using Kalman filter. *CSSim 2009 - 1st International Conference on Computational Intelligence, Modelling, and Simulation*, 153–158. https://doi.org/10.1109/CSSim.2009.22

Huang, P., Meyr, H., Dörpinghaus, M., & Fettweis, G. (2020). Observability Analysis of Flight State Estimation for UAVs and Experimental Validation. *IEEE International Conference on Robotics and Automation (ICRA)*, 4659–4665.

Imtiaz, S. A., Roy, K., Huang, B., Shah, S. L., & Jampana, P. (2006). Estimation of states of nonlinear systems using a particle filter. *Proceedings of the IEEE International Conference on Industrial Technology*, 2432–2437. https://doi.org/10.1109/ICIT.2006.372687

Jagadeesan, P., Mcauley, K., Prasad, V., Shah, S., Shenoy, A. V, Prakash, J., Mcauley, K. B., Prasad, V., & Shah, S. L. (2011). *Practical issues in the application of the particle filter for estimation of chemical processes*. https://www.researchgate.net/publication/266066353

James, M. R. (1987). *Controllability and Observability of Nonlinear Systems*. University of Maryland.

Jayaprasanth, D., & Kanthalakshmi, S. (2018). Fault Detection and Isolation in Stochastic Nonlinear Systems using Unscented Particle Filter based Likelihood Ratio Approach. In *CEAI* (Vol. 20, Issue 1).

Jones, R. T. (2005). An overview of Southern African PGM smelting. *Nickel and Cobalt 2005: Challenges in Extraction and Production*, 147–178.

Julier, S. J., & Uhlmann, J. K. (1997). A New Extension of the Kalman Filter to Nonlinear Systems. *The 11th International Symposium of Aerospace/Defense Sensing, Simulation and Controls, Multi Sensor Fusion, Tracking and Resource Management II*, 182–193.

Julier, S. J., & Uhlmann, J. K. (2004). Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, *92*(3), 401–422. https://doi.org/10.1109/JPROC.2003.823141

Jumaa, H., Fayn, J., & Rubel, P. (2010). QR-Duality Tuning of Standard Kalman Filter oriented to Rocket Velocity Indirect Measurement. *2010 12th International Conference on Computer Modelling and Simulation*, 509–514. https://doi.org/10.1109/UKSIM.2010.99

Jung, D., & Sundstrom, C. (2019). A Combined Data-Driven and Model-Based Residual Selection Algorithm for Fault Detection and Isolation. *IEEE Transactions on Control Systems Technology*, *27*(2), 616–630. https://doi.org/10.1109/TCST.2017.2773514

Jurafsky, D., & Martin, J. H. (2021, December 29). *Hidden Markov Models- Speech and Language Processing*. Stanford Education. https://web.stanford.edu/~jurafsky/slp3/A.pdf

Kaced, R., Kouadri, A., Baiche, K., & Bensmail, A. (2021). Multivariate nuisance alarm management in chemical processes. *Journal of Loss Prevention in the Process Industries*, *72*. https://doi.org/10.1016/j.jlp.2021.104548

Kadirkamanathan, V., Li, P., Jaward, M. H., & Fabri, S. G. (2002). Particle filtering-based fault detection in non-linear stochastic systems. *International Journal of Systems Science*, *33*(4), 259–265. https://doi.org/10.1080/00207720110102566

Kadirkamanathan, V., Li, P., & Kirubarajan, T. (2001, January). Sequential Monte Carlo filtering vs. the IMM estimator for fault detection and isolation in nonlinear systems. *Proceedings of SPIE - The International Society for Optical Engineering*. http://spiedl.org/terms

Khorasgani, H., Farahat, A., Ristovski, K., Gupta, C., & Biswas, G. (2018a). A Framework for Unifying Model-based and Data-driven Fault Diagnosis. *Annual Conference of the Prognostics and Health Management Society*.

Khorasgani, H., Farahat, A., Ristovski, K., Gupta, C., & Biswas, G. (2018b). A Framework for Unifying Model-based and Data-driven Fault Diagnosis. *A Combined Diagnosis System Design Using Model-Based and Data-Driven Methods.*

Khorasgani, H., Jung, D. E., Biswas, G., Frisk, E., & Krysander, M. (2014). Robust residual selection for fault detection. *Proceedings of the IEEE Conference on Decision and Control*, *2015-February*(February), 5764–5769. https://doi.org/10.1109/CDC.2014.7040291

Kopper, A., Karkare, R., Paffenroth, R. C., & Apelian, D. (2020). Model Selection and Evaluation for Machine Learning: Deep Learning in Materials Processing. *Integrating Materials and Manufacturing Innovation*, *9*(3), 287–300. https://doi.org/10.1007/s40192-020-00185-1

Kourti, T. (2002). Process analysis and abnormal situation detection from theory to practice. *IEEE Control Systems Magazine*, *22*(5), 10–25.

Ku, W., Storer, R. H., & Georgakis, C. (1995). Disturbance detection and isolation by dynamic principal component analysis. *Chemometrics and Intelligent Laboratory Systems* , *30*, 179–196.

Kumar, A. S., & Ahmad, Z. (2012). Model Predictive Control (MPC) and Its Current Issues in Chemical Engineering. *Chemical Engineering Communications*, *199*(4), 472–511. https://doi.org/10.1080/00986445.2011.592446

Lambert, J. (2018, December 27). *What is State Estimation and the Bayes Filter?* Github. https://johnwlambert.github.io/bayes-filter/

Larsson, T. (2015). *Moving Horizon Estimation for JModelica.org*.

Laviola, J. J. (2003). A Comparison of Unscented and Extended Kalrnan Filtering for Estimating Quaternion Motion. *Proceedings of the American Control Conference*.

Li, K., Tan, H.-S., & Hedrick, J. K. (2009, December 16). Map-Aided GPS/INS Localization Using a Low-Order Constrained Unscented Kalman Filter. *48th IEEE Conference on Decision and Control and 28th Chinese Control Conference*.

Li, P., & Kadirkamanathan, V. (2001). Particle Filtering Based Likelihood Ratio Approach to Fault Diagnosis in Nonlinear Stochastic Systems. *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, *31*(3).

Li, R., & Olson, J. H. (1991). Fault Detection and Diagnosis in a Closed-Loop Nonlinear Distillation Process: Application of Extended Kalman Filters. *Ind. Eng. Chem. Res*, *30*, 898–908.

Li, T., Sun, S., Sattar, T. P., & Corchado, J. M. (2014). Fight sample degeneracy and impoverishment in particle filters: A review of intelligent approaches. *Expert Systems with Applications*, *41*(8), 3944–3954. https://doi.org/10.1016/j.eswa.2013.12.031

Logar, V., Dovžan, D., & Škrjanc, I. (2012). Modeling and Validation of an Electric Arc Furnace: Part 1, Heat and Mass Transfer. *ISIJ International*, *52*(3), 402–412.

Louédec, M., & Jaulin, L. (2021). Interval extended Kalman filter—application to underwater localization and control. *Algorithms*, *14*(5). https://doi.org/10.3390/a14050142

Mangold, M., Bück, A., Schenkendorf, R., Steyer, C., Voigt, A., & Sundmacher, K. (2009). Two state estimators for the barium sulfate precipitation in a semi-batch reactor. *Chemical Engineering Science*, *64*(4), 646–660. https://doi.org/10.1016/j.ces.2008.05.039

Martinelli, A. (2011). State estimation based on the concept of continuous symmetry and observability analysis: The case of calibration. *IEEE Transactions on Robotics*, *27*(2), 239–255. https://doi.org/10.1109/TRO.2011.2109210

Martínez, N., & Villaverde, A. F. (2020). Nonlinear observability algorithms with known and unknown inputs: Analysis and implementation. *Mathematics*, *8*(11), 1–27. https://doi.org/10.3390/math8111876

Martinez-Cantin, R., & Castellanos, J. A. (2005). Unscented SLAM for large-scale outdoor environments. *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 3427–3432. https://doi.org/10.1109/IROS.2005.1545002

Mehra, R. K., & Peschon, J. (1971). An Innovations Approach to Fault Detection and Diagnosis in Dynamic Systems. *Automatica*, *7*, 637–640.

Mesbah, A., Huesman, A. E. M., Kramer, H. J. M., & Van Den Hof, P. M. J. (2011). A comparison of nonlinear observers for output feedback model-based control of seeded batch crystallization processes. *Journal of Process Control*, *21*(4), 652–666. https://doi.org/10.1016/j.jprocont.2010.11.013

Mohan, S., Naik, N., Gemson, R. M. O., & Ananthasayanam, M. R. (2015). *Introduction to the Kalman Filter and Tuning its Statistics for Near Optimal Estimates and Cramer Rao Bound*. http://arxiv.org/abs/1503.04313

Mujica, L. E., Rodellar, J., Fernández, A., & Güemes, A. (2011). Q-statistic and t2-statistic pca-based measures for damage assessment in structures. *Structural Health Monitoring*, *10*(5), 539–553. https://doi.org/10.1177/1475921710388972

Nakhaeinejad, M., & Bryant, M. D. (2011). Observability analysis for model-based fault detection and sensor selection in induction motors. *Measurement Science and Technology*, *22*(7). https://doi.org/10.1088/0957-0233/22/7/075202

Nasir, A. (2020, April 26). *Controllability and Observability of Nonlinear Systems Part II*. https://www.youtube.com/watch?v=Lqt4MnLrZP4

Nell, J. (2004). Smelting of platinum group metal concentrates in South Africa. *The Journal of The South African Institute of Mining and Metallurgy*.

Owen, J.-A., & Demirkiran, I. (2014). Principal Component Analysis: Data Reduction and Simplification. *McNair Scholars Research Journal*, *1*(2).

Pan, Y., Sun, S., & Jahanshahi, S. (2011). Mathematical modelling of heat transfer in six-in-line electric furnaces for sulphide smelting. *The Journal of The Southern African Institute of Mining and Metallurgy*, *11*, 717–732.

Paradowski, T. (2021, February 12). *Lectures on Dynamic Systems and Control*. Bergische University. https://cdn.website-editor.net/cef40558e07d4eef8c29b78a5945738a/files/uploaded/seminar_11_Paradowski.pdf

Pardal, P. C. P. M., Kuga, H. K., & De Moraes, R. V. (2015). The Particle Filter Sample Impoverishment Problem in the Orbit Determination Application. *Mathematical Problems in Engineering*, *2015*. https://doi.org/10.1155/2015/168045

Patton, R. J., Frank, P. M., & Clark, R. N. (2000). *Issues of Fault Diagnosis for Dynamic Systems* (1st ed.). Springer London. https://doi.org/10.1007/978-1-4471-3644-6

Patwardhan, S. C., Narasimhan, S., Jagadeesan, P., Gopaluni, B., & L. Shah, S. (2012). Nonlinear Bayesian state estimation: A review of recent developments. In *Control Engineering Practice* (Vol. 20, Issue 10, pp. 933–953). https://doi.org/10.1016/j.conengprac.2012.04.003

Peng, X., Tang, Y., Du, W., & Qian, F. (2017). Multimode process monitoring and fault detection: A sparse modeling and dictionary learning method. *IEEE Transactions on Industrial Electronics*, *64*(6), 4866–4875. https://doi.org/10.1109/TIE.2017.2668987

Psiaki, M. L. (2013). The blind tricyclist problem and a comparative study of nonlinear filters: A challenging benchmark for evaluating nonlinear estimation methods. *IEEE Control Systems*, *33*(3), 40–54. https://doi.org/10.1109/MCS.2013.2249422

Qu, C. (2009). *Nonlinear Estimation for Model Based Fault Diagnosis of Nonlinear Chemical Systems*. Texas A&M University.

Qu, C., & Hahn, J. (2009). Computation of arrival cost for moving horizon estimation via unscented Kalman filtering. *Journal of Process Control*, *19*(2), 358–363. https://doi.org/10.1016/j.jprocont.2008.04.005

Rahoma, A. A. (2021). *Fault Detection and Root Cause Diagnosis Using Sparse Principal Component Analysis (SPCA)*. Memorial University of Newfoundland.

Ramalingam, M. K. (2013). *Moving Horizon Estimation with Dynamic Programming* [Masters, Cleveland State University]. https://engagedscholarship.csuohio.edu/etdarchive

Rao, C. V, Rawlings, J. B., & Lee, J. H. (2001). Constrained linear state estimation*a moving horizon approach. In *Automatica* (Vol. 37).

Rawlings, J. B., Mayne, D. Q., & Diehl, M. M. (2018). *Model Predictive Control: Theory, Computation, and Design* (2nd ed.). Nob Hill Publishing. http://www.nobhillpublishing.com

Reid, I. (2001). *Estimation II*. Oxford University. https://www.robots.ox.ac.uk/~ian/Teaching/Estimation/LectureNotes2.pdf

Ricker, N. L. (1990). Model Predictive Control with State Estimation. *Ind. Eng. Chem. Res*, *29*, 374–382. https://pubs.acs.org/sharingguidelines

Ritchie, S., & Eksteen, J. J. (2011). Investigating the effect of slag bath conditions on the existence of multiphase emulsion zones in PGM smelting furnaces using computation fluid dynamics. *Minerals Engineering*, *24*(7), 661–675. https://doi.org/10.1016/j.mineng.2010.09.017

Röbenack, K., & Reinschke, K. J. (2000). An efficient method to compute Lie derivatives and the observability matrix for nonlinear systems. *Int. Symposium on Nonlinear Theory and Its Applications (NOLTA)*.

Ross, K. (2021, December 13). *Transformations of random variables*. An Introduction to Probability and Simulation. https://bookdown.org/kevin_davisross/probsim-book/transformations-of-random-variables.html

Rowell, D. (2004). *Analysis and Design of Feedback Control Systems*.

Roy, T., & Dey, S. (2019). Fault Detectability Conditions for Linear Deterministic Heat Equations. *IEEE Control Systems Letters*, *3*(1), 204–209. https://doi.org/10.1109/LCSYS.2018.2872215

Rutgers Electrical & Computer Engineering. (2007). Controllability and Observability. In *332:505 Control Theory I* .

Saha, M., Ghosh, R., & Goswami, B. (2014). Robustness and sensitivity metrics for tuning the extended kalman filter. *IEEE Transactions on Instrumentation and Measurement*, *63*(4), 964–971. https://doi.org/10.1109/TIM.2013.2283151

Saha, M., Goswami, B., & Ghosh, R. (2011). *Two novel costs for determining the tuning parameters of the Kalman Filter*. http://arxiv.org/abs/1110.3895

Sahlberg, A. (2016). *Ensemble for deterministic sampling with positive weights*. www.stralsakerhetsmyndigheten.se

Salfner, F., Lenk, M., & Malek, M. (2010). A survey of online failure prediction methods. *ACM Computing Surveys*, *42*(3). https://doi.org/10.1145/1670679.1670680

Schneider, R., & Georgakis, C. (2013). How to NOT make the extended kalman filter fail. *Industrial and Engineering Chemistry Research*, *52*(9), 3354–3362. https://doi.org/10.1021/ie300415d

Shao, X., Huang, B., & Lee, J. M. (2009). Practical issues in particle filters for state estimation of complex chemical processes. *IFAC Proceedings Volumes (IFAC-PapersOnline)*, *15*(PART 1), 792–797. https://doi.org/10.3182/20090706-3-FR-2004.0330

Sheng, Y. Y., Irons, G. A., & Tisdale, D. G. (1998). Transport Phenomena in Electric Smelting of Nickel Matte: Part I. Electric Potential Distribution. *Metallurgical and Materials Transactions* , *29*, 77–83.

Shyamal, S. (2018). *Dynamic Optimization, State Estimation and Control of Electric Arc Furnace Operation*. McMaster University.

Simon, D. (2006). *Optimal State Estimation*. John Wiley & Sons.

Simon, D. (2010). Kalman filtering with state constraints: A survey of linear and nonlinear algorithms. *IET Control Theory and Applications*, *4*(8), 1303–1318. https://doi.org/10.1049/iet-cta.2009.0032

Smith, P. (2019). *Particle filters in practice*.

Snyder, C., Bengtsson, T., & Morzfeld, M. (2015). Performance bounds for particle filters using the optimal proposal. *Monthly Weather Review*, *143*(11), 4750–4761. https://doi.org/10.1175/MWR-D-15-0144.1

Sorenson, H. W. (1970). Least-squares estimation: from Gauss to Kalman. *IEEE Spectrum*, *7*(7), 63–68.

Souibgui, F., BenHmida, F., & Chaari, A. (2011). Particle filter approach to fault detection and isolation in nonlinear systems. *International Multi-Conference on Systems, Signals and Devices, SSD'11 - Summary Proceedings*. https://doi.org/10.1109/SSD.2011.5767499

Southall, B., Buxton, B. F., & Marchant, J. A. (2013). Controllability and Observability: Tools for Kalman Filter Design. *British Machine Vision Conference*, 17.1-17.10. https://doi.org/10.5244/c.12.17

Spiegelberg, J., & Rusz, J. (2017). Can we use PCA to detect small signals in noisy data? *Ultramicroscopy*, *172*, 40–46. https://doi.org/10.1016/j.ultramic.2016.10.008

Stigter, J. D., Joubert, D., & Molenaar, J. (2017). *Observability of Complex Systems: Finding the Gap*. www.nature.com/scientificreports/

Stigter, J. D., & Molenaar, J. (2015). A fast algorithm to assess local structural identifiability. *Automatica*, *58*, 118–124. https://doi.org/10.1016/j.automatica.2015.05.004

Straka, O., & Miroslavšimandl, M. M. (2006). PARTICLE FILTER ADAPTATION BASED ON EFFICIENT SAMPLE SIZE. *IFAC Symposium on System Identification*.

Subbararn, D., & Calise, A. J. (2001). Singular Perturbations and Time Scales in Guidance and Control of Aerospace Systems: A Survey. *JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS*, *24*(6).

Sui, D., & Johansen, T. A. (2011). Moving horizon observer with regularisation for detectable systems without persistence of excitation. *International Journal of Control*, *84*(6), 1041–1054. https://doi.org/10.1080/00207179.2011.589081

Sun, X. (2013). *Unknown Input Observer Approaches to Robust Fault Diagnosis THESIS*. University of Hull.

Sunahara, Y. (1970). An Approximate Method of State Estimation for Nonlinear Dynamical Systems. *Journal of Basic Engineering* , *92*(2), 385–393. http://fluidsengineering.asmedigitalcollection.asme.org/

The Pennsylvania State University- Department of Statistics Online Programs. (2018). *16.1 - Singular Value Decomposition*. https://online.stat.psu.edu/stat555/node/94/

Theodoridis, S. (2020). *Machine Learning A Bayesian and Optimization Perspective* (2nd ed., Vol. 1). Academic Press.

Thethwayo, B. M. (2010). *Sulphidation of Copper Coolers in PGM Smelters*. University of Pretoria.

Theunissen, C. D. (2021). *Fault Pattern Recognition in Simulated Furnace Data* [Chemical Engineering, Stellenbosch University]. https://scholar.sun.ac.za

Thrun, S., Burgard, W., & Fox, D. (2005). *Probabalistic Robotics*. The MIT Press.

Tian, J., Lu, H., Yang, K., Qin, J., Zhao, L., Zhou, J., Jiang, Y., & Ma, X. (2023). Quick estimation of parameters for the land surface data assimilation system and its influence based on the extended Kalman filter and automatic differentiation. *Science China Earth Sciences*, *66*(11), 2546–2562. https://doi.org/10.1007/s11430-022-1180-8

Tong, C. D., Yan, X. F., & Ma, Y. X. (2013). Statistical process monitoring based on improved principal component analysis and its application to chemical processes. *Journal of Zhejiang University: Science A*, *14*(7), 520–534. https://doi.org/10.1631/jzus.A1300003

Tyler, M. L., Asano, K., & Morari, M. (2000). Application of moving horizon estimation based fault detection to cold tandem steel mill. *International Journal of Control*, *73*(5), 427–438. https://doi.org/10.1080/002071700219605

Valappil, J., & Georgakis, C. (2004). Systematic Estimation of State Noise Statistics for Extended Kalman Filters. *AIChe Journal*, *46*(2).

Villaverde, A. F., Tsiantis, N., & Banga, J. R. (2019). Full observability and estimation of unknown inputs, states and parameters of nonlinear biological models. *Journal of the Royal Society Interface*, *16*(156). https://doi.org/10.1098/rsif.2019.0043

Wall, M. E., Rechtsteiner, A., & Rocha, L. M. (2003). Singular value decomposition and principal component analysis. *Springer US*, 91–109.

Wan, Y., & Keviczky, T. (2010). Real-time nonlinear moving horizon observer with pre-estimation for aircraft sensor fault detection and estimation †. *INTERNATIONAL JOURNAL OF ROBUST AND NONLINEAR CONTROL*. https://doi.org/10.1002/rnc

Wiener, N. (1956). *I Am a Mathematician*. MIT Press.

Wigren, A., Murray, L., & Lindsten, F. (2018, January 22). Improving the particle filter in high dimensions using conjugate artificial process noise. *Proceedings of the 18th IFAC Symposium on System Identification (SYSID)*. https://doi.org/10.1016/j.ifacol.2018.09.207

Wu, Y., Zhang, H., Wu, M., & Hu Dewen Hu, X. (2012). Observability of Strapdown INS Alignment: A Global Perspective. *IEE Transactions on Aerospace and Electronic Systems*, *48*(1).

Xiong, K., Chan, C. W., & Zhang, H. Y. (2007). Detection of Satellite Attitude Sensor Faults using the UKF. *IEEE TRANSACTIONS ON AEROSPACE AND ELECTRONIC SYSTEMS*, *43*(2).

Yang, Q. (2004). *Model-based and data driven fault diagnosis methods with applications to process monitoring*.

Yin, S., & Zhu, X. (2015). Intelligent particle filter and its application to fault detection of nonlinear system. *IEEE Transactions on Industrial Electronics*, *62*(6), 3852–3861. https://doi.org/10.1109/TIE.2015.2399396

Yin, X., & Liu, J. (2017). Distributed moving horizon state estimation of two-time-scale nonlinear systems. *Automatica*, *79*, 152–161. https://doi.org/10.1016/j.automatica.2017.01.023

Zavala, V. M., Laird, C. D., & Biegler, L. T. (2008). A Fast Moving Horizon Estimation Algorithm Based on Nonlinear Programming Sensitivity. *Journal of Process Control*, *18*(9), 876–884.

Zhang, K., Ding, S. X., Shardt, Y. A. W., Chen, Z., & Peng, K. (2017). Assessment of T2- and Q-statistics for detecting additive and multiplicative faults in multivariate statistical process monitoring. *Journal of the Franklin Institute*, *354*(2), 668–688. https://doi.org/10.1016/j.jfranklin.2016.10.033

APPENDIX A – DERIVATIONS

A.1 Least-squares derivation of the standard KF

The standard Kalman filter algorithm derivation is based on the derivation presented by Simon (2006).

