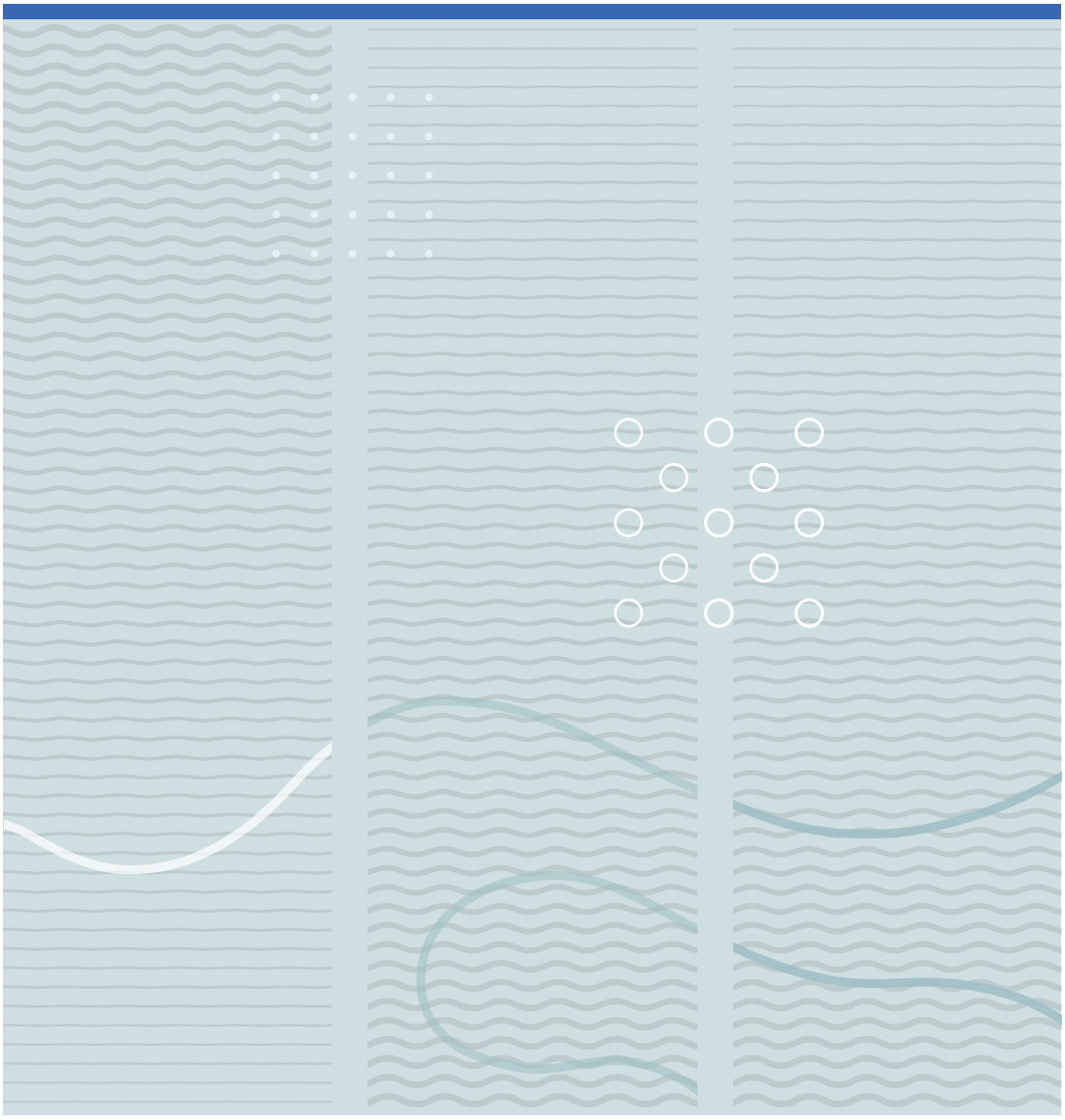
University College of Southeast Norway
Faculty of Technology

Doctoral dissertation no. 6
2016

Magamage Anushka Sampath Perera

State Estimation and Optimal Control of an Industrial Copper Electrowinning



Magamage Anushka Sampath Perera

State Estimation and Optimal Control of an Industrial Copper Electrowinning

A PhD dissertation in
Process, Energy and Automation Engineering

Faculty of Technology
University College of Southeast Norway
Kongsberg, 2016

Doctoral dissertations at the University College of Southeast Norway no. 6

ISSN: 2464-2770 (print)
ISSN: 2464-2483 (online)
ISBN: 978-82-7206-417-3 (print)
ISBN: 978-82-7206-418-0 (online)

Print: **University College of Southeast Norway**

To my mother, Jenat Fernando

Preface

This dissertation is prepared for the degree of Philosophiae Doctor (PhD) offered by the University College of Southeast Norway. The research was partially founded by Glencore Nikkelverk, Kristiansand. During the period of 2011-2014, professor Bernt Lie was my main supervisor. From 2014, the main supervisor was professor Carlos Pfeiffer and the co-supervisor was professor Bernt Lie. Both professors work in the Department of Process, Energy and Automation at the University College of Southeast Norway. The research is related to the copper leaching process, which is a subprocess at the Nikkelverk. Dr. Tor Anders Hauge has been my contact person on behalf of the Nikkelverk, he works there as a senior control engineer.

This is an applied research work, where the research objectives were initially suggested by the Nikkelverk, and have been modified and extended later on as the research progressed. In general, my research domain lies within the areas of Applied State-Parameter-Disturbance Estimation and Optimal Control of Large-Scale Complex Systems. The Copper leaching process is a large-scale complex system, hence it is a good candidate to be used in the case study. Even though this is a specific research problem, the research accomplishments can be extended to handle a much broader class of problems appearing within systems and control engineering. Three research papers have already been published (one in 55th International Conference on Simulation and Modelling and the other two in the journal of Modeling, Identification and Control) and one will be published in the International Journal of Modeling and Optimization (IJMO) Vol. 6, No. 5. October 2016.

I have enjoyed the time I have spent with the research work, and I am happy to finally complete my PhD. It is my hope that my contribution will be useful for others who work within the field of systems and control engineering.

Magamage Anushka Sampath Perera

Porsgrunn, Norway
August 2016

Acknowledgments

It is my pleasure to acknowledge those who have supported me throughout my PhD studies in many ways. First of all, my heartfelt gratitude goes to my supervisors Professor Carlos F. Pfeiffer and Professor Bernt Lie for their immense support. Professor Bernt Lie and Dr. P. G. Rathnasiri were the ones who initially inspired me to pursue the subject Systems and Control Engineering, and I am sincerely thankful to them both.

Thanks to Glencore Nikkelverk for partially funding my research. In particularly, I appreciate Dr. Tor Anders Hauge's contributions.

I am grateful to Randi Toreskås Holta, Eldrid Eilertsen and Inger Johanne Kristiansen for all their support during the completion of my PhD. I would also like to thank my teachers at the Dept. of Chemical and Process Engieering, University of Moratuwa, Sri Lanka as well as the University College of Southeast Norway.

Special thanks to Robin Omlid Wold, who has always made his IT skills and resources freely available for me and who's help saved me in many occasions. I am also grateful to Dietmar Winkler for helping me to handle many Linux-OS related installation hassles as well as providing support related to Modelica.

Thanks to all of my loving and caring friends who have been around me during my stay in Norway since 2009.

Ingrid Bokn Haugland assisted me with the proofreading of the thesis. It was a huge help for me, her effort is very much appreciated ♡

Finally, my gratitude goes towards my beloved parents (Wilfred Perera - Jenat Fernando), brother (Asanka), sister (Achala) and late grand father (Benedict Fernando) for all of their dedications.

Summary

This dissertation contains a solution for an industrial large-scale complex control problem. There are several challenges to be faced when handling control problems. One such challenge is to be able to handle the system's scale. Developing mathematical models for large-scale complex control systems have become easier with the emergence of object-oriented, declarative, multi-domain modeling languages for component-oriented modeling, such as Modelica. However, the analysis and synthesis of large-scale complex control systems are still highly demanding; new tools and methodologies are needed to simplify the task. The need of making Modelica models available for general use is emphasized. In order to stress the idea, a paper (Appendix A) is published where we present a case study demonstrating how to combine Modelica models and the Python Control Systems Library.

The central interest of this applied research is to implement optimal control strategies in relation to large-scale complex systems. Generally, an optimal control problem poses a state-parameter-disturbance estimation problem. Both these types of problems are challenging due to the scale and the complexity of the systems of interest. Even for moderately small systems, the algebraic analysis and synthesis may become complicated, tedious or even impractical. When investigating a system, several questions needs to be answered, such as if the system is controllable, if its state is observable, if parameters and disturbances can be estimated, if there are any disturbances which can be completely rejected via some state feedback and how to select appropriate models for disturbance augmentation. It is difficult to answer these questions using analytical tools, thus a new perspective is necessary. Interestingly, there are system properties which (almost) do not depend on the actual values of model parameters. These properties are so-called generic or structural properties, such as controllability and observability. The decisions (for example, about algebraic controllability) which are made by means of the state-space theory are susceptible to the uncertainties in the model. Therefore, a better approach is to structurally decompose the system into smaller systems and then analyze these subsystems. In large-scale systems, the state variables, inputs, disturbances and outputs are usually affecting each other. These dependencies often exhibit a nice sparsity pattern. Variable dependencies are captured by the system's structure. Generic properties can be investigated with the use of graph theory. This approach constitutes a very powerful and simple-to-use tool for structural analysis. The structural investigation in the analysis of large-scale complex nonlinear control systems is a significant portion of this research. All proposed methodologies are tested using simulated data related to the copper leaching process. A mechanistic model containing many state variables (more than 50), unknown disturbances and uncertain parameters along with 4 output measurements is available. The system in question is of large-scale and the process is highly coupled. Consequently, the copper leaching process is an ideal candidate for testing the methodologies presented in the dissertation. The following paragraphs discuss the main milestones of my research work.

Linearization of DAEs: In general, mechanistic models of real physical systems are systems of High-Index Differential-Algebraic Equations (DAEs). The JModelica.org-CasADi interface provides a way of making Modelica models available in Python as symbolic DAE systems. Python is a powerful programming language for technical computing. This opens up a possibility

of linearizing any Modelica model in Python. The procedure is demonstrated in [1].

Linear Analysis: Linear system theory is often used in nonlinear system analysis and design. There has been a lack of a general control tool for Python until the Python Control Systems Library (the python-control package) was developed. However, the Python Control Systems Library is underdeveloped as compared to MATLAB's Control System Toolbox. An article is published showing a procedure of making Modelica models available for linear analysis in Python [2].

Structural Observability/Controllability Analysis: Estimating the internal state of a given dynamic system based on input-output information is crucial for optimal control. However, for some systems it may not be possible to uniquely estimate the state. Such systems are called unobservable systems. It is possible to define algebraic conditions for observability. For linear time invariant systems, we may for example check the rank of the observability matrix. For nonlinear systems, local observability should be checked. An observability rank condition is always associated with the rank of a matrix. For large-scale complex systems, the rank test becomes impractical to implement. An algebraic test merely tells whether the system is observable or not. Sometimes, the inverse observability problem could be more interesting, that is; defining the (minimum number of) output measurements which is needed to make the system observable. A rank test would not solve the inverse observability problem in an easy way, while the structural observability test does. Although the system's structure provides valuable information, it still does not provide all we need — i.e., structural conditions always provide necessary, but not sufficient, conditions. For example, if a system fails to achieve the structural observability condition, then the system is (locally) unobservable. The converse is however not always true. Thus, structural observability/controllability analysis of a system should be conducted as follows: (1) Analyze the large system for structural observability; (2) as a consequence of structural observability analysis, we can decompose the system into observable and unobservable subsystems; and (3) these subsystems are analyzed algebraically as necessary. Structural observability analysis can be done with the aid of a graph-theoretic analogy which makes the analysis much simpler. A way of automating structural observability analysis using the NetwokX and PyGraphviz Python packages, where the copper leaching process is used as an example, is presented in [3].

State-Parameter-Disturbance Estimation: In reality, it is impractical or impossible to measure all the state variables. It is however often possible to reconstruct the system's state based on a finite set of input-output measurements, providing that the state is observable. The Extended Kalman Filter, the Moving Horizon Estimate and the Unscented Kalman Filter are examples of available nonlinear estimators. An implementation of several state estimators in Python is presented in [4].

Optimal Control: The final objective of this research is to implement an optimal control strategy. An optimal control strategy is always coupled with an appropriate state estimator. In order to determine the control trajectory, the state information should be known. In this work, the Moving Horizon Estimator is used as the nonlinear estimator. A complete implementation is presented in [5], where two uncertain parameters and two unmeasured disturbances are estimated. [5] also contain a discussion about the divergence issue of the Extended Kalman Filter. It is shown that by some modifications to the Extended Kalman Filter algorithm, we can get comparable results from both of the estimators.

The dissertation contains two parts: Part I and Part II. Part I explains theories and methodologies used in Part II and additional information which is not presented in Part II. Part II consists of a list of published and submitted articles.

Contents

List of Figures

List of Tables

Nomenclature

Superscript & Subscripts

$\mu_j^{(i)}$	Reaction rate per unit volume connected to the tank i within the section j
$\rho_{j,k}^{(i)}$	Mass concentration of the component k connected to the tank i within the section j $[kg.m^{-3}]/[g.l^{-3}]$
$V_j^{(i)}$	Volume of the tank i within the section j

Chemical Compounds

CuO	Copper oxide
$CuSO_4$	Copper sulfate
CuS	Copper sulfide
H_2O	Water
H_2SO_4	Sulfuric acid
$NiCl_2$	Nickel chloride
O_2	Oxygen
SO_2	Sulfur dioxide

Chemical Elements

Ag	Silver
Au	Gold
Bi	Bismuth
Co	Cobalt
Cu	Copper
Ni	Nickel
Pd	Palladium
Pt	Platinum
Se	Selenium
S	Sulfur
Te	Tellurium

Greek Symbols

ε_i	Void fraction of i^{th} cementation tank	
μ	Reaction rate per unit volume	$[mol \cdot m^{-3} \cdot s^{-1}]$
ρ	Mass concentration	$[kg \cdot m^{-3}]$
ρ_i	Mass concentration of chemical component i	
τ	Time constant or time delay	$[s]/[min]/[h]$
$\tau_{\dot{a}}$	Time delay connected to $\dot{a}$	

Chemical Ions

Bi^{4+}	Bismuth (IV)
Cu^{2+}	Copper (II)
SO_4^{2-}	Sulfate

Subscripts

(aq)	Aqueous solution
(g)	Gas phase
(l)	Liquid phase
(s)	Solid phase
$\dot{V}_{i2j}$	Volumetric flow rate from unit i to unit j
c_i	Molar concentration of chemical component i
M_i	Molar mass of chemical component i

Other Symbols

e	Electron
$\dot{a}$	Mass/volume flow rate or time derivative of the property a $[kg.s^{-1}]/[m^3.s^{-1}]$
$\dot{V}_a$	Volumetric flow rate of the H_2SO_4 acid stream in to the first leaching tank
$\dot{V}_{l1o}$	Volumetric flow through the leaching section
c	Molar concentration $[mol \cdot m^{-3}]$
k	Reaction rate constant (units depend on the chemical reaction)
M	Molar mass $[g \cdot mol^{-1}]$
pb	Denotes buffer tank
ps	Denotes cementation tank
V	Volume $[m^3]$
$x_{c,\text{CuO}}$	CuO mass fraction of raw material

Part I

THEORY AND METHODOLOGY

Chapter 1

Introduction

The research base of this dissertation is connected with the two problems state-parameter-disturbance estimation and optimal control with respect to the copper leaching process which is a subprocess of the metal refinery Glencore Nikkelverk, Kristiansand. This chapter provides a mathematical model. The structure of this chapter is as follows:

- Section 1.1 - the process description of the copper leaching process (with a brief overview of the Glencore Nikkelverk);
- Section 1.2 - a mechanistic model for the copper leaching process; and
- Section 1.3 - the problem description.

1.1 Process Description

Glencore Nikkelverk is a metal refinery in Kristiansand, Norway. It is the largest nickel refinery in the EU/EEA area. The Nikkelverk produces nickel (Ni), copper (Cu), cobalt (Co), sulfuric acid (H_2SO_4) and other precious metals such as gold (Au), platinum (Pt), selenium (Se) and palladium (Pd). An overview of the plant's production process is given in Figure 1.1. The raw material used in Glencore Nikkelverk is primarily a granulated matte which is produced by smelters in Canada and Botswana. Further grinded matte (using ball mills) is transported to the chlorine leaching plant. The grinded matte mostly contains Ni, Cu, and sulfur (S). In the chlorine leaching plant, Ni is selectively leached by controlling the redox potential in the medium — the leachate contains extracted Ni as nickel chloride ($NiCl_2$). Most of the Cu in the grinded matte goes to the leach residue, for example in the form of copper sulfide (CuS). Filter presses are used to separate the leach residue and the leachate. The leachate is sent to the Ni refining section. The leach residue is roasted in a roasting furnace (fluidized-bed roaster). The roasting furnace produces copper oxide (CuO) and sulfur dioxide (SO_2). SO_2 is converted to H_2SO_4. The Nikkelverk has its own H_2SO_4 production plant as H_2SO_4 is a necessary ingredient to the copper leaching process. The roasting furnace provides the raw material (so called calcine) containing CuO to the copper plant. Calcine is slurrified using H_2SO_4 to leach out Cu as copper sulfate ($CuSO_4$), and then the leachate is electrolyzed using a Cu cathode and Pb anode to produce solid Cu. A detailed discussion in relation to the chemical process associated with the production plant can be found in [6], [7] and [8].

The copper leaching process is considered as the case study of this research. See Figure 1.2 for the process flow sheet. The process consists of four sections:

- The slurrification section, where the slurrification of calcine is done using recycled anolyte flow (the anolyte flow is taken from the electrowinning section) which contains H_2SO_4;

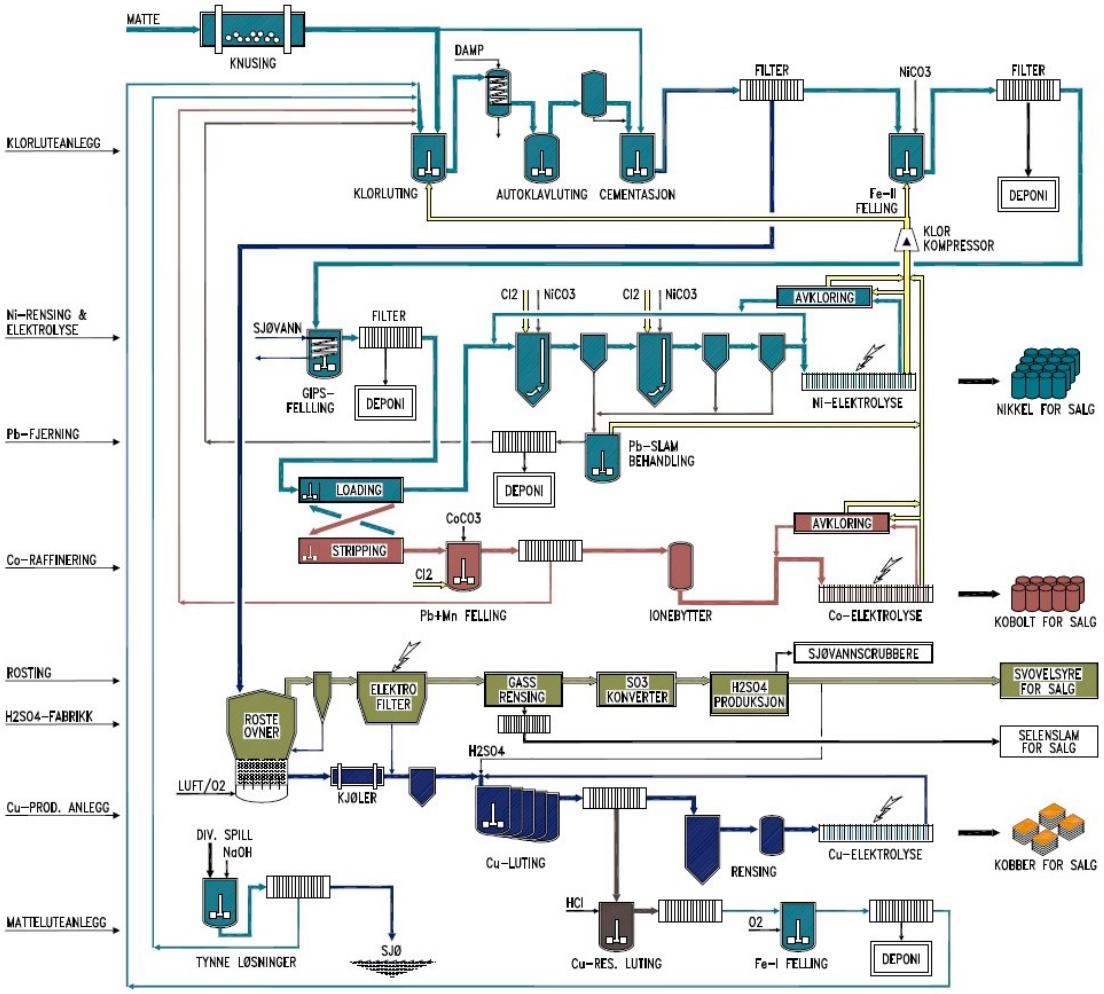


Figure 1.1: Overall production process of Glencore Nikkelverk (taken from `http://www.xstratanickel.no`). Note: labels are in Norwegian language.

- the leaching section, where H_2SO_4 is added to the slurry in order to leach out more Cu into the leachate;
- the purification section, where the slurry is first filtered to extract the solution containing $CuSO_4$, followed by the cementation and fine filtering processes;
- and the electrowinning section, where the solution containing Cu^{2+} is electrolyzed to release solid Cu at the cathode.

Calcine contains Ni (≈ 11.48 *wt%*), Co (≈ 2.17 *wt%*), Fe (≈ 0.99 *wt%*) in addition to Cu (≈ 62.98 *wt%*) — the chemical composition analysis is done at the belt conveyor T5043 (in Figure 1.2) a few times (usually 3 times) per week. Calcine mass flow rate is measured after the second bucket elevator T5002 while it is controlled via the speed of the belt conveyor T5038. There are two bucket elevators and several belt conveyors in between the calcine storage 5037 and the first slurrification tank, causing unavoidable transport delays.

In the slurrification section the leaching reaction (1.1) takes place. There are two slurrification tanks 5045 and 5302 (Figure 1.2) connected in series. It is possible that some other metal oxides than CuO also get leached into the solution, such as Ni, Co and Fe oxides.

$$\boxed{\mathrm{CuO_{(s)} + H_2SO_{4(aq)} \xrightarrow{k} CuSO_{4(aq)} + H_2O_{(l)}}} \tag{1.1}$$

There are four leaching tanks connected in series, H_2SO_4 is added to the first tank. The same chemical reaction (1.1) occurs in all the leaching tanks. Water is added to the fourth leaching tank to replenish liquid spillages throughout the process. The liquid levels of the two slurrification and all four leaching tanks are always constant.

In the purification section, the leach residue and the leachate (containing $CuSO_4$) are separated by sending the output from the leaching section through five filter presses connected in parallel. The leachate has metal ions, such as Ag, Te and Bi ions, contained in it. These metals are removed using the cementation process — there are three scrap columns (5503A-C) where the cementation takes place. For example, Bi^{4+} ions are cemented by sending the solution through the scrap columns which contains pure Cu particles. The cementation reaction for Bi^{4+} ions is (1.2). The cemented metals are filtered by means of downstream fine filters 5508A-D. Also, there are three buffer tanks: 5501, 5506 and 5510.

$$\boxed{\mathrm{Bi^{4+}_{(s)} + 2Cu_{(s)} \longrightarrow Bi_{(s)} + 2Cu^{2+}_{(aq)}}} \tag{1.2}$$

The electrowinning section is the heart of the copper leaching process. There are ca. 430 electrolysis tanks each with the volume ca. $5m^3$ connected in parallel. Cathode and anode half-cell reactions are (1.3) and (1.4), respectively.

$$\boxed{\begin{aligned} &\mathrm{Cu^{2+}_{(aq)} + 2e^- \longrightarrow Cu_{(s)}} && (1.3)\\ &\mathrm{SO^{2-}_{4(aq)} + H_2O_{(l)} \longrightarrow H_2SO_{4(l)} + \frac{1}{2}O_{2(g)} + 2e^-} && (1.4) \end{aligned}}$$

1.2 Mathematical model

The initial attempt of developing a mechanistic model for the copper leaching process is found in [9], where steady state mass balances and dynamic specie balances are used to formulate the model. This model consists of 39 state variables, 4 inputs, 4 outputs and several uncertain parameters. [9] provides estimates for two parameters (by fixing other uncertain parameters with given values) based on available process knowledge and the steady state model. Also, step response analysis in outputs with respect to inputs are given. Most of the suggestions — associated with state-parameter-estimation and model based controller design[1] — given in [9] for planned future work are covered in this dissertation.

The model presented in [9] is slightly extended in this research. The level dynamics in most of the tanks (with the exception of slurrification, leaching, cementation and electrowinning tanks) are included, in contrary to the steady state mass balances in the original model. Steady state mass balances destroys the system structure. A new model is suggested for the rate of reaction between CuO and H_2SO_4[11], which is different from the one used in [9]. The assumptions in [9] are as follows: Liquid density is the same everywhere, perfect cementation of unwanted metals, and perfect filters.

[1] A follow up publication [10] offers an overview about the *Nikkelverk*'s intentions moving towards implementing optimal control strategies.

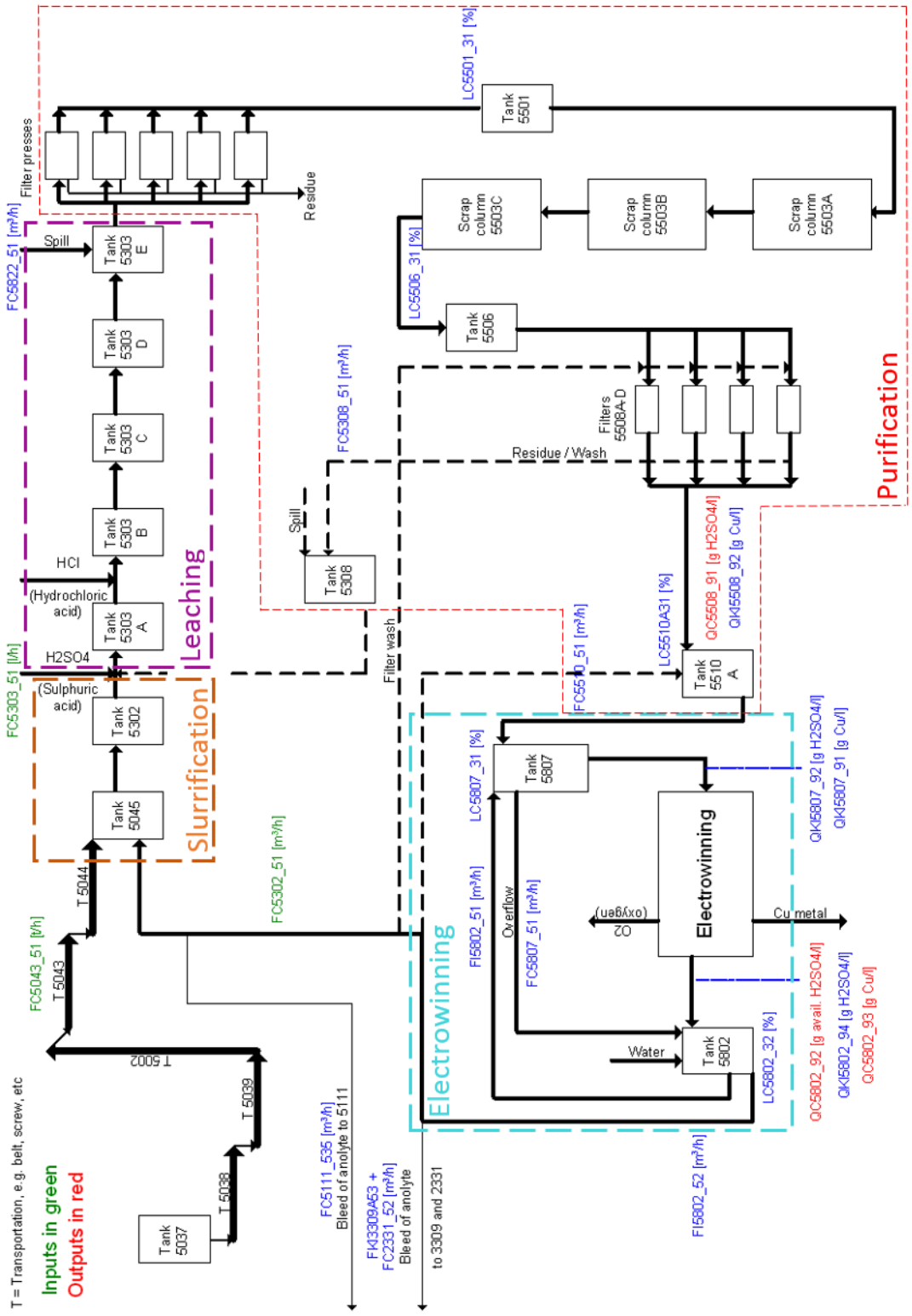


Figure 1.2: A process flow diagram of the copper leaching process (provided by the Nikkelverk).

The model development links to Figure 1.3 — the slurrification, leaching, purification and electrowinning sections are highlighted. In the slurrification section, the heterogeneous reaction (1.1) takes place and a model for the rate of reaction between CuO and H_2SO_4 is necessary. From [11] it is seen that the shrinking core model is appropriate. The reaction kinetics μ is

given by (1.5), where $c_{H_2SO_4}$ denotes molar concentration of H_2SO_4 and k is a temperature dependent parameter which may be modeled using Arrhenius equation [12]. Molar concentrations are replaced by mass concentrations — see (1.6), where $M_{H_2SO_4}$ is the molar mass of H_2SO_4.

$$\boxed{\begin{aligned}\mu &= k \cdot c_{H_2SO_4} \\ c_{H_2SO_4} &= \frac{\rho_{H_2SO_4}}{M_{H_2SO_4}}\end{aligned}} \tag{1.5}$$

$$\mu = \frac{k}{M_{H_2SO_4}} \cdot \rho_{H_2SO_4} \tag{1.6}$$

Species balances for slurrification tanks result in the equations (1.7)-(1.12). In a similar way, dynamic equations from (1.13) to (1.27) for the leaching tanks are obtained. $V_s^{(i)}$ and $V_l^{(j)}$ are constants. $\tau_{\dot{m}_c}$ in (1.7) counts raw material (calcine) transportation delays — according to process experience $\tau_{\dot{m}_c} \approx 20min$. $\tau_{\dot{V}_a}$ is the delay with respect to $\dot{V}_a$. (1.28) gives the reaction kinetics $\mu_i^{(j)}$ of the j^{th} tank of the section i ($i = s$ is for the slurrification section and so on). Additional equations (1.29)-(1.31) are due to mass balances in the slurrification and leaching sections.

$$\boxed{\begin{aligned}\frac{d}{dt}\rho_{s,CuO}^{(1)} &= \frac{\dot{m}_c(t-\tau_{\dot{m}_c}) \cdot x_{c,CuO} - \rho_{s,CuO}^{(1)} \cdot \dot{V}_{s2l}}{V_s^{(1)}} - M_{CuO} \cdot \mu_s^{(1)} \qquad (1.7)\\ \frac{d}{dt}\rho_{s,CuSO_4}^{(1)} &= \frac{\rho_{em,CuSO_4} \cdot \dot{V}_{e2s} - \rho_{s,CuSO_4}^{(1)} \cdot \dot{V}_{s2l}}{V_s^{(1)}} + M_{CuSO_4} \cdot \mu_s^{(1)} \qquad (1.8)\\ \frac{d}{dt}\rho_{s,H_2SO_4}^{(1)} &= \frac{\rho_{em,H_2SO_4} \cdot \dot{V}_{e2s} - \rho_{s,H_2SO_4}^{(1)} \cdot \dot{V}_{s2l}}{V_s^{(1)}} - M_{H_2SO_4} \cdot \mu_s^{(1)} \qquad (1.9)\end{aligned}}$$

$$\boxed{\begin{aligned}\frac{d}{dt}\rho_{s,CuO}^{(2)} &= \frac{\left(\rho_{s,CuO}^{(1)} - \rho_{s,CuO}^{(2)}\right) \cdot \dot{V}_{s2l}}{V_s^{(2)}} - M_{CuO} \cdot \mu_s^{(2)} \qquad (1.10)\\ \frac{d}{dt}\rho_{s,CuSO_4}^{(2)} &= \frac{\left(\rho_{s,CuSO_4}^{(1)} - \rho_{s,CuSO_4}^{(2)}\right) \cdot \dot{V}_{s2l}}{V_s^{(2)}} + M_{CuSO_4} \cdot \mu_s^{(2)} \qquad (1.11)\\ \frac{d}{dt}\rho_{s,H_2SO_4}^{(2)} &= \frac{\left(\rho_{s,H_2SO_4}^{(1)} - \rho_{s,H_2SO_4}^{(2)}\right) \cdot \dot{V}_{s2l}}{V_s^{(2)}} - M_{H_2SO_4} \cdot \mu_s^{(2)} \qquad (1.12)\end{aligned}}$$

$$\boxed{\begin{aligned}\frac{d}{dt}\rho_{l,CuO}^{(1)} &= \frac{\rho_{s,CuO}^{(2)} \cdot \dot{V}_{s2l} - \rho_{l,CuO}^{(1)} \cdot \dot{V}_{l1o}}{V_l^{(1)}} - M_{CuO} \cdot \mu_l^{(1)} \qquad (1.13)\\ \frac{d}{dt}\rho_{l,CuSO_4}^{(1)} &= \frac{\rho_{s,CuSO_4}^{(2)} \cdot \dot{V}_{s2l} - \rho_{l,CuSO_4}^{(1)} \cdot \dot{V}_{l1o}}{V_l^{(1)}} + M_{CuSO_4} \cdot \mu_l^{(1)} \qquad (1.14)\\ \frac{d}{dt}\rho_{l,H_2SO_4}^{(1)} &= \frac{\rho_{a,H_2SO_4} \cdot \dot{V}_a(t-\tau_{\dot{V}_a}) + \rho_{s,H_2SO_4}^{(2)} \cdot \dot{V}_{s2l} - \rho_{l,H_2SO_4}^{(1)} \cdot \dot{V}_{l1o}}{V_l^{(1)}} \\ &\quad - M_{H_2SO_4} \cdot \mu_l^{(1)} \qquad (1.15)\end{aligned}}$$

$$\frac{d\rho_{l,\mathrm{CuO}}{}^{(2)}}{dt} = \frac{\left(\rho_{l,\mathrm{CuO}}^{(1)} - \rho_{l,\mathrm{CuO}}^{(2)}\right)\cdot \dot{V}_{l1o}}{V_l^{(2)}} - M_{\mathrm{CuO}}\cdot \mu_l^{(2)} \tag{1.16}$$

$$\frac{d}{dt}\rho_{l,\mathrm{CuSO_4}}^{(2)} = \frac{(\rho_{l,\mathrm{CuSO_4}}^{(1)} - \rho_{l,\mathrm{CuSO_4}}^{(2)})\cdot \dot{V}_{l1o}}{V_l^{(2)}} + M_{\mathrm{CuSO_4}}\cdot \mu_l^{(2)} \tag{1.17}$$

$$\frac{d}{dt}\rho_{l,\mathrm{H_2SO_4}}^{(2)} = \frac{(\rho_{l,\mathrm{H_2SO_4}}^{(1)} - \rho_{l,\mathrm{H_2SO_4}}^{(2)})\cdot \dot{V}_{l1o}}{V_l^{(2)}} - M_{\mathrm{H_2SO_4}}\cdot \mu_l^{(2)} \tag{1.18}$$

$$\frac{d}{dt}\rho_{l,\mathrm{CuO}}^{(3)} = \frac{\left(\rho_{l,\mathrm{CuO}}^{(2)} - \rho_{l,\mathrm{CuO}}^{(3)}\right)\cdot \dot{V}_{l1o}}{V_l^{(3)}} - M_{\mathrm{CuO}}\cdot \mu_l^{(3)} \tag{1.19}$$

$$\frac{d}{dt}\rho_{l,\mathrm{CuSO_4}}^{(3)} = \frac{\left(\rho_{l,\mathrm{CuSO_4}}^{(2)} - \rho_{l,\mathrm{CuSO_4}}^{(3)}\right)\cdot \dot{V}_{l1o}}{V_l^{(3)}} + M_{\mathrm{CuSO_4}}\cdot \mu_l^{(3)} \tag{1.20}$$

$$\frac{d}{dt}\rho_{l,\mathrm{H_2SO_4}}^{(3)} = \frac{\left(\rho_{l,\mathrm{H_2SO_4}}^{(2)} - \rho_{l,\mathrm{H_2SO_4}}^{(3)}\right)\cdot \dot{V}_{l1o}}{V_l^{(3)}} - M_{\mathrm{H_2SO_4}}\cdot \mu_l^{(3)} \tag{1.21}$$

$$\frac{d}{dt}\rho_{l,\mathrm{CuO}}^{(4)} = \frac{\left(\rho_{l,\mathrm{CuO}}^{(3)} - \rho_{l,\mathrm{CuO}}^{(4)}\right)\cdot \dot{V}_{l1o}}{V_l^{(4)}} - M_{\mathrm{CuO}}\cdot \mu_l^{(4)} \tag{1.22}$$

$$\frac{d}{dt}\rho_{l,\mathrm{CuSO_4}}^{(4)} = \frac{\left(\rho_{l,\mathrm{CuSO_4}}^{(3)} - \rho_{l,\mathrm{CuSO_4}}^{(4)}\right)\cdot \dot{V}_{l1o}}{V_l^{(4)}} + M_{\mathrm{CuSO_4}}\cdot \mu_l^{(4)} \tag{1.23}$$

$$\frac{d}{dt}\rho_{l,\mathrm{H_2SO_4}}^{(4)} = \frac{\left(\rho_{l,\mathrm{H_2SO_4}}^{(3)} - \rho_{l,\mathrm{H_2SO_4}}^{(4)}\right)\cdot \dot{V}_{l1o}}{V_l^{(4)}} - M_{\mathrm{H_2SO_4}}\cdot \mu_l^{(4)} \tag{1.24}$$

$$\frac{d}{dt}\rho_{l,\mathrm{CuO}}^{(5)} = \frac{\left(\rho_{l,\mathrm{CuO}}^{(4)}\cdot \dot{V}_{l1o} - \rho_{l,\mathrm{CuO}}^{(5)}\cdot \dot{V}_{l2p}\right)}{V_l^{(5)}} - M_{\mathrm{CuO}}\cdot \mu_l^{(5)} \tag{1.25}$$

$$\frac{d}{dt}\rho_{l,\mathrm{CuSO_4}}^{(5)} = \frac{\left(\rho_{l,\mathrm{CuSO_4}}^{(4)}\cdot \dot{V}_{l1o} - \rho_{l,\mathrm{CuSO_4}}^{(5)}\dot{V}_{l2p}\right)}{V_l^{(5)}} + M_{\mathrm{CuSO_4}}\cdot \mu_l^{(5)} \tag{1.26}$$

$$\frac{d}{dt}\rho_{l,\mathrm{H_2SO_4}}^{(5)} = \frac{\left(\rho_{l,\mathrm{H_2SO_4}}^{(4)}\cdot \dot{V}_{l1o} - \rho_{l,\mathrm{H_2SO_4}}^{(5)}\cdot \dot{V}_{l2p}\right)}{V_l^{(5)}} - M_{\mathrm{H_2SO_4}}\cdot \mu_l^{(5)} \tag{1.27}$$

$$\mu_i^{(j)} = \frac{k}{M_{\mathrm{H_2SO_4}}}\cdot \rho_{i,\mathrm{H_2SO_4}}^{(j)} \tag{1.28}$$

$$\dot{V}_{e2s} = \dot{V}_{s2l} \tag{1.29}$$

$$\dot{V}_{s2l} + \dot{V}_a = \dot{V}_{l1o} \tag{1.30}$$

$$\dot{V}_{l1o} + \dot{V}_{w2l} = \dot{V}_{l2p} \tag{1.31}$$

The purification section consists of three buffer tanks, three cementing tanks, filter presses and fine filters. The filter presses and fine filters are assumed to be perfect — i.e. no liquid leakages and complete liquid-solid separation. From (1.32) to (1.34) gives corresponding dynamic equations. $V_{pb}^{(1)}$ is controlled.