The linear discrete-time system is represented by Equations 126 and 127. The standard form of the linear recursive estimator is represented by Equation 128.

$$x_k = F_{k-1}x_{k-1} + G_{k-1}u_{k-1} + w_{k-1} \quad [126]$$

$$y_k = H_k x_k + v_k \quad [127]$$

$$\hat{x}_k = \hat{x}_{k-1} + K_k(y_k - H\hat{x}_{k-1}) \quad [128]$$

For the Kalman filter derivation, the process and measurement noises are white, zero-mean, uncorrelated, and the variances are known.

$$w_k \sim (0, Q_k)$$

$$v_k \sim (0, R_k)$$

$$E\left[w_k w_j^T\right] = Q_k \delta_{k-j}$$

$$E\left[v_k v_j^T\right] = R_k \delta_{k-j}$$

Whereby δ_{k-j} is the Kronecker delta function.

$$\delta_{k-j} = \begin{cases} 1, & k = j \\ 0, & k \neq j \end{cases}$$

The Kalman filter is derived by finding the optimal value for the gain matrix K in the linear recursive estimator that minimizes the sum of the variances of the estimation errors. The estimation error is represented as:

$$\epsilon_{x,k} = x_k - \hat{x}_k \quad [129]$$

With the variance of the estimation errors is the estimation error squared.

$$\epsilon_{x,k}^2 = (x_k - \hat{x}_k)^2 \quad [130]$$

The estimation error mean can be computed as the expected value of the estimation error.

$$E\left(\epsilon_{x,k}\right) = E(x - \hat{x}_k) \quad [131]$$

Equations 126 and 128 are substituted into Equation 131 to obtain:

$$E\left(\epsilon_{x,k}\right) = E\left(x - \hat{x}_{k-1} - K_k(H_k x + v_k - H\hat{x}_{k-1})\right) \quad [132]$$

$$= E\left(\epsilon_{x,k-1} + K_k H_k(x - \hat{x}_{k-1}) - K_k v_k\right)$$

$$= (I - K_K H_k)E\left(\epsilon_{x,k-1}\right) - K_k E(v_k)$$

The estimation error covariance represents the uncertainty in the state estimates. The estimation error covariance, P, is defined as:

$$P = E\left(\epsilon_{x,k}\epsilon_{x,k}^T\right) \quad [133]$$

Equation 132 is substituted into Equation 133 and simplified using the knowledge that the measurement noise is zero-mean, meaning $E(v_k) = 0$.

$$P = (I - K_k H_k)P_{k-1}(I - K_k H_k)^T + K_k R_k K_k^T \quad [134]$$

The optimization problem aims to achieve a state estimate, $\hat{x}_k$, equal to the actual value of the state, x_k. Therefore, the loss function of the optimization problem must be chosen to satisfy this goal. The loss function, J_k is the expected value of the sum of variances of the estimation errors at time k.

$$J_k = E\left[\left(x_1 - \hat{x}_{1,k}\right)^2\right] + \cdots + E\left[\left(x_n - \hat{x}_{n,k}\right)^2\right] \quad [135]$$

$$= E(\epsilon_{x1,k}^2 + \cdots + \epsilon_{xn,k}^2)$$

$$= E\left(\epsilon_{x,k}^T \epsilon_{x,k}\right) = TrP_k$$

The loss function J_k must be minimized by adjusting the value of K_k. Therefore, the derivative of J_k with respect to K_k needs to be set equal to zero and K_k can be solved for.

$$\frac{\partial J_k}{\partial K_k} = 2(I - K_k H_k)P_{k-1}\left(-H_k^T\right) + 2K_k R_k \quad [136]$$

$$0 = 2(I - K_k H_k)P_{k-1}\left(-H_k^T\right) + 2K_k R_k$$

$$K_k = P_{k-1}H_k^T\left(H_k P_{k-1} H_k^T + R_k\right)^{-1}$$

Solved via the chain rule, $\frac{\partial Tr(ABA^T)}{\partial A} = 2AB$.

This Kalman gain is substituted into the state observer equation to obtain the update step of the Kalman filter.

$$\hat{x}_k = \hat{x}_{k-1} + K_k(v_k) \quad [137]$$

$$K_k = P_{k-1}H_K^T(S_k)^{-1}$$

Where:

$$v_k = y_k - H\hat{x}_{k-1} \quad [138]$$

And S_k is the innovation covariance:

$$S_k = H_k P_{k-1} H_k^T + R_k \quad [139]$$

The covariance is updated in the update step using Equation 134.

Equation 137 represents the update step of the standard Kalman filter.

The way in which the states and covariances propagate through time can be derived by taking the expected value for both sides of Equation 126. The expected value of the right-hand-side of Equation 126 is given by Equation 141. The expected value, and thus the mean, of x_k is represented by $\bar{x}_k$. The expected value of the constant, $G_{k-1}u_{k-1}$, is simply itself. The expected value of the noise, w_{k-1}, is zero as w represents a zero-mean random variable.

$$E(F_{k-1}x_{k-1} + G_{k-1}u_{k-1} + w_{k-1}) = F_{k-1}\bar{x}_{k-1} + G_{k-1}u_{k-1} + 0 \quad [140]$$

$$\bar{x}_k = E(x_k) = F_{k-1}\bar{x}_{k-1} + G_{k-1}u_{k-1} \quad [141]$$

Using Equations 126 and 141:

$$(x_k - \bar{x}_k)(x_k - \bar{x}_k)^T = (F_{k-1}(x_{k-1} - \bar{x}_{k-1}) + w_{k-1})\,(\cdots)^T \quad [142]$$

Therefore, the covariance can be computed as:

$$P_k = E[(x_k - \bar{x}_k)(x_k - \bar{x}_k)^T] = F_{k-1}P_{k-1}F_{k-1}^T + Q_{k-1} \quad [143]$$

The process noise, Q, is added to this covariance as it represents the uncertainty in the process model which must be considered during the propagation of the state. Equations 141 and 143 represent the prediction step of the standard Kalman filter.

A.2 Bayesian derivation of the standard KF

A basic introduction to random variables is given to understand the Bayesian derivation of the Kalman filter. A stochastic process involves random variables changing over time. The systems that will be used for the entirety of the study are assumed to be stochastic processes with random variables of the state, x, and the measurements, y, changing over time. A random variable is a representation of a set of outcomes as a set of real numbers. The properties of random variables include the probability distribution function (PDF), which can be defined for a random variable X and a nonrandom variable or constant x as:

$$P_X(x) = P(X \leq x) \quad [144]$$

Another property of a random variable is its probability density function (pdf) which is defined as:

$$p_X(x) = \frac{dP_X(x)}{dx} \quad [145]$$

The joint pdf of two random variables is $p_{XY}(x, y)$. The joint pdf is symmetrical, therefore, $p_{XY}(x, y) = p(y, x)$.

Some important probability rules that are used in the derivation of the Bayesian filter and the subsequent derivation of the Kalman filter are the Chain Rule, Marginalization, and Bayes Rule.

The Chain Rule, also known as the fundamental rule, is a way to write the joint pdf of two random variables in terms of the conditional pdf and the marginal pdf:

$$p(x, y) = p(x|y)p(y) = p(y|x)p(y) \quad [146]$$

(Brownlee, 2019)

Marginalization is a property of the joint pdf whereby the marginal distribution of x and y can be written in terms of the joint pdf.

$$p(x) = \int_{-\infty}^{\infty} p(x, y)dy \quad [147]$$

$$p(y) = \int_{-\infty}^{\infty} p(x,y)dx$$

Bayes rule can be defined as:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} = \frac{p(x,y)}{p(y)} \quad [148]$$

(Simon, 2006)

By marginalization and the Chain rule, Bayes rule can be written as:

$$p(x|y) = \frac{p(x,y)}{\int_{-\infty}^{\infty} p(x,y)dx} = \frac{p(y|x)p(x)}{\int_{-\infty}^{\infty} p(y|x)p(x)dx} \quad [149]$$

Bayesian filter derivation

A discrete-time system can be described by:

$$x_{k+1} = f_k(x_k, w_k) \quad [150]$$

$$y_k = h_k(x_k, v_k) \quad [151]$$

The underlying model in the Bayesian filter is the Markov model which makes use of the Markov assumption and the output independence assumption. The Markov assumption states that the current state, x_k, is conditionally dependent on only the state in the previous time step, x_{k-1}, and is therefore conditionally independent of all other past states (Bickel et al., 1996). This can be written in terms of the conditional pdf as:

$$p(x_k|x_{0:k-1}, Y_{k-1}) = p(x_k|x_{k-1}) \quad [152]$$

Similarly, the output independence assumption states that the measurement at any time step, y_k, depends solely on the state at that particular timestep, x_k, and is independent of any other states.

$$p(y_k|x_{0:k-1}, Y_{k-1}) = p(y_k|x_k) \quad [153]$$

(Jurafsky & Martin, 2021)

This means that when modelling the dynamics of the state, and when modelling the state-measurement relationship, all past values of the state besides x_{k-1}, and all past measurements besides y_k, do not need to be considered (Elfring et al., 2021).

In the prediction step of the filter, a pdf of the current state, x_k, given all the measurements prior to time k, needs to be computed. This is the conditional pdf of x_k given $Y_{k-1} = y_1, y_2, \cdots y_{k-1}$, which is represented as $p(x_k|Y_{k-1})$. To derive the equation for the conditional pdf of x_k given Y_{k-1}, Bayes rule and marginalization need to be used. By marginalizing out the previous states, x_{k-1}, the prediction step can be computed as:

$$p(x_k|y_{1:k-1}) = \int p[(x_k, x_{k-1})|Y_{k-1}]dx_{k-1} \quad [154]$$

Via the Chain rule:

$$p(x_k|y_{1:k-1}) = \int p[x_k|(x_{k-1}, Y_{k-1})]p[x_{k-1}|Y_{k-1}]dx_{k-1} \quad [155]$$

Using the Markov assumption that $p[x_k|(x_{k-1}, Y_{k-1})] = p(x_k|x_{k-1})$:

$$p(x_k|y_{1:k-1}) = \int p[x_k|x_{k-1}]p[x_{k-1}|Y_{k-1}]dx_{k-1} \quad [156]$$

This is the *a priori* pdf.

The update step of the filter involves obtaining the current state estimate using all the measurements, including the measurement at the current time. This is the conditional pdf of x_k given Y_k. The conditional pdf is derived from Bayes rule as:

$$p(x_k|Y_k) = \frac{p(Y_k|x_k)p(x_k)}{\int_{-\infty}^{\infty} p(Y_k|x_k)p(x_k)dx_k} \quad [157]$$

Via the Chain rule, the numerator can be rewritten as:

$$p(x_k|Y_k) = \frac{p(Y_k|x_k, Y_{k-1})p(Y_{k-1}|x_k)p(x_k)}{\int_{-\infty}^{\infty} p(Y_k|x_k)p(x_k)dx_k} \quad [158]$$

According to the output dependence assumption $p(y_k|x_k, Y_{k-1}) = p(y_k|x_k)$, then:

$$p(x_k|Y_k) = \frac{p(Y_k|x_k)p(Y_{k-1}|x_k)p(x_k)}{\int_{-\infty}^{\infty} p(Y_k|x_k)p(x_k)dx_k} \quad [159]$$

From Bayes rule we know that:

$$p(x_k|Y_{k-1}) = \frac{p(Y_{k-1}|x_k)p(x_k)}{p(Y_{k-1})} \quad [160]$$

Therefore,

$$p(x_k|Y_{k-1})p(Y_{k-1}) = p(Y_{k-1}|x_k)p(x_k) \quad [161]$$

This can be substituted into Equation 147 and cancelling out the $p(Y_{k-1})$ term in the numerator and denominator gives:

$$p(x_k|Y_k) = \frac{p(Y_k|x_k)p(x_k|Y_{k-1})}{\int_{-\infty}^{\infty} p(Y_k|x_k)p(x_k|Y_{k-1})dx_k} \quad [162]$$

This is the *a posteriori* pdf.

(Lambert, 2018)

Kalman filter derivation from Bayesian filter

The analytical solutions for the a priori state estimate and a posteriori state estimation in Equations 156 and 162, respectively, can only be computed for linear systems with Gaussian distributions of the initial state estimate x_0, process noise w_k and measurement noise v_k (Simon, 2006). The analytical solution for the state estimates under these conditions is equivalent to the Kalman filter equations.

In the prediction step of the Bayesian filter, the *a priori* pdf was obtained as Equation 156. To compute the analytical solution for this equation, two assumptions need to be made:

1) The *a posteriori* pdf from the previous time step, $p(x_{k-1}|y_{1:k-1})$, is Gaussian with a known mean, $\hat{x}_{k-1}$, and a known covariance, P_{k-1}.

$$x_{k-1} \sim N(\hat{x}_{k-1}, P_{k-1})$$

2) The linear process model is given by:

$$x_k = F_{k-1}x_{k-1} + G_{k-1}u_{k-1} + w_{k-1} \quad [163]$$

$$y_k = H_k x_k + v_k$$

Where the noise terms are Gaussian, zero-mean noise with known covariance:

$$w_k \sim N(0, Q_k)$$

$$v_k \sim N(0, R_k)$$

The analytical solution also requires the known property of random variable transformations. When a scalar value, α, is added to a Gaussian random variable $z_1 \sim N(\mu_1, \sigma_1^2)$, the resulting random variable is:

$$z = z_1 + \alpha \sim (\mu_1 + \alpha, \sigma_1^2) \quad [164]$$

The scalar is simply added to the mean and the covariance remains the same as the original random variable.

The other property involves two independent Gaussian random variables $z_1 \sim N(\mu_1, \sigma_1^2)$ and $z_2 \sim N(\mu_2, \sigma_2^2)$ who undergo a linear transformation $z = B_1 z_1 + B_2 z_2$. The resulting random variable z, is a Gaussian random variable:

$$z \sim (B_1\mu_1 + B_2\mu_2, B_1\sigma_1^2 B_1^T + B_2\sigma_2^2 B_2^T) \quad [165]$$

Therefore, the linear transformation of x_{k-1} and w_{k-1} through Equation 163 results in a Gaussian random variable x_k with a mean calculated by adding the scalar quantity $G_{k-1}u_{k-1}$ and adding the mean of w_{k-1} , which is zero, to the mean of the original random variable:

$$\hat{x}_k = F_{k-1}\hat{x}_{k-1} + G_{k-1}u_{k-1} + 0$$

This equation is equivalent to the Equation 5 derived earlier in the Kalman filter derivation from least squares.

The covariance of the random variable x_k is calculated as:

$$P_k = F_{k-1}P_{k-1}F_{k-1}^T + Q_{k-1}$$

This is the same as Equation 6 derived earlier.

In the update step of the Bayesian filter, the *a posteriori* pdf was obtained by Equation 162. To compute the analytical solution for this equation, a known property of the conditional distributions of Gaussian random variables needs to be understood. This property states that if two Gaussian random variables have the joint probability density function:

$$\begin{bmatrix} x \\ y \end{bmatrix} \sim N\left(\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} P_{xx} & P_{xy} \\ P_{yx} & P_{yy} \end{bmatrix}\right)$$

Then the conditional density of x given y is:

$$p(x|y) = N(x;\ \mu_x + P_{xy}P_{yy}^{-1}(y - \mu_y), P_{xx} - P_{xy}P_{yy}^{-1}P_{yx}$$

From the prediction step we have the random variable $x_k \sim N(\hat{x}_k, P_k)$. Now, the conditional density of x_k given y_{k-1} is:

$$\begin{bmatrix} x \\ y \end{bmatrix} | \, y_{k-1} \sim N\left(\begin{bmatrix} \hat{x}_k \\ H_k \hat{x}_k \end{bmatrix}, \begin{bmatrix} P_k & P_k H_k^T \\ H_k P_k & H_k P_k H_k^T + R_k \end{bmatrix} \right) \quad [166]$$

From the property above, the conditional density of x_k given y_k can be calculated as:

$$p(x_k | y_k) = N(x; \mu_x + P_{xy} P_{yy}^{-1}(y - \mu_y), P_{xx} - P_{xy} P_{yy}^{-1} P_{yx}) \quad [167]$$

The mean is calculated as:

$$\hat{x}_k = \hat{x}_{k-1} K_k (y_k - H \hat{x}_{k-1}) \quad [168]$$

With $K_k = P_{k-1} H_K^T \left(H_k P_{k-1} H_k^T + R_k \right)^{-1}$. This is equivalent to the Kalman filter update equation.

The covariance is calculated as:

$$P_k = (I - K_k H_k) P_{k-1} (I - K_k H_k)^T + K_k R_k K_k^T \quad [169]$$

Equivalent to the Kalman filter update equation.

(Gurajala et al., 2021)

A.3 Continuous-time KF

The following derivation of the continuous-time Kalman filter equations is based on the derivation presented by Simon (2006).

The linear continuous-time system can be represented by:

$$\dot{x} = Ax + Bu + w \quad [170]$$

$$y = Cx + v$$

$$w \sim (0, Q_c)$$

$$v \sim (0, R_c)$$

Where A represents the state transition matrix, B represents the input matrix, and C represents the measurement matrix.

The exact solution of the continuous-time linear system is given by:

$$x(t) = e^{A(t-t_0)} x(t_0) + \int_{t_0}^{t} e^{A(t-\tau)} B u(\tau) d\tau \quad [171]$$

Assuming that $A(\tau), B(\tau)$, and $u(\tau)$ remain constant over the integration period. Let $t = t_k$, a discrete time point, and $t_0 = t_{k-1}$. With $\Delta t = t_k - t_{k-1}$ and $\alpha = \tau - t_{k-1}$.

$$x(t_k) = e^{A\Delta t} x(t_{k-1}) + e^{A\Delta t} \int_0^{\Delta t} e^{-A\alpha} d\alpha \, B u(t_{k-1}) \quad [172]$$

$$x_k = F_{k-1}x_{k-1} + G_{k-1}u_{k-1}$$

Whereby $F = e^{A\Delta t}$ and $G = F\int_0^{\Delta t} e^{-A\tau}d\tau\, B = F\left[I - e^{-A\Delta t}\right]A^{-1}B$. This uses the Taylor series expansion, $e^X = \sum_{j=0}^{\infty}\frac{X^j}{j!}$, considering the first two terms of the expansion. If A is invertible, then:

$$\int_0^{\Delta t} e^{-A\tau}d\tau = \int_0^{\Delta t}\sum_{j=0}^{\infty}\frac{(-A\tau)^j}{j!}d\tau \approx \left[I - e^{-A\Delta t}\right]A^{-1} \quad [173]$$

The continuous-time system is discretized, for small time steps of Δt, as:

$$x_{k+1} = Fx_k + Gu_k + w_k \quad [174]$$

$$y_k = H_k x_k + v_k \quad [175]$$

$$F \approx (I + A\Delta t)$$

$$G \approx B\Delta t$$

$$H = C$$

$$w_k \sim (0, Q) \qquad Q = Q_c\Delta t$$

$$v_k \sim N(0, R) \qquad R = \frac{R_c}{\Delta t}$$

The estimation error covariance from Equation 143 can be rewritten for a small sampling time of Δt by substituting the approximation for F and Q.

$$P_{k+1}^- = (I + A\Delta t)P_k^+(I + A\Delta t)^T + Q_c\Delta t \quad [176]$$

$$= P_k^+ + (AP_k^+ + P_k^+A^T + Q_c)\Delta t + AP_k^+A^T\Delta t^2$$

Substituting in the value of P_k^+ from Equation 134 gives:

$$P_{k+1}^- = (I - K_kC)P_k^- + AP_k^+A^T\Delta t^2 + [A(I - K_kC)P_k^- + (I - K_kC)P_k^-A^T + Q_c]\Delta t \quad [177]$$

Subtract P_k^- from both sides and divide by Δt. Then take the limit as $\Delta t \to 0$ to obtain the continuous-time covariance update equation.

The discrete-time Kalman gain is given in Equation 136. From this, it can be derived that:

$$\frac{K_k}{\Delta t} = P_k^-C^T\left(CP_k^-C^T + \frac{R_c}{\Delta t}\right)^{-1} \div \Delta t = P_k^-C^T(CP_k^-C^T\Delta t + R_c)^{-1}$$

$$\lim_{\Delta t\to\infty}\frac{K_k}{\Delta t} = P_k^-C^TR_c^{-1}$$

Which allows for the following equation to be derived.

$$\dot{P} = -PC^TR_c^{-1}CP + AP + PA^T + Q_c \quad [178]$$

This is the continuous-time differential Riccati equation for propagation of the error covariance through time. Solving Equation 178 gives the priori error covariance at time k, $P(k\Delta t) = P_k^-$ (Simon, 2006).

A.4 First order Taylor series approximation of a nonlinear function

The following proof is based on the derivation of the approximation error of a first order Taylor series approximation presented by Simon (2006).

For a continuous-time, nonlinear dynamic model of a system that is represented as:

$$\dot{x} = f(x, u, w, t) \qquad [179]$$

$$y = h(x, v, t)$$

$$w \sim (0, Q)$$

$$v \sim (0, R)$$

The Taylor series expansion of a single variable, x, around a nominal point, x_0, can be expressed as:

$$f(x) = f(x_0) + \frac{\partial f}{\partial x}\Big|_{x_0}(x - x_0) + \frac{\partial f^2}{\partial x^2}\Big|_{x_0}(x - x_0)^2 + \frac{\partial f^3}{\partial x^3}\Big|_{x_0}(x - x_0)^3 + \cdots \qquad [180]$$

$$= f(x_0) + \frac{\partial f}{\partial x}\Big|_{x_0}(\Delta x) + \frac{\partial f^2}{\partial x^2}\Big|_{x_0}(\Delta x)^2 + \frac{\partial f^3}{\partial x^3}\Big|_{x_0}(\Delta x)^3 + \cdots$$

But for small values of Δx, this can be approximated as:

$$f(x) \approx f(x_0) + \frac{\partial f}{\partial x}\Big|_{x_0}(\Delta x) \qquad [181]$$

This approximation only uses the first order of the Taylor series expansion, only considering the first two terms of the total Taylor series expansion.

The linearization of Equation 179 by a first order Taylor series expansion around a nominal control, u_0, nominal state, x_0, nominal output, y_0 and nominal noise, w_0 and v_0, results in:

$$\dot{x} \approx f(x_0, u_0, w_0, t) + \frac{\partial f}{\partial x}\Big|_{x_0}(x - x_0) + \frac{\partial f}{\partial u}\Big|_{u_0}(u - u_0) + \frac{\partial f}{\partial w}\Big|_{w_0}(w - w_0) \qquad [182]$$

$$= f(x_0, u_0, w_0, t) + A\Delta x + B\Delta u + L\Delta w$$

$$y \approx h(x_0, v_0, t) + \frac{\partial h}{\partial x}\Big|_{x_0}(x - x_0) + \frac{\partial h}{\partial v}\Big|_{v_0}(v - v_0) \qquad [183]$$

$$= h(x_0, v_0, t) + C\Delta x + M\Delta v$$

Further assumptions made are that the nominal noise values have an expected value equal to zero, therefore, $\Delta w(t) = w(t)$ and $\Delta v(t) = v(t)$. It is also assumed that the control input, $u(t)$, is perfectly known, so $u_0(t) = u(t)$. If the control input is not known perfectly, then the control is expressed as $u_0(t) + \Delta u(t)$ where the nominal value is known and $\Delta u(t)$ is a zero-mean random variable.

The approximation error of the first order Taylor series approximation can be calculated using a general nonlinear equation $z = g(x)$. The nonlinear equation, $z = g(x)$, can be expanded in the Taylor series around $\bar{x}$ as:

$$z = g(\bar{x}) + D_{\tilde{x}}h + \frac{1}{2!}D_{\tilde{x}}^2 h + \cdots \quad [184]$$

The mean of z can be computed as:

$$\bar{z} = g(\bar{x}) + E\left[D_{\tilde{x}}h + \frac{1}{2!}D_{\tilde{x}}^2 h + \cdots\right] \quad [185]$$

Where $D_{\tilde{x}} = (x - \bar{x})\frac{\partial g}{\partial x}\Big|_{\bar{x}}$. We know $E[D_{\tilde{x}}g] = E[(x - \bar{x})\frac{\partial g}{\partial x}\Big|_{\bar{x}}]$. But $E[(x - \bar{x})] = 0$.

Likewise, the expected values of third order moments will always be zero since $E[(x - \bar{x})] = 0$. Therefore, the true mean of z is:

$$\bar{z} = g(\bar{x}) + \frac{1}{2!}E[D_{\tilde{x}}^2 g] + \frac{1}{4!}E[D_{\tilde{x}}^4 g] + \cdots \quad [186]$$

By performing first order linearization of $z = g(x)$, the approximated mean is:

$$\bar{z} = g(\bar{x}) + E[D_{\tilde{x}}g] \quad [187]$$

Therefore, the linear approximation for the mean propagation is only correct up until the first order.

Similarly, the true covariance of the random variable z is given as:

$$P_z = E[(z - \bar{z})(z - \bar{z})^T] \quad [188]$$

$$= E[D_{\tilde{x}}g(D_{\tilde{x}}g)^T] + E\left[\frac{D_{\tilde{x}}g\left(D_{\tilde{x}}^3 g\right)^T}{3!} + \frac{D_{\tilde{x}}^2 g\left(D_{\tilde{x}}^2 g\right)^T}{2!\,2!} + \frac{D_{\tilde{x}}^3 g(D_{\tilde{x}}g)^T}{3!}\right] + E\left[\frac{D_{\tilde{x}}^2 g}{2!}\right]E\left[\frac{D_{\tilde{x}}^2 g}{2!}\right]^T + \cdots$$

$$= HPH^T + E\left[\frac{D_{\tilde{x}}g\left(D_{\tilde{x}}^3 g\right)^T}{3!} + \frac{D_{\tilde{x}}^2 g\left(D_{\tilde{x}}^2 g\right)^T}{2!\,2!} + \frac{D_{\tilde{x}}^3 g(D_{\tilde{x}}g)^T}{3!}\right] + E\left[\frac{D_{\tilde{x}}^2 g}{2!}\right]E\left[\frac{D_{\tilde{x}}^2 g}{2!}\right]^T + \cdots$$

The linear approximation for the covariance is $P \approx HPH^T$, which is only the first term of the true covariance.

A.5 Unscented transformation

The approximate mean using the unscented transformation is given by:

$$\bar{z}_{approx} = \sum_{i=1}^{2n} W^i y^i \quad [189]$$

Where the weighting coefficient is defined as:

$$W^0 = \frac{\kappa}{n+\kappa} \quad [190]$$

$$W^i = \frac{1}{2(n+\kappa)} \quad i = 1, \cdots, 2n$$

If $\kappa = 0$, then the sigma point representing the mean has zero weighting and the $2n$ sigma points have equal weightings. For Gaussian distributions, $\kappa = 3 - n$ reduces the approximation error of the unscented transformation by minimizing the fourth order difference between the true mean and covariance and the approximate mean and covariance (Ebeigbe et al., 2021).

When $\kappa = 0$, the approximate mean can be expanded to:

$$\bar{z}_{approx} = g(\bar{x}) + \frac{1}{2!}E\left[D^2_{\tilde{x}^i}g\right] + \frac{1}{2n}\sum_{i=1}^{2n}\left(\frac{1}{4!}D^4_{\tilde{x}^i}g + \frac{1}{6!}D^6_{\tilde{x}^i}g + \cdots\right) \quad [191]$$

The true mean is given by equation 186. Thus, the unscented transformation mean approximation matches the true mean up until the third order.

The unscented transformation covariance approximation is given by:

$$P_{approx} = \sum_{i=1}^{2n} W^i\left(z^i - \bar{z}_{approx}\right)\left(z^i - \bar{z}_{approx}\right)^T \quad [192]$$

This can be expanded to:

$$P_{approx} = HPH^T + higher\ order\ terms$$

The unscented transformation covariance approximation matches the true covariance up until the third order (Simon, 2006).

A.6 Full information estimator

The following derivation is based on the derivation for the full information estimator presented by Larsson (2015) and the derivation presented by Haseltine & Rawlings (2003).

To derive the full information estimator, the maximum a posteriori (MAP) estimate must first be defined. The MAP estimate aims to find the most likely values for the state trajectory $\{\hat{x}_0, \ \dots \ , \hat{x}_N\}$ based on the set of past measurements. This is done by finding the values of $\{x_0, \ \dots \ , x_N\}$ that maximize the objective function $p(x_0, \dots, x_N | y_0, \dots, y_N)$, which is the pdf of x conditioned on y.

$$\{\hat{x}_0, \ \dots \ , \hat{x}_N\} = \max_{x_0, \dots, x_N} p(x_0, \dots, x_N | y_0, \dots, y_N) \quad [193]$$

This MAP estimate can be rewritten as a minimization:

$$\max_{x_0, \dots, x_N} p(x_0, \dots, x_N | y_0, \dots, y_N) = \min_{x_0, \dots, x_N} -\log p(x_0, \dots, x_N | y_0, \dots, y_N) \quad [194]$$

Using Bayes rule and the Markov property, the objective function $p(x_0, \dots, x_N | y_0, \dots, y_N)$ can be rewritten as:

$$p(x_0, \dots, x_N | y_0, \dots, y_N) = p(x_0)\prod_{k=1}^{N} p(y_k | x_k)\prod_{k=0}^{N-1} p(x_{k+1} | x_k) \quad [195]$$

Therefore, the MAP estimate can be rewritten as:

$$\min_{x_0, \dots, x_N} -\log p(x_0) - \sum_{k=1}^{N} \log p(y_k | x_k) - \sum_{k=0}^{N-1} \log p(x_{k+1} | x_k) \quad [196]$$

Assuming that the system consists of process dynamics and measurements defined by:

$$x_{k+1} = f(x_k, w_k) \quad [197]$$

$$y_k = h_k(x_k, v_k)$$

$$w_k \sim (0, Q)$$

$$v_k \sim (0, R)$$

The probabilities in Equation 196 can be written as:

$$\log p(y_k|x_k) = p(v_k) \quad [198]$$

$$\log p(x_{k+1}|x_k) = p(w_k) \quad [199]$$

Assuming the measurement noise is Gaussian, zero-mean with covariance R, the probability of the measurement noise is given by:

$$p(v_k) = \frac{1}{(2\pi)^{n/2}|R|^{1/2}} \exp\left(-\frac{1}{2}(v_k - \overline{v_k})^T R^{-1}(v_k - \overline{v_k})\right) \quad [200]$$

Taking the logarithm of both sides yields:

$$-\log p(v_k) = \frac{1}{2} v_k^T R^{-1} v_k \quad [201]$$

Similarly for the process noise, assuming the process noise is Gaussian, zero-mean with covariance Q:

$$-\log p(w_k) = \frac{1}{2} w_k^T Q^{-1} w_k \quad [202]$$

The logarithm of the probability of the initial state is given as:

$$-\log p(x_0) = \frac{1}{2}(x_0 - \bar{x}_0)^T P_0^{-1}(x_0 - \bar{x}_0) \quad [203]$$

Substituting these back into the MAP estimate in Equation 196 Gives the full information estimate:

$$\min_{x_0,\cdots,x_k} ||x_0 - \bar{x}_0||^2_{P_0^{-1}} + \sum_{j=1}^{k} \left||y_j - h(x_j)\right||^2_{R^{-1}} + \sum_{j=0}^{k-1} \left||x_{j+1} - f(x_j)\right||^2_{Q^{-1}} \quad [204]$$

A.7 Linear discrete-time observability matrix

The observability matrix is derived for the following linear discrete-time system (Dahleh et al., 2011):

$$x_{k+1} = Fx_k + Gu_k \quad [205]$$

$$y_k = Hx_k \quad [206]$$

To test system observability, the observability of states, x, from the available measurements, y, is checked at time zero to time $n-1$. This is done up until time $n-1$ as it is assumed that a maximum number of n measurements is needed to fully reconstruct the n dimensional state vector (Rutgers Electrical & Computer Engineering, 2007).

$$y_0 = Hx_0 \quad [207]$$

$$y_1 = Hx_1 = HFx_0$$

$$y_2 = Hx_2 = HFx_1 = HF^2x_0$$

$$\vdots$$

$$y_{n-1} = Hx_{n-1} = HF^{n-1}x_0$$

If the initial state, x_0, is known, then the state at any other point in time can be solved for using Equation 205. In other words, to fully construct the states of the system, the only value that is required is the initial state. Therefore, a system is observable if the initial state can be derived from the outputs. The relationship between the outputs, y, and the initial state, x_0, is written as:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = Ox_0 \quad [208]$$

Where O is defined as:

$$O = \begin{bmatrix} H \\ HF \\ \vdots \\ HF^{n-1} \end{bmatrix} \quad [209]$$

A.8 Linear continuous-time observability matrix

A continuous-time linear system is represented by the following system equations.

$$\dot{x} = Ax + Bu \quad [210]$$

$$y = Cx \quad [211]$$

Using Equation 210, the state at any time can be solved for from the state at time zero.

$$x(t) = e^{A(t-t_0)}x(t_0) \quad [212]$$

The observability at different time points can be assessed by taking the time derivatives of the observation at time zero.

$$y(t_0) = Cx(t_0) \quad [213]$$

$$\dot{y}(t_0) = C\dot{x}(t_0) = CAx(t_0)$$

$$\vdots$$

$$y^{(n-1)}(t_0) = Cx^{(n-1)}(t_0) = CA^{n-1}x(t_0)$$

This derivation results in the same observability matrix as in Equation 209, however, the state and measurement matrices are now in continuous form.

$$O = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix} \quad [214]$$

The observability matrix for continuous-time linear systems can also be derived from the Lie derivatives of the linear function.

Firstly, the definition of a Lie derivative is established. For a smooth scalar function $h(x)$ of the $n \times 1$ state vector x, the gradient of the function is given by:

$$\bar{V}h = \frac{\partial h}{\partial x} = \begin{bmatrix} \frac{\partial h}{\partial x_1} & \frac{\partial h}{\partial x_2} & \cdots & \frac{\partial h}{\partial x_n} \end{bmatrix} \quad [215]$$

For a vector function f of the $n \times 1$ state vector x forms a vector field that spans over $x \in \mathbb{R}^n$. The gradient of the vector field is given by the Jacobian matrix:

$$\bar{V}f = \frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \quad [216]$$

The Lie derivative is the derivative of a scalar function along a vector field, which is the dot product of the two vectors. The two vectors are the gradient of h and the vector function f.

$$L_f h = \bar{V}h \cdot f = \sum_{i=1}^{n} \frac{\partial h}{\partial x_i} f_i \quad [217]$$

$$= \begin{bmatrix} \frac{\partial h}{\partial x_1} & \frac{\partial h}{\partial x_2} & \cdots \frac{\partial h}{\partial x_n} \end{bmatrix} \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{bmatrix}$$

The Lie derivative is a scalar value. (Hedrick & Girard, 2005)

The first order Lie derivative of h with respect to f is the time derivative of h (Paradowski, 2021). From the chain rule, the time derivative of h is the gradient of h multiplied by the function f:

$$\dot{h} = \frac{\partial h}{\partial x}\left(\frac{\partial x}{\partial t}\right) = \bar{V}h \cdot f = L_f^1 h \quad [218]$$

Higher order Lie derivatives can be defined as:

$$L_f^0 h = h \quad [219]$$

$$L_f^1 h = \bar{V}h \cdot f$$

$$L_f^2 h = \bar{V}[L_f^1 h] \cdot f$$

$$\vdots$$

$$L_f^n h = \bar{V}[L_f^{n-1} h] \cdot f$$

(Nasir, 2020)

The linear continuous-time system. Let $h(x) = Cx$ and $f(x) = Ax$. Therefore, $\bar{V}h = \frac{\partial h}{\partial x} = C$. Then the Lie derivatives can be calculated as:

$$L_f^0 h = h = Cx$$

$$L_f^1 h = \bar{V}h \cdot f = CAx$$

$$L_f^2 h = \bar{V}\left[L_f^1 h\right] \cdot f = \bar{V}[CAx] \cdot (Ax) = CA^2 x$$

$$\vdots$$

$$L_f^n h = CA^{n-1} x$$

The observability matrix is the gradient of the Lie derivatives with respect to x:

$$O = \begin{bmatrix} dL_f^0 h \\ dL_f^1 h \\ \vdots \\ dL_f^{n-1} h \end{bmatrix} \quad [220]$$

Which corresponds to:

$$O = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix} \quad [221]$$

(Nasir, 2020)

A.9 Nonlinear continuous-time observability matrix

Based on the work done by Hermann & Krener (1977).

For a nonlinear system defined by:

$$\dot{x} = f(x, u) \quad [222]$$

$$y = h(x) \quad [223]$$

Let there exist two initial conditions, $x_0 = z^1$ and $x_0 = z^2$. The trajectories of the states through a nonlinear function are given by $\varphi(t, z^1)$ and $\varphi(t, z^2)$, respectively. The states, z^1 and z^2, are distinguishable if the output mapping of the trajectories are not equal.

$$h\big(\varphi(t, z_1)\big) \neq h\big(\varphi(t, z_2)\big) \quad [224]$$

If all pairs of z^1 and z^2 are distinguishable then a system is globally observable. The system is locally weakly observable when the unique initial conditions z^1 and z^2 have unique outputs $y^1 \neq y^2$.

To verify the output mapping of the trajectories are not equal, a Taylor series expansion of the output can be used.

$$y = h\big(\varphi(t, x)\big) = \sum_{k=0}^{\infty} \frac{t_k}{k!} \big(L_f^k h\big)(x_0) \quad [225]$$

The coefficients in this expansion are the Lie derivatives.

To prove that $y^1 \neq y^2$ for initial conditions z^1 and z^2, it must be proven that:

$$\big(L_f^k h\big)(z_1) \neq \big(L_f^k h\big)(z_2) \quad k = 0,1, \dots, \infty \quad [226]$$

This is a system of infinite nonlinear equations which can be summarized in an observability mapping defined by $G(x)$.

$$G(x) = \begin{bmatrix} L_f^0 h(x) \\ \vdots \\ L_f^{\infty} h(x) \end{bmatrix} \quad [227]$$

The Jacobian of this observability mapping is given by dG and Hermann & Krener prove that the system is locally weakly observable if the rank of $dG = n$.

This derivation of the Lie derivative of the output function, h, along the vector field, f, assumes that the time derivatives of the control variable are zero and thus the control inputs are constant. To take into account time-varying control inputs, the extended Lie derivative can be defined.

The local observability matrix at x_0 for constant input variables, u, when there exists a single measurement is:

$$O(x_0, u) = \frac{\partial l(x_0)}{\partial x}\Big|_{x=x_0} \quad [228]$$

$$l(x_0, u) = \begin{bmatrix} L_f^0 h_1 \\ L_f^1 h_1 \\ \vdots \\ L_f^{n-1} h_1 \end{bmatrix}$$

For n_y measurements where $h(x) = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_p \end{bmatrix}$, then the local observability matrix at x_0 is defined as:

$$O(x_0, u) = \begin{bmatrix} dL_f^0 h_1 \\ \vdots \\ dL_f^0 h_{n_y} \\ \vdots \\ dL_f^{n-1} h_1 \\ \vdots \\ dL_f^{n-1} h_{n_y} \end{bmatrix} \quad [229]$$

A.10 Scaling a linear continuous-time system of equations

A linear continuous-time system is represented by:

$$\dot{x} = Ax \quad [230]$$

$$y = Cx \quad [231]$$

The unscaled vector of states, x, is a vector containing the n_x states.

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_{n_x} \end{bmatrix} \quad [232]$$

The unscaled vector of measurements, y, is a vector containing the n_y measurements.

$$\bar{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_{n_y} \end{bmatrix} \quad [233]$$

The scaled vector of states, $\tilde{x}$, and scaled vector of measurements, $\tilde{y}$, are calculated using the range of the NOCs of the system.

$$\tilde{x} = S_x(x - x_{min}) \quad [234]$$

$$\tilde{y} = S_y(y - y_{min}) \quad [235]$$

$$S_x = \begin{bmatrix} \frac{1}{x_{1,max}-x_{1,min}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{x_{n_x,max}-x_{n_x,min}} \end{bmatrix} \quad [236]$$

$$S_y = \begin{bmatrix} \frac{1}{y_{1,max}-y_{1,min}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{y_{n_y,max}-y_{n_y,min}} \end{bmatrix} \quad [237]$$

Equation 234 is rearranged to give $x = S_x^{-1}\tilde{x} + x_{min}$, with the time derivative of this equation being $\frac{dx}{dt} = S_x^{-1}\frac{d\tilde{x}}{dt}$. This is substituted into Equation 230 to give the linearized and scaled dynamic model equation:

$$\dot{\tilde{x}} = S_x A S_x^{-1}\tilde{x} + S_x A x_{min} \quad [238]$$

Equation 235 is rearranged to give $y = S_y^{-1}\tilde{y} + y_{min}$ and substituted into Equation 231 to give the linearized and scaled measurement equation:

$$\tilde{y} = S_y C S_x^{-1}\tilde{x} \quad [239]$$

APPENDIX B – STATE ESTIMATION ALGORITHMS

B.1. Extended Kalman filter algorithm

For a nonlinear hybrid system represented by:

$$\dot{x} = f(x, u, w, t) \qquad [240]$$

$$y_k = h_k(x_k, v_k)$$

Initialize the filter with an initial guess for the state estimation and estimation error covariance at time $k = 0$.

$$\hat{x}_0^+ = E(x_0) \qquad [241]$$

$$P_0^+ = E[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T]$$

For each successive timestep k:

Prediction step

1) Integrate the state estimate using the continuous process model Equation 240. Integration occurs from timestep $k-1$ to timestep k.

$$\dot{x} = f(\hat{x}_{k-1}^+, u, t)$$

The integration is initialized using the *a posteriori* state estimate from the previous time step, $\hat{x}_{k-1}^+$. The state estimate at time k at the end of the integration period represents the *a priori* state estimate, $\hat{x}_k^-$.

2) The covariance is integrated from timestep $k-1$ to k using Equation 13. This propagates the covariance through the continuous model.

$$\dot{P} = AP + PA^T + LQL^T$$

The integration is initialized using the *a posteriori* estimation error covariance from the previous time step, P_{k-1}^+. The estimation error covariance at time k at the end of the integration period represents the *a priori* estimation error covariance, P_k^-.

The system matrices in the ODE have been linearized around the *a posteriori* state estimate from the previous time step, $\hat{x}_{k-1}^+$.

$$A = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{k-1}^+}$$

$$L = \left.\frac{\partial f}{\partial w}\right|_{\hat{x}_{k-1}^+}$$

Update step

The standard KF equations from the discrete KF are used in the update step of the EKF.

3) The Kalman gain is calculated using Equation 8.

$$K_k = P_k^- H_K^T \left(H_k P_k^- H_k^T + M_k R_k M_k^T\right)^{-1}$$

Where H_k and M_k are the partial derivates evaluated at the *a priori* state estimate, $\hat{x}_k^-$.

$$H_k = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_k^-}$$

$$M_k = \left.\frac{\partial h}{\partial v}\right|_{\hat{x}_k^-}$$

4) The *a posteriori* state estimate is calculated using Equation 7.

$$\hat{x}_k^+ = \hat{x}_k^- + K_k\left(y_k - h(\hat{x}_k^-, v_k)\right)$$

5) The *a posteriori* covariance is calculated using Equation 9.

$$P_k^+ = (I - K_k H_k) P_k^- (I - K_k H_k)^T + K_k R_k K_k^T$$

(Simon, 2010)

B.2. Unscented Kalman filter algorithm

For a nonlinear hybrid system represented by:

$$\dot{x} = f(x, u, w, t) \qquad [242]$$

$$y_k = h_k(x_k, v_k)$$

Initialize the filter with an initial guess for the state estimation and estimation error covariance at time $k = 0$.

$$\hat{x}_0^+ = E(x_0) \qquad [243]$$

$$P_0^+ = E[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T]$$

For each successive timestep k:

Prediction step

1) Calculate the sigma points using Equation 17. The sigma points are generated based on the known mean, the *a posteriori* state estimate from the previous time step, and the known covariance, the *a posteriori* estimation error covariance from the previous time step, $\hat{x}_{k-1}^+$ and P_{k-1}^+, respectively.

$$\hat{x}_{k-1}^0 = \hat{x}_{k-1}^+$$

$$\hat{x}_{k-1}^{(i)} = \hat{x}_{k-1}^+ + \tilde{x}^{(i)} \qquad i = 1, \cdots, 2n$$

$$\tilde{x}^{(i)} = \left(\sqrt{(n+\kappa)P_{k-1}^+}\right)_i^T \qquad i = 1, \cdots, n$$

$$\tilde{x}^{(n+i)} = -\left(\sqrt{(n+\kappa)P_{k-1}^+}\right)_i^T \qquad i = 1, \cdots, n$$

$\kappa = n - 3$, as the state estimate distribution is assumed to be approximately Gaussian.

2) Propagate each of the sigma points from timestep $k-1$ to timestep k using the non-linear state transition equation, f, to obtain the i transformed state vectors, $\hat{x}_k^i$.

$$\dot{x} = f(x_{k-1}^i, u, t)$$

The integration is initialized with the *a posteriori* sigma points from the previous timestep x_{k-1}^i. The transformed sigma points at timestep k at the end of the integration period represents the *a priori* state estimate sigma points, $\hat{x}_k^{i\,-}$.

3) Calculate the approximate mean and covariance via weighted linear regression to obtain the *a priori* state estimate at timestep k, $\hat{x}_k^-$.

$$\hat{x}_k^- = W^i \sum_{i=1}^{2n} \hat{x}_k^i$$

$$W^0 = \frac{\kappa}{n+\kappa}$$

$$W^i = \frac{1}{2(n+\kappa)} \qquad i = 1, \cdots, 2n$$

4) Calculate the *a priori* estimation error covariance using weighted linear regression and adding in the process noise.

$$P_k^- = W^i \sum_{i=1}^{2n} (\hat{x}_k^i - \hat{x}_k^-)\,(\hat{x}_k^i - \hat{x}_k^-)^T + Q_{k-1}$$

Update step

5) Calculate the sigma points based on the *a priori* state estimate and covariance from the prediction step, $\hat{x}_k^-$ and P_k^-. Alternatively, the same sigma points from the time-update section can be used.
6) Transform each of the sigma points using the nonlinear measurement equation, h. This obtains i vectors, which represent the transformed predicted measurements, $\hat{y}_k^i$.

$$\hat{y}_k^i = h(\hat{x}_k^i, t_k)$$

7) Calculate the approximate mean of the predicted measurement at timestep k using weighted linear regression.

$$\hat{y}_k = W^i \sum_{i=1}^{2n} \hat{y}_k^i$$

8) Calculate the approximate covariance of the predicted measurements, adding in the measurement noise covariance.

$$P_{yy} = W^i \sum_{i=1}^{2n} (\hat{y}_k^i - \hat{y}_k)\,(\hat{y}_k^i - \hat{y}_k)^T + R_k$$

9) Calculate the cross covariance between the *a priori* state estimate and the predicted measurement at timestep k.

$$P_{xy} = W^i \sum_{i=1}^{2n} (\hat{x}_k^i - \hat{x}_k^-)\,(\hat{y}_k^i - \hat{y}_k)^T$$

10) Calculate the Kalman gain, the *a posteriori* state estimate, and the *a posteriori* estimation error covariance at timestep k using the standard KF update equations.

$$K_k = P_{xy} P_{yy}^{-1}$$

$$\hat{x}_k^+ = \hat{x}_k^- + K_k(y_k - \hat{y}_k)$$

$$P_k^+ = P_k^- - K_k P_{yy} K_k^T$$

(Simon, 2006)

B.3. Bootstrap particle filter algorithm

For a nonlinear hybrid system represented by:

$$\dot{x} = f(x, u, w, t) \qquad [\ 244\ ]$$

$$y_k = h_k(x_k, v_k)$$

Initialization:

Assume that the initial state estimate at time $k = 0$. can be accurately approximated by a Gaussian distribution with a mean equal to the guess for the initial state estimate and covariance equal to the guess for the initial state estimation error covariance. Randomly generate N particles on the basis of this pdf. The distribution of the initial state estimate has a mean and covariance:

$$\hat{x}_0^+ = E(x_0)$$

$$P_0^+ = E[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T]$$

The particles generated are denoted as:

$$\hat{x}_{0,i}^+ \quad (i = 1, \cdots, N)$$

For each successive timestep k:

Prediction step

1. Perform the prediction step by propagating each particle from the previous time step using the nonlinear function, f:

$$\dot{x} = f(x_{k-1,i}^+, u, t)$$

The integration is initialized with the *a posteriori* particles from the previous timestep $\hat{x}_{k-1,i}^+$. The transformed particles at timestep k at the end of the integration period represents the *a priori* particles, $\hat{x}_{k,i}^-$.