$$\frac{d}{dt}V_{pb}^{(1)} = \dot{V}_{l2p} - \dot{V}_{pb^{(1)}2ps^{(1)}} \tag{1.32}$$

$$\frac{d}{dt}\rho_{pb,\mathrm{CuSO_4}}^{(1)} = \frac{\left(\rho_{l,\mathrm{CuSO_4}}^{(5)} - \rho_{pb,\mathrm{CuSO_4}}^{(1)}\right)\cdot \dot{V}_{l2p}}{V_{pb}^{(1)}} \tag{1.33}$$

$$\frac{d}{dt}\rho_{pb,\mathrm{H_2SO_4}}^{(1)} = \frac{\left(\rho_{l,\mathrm{H_2SO_4}}^{(5)} - \rho_{pb,\mathrm{H_2SO_4}}^{(1)}\right)\cdot \dot{V}_{l2p}}{V_{pb}^{(1)}} \tag{1.34}$$

The cementation tanks are merely volumes filled with solid Cu particles and the liquid medium flows through the voids between the Cu particles, see (1.35)-(1.43). ε_1, ε_2, and ε_3 are void fractions, which are uncertain parameters. Since there is a constant flow rate across the cementation tanks, we have (1.44)-(1.46). $\dot{V}_{pb^{(1)}2ps^{(1)}}$ in (1.44) is the volumetric flow rate between the output of the first buffer to the input of the fist cementation tank. $pb^{(i)}$ denotes the buffer tank i and $ps^{(i)}$ denotes the cementation tank i. Other flow rates in the equations (1.45)-(1.46) are defined in accordingly. The buffer tank level control functions $u_{V_{pb}^{(1)}}$, $u_{V_{pb}^{(2)}}$ and $u_{V_{pb}^{(3)}}$ given in (1.53)-(1.55). $V_{pb,SP}^{(1)}$ is the set point of $V_{pb}^{(1)}$.

$$\frac{d}{dt}\rho_{ps,\mathrm{CuSO_4}}^{(1)} = \frac{\left(\rho_{pb,\mathrm{CuSO_4}}^{(1)} - \rho_{ps,\mathrm{CuSO_4}}^{(1)}\right)\cdot \dot{V}_{pb^{(1)}2ps^{(1)}}}{V_{ps}^{(1)}\cdot\varepsilon_1} \tag{1.35}$$

$$\frac{d}{dt}\rho_{ps,\mathrm{H_2SO_4}}^{(1)} = \frac{\left(\rho_{pb,\mathrm{H_2SO_4}}^{(1)} - \rho_{ps,\mathrm{H_2SO_4}}^{(1)}\right)\cdot \dot{V}_{pb^{(1)}2ps^{(1)}}}{V_{ps}^{(1)}\cdot\varepsilon_1} \tag{1.36}$$

$$\frac{d}{dt}\varepsilon_1 = 0 \tag{1.37}$$

$$\frac{d}{dt}\rho_{ps,\mathrm{CuSO_4}}^{(2)} = \frac{\left(\rho_{ps,\mathrm{CuSO_4}}^{(1)} - \rho_{ps,\mathrm{CuSO_4}}^{(2)}\right)\cdot \dot{V}_{ps^{(1)}2ps^{(2)}}}{V_{ps}^{(2)}\cdot\varepsilon_2} \tag{1.38}$$

$$\frac{d}{dt}\rho_{ps,\mathrm{H_2SO_4}}^{(2)} = \frac{\left(\rho_{ps,\mathrm{H_2SO_4}}^{(1)} - \rho_{ps,\mathrm{H_2SO_4}}^{(2)}\right)\cdot \dot{V}_{ps^{(1)}2ps^{(2)}}}{V_{ps}^{(2)}\cdot\varepsilon_2} \tag{1.39}$$

$$\frac{d}{dt}\varepsilon_2 = 0 \tag{1.40}$$

$$\frac{d}{dt}\rho_{ps,\mathrm{CuSO_4}}^{(3)} = \frac{\left(\rho_{ps,\mathrm{CuSO_4}}^{(2)} - \rho_{ps,\mathrm{CuSO_4}}^{(3)}\right)\cdot \dot{V}_{ps^{(2)}2ps^{(3)}}}{V_{ps}^{(3)}\cdot\varepsilon_3} \tag{1.41}$$

$$\frac{d}{dt}\rho_{ps,\mathrm{H_2SO_4}}^{(3)} = \frac{\left(\rho_{ps,\mathrm{H_2SO_4}}^{(2)} - \rho_{ps,\mathrm{H_2SO_4}}^{(3)}\right)\cdot \dot{V}_{ps^{(2)}2ps^{(3)}}}{V_{ps}^{(3)}\cdot\varepsilon_3} \tag{1.42}$$

$$\frac{d}{dt}\varepsilon_3 = 0 \tag{1.43}$$

$$\dot{V}_{pb^{(1)}2ps^{(1)}} = \dot{V}_{ps^{(1)}2ps^{(2)}} \tag{1.44}$$

$$\dot{V}_{ps^{(1)}2ps^{(2)}} = \dot{V}_{ps^{(2)}2ps^{(3)}} \tag{1.45}$$

$$\dot{V}_{ps^{(2)}2ps^{(3)}} = \dot{V}_{ps^{(3)}2pb^{(2)}} \tag{1.46}$$

$$\frac{d}{dt}V_{pb}^{(2)} = \dot{V}_{ps^{(3)}2pb^{(2)}} - \dot{V}_{pb^{(2)}2pb^{(3)}} \tag{1.47}$$

$$\frac{d}{dt}\rho_{pb,\mathrm{CuSO_4}}^{(2)} = \frac{\left(\rho_{ps,\mathrm{CuSO_4}}^{(3)} - \rho_{pb,\mathrm{CuSO_4}}^{(2)}\right)\cdot \dot{V}_{ps^{(3)}2pb^{(2)}}}{V_{pb}^{(2)}} \tag{1.48}$$

$$\frac{d}{dt}\rho_{pb,\mathrm{H_2SO_4}}^{(2)} = \frac{\left(\rho_{ps,\mathrm{H_2SO_4}}^{(3)} - \rho_{pb,\mathrm{H_2SO_4}}^{(2)}\right)\cdot \dot{V}_{ps^{(3)}2pb^{(2)}}}{V_{pb}^{(2)}} \tag{1.49}$$

$$\frac{d}{dt}V_{pb}^{(3)} = \dot{V}_{pb^{(2)}2pb^{(3)}} - \dot{V}_{p2e} \tag{1.50}$$

$$\frac{d}{dt}\rho_{pb,\mathrm{CuSO_4}}^{(3)} = \frac{\left(\rho_{pb,\mathrm{CuSO_4}}^{(2)} - \rho_{pb,\mathrm{CuSO_4}}^{(3)}\right)\cdot \dot{V}_{pb^{(2)}2pb^{(3)}}}{V_{pb}^{(3)}} \tag{1.51}$$

$$\frac{d}{dt}\rho_{pb,\mathrm{H_2SO_4}}^{(3)} = \frac{\left(\rho_{pb,\mathrm{H_2SO_4}}^{(2)} - \rho_{pb,\mathrm{H_2SO_4}}^{(3)}\right)\cdot \dot{V}_{pb^{(2)}2pb^{(3)}}}{V_{pb}^{(3)}} \tag{1.52}$$

$$\dot{V}_{pb^{(1)}2ps^{(1)}} = u_{V_{pb}^{(1)}}\left(V_{pb,SP}^{(1)}, V_{pb}^{(1)}\right) \tag{1.53}$$

$$\dot{V}_{pb^{(2)}2pb^{(3)}} = u_{V_{pb}^{(2)}}\left(V_{pb,SP}^{(2)}, V_{pb}^{(2)}\right) \tag{1.54}$$

$$\dot{V}_{p2e} = u_{V_{pb}^{(3)}}\left(V_{pb,SP}^{(3)}, V_{pb}^{(3)}\right) \tag{1.55}$$

(1.47)-(1.55) are the dynamic equations for the second and the third buffer tanks and their levels are controlled. Equations (1.56)-(1.65) represent the electrowinning section. (1.37), (1.40), (1.43), (1.61) and (1.62) are due to parameter augmentations.

$$\frac{d}{dt}V_{ed} = \dot{V}_{p2e} + \dot{V}_{em2d} - \dot{V}_{ed2m} - \dot{V}_{ed2w} \tag{1.56}$$

$$\frac{d}{dt}\rho_{ed,\mathrm{CuSO_4}} = \frac{\dot{V}_{p2e}\cdot\left(\rho_{pb,\mathrm{CuSO_4}}^{(3)} - \rho_{ed,\mathrm{CuSO_4}}\right)}{V_{ed}} + \frac{\dot{V}_{em2d}\cdot\left(\rho_{em,\mathrm{CuSO_4}} - \rho_{ed,\mathrm{CuSO_4}}\right)}{V_{ed}} \tag{1.57}$$

$$\frac{d}{dt}\rho_{ed,\mathrm{H_2SO_4}} = \frac{\dot{V}_{p2e}\cdot\left(\rho_{pb,\mathrm{H_2SO_4}}^{(3)} - \rho_{ed,\mathrm{H_2SO_4}}\right)}{V_{ed}} + \frac{\dot{V}_{em2d}\cdot\left(\rho_{em,\mathrm{H_2SO_4}} - \rho_{ed,\mathrm{H_2SO_4}}\right)}{V_{ed}} \tag{1.58}$$

$$\boxed{\begin{aligned}
\frac{d}{dt}\rho_{ew,\mathrm{CuSO_4}} &= \frac{\dot{V}_{ed2w}\cdot(\rho_{ed,\mathrm{CuSO_4}}-\rho_{ew,\mathrm{CuSO_4}})}{V_{ew}}+ \\
&\quad \frac{\dot{V}_{vap}\cdot\rho_{ew,\mathrm{CuSO_4}}}{V_{ew}}- \\
&\quad \frac{\frac{M_{\mathrm{CuSO_4}}}{z_{Cu}\cdot C}\cdot\bar{\eta}\cdot\bar{I}}{V_{ew}} \qquad (1.59)\\
\frac{d\rho_{ew,\mathrm{H_2SO_4}}}{dt} &= \frac{\dot{V}_{ed2w}\cdot(\rho_{ed,\mathrm{H_2SO_4}}-\rho_{ew,\mathrm{H_2SO_4}})}{V_{ew}}+ \\
&\quad \frac{\dot{V}_{vap}\cdot\rho_{ew,\mathrm{H_2SO_4}}}{V_{ew}}+ \\
&\quad \frac{\frac{M_{\mathrm{CuSO_4}}}{z_{Cu}\cdot C}\cdot\bar{\eta}\cdot\bar{I}}{V_{ew}} \qquad (1.60)\\
\frac{d\bar{\eta}}{dt} &= 0 \qquad (1.61)\\
\frac{dV_{ew}}{dt} &= 0 \qquad (1.62)
\end{aligned}}$$

(1.66)-(1.68) are measurement equations. Three control inputs are defined in (1.69)-(1.71). All the inputs act on the outputs with some time delays, in particular u_1 and u_3 have the most influential delays.

$$\boxed{\begin{aligned}
\frac{d}{dt}V_{em} &= \dot{V}_{ed2m}+\dot{V}_{ew2m}+\dot{V}_{w2em}- \\
&\quad \dot{V}_{e2s}-\dot{V}_{em2d}-\dot{V}_{em2bl} \qquad (1.63)\\
\frac{d}{dt}\rho_{em,\mathrm{CuSO_4}} &= \frac{\dot{V}_{ed2m}\cdot\left(\rho_{ed,\mathrm{CuSO_4}}-\rho_{em,\mathrm{CuSO_4}}\right)}{V_{em}}+ \\
&\quad \frac{\dot{V}_{ew2m}\cdot\left(\rho_{ew,\mathrm{CuSO_4}}-\rho_{em,\mathrm{CuSO_4}}\right)}{V_{em}}- \\
&\quad \frac{\dot{V}_{w2em}\cdot\rho_{em,\mathrm{CuSO_4}}}{V_{em}} \qquad (1.64)\\
\frac{d}{dt}\rho_{em,\mathrm{H_2SO_4}} &= \frac{\dot{V}_{ed2m}\cdot\left(\rho_{ed,\mathrm{H_2SO_4}}-\rho_{em,\mathrm{H_2SO_4}}\right)}{V_{em}}+ \\
&\quad \frac{\dot{V}_{ew2m}\cdot\left(\rho_{ew,\mathrm{H_2SO_4}}-\rho_{em,\mathrm{H_2SO_4}}\right)}{V_{em}}- \\
&\quad \frac{\dot{V}_{w2em}\cdot\rho_{em,\mathrm{H_2SO_4}}}{V_{em}} \qquad (1.65)
\end{aligned}}$$

$$\boxed{\begin{aligned}
y_1 &= \rho^{(3)}_{pb,\mathrm{H_2SO_4}} \qquad (1.66)\\
y_2 &= \frac{M_{\mathrm{Cu}}}{M_{\mathrm{CuSO_4}}}\rho_{ew,\mathrm{CuSO_4}} \qquad (1.67)\\
y_3 &= \rho_{ew,\mathrm{H_2SO_4}}+\frac{M_{\mathrm{H_2SO_4}}}{M_{\mathrm{CuSO_4}}}\rho_{ew,\mathrm{CuSO_4}} \qquad (1.68)
\end{aligned}}$$

$$\boxed{\begin{aligned}
u_1 &= \dot{m}_c \qquad (1.69)\\
u_2 &= \dot{V}_{e2s} \qquad (1.70)\\
u_3 &= \dot{V}_a \qquad (1.71)
\end{aligned}}$$

$$\boxed{\dot{x} = f(x, u, w_1, w_2, p)} \tag{1.72}$$
$$\boxed{y = h(x)} \tag{1.73}$$

The model can be written in the state space form (1.72)-(1.73), where x, u, w_1, w_2, p and y are state, input, known disturbance, unknown disturbance, parameter and output vectors — see Table (1.1) for a detailed description. f and h are known functions.

1.3 Problem Description

Most of Glencore Nikkelverk's control problems are satisfactorily solved via PID controllers and the control structure is quite complicated — there are cascaded and split-range controllers [10]. Conventional control approaches gives poor results for the control problem connected to the copper leaching process. The degree of fluctuation of the measured quality variables — in equations (1.66)-(1.68) — are not within the expected bounds. The process model features time delays, multivariable behavior and nonlinearities in addition to being large-scale and very sluggish, causing the control problem to be quite challenging. Some other challenges are the availability of few quality measurements, and there are many unknown disturbances and parameters. Thus, in the interest of the quality control within the copper leaching process, it is proposed to consider advanced control strategies.

The control problem poses a state-parameter-disturbance estimation problem. It is desirable that outputs carry complete information about the system state, parameters, and unknown disturbances; in other words, reconstructing these variables via available outputs is expected. The degree of interactions among the variables to be estimated and the outputs determines the success of the estimation process. An analysis should be carried out to identify unobservable state variables, parameters and disturbances as well as to suggest additional measurements to be included to achieve complete observability.

A suitable state estimator needs to be proposed. Mainly, the Extended Kalman Filter and Moving Horizon Estimator are to be considered. Some other choices are the Unscented Kalman Filter and Particle Filter. In particular, the stability properties of the estimators should be compared and contrasted.

Often, large-scale control and estimation problems may be decomposed into small-scale subproblems. This can be done with the aid of structural controllability and observability analysis. Also, the model reduction is an important aspect in controller design and synthesis. A possible model reduction technique is proposed through a graph-theoretic approach.

Advanced control strategies are implemented for those subsystems which are observable with respect to available measurements. Some suggestions should be made regarding the decentralization of the control structure.[2] The main goal of this work is to stabilize both Cu and H_2SO_4 concentrations within the copper electrowinning process, hence the copper electrowinning process is given special attention.

1.4 Previous Work and New Contributions

[9] is the only previous modeling attempt on the copper leaching process. The model captures most of the system's structure. The tank level dynamics are neglected in the initial model. This

[2] "The sheer size (i.e., dimensionality) and complexity of these large-scale dynamical systems often necessitates a hierarchical decentralized architecture for analyzing and controlling these systems."[13]

Vector	Description	Vector Elements
x	State	$\rho^{(1)}_{s,\mathrm{CuO}}$, $\rho^{(1)}_{s,\mathrm{CuSO_4}}$, $\rho^{(1)}_{s,\mathrm{H_2SO_4}}$, $\rho^{(2)}_{s,\mathrm{CuO}}$, $\rho^{(2)}_{s,\mathrm{CuSO_4}}$, $\rho^{(2)}_{s,\mathrm{H_2SO_4}}$, $\rho^{(1)}_{l,\mathrm{CuO}}$, $\rho^{(1)}_{l,\mathrm{CuSO_4}}$, $\rho^{(1)}_{l,\mathrm{H_2SO_4}}$, $\rho^{(2)}_{l,\mathrm{CuO}}$, $\rho^{(2)}_{l,\mathrm{CuSO_4}}$, $\rho^{(2)}_{l,\mathrm{H_2SO_4}}$, $\rho^{(3)}_{l,\mathrm{CuO}}$, $\rho^{(3)}_{l,\mathrm{CuSO_4}}$, $\rho^{(3)}_{l,\mathrm{H_2SO_4}}$, $\rho^{(4)}_{l,\mathrm{CuO}}$, $\rho^{(4)}_{l,\mathrm{CuSO_4}}$, $\rho^{(4)}_{l,\mathrm{H_2SO_4}}$, $\rho^{(5)}_{l,\mathrm{CuO}}$, $\rho^{(5)}_{l,\mathrm{CuSO_4}}$, $\rho^{(5)}_{l,\mathrm{H_2SO_4}}$, $V^{(1)}_{pb}$, $\rho^{(1)}_{pb,\mathrm{CuSO_4}}$, $\rho^{(1)}_{pb,\mathrm{H_2SO_4}}$, $\rho^{(1)}_{ps,\mathrm{CuSO_4}}$, $\rho^{(1)}_{ps,\mathrm{H_2SO_4}}$, $\rho^{(2)}_{ps,\mathrm{CuSO_4}}$, $\rho^{(2)}_{ps,\mathrm{H_2SO_4}}$, $\rho^{(3)}_{ps,\mathrm{CuSO_4}}$, $\rho^{(3)}_{ps,\mathrm{H_2SO_4}}$, $V^{(2)}_{pb}$, $\rho^{(2)}_{pb,\mathrm{CuSO_4}}$, $\rho^{(2)}_{pb,\mathrm{H_2SO_4}}$, $V^{(3)}_{pb}$, $\rho^{(3)}_{pb,\mathrm{CuSO_4}}$, $\rho^{(3)}_{pb,\mathrm{H_2SO_4}}$, V_{ed}, $\rho_{ed,\mathrm{CuSO_4}}$, $\rho_{ed,\mathrm{H_2SO_4}}$, $\rho_{ew,\mathrm{CuSO_4}}$, $\rho_{ew,\mathrm{H_2SO_4}}$, V_{em}, $\rho_{em,\mathrm{CuSO_4}}$, $\rho_{em,\mathrm{H_2SO_4}}$
u	Input	$\dot{m}_c$, $\dot{V}_{e2s}$, $\dot{V}_a$, $\dot{V}_{pb^{(1)}2ps^{(1)}}$, $\dot{V}_{pb^{(2)}2pb^{(3)}}$
w_1	Known disturbance	$\dot{V}_{w2l}$, $\dot{V}_{em2d}$, $\dot{V}_{ed2m}$, $\dot{V}_{em2bl}$, $\dot{V}_{w2em}$
w_2	Unknown disturbance	$x_{c,\mathrm{CuO}}$, $\rho_{a,\mathrm{H_2SO_4}}$, $\dot{V}_{ed2w}$, $\dot{V}_{ew2m}$
p	Parameter	ε_1, ε_2, ε_3, $\tau_{\dot{m}_c}$, $\tau_{\dot{V}_a}$, $\tau_{\dot{V}_{e2s}}$ V_{ew}, $\bar{\eta}$, k, $V^{(1)}_s$, $V^{(2)}_s$, $V^{(1)}_l$, $V^{(2)}_l$, $V^{(3)}_l$, $V^{(4)}_l$, $V^{(5)}_l$, $V^{(1)}_{ps}$, $V^{(2)}_{ps}$, $V^{(3)}_{ps}$
y	Output	$\rho^{(3)}_{pb,\mathrm{H_2SO_4}}$, $\frac{M_{\mathrm{Cu}}}{M_{\mathrm{CuSO_4}}}\rho_{ew,\mathrm{CuSO_4}}$, $\rho_{ew,\mathrm{H_2SO_4}} + \frac{M_{\mathrm{H_2SO_4}}}{M_{\mathrm{CuSO_4}}}\rho_{ew,\mathrm{CuSO_4}}$

Table 1.1: State space model: $\dot{x} = f(x,u,w_1,w_2,p)$ and $y = h(x)$.

is a realistic assumption as the chemical compositions have very slow dynamics. However, this assumption destroys the actual system structure. Therefore, in this research an extended model including level dynamics is used.

[14] introduced the two concepts structure and structural controllability for linear time invariant systems; the paper also revealed the graph-theoretic analogy to structural controllability. Later, the idea was extended to the concept of structural observability. For nonlinear systems, a linearized model can be used to check local structural properties. In this dissertation it is demonstrated that the structural analysis provides further useful information in addition to controllability and observability. Part of the research shows how to best use structural analysis in relation to parameter, disturbance, and time delay augmentations in estimation and control applications.

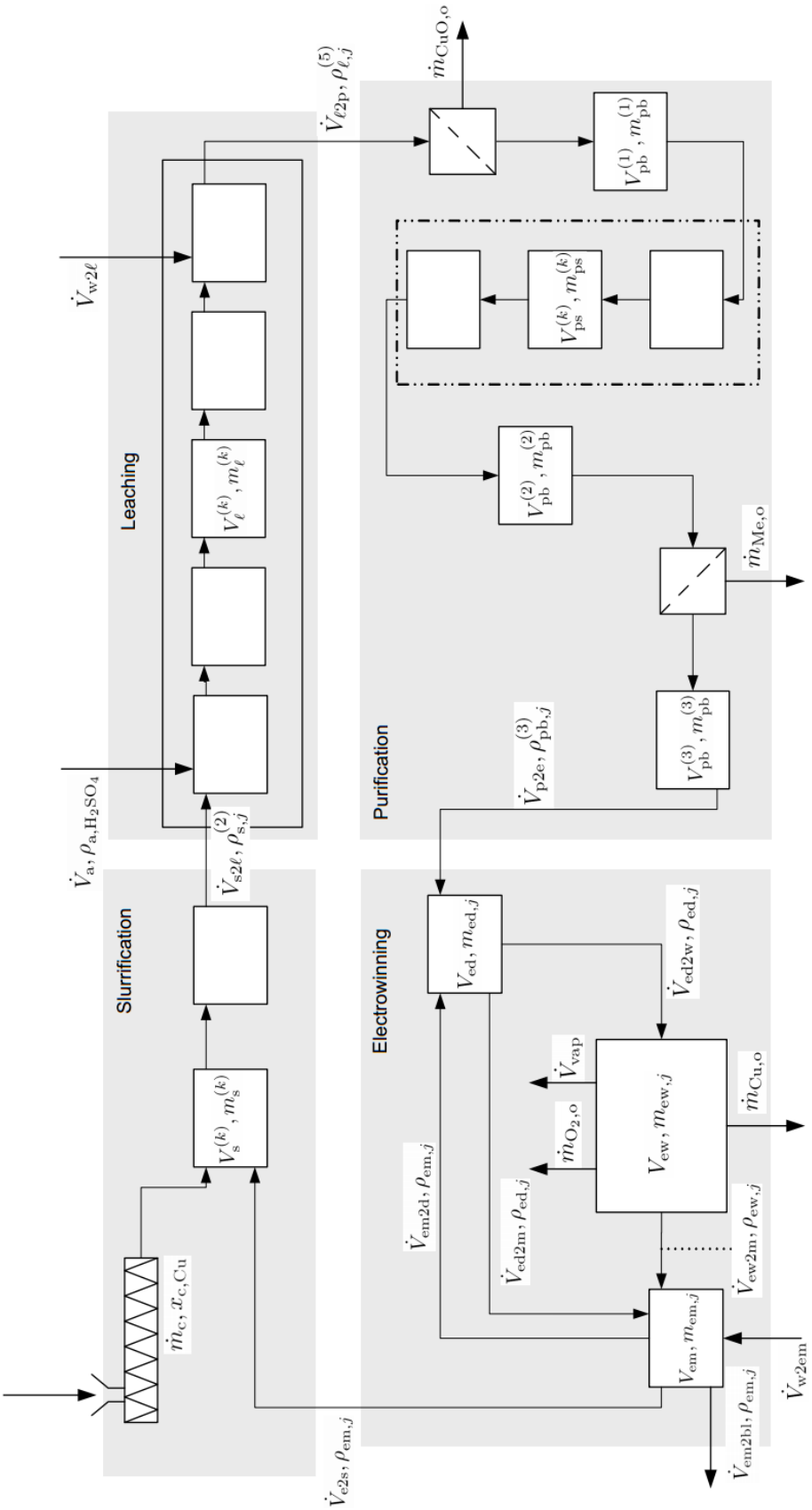


Figure 1.3: The process flow diagram of the copper leaching process which is used for the modeling process (taken from [9]).

Chapter 2

Large-Scale Complex Dynamic Systems

This chapter provides necessary theoretical information to be used in this dissertation and Part II of this dissertation. The following sections elucidate the terminologies observability/controllability, structural observability/controllability, stochastic observability/controllability, state estimation and optimal control.

The state space model for a given dynamic system (2.1)-(2.2) is considered, where $x \equiv x(t)$, $u \equiv u(t)$, $w^{(1)} \equiv w^{(1)}(t)$ and $y \equiv y(t)$ denote state, input, known disturbance and output vectors respectively. $f(.,.,.)$ and $h(.)$ are known vector-valued functions. Generally, the system is of a large-scale if the state space has high dimensionality [15]. The model complexity arises due to the presence of highly interconnected subsystems, transport delays, etc.

$$\dot{x} = f\left(x, u, w^{(1)}\right) \tag{2.1}$$

$$f\left(x, u, w^{(1)}\right) = \left[f_1\left(x, u, w^{(1)}\right) \quad f_2\left(x, u, w^{(1)}\right) \quad \cdots \quad f_{n_x}\left(x, u, w^{(1)}\right)\right]^T$$

$$y = h(x) \tag{2.2}$$

$$x = \begin{bmatrix} x_1 & x_2 & \cdots & x_{n_x} \end{bmatrix}^T$$

$$u = \begin{bmatrix} u_1 & u_2 & \cdots & u_{n_u} \end{bmatrix}^T$$

$$w^{(1)} = \begin{bmatrix} w_1^{(1)} & w_2^{(1)} & \cdots & w_{n_{w^{(1)}}}^{(1)} \end{bmatrix}^T$$

$$y = \begin{bmatrix} y_1 & y_2 & \cdots & y_{n_y} \end{bmatrix}^T$$

$$n_x = dim(x)$$

$$n_u = dim(u)$$

$$n_{w^{(1)}} = dim(w^{(1)})$$

$$n_y = dim(y)$$

System models are in general nonlinear implicit Differential Algebraic Equations (DAEs).[1] A system of implicit DAEs may be reformulated into a systems of explicit Ordinary Differential Equations (ODEs) via some algebraic manipulations, either manually or algorithmically [16] — index reduction should be done if necessary [17]. (2.1)-(2.2) represent a set of explicit ODEs. A discrete time version of (2.1)-(2.2) is given by (2.3)-(2.4), where u_k and $w_k^{(1)}$ are piecewise constant functions and Δt is the sample time. An appropriate integrator should be used here, for instance [18].

[1] More precisely, Differential Algebraic Discrete Equations.

$$x_{k+1} = f_k\left(x_k, u_k, w_k^{(1)}\right) \tag{2.3}$$
$$y_k = h_k\left(x_k\right) \tag{2.4}$$

$k = 0, 1, \ldots$
x_0 is given.
$x_k = x(k\Delta t)$
$u_k = u(k\Delta t)$
$w_k^{(1)} = w^{(1)}(k\Delta t)$
$y_k = y(k\Delta t)$

Note: Usually, physical system models involves parameter (p) and unknown disturbance ($w^{(2)}$) vectors, i.e., $\dot{x} = f\left(x, u, w^{(1)}\right)$ in (2.1) would be on the form of $\dot{x} = f\left(x, u, w^{(1)}, w^{(2)}, p\right)$. However, by including $\dot{p} = 0$ and a dynamic model for $w^{(2)}$, it is possible to arrive to the form (2.1). Similar arguments can be applied for augmenting time delays via dynamic models. Therefore, without loss of generality, we consider the state space model (2.1)-(2.2) in the following discussions.

2.1 Observability and Controllability

A dynamic system is observable if its internal states can be inferred based on available output-input information — more precisely stated, observability measures the ability of estimating x_0, $x_1, \ldots,$ and x_k from $y_0, y_1, \ldots,$ and y_k. Controllability can be defined with respect to the state and the output of a system. State controllability characterizes the ability of moving from any initial state to any other final state by applying some admissible input within a finite time span. Output controllability can be defined similarly. Algebraic tests for observability and controllability should be given. First, observability and controllability rank conditions for linear time invariant and variant systems are given [19], followed by the conditions for nonlinear systems. If the rank of the observability matrix $\mathbb{O}_n$ defined in (2.7) is equal to n, then the linear time invariant system given by (2.5)-(2.6) satisfies the observability rank condition. The controllability rank condition is defined similarly: $rank\left(\mathbb{C}_n\right) = n$, where the controllability matrix $\mathbb{C}_n$ is given in (2.8). For the linear time varying case, controllability and observability Gramians [20] are considered (2.11)-(2.12).

$$x_{k+1} = f_k\left(x_k, u_k, w_k^{(1)}\right) = A\,x_k + B\,u_k + L\,w_k^{(1)} \tag{2.5}$$
$$y_k = h_k\left(x_k\right) = C\,x_k \tag{2.6}$$
A, B, L and C are constant matrices.

$$\mathbb{O}_n = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{2.7}$$

$$\mathbb{C}_n = \begin{bmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{bmatrix} \tag{2.8}$$

$$x_{k+1} = f_k\left(x_k, u_k, w_k^{(1)}\right) = A_k\,x_k + B_k\,u_k + L_k\,w_k^{(1)} \tag{2.9}$$
$$y_k = h_k\left(x_k\right) = C_k\,x_k \tag{2.10}$$
A_k, B_k, L_k and C_k are time dependent matrices.

$$\boxed{\begin{array}{l}\mathbb{G}_o(k) = \sum_{i=k}^{k_f} \Phi^T(i,k) C_i^T C_i \Phi(i,k) \\ k_0 \leq k \leq k_f \\ \Phi(k,k) = I \\ \Phi(k,k_0) = A_{k-1} A_{k-2} \cdots A_{k_0} \\ \text{If } \mathbb{G}_o(k) \text{ is nonsingular, the system is observable.}\end{array}} \tag{2.11}$$

$$\boxed{\begin{array}{l}\mathbb{G}_c(k) = \sum_{i=k_0}^{k-1} \Phi(k,i+1) B_i B_i^T \Phi^T(k,i+1) \\ k_0 \leq k \leq k_f \\ \Phi(k,k) = I \\ \Phi(k,k_0) = A_{k-1} A_{k-2} \cdots A_{k_0} \\ \text{If } \mathbb{G}_c(k) \text{ is nonsingular, the system is controllable.}\end{array}} \tag{2.12}$$

Controllability and observability for general nonlinear systems are difficult to handle. One simple possibility to deal with such systems is to consider rank tests for a linearized model. However, full rank of the controllability (observability) matrix of the linearized system is not a necessary condition for controllability (observability) [21]. For linear time invariant/varying systems, there is no distinction between global and local properties such as global and local controllability, while for nonlinear systems there is. Local controllability (observability) implies global controllability (observability), while global properties do not necessarily imply local properties — in other words, local controllability (observability) is a sufficient condition for global controllability (observability).

With the help of the Lie derivatives, we can define a rank test for local observability [22][23][24] in connection to (2.2). The Lie derivative of y with respect to $f^{(i)}$ (for a piecewise constant $u = u_i$ and $w^{(1)}$) is defined in (2.13). Higher order Lie derivatives can also be defined: e.g. $L_{f^{(j)}}(L_{f^{(i)}}h) \equiv (L_{f^{(j)}}L_{f^{(i)}})h$, $(L_{f^{(k)}}L_{f^{(j)}}L_{f^{(i)}})h$. Consider Lie derivatives of order 0, 1, 2, ..., for all piecewise constant u: $h(x)$, $L_{f^{(1)}}h(x)$, $L_{f^{(2)}}h(x)$, $\cdots$, $(L_{f^{(1)}}L_{f^{(2)}})h(x)$, $\cdots$, $(L_{f^{(1)}}L_{f^{(2)}}L_{f^{(3)}})h(x)$, $\cdots$. If it is possible to find n linearly independent rows among these Lie derivatives, then we say the observability rank condition is satisfied locally at x. For linear systems, higher order Lie derivatives are always taken up to order $n-1$, while this is not always true for general nonlinear systems.

$$\boxed{L_{f^{(i)}}h(x) = \frac{\partial}{\partial x} h(x) f\left(x, u_i, w^{(1)}\right)} \tag{2.13}$$

Also, the solvability of the system of nonlinear equations (2.14) can be connected to local observability of x_k for given y_i for $i = k, k+1, \ldots, k+n-1$ and u_i and $w_i^{(1)}$ for $i = k, k+1, \ldots, k+n-2$.

$$\boxed{\begin{array}{l}y_k = h_k(x_k) \\ y_{k+1} = h_k(x_{k+1}) \\ y_{k+2} = h_k(x_{k+2}) \\ \cdots \\ y_{k+n-1} = h_k(x_{k+n-1}) \\ x_{k+i} = f_{k+i-1}\left(x_{k+i-1}, u_{k+i-1}, w_{k+i-1}^{(1)}\right), i = 1, 2, \ldots, n-1\end{array}} \tag{2.14}$$

State observability is related with state distinguishability.[2] If two distinct initial states $x(0)_1$ and $x(0)_2$ result in the same output trajectory, then $x(0)_1$ and $x(0)_2$ are said to be indistinguishable. Loosely speaking, indistinguishability implies observability.