The process noise is incorporated within the prediction step by randomly sampling w_{k-1}^{i} from the known pdf of w_{k-1} and adding this to the nonlinear transformation of each particle.

Update step

1. Calculate the predicted measurements corresponding to each *a priori* particle using the function h_k:
$$\hat{y}_{k,i}^{-} = h_k(\hat{x}_{k,i}^{-})$$
2. The relative likelihood of each particle is calculated using Equation 27:
$$q_i = \frac{1}{(2\pi)^{\frac{n_y}{2}}|R|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}\left(y_k - \hat{y}_{k,i}^{-}\right)^T R^{-1}\left(y_k - \hat{y}_{k,i}^{-}\right)\right)$$
3. Scale the relative likelihoods to ensure the sum is equal to 1:
$$q_i = \frac{q_i}{\sum_{j=1}^{N} q_j}$$

Resampling step

1. Carry out resampling according to the low variance resampling algorithm presented in sub-section 2.1.5.3 to obtain the *a posteriori* particles $x_{k,i}^{+}$ $(i = 1, \cdots, N)$.
2. Calculate the mean and covariance of the new distribution using the algebraic mean and covariance of the *a posteriori* particles:
$$\hat{x}_k^{+} = \sum_{i=1}^{N} x_{k,i}^{+}$$
$$P_k^{+} = COV(x_{k,i}^{+})$$

Table B.1: Process model uncertainty and measurement uncertainty associated with each of the states

State	Unit	Process model uncertainty	Associated measurement	Measurement standard deviation	Conversion of measurement unit to state unit
$N_{C(B),X0}$	mol	$\pm 1000\ mols$			
$N_{C(B),XS}$	mol	$\pm 100\ mols$			
$N_{C(B),XS_2}$	mol	$\pm 100\ mols$			
$T_{C(B)}$	K	$\pm 1\ K$	$T_{C(B)}$	$\pm 10.36\ K$	
$N_{C(S),X0}$	mol	$\pm 100\ mols$			
$N_{C(S),XS}$	mol	$\pm 100\ mols$			
$N_{C(S),XS_2}$	mol	$\pm 1\ K$			
$T_{C(S)}$	K	$\pm 1\ K$			
$N_{C(R)}$	mol	$\pm 100\ mols$			
N_S	mol	$\pm 1000\ mols$	L_S	$\pm 0.05\ m$	$6 \times 10^5\ mols$
T_S	K	$\pm 1\ K$	T_S	$\pm 19.03\ K$	
N_M	mol	$\pm 1000\ mols$	L_M	$\pm 0.05\ m$	$8\times 10^5\ mols$
T_M	K	$\pm 1\ K$	T_M	$\pm 17.55\ K$	
$N_{G,A}$	mol	$\pm 100\ mols$			
$N_{G,R}$	mol	$\pm 100\ mols$	$C_{G,R}$	$\pm 0.1 \frac{mol}{m^3}$	$15\ mols$
T_G	K	$\pm 1\ K$	T_G	$\pm 2.2\ K$	
T_W	K	$\pm 1\ K$	T_W	$\pm 1\ K$	

APPENDIX C – SUBMERGED ARC FURNACE MODEL

Table C.1: SAF model symbols and subscripts

SAF model symbols		
C	Concentration	mol/m^3
F	Molar flow	mol/s
J	Molar flux	$mol/m^2 \cdot s$
L	Length	m
N	Molar amount	mol
P	Pressure	Pa
Q	Heat transfer	kW
r	Reaction rate	mol/s
R	Resistivity	V^2/kW
T	Temperature	K
V	Volume	m^3
v	Volume transfer	m^3/s
Subscripts		
A	Air component	
$C(B)$	Bulk concentrate zone	
$C(R)$	Reaction gases in the concentrate zone	
$C(S)$	Smelting concentrate zone	
G	Furnace freeboard zone	
M	Matte zone	
R	Reaction gas component	
S	Slag zone	
W	Copper cooling unit zone	

XO	Slag component
XS	Matte component
XS_2	Sulphurized matte component

Table C.2: Initial values for the state variables

State variable	Initial value	Unit
$N_{C(B),X0}$	8.894×10^5	mol
$N_{C(B),XS}$	5.536×10^5	mol
$N_{C(B),XS_2}$	1035	mol
$T_{C(B)}$	1036	K
$N_{C(S),X0}$	7.714×10^5	mol
$N_{C(S),XS}$	5.593×10^5	mol
$N_{C(S),XS_2}$	1.069	mol
$T_{C(S)}$	1379	K
$N_{C(R)}$	555	mol
N_S	1.113×10^7	mol
T_S	1903	K
N_M	6.603×10^6	mol
T_M	1755	K
$N_{G,R}$	1082.570	mol
$N_{G,A}$	702.990	mol
T_G	1024	K
T_W	311	K

Table C.3: Model parameters

Parameter	Symbol	Value	Unit
Molar masses:			
Freeboard gas molar mass	M_G	$29 \cdot 10^{-3}$	$kg \cdot mol^{-1}$

Lumped slag molar mass	M_{XO}	$72 \cdot 10^{-3}$	$kg \cdot mol^{-1}$
Lumped matte molar mass	M_{XS}	$88 \cdot 10^{-3}$	$kg \cdot mol^{-1}$
Sulphurised matte molar mass	M_{XS_2}	$120 \cdot 10^{-3}$	$kg \cdot mol^{-1}$
Densities			
Bulk concentrate density	ρ_C	1600	$kg \cdot m^{-3}$
Liquid slag density	ρ_S	2960	$kg \cdot m^{-3}$
Liquid matte density	ρ_M	4800	$kg \cdot m^{-3}$
Heat of fusion:			
Concentrate heat of fusion	λ_C	133	$kJ \cdot mol^{-1}$
Heat capacities:			
Concentrate heat capacity	$c_{P,C}$	$75 \cdot 10^{-3}$	$kJ \cdot mol^{-1} \cdot K^{-1}$
Liquid slag heat capacity	$c_{P,S}$	$99 \cdot 10^{-3}$	$kJ \cdot mol^{-1} \cdot K^{-1}$
Liquid matte heat capacity	$c_{P,M}$	$78 \cdot 10^{-3}$	$kJ \cdot mol^{-1} \cdot K^{-1}$
Freeboard heat capacity	$c_{P,G}$	$30 \cdot 10^{-3}$	$kJ \cdot mol^{-1} \cdot K^{-1}$
Cooling water heat capacity	$c_{P,W}$	$75 \cdot 10^{-3}$	$kJ \cdot mol^{-1} \cdot K^{-1}$
Reaction activation energies:			
Desulphurization reaction activation energy	$E_{A,F}$	$150 \cdot 10^3$	$J \cdot mol^{-1}$
Electrode oxidation activation energy	$E_{A,C}$	$120 \cdot 10^3$	$J \cdot mol^{-1}$
Rate constants:			
Concentrate mixing constant	k_v	$2 \cdot 10^{-4}$	s^{-1}
Freeboard-to-atmosphere pressure constant	k_{PR}	7	$mol \cdot Pa^{-1} \cdot s^{-1}$
Freeboard extraction pressure constant	k_{PE}	3	$mol \cdot Pa^{-1} \cdot s^{-1}$
Desulphurization rate constant	k_F	$1 \cdot 10^5$	s^{-1}
Electrode oxidation rate constant	k_C	$1.25 \cdot 10^4$	$mol \cdot s^{-1}$

Packed bed reactor flux constant	k_{PBR}	10^{-8}	$mol \cdot m^{-1} \cdot Pa^{-1} \cdot s^{-1}$
Channelling flux constant	k_{Ch}	$2 \cdot 10^{-6}$	$mol \cdot m^{-1} \cdot Pa^{-1} \cdot s^{-1}$
Furnace dimensions:			
Bath area	A	300	m^2
Bath perimeter	ρ_{SAF}	80	m
Freeboard volume	V_G	150	m^3
Heat generation/transfer constants:			
Electrode voltage	$V_{electrode}$	120	V
Joule heating constant	α	$3 \cdot 10^{-4}$	$V^2 \cdot kW^{-1} \cdot K^{-1}$
Reference slag resistance	R_0	0.21	$V^2 \cdot kW^{-1}$
Reference slag temperature	$T_{S,0}$	1900	K
Heat transfer coefficient between smelting and bulk concentrate	$h_{C(S):C(B)}$	$2.75 \cdot 10^{-2}$	$kW \cdot m^{-2} \cdot K^{-1}$
Heat transfer coefficient between freeboard and bulk concentrate	$h_{G:C(B)}$	$5.5 \cdot 10^{-2}$	$kW \cdot m^{-2} \cdot K^{-1}$
Heat transfer coefficient between slag and smelting concentrate	$h_{S:C(S)}$	$3.1 \cdot 10^{-1}$	$kW \cdot m^{-2} \cdot K^{-1}$
Heat transfer coefficient between liquid matte and slag	$h_{M:S}$	$8.5 \cdot 10^{-2}$	$kW \cdot m^{-2} \cdot K^{-1}$
Heat transfer coefficient between cooling units and liquid slag	$h_{W:S}$	$7 \cdot 10^{-3}$	$kW \cdot m^{-2} \cdot K^{-1}$

Heat transfer coefficient between cooling units and liquid matte	$h_{W:M}$	$2.7 \cdot 10^{-2}$	$kW \cdot m^{-2} \cdot K^{-1}$
Cooling water constants:			
Cooling water flowrate	F_W	2400	$mol \cdot s^{-1}$
Moles of cooling water	N_W	10^4	mol
Cooling water inlet temperature	$T_{W,0}$	300	K
Charging and tapping constants:			
Flowrate of the charged concentrate	$F_{charge,const.}$	410	mol/s
Flowrate of the tapped slag	$F_{tap,S,const.}$	400	mol/s
Flowrate of the tapped matte	$F_{tap,M,const.}$	100	mol/s
Lower slag bed height limit	$L_{S,lower}$	0.1	m
Upper slag bed height limit	$L_{S,upper}$	0.2	m
Lower matte bed height limit	$L_{S,lower}$	0.04	m
Upper matte bed height limit	$L_{S,upper}$	0.08	m
Other constants:			
Universal gas constant	R	8.314	$J \cdot mol^{-1} \cdot K^{-1}$
Gravitational acceleration constant	g	9.81	$m \cdot s^{-2}$
Concentrate bed void fraction	ε_C	0.4	$-$
Atmospheric pressure	P_{atm}	8101325	Pa
Extraction pressure	P_{ext}	8101315	Pa
Concentrate charging temperature	T_{charge}	700	K
Concentrate melting temperature	T_{melt}	1500	K
Atmospheric temperature	T_{atm}	300	K

Table C.4: Standard deviation of the measurement noise for each of the measurements

Measurement	Standard deviation	Unit	Justification
$T_{C(B)}$	10	K	The temperature of the bulk concentrate is not usually measured. Based on the noise associated with the other temperature measurements, the noise is selected as 1% of the NOC.
L_S	0.05	m	The slag height is measured by on-plant personnel by hand using a sounding pole. Measurement noise is large due to human error.
T_S	19	K	The temperature of the slag is measured using a pyrometer. These pyrometers usually have an accuracy of 1% of the NOC of the slag.
L_M	0.05	m	The matte height is measured by on-plant personnel by hand using a sounding pole. Measurement noise is large due to human error.
T_M	18	K	The temperature of the matte is measured using a pyrometer with accuracy of 1% of the NOC.
$C_{G,R}$	0.1	$\frac{mol}{m^3}$	Based on industrial values, it can be safely assumed that the noise on the concentration of reaction gases measurement is 5% of the NOC.
P_G	2	Pa	The pressure in the freeboard is measured using a pressure transmitter. The relative accuracy of the pressure transmitter is usually between $\pm 2\ Pa.$
T_G	2.2	K	Instrumentation information on the temperature readings of the freeboard in industrial application indicate that the uncertainty is $2.2\ K$.
T_W	1	K	Instrumentation information on the temperature readings of the cooling water in

			industrial application indicate that the uncertainty is $\pm 1\ K$.

APPENDIX D – MATLAB CODE

D.1. Implementation of the state estimation algorithms

The main script is presented for numerical implementation of the state estimation algorithms, the EKF, UKF, PF, and MHE, in MATLAB, to estimate the states of the SAF smelting process.

```
%% Generate the synthetic measurement data and ground truth values
tDuration = 1.5;  % Measurement simulation time, days
seed = 1;         % seed the rng to ensure repeatability of measurements
percentage = 0;   % percentage of stochastic variation in Fcharge
faultindex = 0;   % for obtaining faulty measurements
                  % = 1; Cooling water flowrate step change at t = 1 day
                  % = 2; Pext step change at t = 1 day
                  % = 3; Concentrate composition step change at t = 1 day
                  % = 4; Blowback fault at t = 1 day
[y, meas_info, x_ground_truth, t_ground_truth] = SAF_measurements(tDuration, ...
                                        faultindex, percentage, seed);
% Measurement data stored in y matrix, the rows represent the ny measured variables and the
% columns are the measurements in time simulated for tDuration
% The measurement field, variance, and sampling rate is stored in meas_info
% Ground truth values of the state variables stored in x_ground_truth with corresponding
% timepoints in t_ground_truth
ny = size(y,1);           % Number of measured variables
N_meas = meas_info.TCB.T;  % s, Measurement period
                        % This defines how often the state estimate is updated


%% Define the state estimation tuning parameters
% Initialization tuning parameters
[N,T] = variableInitialization;         % State variable initial values
x_0 = state2ode(N,T);                   % Convert structure to vector
percentage_error = 1;                   % Percentage initialization error
xhat_0 = x_0+x_0*(percentage_error/100); % Initial state estimates
nx = length(xhat_0);                    % Number of states being estimated
```

```
P0 = eye(length(xhat_0)).*((x_0-xhat_0)*(x_0-xhat_0)'); % Initial estimation error covariance


% Noise covariance matrices
for i = 1:length(meas_info.fields)
    current = meas_info.(meas_info.fields{i});
    MeasNoise(i) = sqrt(current.var);          % Measurement noise standard deviation
end
R = eye(ny).* MeasNoise.^2;                    % Measurement noise covariance matrix


% Select the SAF model
% Model type = 0: original SAF model, uses ode15s in the integration step
% Model type = 1: singular perturbation SAF model, uses RK4 in the integration step
model_type = 0;


% Larger process noise is selected for objective 2: implementing the state estimation algorithms
if model_type == 0
    ProcNoise_objective2 = [100,10,10,0.1,10,10,1,1,10,100,0.1,100,0.1,0.01,0.01,0.01,0.1];
    % process noise standard deviation
end
if model_type == 1
    % Larger process noise associated with states with fast dynamics (x14, x15, x16) in the
    % singular perturbation model
    ProcNoise_objective2 = [100,10,10,0.1,10,10,0.1,0.1,10,100,0.1,100,0.1,10,10,0.1,0.1];
end
ProcNoise = ProcNoise_objective2;


% Smaller process noise selected for objective 3: performing model-based fault detection using
% the state estimators
if faultindex > 0
    ProcNoise_objective3 = 0.01.*ProcNoise_objective2;
    % Process noise for objective 3 is 1% of the process noise used in objective 2
    ProcNoise = ProcNoise_objective3;
```

```
end
Q = eye(nx).*ProcNoise.^2; % Process noise covariance matrix

% Particle filter
n = 80; % number of particles

% Moving horizon estimator
horizon_length = 4; % horizon length

%% State estimation
estimationlength = size(ny,2); % Specify number of state estimates being estimated
                               % For this example, the number of state estimates is equal
                               % to the number of measurements obtained
% Extended Kalman filter
[xhat_EKF, P_EKF, inn_EKF] = EKF(estimationlength,N_meas,xhat_0,P0,R,Q,y,model_type);
% Unscented Kalman filter
[xhat_UKF, P_UKF, inn_UKF] = UKF(estimationlength,N_meas,xhat_0,P0,R,Q,y,model_type);
% Particle filter
[xhat_PF, P_PF, inn_PF,vhat_matrix] = PF(estimationlength, N_meas, xhat_0, P0 , R, Q,  y, ...
                                         n, model_type);
% Moving Horizon Estimator
[xhat_MHE, PArray, innArray] = MHE(estimationlength, xhat_0, P0, R, Q, y, ...
                                   N_meas, horizon_length, model_type);
```

The following functions are the EKF, UKF, PF, and MHE algorithms.

Function inputs	
EKF, UKF, PF and MHE functions	
estimationlength	Specify the length of the state estimation period by the number of state estimates obtained at the end of the simulation.
N_meas	Specify the measurement sampling rate. This dictates the timepoints at which state estimates are obtained at. For this study, the measurement sampling rate is $10s$, thus, a state estimate will be obtained every $10s$.

xhat_0	Initial state estimates.
P0	Initial state estimation error covariance matrix.
R	Measurement noise covariance matrix.
Q	Process noise covariance matrix.
y	Matrix of measurement data. Number of rows is equal to number of measured variables. Number of columns is equal to number of timepoints at which measurement data was generated for.
model_type	Select the model type. Model type 0 is the original SAF model. Model type 1 is the SAF model with singular perturbation.
PF function	
n	Specify the number of particles in the PF.
MHE function	
horizonlength	Specify the horizon length used in the MHE algorithm.
Function outputs	
EKF, UKF, PF, and MHE	
xhatArray	Matrix containing the state estimates. Number of rows is equal to number of state variables. Number of columns is the number of state estimates obtained, this is equal to the estimationlength.
PArray	Matrix containing the diagonal elements of the estimation error covariance matrix corresponding to the estimation error of the state estimates obtained at each timepoint. Number of rows is equal to the number of state variables. Number of columns is equivalent to the estimationlength.
innArray	Matrix containing the innovation/residual term at each timepoint. Number of rows is equivalent to the number of measured variables. Number of columns is equivalent to the estimationlength-1, as there is no innovation term at timepoint 0 as there is no measurement available at timepoint 0.
PF	
vhat_matrix	Three-dimensional matrix containing the innovations for each of the particles at each timestep. The number of elements in the first dimension is equal the number of measured variables. The number of elements in the second dimension is equal to the number of particles. The number of elements in the third dimension is equal to the estimationlength-1.

```
%% EKF algorithm

function [xhatArray, PArray, innArray] = EKF(estimationlength,N_meas,xhat_0,P0,R,Q,y,model_type)

  % Generate the storage matrices

  xhatArray = [xhat_0]; % Array of state estimates

  PArray = [diag(P0)];  % Array of state estimate error covariances (vector form)

  innArray = [];        % Array of innovation terms


  % Initialize the SAF model

  [mode, options, p] = SAF_model_initialization(estimationlength);


  % Derive the system Jacobians

  x = sym('x',[17 1]) ; % Generate state variable symbols and store them in the vector 'x'

  syms c m s;           % Generate the input symbols and store them in the structure input

  input.c = c;          % Furnace charging input

  input.m = m;          % Matte tapping input

  input.s = s;          % Slag tapping input

  input.chargetime = mode.chargetime; % Charging timepoints

  input.stoptime = mode.stoptime;     % Halting of charge timepoints

  % State transition Jacobian

  f = furnaceModelODEs(x,p,input,0,model_type);

  dfdx = jacobian(f,x);

  % Measurement Jacobian

  h = hfunc(x',p);

  dhdx = jacobian(h,x);


  % Initialize the state estimator

  xhatpos = xhat_0;              % initial state estimate

  P0_vec = reshape(P0, [], 1);    % initial state estimation error covariance in vector form

  Ppos = P0_vec;


 % Calculate the state estimate at each timestep

 timesteps = 0 : N_meas : estimationlength*N_meas; % Define the time points in seconds
```

```
                                % at which the state estimates are
                                % calculated at based on the measurement
                                % sampling rate 'N_meas'
for i = 1 : estimationlength
    % PREDICTION STEP
    % Calculate the a priori state estimate from the model prediction
    tStart = timesteps(i); % Define the start of the integration period
    tEnd = timesteps(i+1); % Define the end of the integration period
    [xhatneg, mode] = SAF_prediction(tStart, tEnd, xhatpos, xhatpos, p, mode, options, ...
                            model_type);  % Prior state estimate

    % Calculate the prior covariance matrix
    A = linA(x, c, m, s, dfdx, xhatneg, mode); % Transition matrix linearized around
                                % the prior state estimate
    solP = ode45(@(t,P_vec) P_ODE(A, P_vec, Q), timesteps(i:i+1), Ppos); % Propagate the
                                                % covariance
    Pneg = solP.y(:,end);                   % Prior covariance (vector form)
    Pnegmatrix = reshape(Pneg, sqrt(length(Pneg)), []); % Prior covariance (matrix form)

    % UPDATE STEP
    H = double(subs(dhdx,x,xhatneg));       % Linearized measurement matrix
    K = Pnegmatrix*H'*inv(H*Pnegmatrix*H'+R); % Kalman gain
    yhat = hfunc(xhatneg,p);                % Measurement prediction
    inn = y(:,i)-yhat;                      % Innovation term
    xhatpos = xhatneg + K*(inn);            % Posterior state estimate
    Pposmatrix = (eye(size(Pnegmatrix))-K*H)*Pnegmatrix*(eye(size(Pnegmatrix))-K*H)'+K*R*K';
                            % Posterior estimation error covariance (matrix form)
    Ppos = reshape(Pposmatrix, [], 1); % Posterior estimation error covariance (vector form)
    chol(Pposmatrix); % Check positive definiteness of posterior estimation error

    % Store the state estimate, estimation error covariance, and the innovation term
    xhatArray = [xhatArray xhatpos];
```

```
        PArray = [PArray diag(Pposmatrix)];
        innArray = [innArray inn];
    end
end

% 'linA' computes the linearized state matrix, A, linearized around the prior state estimate
% 'xhatneg'
function A = linA(x, c, m, s, dfdx, xhatneg, mode)
    A = subs(dfdx,x,xhatneg);
    A = subs(A,[c, m s], [mode.c,mode.m,mode.s]);
    A = double(A);
end

% 'P_ODE' computes the rate of change in the estimation error covariance matrix for the
% prediction step of the EKF covariance propagation
function dpdt = P_ODE(A, P_vec, Q)
        P_mat = reshape(P_vec, sqrt(length(P_vec)), []); % reshape the P vector into a matrix
        dpdtmat = A*P_mat + P_mat*A' + Q;               % rate of change of P wrt time
        dpdt = reshape(dpdtmat, [], 1);                 % reshape the dpdt matrix into a vector
end
```

```
%% UKF algorithm
function [xhatArray, PArray, innArray] = UKF(estimationlength,N_meas,xhat_0,P0,R,Q,y,model_type)
    % Generate the storage matrices
    xhatArray = [xhat_0]; % Array of state estimates
    PArray = [diag(P0)]; % Array of state estimate error covariances (vector form)
    innArray = [];       % Array of innovation terms

    % Initialize the SAF model
    [mode, options, p] = SAF_model_initialization(estimationlength);

    % Initialize the state estimator
```

```
xhatpos = xhat_0;    % Initial state estimate
Ppos = P0;           % Initial state estimation error covariance
nx = length(xhat_0); % Number of state variables
ny = size(y,1);      % Number of measurement variables


% Generate the sigma point weights
k = nx-3;
Ws = [];
Ws(1) = k/(nx+k);
for i = 2: 2*nx+1
    Ws(i) = 1/(2*(nx+k));
end


% Calculate the state estimate at each timestep
timesteps = 0 : N_meas : estimationlength*N_meas; % Define the time points in seconds
                                                  % at which the state estimates are
                                                  % calculated at based on the measurement
                                                  % sampling rate (N_meas)
 for j = 1 : estimationlength
    % PREDICTION STEP
    % Generate the sigma points for the prediction step
    root_pred = chol((nx+k)*Ppos);
    xsigma(:,1) = xhatpos; % The first sigma point is equal to the posterior state estimate
                           % from the previous timestep 'xhatpos'
    for i = 2 : nx + 1
        xsigma(:,i) = xhatpos + root_pred(i-1,:)';
        xsigma(:,i+nx) = xhatpos - root_pred(i-1,:)';
    end


    % Transform the sigma points in the prediction step
    xsimganeg = [];
    for i = 1 : 2 * nx + 1
```

```
    tStart = timesteps(j); % Define the start of the integration period
    tEnd = timesteps(j+1); % Define the end of the integration period
    [xsigmaneg(:,i), mode] = SAF_prediction(tStart, tEnd, xsigma(:,i), xhatpos, p, ...
                        mode, options, model_type);  % SAF model prediction
end

% Calculate the prior state estimate from the weighted sum of the prior sigma points
xhatneg = zeros(nx,1);
for i = 1: 2*nx + 1
    xhatneg = xhatneg + Ws(i) * xsigmaneg(:, i); % Prior state estimate
end

% Calculate prior estimation error covariance
Pneg = zeros(nx,nx);
for i = 1 : 2*nx+1
    Pneg = Pneg + Ws(i) * (xsigmaneg(:,i) - xhatneg) * (xsigmaneg(:,i) - xhatneg)';
end
Pneg = Pneg + Q; % Prior state estimation error covariance

% Calculate the innovation term
yhat = hfunc(xhatneg,p);   % Measurement prediction
inn = y(:,j)-yhat;         % Innovation
innArray = [innArray inn]; % Store the innovation

% UPDATE STEP
% Generate the sigma points for the update step
root_update = chol(nx*Pneg);
xsigma_update(:,1) = xhatneg; % The first sigma point is equal to the prior state estimate
                        % 'xhatneg'
for i = 2 : nx+1
    xsigma_update(:,i) = xhatneg + root_update(i-1,:)';
    xsigma_update(:,i+nx) = xhatneg - root_update(i-1,:)';
```

```
end

% Transform the sigma points using the measurement function 'hfunc'
ysigma = [];
for i = 1 : 2*nx + 1
    ysigma(:,i) = hfunc(xsigma_update(:,i),p); % Predicted measurements
end

% Calculate the approximate mean of the predicted measurements using the weighted
% sum of the transformed sigma points
yhat = zeros(ny,1);
for i = 1 : 2*nx + 1
    yhat = yhat + Ws(i) * ysigma(:, i);
end

% Calculate the approximate covariance of the predicted measurements
Py = 0;
Pxy = zeros(nx,1);
 for i = 1 : 2*nx+1
    Py = Py + Ws(i) * (ysigma(:,i) - yhat) * (ysigma(:,i) - yhat)';
    Pxy = Pxy + Ws(i) * (xsigma_update(:,i) - xhatneg) * (ysigma(:,i) - yhat)';
 end
Py = Py + R; % Add the measurement noise covariance

% Kalman update
K = Pxy * inv(Py);                    % Kalman gain
xhatpos = xhatneg + K * (y(:,j) - yhat); % Posterior state estimate
Ppos = Pneg - K * Py * K';            % Posterior estimation error covariance
chol(Ppos);                           % Check the positive definiteness

% Store the final state estimate and estimation error covariance
xhatArray = [xhatArray xhatpos];
```

```
        PArray = [PArray diag(Ppos)];
    end
end
```

```
%% PF algorithm
function [xhatArray, PArray, innArray, vhat_matrix] =  ...
                                PF(estimationlength,N_meas,xhat_0,P0,R,Q,y,n,model_type)
    % Generate the storage matrices
    xhatArray = [xhat_0]; % Array of state estimates
    PArray = [diag(P0)]; % Array of state estimate error covariances (vector form)
    innArray = [];       % Array of innovations
    vhat_matrix = [];    % 3D matrix of innovations of each particle at each timestep

    % Initialize the SAF model
    [mode, options, p] = SAF_model_initialization(estimationlength);

    % Initialize PF by generating n particles on the basis of the initial pdf
    xhatpos = [];
    for i = 1 : n
        xhatpos(:,i) = xhat_0 + sqrt(P0) * randn(length(xhat_0),1);
    end

    % Calculate the state estimate at each timestep
    timesteps = 0 : N_meas : estimationlength*N_meas; % Define the time points in seconds
                                        % at which the state estimates are
                                        % calculated at based on the measurement
                                        % sampling rate (N_meas)
    for j = 1 : estimationlength
        % PREDICTION STEP
        xhatneg = [];
        vhat = [];
```

```
% Transform each of the particles through the process model prediction
for i = 1 : n
    tStart = timesteps(j); % Define the start of the integration period
    tEnd = timesteps(j+1); % Define the end of the integration period
    [xhatneg(:,i), mode] = SAF_prediction(tStart, tEnd, xhatpos(:,i), ...
                xhatArray(:,j), p, mode, options,model_type);  % Model prediction
    xhatneg(:,i) = xhatneg(:,i)+ sqrt(Q)' * randn(length(xhat_0),1);
    % Add random process noise to the model predictions of the particles
    yhat(:,i) = hfunc(xhatneg(:,i),p);   % Obtain the predicted measurement
    vhat(:,i) = abs(y(:,j) - yhat(:,i)); % Innovation of individual particles
end
vhat_matrix(:,:,j) = vhat;     % Store the particle innovations
xhatmean_neg = mean(xhatneg'); % Prior state estimate is the mean of the prior particles
% Calculate and store the innovation
inn = y(:,j)-hfunc(xhatmean_neg,p);
innArray = [innArray inn];

% UPDATE STEP
% Calculate the relative likelihood for each of the n particles
qsum = 0;
q_Arr = [];
for i = 1 : n
    q = (( inv((det(R)^0.5) * (2*pi)^(length(yhat)/2)) ) * ...
        ( exp((-vhat(:,i)'*inv(R)*vhat(:,i))*0.5) ));
    q_Arr = [q_Arr q];  % Store the relative likelihood of each particle
    qsum = qsum + q;    % Sum of the relative likelihoods of the particles
end

% Normalize the relative likelihood
qnorm = [];
for i = 1 : n
    qnorm(:,i) = q_Arr(:,i)/qsum;
```

```
end


% Low Variance Resampling
r = rand*(1/n);     % Generate a random number between 0->1/n
w = qnorm(:,1);     % Initialize with the first particles likelihood
z = 1;              % Tracks the particle number in the original sample (xhatneg)
q = 1;              % Tracks the elements in the new sample (xhatpos)

for m = 1:n
   U = r + (m - 1)/n;      % Ensures we move through the particles systematically
   % While loop adds the relative likelihoods until the cumulative relative likelihood
   % is greater than U
   while U > w
      z = z + 1;          % z keeps track of the particle number
      w = w + qnorm(:,z); % w is the cumulative likelihood
   end
   % Once w > U, the particle z from the previous sample (xhatneg) gets added to the
   % new sample (xhatpos)
   xhatpos(:,q) = xhatneg(:,z);
   q = q + 1; % Move to the next element in the new sample
end


% Calculate the mean and covariance of the particles
xhatmean = mean(xhatpos'); % Mean of the particles represents the posterior state
                           % estimate
P = diag(var(xhatpos'));   % Covariance of the particles represents the posterior state
                           % estimation error covariance
chol(P);                   % Check the positive definiteness


% Store the final state estimate, estimation error covariance, and innovation term
xhatArray = [xhatArray xhatmean'];
PArray = [PArray diag(P)];
```

```
    end
end
```

```
%% MHE algorithm
function [xhatArray, PArray, innArray] = ...
                        MHE(estimationlength,xhat_0,P0,R,Q,y,N_meas,horizonlength,model_type)
    % Generate the storage matrices
    xhatArray = []; % Array of state estimates
    PArray = [];    % Array of state estimation error covariances
    innArray = [];  % Array of innovation terms
    xk_matrix = []; % Matrix of state estimate trajectories at each timestep

    % Initialize the SAF model
    [mode, options, p] = SAF_model_initialization(estimationlength);

    % Initialize the MHE
    % 1. Initialize the EKF
    % The EKF is used to approximate the estimation error covariance, Pplus, used in the
    % arrival cost calculation in the MHE optimization
    % Derive the system Jacobians
    x = sym('x',[17 1]) ; % Generate state variable symbols and store them in the vector 'x'
    syms c m s;          % Generate the input symbols and store them in the structure input
    input.c = c;
    input.m = m;
    input.s = s;
    input.chargetime = mode.chargetime;
    input.stoptime = mode.stoptime;
    % State transition Jacobian
    f = furnaceModelODEs(x,p,input,0,0);
    dfdx = jacobian(f,x);
    % Measurement Jacobian
    h = hfunc(x',p);
```

```
dhdx = jacobian(h,x);


% 2. Create structure of parameters for the MHE objective function
p_MHE.n = length(xhat_0);     % number of states
p_MHE.m = horizonlength;      % horizon length
p_MHE.Q_inv = inv(Q);         % inverse of the process noise covariance
p_MHE.R_inv = inv(R);         % inverse of the measurement noise covariance
p_MHE.yk = y;                 % matrix of measurements
p_MHE.N_meas = N_meas;        % measurement sampling rate
p_MHE.p = p;                  % SAF model parameters
p_MHE.options = options;      % SAF model options
p_MHE.mode = mode;            % SAF modes
p_MHE.modeltype = model_type; % SAF model type
p_MHE.xprev = xhat_0;         % Initialize the model prediction with xhat0


% 3. Initialize the optimization routine
% Create vector of initial guesses of the state trajectory in the first
% horizon using the EKF state estimates,
% Pass the horizon length as the estimationlength to the EKF
% Always use original model (model_type = 0) for EKF as the computational requirements of the
% EKF are small
[xk_guess, P_guess] = EKF(p_MHE.m,N_meas,xhat_0,P0,R,Q,y,0); % EKF state estimates
xk_guess = xk_guess(:,2:end);            % Remove the state estimate at time zero
p_MHE.xinit = xk_guess(:,1);             % Matrix of initial state estimates in first horizon
xk_guess = reshape(xk_guess,[],1);       % Reshape to vector of initial guesses
% Initialize the EKF used to approximate the arrival cost with the state estimate and
% estimation error covariance EKF guesses
Ppos = eye(length(xk_guess)).*P_guess(:,2); % Set the initial EKF estimation error covariance
xk_initial = xk_guess(:,1);              % Set the initial EKF state estimate



for i = 0:1:estimationlength-p_MHE.m
```

```
% Create the timesteps used in the prediction step
p_MHE.timesteps = (i*N_meas):N_meas:(p_MHE.m*N_meas+i*N_meas);


% EKF ARRIVAL COST APPROXIMATION
% Use the EKF to calculate Ppos in the arrival cost approximation:
% Prediction step of the EKF: Calculate the prior estimation error covariance matrix
A = linA(x, c, m, s, dfdx, xk_initial, mode); % Transition matrix linearized around the
                                              % first state estimate in the previous
                                              % horizon
solP = ode45(@(t,P_vec) P_ODE(A, P_vec, Q), p_MHE.timesteps(1:2), reshape(Ppos,[],1));
% Propagate the estimation error from the first timestep in the previous horizon
% to the first timestep in the current horizon
Pneg = solP.y(:,end);                         % Prior covariance (vector form)
Pnegmatrix = reshape(Pneg, sqrt(length(Pneg)), []); % Prior covariance (matrix form)
% Update step of EKF: Update the estimation error covariance
H = double(subs(dhdx,x, xk_guess(1:p_MHE.n))); % Linearized measurement matrix
K = Pnegmatrix*H'*inv(H*Pnegmatrix*H'+R);      % Kalman gain
Ppos = (eye(size(Pnegmatrix))-K*H)*Pnegmatrix*(eye(size(Pnegmatrix))-K*H)'+K*R*K';
% 'Ppos' is the estimation error covariance associated with first state estimate in
% the current horizon
p_MHE.P_inv = inv(Ppos); % Use the inverse of Ppos in the optimization routine in the
                         % arrival cost approximation
PArray = [PArray diag(Ppos)]; % Store the estimation error covariance
chol(Ppos);                   % Check Ppos is positive semidefinite


% OPTIMIZATION
% Perform the optimization using fminunc
opts = optimset('MaxIter', 10000,'MaxFunEvals', 10000, 'Algorithm', 'sqp');
[xk, fval] = fminunc(@(xk) optimfunc(xk, p_MHE), xk_guess, opts);


% STORE THE RESULTS
xk = reshape(xk,p_MHE.n,p_MHE.m);  % Reshape the vector to a matrix representing the
```

```
                        % 'n' state estimates over the horizon length 'm'
xk_matrix(i+1,:,:) = xk;          % Store the matrix of state trajectories
xk_final =  xk(:,end);            % Use the last state estimate in the current horizon
                        % as the final state estimate (the output from the
                        % MHE function)
xhatArray = [xhatArray xk_final];  % Store the last state estimate
% Store the innovation term
yhat = hfunc(xk_final, p_MHE.p); % Prediction of last measurement in horizon
measurement = p_MHE.yk(:,p_MHE.timesteps(end)/p_MHE.N_meas); % Measurement at last
                                        % timepoint in the
                                        % current horizon
inn = measurement - yhat; % Innovation term
innArray = [innArray inn];

% INITIATE THE FILTER FOR THE NEXT TIMESTEP
xk_initial = xk(:,1); % Store the first state estimate in current horizon
                % used to initialize the EKF arrival cost approximation
 % Obtain the mode of first state estimate in the current horizon for the EKF
[~,mode] = SAF_prediction(p_MHE.timesteps(1),p_MHE.timesteps(2)+N_meas, ...
        xk_initial, xk_initial, p_MHE.p, p_MHE.mode, p_MHE.options, p_MHE.modeltype);
p_MHE.mode = mode; % Use this mode in linearization of A in next horizon

% Create the vector of guesses for the next state trajectory
% Obtain the last guess in the trajectory from the model prediction
% Always use original model (model type 0) for this guess
xk_guess_end = SAF_prediction(p_MHE.timesteps(end),p_MHE.timesteps(end)+N_meas, ...
                    xk_final, xk_final, p_MHE.p, p_MHE.mode, p_MHE.options, 0);
% Delete the first state estimate in the current horizon
xk(:,1) = [];
% Form the trajectory of guesses in the next horizon from the state estimate trajectory
% in the current horizon and the last guess 'xk_guess_end' from the model prediction
xk_guess = [xk xk_guess_end(:,end)];
```

```
        p_MHE.xinit = xk_guess(:,1);
        xk_guess = reshape(xk_guess, [], 1); % Vector of initial guesses for optimization
        p_MHE.xprev = xk_initial; % Initialize the model prediction in the next optimization
    end
end

% Objective function of the MHE algorithm
function J = optimfunc(xk_vec, p_MHE)
    % Convert the vector of state estimates, xk, to a matrix
    xk = reshape(xk_vec,p_MHE.n,p_MHE.m);

    % Calculate the sum of the model prediction errors in the current horizon
    sum1 = 0;
    xpred_prev = p_MHE.xprev; % Initialize the first model prediction with the first state
                              % estimate from the previous horizon
    for i = 1:1:p_MHE.m
        [fx_k, mode] = SAF_prediction(p_MHE.timesteps(i),p_MHE.timesteps(i+1), ...
                xpred_prev, xpred_prev, p_MHE.p, p_MHE.mode, p_MHE.options, p_MHE.modeltype);
        p_MHE.mode = mode;
        diff1 = xk(:,i)-fx_k(:,end); % Difference between the model prediction for the state and
                                     % the state estimate
        normdiff1 = diff1'*p_MHE.Q_inv*diff1; % Weight the difference with the process noise
        sum1 = sum1 + normdiff1; % Sum of the model prediction errors in the horizon
        xpred_prev = xk(:,i); % Initialize the next model prediction with the previous estimate
    end

    % Calculate the sum of measurement errors in the current horizon
    sum2 = 0;
    for i = 1:1:p_MHE.m
        hx_k = hfunc(xk(:,i), p_MHE.p); % Predicted measurement
        measurementyk = p_MHE.yk(:,p_MHE.timesteps(i+1)/p_MHE.N_meas); % True measurement
        diff2 = p_MHE.yk(:,p_MHE.timesteps(i+1)/p_MHE.N_meas) - hx_k; % Difference between the
```

```
                                        % predicted and true
                                        % measurements
    normdiff2 = diff2'*p_MHE.R_inv*diff2; % Weight the difference with the measurement noise
                                % covariance
    sum2 = sum2 + normdiff2; % Sum of measurement prediction errors in the horizon
  end

  % Arrival cost
  diff3 = xk(:,1) - p_MHE.xinit; % Difference between the first optimized state estimate in
                        % the horizon and the guess for the initial state estimate
  normdiff3 = diff3'*p_MHE.P_inv*diff3; % Weight the initialization error with the EKF
                                % estimation error covariance

  % Sum of the normalized differences
  J = sum1+sum2+normdiff3;
end
```

The supporting functions for the state estimation algorithms include 'hfun' below that transforms a vector of states variables, 'x', into a vector of measurement predictions, 'ypred', using the nonlinear measurement equations.

```
function ypred = hfunc(x,p)
  ypred(1) = x(4); % K, Temperature of the bulk concentrate measurement
  ypred(2) = (p.M.XO*x(10))/(p.D.S*p.dim.A); % m, Slag zone height measurement
  ypred(3) = x(11); % K, Temperature of the slag measurement
  ypred(4) = (p.M.XS*x(12))/(p.D.M*p.dim.A); % m, Matte zone height measurement
  ypred(5) = x(13); % K, Temperature of the matte measurement
  ypred(6) = x(15)/p.dim.V; % mol/m^3, Concentration of reaction gases in the furnace
                        % freeboard measurement
  ypred(7) = (x(15)+x(14))*x(16)*(p.other.R/p.dim.V); % Pa, Pressure in the furnace freeboard
                                        % measurement
  ypred(8) = x(16); % K, Temperature in the freeboard measurement
  ypred(9) = x(17); % K, Temperature of the cooling water measurement
```

```
    ypred = ypred';
end
```

The function ‘SAF_model_initialization’ initializes the SAF model. The period over which state estimation takes place, ‘estimationlength’, is passed to the function. The function initializes the SAF process model with initial values for the modes of the furnace, stored in the structure ‘mode’, the event function is stored in ‘options’, and the model parameters are stored in the structure ‘p’. ‘mode’ contains the substructures ‘chargetime’ and ‘stoptime’ for controlling the timed charging of the furnace. The model initializes the sub-structure ‘c’ that controls charging with a ‘1’, indicating the furnace model is being charged upon initialization. The substructures ‘m’ and ‘s’ are initialized with a ‘0’, indicating there is no tapping of matte or slag phase upon model initialization.

```
function [mode, options, p] = SAF_model_initialization(estimationlength)
    % Set the initial charging/tapping modes
    tcharge = 20500; % s, charging period
    tstop = 4000;    % s, period of no charging
    imax = estimationlength*24*3600/(tstop+tcharge);
    imax = ceil(imax);
    icharge = [];
    istop = [];
    for i = 1:imax
        icharge(i) = i*tcharge + i*tstop;
        istop(i) = i*tcharge + (i-1)*tstop;
    end
    mode.chargetime = icharge; % s, time points at which charging occurs
    mode.stoptime = istop;     % s, time points at which charging stops
    mode.c  = 1;             % Model initializes with concentrate being charged
    mode.m  = 0;             % Model initializes with matte not being tapped
    mode.s  = 0;             % Model initializes with slag not being tapped
    p = parameters;          % Load model parameters and store in 'p'
    options = odeset('Events',@(t,y) eventFcn(y,p),'AbsTol',1e-3,'RelTol',1e-4);
end
```

The function ‘SAF_prediction’ is used to carry out the prediction step in each of the state estimator algorithms. The prediction step involves propagating the current vector of states ‘x_current’ from

timestep ‘tStart’ to timestep ‘tEnd’. The model parameters, ‘p’, and event function, ‘options’, are passed to the model as well.

For the EKF and MHE, the input state vector ‘x_mean’ is equal to the input state vector ‘x_current’. For the PF and the UKF, the input ‘x_current’ is the state vector representing a single particle or single sigma point. Each particle or sigma point at the current timestep ‘tStart’ is individually propagated through the function ‘SAF_prediction’. To ensure that all particles and sigma points at a single timestep undergo the same inputs (charging and tapping conditions), the mean of the particles or sigma points at the current timestep is passed to the function to calculate the appropriate mode.

The last input to the function is ‘model_type’. When ‘model_type’ is 0, the original furnace model ODEs are used and the ODE is solved using the function ‘ode15s’. When ‘model_type’ is 1, the furnace singular perturbation model is used. The solver used to solve the ODEs of the singular perturbation model is RK4 integration with a larger specified integration step size of $5s$. This is used in the MHE optimization prediction step to reduce the computational effort of the MHE algorithm.

The function ‘SAF_prediction’ outputs a vector of states ‘x_pred’ representing the state solution at the timepoint ‘tEnd’ and a structure of modes ‘mode’ representing the furnace mode at ‘tEnd’.

```
function [x_pred, mode] = SAF_prediction(tStart, tEnd, x_current, x_mean, p, mode, options, ... model_type)
  tScope = [tStart tEnd];
  % Calculate the appropriate furnace mode at the beginning of the integration period
  profilevar = ode2profile(x_mean',p);
  if mode.m == 1
    if profilevar.L.M < 0.3
    mode.m = 0;
    end
  end
  if mode.m == 0
    if profilevar.L.M > 0.5
    mode.m = 1;
    end
  end
  if mode.s == 1
    if profilevar.L.S < 0.7
    mode.s = 0;
    end
```

```
end
if mode.s == 0
    if profilevar.L.S > 1.1
    mode.s = 1;
    end
end


if model_type == 0
% Original SAF model ODEs are used with the solver 'ode15s'
    while tStart < tEnd
        solx = ode15s(@(t,x) furnaceModelODEs(x,p,mode,t,model_type),tScope, ...
                        x_current,options);
        % The solver stops if t reaches tEnd or if an event function is triggered
        tStart = solx.x(end);      % Update tStart with the last timepoint in the solution
        tScope = [tStart tEnd];    % Update tScope
        x_current = solx.y(:,end); % Update x_current with the last solver solution
        if isempty(solx.ie) == 1
        solx.ie = 0;
        end
        % Update the modes of the furnace
        ieSize = size(solx.ie,2);
        for ii = 1:ieSize
         g = solx.ie(ii);
           switch g
            case 1
              mode.s = 0;
            case 2
              mode.s = 1;
            case 3
              mode.m = 0;
            case 4
            mode.m = 1;
```

```
        end
      end
    end
    % The solution 'xpred' at tEnd is given by the last solver solution in solx.y
    x_pred = solx.y(:,end);
end


if model_type == 1
% Singular perturbation model ODEs are used with RK4 integration
    dt = 5;                % s, Specified step size
    tspan = tStart:dt:tEnd; % Integration timepoints
    xvec(:,1) = x_current;  % Initial model prediction
    for i = 2:length(tspan)
        % Calculate the appropriate furnace mode for the start of the integration period
        profilevar = ode2profile(xvec(:,i-1)',p);
        if mode.m == 1
            if profilevar.L.M < 0.3
            mode.m = 0;
            end
        end
        if mode.m == 0
            if profilevar.L.M > 0.5
            mode.m = 1;
            end
        end
        if mode.s == 1
            if profilevar.L.S < 0.7
            mode.s = 0;
            end
        end
        if mode.s == 0
            if profilevar.L.S > 1.1
```

```
            mode.s = 1;
            end
        end
        t = tspan(i);
        xk = xvec(:, i-1); % Set the initial state values for the RK4 step
        % Using a RK4 step
        k1 = furnaceModelODEs(xk,          p,mode,t,model_type);
        k2 = furnaceModelODEs(xk + dt/2*k1, p,mode,t,model_type);
        k3 = furnaceModelODEs(xk + dt/2*k2, p,mode,t,model_type);
        k4 = furnaceModelODEs(xk + dt*k3,   p,mode,t,model_type);
        xnew = (xk + dt/6*(k1 + 2*k2 + 2*k3 + k4))';
        xvec(:, i) = xnew; % Store the vector xnew in the solution xvec
    end
    % The solution 'xpred' at tEnd is given by the last solver solution in xvec
    x_pred = xvec(:,end);
  end
end
```

The following functions represent the SAF process model ODEs used as the process model in the state estimation algorithms. This is the furnace model ODEs and related MATLAB code written and presented by Theunissen (2021), with slight variations to the model structure to accommodate the furnace blowback simulated for this study.