A local controllability rank condition is defined with the help of the Lie bracket which is defined in (2.15). It is possible to define nested Lie brackets such as $[f^{(1)},[f^{(2)},f^{(3)}]]$ and $[f^{(1)},[f^{(2)},[f^{(3)},f^{(4)}]]]$. If there exists n linearly dependent vectors, which are made up from Lie brackets or nested Lie brackets for all possible admissible inputs, then we say that the local controllability rank condition is satisfied. For more complete discussions on the topics nonlinear observability and controllability [25] and [26] are refereed.

$$\boxed{\left[f^{(i)},f^{(j)}\right]=\frac{\partial f^{(j)}}{\partial x}f^{(i)}-\frac{\partial f^{(j)}}{\partial x}f^{(i)}} \tag{2.15}$$

2.2 Structural Observability and Controllability

In Section 2.1, the state space system characterization is used to discuss the concepts algebraic controllability and observability. This section explains a different approach to describe controllability and observability based on the structure of the state space model. Several factors supports using such an approach [14][27][28]: (1) In particular, for large-scale nonlinear systems the analysis involves searching for the rank of matrices with higher dimensions; (2) symbolic manipulations of these matrices (even for linear time invariant systems) are tedious and may be impractical; and (3) often, the model contains uncertain or unknown parameters — for example, the elements of system matrices A_s, B_s, L_s and C_s of $\dot{x}=A_sx+B_su+L_sw^{(1)}$ and $y=C_sx$ — causing the conclusions drawn about controllability and observability susceptible for parameter uncertainties [29]. In the following paragraph, we discuss the graph-theoretic approach for structural analysis starting with linear time invariant systems and extending to nonlinear systems.

Consider the linear time invariant system (2.16)-(2.17). a_{ij}, b_{ij} and c_{ij} are $<i,j>^{\text{th}}$ elements of the A_s, B_s and C_s matrices, respectively. a_{ij}, b_{ij} and c_{ij} may vanish for some i and j; such elements are called structural zeros. An example is given in (2.20). It is found that $det(\mathbb{O}_3)=c_{11}^3a_{12}^2a_{23}$ and $det(\mathbb{C}_3)=-b_{21}$. $rank(\mathbb{O}_3)=3$ and $rank(\mathbb{C}_3)=3$ for almost all parameter values except for some pathological situations, i.e., $rank(\mathbb{O}_3)<3$ for $c_{11}=0$ or $a_{12}=0$ or $a_{23}=0$ and $rank(\mathbb{C}_3)<3$ for $b_{21}=0$. In structural analysis the term "almost all" is key.

$$\boxed{\begin{aligned}&\dot{x}=A_sx+B_su+L_sw^{(1)} &(2.16)\\ &y=C_sx &(2.17)\\ &A_s,\ B_s,\ L_s \text{ and } C_s \text{ are constant matrices.}\\ &u \text{ and } w^{(1)} \text{ are scalars.}\end{aligned}}$$

$$\boxed{\begin{aligned}&\text{Pair}(A_s,C_s) \text{ is observable if and only if} &(2.18)\\ &rank\begin{bmatrix}\lambda_iI-A_s\\ C_s\end{bmatrix}=n\\ &\text{for all eigenvalues of } A_s,\ \lambda_1,\ \lambda_2,\cdots, \text{ and } \lambda_n.\end{aligned}}$$

$$\boxed{\begin{aligned}&\text{Pair}(A_s,B_s) \text{ is controllable if and only if} &(2.19)\\ &rank\begin{bmatrix}\lambda_iI-A_s & B_s\end{bmatrix}=n\\ &\text{for all eigenvalues of } A_s,\ \lambda_1,\ \lambda_2,\cdots, \text{ and } \lambda_n.\end{aligned}}$$

[2] Observability implies that the state estimation may be possible for some inputs, but not for all.

$$A_s = \begin{bmatrix} 0 & a_{12} & 0 \\ 0 & 0 & a_{23} \\ 0 & a_{32} & a_{33} \end{bmatrix} \tag{2.20}$$

$$B_s = \begin{bmatrix} 0 \\ b_{21} \\ 0 \end{bmatrix}$$

$$L_s = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$C_s = \begin{bmatrix} c_{11} & 0 & 0 \end{bmatrix} \tag{2.21}$$

Three directed graphs (or digraphs) G_o, G_{c_x} and G_{c_y} are created for structurally observability and (output/state) controllability analysis. If v_i and v_j are nodes, then $v_i \to v_j$ denotes a directed edge emanating from v_i and ending at v_j. The digraphs are structured in the following way:

- G_o: The nodes corresponds to state and output variables — i.e. $x_1, x_2, \cdots, x_{n_x}, y_1, y_2, \cdots,$ and y_{n_y}. If $a_{ij} \neq 0$, then $x_i \to x_j$ exists and if $c_{ij} \neq 0$, then $y_i \to x_j$ exists.
- G_{c_x}: The nodes corresponds to state and input variables — i.e. $x_1, x_2, \cdots, x_{n_x}, u_1, u_2, \cdots,$ and u_{n_u}. If $a_{ij} \neq 0$, then $x_j \to x_i$ exists and if $b_{ij} \neq 0$, then $u_j \to x_i$ exists.
- G_{c_y}: The nodes corresponds to state, input and output variables — i.e. $x_1, x_2, \cdots, x_{n_x}, u_1, u_2, \cdots, u_{n_u}, y_1, y_2, \cdots$ and y_{n_y}. If $a_{ij} \neq 0$, then $x_j \to x_i$ exists, if $b_{ij} \neq 0$, then $u_j \to x_i$ exists and if $c_{ij} \neq 0$, then $x_j \to y_i$ exists.

G_o, G_{c_x} and G_{c_y} for the example (2.20) are given in Figures 2.1, 2.2 and 2.3, respectively. The term "structure" [14] should be clarified. It is convenient to consider the example given above. The structure matrices of A_s, B_s, L_s, and C_s are represented by $[A_s]$, $[B_s]$, $[L_s]$, and $[C_s]$. For example,

$$[A_s] = \begin{bmatrix} 0 & * & 0 \\ 0 & 0 & * \\ 0 & * & * \end{bmatrix}.$$

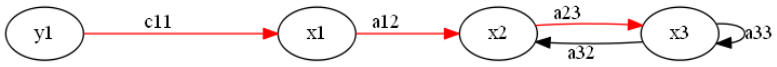


Figure 2.1: G_o for the example in (2.20): $y_1 \to x_1 \to x_2 \to x_3$ is the stem and there are no buds.

In $[A_s]$, $*$ replaces all nonzero elements. The idea is that the numerical values of matrix entities are taken as indeterminate. By doing so, we let structural properties (so-called generic properties) to be held for almost all realizations of structure matrices — a realization of a structure matrix is found by assigning numerical values to its indeterminate elements. Also, a generic property always give a necessary condition for the property of interest — e.g., not structural observable $\Rightarrow$ not observable.[3] Pair($[A,B]$) is structurally controllable if we can find at least one realization of the pair($[A,B]$) which is controllable. Structural observability is defined similarly. A structural or generic property of interest can be mapped into a property of the related directed graph.

[3]Controllability with respect to $w^{(1)}$ can also be considered in a similar fashion.

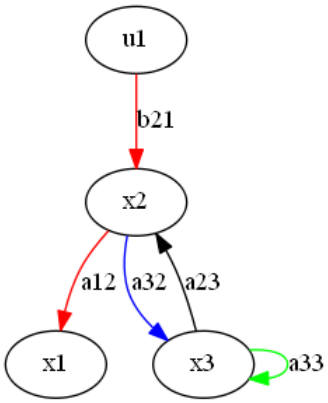


Figure 2.2: G_{c_x} for the example in (2.20): $u_1 \to x_2 \to x_1$ is the stem; $x_3 \to x_3$ is the bud; x_2 is the origin of the bud; and $x_2 \to x_3$ is the distinguish edge.

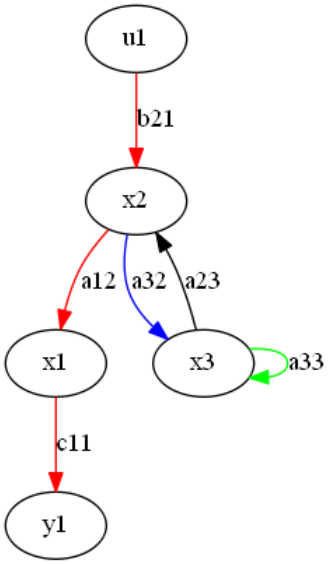


Figure 2.3: G_{c_y} for the example in (2.20): $u_1 \to x_2 \to x_1 \to y_1$ is the stem; $x_3 \to x_3$ is the bud; x_2 is the origin of the bud; and $x_2 \to x_3$ is the distinguished edge.

Generic or structural rank $\rho([M])$ of the structure matrix $[M]$ is defined to be the maximal rank which can be found out of the ranks of all possible realizations of $[M]$. In the above example (2.20), $\rho([A_s, B_s]) = 3$ and $\rho([A_s, C_s]) = 3$. In order to achieve structural state controllability, two conditions must be met: (1) State-nodes are input-connected and (2) $\rho([A, B]) = n$. If there is at least one directed path starting from any input-node to each state-node, then the state-nodes are input connected. The conditions (1) and (2) can be combined to give a single, complete graph-theoretic equivalent: If G_{c_x} is spanned by a cactus, then the pair(A, B) is structurally state controllable [14] — structural observability and output controllability are defined similarly for G_o and G_{c_y}.

A cactus is a special graph structure which satisfies both requirements (1) and (2). First, consider systems with a single input u_1. A cactus is made out of a stem and one or more buds. Figures 2.4 and 2.5 show bud and stem structures. It is easy to prove that both of these digraphs represents structurally controllable systems. Structural controllability is collapsed by removing at least one of the edges of the stem and bud structures. Now, a way of constructing a cactus is given. Let $u_1 \to x_1 \to x_2 \to \cdots \to x_m$ be a stem S_1, where m is a positive integer. If a bud B_1 with its origin

x_{m+1} can be connected to any nodes of S_1, then $S_1 \cup B_1$ is a cactus. Note that it is possible to attach B_1 either at the top (i.e. at x_m) or the root (i.e. at u_1) of the stem. We can add another bud B_2 to $S_1 \cup B_1$ with some constraints about location where the origin of B_2 is placed: The origin of B_2 cannot be connected to the ending node of the distinguished edge of B_1. The new digraph is $S_1 \cup B_1 \cup B_2$. We can keep on adding buds B_3, B_4, and so on. Figure 2.6 shows an example of a cactus. Adding a bud at the top of the stem is equivalent to lengthening the stem. If we reconsider the example (2.20), we can now see that the given system is structurally observable, state controllable and output controllable. For multiple input systems, the digraphs should be spanned by cacti.

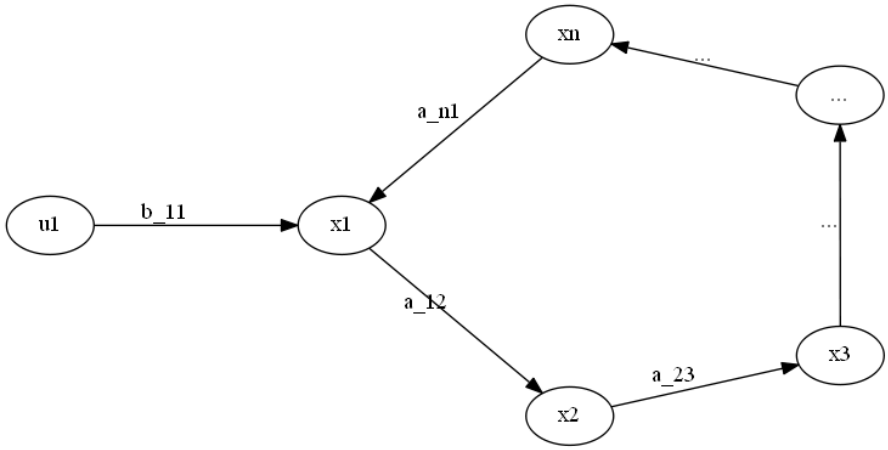


Figure 2.4: A bud: u_1 is the origin and $u_1 \to x_1$ is the distinguished edge.

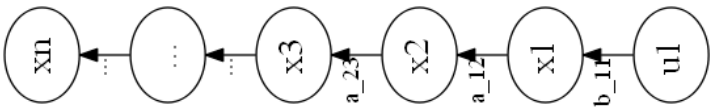


Figure 2.5: A stem: x_1 is the root and x_n is the top.

Structural analysis is extended to nonlinear systems. Consider the dynamic system given in (2.1)-(2.2). Instead of applying systems matrices like in the linear time invariant case, symbolic Jacobian matrices are used to generate digraphs. For example, consider structural output controllability analysis. If

$$\boxed{\partial f_i\left(x,u,w^{(1)}\right)/\partial x_j \neq 0,}$$

— i.e., x_j appears in $f_i\left(x,u,w^{(1)}\right)$ — then there is an edge from x_j to x_i. Section 2.1 discusses algebraic rank tests for observability and controllability. For example, if we can prove that a system is not structurally controllable, then there exists no n linear independent vectors among all possible Lie brackets. Consequently the system is not locally controllable. Similarly, other structural tests can be linked to relevant algebraic tests.

So far we have considered $f\left(x,u,w^{(1)}\right)$. In order to extend the discussion to include parameter and disturbance estimation, we consider $f\left(x,u,w^{(1)},w^{(2)},p\right)$ instead. $w^{(2)}$ and p are to be estimated, hence they are augmented as state variables. The new dynamic model is called the augmented state space model. Observability/controllability should be checked on the augmented model, see (2.22)-(2.25). Parameter and/or disturbance augmentation may degrade observability. Parameters must be augmented as given in (2.23) — the time derivative of a parameter is always

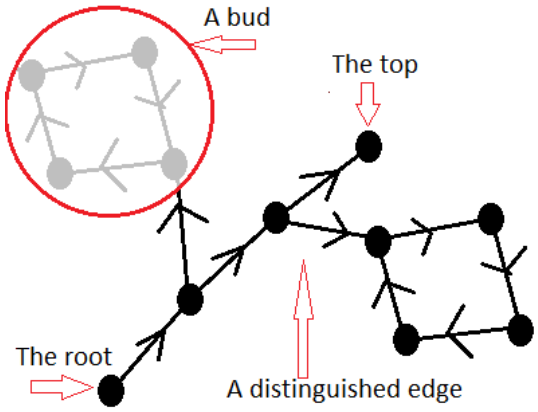


Figure 2.6: A cactus.

zero. However, there could be more possibilities to augment unknown disturbances. Models for the augmentation should be picked in such a way that structural observability is maintained — i.e. $\psi(w^{(2)})$ in (2.24) should be carefully defined.

Due to $\dot{p}_i = 0$, edges emanating from a parameter is impossible. A parameter node may have edges coming towards it from state and/or output nodes. Also, edges among parameter nodes and p_i cannot be a part of any cycle. These characteristics of parameter nodes make it harder to achieve structural observability, especially when the state space dimension is large. The one and only possibility for the parameter nodes to be included in G_o, without collapsing the spanning cacti of G_o, is to attach them at the end of each stem (providing that the system equations permits this). In this case, the upper bound of the number of parameters that can be estimated is equal to the number of stems, which is of course less than the number of measurements. In the case where the parameter nodes collapse the spanning cacti of the original digraph of G_o, we have to search for a new spanning cacti for the digraph of the augmented model. The success of such a search is determined by the degree of dependency among state and output nodes. For the digraph of the parameter-augmented system $G_{o,p}$, the parameter nodes must satisfy several constraints in order to have a spanning cacti covering all parameter and state nodes: (1) Each stem must ends at a parameter node, (2) there cannot be any parameter that is only connected to one output node which is not included in a cactus, and (3) if there are multiple parameter nodes connected to a state node x_i, then those parameters cannot just appear only in $\dot{x}_i = f_i(\cdot)$. (3) implies that there are too many parameters in the equation $\dot{x}_i = f_i(\cdot)$ and one possibility to reduce this number is to lump parameters. If G_o is not structurally observable, then $G_{o,p}$ cannot be observable. Nevertheless, if G_o is structurally observable, $G_{o,p}$ may or may not be structurally observable.

On the other hand, $\dot{w^{(2)}} = \psi(w^{(2)})$ in (2.24) is less demanding as compared to $\dot{p} = 0$ in the sense of preserving structural observability, unless $w^{(2)}$ is augmented as $w^{(2)} = 0$ which is similar to parameter augmentation. For example, $\dot{w}_i^{(2)} = -\beta_i w_i^{(2)}$ adds a self-cycle ($\beta_i > 0$). Adding a cycle is similar to attaching a bud to G_o, and consequently increase the chances of achieving structural observability. Also, the number of output nodes restricts the number of parameters that can be observed, although this may not be the case if disturbances are augmented with buds. Another interesting fact is that for given augmented system it is possible to find a minimum set of additional measurements, in the structural observability sense, to be able to estimate state variable, augmented parameters and disturbances.

$$\boxed{\begin{aligned}
&\dot{x} = f\left(x, u, w^{(1)}, w^{(2)}, p\right) && (2.22)\\
&\dot{p} = 0 && (2.23)\\
&\dot{w^{(2)}} = \psi(w^{(2)}) && (2.24)\\
&p = \left[p_1, p_2, \cdots p_{n_p}\right]\\
&w^{(2)} = \left[w_1^{(2)}, w_2^{(2)}, \cdots w_{n_{w^{(2)}}}^{(2)}\right]\\
&n_p = dim(p)\\
&n_{w^{(2)}} = dim(w^{(2)})
\end{aligned}}$$

2.3 State-Parameter-Disturbance Estimation

State observability implies that input-output information contain a complete description of a system's internal state. Sections 2.1 and 2.2 describes algebraic observability and controllability where noise-free input-output information is assumed. For example, if (2.5)-(2.6) is observable then x_k can be estimated by algebraically solving $y_k = Cx_k$, $y_{k+1} = Cx_{k+1}$, $\cdots$ and $y_{k+n-1} = Cx_{k+n-1}$. An estimator of this type does not consider noisy-data. The practical interest is to have a state estimator which extracts state information buried in noisy measurements. For example, let the noisy-system of (2.5)-(2.6) be $y_k = Cx_k + v_k$ and $x_{k+1} = A_k\, x_k + B_k\, u_k + L_k\, w_k^{(1)} + \varepsilon_k$ where v_k and ε_k are measurement and process noises, respecively. Now, the success of the estimation process demands more than algebraic observability. In addition to algebraic observability, stochastic observability and controllability [30] should be checked. These concepts will be discussed within this section.

Reconstruction of x_k is needed for many applications. In the implementation of optimal control strategies, an estimate of the current state x_k is required to calculate the current control action u_k. If x is an augmented state, then an estimate of x_k contains information about augmented variables such as parameters and unknown disturbances. The majority of practical estimation problems are nonlinear. Available linear estimation methods, such as the Extended Kalman Filter [31], may be adapted to solve nonlinear estimation problems. The Nonlinear Moving Horizon Estimate [32] is a nonlinear estimate. The following sections describes various estimators in detail considering the noisy system (2.25)-(2.26) or (2.27)-(2.28), where $\hat{x}$ is an estimate of x.

$$\boxed{\begin{aligned}
&\dot{\hat{x}} = f\left(\hat{x}, u, w^{(1)}\right) + \varepsilon && (2.25)\\
&\hat{y} = h(\hat{x}) + v && (2.26)
\end{aligned}}$$

$$\boxed{\begin{aligned}
&\hat{x}_{k+1} = f_k\left(\hat{x}_k, u_k, w_k^{(1)}\right) + \varepsilon_k && (2.27)\\
&\hat{y}_k = h_k(\hat{x}_k) + v_k && (2.28)
\end{aligned}}$$

2.3.1 Linear Filtering

Consider a discrete linear time-varying noisy-system of the form (2.29)-(2.30). $\hat{x}_{i|k}$ $(i \leq k)$ is the estimate of x_i using measurements of y_j for $j \leq k$. When $i = k$, we have a filtering problem ($i < k$ is for state smoothing and $i > k$ is for state prediction problems [33]. These cases are not considered). Let $\{y_0, y_1, \cdots, y_N\}$, $\{u_0, u_1, \cdots, u_{N-1}\}$, and $\{w_0^{(1)}, w_1^{(1)}, \cdots, w_{N-1}^{(1)}\}$ be measurement sets. We can formulate a least (weighted) squares problem as given in (2.31). By solving (2.31), we obtain $\hat{x}_0$, $\hat{x}_1$, $\cdots$, $\hat{x}_N$, $\hat{v}_0$, $\hat{v}_1$, $\cdots$, $\hat{v}_N$, $\hat{\varepsilon}_0$, $\hat{\varepsilon}_1$, $\cdots$, and $\hat{\varepsilon}_{N-1}$ at once — i.e. minimizing $\sum_{i=0}^{N}\left(v_i^2/r_i\right) + \sum_{i=0}^{N-1}\left(\varepsilon_i^2/q_i\right)$ where r_i and q_i are given weights. However, a recursive method is preferable. [34] shows a way of deriving a recursive estimate purely using linear algebraic techniques

starting from (2.31). If measurements are reliable then choose $r_i/q_i \ll 1$ and if measurements are not so reliable then choose $r_i/q_i \gg 1$.

$$\boxed{\hat{x}_{k+1} = A_k\hat{x}_k + B_k u_k + L_k w_k^{(1)} + \varepsilon_k} \tag{2.29}$$

$$\boxed{\hat{y}_k = C_k\hat{x}_k + v_k} \tag{2.30}$$

$$\begin{bmatrix} y_0 \\ 0 \\ y_1 \\ \vdots \\ 0 \\ y_N \end{bmatrix} = \zeta \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix} + \begin{bmatrix} 0 \\ B_{N-1}u_0 + L_{N-1}w_0^{(1)} \\ 0 \\ B_1 u_1 + L_1 w_1^{(1)} \\ \vdots \\ B_{N-1}u_{N-1} + L_{N-1}w_{N-1}^{(1)} \\ 0 \end{bmatrix} + \begin{bmatrix} v_0 \\ \varepsilon_0 \\ v_1 \\ \varepsilon_1 \\ \vdots \\ \varepsilon_{N-1} \\ v_N \end{bmatrix} \tag{2.31}$$

$$\zeta = \begin{bmatrix} C_0 & & & & & & \\ -A_0 & I & & & & & \\ & C_1 & & & & & \\ & -A_1 & I & & & & \\ & & \ddots & & & & \\ & & & -A_{N-1} & I & & \\ & & & & & C_N \end{bmatrix}$$

The Kalman Filter algorithm is given in (2.32) [33]. v_i and ε_i are independent, zero mean, and normally distributed random variables. $R_k > 0$ and $Q_k \geq 0$ are covariance matrices of v_i and ε_i. To initiate the Kalman Filter, $\hat{x}_0^+$ and $P_0^+ \geq 0$ are given. Note that the equations in (2.32) can be derived from (2.31) [34].

$$\boxed{\begin{aligned} P_k^- &= A_{k-1}P_{k-1}^+A_{k-1}^T + Q_{k-1} \\ K_k &= P_k^- C_k^T (C_k P_k^- C_k^T + R_k)^{-1} \\ \hat{x}_k^- &= A_{k-1}\hat{x}_{k-1}^+ + B_{k-1}u_{k-1} + L_{k-1}w_{k-1}^{(1)} \\ \hat{x}_k^+ &= \hat{x}_k^- + K_k\left(y_k - C_k\hat{x}_k^-\right) \\ P_k^+ &= (I - K_k C_k) P_k^- \end{aligned}} \tag{2.32}$$

The convergence characteristics needs to be discussed [35][36]. Loosely speaking, the convergence means the convergence of covariance P_k^+ for $k \to \infty$. Linear time-invariant systems are discussed first. The Riccati difference equation [35]

$$\boxed{P_k^+ = AP_{k-1}^+A^T - AP_{k-1}^+C^T(CP_{k-1}^+C^T + R)^{-1}CP_{k-1}^+A^T + Q,}$$

is analyzed. By replacing A from $I + A\Delta t$ and evaluating

$$\boxed{\underset{\Delta t \to 0}{Limit} \frac{P_k^+ - P_{k-1}^+}{\Delta t},}$$

we get,

$$\boxed{\dot{P} = AP + PA^T - PC^T R_c^{-1} CP + Q_c.}$$

If unstable modes of A are (1) observable (detectability) and (2) controllable with respect to process noise (stabilizability with respect to process noise), then there exists a unique positive semidefinite solution for $\dot{P} = 0$ and the solution is asymptotically stable [37][38]. Here we assume $Q_c \geq 0$ and $R_c > 0$. In the case of parameter estimation, we use $\dot{p}_i = 0$ type of state equations where it is, theoretically, not allowed to include a process noise variable. Therefore,

the stabilizability condition may need to be relaxed [35][39]. If the stabilizability condition is removed, then additional constraints must be prevailed on the state transition matrix [39].

For linear time-varying system the conditions for the Kalman Filter's convergence are given in [30]. Sufficient conditions for uniformly completely controllability (corresponds to exciting state variables by the process noise) and uniformly completely observability are given: (2.33)-(2.34). By adding fictitious process noise variables, it may be possible to make sure the above mentioned conditions are satisfied [36]. In practical situations it is not always necessary to satisfy these two conditions [40].

$$\boxed{\begin{array}{l}\text{If } \beta_1 I \leq \sum_{i=k-m}^{k-1} \Phi(k,i+1) Q_i \Phi^T(k,i+1) \leq \beta_2, \text{ for some } m > 0, \\ \text{then the system is uniformly completely controllability, where } \beta_1, \beta_2 \geq 0.\end{array}} \tag{2.33}$$

$$\boxed{\begin{array}{l}\text{If } \alpha_1 I \leq \sum_{i=k-m}^{k-1} \Phi^T(i,k) C_i^T R_i^{-1} C_\Phi(i,k) \leq \alpha_2, \text{ for some } m > 0, \\ \text{then the system is uniformly completely controllability, where } \beta_1, \beta_2 \geq 0.\end{array}} \tag{2.34}$$

Another alternative is the *Moving Horizon Approach*. The *Moving Horizon Estimate* estimates x_k using the measurements within the time span $[t_k - N_h\Delta t, t_k]$ (or y_j for $k - N_h \leq j \leq k$). The objective function J_{mhe} is minimized subjected to (2.29)-(2.30). The decision variables are x_{k-N_h}, $\cdots, x_{k-1}, x_k, \varepsilon_{k-N_h}, \cdots, \varepsilon_{k-2}$, and ε_{k-1}. It is also possible to include additional constraints such as equality and/or inequality constraints on decision variables. $\left[(x_{k-N_h} - \hat{x}_{k-N_h})^T Q_0 (x_{k-N_h} - \hat{x}_{k-N_h})\right]$ is the arrival cost which affects the estimate's performance. By choosing a longer horizon, the effect of the arrival cost can be mitigated.

$$\boxed{\begin{array}{l} J_{mhe} = \sum_{i=k-N_h}^{k-1} \left[\varepsilon_i^T Q_i \varepsilon_i\right] + \sum_{i=k-N_h}^{k} \left[(y_i - C_i x_i)^T R_i (y_i - C_i x_i)\right] + \\ \left[(x_{k-N_h} - \hat{x}_{k-N_h})^T Q_0 (x_{k-N_h} - \hat{x}_{k-N_h})\right] \end{array}} \tag{2.35}$$

2.3.2 Nonlinear Filtering

A straight forward approach is to implement nonlinear weighted least-squares estimation. That means minimizing

$$\boxed{\sum_{i=1}^{k} \left[(y_i - h_i(\hat{x}_i))^T w_i (y_i - h_i(\hat{x}_i))\right],}$$

subjected to $\hat{x}_{k+1} = f_k\left(\hat{x}_k, u_k, w_k^{(1)}\right)$. In these situations, no statistical assumptions are made on ε_i and v_i. Usually, the *Extended Kalman Filter* seems to be the starting point of given nonlinear estimation problem. The *Extended Kalman Filter* may cause diverge in some situations [36][41]. A_{k-1} and C_k in 2.32 are replaced by

$$\boxed{A_{k-1} = \frac{\partial}{\partial x_{k-1}} f_{k-1}\left(x_{k-1}, u_{k-1}, w_{k-1}^{(1)}\right)|_{\hat{x}_{k-1}^+}}$$

and

$$\boxed{C_k = \frac{\partial}{\partial x_k} h_k(x_k)|_{\hat{x}_k^-}.}$$

The Extended Kalman Filter's performance is susceptible to linearization errors. It can be

showed that even in simple system parameter estimation problems, the filter can fail. With the inclusion of some instability term, we can restore the convergence [42].

In (2.27)-(2.28) additive process and measurement noises are considered. However, the more general Nonlinear Moving Horizon Estimate [32] is used when more general nonlinear systems are involved. There are a few advantages of moving horizon type approaches: The performance is not affected by linearization errors and this option has better convergence characteristics. The disadvantage is that the implementation requires more computer power. The following objective function is minimized subjected to $\hat{x}_{k+1} = f_k(\hat{x}_k, u_k, w_k^{(1)}, \varepsilon_k)$, $\hat{y}_k = h_k(\hat{x}_k, v_k)$ and other constraints. Some possible other nonlinear filters are the Unscented Kalman Filter and the Particle Filter.

$$\boxed{\sum_{i=k-N_h}^{k-1}\left[\varepsilon_i^T Q_i \varepsilon_i\right] + \sum_{i=k-N_h}^{k}\left[(y_i - h_i(x_i))^T R_i (y_i - h_i(x_i)x_i)\right] + \left[(x_{k-N_h} - \hat{x}_{k-N_h})^T Q_0 (x_{k-N_h} - \hat{x}_{k-N_h})\right]} \tag{2.36}$$

2.4 Nonlinear Programming

In this section an overview is given for nonlinear programming within the context of solving optimal control problems numerically. In nonlinear programming, the objective is to optimize (minimize/maximize) a scalar objective/cost function $J(x)$ which satisfies some given equality and/or inequality constraints. The elements of x are called the decision variables or the degree of freedom in the optimization process. An optimizer solves a nonlinear programming or optimization problem of interest. There are many optimizers available such as IPOPT, the Interior Point OPTimizer [43][44][45]. An optimal control problem can be formulated into a nonlinear programming/optimization problem as mentioned above. The first subsection discusses basics of nonlinear programming, and the second one considers numerical solutions of optimal control problems.

2.4.1 Nonlinear Programming: Fundamentals

Most of the subject material presented in this subsection is taken from [46]. The nonlinear optimization problems is given as follows:

$$\begin{aligned} \underset{x\in R^{n1}}{\text{minimize}} \quad & J(x) \\ \text{subject to} \quad & c_i(x) = 0,\ i = 1,2,...,n_2, \\ & c_i(x) \geq 0, i = n_2+1, n_2+2, ..., n_2+n_3. \end{aligned} \tag{2.37}$$

where n_1, n_2 and n_3 are positive integers. The feasibility region S is a set of all the points which satisfy all equality and inequality constraints. We consider the case when the decision variables are real numbers (i.e. to avoid mixed integer programming). If no constraints are present, then it is an unconstrained optimization problem. If there exists an x^*, such that $J(x) \geq J(x^*)$, for all $x \in S$ then x^* is a global minimum. If $J(x) \geq J(x^*)$ holds for some neighborhood of x^*, x^* is a local minimum. A strict global/local minimum is defined when $J(x) > J(x^*)$ for all $x \neq x^*$. An interesting class of optimization problems arises when both $J(x)$ and S possess the convex property — i.e., if $J(\alpha x + (1-\alpha)y) \leq \alpha J(x) + (1-\alpha)J(y)$ for all $x, y \in S$ and $\alpha \in [0,1]$. Then J is a convex function and similar definition is given for convex sets. Convexity of $J(x)$ and S implies any local minimum is a global minimum. Sufficient conditions for x^* to be a strict local minimum are that $\partial J(x)/\partial x|_{x^*} = 0$ and that $\partial^2 J(x)/\partial x^2|_{x^*}$ is positive definite. Solution searching algorithms are iterative, they start with a given initial guess x_0 and iterate until an x^*

is found such that $\mid x_k - x^* \mid \leq \varepsilon$, where ε is the tolerance and x_k is the solution after the k$^{\text{th}}$ iteration. First we consider the unconstrained optimization, followed by the constrained case.

Define $\partial J(x)/\partial x \triangleq F(x)$. The classical Newton's solution is given by 2.38 to 2.40, where G_k should be invertible. p_k is the search direction at x_k. Instead of $x_{k+1} = x_k + p_k$, $x_{k+1} = x_k + \alpha p_k$ ($\alpha > 0$) is used in line search and trust region methods.

$$x_{k+1} = x_k + p_k, \tag{2.38}$$

$$G_k p_k = -F(x_k), \tag{2.39}$$

$$G_k = \left(\frac{\partial F}{\partial x}\right)\Big|_{x_k} = \left(\frac{\partial^2 J}{\partial x^2}\right)\Big|_{x_k}, \tag{2.40}$$

In the line search method, α is calculated by minimizing $F(x_k + \alpha p_k)$. We may settle for the very first minimum found when moving along p_k. On the other hand, in trust region techniques, an approximate function $m_k(p) \approx F(x_k + p)$ — e.g., using the first three element of the Taylor series expansion of $F(x_k + p) \approx F(x_k) + \partial F(x)/\partial x\Big|_{x_k} p + \frac{1}{2} p^T \left[\partial^2 F(x)/\partial x^2\right]\Big|_{x_k} p$ — is defined within the trust region $\mid p - x_k \mid \leq r$ (r>0). Then, p^* is found minimizing $m_k(p)$ and hence, the new solution becomes $x_{k+1} = x_k + p^*$.

Nonlinear conjugate gradient type of methods [43] can also be considered, but thisis not discussed further here. In quasi-Newton methods, the Hessian matrix $\partial^2 F(x)/\partial x^2\Big|_{x_k}$ in $m_k(p) = F(x_k) + \partial F(x)/\partial x\Big|_{x_k} p + \frac{1}{2} p^T \left[\partial^2 F(x)/\partial x^2\Big|_{x_k}\right] p$ is approximated. Different variants of this method exists, for example the BFGS method. So far the methods which have been discussed involved the evaluation of Jacobian (see [47] for algorithmic differentiation for the evaluation of the derivative with high precision) and Hessian matrices, or at least a Jacobian matrix. However, in the situations where the derivatives are not available, we can use derivative free optimization where derivatives are numerically approximated. One immediate approximation procedure is the finite difference method.

In order to solve the constrained optimization, the Lagrangian function $\mathscr{L}$ is defined, see (2.41). By solving (2.42) and (2.43), we can find x^* and λ^*. Note that for equality constraints we always have $\lambda_i > 0$. To handle inequality constraints, the complementary equations (2.44) are used, where if $c_j(x^*)$ is active then $\lambda_j = 0$ — see Karush-Kuhn-Tucker conditions in [46].

$$\mathscr{L}(x,\lambda) = J(x) - \sum_{i=1}^{n_2} \lambda_i c_i(x);\ i = 1,2,...,n_2 \tag{2.41}$$

$$\lambda = \left[\lambda_1, \lambda_2, ..., \lambda_{n_2}\right]^T$$

$$\lambda_i > 0$$

$$\left(\frac{\partial}{\partial x}\mathscr{L}(x,\lambda)\right)\Big|_{x^*,\lambda^*} = 0 = \left(\frac{\partial}{\partial x} J(x)\right)\Big|_{x^*} - \left(\sum_{i=1}^{n_2} \lambda_i \frac{\partial}{\partial x} c_i(x)\right)\Big|_{x^*,\lambda^*} \tag{2.42}$$

$$\left(\frac{\partial}{\partial \lambda_i}\mathscr{L}(x,\lambda)\right)\Big|_{x^*,\lambda^*} = 0 = c_i(x^*) \tag{2.43}$$

$$\lambda_j c_j(x^*) = 0;\ j = n_2 + 1, n_2 + 2, ..., n_2 + n_3 \tag{2.44}$$

$$\lambda_j \geq 0$$

Another practical way of handling constraints is to define a cost function with quadratic penalties, see (2.45). $J_2(x,\mu)$ is minimized and during each iteration μ_k is increased (e.g., $\mu_{k+1} = 1.5\mu_k$ [46]) which forces $c_i(x) \to 0$ rapidly. Other methods on nonlinear programming such as the sequential quadratic programming and interior-point methods are not discussed here, a comprehensive discussion on related subject matters are found in [46] and [44].

$$J_2(x,\mu) = J(x) + \frac{\mu}{2}\sum_{i=1}^{n_2} c_i^2(x);\ 1 = 1,2,...,n_2 \tag{2.45}$$
$$\mu > 0$$

2.4.2 Nonlinear Programming: Dynamic Optimization (Optimal Control)

An optimal control problem may be transcribed into a nonlinear programming problem; this means formulating $J(x)$, and $c_i(x)$ for $i = 1,2,...,n_2,n_2+1,n_2+2,...,n_2+n_3$. The following discussion recalls the discrete system (2.3)-(2.4). To obtain (2.3)-(2.4), we need an ODE solver (such as Runge-Kutta type methods) to integrate the continuous system (2.1)-(2.2). There are various strategies of formulating a dynamic optimization problem. These are briefly discussed in the following paragraphs , for a detailed discussion refer [44][45].