The 'model_type' is an input to the furnaceModelODEs function. This specifies whether the original model is used, model_type = 0, or the singular perturbation model is used, model_type = 1.

```
% Compute the rate of change in state variables
function dx = furnaceModelODEs(x,p,mode,t,model_type)
  % The column vector, 'x', of state variables are stored in the structures 'N' and 'T'
  [N,T] = ode2state(x);
  [N,V,C,L,P] = state2derived(N,T,p);

  % Heat generation and transfer expressions are contained in 'Q' structure
  Q = heatGenerationTransfer(T,L,p);
  % Mass transfer expressions are contained in the 'F' structure.
```

```
    F = concentrateCharging(mode,t);   % Computes the concentrate charging rate
    F = matteSlagTapping(mode,F);      % Computes the matte/slag tapping rate
    F = concentrateMixing(V,C,p,F); % Computes the rate of bulk and smelting concentrate mixing
    F = concentrateMelting(Q,T,N,p,F); % Computes the concentrate melting rate
    F = gasFlow(P,N,p,F);       % Computes the rate gas exchange rate
    J = gasFlux(P,L,p);    % Computes the reaction gas flux rate through concentrate
    r = reactionRates(T,C,p);  % Computes the rate at which reaction gases are formed
    dN = molarChange(F,r,J,V,p,model_type); % Computes rate of change in molar state variables
    dT = temperatureChange(T,Q,N,F,r,J,V,p,model_type); % Computes temperature rate of change
    dx = state2ode(dN,dT); % Converts 'dN' and 'dT' to columnvector, 'dx'
end

% Compute the concentrate charging rate
function F = concentrateCharging(mode,t)
    % Simulate charging flowrate to the furnace
    for i = 1:length(mode.stoptime)
    if (t>mode.stoptime(i)) && (t<=mode.chargetime(i))
        mode.c = 0; % Specify times when charging must stop, otherwise charge the furnace
    end
    end
    F.charge.total = 410*mode.c; % Total concentrate charging rate, mol/s.
    F.charge.XO  = F.charge.total*0.90; % Slag component flowrate, mol/s
    F.charge.XS  = F.charge.total*0.09; % Matte component flowrate, mol/s
    F.charge.XS2 = F.charge.total*0.01; % Sulphurized matte component flowrate, mol/s
end

% Compute the matte and slag tapping rates
function F = matteSlagTapping(mode,F)
    F.matteTap = 100*mode.m; % Matte tapping rate, mol/s.
    F.slagTap = 400*mode.s;  % Slag tapping rate, mol/s.
end
```

```
% Compute the rate of transfer from the bulk to smelting concentrate
function F = concentrateMixing(V,C,p,F)
    kv = p.k.V*(V.CB/V.CS); % 'kv' increases if the volume ratio between the
                            % bulk- and smelting concentrate increases. This
                            % avoids numerical instability when the amount of
                            % smelting concentrate reaches zero, with bulk
                            % concentrate remaining.
    F.mix.total = kv*V.CB*(C.CB.XO+C.CB.XS+C.CB.XS2); % Total transfer rate, mol/s
    F.mix.XO    = kv*V.CB*C.CB.XO; % Transfer rate of slag, mol/s
    F.mix.XS    = kv*V.CB*C.CB.XS; % Transfer rate of matte, mol/s
    F.mix.XS2   = kv*V.CB*C.CB.XS2; % Transfer rate of sulphurized matte, mol/s
end

% Compute the rate at which slag and matte components melt from the smelting concentrate
function F = concentrateMelting(Q,T,N,p,F)
    F.melt.XO = Q.S2CS*((T.CS/p.other.Tmelt)^2)/(p.fus.C-p.cP.C*(p.other.Tmelt-T.CS))*...
                N.CS.XO/(N.CS.XO+N.CS.XS); % Slag melting rate, mol/s.
    F.melt.XS = Q.S2CS*((T.CS/p.other.Tmelt)^2)/(p.fus.C-p.cP.C*(p.other.Tmelt-T.CS))*...
                N.CS.XS/(N.CS.XO+N.CS.XS); % Matte melting rate, mol/s.
end

% Compute the rate of heat generation in the slag zone
function Q = heatGenerationTransfer(T,L,p)
    Q.J = (p.Q.Velectrode^2)/(p.Q.R0*(1+p.Q.alpha*(T.S-p.Q.T0))); % Heat generation rate, kW
    Q.CS2CB = p.Q.hCS2CB*p.dim.A*(T.CS-T.CB); % Heat transfer from smelting-bulk concentrate, kW
    Q.G2CB  = p.Q.hG2CB*p.dim.A*(T.G-T.CB); % Heat transfer from freeboard-bulk concentrate, kW
    Q.S2CS  = p.Q.hS2CS*p.dim.A*(T.S-T.CS); % Heat transfer from slag-smelting concentrate, kW
    Q.M2S   = p.Q.hM2S*p.dim.A*(T.M-T.S); % Heat transfer from matte-slag, kW
    Q.W2S   = p.Q.hW2S*p.dim.p*L.S*(T.W-T.S); % Heat transfer from cooling units-slag, kW
    Q.W2M   = p.Q.hW2M*p.dim.p*L.M*(T.W-T.M); % Heat transfer from cooling units-matte, kW
end
```

```
% Computes the desulphurization and electrode oxidation reaction rates in the bulk and smelting
% concentrate zones
function r = reactionRates(T,C,p)
   r.F.CB = p.k.rF*exp(-p.EA.F/(p.other.R*T.CB))...
            *C.CB.XS2; % Desulphurization in bulk concentrate, mol/s
   r.F.CS = p.k.rF*exp(-p.EA.F/(p.other.R*T.CS))...
            *C.CS.XS2; % Desulphurization in smelting concentrate, mol/s
   r.C    = p.k.rC*exp(-p.EA.C/(p.other.R*T.S)); % Electrode oxidation, mol/s
end
```

During nominal operation of the furnace, the furnace freeboard pressure remains fairly constant. The furnace freeboard gauge pressure remains negative, thus, there is molar flow of atmospheric air into the furnace and extraction of air and reaction gases due to the pressure difference created between P_{ext} and P_G. When the reaction gases buildup in the concentrate and the pressure buildup becomes too high, there is a sudden channelling of reaction gases to the freeboard and the furnace freeboard gauge pressure becomes positive. As the furnace freeboard pressure becomes positive, air and reaction gases flow out of the furnace during a phenomenon known as blowback. Therefore, under nominal operation the flowrate of gases out of the freeboard due to blowback is assumed to be zero and the reaction gas flux from the concentrate to the furnace is assumed to always behave as a PBR.

```
% Compute the molar exchange rate between the freeboard and the atmosphere
function F = gasFlow(P,N,p,F)
   F.negP   = p.k.PR*(p.other.Patm-P.G); % Air drawn into the furnace, mol/s
   F.posP.R = 0; % Reaction gases blown out of the furnace during blowback, mol/s
   F.posP.A = 0; % Air blown out of the furnace during blowback, mol/s
   fExt = p.k.PE*(P.G-(p.other.Pext)); % Total rate of freeboard gas extraction, mol/s
   F.ext.R  = fExt*N.G.R/(N.G.R+N.G.A); % Reaction gas extraction, mol/s
   F.ext.A  = fExt*N.G.A/(N.G.R+N.G.A); % Air extraction, mol/s
End
% Compute the reaction gas flux from the concentrate bed to the furnace freeboard
function J = gasFlux(P,L,p)
   J = (P.CR-P.G)*p.k.PBR/L.C; % Unruptured bed behaves as a PBR, mol/m^2.s
end
```

For model_type = 0, the original ODEs for the states with the fast dynamics are used. For the singular perturbation, model_type = 1, the states with fast dynamics, namely the number of moles of air in the

freeboard ‘N.G.A’, the number of moles of reaction gas in the freeboard ‘N.G.R’, and the temperature in the freeboard ‘T.G’, are assumed to have zero dynamics. This is to enable to use of a large integration step size in the RK4 integration step.

```
% Compute rate of change in state variable molar amounts
function dN = molarChange(F,r,J,V,p,model_type)
    dN.CB.XO  = F.charge.XO-F.mix.XO; % Bulk concentrate slag, mol/s
    dN.CB.XS  = F.charge.XS-F.mix.XS+r.F.CB*V.CB; % Bulk concentrate matte, mol/s
    dN.CB.XS2 = F.charge.XS2-F.mix.XS2-r.F.CB*V.CB; % Bulk concentrate sulphurized matte, mol/s
    dN.CS.XO  = F.mix.XO-F.melt.XO; % Smelting concentrate slag, mol/s
    dN.CS.XS  = F.mix.XS-F.melt.XS+r.F.CS*V.CS; % Smelting concentrate matte, mol/s
    dN.CS.XS2 = F.mix.XS2-r.F.CS*V.CS; % Smelting concentrate sulphurized matte, mol/s
    dN.CR = r.F.CB*V.CB+r.F.CS*V.CS+r.C-J*p.dim.A; % Reaction gases in concentrate bed, mol/s
    dN.S  = F.melt.XO-F.slagTap; % Change in slag zone, mol/s
    dN.M  = F.melt.XS-F.matteTap; % Change in matte zone, mol/s

    if model_type == 0;
        % Original SAF model
        dN.G.A = F.negP - F.posP.A - F.ext.A; % Change in air in freeboard, mol/s
        dN.G.R = J*p.dim.A - F.posP.R - F.ext.R; % Change in reaction gases in freeboard, mol/s
    end
    if model_type == 1;
        % Singular perturbation model assumes states with fast dynamics have zero dynamics
        dN.G.A = 0; % Change in air in freeboard, mol/s
        dN.G.R = 0; % Change in reaction gases in freeboard, mol/s
    end
end

% Compute rate of change in state temperature variables
function dT = temperatureChange(T,Q,N,F,r,J,V,p,model_type)
    % Bulk concentrate temperature change, K/s:
    dT.CB = (Q.CS2CB+Q.G2CB+p.cP.C*(p.other.Tcharge-T.CB)*F.charge.total)/...
            (N.CB.total*p.cP.C);
```

```
    % Smelting concentrate temperature change, K/s:
    dT.CS = (Q.S2CS*(1-(T.CS/p.other.Tmelt)^2)-Q.CS2CB+(T.CB-T.CS)*...
            (F.mix.total*p.cP.C+r.F.CB*V.CB*p.cP.G))/(N.CS.total*p.cP.C);
    % Slag temperature change, K/s:
    dT.S  = (Q.J+Q.M2S+Q.W2S-Q.S2CS+(p.other.Tmelt-T.S)*...
            (F.melt.XO*p.cP.S+F.melt.XS*p.cP.M))/(N.S*p.cP.S);
    % Matte temperature change, K/s:
    dT.M  = (Q.W2M-Q.M2S+(T.S-T.M)*F.melt.XS*p.cP.M)/(N.M*p.cP.M);
    % Cooling units temperature change, K/s:
    dT.W  = (p.cP.W*(p.water.T0-T.W)*p.water.F-Q.W2M-Q.W2S)/(p.water.N*p.cP.W);

    if model_type == 0;
        % Original SAF model
        % Freeboard temperature change, K/s:
        dT.G  = (p.cP.G*((T.CS-T.G)*J*p.dim.A+(p.other.Tatm-T.G)*F.negP)-Q.G2CB)/...
                    (N.G.total*p.cP.G);
    end
    if model_type == 1;
        % Singular perturbation model assumes states with fast dynamics have zero dynamics
        % Freeboard temperature change, K/s:
        dT.G  = 0;
    end
end
```

The event function used to initiate the SAF model solver differs from the event function used in the plant simulation as there is no blowback event. Therefore, the only events that stop the solver are slag and matte tapping.

```
% 'eventFcn' is used to halt the ode solver when matte/slag tapping should be started or stopped
function [threshold,isterminal,direction] = eventFcn(y,p)
    [N,T] = ode2state(y);
    [~,~,~,L,~] = state2derived(N,T,p);
    % The ode solver also stops when the slag height exceeds its thresholds:
```

```
    threshold(1,1) = L.S - 0.7; % Height of slag goes below 0.1 m
    threshold(2,1) = L.S - 1.1; % Height of slag goes above 0.2 m
    % The ode solver also stops when the matte height exceeds its thresholds:
    threshold(3,1) = L.M - 0.3; % Height of matte goes below 0.04 m
    threshold(4,1) = L.M - 0.5; % Height of matte goes above 0.08 m


    % If isterminal(i) = 1 when threshold(i) crosses 0, then the ode solver
    % stops. This happens for each threshold.
    isterminal(1,1) = 1;
    isterminal(2,1) = 1;
    isterminal(3,1) = 1;
    isterminal(4,1) = 1;


    % direction(i) determines if threshold(i) should be crossed by decreasing
    % or increasing values for the ode solver to be stopped:
    direction(1,1) = -1; % Slag level goes below lower threshold
    direction(2,1) = 1;  % Slag level goes above upper threshold
    direction(3,1) = -1; % Matte level goes below lower threshold
    direction(4,1) = 1;  % Matte level goes above upper threshold
end
```

D.2. Simulation of synthetic measurement data and ground truth state values

The following function and supporting sub-functions are used to generate the measurement data for the measured variables and the ground truth values of the state variables. The following code represents the ODEs and related MATLAB code written and presented by Theunissen (2021), with slight variations to the model structure to accommodate the furnace blowback simulated for this study.

Function inputs	
tDuration	Duration in days which synthetic measurement data is generated for.
faultindex	Specify which system condition the measurements are generated under. 0: Nominal conditions 1: Initiate fault 1, step change in the flowrate of cooling water, at time = 1 day

	2: Initiate fault 2, step change in the extraction pressure, at time = 1 day 3: Initiate fault 3, step change in the charge composition, at time = 1 day 4: Initiate fault 4, furnace blowback, at time = 1 day
stochastic_percentage	Specify percentage of stochastic variation for simulation of plant-model mismatch via stochastic variation in the input F_{charge}
seed	Seed the random number generated to allow for reproducibility of measurement data
Function outputs	
meas_data	Matrix containing synthetically generated measurements. Number of columns in measurement data is equivalent to number of measured variables. Number of rows is equivalent to number of measurements generated over the period tDuration.
meas_info	Structure containing information about the measurements. measurement_info.data: contains the underlying state variable data used to generate each measurement. measurement_info.var: contains the variance of the measurement noise. measurement_info.T: contains the sampling rates of each measurement.
x_true	Ground truth values of the state variables.
t_true	Time points at which ground truth values, x_true, were obtained at.

```
function [meas_data, meas_info, x_true, t_true] = SAF_measurements( tDuration, ...
                                    faultindex, stochastic_percentage, seed)
%% 1. Model setup

tStart = 0;             % Start time of simulation (s)

tEnd = tDuration*24*3600; % End time of simulation (s)

tScope = [tStart, tEnd];  % Simulation time scope (s)

rng(seed);              % Seed the rng

p = parameters;         % Load model parameters and store in 'p'

options = odeset('Events',@(t,y) eventFcn_plant(y,p),'AbsTol',1e-3,'RelTol',1e-4); % Set solver
                                                    % options


%% 2. Variable initialization and memory pre-allocation
```

```
[N,T] = variableInitialization; % State variable initial values are stored in N and T
x0 = state2ode(N,T);            % Vector of initial states
m = size(x0,1);                 % dimension of pre-allocated memory
n = 1e7;                        % dimension of pre-allocated memory
% Create the structure 'mode' for controlling the inputs
% Substructure m controls matte tapping
mode.m  = 0; % Model initializes with no matte tapping
% Substructure s controls slag tapping
mode.s  = 0; % Model initializes with no slag tapping
% Substructure bb controls if furnace bed is ruptured, initiating furnace blowback
mode.bb = 0; % Model initializes with no furnace blowback
% Substructure c, chargetime, and stoptime control the concentrate charging
mode.c  = 1; % Model initializes with concentrate charging
tcharge = 20500; % s, Specify the time length of charging
tstop = 4000;    % s, Specify the time length of no charging
imax = ceil(tEnd/(tstop+tcharge)); % Maximum time index for duration of simulation
for i = 1:imax
   icharge(i) = i*tcharge + i*tstop;   % Specify timepoints of charging
   istop(i) = i*tcharge + (i-1)*tstop; % Specify timepoints of stopping charge
end
mode.chargetime = icharge;
mode.stoptime = istop;


odeVariables = zeros(n,m); % stores ODE solution
tSimulated = zeros(n,1);   % stores times which ODEs are solved at
j = 1;                     % index for pre-allocated memory
odeVariables(j,:) = x0;    % store initial values as first entry in odeVariables


%% 3. Fault initialization
% Step change in the cooling water flowrate
fault.coolWat.time = 10;  % Time at which step change occurs
fault.coolWat.mag = -500; % Magnitude of cooling water flowrate change (mols/s)
```

```
if faultindex == 1
    % Simulate the fault after 1 day of operation
    fault.coolWat.time = 1;
end
% Step change in the extraction pressure
fault.extPres.time = 10; % Time at which step change occurs
fault.extPres.mag = 1;   % Magnitude of extraction pressure change (Pa)
if faultindex == 2
    % Simulate the fault after 1 day of operation
    fault.extPres.time = 1;
end
% Step change in the composition of the charging concentrate
fault.concCharge.time = 10;  % Time at which step change occurs
fault.concCharge.mag = -0.12; % Fraction by which slag component composition
                              % changes in charged concentrate
if faultindex == 3
    % Simulate the fault after 1 day of operation
    fault.concCharge.time = 1;
end
% Simulate furnace blowback via step change in blowback parameters
fault.blowbackparameter.time = 10;  % Time at which parameter deviation occurs
fault.blowbackparameter.mag1 = 0.7; % Fraction parameter 1 deviation
fault.blowbackparameter.mag2 = 0.8; % Fraction parameter 2 deviation
if faultindex == 4
    % Simulate the fault after 1 day of operation
    fault.blowbackparameter.time = 1;
end


%% 4. Simulate stochastic variation in the input Fcharge
mean_charge = 410; % Specify mean value for input Fcharge
% Define the p, B1, and standard deviation values in the AR1 model
p_charge = 0.8;
```

```
B1_charge = mean_charge*(1-p_charge);

var_charge = ((stochastic_percentage/100)*mean_charge)^2;

stddev_charge = sqrt(var_charge*(1-p_charge^2));

t_Fcharge = linspace(0,tDuration*24*3600);

s_Fcharge(1) = mean_charge;

for i = 2:length(t_Fcharge)

    s_Fcharge(i) = B1_charge + p_charge*s_Fcharge(i-1) + stddev_charge*randn;

end

% Define the stochastically varying Fcharge as a function of time

Fcharge = griddedInterpolant(t_Fcharge, s_Fcharge);


%% 5. ODE solution

% The ODE is solved for from tStart to tEnd, storing the ODE solution in 'odeVariables' and the

% time points of the solution in 'tSimulated'.

while tStart<tEnd

% 'ode15s' solves the SAF model within the span specified by 'tScope' using

% the initial values in 'xi'. The 'furnaceModelODEs' subfunction is used in

% the 'ode15s' solver. The output generated by 'ode15s' is in 'yout'.

% 'ie'is the number of the specific event triggered, as given in the 'eventFcn'

% subfunction.

[tout,yout,~,~,ie]= ode15s(@(t,x) furnaceModelODEs_plant (x, p, mode, t, fault,  Fcharge), ...

                           tScope, x0, options);

% 'ode15s' stops if 'eventFcn' is triggered or 'tEnd' is reached.

% If 'tEnd' is reached, ie will be empty. '0' is assigned to ie to allow 'switch' to work.

if isempty(ie) == 1

    ie = 0;

end


% Update 'tStart', 'x0' and 'tScope'

tStart = tout(end);

tScope = linspace(tStart, tEnd, (tEnd-tStart)/10);

x0 = yout(end,:)';
```

```
% The values in 'yout' and 'tout' are assigned to 'odeVariables' and 'tSimulated'
nSample = size(yout,1)-2;
odeVariables(j+1:j+nSample,:) = yout(2:end-1,:);
tSimulated(j+1:j+nSample,:) = tout(2:end-1,:);
% Update index variable j
j = j+nSample;

% If two events specified in the event funtions are triggered
% simultaneously, then ie is a rowvector with entries specifying which
% events were triggered. A for loop cycles through these entries.
ieSize = size(ie,2);
for i = 1:ieSize
    g = ie(i);
    switch g
     case 1
        mode.s = 0;
     case 2
        mode.s = 1;
     case 3
        mode.m = 0;
     case 4
        mode.m = 1;
     case 5
        mode.bb = 1;
     case 6
        mode.bb = 0;
    end
end
end

%% 6. Data cleanup
```

```
% Remove unused pre-allocated memory
idxDelete = [(j+1):n]';
odeVariables(idxDelete,:) = [];
tSimulated(idxDelete) = [];
dataProfile = ode2profile(odeVariables);
simulationData.time = tSimulated/(24*3600);
simulationData.data = dataProfile;
% Store ground truth state values and the timepoints they are obtained at
x_true = odeVariables;
t_true = tSimulated;


%% 7. Generate measurements
% Create measurement structures:
%  fields: names of measurements
%  var:    assume gaussian noise with variance "var"
%  T:      sampling rate of measurement (s)
meas_info.fields = {'TCB','LS','TS', 'LM', 'TM', 'CGR', 'PG', 'TG', 'TW' };
meas_info.TCB = struct('data', simulationData.data.T.CB, 'var', (x0(4)*(1/100))^2, 'T', 10); % K
meas_info.LS = struct('data', simulationData.data.L.S, 'var', (0.05)^2, 'T', 10); % m
meas_info.TS = struct('data', simulationData.data.T.S, 'var', (x0(11)*(1/100))^2, 'T', 10); % K
meas_info.LM = struct('data', simulationData.data.L.M, 'var', (0.05)^2, 'T', 10); %  m
meas_info.TM = struct('data', simulationData.data.T.M, 'var', (x0(13)*(1/100))^2, 'T', 10); %  K
meas_info.CGR = struct('data', simulationData.data.C.G.R, 'var', 0.1^2, 'T', 10); % mol/m^3
meas_info.PG = struct('data', simulationData.data.P.G, 'var', 2^2, 'T', 10);    %  Pa
meas_info.TG = struct('data', simulationData.data.T.G, 'var', 2.2^2, 'T', 10);    %  K
meas_info.TW = struct('data', simulationData.data.T.W, 'var', 1^2, 'T', 10);    %  K
% Record the measurements
y = Measurements_SAF(tSimulated, meas_info);
% Save measurement data
meas_data = y;
```