Consider an optimal control problem. The time span is $t \in [t_0,t_f]$ and $\Delta t = (t_f - t_0)/N_p$. The control sequence is $\{u_0,u_1,...,u_{N_p-1}\}$ ($\{w_0^{(1)},w_1^{(1)},...,w_{N_p-1}^{(1)}\}$ is assumed to be known), where $u_i = u_{i+1}$ for $i \geq N_u - 1$. N_u is the control horizon and $N_u \leq N_p$. N_p is the output horizon. $\{x_1,x_2,...,x_{N_p}\}$ and $\{y_1,y_2,...,y_{N_p}\}$ are state and output sequences. The initial state x_0 is given, thereby, y_0 is also known. The decision variables are $\{u_0,u_1,...,u_{N_p-1},x_1,x_2,...,x_{N_p},y_1,y_2,...,y_{N_p}\}$; Altogether there are $3N_p$ number of variables. y_{sp} is the set point of y. The objective function,

$$J(x) = \sum_{k=1}^{N_p}\left[(y_k - y_{sp})^T Q_y (y_k - y_{sp})\right] + \sum_{k=1}^{N_u-1}\left[(u_k - u_{k-1})^T Q_u (\underbrace{u_k - u_{k-1}}_{\Delta u_k})\right] \tag{2.46}$$
$$x = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N_p-1} \\ x_1 \\ x_2 \\ \vdots \\ x_{N_p} \\ y_1 \\ y_2 \\ \vdots \\ y_{N_p} \end{bmatrix}$$

is considered. (2.47) gives equality constraints $c_i(x)$ for $i = 1,2,...,3N_p - N_u$. For example, we can define the Lagrangian function and follow the procedure given in Subsection 2.4.1. See (2.48). Notice that we could have included inequality constraints on decision variables as well. The objective function may be expressed as function of controls only. To do so, $\{x_1,x_2,...,x_{N_p}\}$ and $\{y_1,y_2,...,y_{N_p}\}$ are expressed in terms of controls $\{u_0,u_1,...,u_{N_p-1}$. The latter objective function is involves fewer decision variables and the function's complexity is higher. Also, $N_p \to \infty$ (equivalently, $\Delta t \to 0$) makes the solution to above the optimal control problem reaches to a limiting solution.

$$
\begin{aligned}
&c_1(x) = x_1 - f_0\left(x_0, u_0, w_0^{(1)}\right) = 0 \\
&c_2(x) = x_2 - f_1\left(x_1, u_1, w_1^{(1)}\right) = 0 \\
&\ldots \\
&c_{N_p}(x) = x_{N_p} - f_{N_p-1}\left(x_{N_p-1}, u_{N_p-1}, w_{N_p-1}^{(1)}\right) = 0 \\
&c_{N_p+1}(x) = y_1 - h_1(x_1) = 0 \\
&c_{N_p+2}(x) = y_2 - h_2(x_2) = 0 \\
&\ldots \\
&c_{2N_p}(x) = y_{N_p} - h_{N_p}\left(x_{N_p}\right) = 0 \\
&c_{2N_p+1}(x) = u_{N_u-1} - u_{N_u} = 0 \\
&c_{2N_p+2}(x) = u_{N_u} - u_{N_u+1} = 0 \\
&\ldots \\
&c_{3N_p-N_u}(x) = u_{N_u-2} - u_{N_p-1} = 0
\end{aligned}
\tag{2.47}
$$

$$
\mathscr{L}(x, \lambda) = J(x) - \sum_{i=1}^{3N_p-N_u} \lambda_i c_i(x); \tag{2.48}
$$

Numerical approaches to solve optimal control problems are of kinds: indirect and direct methods [44]. The calculus of variations is used in indirect methods — optimize before discretize. The example given above was handled by direct methods — discretize before optimize. Two type variants were considered: (1) parameterization of $u(t)$, $x(t)$ and $y(t)$; and parameterization of only $u(t)$.

Chapter 3

Results and Discussion

The main findings are presented in Part II of this dissertation. The following sections gives an overview of scientific papers, discussion and future work.

3.1 Overview of Scientific Papers

In this section, the accomplishments of the research work will be summarized. There are four research publications, three of them have already been published and the fourth one has been peer-reviewed and accepted to be published in the International Journal of Modeling and Optimization Vol. 6, No. 5. October 2016. One of the main focuses in this work is to only use free software tools. Throughout the research, Modelica as well as Modeling tools such as JModelica.org and the scripting language Python has been used.

3.1.1 Publication A - Modelica models in linear analysis

The main focus of the Modelica simulation environments is the model simulation; see Figure. 3.1 for the general execution flow of Modelica models. A flat model is usually a system of high index differential algebraic equations. The Modelica optimizer implements an index reduction. Then lower index DAEs are compiled into a C code to be used in model simulations. However, the model simulation is not the sole objective of the modeling process. Some of the other objectives are linear system analysis, optimal controller design, parameter estimation, etc. Therefore it may be useful to extract flat Modelica models into a scripting tool such as Python — this corresponds to the point (2) in Figure. 3.1. The disadvantage of extracting the model from point (1) (in Figure. 3.1) is that flat models are neither index reduced nor sorted. However, it is always a possibility to implement index reduction, sorting algorithms etc. on flat models in Python. If there is a mechanism to access the model from the point (4) or (5), then it is the best option because this avoids re-programming ("reinventing the wheel") already implemented index reduction, sorting and other algorithms in Modelica compilers. The key idea emphasized here is that the availability of Modelica models for general use is much more important in practice than model simulations. The JModelica.org platform is a Python-based Modelica simulation environment which provides the possibility of importing flattened Modelica models into Python via a JModelica.org-CasADi interface as symbolic DAEs. Though this is a positive development, it has some limitations. For example, Modelica models containing the function `Modelica.ComplexMath.abs()` is not supported. The Python control systems library (`python-control` package) can be used for linear analysis which tries to follow the functionality within MATLAB's control system toolbox. The article given in Appendix A discusses an industrial case study which is considered to demonstrate the idea.

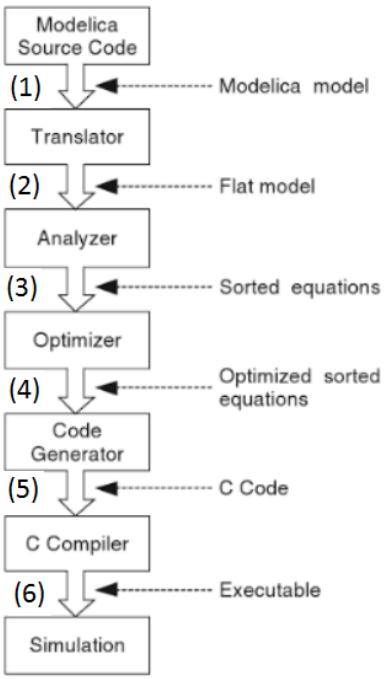


Figure 3.1: Implementation and execution of Modelica models (taken from the page 94 of [48]).

3.1.2 Publication B - Structural Observability Analysis

Consider a flat Modelica model referring to the Fig. 3.1. After reducing the index, we have a zero index system — i.e. a system of ordinary differential equations. The index reduction process may be automated in Python using the Pantelides algorithm. Now, the state observability analysis can be done for reduced index systems.

Often, the state variables are not independent of each other. A finite subset of state variables may contain some useful information about some other subset of the state variables or even a complete description about the entire state space. Especially for large-scale complex dynamic systems, it is advisable to make maximum use of structural observability analysis prior to implementing, for example, state estimators. Structural observability analysis can be automated using free computer-aided tools. There are several well-developed Python packages — `networkx`, `pygraphviz`, `pydot`, etc. — for complex network analysis and visualization. These packages can be used in structural observability analysis. By importing Modelica models into Python and calculating necessary Jacobian matrices symbolically using the `casadi` package, it is possible to construct the system digraph, and thereby analyze the system for structural observability. See the article given in Appendix B for the implementation — the method is demonstrated with a Modelica model created for the copper leaching process.

3.1.3 Publication C - Parameter and State Estimation

Publication C discusses the topics related to automating parameter, disturbance and state estimation analysis of large-scale complex nonlinear dynamic systems using free programming tools. Large-scale complex systems should be analyzed for structural observability before implementing any state estimator. The structural observability analysis can be automated using Modelica and Python. As a result of structural observability analysis, the system may be decomposed into subsystems, some of which may be observable while some may not. The state estimation process is carried out on observable subsystems and the optimum number of additional measurements needed to make unobservable subsystems observable are prescribed. In this paper, an

industrial case study is considered: The copper leaching process. It is shown how to implement various state estimators in Python and how to estimate parameters and disturbances as well as state variables.

3.1.4 Publication D - State Estimation and Optimal Control

Publication D uses the results from both Publications B and C. One of the objectives of the research presented in this paper is to better stabilize both $CuSO_4$ and H_2SO_4 within the copper leaching process, in particular within the electrowinning section. Since the electrowinning section is the only observable part, an optimal control strategy is implemented for that section. An optimal control algorithm is dependent on information about the system's state to calculate the optimal control trajectory. Since the complete state-disturbances are not available, they are estimated instead and used in the the optimal control algorithm. See the article given in Appendix D for the Python implementation.

3.2 Discussion, Conclusion and Future Work

It is beneficial to make Modelica models available for general usages within scripting languages such as Python as it enables more extensive system analysis and synthesis. Currently, a few interfaces capable of doing this exists — e.g., JModelica.org-CasADi. The goal is to combine the modeling power of Modelica and the scripting power in Python (or a similar tool) for technical computing. In order to achieve this, it is necessary to develop software tools which (1) optimize Modelica models (index reduction, BLT transformation, etc.) and (2) represent optimized flat Modelica models (index reduced and sorted ODE systems) as functions or methods within a given programming language. Consider the example given below, where the Modelica model `fModelica.mo` may be converted to a Python method `f`:[1]

```
model fModelica
    parameter Real a0;
    Real x0;
    Real x1;
    input Real u0;
equation
    a0*der(x0) = x0 + x1 + u0;
    a1*der(x1) = -x0;
end fModelica;

def f(dxdt,t,x,u,p):
    res0 = p[0]*dxdt[0] - (x[0] + x[1] + u[0])
    res1 = p[1]*dxdt[1] + x[0]
    res = [res0,res1]
    return res
```

Now `f` is accessible for any control engineering application. It is sometimes preferable that `f` is a symbolic function. The JModelica.org-CasADi interface is a promising development, where `f` provides a symbolic flat copy of a given Modelica model (without index reduction). Also, Dymola models can be imported to Simulink as s-functions. Although this is a positive development, Dymola and Simulink are commercial tools. The FMI (Functional Mock-up Interface) standard provides support for model exchange (and co-simulation). A tool which supports FMI model export can encode dynamic models into two files: An XML file (model variable information) and a C-file (with optimized dynamic equations). Since we are interested in free, efficient and powerful tools related to large-scale system modeling and applications, the Modelica-Python (or Modelica-Octave or similar) combination is emphasized. JModelica.org comply with both FMI import (pyFMI) and export (with some limitations). Therefore, in principle, JModelica.org can import any Modelica model and map it into a compiled C-code. The C-code may then be used in simulation and optimization problems (with the use of the XML file). There are numerous different, free scripting languages that can be used in systems and control applications;

[1] Taken from `http://book.xogeny.com`.

Python, C and Octave are some possible alternatives. Which language that should be selected depends on many factors. Python has several features which makes it a suitable tool for systems and control engineering applications. One of them is that there are many Python packages which provide functionality in linear algebra (NumPy), nonlinear optimization (SciPy), DAEs integration (Assimulo), graph-theoretic analysis (NetworkX), plotting (matplotlib), graph layout and visualization (PyGraphviz), etc. This is one of the reasons why Python has become increasingly popular in academia. It is suggested to develop some tool to extract Modelica models from the point (5) (referring to Figure 3.1) to Python as Python methods or objects in future work.

The initial copper leaching process model [9] contains 39 state variables. In the extension of the model some additional state variables are included. A few output measurements are available. These measurements gives enough information to reconstruct the system's state, but not enough to estimate all the parameters and the disturbances. It is possible to estimate the minimum number of additional output measurements that should be added to make the parameters and disturbances observable. However, the addition of extra sensors should be practically viable for this to be a realistic option. Several important conclusions are made referring to Figure 3.2. In Section 2.2 it is explained that in order to estimate the parameters, we should have at least as many outputs as number of parameters. This is not the case in Figure 3.2. Augmenting parameters as $\dot{p}_i = \varepsilon_{p_i}$ (ε_{p_i} is a white noise) could be too strict in the structural sense; we may relax it to $\dot{p}_i = -\beta_i p_i + \varepsilon_{p_i}$, where $\beta_i > 0$. It can be seen from the digraph that by augmenting all the parameters via $\dot{p}_i = -\beta_i p_i + \varepsilon_{p_i}$, we can achieve the structural observability. In reality, parameters do change over time, and therefore it is not unrealistic to have $\dot{p}_i = -\beta_i p_i + \varepsilon_{p_i}$. We can do the same for disturbances. By merely tweaking parameter and disturbance augmentation models, it is possible to reach the observability. It is recommended that augmentation models always should be defined such that they add buds to the digraph — there are many possible ways to augment disturbances [49] and it is a matter of picking the right ones in the structural sense.
In some situations, simplified/reduced models for large-scale systems are needed. Loosely speaking, the idea is to have a sufficiently large and complex model to be able to capture the necessary system dynamics, but at the same time the model should be small and simple enough to be fit for the purpose of modeling; there is a tradeoff between the two objectives. The dimensions of the state space determines if the model is large or small. Model simplifications are done during all possible phases of the modeling process (i.e., from creating the block diagrams to creating the state space model) when having a simpler model is of interest. There are basically three concerns in the model simplification [50]: (1) Making assumptions such as constant fluid density everywhere; (2) if there are different time scales (i.e., some variables have faster dynamics while others are slower), then the focus should be on the time scale of interest;[2] (3) and lumping variables, parameters, and subsystems. Theoretically rigorous discussions of the simplification of large-scale systems can be found in [51][52]. In qualitative terms, the key concern of model reduction is to obtain a simpler and smaller model for which the error between the old and new systems' outputs lie within a given tolerance. It is proposed to perform a detailed model reduction analysis on the copper leaching process model in future work.

This work presents a new perspective on the model simplification; model simplification in the view of state-parameter-disturbance estimation and controllability.[3] Maximum use of the structural system theory is made [54]. Closely studying Figure 3.2, it can be seen that all chemical compositions are structurally observable if parameter-disturbance estimation is not in our interest. The digraph in Figure 3.2 may be reduced by lumping eps_1, eps_2 and eps_2 into one

[2]The ODE system $\frac{d}{dt}x_1 = f_1(t,x_1,x_2,u)$ and $\varepsilon\frac{d}{dt}x_2 = f_2(t,x_1,x_2,u)$ approaches a DAE system $\frac{d}{dt}x_1 = f_1(t,x_1,x_2,u)$ and $0 = f_2(t,x_1,x_2,u)$ when $\varepsilon \to 0$. The latter model is of a lower dimensions than the former. By this manner, we remove the dynamics of x_2.

[3][53] offers a discussion on model reduction with regards to model simulation.

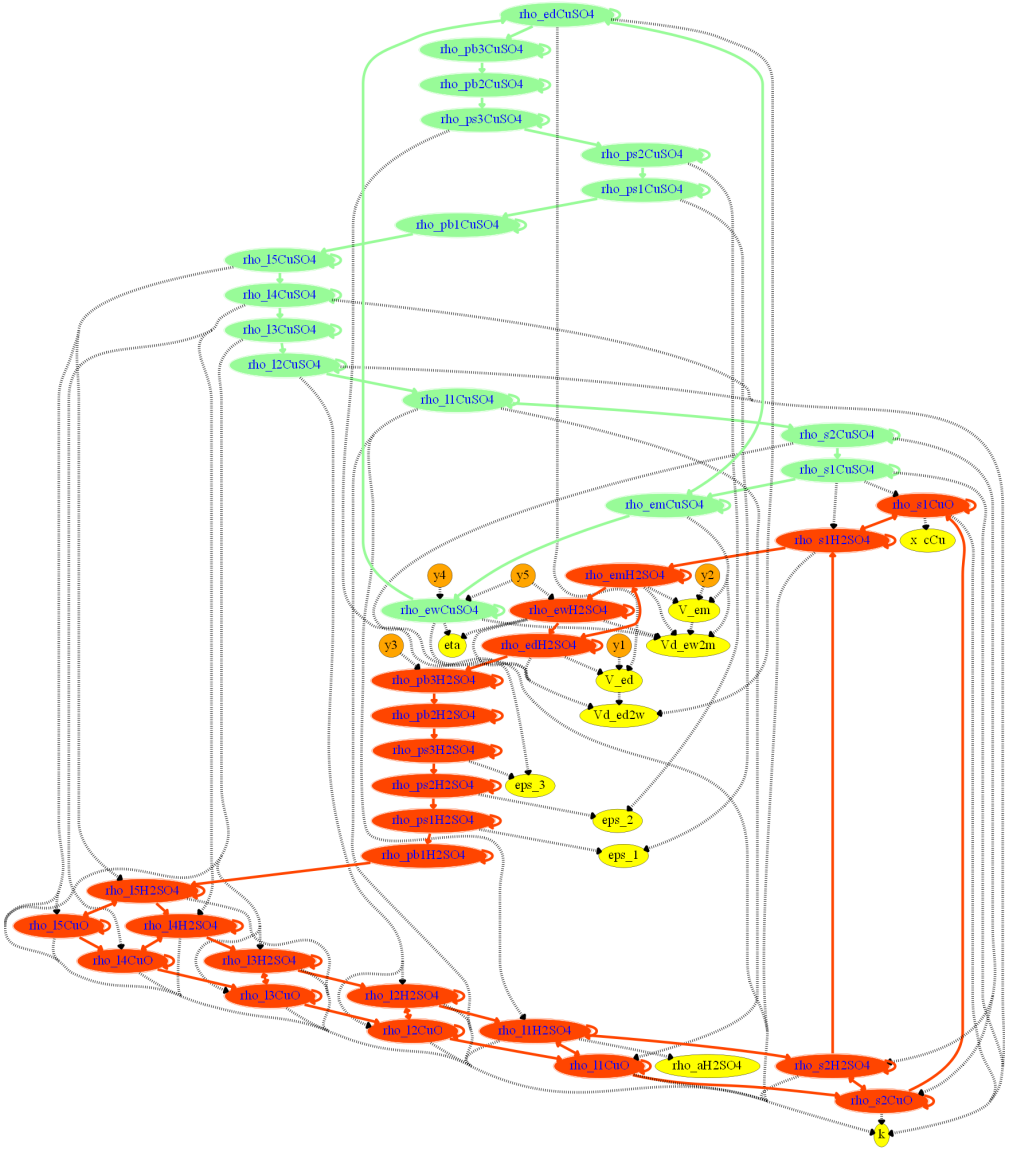


Figure 3.2: A digraph given in Appendix B (Figure 9).

parameter. The parameters eps_1, eps_2 and eps_2 appear in a symmetrical manner in the digraph. Symmetries in the structure of the digraph indicates that it may be possible to reduce the model in the structural sense. In Figure 3.3 an area from Figure 3.2 is enlarged, showing a symmetry among three cementation tanks (referring to Figure 1.3) connected in series (the relevant equations are (1.35) to (1.43)). All three cementation tanks are lumped and considered as one unit. Consequently, lumped quantities $\bar{\rho}_{ps,\mathrm{CuSO_4}}$, $\bar{\rho}_{ps,\mathrm{H_2SO_4}}$ and $\bar{\varepsilon}_{ps}$ are defined:

$$\bar{\rho}_{ps,\mathrm{CuSO_4}} = \frac{\rho^{(1)}_{ps,\mathrm{CuSO_4}} V^{(1)}_{ps} \varepsilon_1 + \rho^{(2)}_{ps,\mathrm{CuSO_4}} V^{(2)}_{ps} \varepsilon_2 + \rho^{(3)}_{ps,\mathrm{CuSO_4}} V^{(3)}_{ps} \varepsilon_3}{V^{(1)}_{ps} \varepsilon_1 + V^{(2)}_{ps} \varepsilon_2 + V^{(3)}_{ps} \varepsilon_3}$$

$$\bar{\rho}_{ps,\mathrm{H_2SO_4}} = \frac{\rho^{(1)}_{ps,\mathrm{H_2SO_4}} V^{(1)}_{ps} \varepsilon_1 + \rho^{(2)}_{ps,\mathrm{H_2SO_4}} V^{(2)}_{ps} \varepsilon_2 + \rho^{(3)}_{ps,\mathrm{H_2SO_4}} V^{(3)}_{ps} \varepsilon_3}{V^{(1)}_{ps} \varepsilon_1 + V^{(2)}_{ps} \varepsilon_2 + V^{(3)}_{ps} \varepsilon_3}$$

$$\bar{\varepsilon}_{ps} = \frac{V^{(1)}_{ps} \varepsilon_1 + V^{(2)}_{ps} \varepsilon_2 + V^{(3)}_{ps} \varepsilon_3}{V^{(1)}_{ps} + V^{(2)}_{ps} + V^{(3)}_{ps}}$$

$$\dot{V}_{pb^{(1)}2ps^{(3)}} = \dot{V}_{pb^{(1)}2ps^{(1)}} = \dot{V}_{ps^{(1)}2ps^{(2)}} = \dot{V}_{ps^{(2)}2ps^{(3)}}$$

Assuming $\bar{\rho}_{ps,\mathrm{CuSO_4}}$, $\bar{\rho}_{ps,\mathrm{H_2SO_4}}$ and $\bar{\varepsilon}_{ps}$ are uniformly distributed in all cementation tanks, we get the simplified equations (3.1) to (3.3). These equations gives approximately first order ordinary differential equations with time-varying time delays. We may relax the time-varying delay by a constant delay. It is also possible to lump leaching as well as slurrification tanks in a similar fashion.

$$\frac{d}{dt}\bar{\rho}_{ps,\mathrm{CuSO_4}} = \frac{\left(\rho^{(1)}_{pb,\mathrm{CuSO_4}} - \bar{\rho}_{ps,\mathrm{CuSO_4}}\right)\cdot \dot{V}_{pb^{(1)}2ps^{(3)}}}{\left(V^{(1)}_{ps} + V^{(2)}_{ps} + V^{(3)}_{ps}\right)\cdot \bar{\varepsilon}_{ps}} \tag{3.1}$$

$$\frac{d}{dt}\bar{\rho}_{ps,\mathrm{H_2SO_4}} = \frac{\left(\rho^{(1)}_{pb,\mathrm{H_2SO_4}} - \bar{\rho}_{ps,\mathrm{H_2SO_4}}\right)\cdot \dot{V}_{pb^{(1)}2ps^{(3)}}}{\left(V^{(1)}_{ps} + V^{(2)}_{ps} + V^{(3)}_{ps}\right)\cdot \bar{\varepsilon}_{ps}} \tag{3.2}$$

$$\frac{d}{dt}\bar{\varepsilon}_{ps} = 0 \tag{3.3}$$

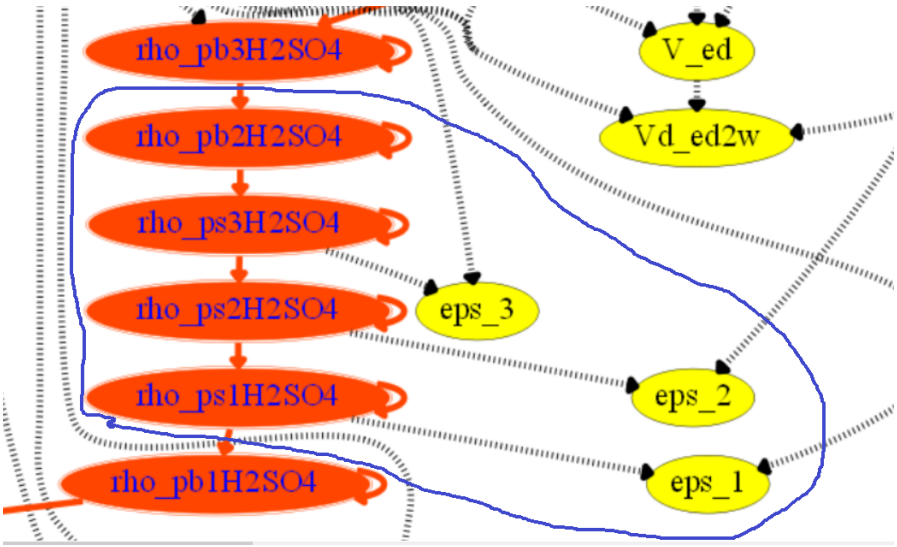


Figure 3.3: Symmetric structural distribution of eps_1, eps_2 and eps_2. This figure is a part of Figure 3.2.

As can be seen in Figure 3.2 there are no spanning cacti covering all the nodes in the digraph. Consider the node sets $S_1 = \{y_1, V_{ed}, \dot{V}_{ed2w}\}$, $S_2 = \{y_2, V_{em}, \dot{V}_{ew2m}\}$ and S_3, where S_3 contains the nodes which are not in S_1 and S_2. The subgraphs corresponding to S_1 and S_2 are stems; none of the stems have any outgoing edge starting any node belong to them. Hence, the state variables in S_1 and S_2 can be estimated by formulating two independent estimation problems. The same conclusion can be drawn easily by inspecting the two equations (1.56) and(1.63). However, isolating completely independently solvable subproblems — e.g., S_1 and S_2 — by manually inspecting the model equations are not trivial.[4] A more formal description is given as follows: Let S be a cactus (for which the root is an output node) and a subgraph ($S \subset G_o$) of a given digraph (for observability analysis) G_o. We can prove that if there exists at least one node v_j such that $v_j \notin S$ and the edges $v_i \to v_j \in S$ exists for any $v_i \in S$, then we cannot define an independent estimated problem for S.

The copper leaching process is affected by delay processes; estimation and control problems including time delays are not covered within this dissertation. It is recommended to implement

[4]It is possible to automate the process for example using Kalman Decomposition [24]. This decomposition convert variables with physical interpretations by new set of variables via some coordinate change.

an analysis including time delays in future work. However, the following discussion gives some background information which may be needed for controllability and observability. The article in Appendix D focus on controlling the chemical compositions in the electrowinning tanks and liquid volumes (or thereby, levels) in the dilution and the mixing tanks. $\dot{V}_a$ and $\dot{m}_c$ are used to influence the chemical compositions in the electrowinning sections; see equations (1.7)-(1.15). The changes in $\dot{V}_a$ and $\dot{m}_c$ have to propagate through several processes, adding significant time delays prior to affecting the chemical compositions in the third buffer tank. To keep the model simple, we propose the dynamic models (3.4) and (3.5) to incorporate time delays. The transfer function $e^{-s\tau}$ may be approximated by $\frac{1}{1+\tau s}$, where s is Laplace operator and $\tau > 0$. This simplifies $\bar{v}(t) = v(t-\tau)$ into $\tau \frac{d}{dt}\bar{v}(t) + \bar{v}(t) = v(t)$. Consequently, $\dot{V}_a(t-\tau_{\dot{V}_a}) \approx \tilde{\dot{V}}_a(t)$ and $\dot{m}_c(t-\tau_{\dot{m}_c}) \approx \tilde{\dot{m}}_c(t)$.

$$\tau_{\dot{V}_a}\frac{d}{dt}\tilde{\dot{V}}_a + \tilde{\dot{V}}_a = \dot{V}_a \tag{3.4}$$

$$\tau_{\dot{m}_c}\frac{d}{dt}\tilde{\dot{m}}_c + \tilde{\dot{m}}_c = \dot{m}_c \tag{3.5}$$

In the publication given in Appendix D, H_2SO_4 and $CuSO_4$ compositions in the third buffer tank are taken as control variables (neglecting time delays). In order to better comply with the reality, $\rho^{(3)}_{pb,H_2SO_4} \triangleq \tilde{\dot{V}}_a$ and $\rho^{(3)}_{pb,CuSO_4} \triangleq \tilde{\dot{m}}_c$ are taken. Also, $\dot{V}_a$ and $\dot{m}_c$ are expressed in terms of H_2SO_4 and $CuSO_4$ equivalents: I.e.,

$$\dot{V}_a \triangleq \rho^{\dot{V}_a}_{pb,H_2SO_4}$$
and
$$\dot{m}_c \triangleq \rho^{\dot{m}_c}_{pb,CuSO_4}.$$

If the outputs are structurally controllable by $\tilde{\dot{V}}_a$ and $\tilde{\dot{m}}_c$, then it follows that they are also structurally controllable by $\dot{V}_a$ and $\dot{m}_c$. This fact is easy to prove. Also the above mentioned delay models do not affect structural observability. We can even consider higher order delay models with

$$e^{-s\tau} \approx \frac{1}{(1+\frac{\tau s}{n})^n}$$

for $n = 2,3,\ldots,\to\infty$, without affecting structural observability and controllability.

A tracking problem concerns finding an admissible control law such that the systems outputs always follows their set points, while (1) bounding the state variables within given limits and (2) mitigating or completely rejecting the effects from disturbances. More precisely saying, the asymptotic output tracking is defined as follows: There exists $u_{min} \le u \le u_{max}$ such that $y(t) - r(t) \to 0$ when $t \to \infty$ and $x_{min} \le x \le x_{max}$ for given bounded disturbance. Instead of $x_{min} \le x \le x_{max}$, it is desirable to stabilize x, of course within given bounds — i.e., the state stabilization problem. For the linearized model $d/dt\Delta x = A\Delta x + B\Delta u$ of a given nonlinear model $d/dtx = f(x,u)$, the (local) state stabilization can be achieved by the state feedback law $\Delta u = -K\Delta x$ when the pair(A,B) is stabilizable and the eigenvalues of $A-BK$ are strictly stable [22]. Disturbance rejection can be done in several ways. A control with integral action is one such strategy, for example Proportional–Integral (PI) controllers. When disturbances are known or are estimated, then the feedforward control actions can be used. For better set point tracking, both feedback and feedforward strategies are needed [23]. It is also advisable to include integral actions $\dot{z}_i = r_i - y_i$ (see equation 26 in Appendix D) for offset-free reference tracking [55]. In the essence, the control action is a combined result of the state feedback, disturbance feedforward

and integral action (equation 30 in Appendix D). Appendix D presents an implementation of a regulator with state feedback, disturbance feedforward of some disturbances and integral actions of all outputs (and rate control in u). See equations 29 and 30. A linearized model is used in the implementation. The system state and the disturbance vector $w^{(2)}$, which are included in the control action are not measured and therefore they are replaced by some estimates.

There are altogether 7 disturbance variables in $w^{(1)}$ and $w^{(2)}$; the influence of these are analyzed in the structural point of view. Since, $w^{(1)}$ known and $w^{(2)}$ can be estimated, these quantities are available for feedforward control. See Figure 3.4 for the digraph. Consider the stem $\dot{V}_{p2e} \rightarrow V_{ed} \rightarrow y_1$. The node V_{ed} is directly affected by $\dot{V}_{em2d}$, $\dot{V}_{ed2w}$ and $\dot{V}_{ed2m}$, hence $\dot{V}_{em2d}$, $\dot{V}_{ed2w}$ and $\dot{V}_{ed2m}$ are used in feedforward control for the $\dot{V}_{p2e} - y_1$ loop. In this case, there exists no state feedback which rejects $\dot{V}_{em2d}$, $\dot{V}_{ed2w}$ and $\dot{V}_{ed2m}$. The same apply for $\dot{V}_{e2s} - y_2$ loop. Also consider,

$$\boxed{\rho^{(3)}_{pb,\mathrm{CuSO_4}} \rightarrow \rho_{ed,\mathrm{CuSO_4}} \rightarrow \rho_{ew,\mathrm{CuSO_4}} \rightarrow y_3,}$$

where some of the disturbances influence y_3 directly, while others have a remote influence through other state-nodes. For example, $\dot{V}_{w2em}$ acts on y_3 in the following way: $\dot{V}_{w2em} \rightarrow \rho_{em,\mathrm{CuSO_4}} \rightarrow \rho_{ed,\mathrm{CuSO_4}} \rightarrow \rho_{ew,\mathrm{CuSO_4}} \rightarrow y_3$. Consequently, a state feedback can be defined so that $\dot{V}_{w2em}$ does not affect y_3. On the other hand, I is directly connected to the path

$$\boxed{\rho^{(3)}_{pb,\mathrm{CuSO_4}} \rightarrow \rho_{ed,\mathrm{CuSO_4}} \rightarrow \rho_{ew,\mathrm{CuSO_4}} \rightarrow y_3,}$$

making it impossible to reject it by a state feedback. More information on structural disturbance rejection strategies are available in [56][27]. Noninteracting control [57] though state feedback is another possible consideration. A noninteracting control strategy decouple control loops from each other. The controller implemented in Appendix D involves neither disturbance rejection nor noninteracting control. It is recommended to consider disturbance rejection and noninteracting control strategies via state feedback as a future work.

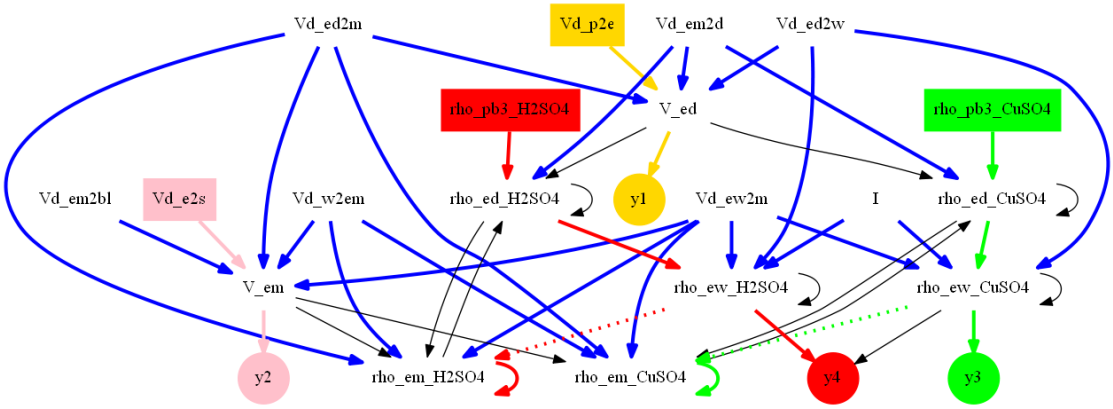


Figure 3.4: This digraph is used to analyze for disturbance rejection.

It is also suggested to validate the results against experimental data in future work. An online parameter-state-disturbance estimation process may be implemented for the electrowinning section to validate the results in Appendix C. In this paper, the parameter η is estimated with several of other disturbances. It is also possible to estimate V_{ew}, by augmenting it as $\dot{V}_{ew} = \beta_{ew} V_{ew} + \varepsilon_{ew}$. Also, anomalous data is inevitable. Hence, it is necessary to clean the data before using them. Usually, it is tedious to pick anomalous data points via manual inspections. Therefore, it is recommended to use available automatic data cleaning strategies in parallel to estimators.

Bibliography

[1] Anushka Perera. “Using CasADi for Optimization and Symbolic Linearization/Extraction of Causality Graphs of Modelica Models via JModelica. Org”. In: (2014).

[2] M Anushka S Perera et al. “Making modelica models available for analysis in python control systems library”. In: *Proceedings SIMS*. 2014.

[3] M. Anushka S. Perera, Bernt Lie and Carlos F. Pfeiffer. “Structural Observability Analysis of Large Scale Systems Using Modelica and Python”. In: *Modeling, Identification and Control* 36.1 (2015), pp. 53–65. DOI: 10.4173/mic.2015.1.4.

[4] M. Anushka S. Perera, Tor A. Hauge and Carlos F. Pfeiffer. “Parameter and State Estimation of Large-Scale Complex Systems Using Python Tools”. In: *Modeling, Identification and Control* 36.1 (2015), pp. 53–65. DOI: 10.4173/mic.2015.1.4.

[5] Anushka Perera, Tor A. Hauge and Carlos F. Pfeiffer. “State Estimation and Optimal Control an Industrial Copper Electrowinning Process”. In: *Control Engineering Practice* 36.1 (2015), pp. 53–65. DOI: 10.4173/mic.2015.1.4.

[6] E. O. Stensholt et al. “Development and practice of the Falconbridge chlorine leach process”. In: *CIM Bulletin* 9.1051 (2001), pp. 101–104.

[7] E.O. Stensholt, H. Zachariasen and J.H. Lund. “Falconbridge chlorine leach process”. In: *Transactions of The Institution of Mining and Metallurgy* (1986).

[8] E.O. Stensholt et al. “Recent improvements in the Falconbridge Nikkelverk nickel refinery, Extractive Metallurgy of Nickel and Cobalt”. In: *The Metallurgical Society* (1988), pp. 403–413.

[9] Bernt Lie and Tor Anders Hauge. “Modeling of an industrial copper leaching and electrowinning process, with validation against experimental data”. In: *Proceedings SIMS*. 2008, pp. 7–8.

[10] Tor Anders Hauge, Rune Løkling and Stanley Haga. “Past, Present and Future of Process Control at Xstrata Nikkelverk”. In: *Modeling, Identification and Control* 30.3 (2009), p. 157.

[11] N Habbache et al. “Leaching of copper oxide with different acid solutions”. In: *Chemical Engineering Journal* 152.2 (2009), pp. 503–508.

[12] Octave Levenspiel. *Chemical reaction engineering*. 3rd ed. John Wiley & Sons, 1999.

[13] Wassim M Haddad and Sergey G Nersesov. *Stability and control of large-scale dynamical systems: A Vector Dissipative Systems Approach*. Princeton University Press, 2011.

[14] Ching Tai Lin. “Structural controllability”. In: *Automatic Control, IEEE Transactions on* 19.3 (1974), pp. 201–208.

[15] Simon Haykin and Eric Moulines. “Large-scale dynamic systems”. In: *PROCEEDINGS-IEEE* 95.5 (2007), p. 849.

[16] François E Cellier and Ernesto Kofman. *Continuous system simulation*. Springer Science & Business Media, 2006.

[17] Constantinos C Pantelides. "The consistent initialization of differential-algebraic systems". In: *SIAM Journal on Scientific and Statistical Computing* 9.2 (1988), pp. 213–231.

[18] Linda R Petzold et al. "A description of DASSL: A differential/algebraic system solver". In: *Proc. IMACS World Congress*. 1982, pp. 430–432.

[19] Panos J Antsaklis and Anthony N Michel. *Linear systems*. Springer Science & Business Media, 2006.

[20] Younes Chahlaoui and Paul Van Dooren. "Estimating Gramians of large-scale time-varying systems". In: *IFAC Proceedings Volumes* 35.1 (2002), pp. 325–330.

[21] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[22] Hassan K Khalil and JW Grizzle. *Nonlinear systems*. Vol. 3. Prentice hall New Jersey, 1996.

[23] Jean-Jacques E Slotine, Weiping Li et al. *Applied nonlinear control*. Vol. 199. 1. Prentice-hall Englewood Cliffs, NJ, 1991.

[24] Alberto Isidori. *Nonlinear control systems*. Springer Science & Business Media, 1995.

[25] Matthew R James. "Controllability and Observability of Nonlinear Systems." In: (1987).

[26] Robert Hermann and Arthur J Krener. "Nonlinear controllability and observability". In: *IEEE Transactions on automatic control* 22.5 (1977), pp. 728–740.

[27] Kurt Johannes Reinschke. "Multivariable control: a graph theoretic approach". In: (1988).

[28] Yang-Yu Liu, Jean-Jacques Slotine and Albert-László Barabási. "Observability of complex systems". In: *Proceedings of the National Academy of Sciences* 110.7 (2013), pp. 2460–2465.

[29] Dragoslav D Siljak. *Decentralized control of complex systems*. Courier Corporation, 2011.

[30] J Deyst and C Price. "Conditions for asymptotic stability of the discrete minimum-variance linear estimator". In: *IEEE Transactions on Automatic Control* 13.6 (1968), pp. 702–705.

[31] Dan Simon. *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. Wiley-Interscience, 2006. ISBN: 0471708585.

[32] Kenneth R. Muske and Thomas F. Edgar. "Nonlinear Process Control". In: ed. by Michael A. Henson and Dale E. Seborg. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1997. Chap. Nonlinear State Estimation, pp. 311–370. ISBN: 0-13-625179-X. URL: `http://dl.acm.org/citation.cfm?id=248020.248026`.

[33] Andrew H Jazwinski. *Stochastic processes and filtering theory*. Courier Corporation, 2007.

[34] Gilbert Strang and Kai Borre. *Linear algebra, geodesy, and GPS*. Siam, 1997.

[35] Siew Chan, GC Goodwin and Kwai Sin. "Convergence properties of the Riccati difference equation in optimal filtering of nonstabilizable systems". In: *IEEE Transactions on Automatic Control* 29.2 (1984), pp. 110–118.

[36] Yongkyu Song and Jessy W Grizzle. "The extended Kalman filter as a local asymptotic observer for nonlinear discrete-time systems". In: *American Control Conference, 1992*. IEEE. 1992, pp. 3365–3369.

[37] James E Potter. *A matrix equation arising in statistical filter theory*. Vol. 270. National Aeronautics and Space Administration, 1965.

[38] Robert J Fitzgerald. "Divergence of the Kalman filter". In: *Automatic Control, IEEE Transactions on* 16.6 (1971), pp. 736–747.

[39] C De Souza, M Gevers and G Goodwin. “Riccati equations in optimal filtering of non-stabilizable systems having singular state transition matrices”. In: *IEEE Transactions on Automatic Control* 31.9 (1986), pp. 831–838.

[40] A Gelb. *Applied optimal estimation*. The M.I.T. press, 2001.

[41] Lennart Ljung. “Asymptotic behavior of the extended Kalman filter as a parameter estimator for linear systems”. In: *Automatic Control, IEEE Transactions on* 24.1 (1979), pp. 36–50.

[42] David F Bizup and Donald E Brown. “The over-extended Kalman filter-don’t use it!” In: *Proceedings of the Sixth International Conference of Information Fusion*. Vol. 1. 2003, pp. 40–46.

[43] Andreas Wächter and Lorenz T Biegler. “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming”. In: *Mathematical programming* 106.1 (2006), pp. 25–57.

[44] Lorenz T Biegler. *Nonlinear programming: concepts, algorithms, and applications to chemical processes*. Vol. 10. SIAM, 2010.

[45] John T Betts. *Practical methods for optimal control and estimation using nonlinear programming*. Vol. 19. Siam, 2010.

[46] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[47] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. Siam, 2008.

[48] Peter Fritzson. *Introduction to modeling and simulation of technical and physical systems with Modelica*. John Wiley & Sons, 2011.

[49] Urban Dominik Mäder. *Augmented models in estimation and control*. ETH, 2010.

[50] Lennart Ljung and Torkel Glad. *Modeling of dynamic systems*. PTR Prentice Hall Englewood Cliffs, 1994.

[51] Athanasios C Antoulas. “An overview of approximation methods for large-scale dynamical systems”. In: *Annual reviews in Control* 29.2 (2005), pp. 181–190.

[52] Athanasios C Antoulas. *Approximation of large-scale dynamical systems*. Vol. 6. Siam, 2005.

[53] James Anderson, Yo-Cheng Chang and Antonis Papachristodoulou. “Model decomposition and reduction tools for large-scale networks in systems biology”. In: *Automatica* 47.6 (2011), pp. 1165–1174.

[54] Dragoslav D Šiljak. *Large-scale dynamic systems: stability and structure*. Vol. 2. North Holland, 1978.

[55] Peter Colin Young and JC Willems. “An approach to the linear multivariable servomechanism problem†”. In: *International journal of control* 15.5 (1972), pp. 961–979.

[56] Prodromos Daoutidis and Costas Kravaris. “Structural evaluation of control configurations for multivariable nonlinear processes”. In: *Chemical Engineering Science* 47.5 (1992), pp. 1091–1107.

[57] Shean-lin Liu. “Noninteracting process control”. In: *Industrial & Engineering Chemistry Process Design and Development* 6.4 (1967), pp. 460–468.

Part II

PUBLISHED AND SUBMITTED PAPERS

Paper A

Making Modelica Models Available for Analysis in Python Control Systems Library

MAKING MODELICA MODELS AVAILABLE FOR ANALYSIS IN PYTHON CONTROL SYSTEMS LIBRARY

Anushka Perera, Carlos Pfeiffer and Bernt Lie*
Telemark University College
Kjølnes ring 56, P.O. Box 203
N-3901 Porsgrunn
Norway

Tor Anders Hauge
Glencore Nikkelverk
Kristiansand
Norway

ABSTRACT

Modelica-based simulation environments are primarily targeted on model simulation, therefore they generally lack support for advanced analysis and synthesis needed for general control systems design and particularly for optimal control problems (OCPs), although a Modelica language extension (Optimica) exists to support general optimization problems. On the other hand, MATLAB has a rich set of control analysis and synthesis tools based on linear models. Similarly, Python has increasing support for such tools e.g. the "Python Control System Library" developed in Caltech. In this paper, we consider the possibility of automating the process of extracting linear approximations of Modelica models, and exporting these models to a tool with good support for linear analysis and design. The cost of software is an important aspect in our development. Two widly used free Modelica tools are OpenModelica and JModelica.org. Python is also freely available, and is thus a suitable tool for analysis and design in combination with the "Python Control System Library" package. In this work we choose to use JModelica.org as the Modelica tool because of its better integration with Python and CasADi, a CAS (Computer Algebra System) tool that can be used to linearize Modelica models. The methods that we discuss can in principle also be adapted for other Modelica tools. In this paper we present methods for automatically extracting a linear approximation of a dynamic model encoded in Modelica, evaluated at a given operating point, and making this linear approximation available in Python. The developed methods are illustrated by linearizing the dynamic model of a four tank level system, and showing examples of analysis and design based on the linear model. The industrial application of these methods to the Copper production plant at Glencore Nikkelverk AS, Kristiansand, Norway, is also discussed as current work.

Keywords: Modelica, JModelica.org, Python, CasADi, Symbolic/Numeric Linearization, Linear Analysis, python-control

1 INTRODUCTION

Modelica is becoming a de facto standard for modeling of large-scale complex physical systems. Since Modelica is object-oriented, declarative (acausal), and equation-based, it allows to create reusable components and to built efficient reconfigurable component-models. A Modelica-based simulation environment (a Modelica tool) is needed to simulate Modelica models.[1] Modelica tools mainly focus on model simulation. However, the model sim-

*Corresponding author: Phone: +47 41807744 E-mail: bernt.lie@hit.no

[1] A complete list of Modelica tools is available at https://www.modelica.org/tools.

ulation is not the only objective of mathematical modeling. Among others optimal control problems (OCPs), control analysis and synthesis, and state estimations are several aspects that require dynamic systems modeling. Optimica [1] extends Modelica language specifications to handle OCPs and JModelica.org provides Optimica compilers. OpenModelica partially support Optimica extension at the moment.

In order to exploit Modelica, either a simulation environment should be equipped with necessary tools for model analysis (e.g. good enough scripting facilities and/or GUI options) or Modelica should be interfaced with other existing tools. Some comercial Modelica tools provide interfaces to integrate toexternal software, such as Dymola integrating with MATLAB/Simulink. However, these tools are very expensive. A free tool, the JModelica.org platform, integrates completely with Python through two core Python packages: `pymodelica` for compilers and `pyfmi`/`pyjmi` for model import.[2]

Since we are mainly interested in free software tools that easily interface with other tools, we selected JModelica.org. The JModelica.org platform has also an interface with CasADi [2] and hence, it is possible to make Modelica/Optimica models available as symbolic model objects in Python. The `casadi` Python package is used to linearize Modelica models symbolically/numerically (see [3] for a detailed description.) and then the system matrices may be used in linear system analysis and in algorithms, in particular using the `python-control` package.[3]

This paper demonstrates usefulness of interfacing Modelica with Python via CasADi. The method is explained with a simple example (a four tanks system). As a case study of a real process the Copper production plant [4] at Glencore Nikkelverk AS, Kristiansand, Norway is considered by showing how to design a LQR (Linear Quadratic Regulator) optimal state feedback controller using the `python-control` package.

2 STRUCTURE OF LINEARIZATION

2.1 MODELICA AND DAES

The execution of a Modelica model is started with a model flattening process that removes the hierarchical structure (i.e. expansion of inherited base classes, adding connector equations, etc.) of the Modelica model into a flat model [5]. A flattened model provides a set of acausal differential-algebraic-discrete equations, or so called hybrid DAEs form, which is given by

$$F\left(t,\dot{x},x,u,z,m,p\right)=\left[\begin{array}{c} F_1\left(t,\dot{x},x,u,z,m,p\right) \\ F_2\left(t,\dot{x},x,u,z,m,p\right) \\ \cdots \\ F_m\left(t,\dot{x},x,u,z,m,p\right) \end{array}\right]=0, \tag{1}$$

where x, u, z, m, p, and t are respectively, the dynamic state vector, the input vector, the algebraic state vector, the piece-wise constant vector, the parameter vector, and time. The keyword `input` is used to define input variables and `output`[4] for defining output variables. Output variables are also algebraic variables, hence they are included in z. An output vector, y, may be expressed as:

$$y=H\left(t,x,u,z,m,p\right). \tag{2}$$

For simplicity and notational convenience, m and p are neglected and thereby we have:

$$F\left(t,\dot{x},x,u,z\right)=\left[\begin{array}{c} F_1\left(t,\dot{x},x,u,z\right) \\ F_2\left(t,\dot{x},x,u,z\right) \\ \cdots \\ F_m\left(t,\dot{x},x,u,z\right) \end{array}\right]=0. \tag{3}$$

Where, $m=dim(x)+dim(y)$. A flattened Modelica model is not yet ready to be solved for $\dot{x}$ and z. A complicated set of manipulations are done on flattened models: sorting equations $(F_1,F_2,\ldots,F_m)$, index reduction, common subexpression elimination, etc. prior to solving the equation 3 [5][7].

2.2 CONVERSION TO EXPLICIT STATE SPACE FORM

Consider the DAEs in the equation 3. Converting DAEs into explicit ODEs may be required in many

[2] See the JModelica.org user guide available at `http://www.jmodelica.org`.

[3] `http://www.cds.caltech.edu/~murray/wiki/Control_Systems_Library_for_Python`.

[4] The variables which are prefixed with `input`/`output` keywords within the Modelica components at the highest hierarchy of a component-model are appeared as input/output variables after flattening.

applications or to use most standard ODE solvers. If $\frac{\partial F}{\partial [\dot{x},z]^T}$ is not singular (a necessary condition for implicit to explicit transformation), then $[\dot{x},z]^T$ can be written as continuous functions of t, x, and u.[5] On the other hand, if $\frac{\partial F}{\partial [\dot{x},z]^T}$ is singular, then implicit to explicit transformation may not be possible. Algebraic constraint among t, x, z and u can make $\frac{\partial F}{\partial [\dot{x},z]^T}$ singular and in such situations, the constraint equations are differentiated with respect to time, t.

Theorem 1 *The index of a DAE, $F(t,\dot{x},x,u,z)=0$, is the minimum number of times that all or part of the DAE must be differentiated with respect to t in order to determine $[\dot{x},\dot{z}]^T$ as a continuous function of x, z, u, and t [8].*

The definition to the index of a system of DAEs is given in theorem 1. Higher index (i.e. index > 1) problems may be reduced into at most index 1 problems systematically using the Pantelides algorithm [9]. For simplicity, the DAEs

$$f(t,\dot{x},x,u,z)=0 \tag{4}$$

and

$$g(t,x,u,z)=0 \tag{5}$$

are considered in the following discussion (a special case of the equation 3). Sometimes, it may be possible to express algebraic state variables ($\in z$), explicitly in terms of t, x and u and in such cases the index of the problem is said to be 0. By differentiating the equation 5, we get:

$$\frac{\partial g}{\partial t}+\frac{\partial g}{\partial x}\cdot\dot{x}+\frac{\partial g}{\partial u}\cdot\dot{u}+\frac{\partial g}{\partial z}\cdot\dot{z}=0. \tag{6}$$

If $\frac{\partial g}{\partial z}$ is not singular, then the equation 6 is used to find $\dot{z}$ and hence, the initial problem (equations 4 and 5) is said to be an index 1 problem and equations 4 and 6 gives an index 0 problem. The equation 4, in the general case, gives implicit ODEs, however often they appear as explicit ODEs (i.e. $\dot{x}=f(t,x,u,z)$). If $\frac{\partial g}{\partial z}$ is singular, it means there are algebraic dependencies among t, x, and u. In this case the algebraic constraints in equation 6 are differentiated once more and if this gives a possibility to find $\dot{z}$, then the initial system of DAEs is an index 2 problem. Constraint equations are differentiated, as many times as the index of the initial problem, until an index 0 problem is obtained. Note that a reduced index 0 (or 1) problem may not necessarily give the solution to the initial high index problem, unless consistent initial conditions are given [10].
After reducing the index and BLT sorting[6], we have a causal system of DAEs,

$$\tilde{f}\left(t,\dot{\tilde{x}},\tilde{x},\tilde{u},\tilde{z}\right)=0, \tag{7}$$

and

$$\tilde{g}(t,\tilde{x},\tilde{u},\tilde{z})=0 \tag{8}$$

with index 1. $\dot{\tilde{x}}$ is the new dynamic state vector of the reduced problem and $\tilde{u}=\left[u,\frac{du}{dt},\frac{d^2u}{dt^2},\ldots\right]$. The index reduction process may result in adding additional variables and those variables are stacked in $\tilde{z}$. For examples, the dummy derivatives, the state variables which has become algebraic, etc [11]. As $\frac{\partial \tilde{g}}{\partial \tilde{z}}$ is not singular and thereby, it is thus possible to explicitly find $\dot{\tilde{z}}$ (if needed) by:

$$\dot{\tilde{z}}=-\left(\frac{\partial \tilde{g}}{\partial \tilde{z}}\right)^{-1}\cdot\left[\frac{\partial \tilde{g}}{\partial \tilde{t}}+\frac{\partial \tilde{g}}{\partial \tilde{x}}\cdot\dot{\tilde{x}}+\frac{\partial \tilde{g}}{\partial \tilde{u}}\cdot\dot{\tilde{u}}\right]. \tag{9}$$

Consistent initialization gives the solution to the equations 7 and 9 identical to the initial higher index problem in the equations 4 and 5.

2.3 LINEARIZATION

Suppose that $\left(t_0,\dot{\tilde{x}}_0,\tilde{x}_0,\tilde{u}_0,\tilde{z}_0\right)$ exists such that $\tilde{f}\left(t_0,\dot{\tilde{x}}_0,\tilde{x}_0,\tilde{u}_0,\tilde{z}_0\right)=0$ and $\tilde{g}(t_0,\tilde{x}_0,\tilde{u}_0,\tilde{z}_0)=0$, then $\left(t_0,\dot{\tilde{x}}_0,\tilde{x}_0,\tilde{u}_0,\tilde{z}_0\right)$ is an operating point. In many cases, it is required to find the linear approximation for given nonlinear model with respect to an operating point. The linear approximation to equations 7 and 8 are given by:

$$\frac{\partial \tilde{f}}{\partial \tilde{t}}+\frac{\partial \tilde{f}}{\partial \dot{\tilde{x}}}\cdot\delta\dot{\tilde{X}}+\frac{\partial \tilde{f}}{\partial \tilde{x}}\cdot\delta\tilde{X}+\frac{\partial \tilde{f}}{\partial \tilde{u}}\cdot\delta\dot{\tilde{U}}+\frac{\partial \tilde{f}}{\partial \tilde{z}}\cdot\delta\dot{\tilde{Z}}=0 \tag{10}$$

and

$$\frac{\partial \tilde{g}}{\partial \tilde{t}}+\frac{\partial \tilde{g}}{\partial \tilde{x}}\cdot\delta\tilde{X}+\frac{\partial \tilde{g}}{\partial \tilde{u}}\cdot\delta\dot{\tilde{U}}+\frac{\partial \tilde{g}}{\partial \tilde{z}}\cdot\delta\dot{\tilde{Z}}=0. \tag{11}$$

Jacobian matrices are evaluated at $\left(t_0,\dot{\tilde{x}}_0,\tilde{x}_0,\tilde{u}_0,\tilde{z}_0\right)$. $\frac{\partial \tilde{f}}{\partial \dot{\tilde{x}}}$ and $\frac{\partial \tilde{g}}{\partial \tilde{z}}$ are not singular and as a result equations 10 and 11 can be transformed into a state space form.
The usual procedure to obtain numerical Jacobian

[5] The implicit function theorem.

[6] BLT stands for Block-Lower-Triangular.

matrices is to use finite difference methods. For example, a finite difference approximation to $\frac{\partial \tilde{f}}{\partial \dot{\tilde{x}}}$ using the central difference method is

$$\frac{\tilde{f}\left(t,\dot{\tilde{x}}+I_{dim(x)}\cdot h,\tilde{x},\tilde{u},\tilde{z}\right)-\tilde{f}\left(t,\dot{\tilde{x}}-I_{dim(x)}\cdot h,\tilde{x},\tilde{u},\tilde{z}\right)}{2\cdot h^2}. \tag{12}$$

Where h is a small-enough positive number and $I_{dim(x)}$ is a $dim(x)$-by-$dim(x)$ unit matrix. There are several drawbacks in finite difference methods: truncation errors, choosing h is harder, and the results depends on h. In order to avoid such problems, automatic/algorithmic differentiation (AD) techniques can be used, where derivatives are calculated as accurate as up to the working precision of a given computer. AD techniques are used to evaluate derivatives of functions defined by means of a high-level programming language such as Python/C++/etc.[7] The AD is implemented with the help of a computer algebra system (CAS) tool, which provides symbolic manipulations over mathematical expressions. A CAS tool is used to create symbolic variables, matrices, expressions, functions and do symbolic mathematical manipulations on them such as symbolic differentiation[8], integration, etc. There are many CAS tools available such as Maple, Mathematica, SymPy, CasADi, Maxima, etc.[9] A CAS tool may or many not support AD. For example the `sympy` python package doesn't support AD while Maple does. If $\tilde{f}$ and $\tilde{g}$ in equations 7 and 8 can be symbolically expressed using a CAS tool which support AD, then the Jacobian matrices in equations 10 and 11 can be evaluated efficiently using AD techniques.

2.4 JMODELICA.ORG OPTIONS

There are several ways of creating Modelica/Optimica model objects, so called model export, in JModelica.org: FMU, JMU, and FMUX.[10] FMUs are based on FMI (Functional Mock-up Interface) standards[11] and all others are JModelica.org platform specific. The `pymodelica` package contains compilers for compiling Modelica/Optimica models into FMUs, JMUs, and FMUXes. But FMU-export doesn't support Optimica. FMUXes are crucial here because in order to work with symbolic DAEs, FMUX model units should be used and the relevant compiler is `compile_fmux`. A compiled model is stuffed in a zip file (with the file extension '.fmux') and the modelDescription.xml file is contained in it. modelDescription.xml file gives a flat model description of Modelica/Optimica models. JMUs closely follow FMI standards. zip files of both FMUs/JMUs provide a compiled C-codes and binaries besides modelDescription.xml files while in FMUXes only the modelDescription.xml file is given. The model import (loading FMU/JMU/FMUX model objects into Python) may be done via two Python packages: PyFMI and PyJMI. PyFMI is for FMUs while PyJMI for JMUs/FMUXes.

CasADi is a symbolic framework for AD and non-linear optimization as well as it is a CAS tool. CasADi can import Modelica/Optimica models, where those models have been transformed into compatible XML-files (modelDescription.xml) [13][14] and generates symbolic DAEs/OCPs. the `parseFMI()` method which is defined within the CasADi class `SymbolicOCP` is used to import XML-based Modelica/Optimica models. See [15] for more details. CasADi integra-

[7] Consider a function $\bar{f}$ such that $\bar{y}=\bar{f}(\bar{x})$, where $\bar{x}=[\bar{x}_1,\bar{x}_2,\ldots,\bar{x}_n]$ and $\bar{y}=[\bar{y}_1,\bar{y}_2,\ldots,\bar{y}_m]$. $\bar{x}$ and $\bar{y}$ are the independent and the dependent variable vectors respectively. Often, it is possible to represent y_i-x_j relationships using elementary unary/binary operations (+, -, etc.) and elementary functions (sin, cos, etc.). Let $\bar{y}_1=\bar{x}_1\cdot e^{\bar{x}_1\cdot\bar{x}_2}$. $\bar{y}_1$ can be expressed in terms of basic functions and unary/binary operators using a set of intermediate variables ($\bar{z}_k$'s): $\bar{z}_0=\bar{x}_1$, $\bar{z}_{-1}=\bar{x}_2$, $\bar{z}_1=\bar{z}_0\cdot\bar{z}_{-1}$, $\bar{z}_2=e^{\bar{z}_1}$, $\bar{z}_3=\bar{z}_0\cdot\bar{z}_2$, $\bar{z}_4=\bar{z}_3$, and $\bar{y}_1=\bar{z}_4$. $\bar{z}_k$ can be written as

$$\bar{z}_k=\bar{f}_e^k(\bar{z}_i), \tag{13}$$

where $i<k$ and $\bar{f}_e^k$ contains elementary functions and operations. Now, for example $\frac{\partial\bar{y}_1}{\partial\bar{x}_1}=\frac{\partial\bar{z}_4}{\partial\bar{z}_0}=\frac{\partial\bar{f}_e^4}{\partial\bar{z}_0}$ is given, by applying chain-rule, by

$$\frac{\partial\bar{z}_4}{\partial\bar{z}_0}=\Sigma_{i=1}^{4-1}\frac{\bar{f}_e^4(\bar{z}_i)}{\partial\bar{z}_i}\cdot\frac{\partial\bar{z}_i}{\partial\bar{z}_0}. \tag{14}$$

$\frac{\bar{f}_e^4(\bar{z}_i)}{\partial\bar{z}_i}$ is known as $\bar{f}_e^4$ contains known elementary functions. In order to find $\frac{\partial\bar{z}_i}{\partial\bar{z}_0}$, the equation 14 is applied again and so on. The derivative evaluation may be done in one of two modes: forward and reverse. The method just mentioned above is the forward mode. For further details, refer [12].

[8] Note that symbolic differentiation is not AD.

[9] `http://www.autodiff.org/` gives a list of available AD tools.

[10] The latest JModelica.org version 1.14 has introduced a new model class using the compiler `transfer_optimization_problem`, which is available in `pyjmi` package. See the user guide for further details. In this paper JModelica.org version 1.12 is considered.

[11] `https://www.fmi-standard.org/`.

tion with JModelica.org [16] opens up a provision to use Modelica/Optimica models with complete flexibility within Python, and making it possible to exploit modeling power in Modelica as well as scripting power in Python.

3 A PYTHON IMPLEMENTATION WITH AN EXAMPLE

3.1 STRUCTURE OF PYTHON SCRIPT

A simple four-tank system is considered (taken from [17]). See Figure 1 for the schematic model of the system. The mathematical model is given by equations 15 - 20. The table 1 contains parameters. The Optimica model is stored in a text file named `TankSystems` with the file extension `.mop` (in this case, the file extension may have been used to be `.mo`). `TankSystems.mop` contains `TankSystems` package and this package contained two Modelica models: `FourTanks` for the dynamic model and `FourTanks_init` for the steady state model in order to find steady state. See appendix A for the Optimica code.

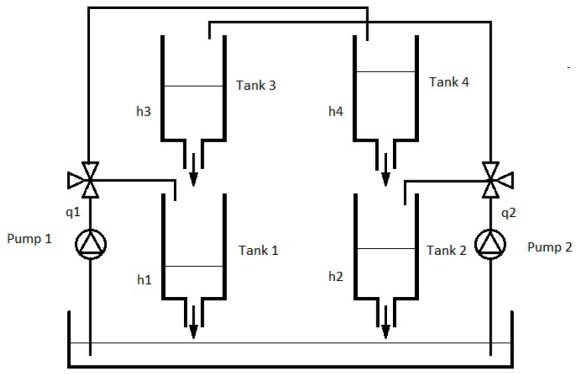


Figure 1: A schematic diagram for the four tank systems.

$$\frac{dh_1(t)}{dt} = -\frac{c_1 \cdot \sqrt{h_1(t)}}{A_1} + \frac{c_3 \cdot \sqrt{h_3(t)}}{A_1} + \frac{\gamma_1 \cdot q_1(t)}{A_1} \quad (15)$$

$$\frac{dh_2(t)}{dt} = -\frac{c_2 \cdot \sqrt{h_2(t)}}{A_2} + \frac{c_4 \cdot \sqrt{h_4(t)}}{A_2} + \frac{\gamma_2 \cdot q_2(t)}{A_2} \quad (16)$$

$$\frac{dh_3(t)}{dt} = -\frac{c_3 \cdot \sqrt{h_3(t)}}{A_3} + \frac{(1-\gamma_2) \cdot q_2(t)}{A_3} \quad (17)$$

$$\frac{dh_4(t)}{dt} = -\frac{c_4 \cdot \sqrt{h_4(t)}}{A_4} + \frac{(1-\gamma_1) \cdot q_1(t)}{A_4} \quad (18)$$

$$\frac{dq_2(t)}{dt} = -\frac{1}{\tau_2} \cdot q_2(t) + \frac{k_2}{\tau_2} \cdot v_2(t) \quad (19)$$

$$\frac{dq_1(t)}{dt} = -\frac{1}{\tau_1} \cdot q_1(t) + \frac{k_1}{\tau_1} \cdot v_1(t) \quad (20)$$

Variable	Value	Units
A_1	12.57	cm^2
A_2	12.57	cm^2
A_3	12.57	cm^2
A_4	12.57	cm^2
c_1	9.82	$c \cdot m^{5/2}/s$
c_2	5.76	$c \cdot m^{5/2}/s$
c_3	9.02	$c \cdot m^{5/2}/s$
c_4	8.71	$c \cdot m^{5/2}/s$
K_1	6.94	$c \cdot m^3/V$
K_2	8.72	$c \cdot m^3/V$
τ_1	6.15	s
τ_2	13.2	s

Table 1: Four-Tanks System Model Parameters.

The Python code used to compile the Modelica/Optimica Four Tanks is given below.

```
# Import compiler compile_fmux
from pymodelica import compile_fmux
# Compile Modelica/Optimica models
file_name = 'TankSystems.mop'
model_name = 'FourTanks'
compile_fmux(model_name,file_name)
# Note: name of the '.zip' file created is
\'FourTanks.fmux'
```

Now, the FMUX model object is imported as a `CasadiModel` object. See below:

```
from pyjmi import CasadiModel
casadiModelObject = CasadiModel('FourTanks.fmux')
# Get flat ocp representation
ocp = casadiModelObject.ocp
```

`ocp` gives a flat representation of Modelica/Optimica models based on the modelDescription.xml. `ocp.ode` and `ocp.alg` represent symbolic expressions for ordinary differential equations (ODEs) and algebraic equations respectively. Use Python commands `print ocp` and `help(ocp)` to get help. Now, the symbolic DAEs are available for general use in Python,[12] hence Modelica/Optimica models can be used in various

[12]For example, it is possible to implement Pantelides algorithm with symbolic DAEs.

algorithms and in analysis using CasADi functionalities, `numpy`[13], `matplotlib`[14], `scipy`[15] and `python-control` like Python packages. Use the following Python code to import CasADi and CasADi tools.

```
from casadi import *
from casadi.tools import *
```

If necessary, `ocp.makeExplicit()` method can be used to transform ODEs from implicit to explicit form.[16] Derivatives ($\in \dot{x}$), dynamic states ($\in x$), algebraic states ($\in z$), independent parameters ($\in p_i$), dependent parameters ($\in p_d$), free parameters ($\in p_f$), time (t), and control signals ($\in u$) are given by respectively `casadiModelObject.dx`, `ocp.x`, `ocp.z`, `ocp.pi`, `ocp.pd`, `ocp.pf`, `ocp.t`, and `ocp.u`. For example, `ocp.x[i]` gives x_{i+1} ($0 \leq i \leq dim(x)-1$). `ocp.x[i]` is in `variable` data type, and it has to be converted into `SX` data type before creating `SXFunction` instances. This is done by `ocp.x[i].var()` [3]. Then all the states variables (in SX type) are stuffed in a Python `list`. The same procedure is applied to other variables as well. Using `ocp.eliminateDependent()`, dependent parameters are eliminated. `ocp.ode` can be taken as a function of t, $\dot{x}$, x, z, and u and let it be $0 = f(t,\dot{x},x,u,z)$. Now, f is defined as an `SXFunction` class instance. Say, `ffun`. See below for the Python code to create it (see appendix B for the complete Python script):

```
# Define DAEs
f = ocp.ode
g = ocp.alg
# Create an SXFunction for f and g
ffun = SXFunction([t,vertcat(xDot),vertcat(x),\
vertcat(u),vertcat(z)],[f])
gfun= SXFunction([vertcat(x),vertcat(u),
vertcat(z)],[g])
ffun.init()
gfun.init()
```

Note that as explained in subsection 2.2, index reduction should be done on f and g, if the problem is higher index, to obtain lower index problems before creating `ffun` and `gfun`. Anyway, the four-tank system model has the index equal to 0. For an example, $\frac{\partial f}{\partial u}$ is given by `ffun.jac(2)`. See the result (by entering `ffun.jac(2)` in the command line) given below.

```
Matrix<SX>(
[[00,  00 ]
 [00,  00 ]
 [00,  00 ]
 [00,  00 ]
 [-1.12846,  00]
 [00,  -0.660606]]
)
```

Numerical Jacobian matrices are then found for given $(t_0,\dot{x}_0,x_0,u_0,z_0)$. See the code given below.

```
f_u_fun.setInput(t0,0)
f_u_fun.setInput(dx0,1)
f_u_fun.setInput(x0,2)
f_u_fun.setInput(u0,3)
f_u_fun.setInput(z0,4)
f_u_fun.evaluate()

f_u_num = f_u_fun.getOutput()
```

Operating points are usually choosen at steady states. A steady state, x_0 is calculated by: (1) compiling the static Modelica model TankSystems.FourTanks_Init into a JMU, (2) loading the JMU model, (3) setting the input vector u_0, and (4) finally, initializing the JMU model using `initialize()` method.[17] Hence, it is possible to find system matrices A, B, C, and D based on the Jacobian matrices just evaluated.
As the system matrices are available, the `python-control` package can be used in control analysis and synthesis. In order to import the `python-control` use:

```
import control
```

or

```
from control import *
```

As a summery to this section, the following points are made: (1) an Optimica package is created with two Modelica models (dynamic and static) in it, (2) use the static model to find the steady state using a JMU model object, (3) import the dynamic model as a CasadiModel object model and use `casadi` to linearization of symbolic DAEs (after reducing the index if needed), and (4) use the linearized model with the `python-control` package. See TankSystem.mop and TankSystem.py in the appendices A and B.

[13] `http://www.numpy.org/`.
[14] `http://matplotlib.org/`.
[15] `http://www.scipy.org/`.
[16] This is possible only if ODEs are linear w.r.t. $\dot{x}$.
[17] Initialization is done by formulating DAEs and equations given within `initial equation` clause in residual form and minimizing sum of square error using the `Ipopt` solver. See JModelica.org user guide.

4 INDUSTRIAL CASE STUDY

We consider the chlorine leaching and electro-winning process which is a part of the nickel refinery of Glencore Nikkelverk in Kristiansand, Norway. A mechanistic models is presented in [4] and it is a MIMO system with 3 inputs (u_1, u_2, u_3), 11 disturbances (w_1, w_2, ..., w_{11}), 3 outputs (y_1, y_2, y_3) and 39 states (x_1, x_2, ..., x_{39}). The process is in large-scale and it is complex (multi variable nature, non-linearities, etc.). Hence, Modelica and Optimica are ideal for the modeling and optimization. Also the process is a good candidate for model based control. In this section, what is explained in subsection 3.1 will be applied to the copper plant model.

The following demonstrations shows how to use the `python-control` tool to design a (infinite-horizon, continuous-time) LQR state feedback controller[18] for the linearized Copper plant model. The linearized model is given by

$$\dot{\delta x} = A \cdot \delta x + B \cdot \delta u, \tag{21}$$

and

$$\delta y = C \cdot \delta x + D \cdot \delta u, \tag{22}$$

where $\dot{\delta x}$, δx, δu, and δy are deviation variables with respect to a steady state point, x_0. Thus, $\dot{\delta x}_0 = 0$, $\delta x_0 = 0$, $\delta u_0 = 0$ and $\delta y_0 = 0$. The procedure to find A, B, C and D as well as x_0 is already given (see subsection 3.1). Use the following script to create a state space model (`sys`) object and optionally, the state space model may be transformed into transfer function form (`sys2`).

```
#Import python-control package
import control as ctrl
#Create state space model
sys = ctrl.ss(A,B,C,D)
#If needed, state space==>transfer function
sys2 = ctrl.ss2tf(sys)
```

The `ctrl.lqr()` method calculates the optimal feedback controller, $\delta u = -K \cdot \delta x$, such that minimizing the cost function J:

$$J = \int_0^\infty \left(\delta x^T \cdot Q \cdot \delta x + \delta u^T \cdot R \cdot \delta u + 2 \cdot \delta x^T \cdot N \cdot \delta u\right) \cdot dt. \tag{23}$$

[18] This paper mainly focus on demonstrating the idea of making Modelica models available in Python in general and in particular using the `python-control` package. Therefore, a detailed theoretical discussion about LQR state feedback controllers is not given here. For more details about LQR state feedback controllers refer [6].

Let, N is a zero matrix. K is the state feedback gain matrix and it is given by $K = R^{-1} \cdot B^T \cdot S$. S is found by solving the algebraic Riccati equation (ARE)

$$A^T \cdot S + S \cdot A^T - S \cdot B \cdot R^{-1} \cdot B^T \cdot S + Q = 0. \tag{24}$$

Use `K,S,E=ctrl.lqr(sys,Q,R,N)` finds K, S, and E. E gives Eigenvalues of the closed loop system. Now, the closed loop system is given by

$$\dot{\delta x} = (A - B \cdot K) \cdot \delta x, \tag{25}$$

and

$$\delta y = (C - D \cdot K) \cdot \delta x. \tag{26}$$

The closed loop system is simulated for a small perturbation in δx, say 0.01. Q and R are positive definite matrices and are used as the tunning parameters. A possibility is to set Q to be a unit matrix while R is a diagonal matrix and its elements are used in tunning. See the code given below and the results are given in figure 2.

```
Q = 1.0*np.eye(n_x,n_x)
R = 0.001*np.eye(n_u,n_u)
N = np.eye(n_x,n_u)
K,S,E=ctrl.lqr(sys,Q,R,N)

A1 = A - np.dot(B,K)
B1 = np.zeros((n_x,n_u))
C1 = C - np.dot(D,K)
D1 = np.zeros((n_y,n_u))
sys3 = ctrl.ss(A1,B1,C1,D1)

t0 = 0.
tf = 20.
X0 = 0.1*np.ones((n_x,1))
N = 500
T = np.linspace(t0,tf,N)

plt.figure(0)
plt.hold(False)
for i in range(n_y):
    Y,T=ctrl.step(sys3,T,x0,0,i)
    plt.plot(T,Y,label = 'y_{0}'.format(i+1))
    plt.hold(True)
    plt.xlabel('Time')
    plt.title('Outputs')
    plt.legend(loc='upper right', numpoints = 1)
    plt.grid(True)
    plt.show()
```

Note that both disturbances and control inputs are stacked in δu such that 12th, 6th and 5th elements are δu_1, δu_2, and δu_3 respectively.[19] When designing the LQR state feedback controller above, δu is considered to contain only control variables. However, this is not realistic. δu should have been decomposed as $\delta u := [\delta u, \delta w]^T$ and handled disturbances

[19] Check `print ocp.u`.

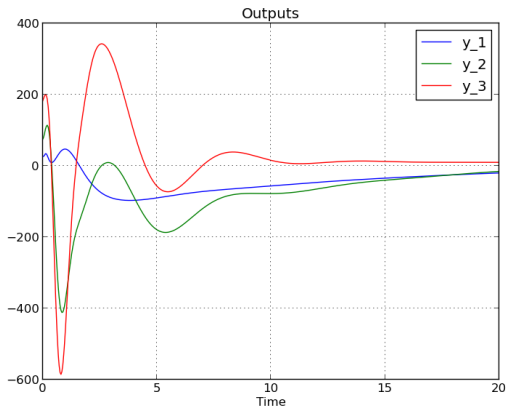


Figure 2: The $\delta y_1 - \delta y_2 - \delta y_3$ vs. t plot.

accordingly. Since, this paper is mainly concentrated on demonstrating the possibility of analyzing Modelica models in Python, in detail discussions on controller syntheis is not given here.