The function ‘Measurements_SAF’ is used to generate the measurements from the solver solution. A matrix of measurements ‘y’ containing the timeseries data of the measured variables is the output of this function.

```
% This function uses the measurement structure to generate a set of measured values

function y = Measurements_SAF(t, meas)

   % For each measurement field

   for i = 1:length(meas.fields)

      current = meas.(meas.fields{i}); % Current measurement

      values = current.data + sqrt(current.var)*randn(size(t)); % Add random noise with

                                                % with variance ‘var’

      times = [0 + current.T : current.T : t(end)]'; % Specify the measurement time points

      times(times<min(t)) = []; % This ensures that 'tInterp' only contains entries at which

                           % 'tSimulated' can be interpolated.

      interp_values = interp1(t, values, times, 'nearest'); % Interpolate

      y(:,i) = interp_values;

   end

end
```

The functions used to generate the SAF process ODEs are presented below. These represent the true underlying ODEs that drive the SAF plant process. The ‘plant’ ODEs differ from the process model ODEs used in the state estimation algorithms.

```
%% Furnace ODE subfunctions

% Compute the rate of change in state variables

function dx = furnaceModelODEs_plant(x,p,mode,t,fault,Fcharge)

   % The column vector, 'x', of state variables are stored in the structures ‘N’ and ‘T’

   [N,T] = ode2state(x);

   [N,V,C,L,P] = state2derived(N,T,p);

   % Heat generation and transfer expressions are contained in 'Q' structure

   Q = heatGenerationTransfer_plant(T,L,p,t,fault);
```

```
    % Mass transfer expressions are contained in the 'F' structure.
    F = concentrateCharging_plant(mode,t,fault,Fcharge);% Computes the concentrate charging rate
    F = matteSlagTapping_plant(mode,F); % Computes the matte/slag tapping rate
    F = concentrateMixing_plant(V,C,p,F); % Computes bulk and smelting concentrate mixing rate
    F = concentrateMelting_plant(Q,T,N,p,F); % Computes the concentrate melting rate
    F = gasFlow_plant(P,N,p,F,fault,t);       % Computes the rate gas exchange rate
    J = gasFlux_plant(P,L,p,mode,t,fault); % Computes reaction gas flux rate through concentrate
    r = reactionRates_plant(T,C,p);  % Computes the rate at which reaction gases are formed
    dN = molarChange_plant(F,r,J,V,p); % Computes the rate of change in molar state variables
    dT = temperatureChange_plant(T,Q,N,F,r,J,V,p,fault,t); % Computes temperature rate of change
    dx = state2ode(dN,dT); % Converts 'dN' and 'dT' to columnvector, 'dx'
end
```

The plant is subject to potential stochastic variation in the concentrate charging flowrate. This differs from the process model used in the state estimation algorithm, which assumes 'Fcharge.total' is constant. Furthermore, the plant is subject to faulty conditions. Simulation of fault 3, a sudden change in the charging composition, is simulated below.

```
% Compute the concentrate charging rate
function F = concentrateCharging_plant(mode,t,fault,Fcharge)
    % Simulate the potentially stochastically varying charging flowrate to the furnace
    for i = 1:length(mode.stoptime)
        if (t>mode.stoptime(i)) && (t<=mode.chargetime(i))
            mode.c = 0; % Specify times when charging must stop, otherwise charge the furnace
        end
    end
    F.charge.total = Fcharge(t)*mode.c; % Total concentrate charging rate, mol/s.
    f1 = 0.9; % Slag composition in charging concentrate
    f2 = 0.09; % Matte composition in charging concentrate

    % Simulate fault 3: sudden change in the charging composition
    if t > fault.concCharge.time*24*3600
        f1 = 0.9 + fault.concCharge.mag; % Slag composition in charging concentrate
```

```
        f2 = 0.09 - fault.concCharge.mag; % Matte composition in charging concentrate
    end
    F.charge.XO  = F.charge.total*f1; % Slag component flowrate, mol/s
    F.charge.XS  = F.charge.total*f2; % Matte component flowrate, mol/s
    F.charge.XS2 = F.charge.total*0.01; % Sulphurized matte component flowrate, mol/s
end


% Compute the matte and slag tapping rates
function F = matteSlagTapping_plant(mode,F)
    F.matteTap = 100*mode.m; % Matte tapping rate, mol/s.
    F.slagTap = 400*mode.s;  % Slag tapping rate, mol/s.
end


% Compute the rate of transfer from the bulk to smelting concentrate
function F = concentrateMixing_plant(V,C,p,F)
    kv = p.k.V*(V.CB/V.CS); % 'kv' increases if the volume ratio between the
                            % bulk- and smelting concentrate increases. This
                            % avoids numerical instability when the amount of
                            % smelting concentrate reaches zero, with bulk
                            % concentrate remaining.
    F.mix.total = kv*V.CB*(C.CB.XO+C.CB.XS+C.CB.XS2); % Total transfer rate, mol/s
    F.mix.XO    = kv*V.CB*C.CB.XO; % Transfer rate of slag, mol/s
    F.mix.XS    = kv*V.CB*C.CB.XS; % Transfer rate of matte, mol/s
    F.mix.XS2   = kv*V.CB*C.CB.XS2; % Transfer rate of sulphurized matte, mol/s
end


% Compute the rate at which slag and matte components melt from the smelting concentrate
function F = concentrateMelting_plant(Q,T,N,p,F)
    F.melt.XO = Q.S2CS*((T.CS/p.other.Tmelt)^2)/(p.fus.C-p.cP.C*(p.other.Tmelt-T.CS))*...
                N.CS.XO/(N.CS.XO+N.CS.XS); % Slag melting rate, mol/s.
    F.melt.XS = Q.S2CS*((T.CS/p.other.Tmelt)^2)/(p.fus.C-p.cP.C*(p.other.Tmelt-T.CS))*...
                N.CS.XS/(N.CS.XO+N.CS.XS); % Matte melting rate, mol/s.
```

```
end


% Computes the desulphurization and electrode oxidation reaction rates in the bulk and smelting
% concentrate zones
function r = reactionRates_plant(T,C,p)
   r.F.CB = p.k.rF*exp(-p.EA.F/(p.other.R*T.CB))...
            *C.CB.XS2; % Desulphurization in bulk concentrate, mol/s
   r.F.CS = p.k.rF*exp(-p.EA.F/(p.other.R*T.CS))...
            *C.CS.XS2; % Desulphurization in smelting concentrate, mol/s
   r.C    = p.k.rC*exp(-p.EA.C/(p.other.R*T.S)); % Electrode oxidation, mol/s
end
```

The plant is also subject to another faulty condition, furnace blowback due to a buildup of reaction gases within the concentrate. To simulate furnace blowback, the heat transfer coefficient between the slag and smelting concentrate is suddenly decreased to simulate a decrease in the melting rate and therefore buildup of concentrate. This is simulated below.

```
% Compute the rate of heat generation in the slag zone
function Q = heatGenerationTransfer_plant(T,L,p,t,fault)
   Q.J = (p.Q.Velectrode^2)/(p.Q.R0*(1+p.Q.alpha*(T.S-p.Q.T0))); % Heat generation rate, kW
   Q.CS2CB = p.Q.hCS2CB*p.dim.A*(T.CS-T.CB); % Heat transfer from smelting-bulk concentrate, kW
   Q.G2CB  = p.Q.hG2CB*p.dim.A*(T.G-T.CB); % Heat transfer from freeboard-bulk concentrate, kW
   Q.S2CS  = p.Q.hS2CS*p.dim.A*(T.S-T.CS); % Heat transfer from slag-smelting concentrate, kW
   Q.M2S   = p.Q.hM2S*p.dim.A*(T.M-T.S); % Heat transfer from matte-slag, kW
   Q.W2S   = p.Q.hW2S*p.dim.p*L.S*(T.W-T.S); % Heat transfer from cooling units-slag, kW
   Q.W2M   = p.Q.hW2M*p.dim.p*L.M*(T.W-T.M); % Heat transfer from cooling units-matte, kW
   % Simulate fault 4: furnace blowback
   if t > fault.blowbackparameter.time*24*3600
      Q.S2CS  = p.Q.hS2CS*(fault.blowbackparameter.mag2)*p.dim.A*(T.S-T.CS);
   end
end
```

In the simulation of the SAF plant, when the freeboard gauge pressure becomes positive, reaction gases and air are blown out of the furnace during furnace blowback. In the process model used in the state

estimators, these are always assumed to be zero. In addition, fault 2, a sudden change in the extraction pressure, is simulated in the plant as seen below.

```
% Compute the molar exchange rate between the freeboard and the atmosphere
function F = gasFlow_plant(P,N,p,F,fault,t)
   F.negP   = p.k.PR*(p.other.Patm-P.G); % Air drawn into the furnace, mol/s
   % Reaction gases blown out of the furnace , mol/s
   F.posP.R = p.k.PR*(P.G-p.other.Patm)*(P.G>=p.other.Patm)*N.G.R/(N.G.R+N.G.A);
   % Air blown out of the furnace, mol/s
   F.posP.A = p.k.PR*(P.G-p.other.Patm)*(P.G>=p.other.Patm)*N.G.A/(N.G.R+N.G.A);
   fExt = p.k.PE*(P.G-(p.other.Pext)); % Total rate of freeboard gas extraction, mol/s
   % Simulate fault 2: sudden change in Pext
   if t > fault.extPres.time*24*3600
      fExt = p.k.PE*(P.G-(p.other.Pext+fault.extPres.mag));
   end
   F.ext.R  = fExt*N.G.R/(N.G.R+N.G.A); % Reaction gas extraction, mol/s
   F.ext.A  = fExt*N.G.A/(N.G.R+N.G.A); % Air extraction, mol/s
end
```

In the plant model, reaction gas buildup within the concentrate causes pressure in the concentrate to exceed the critical pressure and the reaction gas flux occurs in a channeling motion due to the ruptured concentrate bed. This phenomenon is simulated within the plant model. In the process model used in the state estimation algorithms, the flux is always assumed to behave as a PBR. Furthermore, to simulate furnace blowback the flux coefficient, 'J_pbr', is suddenly decreased to reduce the movement of gases out of the concentrate to simulate a buildup of gases within the concentrate layer.

```
% Compute the reaction gas flux from the concentrate bed to the furnace freeboard
function J = gasFlux_plant(P,L,p,mode,t,fault)
   J_pbr = (P.CR-P.G)*p.k.PBR/L.C; % Unruptured bed behaves as a PBR, mol/m^2.s
   % Simulate fault 4: furnace blowback
   if t > fault.blowbackparameter.time*24*3600
      J_pbr = (P.CR-P.G)*p.k.PBR*(fault.blowbackparameter.mag1)/L.C;
   end
   J_ch  = (P.CR-P.G)*p.k.Ch; % Flux channels when bed is ruptured, mol/m^2.s
   J = J_pbr*(mode.bb==0)+J_ch*(mode.bb==1); % The output of this function depends on whether
```

```
                                % or not the concentrate bed is ruptured
    J = J*(J>=0); % 'J' is either positive or zero
end


% Compute rate of change in state variable molar amounts
function dN = molarChange_plant(F,r,J,V,p)
    dN.CB.XO  = F.charge.XO-F.mix.XO; % Bulk concentrate slag, mol/s
    dN.CB.XS  = F.charge.XS-F.mix.XS+r.F.CB*V.CB; % Bulk concentrate matte, mol/s
    dN.CB.XS2 = F.charge.XS2-F.mix.XS2-r.F.CB*V.CB; % Bulk concentrate sulphurized matte, mol/s
    dN.CS.XO  = F.mix.XO-F.melt.XO; % Smelting concentrate slag, mol/s
    dN.CS.XS  = F.mix.XS-F.melt.XS+r.F.CS*V.CS; % Smelting concentrate matte, mol/s
    dN.CS.XS2 = F.mix.XS2-r.F.CS*V.CS; % Smelting concentrate sulphurized matte, mol/s
    dN.CR = r.F.CB*V.CB+r.F.CS*V.CS+r.C-J*p.dim.A; % Reaction gases in concentrate bed, mol/s
    dN.S  = F.melt.XO-F.slagTap; % Change in slag zone, mol/s
    dN.M  = F.melt.XS-F.matteTap; % Change in matte zone, mol/s
    dN.G.A = F.negP - F.posP.A - F.ext.A; % Change in air in freeboard, mol/s
    dN.G.R = J*p.dim.A - F.posP.R - F.ext.R; % Change in reaction gases in freeboard, mol/s
end


% Compute rate of change in state temperature variables
function dT = temperatureChange_plant(T,Q,N,F,r,J,V,p,fault,t)
    % Bulk concentrate temperature change, K/s:
    dT.CB = (Q.CS2CB+Q.G2CB+p.cP.C*(p.other.Tcharge-T.CB)*F.charge.total)/...
            (N.CB.total*p.cP.C);
    % Smelting concentrate temperature change, K/s:
    dT.CS = (Q.S2CS*(1-(T.CS/p.other.Tmelt)^2)-Q.CS2CB+(T.CB-T.CS)*...
            (F.mix.total*p.cP.C+r.F.CB*V.CB*p.cP.G))/(N.CS.total*p.cP.C);
    % Slag temperature change, K/s:
    dT.S  = (Q.J+Q.M2S+Q.W2S-Q.S2CS+(p.other.Tmelt-T.S)*...
            (F.melt.XO*p.cP.S+F.melt.XS*p.cP.M))/(N.S*p.cP.S);
    % Matte temperature change, K/s:
    dT.M  = (Q.W2M-Q.M2S+(T.S-T.M)*F.melt.XS*p.cP.M)/(N.M*p.cP.M);
```

```
    % Freeboard temperature change, K/s:
    dT.G  = (p.cP.G*((T.CS-T.G)*J*p.dim.A+(p.other.Tatm-T.G)*F.negP)-Q.G2CB)/...
            (N.G.total*p.cP.G);
    % Cooling units temperature change, K/s:
    dT.W  = (p.cP.W*(p.water.T0-T.W)*p.water.F-Q.W2M-Q.W2S)/(p.water.N*p.cP.W);
    % Simulate fault 1: sudden change in the flowrate of cooling water
    if t > fault.coolWat.time*3600*24
        Fcool = p.water.F + fault.coolWat.mag;
        dT.W  = (p.cP.W*(p.water.T0-T.W)*Fcool-Q.W2M-Q.W2S)/(p.water.N*p.cP.W);
    end
end
```

The last fault the plant is subject to is a sudden decrease in the flowrate of cooling water. This is simulated below. None of the aforementioned faults are modelled in the process model equations supplied to the state estimation algorithms.

```
% Compute rate of change in state temperature variables
function dT = temperatureChange_plant(T,Q,N,F,r,J,V,p,fault,t)
    % Bulk concentrate temperature change, K/s:
    dT.CB = (Q.CS2CB+Q.G2CB+p.cP.C*(p.other.Tcharge-T.CB)*F.charge.total)/...
            (N.CB.total*p.cP.C);
    % Smelting concentrate temperature change, K/s:
    dT.CS = (Q.S2CS*(1-(T.CS/p.other.Tmelt)^2)-Q.CS2CB+(T.CB-T.CS)*...
            (F.mix.total*p.cP.C+r.F.CB*V.CB*p.cP.G))/(N.CS.total*p.cP.C);
    % Slag temperature change, K/s:
    dT.S  = (Q.J+Q.M2S+Q.W2S-Q.S2CS+(p.other.Tmelt-T.S)*...
            (F.melt.XO*p.cP.S+F.melt.XS*p.cP.M))/(N.S*p.cP.S);
    % Matte temperature change, K/s:
    dT.M  = (Q.W2M-Q.M2S+(T.S-T.M)*F.melt.XS*p.cP.M)/(N.M*p.cP.M);
    % Freeboard temperature change, K/s:
    dT.G  = (p.cP.G*((T.CS-T.G)*J*p.dim.A+(p.other.Tatm-T.G)*F.negP)-Q.G2CB)/...
            (N.G.total*p.cP.G);
    % Cooling units temperature change, K/s:
```

```
    dT.W  = (p.cP.W*(p.water.T0-T.W)*p.water.F-Q.W2M-Q.W2S)/(p.water.N*p.cP.W);
    % Simulate fault 1: sudden change in the flowrate of cooling water
    if t > fault.coolWat.time*3600*24
        Fcool = p.water.F + fault.coolWat.mag;
        dT.W  = (p.cP.W*(p.water.T0-T.W)*Fcool-Q.W2M-Q.W2S)/(p.water.N*p.cP.W);
    end
end
```

The event function used in the solver that solves for the plant state values differs from the event function used in the state estimation algorithm prediction step. For the plant simulation, the solver must stop when slag or matte tapping occurs or when furnace blowback occurs.

```
% 'eventFcn' is used to halt the ode solver when matte/slag tapping should be started or stopped % and when furnace blowback occurs
function [threshold,isterminal,direction] = eventFcn_plant(y,p)
    [N,T] = ode2state(y);
    [~,V,~,L,P] = state2derived(N,T,p);
    delPcrit = p.D.C*p.other.g*L.C;
    delP = P.CR - P.G;

    % The ode solver also stops when the slag height exceeds its thresholds
    threshold(1,1) = L.S - 0.7; % Height of slag goes below 0.1 m
    threshold(2,1) = L.S - 1.1; % Height of slag goes above 0.2 m
    % The ode solver also stops when the matte height exceeds its thresholds
    threshold(3,1) = L.M - 0.3; % Height of matte goes below 0.04 m
    threshold(4,1) = L.M - 0.5; % Height of matte goes above 0.08 m
    threshold(5,1) = delP - delPcrit*4; % Pressure in the freeboard goes above pcrit
    threshold(6,1) = delP - delPcrit*1; % Pressure in the freeboard goes below pcrit

    % If isterminal(i) = 1 when threshold(i) crosses 0, then the ode solver stops
    isterminal(1,1) = 1;
    isterminal(2,1) = 1;
```

```
    isterminal(3,1) = 1;
    isterminal(4,1) = 1;
    isterminal(5,1) = 1;
    isterminal(6,1) = 1;

    % direction(i) determines threshold(i) should be crossed by decreasing or increasing values
    direction(1,1) = -1; % Concentrate level goes below lower threshold
    direction(2,1) = 1;  % Concentrate level goes above upper threshold
    direction(3,1) = -1; % Slag level goes below lower threshold
    direction(4,1) = 1;  % Slag level goes above upper threshold
    direction(5,1) = 1;  % Reaction gases in concentrate goes above critical P
    direction(6,1) = -1; % Reaction gases in concentrate goes below critical P
end
```