5 CONCLUSIONS

The features of Modelica language, in particular the notion of acausal modeling, have made it a powerful tool for modeling physical systems. However, the Modelica standards target primarily on model simulation, which is just one of the aspects of modeling. It is important that Modelica models are available for general use, but not just for the simulation. CasADi has an interface to Modelica/Optimica and JModelica.org is linked with CasADi. Therefore, Modelica-CasADi-JModelica.org combination provides a useful way to access Modelica/Optimica models in Python. Although, CasADi and JModelica.org has some limitations, they have provided a starting point. In this paper, it was explained the usefulness of interfacing Modelica models with Python. Special emphasis was given on the Python control system library as an up coming Python control tool, which could be an alternative to MATLAB control system toolbox.

Finally, couple of suggestions are made. CasADi-Modelica interface (via XML representation of Modelica models) may be further developed to support Modelica specification as much as possible. At the moment CasADi-Modelica interface is underdeveloped. The idea pointed out in this paper, in principal, for example may also be implemented within MATLAB environment. MathWorks provides the Simscape language and the Symbolic Math Toolbox (a CAS tool). The Simscape language is similar to Modelica. Therefore, Simscape-Symbolic Math Toolbox-MATLAB core may be designed to do the same as what Modelica-CasADi-Python does.

REFERENCES

[1] Åkesson J. *Optimica — An Extension of Modelica Supporting Dynamic Optimization*. 6th International Modelica Conference 2008.

[2] Andersson J. *A General-Purpose Software Framework for Dynamic Optimization*. PhD thesis. Arenberg Doctoral School, KU Leuven: Department of Electrical Engineering (ESAT/SCD) and Optimization in Engineering Center, Kasteelpark Arenberg 10, 3001-Heverlee, Belgium, 2013.

[3] Perera A. *Using CasADi for Optimization and Symbolic Linearization/Extraction of Causality Graphs of Modelica Models via JModelica.Org*. HiT Report No. 5. Porsgrunn: Telemark University College. `https://teora.hit.no/handle/2282/2175`. ISBN 978-82-7206-380-0. 2014.

[4] Lie B, Hauge TA. *Modeling of an industrial copper leaching and electrowinning process, with validation against experimental data*. Proceedings SIMS 2008, 49th Scandinavian Conference on Simulation and Modeling. Oslo University college. Oct 7-8, 2008.

[5] Fritzson P. *Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica*. Wiley, 2011.

[6] Åström KJ, Murray RM. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2008.

[7] Cellier FE, Kofman E. *Continuous System Simulation*. Springer, 2006.

[8] Brenan KE, Campbell SL, Petzold LR. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. SIAM, 1996.

[9] Pantelides CC. *The Consistent Initialization of Differential-Algebraic Systems*. SIAM Journal Scientific Statistical Computation, 1988.

[10] Bendtsen C., Thomsen PG. *Numerical Solution of Differential Algebraic Equations*. TECHNICAL Report No. 5. Porsgrunn: Department of Mathematical Modelling, Technical University of Denmark. http://www2.imm.dtu.dk/documents/ftp/tr99/tr08_99.pdf. 1999.

[11] Mattsson SE, Söderlind G. *Index Reduction in Differential-Algebraic Equations Using Dummy Derivatives*. SIAM Journal Scientific Statistical Computation, 1993.

[12] Griewank A, Walther A. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, 2008.

[13] Pop A, Fritzson P. *ModelicaXML: A Modelica XML Representation with Applications*. Proceedings 3rd International Modelica Conference 2003.

[14] Casella F, Donida F., Åkesson J. *An XML Representation of DAE Systems Obtained from Modelica Models*. Proceedings 7th International Modelica Conference 2009.

[15] Andersson J, Gillis J, Diehl M. *User Documentation for CasADi*, 2014.

[16] Andersson J, Åkesson J, Casella F, Diehl M. *Integration of CasADi and JModelica.org*. 8th International Modelica Conference 2011.

[17] Pfeiffer CF. *Modeling, Simulation and Control for an Experimental Four Tanks Systems using ScicosLab*. 52nd Scandinavian Simulation and Modeling Society Conference 2011.

APPENDIX

A TANKSYSETEMS.MOP

```
package TankSystems
  //=====Dynamic model=====
  model FourTanks
  //Parameters
  parameter Real h1_init = 7.0;
  parameter Real h2_init = 7.0;
  parameter Real h3_init = 8.3;
  parameter Real h4_init = 3.1;
  parameter Real q1_init = 1;
  parameter Real q2_init = 1;
  parameter Real c1 = 9.82;
  parameter Real c2 = 5.76;
  parameter Real c3 = 9.02;
  parameter Real c4 = 8.71;
  parameter Real A1 = 12.57;
  parameter Real A2 = 12.57;
  parameter Real A3 = 12.57;
  parameter Real A4 = 12.57;
  parameter Real gama1 = 0;
  parameter Real gama2 = 0;
  parameter Real tau1 = 6.15;
  parameter Real tau2 = 13.2;
  parameter Real k1 = 6.94;
  parameter Real k2 = 8.72;
  //Dynamic variables
  Real h1(start=h1_init,fixed=true);
  Real h2(start=h2_init,fixed=true);
  Real h3(start=h3_init,fixed=true);
  Real h4(start=h4_init,fixed=true);
  Real q1(start=q1_init,fixed=true);
  Real q2(start=q2_init,fixed=true);
  //Output variables
  Real z1 = sqrt(h1)^2;
  Real z2 = sqrt(h2)^2;
  //Note: if we use z1 = h1, & z2 = h2 instead, then
  //z1 & z2 would not be+9 considered as algebraic
  //variables when the model is imported to CasADi.
  //The reason is that in the modelDescription.xml
  //file, both z1 & h1  would have  the same
  //valueReference. The same applied for z2. However,
  //there could be a better way of handling this!
  //Input variables
  input Real v1;
  input Real v2;
  equation
  der(h1) = ((-c1 * sqrt(h1)) +
  c3 * sqrt(h3) + gama1 * q1) / A1;
  der(h2) = ((-c2 * sqrt(h2)) +
  c4 * sqrt(h4) + gama2 * q2) / A2;
  der(h3) = ((-c3 * sqrt(h3)) +
  (1 - gama2) * q2) / A3;
  der(h4) = ((-c4 * sqrt(h4)) +
  (1 - gama1) * q1) / A4;
  der(q1) = ((-q1) + k1 * v1) / tau1;
  der(q2) = ((-q2) + k2 * v2) / tau2;
  end FourTanks;
  //=====Dynamic model=====
  //=====Static model=====
  model FourTanks_Init
  extends FourTanks(h1(fixed=false),
  h2(fixed=false),  h3(fixed=false),
  h4(fixed=false),q1(fixed=false),
  q2(fixed=false));
  initial equation
  der(h1) = 0;
  der(h2) = 0;
  der(h3) = 0;
  der(h4) = 0;
```

```
  der(q1) = 0;
  der(q2) = 0;
  end FourTanks_Init;
  //=====Static model=====
end TankSystems;
```

B TANKSYSTEMS.PY

Let, $0 = f(\dot{x},x,u,z)$ and $0 = g(x,u,z)$. The linearized model is given by, $0 = \alpha \cdot \delta\dot{x} + \beta \cdot \delta x + \gamma \cdot \delta u + \delta \cdot \delta z$ and $0 = \zeta \cdot \delta x + \eta \cdot \delta u + \sigma \cdot \delta z$, where $\alpha = \frac{\partial f}{\partial \dot{x}}$, $\beta = \frac{\partial f}{\partial x}$,..., and $\sigma = \frac{\partial g}{\partial z}$. δy is taken as $\delta y = [\kappa_x\ \kappa_u\ \kappa_z] \cdot [\delta x\ \delta u\ \delta u]^T$, where κ_x, κ_u, and κ_z should be given.

```
#Importing necessary packages.
import numpy as np
import matplotlib.pyplot as plt
import control as ctrl
from casadi import *
from casadi.tools import *
from pymodelica import compile_jmu
from pymodelica import compile_fmux
from pyjmi import JMUModel
from pyjmi import CasadiModel

#Compiling (to a JMU)/loading
#steady state model.
jmu_init = compile_jmu \
("TankSystems.FourTanks_Init", \
"TankSystems.mop")
init_model = JMUModel(jmu_init)

#Set inputs
v1_0 = 1.
v2_0 = 2.
u_0 = [v1_0, v2_0]
u = ['v1','v2']
init_model.set(u,u_0)

#DAE initialization with Ipopt
init_result = init_model.initialize()

#Store steady state
h1_0 = init_result['h1'][0]
h2_0 = init_result['h2'][0]
h3_0 = init_result['h3'][0]
h4_0 = init_result['h4'][0]
q1_0 = init_result['q1'][0]
q2_0 = init_result['q2'][0]

#Compiling (to a FMUX)/loading dynamic model
fmux_name = compile_fmux \
("TankSystems.FourTanks", \
"TankSystems.mop")
model = CasadiModel(fmux_name)

#Get access to OCP
ocp = model.ocp

# Get differential state
n_x = len(ocp.x)
x = list()
for i in range(n_x):
    x.append(ocp.x[i].var())

#Get derivatives
xDot = list()
for i in range(n_x):
    xDot.append(model.dx[i])

# Get input
n_u = len(ocp.u)
u = list()
for i in range(n_u):
    u.append(ocp.u[i].var())

#Get algebraic states
n_z = len(ocp.z)
z = list()
for i in range(n_z):
    z.append(ocp.z[i].var())

#Eliminating dependent parameters
ocp.eliminateDependent()

#Define DAEs
f = ocp.ode
g = ocp.alg

#Create SXFunction instances for f and g
ffun = SXFunction([vertcat(xDot),vertcat(x), \
vertcat(u),vertcat(z)],[f])
gfun= SXFunction([vertcat(x),vertcat(u), \
vertcat(z)],[g])
ffun.init()
gfun.init()

#Define x0, u0, and z0
x0 = [h1_0,h2_0,h3_0,h4_0,q1_0,q2_0]
xDoto0 = [0.,0,0,0,0,0]
u0 = [v1_0,v2_0]
z0 = [h1_0,h2_0]

#Find symbolic/numeric Jacobian matrices
f_xDot = ffun.jac(0)
f_xDot_fun = SXFunction([vertcat(xDot),vertcat(x), \
vertcat(u),vertcat(z)],[f_xDot])
f_xDot_fun.init()

f_xDot_fun.setInput(xDoto0,0)
f_xDot_fun.setInput(x0,1)
f_xDot_fun.setInput(u0,2)
f_xDot_fun.setInput(z0,3)
f_xDot_fun.evaluate()

f_xDot_num = f_xDot_fun.getOutput()

alpha = np.array(f_xDot_num)
#
f_x = ffun.jac(1)
f_x_fun = SXFunction([vertcat(x),vertcat(u),\
vertcat(z)],[f_x])
f_x_fun.init()

f_x_fun.setInput(x0,0)
f_x_fun.setInput(u0,1)
f_x_fun.setInput(z0,2)
f_x_fun.evaluate()

f_x_num = f_x_fun.getOutput()

beta = np.array(f_x_num)
#
f_u = ffun.jac(2)
f_u_fun = SXFunction([vertcat(x),vertcat(u),\
vertcat(z)],[f_u])
```

```
f_u_fun.init()

f_u_fun.setInput(x0,0)
f_u_fun.setInput(u0,1)
f_u_fun.setInput(z0,2)
f_u_fun.evaluate()

f_u_num = f_u_fun.getOutput()

gamma = np.array(f_u_num)
#
f_z = ffun.jac(3)
f_z_fun = SXFunction([vertcat(x),vertcat(u), \
vertcat(z)],[f_z])
f_z_fun.init()

f_z_fun.setInput(x0,0)
f_z_fun.setInput(u0,1)
f_z_fun.setInput(z0,2)
f_z_fun.evaluate()

f_z_num = f_z_fun.getOutput()

delta = np.array(f_z_num)
#
g_x = gfun.jac(0)
g_x_fun = SXFunction([vertcat(x),vertcat(u), \
vertcat(z)],[g_x])
g_x_fun.init()

g_x_fun.setInput(x0,0)
g_x_fun.setInput(u0,1)
g_x_fun.setInput(z0,2)
g_x_fun.evaluate()

g_z_num = g_x_fun.getOutput()

zeta = np.array(g_z_num)
#
g_u = gfun.jac(1)
g_u_fun = SXFunction([vertcat(x),vertcat(u),\
vertcat(z)],[g_u])
g_u_fun.init()

g_u_fun.setInput(x0,0)
g_u_fun.setInput(u0,1)
g_u_fun.setInput(z0,2)
g_u_fun.evaluate()

g_u_num = g_u_fun.getOutput()

eta = np.array(g_u_num)
#
g_z = gfun.jac(2)
g_z_fun = SXFunction([vertcat(x),vertcat(u), \
vertcat(z)],[g_z])
g_z_fun.init()

g_z_fun.setInput(x0,0)
g_z_fun.setInput(u0,1)
g_z_fun.setInput(z0,2)
g_z_fun.evaluate()

g_z_num = g_z_fun.getOutput()

sigma = np.array(g_z_num)

# Define A, B, C, and D matrices
n_y = 2
kappa_x = np.eye(n_y,n_x)
kappa_u = np.zeros((n_y,n_u))
kappa_z = np.zeros((n_y,n_z))
if np.allclose(np.linalg.det(alpha),0.) != True:
    if np.allclose(np.linalg.det(sigma),0.) \
    != True:
        A = np.dot(np.linalg.inv(alpha),(-beta+\
        np.dot(delta,np.dot(np.linalg.inv(sigma),\
        zeta))))
        B = np.dot(np.linalg.inv(alpha),(-gamma+\
        np.dot(delta,np.dot(np.linalg.inv(sigma),\
        eta))))
        C = kappa_x - np.dot(kappa_z,\
        np.dot(np.linalg.inv(sigma),zeta))
        D = kappa_u - np.dot(kappa_z,\
        np.dot(np.linalg.inv(sigma),eta))
    #Use python-control
    #Create state space model object
        sys = ctrl.ss(A,B,C,D)
        print sys
    #State space to transfer function model object
        sys2 = ctrl.ss2tf(sys)
        print sys2
    # Simulate the system given input
        t0 = 0.
        tf = 120.
        N = 1000
        T = np.linspace(t0,tf,N)
        U = np.dot(np.diag([v1_0,v2_0]),np.ones((n_u,N)))
        t, yout, xout = ctrl.forced_response(sys,T,U,x0)

        plt.figure(0)
        plt.hold(False)
        for i in range(n_y):
            plt.plot(t,yout[i],'.',label = \
            'y_{0}'.format(i+1))
            plt.hold(True)
        plt.xlabel('Time')
        plt.title('Outputs')
        plt.legend(loc='upper right', numpoints = 1)
        plt.show()
        plt.figure(1)
        plt.hold(False)
        for i in range(n_x):
            plt.plot(t,xout[i],'.',label = \
            'x_{0}'.format(i+1))
            plt.hold(True)
        plt.xlabel('Time')
        plt.title('States')
        plt.legend(loc='upper right', numpoints = 1)
        plt.show()
        plt.figure(2)
        plt.hold(False)
        for i in range(n_u):
            plt.plot(t,U[i,:],'.',label = \
            'u_{0}'.format(i+1))
            plt.hold(True)
        plt.xlabel('Time')
        plt.title('Inputs')
        plt.legend(loc='upper right', numpoints = 1)
        plt.show()
    else:
        print 'sigma is singular. This case is \
        not considered.==>check Pantelides algorithm.'
else:
    print 'Alpha is singular. This case is \
    not considered.==>check Pantelides algorithm.'
```

Paper B

Structural Observability Analysis of Large Scale Systems Using Modelica and Python

Modeling, Identification and Control, Vol. 31, No. 1, 2010, pp. 1–9, ISSN 1890–1328

Structural Observability Analysis of Large Scale Systems Using Modelica and Python

M. Anushka S. Perera Bernt Lie Carlos Fernando Pfeiffer

Telemark University College, Kjølnes ring 56, P.O. Box 203, N-3901 Porsgrunn, Norway. E-mail: {from,Bernt.Lie,Carlos.Pfeiffer}@hit.no

Abstract

State observability of dynamic systems is a notion which determines how well the states can be inferred from input-output data. For small-scale systems, observability analysis can be done manually, while for large-scale systems an automated systematic approach is advantageous. Here we present an approach based on the concept of structural observability analysis, using graph theory. This approach can be automated and applied to large-scale, complex dynamic systems modeled using Modelica. Modelica models are imported into Python via the JModelica.org-CasADi interface, and the Python packages NetworkX (for graph-theoretic analysis) and PyGraphviz (for graph layout and visualization) are used to analyze the structural observability of the systems. The method is demonstrated with a Modelica model created for the Copper production plant at Glencore Nikkelverk, Kristiansand, Norway. The Copper plant model has 39 states, 11 disturbances and 5 uncertain parameters. The possibility of estimating disturbances and parameters in addition to estimating the states are also discussed from the graph-theory point of view. All the software tools used on the analysis are freely available.

Keywords: structural observability, Modelica, large-scale systems, CasADi, Python, graph-theory, JModelica.org, NetworkX, PyGraphviz

1 Introduction

Knowing the internal state of a dynamic system is important in many applications such as state feedback. However, measuring all state variables is usually impossible or impractical. What is more realistic is to estimate the state variables based on a finite set of measurements. The notion of observability characterizes whether a given set of measurements is adequate to estimate the state of the system. For linear time invariant systems, if the rank of the observability matrix is equal to the dimension of the state space, then the system is observable Simon (2006). For nonlinear dynamic systems diverse local observability definitions can be considered, for example using Lie derivatives Liu et al. (2012). In addition to analyzing observability for a given set of measurements, it would also be useful (especially for large-scale complex systems) to systematically find the minimum set of measurements which makes the system observable. By exploiting the model structure (algebraic dependencies among state and output variables), we can infer the minimum number of measurements and the possible choices to select from. Structural (or algebraic) observability is a fundamental property that provides a necessary condition for observability, and often it may also be sufficient for many systems Reinschke (1988), Liu et al. (2012). Structural observability analysis can be done using graph-theoretic techniques. Under some assumptions, unknown disturbances/parameters can be estimated (for example using Kalman filtering techniques Simon (2006)) by augmenting the system with them as state variables, making it necessary to check the observability of the augmented system. Struc-

doi:10.4173/mic.2010.1.1

tural observability analysis via graph theory offers a visual means to pinpoint measurements needed to estimate states/disturbances/parameters, or to detect which cannot be estimated at all in the augmented system.

JModelica.org is a Modelica-based simulation tool that makes possible to make Modelica models available as symbolic model objects in Python with the help of the JModelica.org-CasADi interface. The symbolic models can then be used in structural analysis using Networkx, Pygraphviz, and Python packages.[1]

This paper demonstrates the usefulness of using Python and relevant Python packages in analyzing structural observability for large-scale systems via graph theory. As a case study, the Copper production plant at Glencore Nikkelverk AS, Kristiansand, Norway is considered. The Copper plant model contains 39 states, 11 disturbances, and 5 uncertain parameters. The possibility of estimating disturbances and parameters additionally to the states will be discussed in a graph-theocratic point of view.

Section 2 gives a basic description about graph-theoretic concepts which are needed in the subsequent sections. Section 3 discusses structural observability in graph-theoretic point of view. A way of automating structural observability analysis in Python is given in Section 4. A demonstration of our development is done based on a real process and it is given in Section 5.

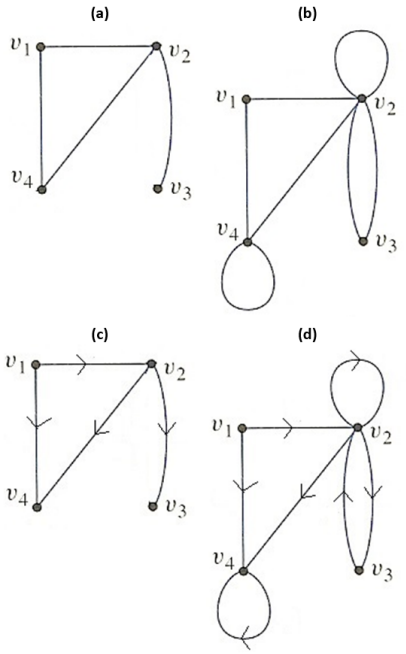


Figure 1: (a) Undirected and without self-cycles/loops and multiple edges. (b) Undirected and with self-cylcles/loops and multiple edges. (c) Directed and without self-cycles/loops and multiple edges. (d) A directed and with self-cycles/loops and multiple edges.

2 Graph-theoretic concepts

A graph G is denoted by $G = (V, E)$ where V is a set of nodes (or vertices) and E is a set of edges.[2] An edge connects two nodes v_i and v_j (where $v_i, v_j \in V$) and denoted by (v_i, v_j). The node v_i and v_j are incident to (v_i, v_j) and v_i and v_j are said to be adjacent nodes. A graph may be directed or undirected. In undirected graphs, edges are marked with directed lines while in undirected graphs it is not. For undirected graphs, (v_i, v_j) and (v_j, v_i) are identical. A directed/undirected graph may allow multiple edges among nodes. In short, digraph stands for directed graph. Let an edge $e_i = (v_i, v_j) \in E$, then v_i is the initial-vertex and v_j is the final-vertex. As a shorthand notation for a directed edge, let $(v_i, v_j) \equiv v_i \to v_j$.

A path is a sequence of edges connected one after another. A path has an initial node and a final node. Number of edges in a path is called the length of it. If a path contains no node appearing more than once, then it is a simple path. A path with initial and final nodes are identical, then it is a closed path. A cycle is a closed path with no node appearing more than once except the initial and the final nodes. Cycles with length 1 are self-cycles/loops. A set of cycles such that no two cycles have at least once common nodes are called a cycle family. A cycle family which covers all the nodes in a graph is called a spanning cycle family. v_i and v_j are strongly connected if paths from v_i to v_j and from v_j to v_i exist. A strongly connected component (SSC) is a sub-graph (of a directed graph) where each vertex in SCC is strongly connected with all other vertices in SCC. A digraph is said to be strongly connected if any two nodes in it are strongly connected. See figures 1 for several examples. Consider figure 1-d. $\{(v_1, v_2), (v_2, v_4), (v_4, v_4)\}$ is a path and its length is 3. $\{(v_1, v_2), (v_2, v_3)\}$ is a simple path. $\{(v_2, v_3), (v_3, v_2)\}$ is a closed path and it is a cycle as well. $\{(v_2, v_2)\}$ and $\{(v_4, v_4)\}$ are self-cycles. Nodes v_2 and v_3 are strongly connected. $\{\{(v_2, v_2)\}, \{(v_4, v_4)\}\}$ and $\{\{(v_4, v_4)\}, \{(v_2, v_3), (v_3, v_2)\}\}$ are two cycle families.

Instead of using the term "simple path", we use "ele-

[1] Alternatively, it is possible to create symbolic mathematical models using the Python package SymPy which is a CAS — CAS stands for Computer Algebra System — tool and then use Networkx and PyGrapViz. However, this method is more limited since it does not support the modeling power available in Modelica.

[2] Refer Bondy and Murty (2008) for graph theory.

mentary path" for digraphs. Similarly, instead of "simple cycle", "elementary cycle" is used. A stem is an elementary path. The initial and final nodes are respectively called the "root" and the "top" of the stem. A root is also called a driver nodes. A bud is an elementary cycle with an additional directed edge where its final node is one of the nodes in the cycle. This additional edge is called the distinguished edge of the bud. A directed cactus is made out of a stem and buds connected in a special way. The initial node of the distinguished edge of a bud is connected to any node in the stem except the top or it may be connected to a node of another bud. See figure 2. A cactus has a driver node which is the root of the stem in the catus. If there are vertex disjoint cacti covering all nodes in a given digraph, then they are called spanning cacti and cacti have more than one driver nodes.[3] See supplementary information to Liu et al. (2011)) for further details.

Consider a subset of edges M in an undirected graph where no two edges share common nodes. Nodes in M are said to be matched. M is a maximum matching if there exist no edge set M' such that $|M'| > |M|$.[4] M is perfect if each node in the graph is in M. See figure 3 and note that thick color lines are matched edges. A path with edges alternating between $E \setminus M$[5] and M is an M-alternating path. M-alternating path is M-alternating path with odd number of edges where the starting and the final edges are not in M. According figure 3, $\{(v_4, v_3), (v_3, v_8), (v_8, v_1), (v_1, v_7), (v_7, v_6)\}$ is an M-augmenting path. For digraphs, a matching is a subset of edges where no two edges share common nodes and a node is said to be matched if that node is the ending node of a matched edge Liu et al. (2011). See figure 4 where $M = \{(v_6, v_7), (v_1, v_8), (v_3, v_4)\}$ and v_7, v_8, and v_4 are matched nodes.

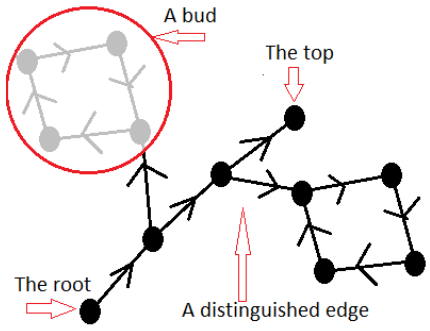


Figure 2: A cactus with two buds.

By formulating a bipartite graph (in short a bigraph)

[3]The plural of "cactus" is "cacti".
[4]$|M|$ is the cardinality of M.
[5]$E \setminus M = \{e | e \in E \ \& \ e \notin M\}$.

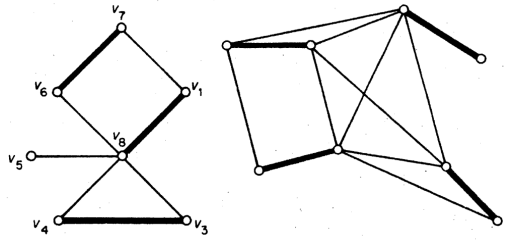


Figure 3: To the left - a maximum matching ($M = \{(v_4, v_3), (v_8, v_1), (v_6, v_7)\}$). To the right - a perfect matching.

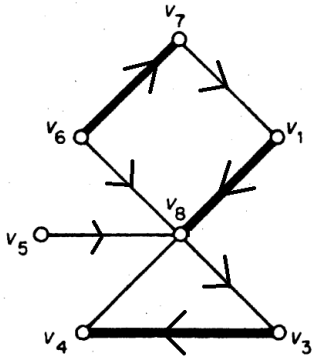


Figure 4: A matching for a digraph.

appropriately, a maximum matching for digraphs can be efficiently found. In a bipartite, there are two disjoint sets of nodes V_A and V_B such that edges only exist between V_A and V_B. See figure 5.

3 Structural observability

Consider the linear time invariant (LTI) state space model

$$\begin{aligned} \dot{x} &= A \cdot x + B \cdot u, \\ y &= C \cdot x + D \cdot u, \end{aligned} \tag{1}$$

where $A \in \mathbb{R}^{n_x \times n_x}$, $B \in \mathbb{R}^{n_x \times n_u}$, $C \in \mathbb{R}^{n_y \times n_x}$, $D \in \mathbb{R}^{n_y \times n_u}$, $x = [x_1, x_2, \ldots, x_{n_x}]^T$, $u = [u_1, u_2, \ldots, u_{n_u}]^T$, and $y = [y_1, y_2, \ldots, y_{n_y}]^T$. Once the output vector $y \in \mathbb{R}^{n_y}$ and the input vector $u \in \mathbb{R}^{n_u}$ are known, then the state vector $x \in \mathbb{R}^{n_x}$ can be estimated if the system is observable. Analyzing observability based on the system structure is called structural (algebraic) observability analysis. Note that structural observability gives a necessary condition for observability, which means that if a system is not structural observable then it is not observable. A detailed discussion on structural observability analysis of linear systems is given in Reinschke (1988). The system structure can be represented

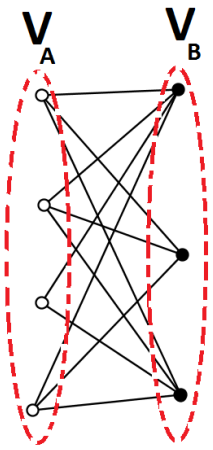


Figure 5: A bigraph. There are two sets of disjoint vertices (white and black colored). No edges among white nodes as well as black nodes.

graphically (using diagraph) and hence, graph-theoretic techniques can be used to analyze structural observability.

The system diagraph G is created in the following way. There are $n_x + n_y$ number of nodes representing state and output variables. If $< i, j >$-th element of A is not zero then there exists a directed edge from x_i to x_j. Similarly, the edges (x_i, y_j) are created by considering non-zero elements in C. The definition 1 gives the condition for output connectivity and the definition 2 defines the conditions to be satisfied for structural observability.

Definition 1 *A class of systems is said to be output-connectable*[6] *(or reachable) if in the digraph G there is a path from at least one of the output-vertices to every state-vertex. Reinschke (1988)*

Definition 2 *A class of systems is s-observable if and only if the digraph G meets the following condition:*

- *G is spanned by cacti. Lin (1974)*[7]*, Reinschke (1988)*

Violation in the definition 2 will make the system is not s-observable, hence not observable. To find the minimum number of driver nodes to achieve s-observability, the minimum input theorem is used Liu et al. (2011). The first step is to create the corresponding bipartite graph of the digraph. Let $G(V, E)$ is a digraph where $V = \{v_1, v_2, \ldots, v_{n_v}\}$ and $E = \{e_1, e_2, \ldots, e_{n_e}\}$. Define two disjoint sets of nodes such that $V^+ = \{v_1^+, v_2^+, \ldots, v_{n_v}^+\}$ and $V^- = \{v_1^-, v_2^-, \ldots, v_{n_v}^-\}$. Then create a bigraph with V^+ and V^-. If $(v_i, v_j) \in E$, then (v_i^+, v_j^-) is an edge of the bigraph. A maximum (or perfect) matching in the bigraph is also give a maximum (or perfect) matching in the digraph. The minimum number of driver nodes needed to achieve the s-observability is equal to the number of unmatched v_i^+'s in the bigraph and the driver nodes are corresponding v_i's. Note that if there is a perfect matching then there is single driver node. Matching algorithms for bipartite graphs are already implemented in NetworkX so that the minimum input theorem can be easily implemented in Python. Also refer the supplementary section to Liu et al. (2011) for more details.

Practical systems are often nonlinear, but interestingly it is possibly to apply the graph-theoretic approach discussed above for LTI systems directly to nonlinear systems Reinschke (1988), Daoutidis and Kravaris (1992), Liu et al. (2012), Boukhobza and Hemlin (2007) and Liu et al. (2011). Consider the nonlinear state space model

$$\begin{aligned} \dot{x} &= f\left(t, x, u\right), \\ y &= g\left(t, x, u\right), \end{aligned} \tag{2}$$

where $f = [f_1, f_2, \ldots, f_{n_x}]^T$, and $g = \left[g_1, g_2, \ldots, g_{n_y}\right]^T$. The digraph (G) containing $n_x + n_y$ number of nodes. If $\frac{\partial f_i}{\partial x_j} \neq 0$ then $x_i \rightarrow x_j$ exists and similarly, if $\frac{\partial g_i}{\partial x_j} \neq 0$ then $y_i \rightarrow x_j$ exists. Note that partial derivatives should be found symbolically Perera (2014), Perera et al. (2014), not numerically. The reason is that, for example even though x_j occurs in f_i, still it is possible to have numerically $\frac{\partial f_i}{\partial x_j} = 0$.

State estimation techniques — for example extended Kalman filtering — may be used to estimate unknown disturbances and unknown/uncertain parameters Simon (2006), Jazwinski (2007), Åström (2006). Let the nonlinear state space model

$$\begin{aligned} \dot{x} &= f\left(t, x, u, w, p\right), \\ y &= g\left(t, x, u, w, p\right), \end{aligned} \tag{3}$$

where $w = [w_1, w_2, \ldots, w_{n_w}]^T$ is the disturbance vector and $p = \left[p_1, p_2, \ldots, p_{n_p}\right]^T$ is the parameter vector. Assume w and p are unknown disturbances and uncertain parameters to estimated. One possibility is to write

$$\begin{aligned} \dot{w} &= 0, \\ \dot{p} &= 0, \end{aligned} \tag{4}$$

[6] Also called Y-topped Boukhobza and Hemlin (2007).

[7] Though Lin (1974) considered s-controllability, the concepts can be easily adapted to s-observability.

and then to augment w and p to the current state x. I.e. $\tilde{x} = [x, w, p]^T$. Now the augmented state space model is

$$\dot{\tilde{x}} = \tilde{f}(t, \tilde{x}, u), \qquad (5)$$
$$y = \tilde{g}(t, \tilde{x}, u).$$

$\tilde{f}$ and $\tilde{g}$ are then used in s-observability analysis as already explained using graph-theoretic techniques.

4 Python implementation

4.1 Modelica, JModelica.org and CasADi options

Modelica is becoming a standard tool for modeling large-scale complex physical systems. CasADi is a symbolic framework — a CAS tool — for numerical optimization and it is available to use it within Python. Modelica models — which result in differential algebraic equations, DAEs — can be imported to Python via CasADi and make symbolic DAEs available for general use in Python. See Perera et al. (2014) and Perera (2014). CasADi comes with JModelica.org and it may be the easiest way of accessing CasADi in Python.

JModelica.org provides three Python packages: `pymodelica`, `pyfmi` and `pyjmi`. `pymodelica` is for compiling (or model export) Modelica models while other two packages are for model import. `pyfmi` is for creating model objects according to FMI (Functional Mock-Up Interface) standards which is not at our interest here in this paper. `pyjmi` is for JModelica.org platform specific model importing. The relevant choices for exporting and importing are: the compiler `compile_fmux` (from `pymodelica`) for compiling and `CasadiModel` (from `pyjmi`) for importing.[8]

4.2 Structure of the Python script

The skeleton of the Python script is depicted in figure 6. First, the system model is encoded as a Modelica model. The Modelica model is then compiled and imported back to Python as a `CasadiModel` model object. The imported model is a symbolic flat representation of the Modelica model. Now using the CasADi Python package, necessary symbolic Jacobian matrices — which appeared in section 3 — are found. Once the Jacobian matrices are available, corresponding digraphs can be easily generated by means of NetworkX and PyGraphViz Python packages.[9] NetworkX supports to create four types of graph objects: Graph, DiGraph, MultiGraph, and MultiDiGraph.[10] In NetworkX, Graph and DiGraph graph objects are used only for graphs without multiple edges. To create graphs with multiple edges MultiGraph/MultiDiGraph graph objects should be used. It is clear that for s-observability analysis MultiDiGraph graph objects must be used. NetworkX provides many network algorithms related to: matching, bipartite graph related, strongly connectivity, cycles, tree, etc. The PyGraphviz Python package can be used as the layout tool.[11] The Matplotpib Python package may also be used for network drawing. The NetworkX and PyGraphviz network objects are convertible each other.

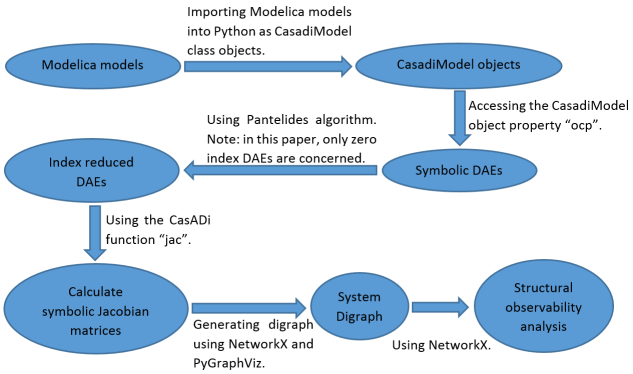


Figure 6: Structure of the Python script.

Let the Modelica model is stored in the file “MyModel.mo” and the model name is “mymodel”. The compilation is done using the Python code given below:[12]

```
# Import compiler compile_fmux
from pymodelica import compile_fmux
# Compile Modelica model
file_name = 'MyModel.mo'
model_name = 'mymodel'
compile_fmux(model_name,file_name)
```

The compiled model object is a “.fmux” file with the name “mymodel” and the model object is imported as a `CasadiModel` object. The code is given below:

[8] Often index of DAEs is greater than one (i.e. higher index problems.). In such cases, index should be reduced zero using Pantelides algorithm before applying the concept discussed in this paper. See Pantelides (1988) and Cellier and Kofman (2006).

[9] See in `http://networkx.github.io/` and `http://pygraphviz.github.io/`.

[10] “Multi” indicates that the graph object support multiple edges.

[11] See in `http://pygraphviz.github.io/`.

[12] The complete Python code will be available on request.

```
from pyjmi import CasadiModel
casadiModelObject = \
    CasadiModel('mymodel.fmux')
# Get flat ocp representation
ocp = casadiModelObject.ocp
```

Also, in order to use the CasADi package, it is imported as given below:

```
from casadi import *
from casadi.tools import *
```

ocp contains the information about the flattened symbolic model. For example ocp.x, ocp.z, ocp.pi, ocp.pd, ocp.pf, ocp.t, ocp.u, ocp.ode, and ocp.alg give respectively dynamic vector, algebraic state, independent parameter vector, dependent parameter vector, free parameter vector, time, input vector, vector of ODEs and vector of algebraic equations. Also casadiModelObject.dx gives dynamic state derivative vector. The main ingredient for generating a MultiDiGraph object is to have functions f and g which are given in equation 3 or $\tilde{f}$ and $\tilde{g}$ in equation 5. f is defined as an SXFunction class instance, say ffun. See below for the Python code:

```
# Define ODEs
f = ocp.ode
# Create an SXFunction for f
ffun = SXFunction([t,vertcat(xDot),vertcat(x),\
    vertcat(u)],[f])
ffun.init()
```

Now, consider how to create a MultiDiGraph. The Jacobian matrix $\frac{\partial f}{\partial x}$, which is given by ffun.jac(1), contains the information about the dependencies among state variables. In a similar way, g is defined as an SXFunction. Then $\frac{\partial g}{\partial x}$ gives information to construct the dependencies among state and output variables. the NetworkX and PyGraphViz packages are imported in the following way:

```
import networkx as nx
import pygraphviz as pgv
```

G = nx.MultiDiGraph() creates the MultiDiaGraph object with no edges and nodes. In order to add nodes for state, input and output variables use the following code:

```
G = nx.MultiDiGraph()
# Create state vertices
for i in x:
    G.add_node('{0}'.format(x[i]))
# Create input vertices
for j in u:
    G.add_node('{0}'.format(u[j]))
# Create output vertices
for k in y:
    G.add_node('{0}'.format(y[k]))
```

In order to add edges we can use the following code:

```
# Create edges among states
for i in range(n_x):
    for j in range(n_x):
        if isEqual(A[i,j],SX('0')) == False:
            G.add_edge('{0}'.format(x[j]),\
    ''.format(x[i])
# Create edges among states and inputs
for i in range(n_x):
    for j in range(n_u):
        if isEqual(B[i,j],SX('0')) == False:
            G.add_edge('{0}'.format(x[j]),\
    '{0}'.format(u[i])
            # Create edges among states \
    and inputs
for i in range(n_y):
    for j in range(n_x):
        if isEqual(C[i,j],SX('0')) == False:
            G.add_edge('{0}'.format(y[j]),\
    '{0}'.format(x[i])
```

Additionally, it is useful to do some formatting on nodes/edges. For example, states, input and output nodes are in different colors and shapes. NetworkX graph object can be converted to PyGraphViz AGraph objects using Gp = nx.to_agraph(G). See the code below:

```
Gp = nx.to_agraph(G)
Gp.write("file.dot")
Gp.layout()
Gp.layout(prog='dot')
G.draw('file.png')
```

In order to have a better structured code, several new functions may be defined within the CasadiModel class: symbolicLinearization(), symbolicDAEs(), createNodes(), createEdges(), generateGraph(), decomposeGraph(), Y_Topped(), Max_Matching(), etc. Now these functions can be called as for instance casadiModelObject.createNodes(). symbolicDAEs() creates symbolic functions for ODEs and algebraic equations. Symbolic Jacobian matrices are found by symbolicLinearization(). Based on Jacobian matrices the nodes and the edges of the

digraph are generated using createNodes() and createEdges(). generateGraph() creates a NetworkX and a PyGraphViz graph objects as well as it creates a ’dot’[13] file. decomposeGraph() decomposes the digraph into strongly connected components. To check the conditions given in the definition 2, Y_Topped() and Max_Matching() are used.
As a summary to this section, the following points are made: (1) a Modelica model is created, (2) import the dynamic model as a CasadiModel object model and use `casadi` to find symbolic Jacobian matrices of symbolic DAEs (after reducing the index if needed), (3) generate a digraph using networkx and pygrapviz, (4) use graph theories to analyze the digraph.

5 Industrial Application Case

The Copper electro-winning process at Glencore Nikkelverk, Kristiansand, Norway is considered. A mechanistic model for the process is given in Lie and Hauge (2008). The system model is in the form of equation 3 while the augmented model — by taking $\dot{p} = 0$ and w as slowly varying (i.e. $\dot{w} \approx 0$) — is in the form of equation 5. The nodes for p and w always have directed edges coming towards them starting from either output/state nodes. I.e. possible edges ending at parameter/disturbance nodes: $x_i \rightarrow p_j$, $y_i \rightarrow p_j$, $x_i \rightarrow w_j$ and $y_i \rightarrow w_j$. The following script is used to generate a digraph for structural observability analysis:

```
#from pyjmi import CasadiModel
from casadi_interface12 import CasadiModel
from pymodelica import compile_fmux
fmux = compile_fmux\
('CopperPlantPackage.CopperPlant',\
    'CopperPlant.mo')
model = CasadiModel(fmux)
model.symbolicDAEs()
model.symbolicLinearization()
model.createNodes()
model.createEdges()
model.generateGraph()
```

The above code creates the digraph G, which is depicted in figure 7. G can be decomposed into SCCs using the function decomposeGraph(). See figure 8. Note that there are two SCCs with more than one node. Each SCC is colored with different colors. It is possible to check whether G is output connected using Y_Topped() (see definition 1). Here we use `networkx.all_simple_paths()` and this function gives all possible paths starting from a given node and ending at a given node. See the script given below for the definition of Y_Topped():

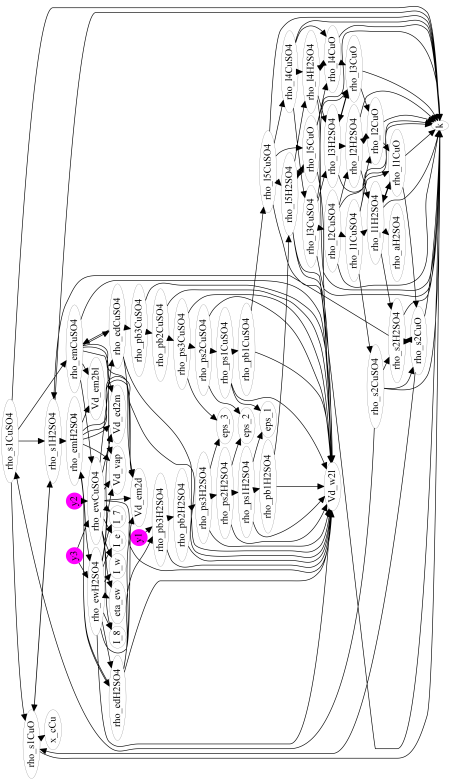


Figure 7: The graph, G.

```
def Y_Topped(self):
        Gnx = self.Gnx
        x = self.x
        for i in ['y1','y2','y3']:
            k = 'Y-topped!'
            for j in x:
                if list(ntwx.all_simple_paths\
(Gnx,source=str(j),target=i))==[]:
                    k = 'Not Y-topped!'
                    break
        print k
```

By implementing the minimum input theorem — the function Max_Matching_BP() —, we get the stems in the digraph and hence we can deduce the driver nodes (the minimum number of measurements) needed to achieve s-observability. The matched nodes are in red color while unmatched are in white. Unmatched nodes must be measured. See figures 9 and 10.

[13] See in http://www.graphviz.org/.

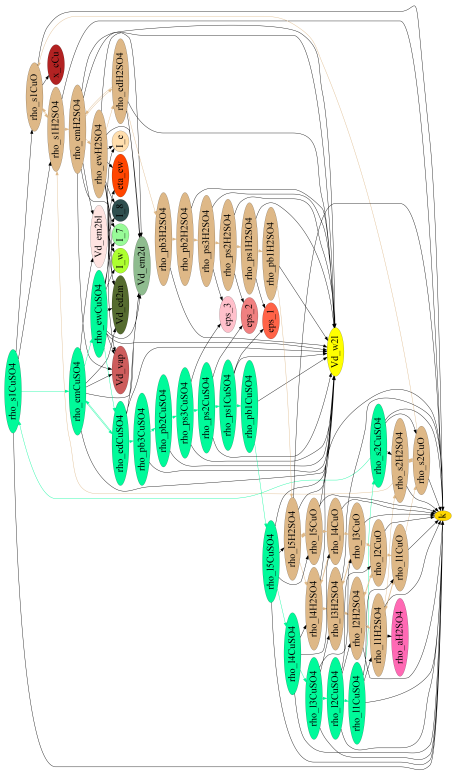


Figure 8: SCCs of the graph, G_decomp.

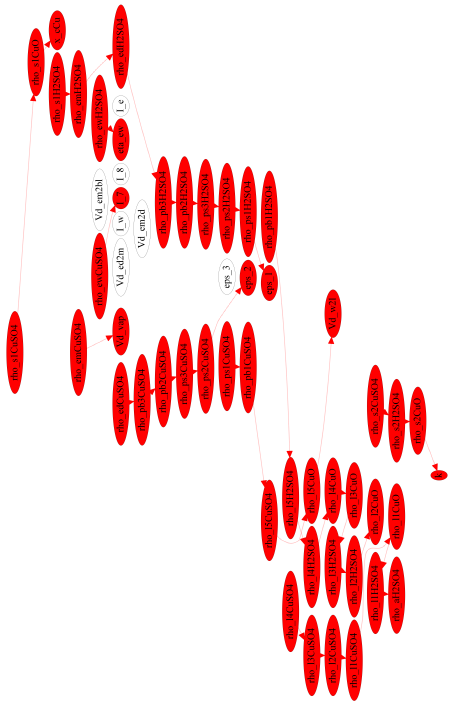


Figure 9: The stems of G.

6 Conclusion

We have demonstrated how to implement structural observability analysis in the view of graph-theoretic approach for large scale complex dynamic system in Python by using NetworkX, PyGraphViz Python packages as well as CasADi's Python front-end. The main result is how to find the spanning cacti for a given digraph in order to find the minimum number of driver nodes. Modelica is used for modeling and it is a standard tool for modeling large scale complex dynamic system. CasADi supports to import Modelica models into Python as flattened symbolic DAEs making it possible to use Modelica models in general use. Importantly, all the software tools which are used in our development are free.

References

Bondy, A. and Murty, U. S. R. *Graph theory*. Graduate texts in mathematics. Springer, 2008.

Boukhobza, T. and Hemlin, F. Observability analysis for structured bilinear systems: a graph-theoretic approach. *Automatica*, 2007. 43(11).

Cellier, F. E. and Kofman, E. *Continuous System Simulation*. Springer, 2006.

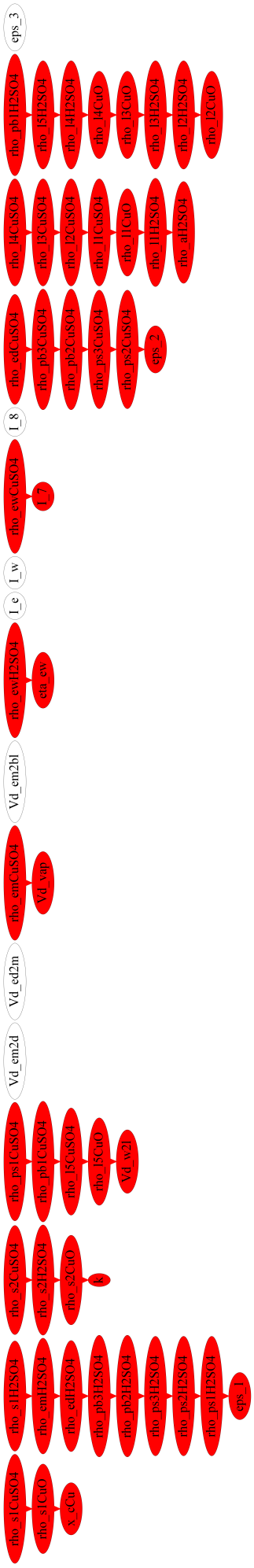
Daoutidis, P. and Kravaris, C. Structural evaluation of control configurations for multivariable nonlinear processes. *Chemical Engineering Science*, 1992. 47:1091–1107.

Jazwinski, A. H. *Stochastic Processes and Filtering Theory*. Dove Publications, Inc., Mineola, New York, 2007.

Lie, B. and Hauge, T. A. Modeling of an industrial copper leaching and electrowinning process, with validation against experimental data. *Proceedings SIMS 2008, 49th Scandinavian Conference on Simulation and Modeling*, 2008.

Lin, C. T. Structural controllability. *IEEE Transactions on Automatic Control*, 1974. 19(3).

Liu, Y.-Y., Slotine, J.-J., and Barabàsi, A.-L. Controllability of complex networks. *Nature*, 2011. 473:167–173. doi:doi:10.1038/nature10011.



Liu, Y.-Y., Slotine, J.-J., and Barabàsi, A.-L. Observability of complex systems. *Proceedings of the National Academy of Sciences of the United States of America*, 2012. 110(7):2460–2465. doi:10.1073/pnas.1215508110.

Pantelides, C. C. The consistent initialization of differential-algebraic systems. *SIAM Journal on Scientific Computing*, 1988. 9(2). doi:10.1137/0909014.

Perera, A. Using casadi for optimization and symbolic linearization/extraction of causality graphs of modelica models via jmodelica.org. Technical Report HiT_rapport_5, Telemark University College, Kjølnes ring 56, P.O. Box 203, N-3901 Porsgrunn, Norway., 2014. URL https://teora.hit.no/handle/2282/2175.

Perera, A., Pfeiffer, C., Hauge, T. A., and Lie, B. Making modelica models available for analysis in python control systems library. *Proceedings SIMS 2014, 55th Scandinavian Conference on Simulation and Modeling*, 2014.

Åström, K. J. *Introduction to Stochastic Control Theory*. Dove Publications, Inc., Mineola, New York, 2006.

Reinschke, K. J. *Multivariable control: a graph theoretic approach*. Lecture notes in control and information sciences. Springer-Verlag, Berlin, New York, 1988. URL http://opac.inria.fr/record=b1086834.

Simon, D. *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. John Wiley & Sons, Inc., Hoboken, New Jersey, 2006.

Paper C

Parameter and State Estimation of Large-Scale Complex Systems Using Python Tools

Modeling, Identification and Control, Vol. 31, No. 1, 2015, pp. 1–10, ISSN 1890–1328

Parameter and State Estimation of Large-Scale Complex Systems Using Python Tools

M. Anushka S. Perera [1] Tor Anders Hauge [2] Carlos F. Pfeiffer [1]

[1] *Telemark University College, Kjlnes ring 56, P.O. Box 203, N-3901 Porsgrunn, Norway. E-mail: carlos.pfeiffer@hit.no*

[2] *Glencore Nikkelverk, Kristiansand, Norway.*

Abstract

This paper discusses the topics related to automating parameter, disturbance and state estimation analysis of large-scale complex nonlinear dynamic systems using free programming tools. For large-scale complex systems, before implementing any state estimator, the system should be analyzed for structural observability and the structural observability analysis can be automated using Modelica and Python. As a result of structural observability analysis, the system may be decomposed into subsystems where some of them may be observable — with respect to parameter, disturbances, and states — while some may not. The state estimation process is carried out for those observable subsystems and the optimum number of additional measurements are prescribed for unobservable subsystems to make them observable. In this paper, an industrial case study is considered: the copper production process at Glencore Nikkelverk, Kristiansand, Norway. The copper production process is a large-scale complex system. It is shown that how to implement various state estimators, in Python, to estimate parameters and disturbances, in addition to states, based on available measurements.

Keywords: Kalman filter, Modelica, Observability, Python, state and parameter estimation

1 Introduction

Consider a class of nonlinear deterministic systems given by Equation 1, where $x \in \mathbb{R}^{n_x}$, $u \in \mathbb{R}^{n_u}$, $w \in \mathbb{R}^{n_w}$, $p \in \mathbb{R}^{n_p}$, and $y \in \mathbb{R}^{n_y}$ are respectively state, input, process noise, parameter, and output vectors and $C \in \mathbb{R}^{n_y \times n_x}$ is a constant matrix and $f(.)$ and $g(.)$ are known vector functions.[1]

$$\begin{aligned} \dot{x} &= f(x, u, w, p) \\ \dot{p} &= 0 \\ y &= h(x) = C \cdot x \end{aligned} \tag{1}$$

u and y are known as well as their time derivatives. x, w, and p are unknowns and the objective is to estimate them based on u-y information. The state space is augmented with p via $\dot{p} = 0$. However, it is not obvious how to augment w, since w is completely unknown. One possibility is to assume w is a random variable with a given probability characteristic. An example Gelb (2001) is given in Equation 2, where w_i — w_i is an element of $w = [w_1, w_2, \ldots, w_{n_w}]^T$ — is a stationary random process with the autocorrelation function $\gamma_i(\tau) = \left(\sigma_i^2 \delta_i / 2\right) \cdot e^{-\delta_i |\tau|}$, ϵ_i is a given white Gaussian process and δ_i, $\sigma_i > 0$.[2]

$$\dot{w}_i = -\delta_i w_i + \sigma_i \delta_i \cdot \epsilon_i \tag{2}$$

There are other alternatives for disturbance augmentation Bona and Smay (1966) and the choice may depends on whether the augmented system is observable

[1] $x \equiv x(t)$, $u \equiv u(t)$, etc.

[2] $\sigma_i \delta_i \cdot \epsilon_i$, in Equation 2, may be replaced by ϵ_i.

doi:10.4173/mic.2015.1.1

or not — there is no point of augmenting parameters and disturbances if the augmented system becomes unobservable. The complete augmented system is given in Equation 3 and corresponding state space representation is in Equation 4, where $g(.)$ is chosen appropriately by augmenting disturbances.[3]

$$\begin{aligned} \dot{x} &= f(x,u,w,p) \\ \dot{w} &= g(w) + \epsilon \\ \dot{p} &= 0 \\ y &= C \cdot x + v \end{aligned} \tag{3}$$

$$\underbrace{\begin{bmatrix}\dot{x}\\ \dot{w}\\ \dot{p}\end{bmatrix}}_{\dot{\hat{x}}} = \underbrace{\begin{bmatrix} f(x,u,w,p)\\ g(w)\\ 0\end{bmatrix}}_{\hat{f}(\hat{x},u)} + \underbrace{\begin{bmatrix}0\\1\\0\end{bmatrix}}_{\Gamma}\epsilon$$

$$\hat{y} = \underbrace{\begin{bmatrix}C & 0 & 0\end{bmatrix} \cdot \underbrace{\begin{bmatrix}x\\ w\\ p\end{bmatrix}}_{\hat{x}}}_{\hat{h}(\hat{x})} + v \tag{4}$$

The noise model related to Equation 1 is given by the stochastic nonlinear system 5, where ϵ and v (measurement noise) are vectors of white Gaussian processes such that $E\{w\} = 0 \in \mathbb{R}^{n_w}$, $E\{v\} = 0 \in \mathbb{R}^{n_y}$, $E\{ww^T\} = Q \in \mathbb{R}^{n_w \times n_w}$, $E\{vv^T\} = R \in \mathbb{R}^{n_y \times n_y}$, $E\{wv^T\} = S \in \mathbb{R}^{n_w \times n_y}$ (S is a zero matrix if w and v are independent), $\Gamma \in \mathbb{R}^{n_x \times n_w}$ is a constant matrix, and x_0 is independent from w and v. Associated discrete time version of 5 is 6.

$$\begin{aligned} \dot{\hat{x}} &= \hat{f}(\hat{x},u) + \Gamma \cdot \epsilon \\ \hat{y} &= \hat{h}(\hat{x}) + v, \end{aligned} \tag{5}$$

$$\begin{aligned} \hat{x}_{k+1} &= \hat{f}_k(\hat{x}_k, u_k) + \Gamma \cdot w_{k+1};\ k = 0,1,\dots \\ \hat{y}_k &= \hat{h}_k(\hat{x}_k) + v_k;\ k = 1,2,\dots \end{aligned} \tag{6}$$

Where,

$$\hat{f}_k(\hat{x}_k, u_k) = \hat{x}_k + \int_{t_k}^{t_{k+1}} \left[\hat{f}(\hat{x}(\theta), u(\theta))\right] d\theta$$

and

$$w_{k+1} = \Gamma \int_{\beta(t_k)}^{\beta(t_{k+1})} d\beta.$$

[3] Note that the number of states of the augmented system could be larger than $n_x + n_p + n_w$. For example, if w_i is augmented by $\ddot{w}_i = 0$. Also 0's and 1's, in Equations 3 and 4, represent zero and unit matrices with appropriate dimensions.

β is a Brownian motion process such that $\epsilon \cdot dt = d\beta$.[4] Now, the system in Equation 1 is simulated for given p, $\{w_k\}$ (w_k with some added noise), $\{u_k\}$, and x_0. $\{x_k\}$ and $\{y_k\}$ (a fictitious noise is added to y_k) are stored for $k = 1,2,3,\dots,n$.[5] Then, based on the simulated data $\{u_k\}$ and $\{y_k\}$, $\{\hat{x}_k\}$ is estimated, for $k = 1,2,\dots,n$, to verify the stability properties of state estimators.

Structure of the paper as follows. Section 2: a brief discussion on nonlinear observability, in particular giving more attention on structural observability. Section 3: the process model. Section 4: Observability analysis. Section 5: a brief introduction to filtering theory. Section 6: structure of the Python code and results. Section 7: conclusions and future work.

2 Nonlinear observability

In order to achieve the state observability of the stochastic system 5, it is a must that the corresponding noise-free system is observable Margaria et al. (2004). Hence, consider the noise-free version of equation 5:[6]

$$\begin{aligned} \dot{\hat{x}} &= \hat{f}(\hat{x},u) \\ \hat{y} &= \hat{h}(\hat{x}) \end{aligned} \tag{7}$$

For nonlinear systems, local observability should be concerned which is often tedious to handle for large-scale complex systems. Let an initial state $\hat{x}_0^i$, then for a given bounded input trajectory $u(t)$, assume there exists a solution trajectory $\hat{y}^i(t) = \hat{h}\left(\hat{x}^i(t)\right)$ for $t \in [0,T]$ and $T < \infty$, satisfying Equation 7. If the points lie within the neighborhood of $\hat{x}^i(0)$, satisfying $\hat{h}\left(\hat{x}^i(t)\right) \neq \hat{h}\left(\hat{x}^j(t)\right)$ such that $\hat{x}_0^i \neq \hat{x}_0^j$, then $\hat{x}_0^i$ and $\hat{x}_0^j$ are said to be (locally) distinguishable. Moreover, the local distinguishability property has a one-to-one correspondence with the local observability: local observability $\Leftrightarrow$ local distinguishable. Loosely speaking, different state trajectories starting from distinct initial states $\hat{x}_0^i$ and $\hat{x}_0^j$, will always generate distinct output trajectories. If not local distinguishable, then it is not possible to uniquely (locally) deduce $x^i(0)$ from $y^i(t)$-$u^i(t)$ information. A criteria for local observability/distinguishability or algebraic observability is given using Lie derivatives and brackets Isidori (1995), however algebraic observability analysis is not easy to use

[4] There are many possibilities of approximating stochastic integral $\Gamma \int_{\beta(t_k)}^{\beta(t_{k+1})} d\beta$. See in Kloeden et al. (2012).

[5] $\{a_k\} \equiv \{a_1, a_2, \dots, a_n\}$ for $k = 1,2,3,\dots,n$. $a_k \equiv a(t_k)$. $a(.)$ is a function of time.

[6] Margaria et al. (2004) considers affine systems. For more general treatment, refer Hermann and Krener (1977), Isidori (1995), Slotine et al. (1991).

for large-scale complex systems because the test related to searching for the rank of a matrix with higher dimensions and algebraic variables. The best solution is to use structural observability instead, preferably its the graph-theoretic associate, where structural dependencies among output and state variables are used to define a necessary condition for observability. Observability is a structural property and by exploiting the system structure, it is possible to extract much more information than the rank test for the algebraic observability check. The structural dependencies of the system 7 are mapped into a directed graph, so called the system digraph. The system digraph G is created as follows Reinschke (1988):

1. define nodes (or vertices) x_1, x_2, ...,$x_{n_{\hat{x}}}$, y_1, y_2, ..., $y_{n_{\hat{y}}}$,
2. there is a directed edge from x_i to x_j ($x_i \to x_j$) if $\frac{\partial \hat{f}_i}{\partial \hat{x}_j} \neq 0$, and
3. there is a directed edge from y_j to x_i ($y_j \to x_i$) if $\frac{\partial \hat{h}_j}{\partial \hat{x}_i} \neq 0$.

The system digraph may be decomposed into strongly connected components (SCCs): a SCC is a largest subgraph where there exists a directed path from each node to every other node in it. A root SCC is subgraph such that there are no incoming edges to its nodes. In order to achieve structural observability, at least one node of each root SCC should be measured. Hence, number of sensor nodes must not be less than number root SCCs Liu et al. (2013). On the hand, suppose that sensor nodes are given. Then, the system is structurally observable if the system digraph is spanned by cacti (the plural of cactus is cacti) covering all nodes Lin (1974). Figure 1 shows a cactus. A cactus consists of a stem and possibly one or more buds attach to the stem. Stem is a directed path where starting node is the root and end is the top. A root node is always a measurement. A bud is an elementary closed path, which connect to the stem via the distinguished edge. Bud should not be connected either to the root or to the top node and no two distinguished edges share the same node.

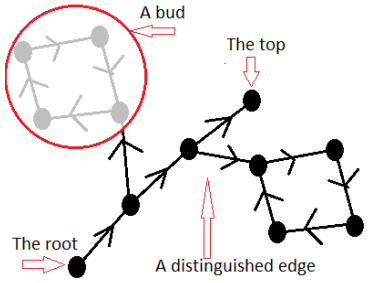


Figure 1: A cactus.

3 Process model

See Figure 2 for the process flow sheet. The process consists of four sections: (i) the slurrification section where powdered raw material containing mostly copper oxide (CuO) is slurrified using recycled anolyte flow, which containing sulfuric acid (H_2SO_4), taken from the electrowinning section, (ii) the leaching section where sulfuric acid is added to the slurry in order to leach more copper (Cu) into the solution, (iii) the purification section where the slurry is first filtered to extract the solution, which contains copper sulphate ($CuSO_4$), followed by the cementation and fine filtering processes, and (iv) the electrowinning section where the solution containing (Cu^{2+}) is electrolyzed to release solid copper at the cathode.

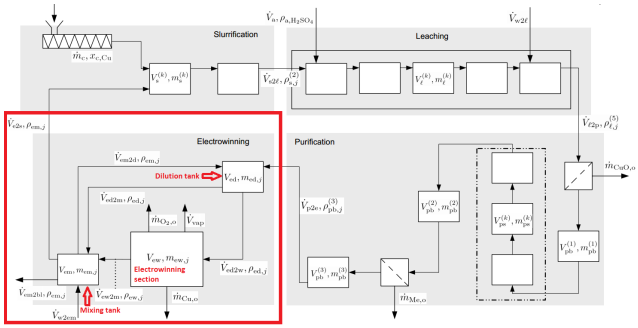


Figure 2: Process flow sheet for the Copper electrowinning process (electrowinning section is highlighted in red).

In the initial model Lie and Hauge (2008),[7] tank-level dynamics are neglected (i.e. static mass balances). However, level dynamics of several tanks are included in the modified model. It is assumed that the tanks in the slurrification and leaching sections as well as the electrowinning tank have no level variations. Level dynamics in the following tanks are included: buffer tank 1, buffer tank 2, buffer tank 3, dilution tank and mixing tank. System digraph is given in Figure 3 and it is seen that it is not possible to isolate a spanning cacti covering all nodes, then the system is not structurally observable and thereby, not observable Perera et al. (2015). Since, it is already proved in that the system is not structurally observable, only a (structurally) observable subsystem — which is the

[7] In order to save space, the complete model is not given in this paper. Refer Lie and Hauge (2008) for the detailed model.

electrowinning section — is considered in the following discussion.

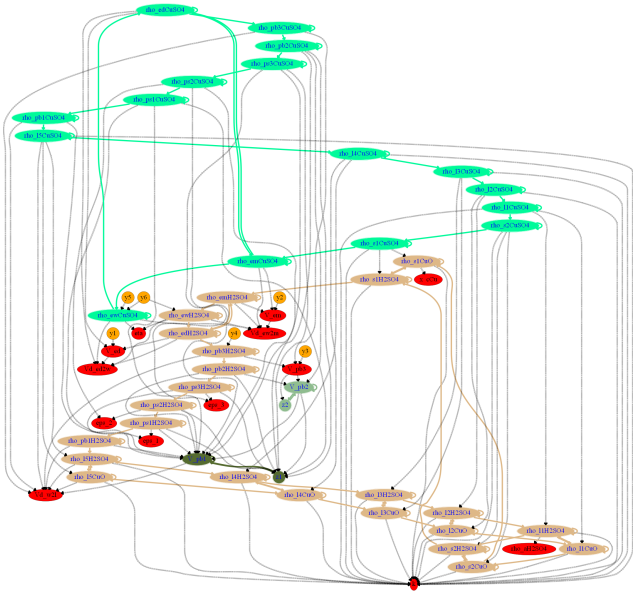


Figure 3: System digraph.

Electrowinning section consists of three subsystems: dilution tank, electrowinning tank, and mixing tank. Model equations are given below, where $\dot{m}_{\mathrm{Cu},o} = \frac{M_{\mathrm{Cu}}}{z_{Cu} \cdot C} \cdot \bar{\eta} \cdot \bar{I} \cdot 3600$.

$$\frac{dV_{ed}}{dt} = \dot{V}_{p2e} + \dot{V}_{em2d} - \dot{V}_{ed2m} - \dot{V}_{ed2w} \quad (8)$$

$$\frac{d\rho_{ed,\mathrm{CuSO_4}}}{dt} = \frac{\dot{V}_{p2e} \cdot \left(\rho^{(3)}_{pb,\mathrm{CuSO_4}} - \rho_{ed,\mathrm{CuSO_4}}\right)}{V_{ed}} + \frac{\dot{V}_{em2d} \cdot \left(\rho_{em,\mathrm{CuSO_4}} - \rho_{ed,\mathrm{CuSO_4}}\right)}{V_{ed}} \quad (9)$$

$$\frac{d\rho_{ed,\mathrm{H_2SO_4}}}{dt} = \frac{\dot{V}_{p2e} \cdot \left(\rho^{(3)}_{pb,\mathrm{H_2SO_4}} - \rho_{ed,\mathrm{H_2SO_4}}\right)}{V_{ed}} + \frac{\dot{V}_{em2d} \cdot \left(\rho_{em,\mathrm{H_2SO_4}} - \rho_{ed,\mathrm{H_2SO_4}}\right)}{V_{ed}} \quad (10)$$

$$\frac{d\rho_{ew,\mathrm{CuSO_4}}}{dt} = \frac{\dot{V}_{ed2w} \cdot \left(\rho_{ed,\mathrm{CuSO_4}} - \rho_{ew,\mathrm{CuSO_4}}\right)}{V_{ew}} + \frac{\dot{V}_{vap} \cdot \rho_{ew,\mathrm{CuSO_4}} - \frac{M_{\mathrm{CuSO_4}}}{M_{\mathrm{Cu}}} \cdot \dot{m}_{\mathrm{Cu},o}}{V_{ew}} \quad (11)$$

$$\frac{d\rho_{ew,\mathrm{H_2SO_4}}}{dt} = \frac{\dot{V}_{ed2w} \cdot \left(\rho_{ed,\mathrm{H_2SO_4}} - \rho_{ew,\mathrm{H_2SO_4}}\right)}{V_{ew}} + \frac{\dot{V}_{vap} \cdot \rho_{ew,\mathrm{H_2SO_4}} + \frac{M_{\mathrm{H_2SO_4}}}{M_{\mathrm{Cu}}} \cdot \dot{m}_{\mathrm{Cu},o}}{V_{ew}} \quad (12)$$

$$\dot{V}_{ed2w} = \dot{V}_{ew2m} + \dot{V}_{vap} \quad (13)$$

$$\frac{dV_{em}}{dt} = \dot{V}_{ed2m} + \dot{V}_{ew2m} + \dot{V}_{w2em} - \left(\dot{V}_{e2s} + \dot{V}_{em2d} + \dot{V}_{em2bl}\right) \quad (14)$$

$$\frac{d\rho_{em,\mathrm{CuSO_4}}}{dt} = \frac{\dot{V}_{ed2m} \cdot \left(\rho_{ed,\mathrm{CuSO_4}} - \rho_{em,\mathrm{CuSO_4}}\right)}{V_{em}} + \frac{\dot{V}_{ew2m} \cdot \left(\rho_{ew,\mathrm{CuSO_4}} - \rho_{em,\mathrm{CuSO_4}}\right)}{V_{em}} - \frac{\dot{V}_{w2em} \cdot \rho_{em,\mathrm{CuSO_4}}}{V_{em}} \quad (15)$$

$$\frac{d\rho_{em,\mathrm{H_2SO_4}}}{dt} = \frac{\dot{V}_{ed2m} \cdot \left(\rho_{ed,\mathrm{H_2SO_4}} - \rho_{em,\mathrm{H_2SO_4}}\right)}{V_{em}} + \frac{\dot{V}_{ew2m} \cdot \left(\rho_{ew,\mathrm{H_2SO_4}} - \rho_{em,\mathrm{H_2SO_4}}\right)}{V_{em}} - \frac{\dot{V}_{w2em} \cdot \rho_{em,\mathrm{H_2SO_4}}}{V_{em}} \quad (16)$$

$$\frac{d}{dt}\bar{\eta}(t) = 0 \quad (17)$$

$$\frac{d}{dt}\dot{V}_{ed2w} = \epsilon_{\dot{V}_{ed2w}} \quad (18)$$

$$\frac{d}{dt}\rho^{(3)}_{pb,\mathrm{CuSO_4}} = -\beta \cdot \rho^{(3)}_{pb,\mathrm{CuSO_4}} + \epsilon_{\rho^{(3)}_{pb,\mathrm{CuSO_4}}} \quad (19)$$

$$\frac{d}{dt}\dot{V}_{ew2m} = \epsilon_{\dot{V}_{ew2m}} \quad (20)$$

Where β is a positive constant which should be specified. The last 4 equations are due to parameter-disturbance augmentation. There are 4 measurements:

- $y_1 = V_{ed}$,
- $y_2 = V_{em}$,
- $y_3 = \frac{M_{\mathrm{Cu}}}{M_{\mathrm{CuSO_4}}} \cdot \rho_{ew,\mathrm{CuSO_4}}$, and
- $y_4 = \rho_{ew,\mathrm{H_2SO_4}} + \frac{M_{\mathrm{H_2SO_4}}}{M_{\mathrm{CuSO_4}}} \cdot \rho_{ew,\mathrm{CuSO_4}}$.

4 Observability analysis

Including augmented parameter $\bar{\eta}$ and disturbances $\dot{V}_{ed2w}$, $\dot{V}_{ew2m}$, and $\rho^{(3)}_{pb,\mathrm{H_2SO_4}}$, altogether there are 12 ($n_x = 12$) states. This makes it harder to analyze for algebraic observability. See Figure 4 for the system digraph. According to the digraph, it is always possible to estimate $\dot{V}_{ed2w}$ and $\dot{V}_{ew2m}$ using y_1 and y_2. The reason is that $y_1 \to V_{ed} \to \dot{V}_{ed2w}$ and $y_2 \to V_{em} \to \dot{V}_{ew2m}$ are two cacti (just two stems without buds). There exists a spanning cacti covering all nodes, hence the system is structurally observable. The spanning cacti is as follows:

- $y_1 \to V_{ed} \to \dot{V}_{ed2w}$,
- $y_2 \to V_{em} \to \dot{V}_{ew2m}$,
- $y_3 \to \rho_{ew,\mathrm{CuSO_4}} \to \rho_{ed,\mathrm{CuSO_4}} \to \rho_{em,\mathrm{CuSO_4}}$ with the bud $\rho^{(3)}_{pb,\mathrm{CuSO_4}} \to \rho^{(3)}_{pb,\mathrm{CuSO_4}}$, and
- $y_4 \to \rho_{ew,\mathrm{H_2SO_4}} \to \eta$ with the bud $\rho_{em,\mathrm{H_2SO_4}} \leftrightarrow \rho_{ed,\mathrm{H_2SO_4}}$.

Consider Equation 19. If $\beta = 0$, then the self-loop $\rho^{(3)}_{pb,\mathrm{CuSO_4}} \to \rho^{(3)}_{pb,\mathrm{CuSO_4}}$ in the digraph, in Figure 4, will disappear. Consequently, it is failed to have a spanning cacti. Therefore, $d\rho^{(3)}_{pb,\mathrm{CuSO_4}}/dt = 0$ is not a useful augmentation. Similarly, it can also be shown that $d^2\rho^{(3)}_{pb,\mathrm{CuSO_4}}/dt^2 = 0$ fails to keep structural observability. See the formulation below:

$$\frac{d}{dt}\rho^{(3)}_{pb,\mathrm{CuSO_4}} = \overline{\rho^{(3)}_{pb,\mathrm{CuSO_4}}} \tag{21}$$

$$\frac{d}{dt}\overline{\rho^{(3)}_{pb,\mathrm{CuSO_4}}} = \epsilon_{\overline{\rho^{(3)}_{pb,\mathrm{CuSO_4}}}} \tag{22}$$

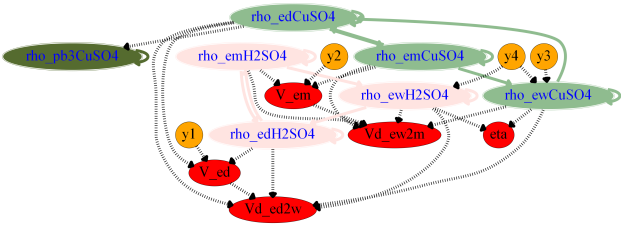


Figure 4: System digraph for the electrowinning section.