The remaining supporting functions are used in both the plant simulation and the process model supplied to the state estimation algorithms.

```
% Store the initial state variable values in structures 'N' and 'T'
function [N,T] = variableInitialization
    N.CB.XO  = 889392;  % Initial slag in bulk concentrate, mol
    N.CB.XS  = 553568; % Initial matte in bulk concentrate, mol
    N.CB.XS2 = 1035;      % Initial sulphurized matte in bulk concentrate, mol
    T.CB = 1036; % Initial bulk concentrate temperature, K

    N.CS.XO  = 771358;  % Initial slag in smelting concentrate, mol
    N.CS.XS  = 559284; % Initial matte in smelting concentrate, mol
    N.CS.XS2 = 1.0687;      % Initial sulphurized matte in smelting concentrate, mol
    T.CS = 1379; % Initial smelting concentrate temperature, K

    N.CR     = 555; % Initial reaction gas in concentrate bed, mol

    N.S      = 11127185; % Initial liquid slag, mol
    T.S  = 1903; % Initial slag zone temperature, K
```

```
  N.M     = 6602984; % Initial liquid matte, mol
  T.M  = 1755; % Initial matte zone temperature, K

  N.G.A    = 1082.57;  % Initial air in freeboard, mol
  N.G.R    = 702.99;     % Initial reaction gas in freeboard, mol
  T.G  = 1023.8;  % Initial freeboard temperature, K

  T.W  = 311;  % Initial cooling units temperature, K
End

% Load model parameters and store them in structure 'p'
function p = parameters
  % Molar masses, kg/mol
  p.M.G   =  30e-3; % Reaction gases/air
  p.M.XO  =  72e-3; % Lumped slag components
  p.M.XS  =  88e-3; % Lumped matte components
  p.M.XS2 = 120e-3; % Lumped sulfurized components

  % Densities, kg/m^3
  p.D.C = 1600; % Concentrate density
  p.D.S = 2960; % Slag density
  p.D.M = 4800; % Matte density

  % Heat of fusion, kJ/mol
  p.fus.C = 133; % Concentrate heat of fusion
  % Heat capacity, kJ/mol.K
  p.cP.C =  75e-3; % Concentrate heat capacity
  p.cP.S =  99e-3; % Slag heat capacity
  p.cP.M =  78e-3; % Matte heat capacity
  p.cP.G =  30e-3; % Reaction gases/air heat capacity
  p.cP.W =  75e-3; % Cooling water heat capacity
```

```
% Activation energy, J/mol
p.EA.F =  150e3; % Desulphurization reaction
p.EA.C =  120e3; % Electrode oxidation reaction
% Rate-determining constants
p.k.V   = 2e-4;   % Concentrate mixing constant
p.k.PR  = 7;      % Freeboard- to atmosphere flow constant, mol/Pa.s
p.k.PE  = 3;      % Freeboard extraction pressure flow constant, mol/Pa.s
p.k.rF  = 1.0e5;  % Desulphurization rate constant, 1/s
p.k.rC  = 1.25e4; % Electrode oxidation rate constant, mol/s
p.k.PBR = 1.0e-8; % PBR flux constant, mol/m.Pa.s
p.k.Ch  = 2e-6;   % Channeling flux constant, mol/m^2.Pa.s
% Furnace dimensions
p.dim.A = 300; % Bath area, m^2
p.dim.p = 80;  % SAF perimeter, m
p.dim.V = 150; % Freeboard volume, m^3
% Heat generation & transfer constants
p.Q.Velectrode = 120;     % Electrode voltage, V
p.Q.alpha      = 0.0003;  % Joule heating constant, V^2/kW.K
p.Q.R0         = 0.21;    % Slag resistivity at 1900 K, V^2/kW
p.Q.T0         = 1900;    % Base slag temperature, K
p.Q.hCS2CB     = 0.0275;  % Heat transfer coefficient between smelting- and
                          % bulk concentrate, kW/m^2.K
p.Q.hG2CB      = 0.055;   % Heat transfer coefficient between freeboard and
                          % bulk concentrate, kW/m^2.K
p.Q.hS2CS      = 0.31;    % Heat transfer coefficient between slag and
                          % smelting concentrate, kW/m^2.K
p.Q.hM2S       = 0.085;   % Heat transfer coefficient between matte and
                          % slag, kW/m^2.K
p.Q.hW2S       = 0.007;   % Heat transfer coefficient between cooling units
                          % and slag, kW/m.K
p.Q.hW2M       = 0.027;   % Heat transfer coefficient between cooling units
                          % and matte, kW/m.K
```

```
    % Other constants
    p.other.R = 8.314; % Universal gas constant, J/mol.K
    p.other.g = 9.81;  % Gravity acceleration, m/s^2
    p.other.e = 0.4;   % Concentrate bed voidage
    p.other.Patm = 101325; % Atmospheric pressure, Pa
    p.other.Pext = 101315; % Extraction pressure, Pa
    p.other.Tcharge = 700; % Concentrate charge temperature, K
    p.other.Tmelt = 1500;  % Concentrate melting temperature, K
    p.other.Tatm  = 300;   % Atmosphere temperature, K
    % Cooling water constants
    p.water.F  = 2400;  % Cooling water flowrate, mol/s
    p.water.N  = 10000; % Cooling water molar amount, mol
    p.water.T0 = 300;   % Cooling water temperature, K
end


%% 11. Representation subfunctions
% Convert a column vector of state variables, x, to structures 'N' and 'T'
function [N,T] = ode2state(x)
    N.CB.XO  = x(1);
    N.CB.XS  = x(2);
    N.CB.XS2 = x(3);
    T.CB     = x(4);
    N.CS.XO  = x(5);
    N.CS.XS  = x(6);
    N.CS.XS2 = x(7);
    T.CS     = x(8);
    N.CR     = x(9);
    N.S      = x(10);
    T.S      = x(11);
    N.M      = x(12);
    T.M      = x(13);
    N.G.A    = x(14);
```

```
    N.G.R    = x(15);
    T.G      = x(16);
    T.W      = x(17);
end
% Convert state variables stored in 'N' and 'T' to various process variables
function [N,V,C,L,P] = state2derived(N,T,p)
    N.CB.total = N.CB.XO + N.CB.XS + N.CB.XS2;
    N.CS.total = N.CS.XO + N.CS.XS + N.CS.XS2;
    N.G.total  = N.G.A + N.G.R;

    V.CB = (p.M.XO*N.CB.XO+p.M.XS*N.CB.XS+p.M.XS2*N.CB.XS2)/p.D.C;
    V.CS = (p.M.XO*N.CS.XO+p.M.XS*N.CS.XS+p.M.XS2*N.CS.XS2)/p.D.C;
    V.C = V.CB+V.CS;

    C.CB.XO  = N.CB.XO/V.CB;
    C.CB.XS  = N.CB.XS/V.CB;
    C.CB.XS2 = N.CB.XS2/V.CB;
    C.CS.XS2 = N.CS.XS2/V.CS;

    L.S = p.M.XO*N.S/(p.D.S*p.dim.A);
    L.M = p.M.XS*N.M/(p.D.M*p.dim.A);
    L.C = (V.CB+V.CS)/p.dim.A;

    P.G  = N.G.total*p.other.R*T.G/p.dim.V;
    P.CR = N.CR*p.other.R*T.CS/(p.other.e*V.C);
end
% Convert structure of state variables to column vector to use in ODE solver
function dx = state2ode(dN,dT)
    dx(1,1) = dN.CB.XO;
    dx(2,1) = dN.CB.XS;
    dx(3,1) = dN.CB.XS2;
    dx(4,1) = dT.CB;
```

```
    dx(5,1) = dN.CS.XO;
    dx(6,1) = dN.CS.XS;
    dx(7,1) = dN.CS.XS2;
    dx(8,1) = dT.CS;
    dx(9,1) = dN.CR;
    dx(10,1) = dN.S;
    dx(11,1) = dT.S;
    dx(12,1) = dN.M;
    dx(13,1) = dT.M;
    dx(14,1) = dN.G.A;
    dx(15,1) = dN.G.R;
    dx(16,1) = dT.G;
    dx(17,1) = dT.W;
end
% Convert matrix of ODE data generated by 'ode15s' to a structure of process variables
function profileVariables = ode2profile(odeVariables)
    g = odeVariables;
    f.N.CB.XO  = g(:,1); % Slag in bulk concentrate, mol
    f.N.CB.XS  = g(:,2); % Matte in bulk concentrate, mol
    f.N.CB.XS2 = g(:,3); % Sulphurized matte in bulk concentrate, mol
    f.T.CB     = g(:,4); % Bulk concentrate temperature, K
    f.N.CS.XO  = g(:,5); % Slag in smelting concentrate, mol
    f.N.CS.XS  = g(:,6); % Matte in smelting concentrate, mol
    f.N.CS.XS2 = g(:,7); % Sulphurized matte in smelting concentrate, mol
    f.T.CS     = g(:,8); % Smelting concentrate temperature, K
    f.N.CR     = g(:,9); % Reaction gases in concentrate, mol
    f.N.S      = g(:,10); % Liquid slag, mol
    f.T.S      = g(:,11); % Liquid slag temperature, K
    f.N.M      = g(:,12); % Liquid matte, mol
    f.T.M      = g(:,13); % Liquid matte temperature, K
    f.N.G.A    = g(:,14); % Air in freeboard, mol
    f.N.G.R    = g(:,15); % Reaction gases in freeboard, mol
```

```
f.T.G      = g(:,16); % Freeboard gas temperature, K

f.T.W      = g(:,17); % Cooling units temperature, K


p = parameters;


f.N.CB.total = f.N.CB.XO + f.N.CB.XS + f.N.CB.XS2; % Total bulk concentrate, mol

f.N.CS.total = f.N.CS.XO + f.N.CS.XS + f.N.CS.XS2; % Total smelting concentrate, mol

f.N.G.total  = f.N.G.A + f.N.G.R; % Total freeboard gas, mol


% Bulk concentrate volume, m^3:

f.V.CB = (p.M.XO*f.N.CB.XO+p.M.XS*f.N.CB.XS+p.M.XS2*f.N.CB.XS2)/p.D.C;

% Smelting concentrate volume, m^3:

f.V.CS = (p.M.XO*f.N.CS.XO+p.M.XS*f.N.CS.XS+p.M.XS2*f.N.CS.XS2)/p.D.C;

f.V.C = f.V.CB+f.V.CS; % Total concentrate volume, m^3


f.C.CB.XO  = f.N.CB.XO./f.V.CB;  % Bulk concentrate slag concentration, mol/m^3

f.C.CB.XS  = f.N.CB.XS./f.V.CB;  % Bulk concentrate matte concentration, mol/m^3

f.C.CB.XS2 = f.N.CB.XS2./f.V.CB; % Bulk concentrate sulphurized matte concentration, mol/m^3

f.C.CS.XS2 = f.N.CS.XS2./f.V.CS; % Smelting conc sulphurized matte concentration, mol/m^3

f.C.G.R    = f.N.G.R/p.dim.V;    % Freeboard reaction gas concentration, mol/m^3


f.L.S = p.M.XO*f.N.S/(p.D.S*p.dim.A); % Slag liquid level, m

f.L.M = p.M.XS*f.N.M/(p.D.M*p.dim.A); % Matte liquid level, m

f.L.C = (f.V.CB+f.V.CS)/p.dim.A;      % Concentrate bed thickness, m


f.P.G  = f.N.G.total*p.other.R.*f.T.G/p.dim.V; % Freeboard pressure, Pa

f.P.CR = f.N.CR*p.other.R.*f.T.CS...

            ./(p.other.e.*f.V.C); % Reaction gas in concentrate pressure, Pa


f.Q.S2CS  = p.Q.hS2CS*p.dim.A*(f.T.S-f.T.CS); % Heat transfer from slag to concentrate, kW

% Heat generated in slag zone, kW:

f.Q.J     = (p.Q.Velectrode^2)./(p.Q.R0*(1+p.Q.alpha*(f.T.S-p.Q.T0)));
```

```
    % Concentrate smelting rate, mol/s:
    f.F.melt.total = f.Q.S2CS.*(f.T.CS/p.other.Tmelt)./(p.fus.C-p.cP.C*(p.other.Tmelt-f.T.CS));
    % Electrode oxidation rate, mol/s:
    f.r.C  = p.k.rC*exp(-p.EA.C./(p.other.R*f.T.S));


    profileVariables = f;
end
```

D.3. Implementation of model-based and data-driven fault detection

The main script is presented for performing the fault detection using the state estimates obtained via the script in appendix D.1. However, now the measurements are generated using any of the faulty conditions below:

```
faultindex = ; % for obtaining faulty measurements
            % 1 = Cooling water flowrate step change at t = 1 day
            % 2 = Pext step change at t = 1 day
            % 3 = Concentrate composition step change at t = 1 day
            % 4 = Blowback fault at t = 1 day
```

And the smaller process noise covariance is employed for generating the state estimates.

```
if faultindex > 0
    ProcNoise_objective3 = 0.01.*ProcNoise_objective2;
    % Process noise for objective 3 is 1% of the process noise used in objective 2
    ProcNoise = ProcNoise_objective3;
end
```

The EKF and PF are used to generate the innovation terms. The innovation terms represent a sequence of residuals from the state estimators.

```
% Extended Kalman filter
[xhat_EKF, P_EKF, inn_EKF] = EKF(estimationlength,N_meas,xhat_0,P0,R,Q,y,model_type);
% Particle filter
[xhat_PF, P_PF, inn_PF,vhat_matrix] = PF(estimationlength,N_meas,xhat_0,P0,R,Q,y,n,model_type);
```

The three fault detection methods investigated are 1) model-based fault detection using the EKF residuals, 2) model-based fault detection using the PF residuals, and 3) data-driven fault detection using the measurements. Select the FDI method to select the corresponding dataset.

```
% Investigate the performance three fault detection methods:
% FDI_method 1: Model-based fault detection via EKF residuals & PCA
% FDI_method 2: Model-based fault detection via PF residuals & PCA
% FDI_method 3: Data-driven fault detection via measurements & PCA
FDI_method = ;


% Datasets for each method are:
% 1: inn_EKF (residuals from the EKF)
% 2: inn_PF (residuals from the PF) & vhat_matrix (residuals from each
% particle of the PF)
% 3: y (measurements)
if FDI_method == 1
   X = inn_EKF;
end
if FDI_method == 2
   X = inn_PF;
end
if FDI_method == 3
   X = y';
end
```

Split the datasets into training data, ‘Xtrain’, and testing data, ‘Xtest’ using the ‘DataSeparation’ function. For the model-based fault detection using the PF residuals, the individual particle innovation sequences stored in ‘vhat_matrix’ are split into testing data ‘vhat_matrix_test’. The ‘DataSeparation’ function also calculates the minimum specificity, ‘minSpec’, required to obtain a minimum precision of 95%, and creates logic vectors ‘iNominal’, containing 1’s at nominal conditions in the test data and 0’s elsewhere, and ‘iFault’, containing 1’s at faulty conditions in the test data and 0’s elsewhere.

```
[Xtrain, Xtest, minPrec, minSpec, iNominal, iFault, vhat_matrix_test] = DataSeparation(X, ...
                                                    N_meas, vhat_matrix, FDI_method);
```

```
function [Xtrain, Xtest, minPrec, minSpec, iNominal, iFault, vhat_matrix_test] = ...
                                    DataSeparation(X, N_meas, vhat_matrix, FDI_method)
  % Specify the training data set:
  % Train the PCA model on data generated under nominal conditions
  traintime = 0.5*3600*24; % s, time at which training set ends
                        % PCA models are trained on residuals/measurements generated under
                        % nominal conditions for half a day of data
  itrain = traintime/N_meas; % index at which training set ends
  Xtrain = X(2000:itrain,:); % Training data set
                         % Start training at index 2000 to avoid large residuals generated
                         % due to initialization error of the filters

  % Specify the test data set:
  faulttime = 1*3600*24;     % s, all faults are simulated at time 1 day in this study
  ifault = faulttime/N_meas; % index at which fault is simulated
  Xtest = X(itrain:end,:);   % Test data set consists of 0.5 days of nominal data and 0.5 days
                         % of faulty data
  vhat_matrix_test = [];
  if FDI_method == 2
     vhat_matrix_test = vhat_matrix(:,:,itrain:end); % If PF residuals are the test data, the
                                          % vhat_matrix obtained from the PF
                                          % algorithm is used to calculate the
                                          % likelihood monitoring statistic
     % 'vhat_matrix_test' represents the residuals obtained for individual particles at each
     % timepoint in the test data set
  end

  % Specify the faulty and nominal indices in test data set and store them in logic vectors
  t = 1:size(Xtest,1);
  iFault= sin(pi*t/(ifault-itrain) < 0; % Generate a 1 at each timestep the fault is
                                  % occurring, otherwise generate a zero
  iNominal = ~iFault;                % Generate a 1 at each timestep that nominal conditions
```

```
                    % are occurring, otherwise generate a zero
    minPrec = 0.95; % User-specified minimum precision of 95%
    F = sum(iFault);   % Total number of faulty observations
    N = sum(iNominal); % Total number of nominal observations
    minSpec = 1+(F/N)*((minPrec-1)/minPrec); % Minimum specificity required for minPrec
end
```

Train the PCA models using the function 'PCA_Model'. The trained PCA models are represented by the PCs contained in the loadings matrix, V. The PCA models can be constructed by retaining one through 'rmax' PCs.

Function inputs	
X	Matrix consisting of the training data. For FDI method 1 and 2, the training data consists of residuals from the EKF and PF, respectively, at the timepoints for which the training data was acquired. For FDI method 3, the training data consists of the measurements from the SAF process at the timepoints for which the training data was acquired.
r	Indicates the maximum number of retained PCs.
Function outputs	
V	Loadings matrix of the trained PCA model. The columns of the loadings matrix represent each PC.
mu	Vector containing the training data means for each residual/measured variable.
sig	Vector containing the training data standard deviation for each residual/measured variable.
T	Scores matrix of the trained PCA model.
sa	Vector containing the variance of the training scores for each number of PCs retained.
cumvar	Vector containing the cumulative variance explained by each of the PCs.

```
rmax = size(X,2); % specify the maximum number of retained PCs
[V,mu,sig,T,sa,cumvar] = PCA_Model(Xtrain,rmax); % derive PCA model
```

```
function [V,mu,sig,T,sa,cumvar] = PCA_Model(X,r)
   mu = mean(X,1);  % Calculate mean of data set
   sig = std(X,1);  % Calculate standard deviation of data set
   S = (X - mu)./sig; % Standardize data set
   C = (S')*S;      % Compute the correlation matrix
   [P,D] = eig(C);  % Compute the eigenvectors which represent the PCs
   D = diag(D);     % Eigenvalues of PCs
   [sigma,i] = sortrows(D,'descend'); % Sort eigenvalues from largest to smallest
   P = P(:,i);   % Sort corresponding eigenvectors
   V = P(:,1:r); % Loadings matrix retaining r PCs
   T = Z*V;      % Scores matrix
   % Calculate the variance of the training scores
   for a = 1:r
      sa(a) = var(T(:,a));
   end
   % Calculate the cumulative variance explained for each retained PC
   vari = (sigma./sum(sigma))*100; % percentage variance explained
   cumvar = cumsum(vari);         % cumulative percentage variance explained
end
```

The fault detection performance of the PCA models is tested using the test data, Xtest. Each of the PCA models, retaining 1 through 'rmax' PCs are investigated. The appropriate monitoring statistics are calculated for each data entry in Xtest. The specificity and sensitivity are obtained using various threshold values for the monitoring statistic. The threshold value defines the minimum value that a monitoring statistic must be to flag the statistic as faulty. The partial area under the ROC curve is calculated from a ROC curve generated from the specificities and sensitivities obtained by varying the threshold of the monitoring statistic. The $pAUC$ is used to evaluate the fault detection performance.

```
% For each number of retained PCs in the PCA model
for r = 1:rmax
    % For all FDI methods, the reconstruction error represents the out-of-plane error of the
    % PCA model
    % Calculate the reconstruction error using the PCA model defined by the PCs 1:r calculated
    % from the loadings matrix V(:,1:r)
    E(:,r) = ReconstructionErrorPCA(Xtest,V(:,1:r),mu,sig);
    % Calculate the specificity and sensitivity obtained at various thresholds for the
    % reconstruction error
    [specificity_E(:,r),sensitivity_E(:,r)] = SpecificitySenstivity(E(:,r), ...
                                        iFault,iNominal);
    % Calculate the partial area under the ROC curve generated using the specificities and
    % sensitivities calculated above
    pAUC_E(r) = PartialAUC(specificity_E(:,r),sensitivity_E(:,r),minSpec);

    % For FDI_method 1, model-based fault detection using the EKF residuals and FDI_method 3,
    % data-driven fault detection using the measurements, the within-plane error of the PCA
    % model is assessed via Hotelling's T2 statistic
    if FDI_method == 1 | FDI_method == 3
        % Calculate Hotelling's T2 statistic using the PCA model defined by the PCs 1:r
        T2(:,r) = HotellingsT2PCA(Xtest,V(:,1:r),mu,sig,sa(1:r));
        % Calculate the specificity and sensitivity obtained at various thresholds for
        % Hotelling's T2 statistic
        [specificity_T2(:,r),sensitivity_T2(:,r)] = SpecificitySenstivity(T2(:,r), ...
                                            iFault,iNominal);
        % Calculate the partial area under the ROC curve generated using the specificities and
        % sensitivities calculated above.
        pAUC_T2(r) = PartialAUC(specificity_T2(:,r),sensitivity_T2(:,r),minSpec);
    end

    % For FDI_method 2, model-based fault detection using the PF residuals, the within-plane
    % error of the PCA model is assessed using the likelihood monitoring statistic
```

```
  if FDI_method == 2
    % Calculate the likelihood using the PCA model defined by the PCs 1:r
    likelihood(:,r) = LikelihoodPCA(vhat_matrix_test,V(:,1:r),R);
    % Calculate the specificity and sensitivity obtained at various thresholds for
    % the likelihood monitoring statistic
    [specificity_likelihood(:,r),sensitivity_likelihood(:,r)] =  ...
                SpecificitySenstivity(likelihood(:,r),iFault,iNominal);
    % Calculate the partial area under the ROC curve generated using the specificities and
    % sensitivities calculated above.
    pAUC_likelihood(r) = PartialAUC(specificity_likelihood(:,r), ...
                sensitivity_likelihood(:,r),minSpec);
  end
end
```

For all three fault detection methods investigated, the out-of-plane error of the PCA model is quantified using the reconstruction error. For the model-based fault detection using the EKF residuals and the data-driven fault detection, the within-plane error of the PCA model is captured by Hotelling's T^2 statistic. For the model-based fault detection using the PF residuals, the within-plane error is quantified using the likelihood statistic.

Function inputs	
X	Matrix consisting of the test data. For FDI method 1 and 2, the test data consists of residuals from the EKF and PF, respectively, at the timepoints for which the test data was acquired. For FDI method 3, the test data consists of the measurements from the SAF process at the timepoints for which the test data was acquired.
V	Loadings matrix of the trained PCA model. The columns of the loadings matrix represent each PC retained.
mu	Vector containing the training data means for each residual/measured variable.
sig	Vector containing the training data standard deviation for each residual/measured variable.
sa	Vector containing the variance of the training scores for each number of PCs retained.

vhat_matrix	Three dimensional matrix from the PF state estimation algorithm containing the innovations for each of the particles at each timestep. The number of elements in the first dimension is equal the number of measured variables. The number of elements in the second dimension is equal to the number of particles. The number of elements in the third dimension is equal to the number of timesteps which the test data is obtained at.
R	Measurement noise covariance matrix.
Function outputs	
E	Vector containing the reconstruction errors calculated for each sample in the test data, X.
T2	Vector containing Hotelling's T^2 statistic calculated for each sample in the test data, X. The T^2 statistic is calculated in the reduced-dimensional space defined by the trained PCA model represented by the PCs in V.
likelihood	Vector containing the -log of the sum of likelihoods calculated for each sample of particles from the PF contained in the test data, vhat_matrix. The likelihoods of each particle are calculated in the reduced-dimension space defined by the trained PCA model represented by the PCs in V.

```
function E = ReconstructionErrorPCA(X,V,mu,sig)
   S = (X - mu)./sig;      % Standardize the test dataset
   Sr = S*V*(V');        % Reconstruct the data set using the PCs
   e = S-Sr;             % Difference between the PCA model prediction for the observations
                         % and the true observations
   e2 = e.^2;            % Squared difference
   E = sum(e2'); % Calculate reconstruction error as the sum of the squared diff
end


function T2 = HotellingsT2PCA(X,V,mu,sig,sa)
   S = (X - mu)./sig; % Standardize the test data set
   T_test = S*V;    % Calculate the scores matrix for test data
   % For each data point, i, in test data set:
   for i = 1:size(T_test,1)
      T2(i) = sum((T_test(i,:)./sa).^2); % Hotelling's T2 statistic
```

```
    end
end


function likelihood = LikelihoodPCA(vhat_matrix,V,R)
% vhat_matrix is 3D matrix of the residuals for each particle of the PF
% 1st dimension = Measurement variables
% 2nd dimension = Particles
% 3rd dimension = Test data points
    % For each test data point j
    for j = 1:size(vhat_matrix,3)
        vhat = vhat_matrix(:,:,j);
        qsum = 0;
        % For each particle calculate the likelihood in the reduced-dimension
        % space defined by the trained PCA model represented by the loadings
        % matrix V
        for i = 1 : size(vhat)
            R_reduced = V'*R*V; % Covariance matrix in reduced space
            vhat_reduced = V'*vhat(:,i); % Residual in reduced space
            q = (( inv((det(R_reduced)^0.5) * (2*pi)^(size(V,2)/2)) )...
                * ( exp((-(vhat_reduced)'*inv(R_reduced)*vhat_reduced*0.5)) ));
            % Likelihood in reduced space
            qsum = qsum + (q); % Sum of likelihoods
        end
        likelihood(j) = -log(qsum); % Negative log of sum of likelihoods
    end
end
```

Additional functions for calculating the specificity, sensitivity, and the partial area under the curve are presented below.

Function inputs

monitoring_statistic	Vector containing the monitoring statistics (reconstruction error, Hotelling's T^2 statistic, or the likelihood statistic) obtained for each new sample in the test data.
iFault	Vector containing 1's where faulty conditions are present in the test data and 0's where nominal conditions are present.
iNominal	Vector containing 1's where nominal conditions are present in the test data and 0's where faulty conditions are present.
minSpec	The minimum specificity required to achieve a precision of 95%.
Function outputs	
specificity	Vector containing the specificities calculated by varying the threshold defining the monitoring statistic as faulty.
sensitivity	Vector containing the sensitivities calculated by varying the threshold defining the monitoring statistic as faulty.
pAUC	The partial area under the ROC curve defining the area under the curve where a minimum precision of 95% is achieved.

```
function [specificity,sensitivity,nThresh] = SpecificitySenstivity(...
                                monitoring_statistic, iFault, iNominal)
  nThresh = 500; % The number of thresholds investigated
  % Calculate monitoring statistic value that categorizes observation as faulty at each
  % threshold value
  ms = (repelem(monitoring_statistic,1,nThresh)-linspace(0,100,nThresh))>=0;
  % For each threshold value:
  TP = sum(ms(iFault,:),1);    % Number of true positives
  TN = sum(~ms(iNominal,:),1); % Number of true negatives
  FP = sum(ms(iNominal,:),1);  % Number of false positives
  FN = sum(~ms(iFault,:),1);   % Number of false negatives
  F = TP + FN;   % Total number of faulty observations
  N = FP + TN; % Total number of nominal observations
  specificity = TN./N;   % Specificity
  sensitivity = TP./F;   % Sensitivity
end
```

```
function pAUC = PartialAUC(specificity,sensitivity,minSpec)

    nThresh = 300; % Number of threshold points evaluated

    % ROC evaluated between the minSpec required for minPrec and the maxSpec = 1:

    pSpec = linspace(minSpec,1,nThresh);

    [~,idx] = unique(specificity); % Indices of the unique specificities

    pSens = interp1(specificity(idx),sensitivity(idx),pSpec); % Sensitivities required

                                                % for minPrec

    pAUC = trapz(pSpec,pSens)/(1-minSpec); % Partial area under the curve

end
```