5 Filtering theory for discrete systems

The objective of filtering/estimation is to estimate system sate from available noisy data Jazwinski (2007). Consider the discrete stochastic dynamical system 6. Assume that system is observable.[8] $\{w_k, k = 1, 2, 3, ...\}$ is a random sequence with given probability distributions. Probability density function of $\hat{x}_0$ is also given. The famous assumptions are $\{w_k\}$ is white Gaussian sequence, $w_k \sim N(0, Q_k)$, and independent of $\hat{x}_0$. Also, $\{v_k, k = 1, 2, 3, ...\}$ is a random sequence with given probability distributions such as $v_k \sim N(0, R_k)$. A solution to the equation 6 is the probability density function of x_k. $\{wk\}$ and $\{vk\}$ may be dependent. We may write:

$$E\{\begin{bmatrix} w_k \\ v_k \end{bmatrix} \begin{bmatrix} w_l^T v_l^T \end{bmatrix}\} = \begin{bmatrix} Q_k & S_k \\ S_k^T & R_k \end{bmatrix} \delta_{kl} \tag{23}$$

Where $\delta_{kl} = 1$ when $k = l$, else $\delta_{kl} = 0$. $S_k = 0$ if $\{w_k\}$ and $\{vk\}$ are independent. The two sequences $Y_N = \{y_1, y_2, \ldots, y_N\}$ and $U_N = \{u_0, u_1, \ldots, u_{N-1}\}$ contain input-output data and $\{\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_{N-1}\}$ to be estimated for given Y_N and U_N. There are several ways to handle nonlinear filtering problems: extended Kalman filtering (EKF) Strang and Borre (1997), unscented Kalman filtering (UKF) Simon (2006), particle filtering (PF) Doucet et al. (2000), etc. Figure 5 gives a comparison about different state estimation techniques with respect to accuracy and computational effort.

Often, EKF could be the starting point for a nonlinear estimation problem, where linear Kalman filtering theory is adapted based on first-order linearization. The extended Kalman filter[9] is as follows for $k = 1, \ldots, n$:

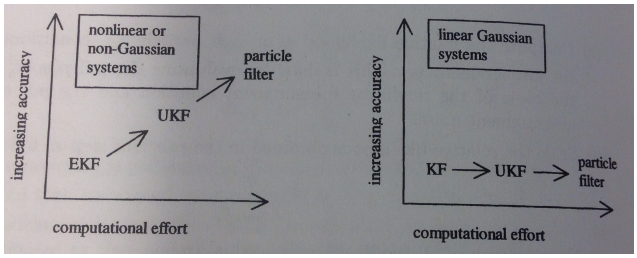


Figure 5: State estimation trade-offs (a scanned page from Simon (2006)).

[8] If the continuous system is observable, the discrete system is also observable Moraal and Grizzle (1995).

[9] For convergence characteristics of EKF, refer Ljung (1979) Cox (1964) Fitzgerald (1971).

- Q_{k-1}, R_{k-1}, $S_{k-1} = 0$, P_{k-1}^{+} and $\hat{x}_{k-1}^{+}$ are given;
- $A_{k-1} = \frac{\partial \hat{f}(\hat{x},u)}{\partial \hat{x}}\Big|_{\hat{x}_{k-1}^{+}}$, A_{k-1} is invertible and pair (A_{k-1}, C_{k-1}) is observable Song and Grizzle (1992);
- $P_k^{-} = A_{k-1} P_{k-1}^{+} A_{k-1}^{T} + \Gamma Q_{k-1} \Gamma^{T}$;
- $\hat{x}_k^{-} = f(\hat{x}_{k-1}^{+}, u_{k-1})$;
- $C_k = \frac{\partial h(\hat{x})}{\partial \hat{x}}\Big|_{\hat{x}_k^{-}}$;
- $K_k = P_k^{-} C_k^{T} \left(C_k P_k^{-} C_k^{T} + R_k\right)^{-1}$;
- $\hat{x}_k^{+} = \hat{x}_k^{-} + K_k \left[y_k - C_k \hat{x}_k^{-}\right]$, and
- $P_k^{+} = \left(I - K_k C_k\right) P_k^{-}$.

Where $\hat{x}_k^{+}$ is the best estimate for x_k. The main reason for the divergence of EKF is the model fidelity. This can be demonstrated easily using even a simple scalar system Fitzgerald (1971) Simon (2006). In Equation 5, $\Gamma \in \mathbb{R}^{n_x \times n_w}$ and $n_w \leq n_x$. If $n_w < n_x$, then there is at least one differential equation where a process noise term does not appear. However, by including a fictitious process noise to such equations, it may be possible to compensate model inaccuracies to some extent. For a constant parameter, ideally, $p_{k+1} = p_k$ follows without a process noise term and but, even for this case, a small fictitious noise is included — i.e $p_{k+1} = p_k + \epsilon_p$. Now, we have a stochastic system given in Equation 24, where $\Gamma_x, \Gamma_w, \Gamma_p \neq 0$ and $[\Gamma_x, \Gamma_w, \Gamma_p]^{T} \in \mathbb{R}^{n_x+n_w+n_p \times n_x+n_w+n_p}$ and $w_k \in \mathbb{R}^{n_x+n_w+n_p}$. Also, in order to increase the stability-convergence characteristics, fading-memory filters may be used Simon (2006). Actually, it is not necessary to include fictitious noises to all differential equations, but it enough to include for some of them. Fictitious noises are inserted such that the stochastic system becomes state stabilizable with respect to process noise vector Potter (1965).

$$\begin{bmatrix} x_{k+1} \\ w_{k+1} \\ p_{k+1} \end{bmatrix} = \begin{bmatrix} f\left(x_k, p_k, u_k\right) \\ g_k(w_k) \\ p_k \end{bmatrix} + \begin{bmatrix} \Gamma_x \\ \Gamma_w \\ \Gamma_p \end{bmatrix} w_{k+1};\ k = 0, 1, \ldots, n \quad (24)$$

EKF is susceptible to linearization errors and if the model is highly nonlinear then the trust on EKF is too low. The unscented Kalman filter Julier et al. (1995) Doucet et al. (2000) is a possible candidate to try with, if the EKF fails. Importantly, UKF doesn't require to calculate Jacobian matrices. The following steps are followed in UKF algorithm:

- Q_{k-1}, R_{k-1}, P_{k-1}^{+} and x_{k-1}^{+} are given;
- Find sigma points $\hat{x}_{k-1}^{(i)}$ such that $\hat{x}_{k-1}^{(i)} = x_{k-1}^{+} + x^{(i)}$ and $x^{(i)} = (-1)^{i} \cdot \left(\sqrt{n \cdot P_{k-1}^{+}}\right)_i^{T}$ for $i = 1, 2, \ldots, 2 \cdot n_x$. $\left(\sqrt{n \cdot P_{k-1}^{+}}\right)_i$ is the $< i >^{\text{th}}$ row of $\sqrt{n \cdot P_{k-1}^{+}}$;
- $x_k^{-} = \frac{1}{2 \cdot n_x} \Sigma_{i=1}^{2 \cdot n_x} \hat{x}_k^{(i)}$, where $\hat{x}_k^{(i)} = f(\hat{x}_{k-1}^{(i)}, u_{k-1})$;
- $P_k^{-} = \frac{1}{2 \cdot n_x} \Sigma_{i=1}^{2 \cdot n_x} \left[\left(\hat{x}_k^{(i)} - x_k^{-}\right)\left(\hat{x}_k^{(i)} - x_k^{-}\right)^{T}\right] + \Gamma Q_{k-1} \Gamma^{T}$;
- Find new set of sigma points $\hat{x}_k^{(i)}$ such that $\hat{x}_k^{(i)} = x_k^{-} + x^{(i)}$ and $x^{(i)} = (-1)^{i} \cdot \left(\sqrt{n \cdot P_k^{-}}\right)_i^{T}$ for $i = 1, 2, \ldots, 2 \cdot n_x$. $\left(\sqrt{n \cdot P_k^{-}}\right)_i$ is the $< i >^{\text{th}}$ row of $\sqrt{n \cdot P_k^{-}}$;
- $y_k^{-} = \frac{1}{2 \cdot n_x} \Sigma_{i=1}^{2 \cdot n_x} \hat{y}_k^{(i)}$, where $\hat{y}_k^{(i)} = C \cdot \hat{x}_k^{(i)}$;
- $P_y = \frac{1}{2 \cdot n_x} \Sigma_{i=1}^{2 \cdot n_x} \left[\left(\hat{y}_k^{(i)} - y_k^{-}\right)\left(\hat{y}_k^{(i)} - y_k^{-}\right)^{T}\right] + R_k$
- $P_{xy} = \frac{1}{2 \cdot n_x} \Sigma_{i=1}^{2 \cdot n_x} \left[\left(\hat{x}_k^{(i)} - x_k^{-}\right)\left(\hat{y}_k^{(i)} - y_k^{-}\right)^{T}\right] + R_k$
- $K_k = P_{xy} P_y^{-1}$;
- $x_k^{+} = x_k^{-} + K_k \left[y_k - y_k^{-}\right]$, and
- $P_k^{+} = P_k^{+} - K_k P_y K_k^{T}$.

6 Structure of the Python code and results

Perera et al. (2015) discusses a procedure for automating structural observability analysis in Python using JModelica.org-Casadi interface. Figure 3 and 4 are generated based on above mentioned article. There are several Python packages for state estimation: pyda, filterpy, pykalman, KF, etc.[10] First simulated noisy data is created. See the Python script given below. `model(x,t,u,w,p)` represents from Equation 8 to 16 and `f(x,dt,u,w,p)` is the discrete system. Simulate

[10] Pyda, filterpy, pykalman, and KF packages are available at the Python package index. See in https://pypi.python.org/pypi.

data is obtained with known variations — see the for loop below — in disturbances and inputs.

```
# Import packages
import numpy as np
from numpy import random
from numpy.random import randn
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import scipy.io as sio
< define necessary parameters >
def model(x,t,u,w,p):
    """
    Continuous dynamic model:
    dx/dt = model(x,t,u,w,p)
    """
    < enter code here >
    return np.array(dxdt)
def f(x,dt,u,w,p):
    """
    State transition function:
    x(t+dt) = f(x(t),dt,u(t))
    """
    res = odeint(lambda X,T:\
                 model(X,T,u,w,p),x,np.linspace(0.0,dt,2))
    return res[-1,:]
def h(x,v):
    """
    Observation function:
    y(t) = h(x(t))
    """
    z0 = x[0] + v[0]
    z1 = x[1] + v[1]
    z2 = (M_Cu/M_CuSO4)*x[2] + v[2]
    z3 = x[5] + (M_H2SO4/M_CuSO4)*x[2] + v[3]
    z = [z0,z1,z2,z3]
    return np.array(z)
# Define initial x,u,w and p
w = w0
x = x0
p = p0
u = u0
# Start simulation
for k in np.arange(1,N+1):
    # calculate x and z using discrete model
    x = f(x,dt,u,w,p)
    z = h(x,v)
    # log x and z
    < enter code here >
    # change disturbances
    # change inputs
    < enter code here >
# Save data
sio.savemat('data.mat',{'t':t,< enter code here >)
```

Now state estimators are considered. First, `model(x,t,u,w,p)` is updated as `model(x,t,u,w)` with augmented from Equation 17 to 20. In order to implement EKF algorithm in Section 5, it is necessary to calculate Jacobian matrices of `f` and `h`: see methods `ABL(x,u,w,dx = 1.e-5,du = 1.e-5,dw = 1.e-5)` and `CM(x,v,dx = 1.e-5,dv = 1.e-5)` below. Finally, two more methods are created: `predict(x,u,P,Q)` and `update(x,z,P,R)` and then EKF is simulated in a for loop. Fading-memory EKF can be easily implemented in a similar way.

```
def model(x,t,u,w):
    """
    Continuous augmented dynamic model:
    dx/dt = model(x,t,u,w,p)
    """
    < enter code here >
    deta_ew_dt = 0. + w_eta_ew
    dVd_ed2w_dt = 0. + w_Vd_ed2w
    drho_pb3CuSO4_dt = -beta*rho_pb3CuSO4 + w_rho_pb3CuSO4
    dVd_ew2m_dt = 0. + w_Vd_ew2m
    return np.array(dxdt)
def ABL(x,u,w,dx = 1.e-5,du = 1.e-5,dw = 1.e-5):
    """
    Calculate A = df/dx, B = df/du, and L = df/dw,
    using finite (central) difference method.
    """
    < enter code here >
    return A,B,L
def CM(x,v,dx = 1.e-5,dv = 1.e-5):
    """
    Calculate C = dh/dx and M = dh/dv,
    using finite (central) difference method.
    """
    < enter code here >
    return C, M
def predict(x,u,P,Q):
    < enter code here>
    return x,P
def update(x,z,P,R):
    < enter code here>
    return x,P
# Simulate
for k in np.arange(1,N+1):
    # predict state
    x,P = predict(x,u,P,Q)
    # update state
    z = np.array([z0[k],z1[k],z2[k],z3[k]])
    x,P = update(x,z,P,R)
# log data
```

For UKF estimation, `filterpy` Python is used. Use following script to create UKP object:

```
from filterpy.kalman import UnscentedKalmanFilter as UKF
ukf = UKF(<enter code here>)
# Initialize UKF
ukf.x = x0
ukf.R = np.diag([0.01, 0.01,0.01, 0.01])
ukf.Q = 5*np.eye(dim_x)
ukf.P = 1.*np.eye(dim_x)
# Simulate
for k in np.arange(1,N+1):
    # predict state
    ukf.predict()
    # measurment update
    z = np.array([z0[k],z1[k],z2[k],z3[k]])
    ukf.update(z)
    x = ukf.x
    # data logging
```

7 Conclusions and future work

We have demonstrated a way of automating parameter-disturbance-state estimation process for large-scale complex dynamic systems completely using free software tools such as Modelica, Python, Casadi, etc. A real world case study is considered and managed to estimate 1 parameter, 3 disturbances and 8 states with 4 measurements using extended Kalman filter and unscented Kalman filter algorithms. Extended Kalman filter, fading-memory Kalman filter and unscented Kalman filter have shown more or less similar results. All three estimators converge. Note that in

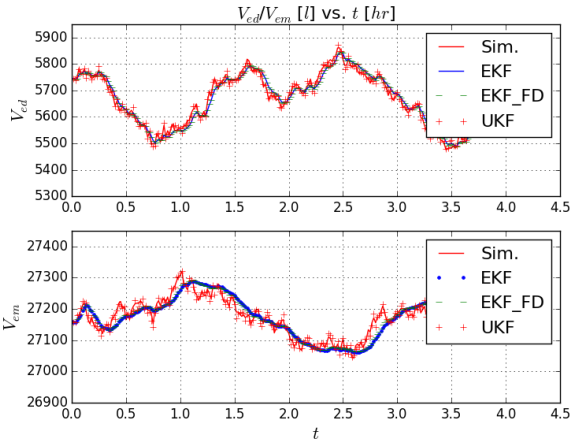


Figure 6: Estimation of V_{ed} and V_{em}.

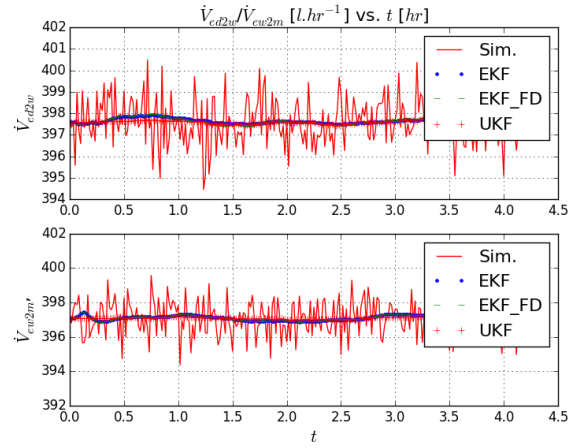


Figure 7: Estimation of $\dot{V}_{ed2w}$ and $\dot{V}_{ew2m}$.

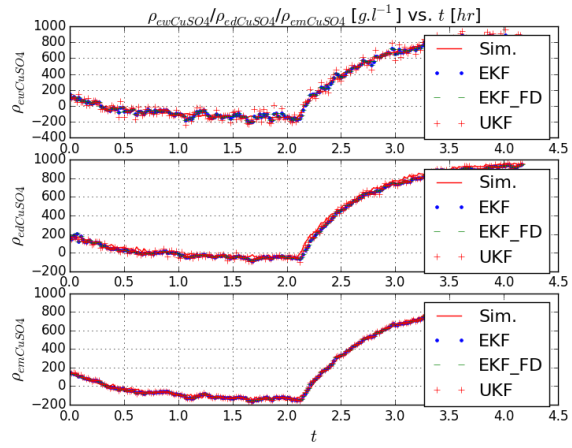


Figure 8: Estimation of $\rho_{ew,\mathrm{CuSO_4}}$, $\rho_{ed,\mathrm{CuSO_4}}$ and $\rho_{em,\mathrm{CuSO_4}}$.

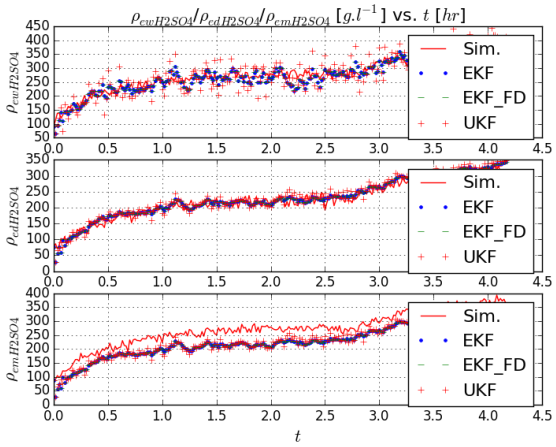


Figure 9: Estimation of $\rho_{ew,\mathrm{H_2SO_4}}$, $\rho_{ed,\mathrm{H_2SO_4}}$ and $\rho_{em,\mathrm{H_2SO_4}}$.

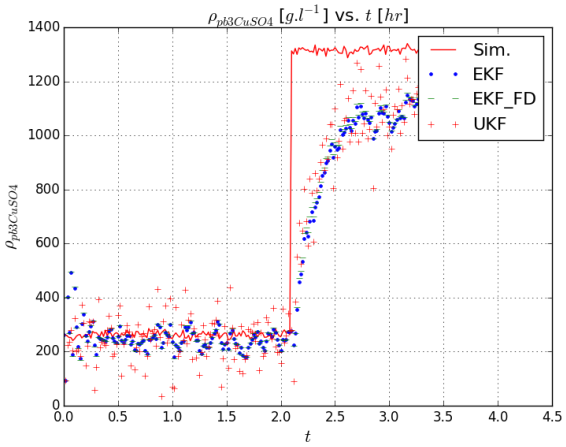


Figure 10: Estimation of $\rho_{pb3,\mathrm{CuSO_4}}$.

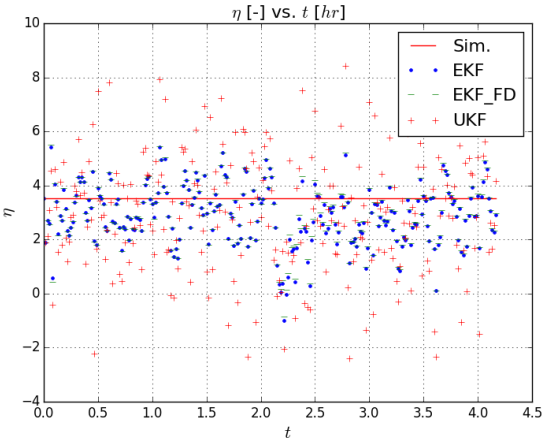


Figure 11: Estimation of η_{ew}.

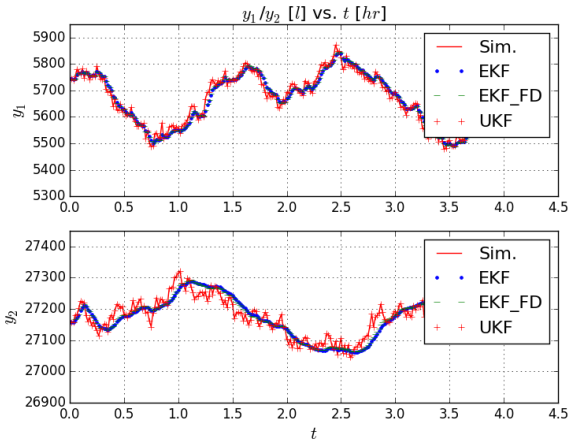


Figure 12: Estimation of $z0$ and $z1$.

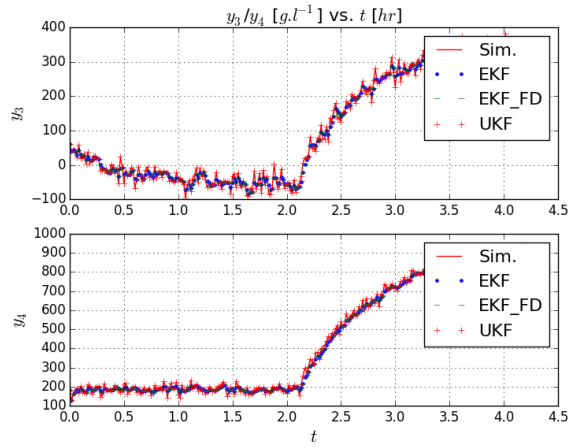


Figure 13: Estimation of $z2$ and $z3$.

this paper, only Gaussian process noises are concerned, often which is not the case in reality. Therefore, in order to test the capabilities of various estimators, those estimators should be applied with real process data where process noises may not be Gaussian and therefore, parameter-disturbance-state estimation with real process data is set as a future work. In particular, it is expected to implement particle filter algorithms with real process data, since particle filters can handle non-Gaussian process noise as well as it works better compared to EKF/UKF algorithms when the system model is highly nonlinear.

Filtering parameters, such as Q, R, P, etc., are tuning parameters, but filter tuning is not considered in this paper. Those parameters directly link with filter-response-time and filter sensitivity (propagation of the error covariance matrix P), therefor sensitivity analysis and filter tuning should be done as a future work. As the full state of the electrowinning system can be reconstructed from measurement data, it opens up many possibilities, among others, implementing an optimal control strategy. Also, estimating disturbances alongside the states will help to improve control performance — disturbance compensation. Finally, comparing the performances among PID and Optimal control strategies would be an interesting future work.

References

Bona, B. and Smay, R. J. Optimum reset of ship's inertial navigation system. *Aerospace and Electronic Systems, IEEE Transactions on*, 1966. (4):409–414.

Cox, H. On the estimation of state variables and parameters for noisy dynamic systems. *Automatic Control, IEEE Transactions on*, 1964. 9(1):5–12.

Doucet, A., Godsill, S., and Andrieu, C. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and computing*, 2000. 10(3):197–208.

Fitzgerald, R. J. Divergence of the kalman filter. *Automatic Control, IEEE Transactions on*, 1971. 16(6):736–747.

Gelb, A. *Applied optimal estimation*. The M.I.T. press, 2001.

Hermann, R. and Krener, A. J. Nonlinear controllability and observability. *IEEE Transactions on automatic control*, 1977. 22(5):728–740.

Isidori, A. *Nonlinear control systems*. Springer Science & Business Media, 1995.

Paper D

A Case Study: State Estimation and Optimal Control of an Industrial Copper Electrowinning Process

A Case Study: State Estimation and Optimal Control of an Industrial Copper Electrowinning Process

M. Anushka S. Perera, Tor Anders Hauge, and Carlos Fernando Pfeiffer

***Abstract*—This paper discusses an industrial case study related to the topics mathematical modeling, state-parameter-disturbance estimation, optimal control of large-scale complex control systems and technical computing. The case study involves the copper electrowinning process, which is a part of the copper leaching plant at Glencore Nikkelverk, Kristiansand, Norway. Improved control of chemical compositions within the electrowinning process through an optimal control strategy is one of our objectives. We present a way to solve this particular control problem, and in principal, the same procedure can be adapted to handle any large-scale complex control problem. State-parameter-disturbance estimation is a sub-problem, and two state estimators --- a modified version of the Extended Kalman Filter (EKF) and the Moving Horizon Estimate (MHE) --- are used to reconstruct the system state using simulated input-output data. It is shown that the EKF fails to estimate one of the parameters unless the algorithm is modified by adding an instability term. The MHE offers promising results in estimating parameters as compared to the classical EKF. Furthermore, the MHE explicitly handle constraints, which is an advantage. We use Modelica (as a systematic and efficient modeling approach for large-scale systems), Python (as a free and powerful tool for technical computing), structural analysis and graph-theory as main ingredients in the development and this combination significantly ease the analysis and synthesis of large-scale control systems. In the essence, our aim is twofold: (1) to demonstrate a simple, but a useful procedure of automating large-scale complex (optimal) controller design and synthesis and parameter estimation using available analytical and free computer-aided tools; and (2) to highlight the need of developing interfaces between Modelica/Simscape or similar modeling standards with a powerful programming language for technical computing, for example Python/MATLAB.**

***Index Terms*—Controllability and observability, large-scale control systems, state and parameter estimation, technical computing.**

I. Introduction

We present a solution to an Optimal Control Problem (OCP) [1] related to a real chemical process which is a part of the copper leaching plant at Glencore Nikkelverk, Kristiansand, Norway. The paper has two main considerations: (1) deploying available analytical tools in the analysis of large-scale complex control systems and (2) the software implementation.

It is beneficial to make use of a modeling language such as Modelica [4] or Simscape or similar for modeling of large-scale complex physical systems. The Simscape language is based on the MATLAB which is a commercial software. We consider only non-commercial tools in this paper. *Modelica* is free and is an object-oriented, declarative, multi-domain modeling language for component oriented modeling. These features makes Modelica a systematic and efficient modeling tool. A system model encoded in Modelica is translated via Modelica based simulation tools such as Dymola, OpenModelica and JModelica.org (or CasADi [5]) to a set of differential, algebraic and discrete equations (so-called flat hybrid DAEs). If possible, it is useful to make flat hybrid DAEs available for general use without merely limiting to model simulations, optimizations, etc. One such attempt is CasADi where Modelica models can be imported to *Python* as flat hybrid DAEs. We emphasis the importance of developing interfaces between Modelica (or similar) and software tools for technical computing such as Python, MATLAB (has its own symbolic core and provides the Simscape language which supports object-oriented, declarative, multi-domain and component based modeling), Maple (inherently a symbolic framework) or similar.

Scale and complexity of nonlinear control systems determines the viability of implementing any theoretical analysis considering the entire system as a whole. Hence, as a first step it is often necessary to exploit the system structure [6] of a given large-scale complex control system prior to implementing any algebraic analysis such as algebraic observability analysis. Structurally decomposing a given large-scale complex system into subsystems, and then using those subsystems in the analysis and synthesis is one possibility. We demonstrate the usage of structural analysis by considering an electrowinning process.

Improved control of chemical compositions within the electrowinning process through an optimal control strategy is the control objective. Classical PID controllers may perform poorly on nonlinear Multiple-Input and Multiple-Output (MIMO) systems and they do not explicitly handle input, output and state constraints. Also, controller tuning in the presence of many PID controllers can be tricky. In order to avoid such drawbacks in classical controllers, an optimal control [7][8] strategy, such as Nonlinear Model Predictive Control (NMPC) [9], can be considered. An OCP consists of two main subproblems: (1) observability analysis with respect to state-parameter-disturbance estimation [10], and (2) controllability analysis. These subproblems are dealt with in detail by exploiting the structural properties of the model. We consider two state estimators: the Extended Kalman Filter

Manuscript received May 1, 2016.

M. Anushka S. Perera is with the University College of Southeast Norway (e-mail: anushka_mrt@yahoo.com / anushka.perera@hit.no).

Tor Anders Hauge is with Glencore Nikkelverk, Kristiansand, Norway (e-mail: tor.hauge@glencore.no).

Carlos Fernando Pfeiffer is with the University College of Southeast Norway (e-mail: carlos.pfeiffer@hit.no).

(EKF) [10][11][12][13] and the Moving Horizon Estimate [14]. In our case study, it is found that the EKF is unable to estimate one of the parameters whilst the Moving Horizon Estimate provides accurate estimates for all state variables, parameters and disturbance variables. Inclusion of an instability [15] term to the EKF completely solve the divergence problem. More details about the divergence properties of the EKF can be found in [16][17]. The structure of the paper is as follows:

- Section II presents a mechanistic model for the copper electrowinning process.
- Section III discusses state estimation and optimal control.
- Section IV explains a software implementation.
- Section V shows the results.
- Section VI is for conclusion and future work.

II. Mathematical Model

A mechanistic model for the copper leaching process is available in [2]. See Fig. 1 for the process flow sheet. The model is given by Eqs. 1 to 12. There are four measurements: y_0, y_1, y_2 and y_3 (see Eq. 20). In order to control the chemical compositions in the electrowinning sections, sulfuric acid (sulfuric acid flow rate is $\dot{V}_a$) and powdered raw material which contains copper oxide (mass flow rate is $\dot{m}_c$) are added in the slurrification and leaching sections respectively. There are time delays — according to process experience time delays vary around 2 hours — before $\dot{V}_a$ and $\dot{m}_c$ affect the chemical compositions in the third buffer tank $V_{pb}^{(3)}$. In this paper, H_2SO_4 and $CuSO_4$ compositions in the third buffer tank are taken as two control variables u_2 and u_3 (i.e. delays are not considered.) Tank volumes (or levels) are controlled by u_0 and u_1 (see Eq. 16). $w^{(1)}$ is the unknown disturbance vector (see Eq. 17) and $w^{(2)}$ is the known disturbance vector (see Eq. 18). The volume of the electrowinning tank V_{ew} and η are unknown parameters and these parameters are augmented as state variables — see Eqs. 11 and 12. p is the parameter vector given in Eq. 19.

$$\frac{d}{dt}V_{ed} = \dot{V}_{p2e} + \dot{V}_{em2d} - \dot{V}_{ed2m} - \dot{V}_{ed2w} \tag{1}$$

$$\frac{d}{dt}\rho_{ed,CuSO4} = \frac{\dot{V}_{p2e}\cdot\left(\rho_{pb,CuSO4}^{(3)} - \rho_{ed,CuSO4}\right)}{V_{ed}} + \frac{\dot{V}_{em2d}\cdot\left(\rho_{em,CuSO4} - \rho_{ed,CuSO4}\right)}{V_{ed}} \tag{2}$$

$$\frac{d}{dt}\rho_{ed,H2SO4} = \frac{\dot{V}_{p2e}\cdot\left(\rho_{pb,H2SO4}^{(3)} - \rho_{ed,H2SO4}\right)}{V_{ed}} + \frac{\dot{V}_{em2d}\cdot\left(\rho_{em,H2SO4} - \rho_{ed,H2SO4}\right)}{V_{ed}} \tag{3}$$

$$\frac{d}{dt}\rho_{ew,CuSO4} = \frac{\dot{V}_{ed2w}\cdot\left(\rho_{ed,CuSO4} - \rho_{ew,CuSO4}\right)}{V_{ew}} + \frac{\dot{V}_{vap}\cdot\rho_{ew,CuSO4} - \left(\frac{M_{CuSO4}}{10^3}\right)\cdot\left(\frac{\dot{m}_{Cu,o}}{M_{Cu}}\right)}{V_{ew}} \tag{4}$$

$$\frac{d}{dt}\rho_{ew,H2SO4} = \frac{\dot{V}_{ed2w}\cdot\left(\rho_{ed,H2SO4} - \rho_{ew,H2SO4}\right)}{V_{ew}} + \frac{\dot{V}_{vap}\cdot\rho_{ew,H2SO4} + \left(\frac{M_{H2SO4}}{10^3}\right)\cdot\left(\frac{\dot{m}_{Cu,o}}{M_{Cu}}\right)}{V_{ew}} \tag{5}$$

$$\dot{V}_{ed2w} = \dot{V}_{ew2m} + \dot{V}_{vap} \tag{6}$$

$$\frac{\dot{m}_{Cu,o}}{M_{Cu}} = \frac{3600\cdot\overline{\eta}}{\underbrace{z_{Cu}\cdot C}_{\eta}}\cdot\bar{I} \tag{7}$$

$$\frac{d}{dt}V_{em} = \dot{V}_{ed2m} + \dot{V}_{ew2m} + \dot{V}_{w2em} - \left(\dot{V}_{e2s} + \dot{V}_{em2d} + \dot{V}_{em2bl}\right) \tag{8}$$

$$\frac{d}{dt}\rho_{em,CuSO4} = \frac{\dot{V}_{ed2m}\cdot\left(\rho_{ed,CuSO4} - \rho_{em,CuSO4}\right)}{V_{em}} + \frac{\dot{V}_{ew2m}\cdot\left(\rho_{ew,CuSO4} - \rho_{em,CuSO4}\right)}{V_{em}} - \frac{\dot{V}_{w2em}\cdot\rho_{em,CuSO4}}{V_{em}} \tag{9}$$

$$\frac{d}{dt}\rho_{em,H2SO4} = \frac{\dot{V}_{ed2m}\cdot\left(\rho_{ed,H2SO4} - \rho_{em,H2SO4}\right)}{V_{em}} + \frac{\dot{V}_{ew2m}\cdot\left(\rho_{ew,H2SO4} - \rho_{em,H2SO4}\right)}{V_{em}} - \frac{\dot{V}_{w2em}\cdot\rho_{em,H2SO4}}{V_{em}} \tag{10}$$

$$\frac{d}{dt}\eta = 0 \tag{11}$$

$$\frac{d}{dt}V_{ew} = 0 \tag{12}$$

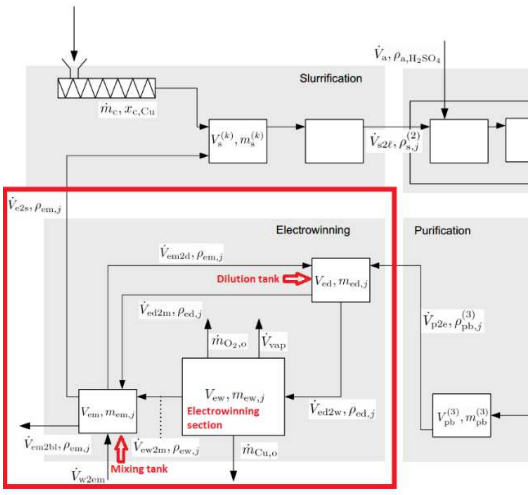


Figure 1: A part of the flow sheet for the copper leaching process (taken from [16]).

Corresponding noisy system is given in Eq. 21, where v is the measurement noise. The state vector x is given in Eq. 15. f and h are given functions while ψ is usually not known. Unknown disturbance may be modeled as

$$\dot{w}^{(2)} = \psi\left(w^{(2)}, \beta\right) + \varepsilon$$

where β is given parameter vector and ε is a given white noise process (see Eqs. 13 and14). See [18] for one possible disturbance model.[1] Eqs. 13 and 14 give possible disturbance models for $\dot{V}_{ed2w}$ and $\dot{V}_{ew2m}$. Eq. 22 gives the discrete version of Eq. 21 — where, $x \equiv x(t)$, $x_k \equiv x(t_k)$ and $t_k = k.\Delta t$ for $k \in \mathbb{Z}_{\geq 0}$. t is discretized with the time step Δt. f_k, ψ_k and h_k are some functions. $u(t)$ and $w^{(1)}(t)$ are taken to be piecewise constant functions — i.e., $u(t) = u_k$ and $w^{(1)}(t) = w_k^{(1)}$ are constants for $t \in [k.\Delta t, (k+1).\Delta t]$.

$$\frac{d}{dt}\dot{V}_{ed2w} = -\underbrace{\beta_{\dot{V}_{ed2w}}}_{\geq 0}\dot{V}_{ed2w} + \varepsilon_{\dot{V}_{ed2w}} \tag{13}$$

$$\frac{d}{dt}\dot{V}_{ew2m} = -\underbrace{\beta_{\dot{V}_{ew2m}}}_{\geq 0}\dot{V}_{ew2m} + \varepsilon_{\dot{V}_{ew2m}} \tag{14}$$

$$x_{max} \geq x = \begin{bmatrix} V_{ed} \\ V_{em} \\ \rho_{ed,CuSO4} \\ \rho_{ew,CuSO4} \\ \rho_{em,CuSO4} \\ \rho_{ed,H2SO4} \\ \rho_{ew,H2SO4} \\ \rho_{em,H2SO4} \end{bmatrix} \geq x_{min} > 0 \tag{15}$$

$$u_{max} \geq u = \begin{bmatrix} \dot{V}_{p2e} \\ \dot{V}_{e2s} \\ \rho^{(3)}_{pb,CuSO4} \\ \rho^{(3)}_{pb,H2SO4} \end{bmatrix} \geq u_{min} > 0 \tag{16}$$

$$w^{(1)} = \begin{bmatrix} \dot{V}_{em2d} \\ \dot{V}_{ed2m} \\ I \\ \dot{V}_{w2em} \\ \dot{V}_{em2bl} \end{bmatrix} > 0 \tag{17}$$

$$w^{(2)} = \begin{bmatrix} \dot{V}_{ed2w} \\ \dot{V}_{ew2m} \end{bmatrix} > 0 \tag{18}$$

$$p = \begin{bmatrix} \eta \\ V_{ew} \end{bmatrix} > 0 \tag{19}$$

$$y = \begin{bmatrix} V_{ed} \\ V_{em} \\ \frac{M_{Cu}}{M_{CuSO4}} \cdot \rho_{ew,CuSO4} \\ \rho_{ew,H2SO4} + \frac{M_{H2SO4}}{M_{CuSO4}} \cdot \rho_{ew,CuSO4} \end{bmatrix} > 0 \tag{20}$$

$$\Sigma: \begin{cases} \dot{x} = f\left(x, u, w^{(1)}, w^{(2)}, p\right) \\ \dot{p} = 0 \\ \dot{w}^{(2)} = \psi\left(w^{(2)}, \beta\right) + \varepsilon \\ y = h(x) + v \end{cases} \tag{21}$$

$$\Sigma_k: \begin{cases} x_{k+1} = f_k\left(\Delta t, x_k, u_k, w_k^{(1)}, w_k^{(2)}, p_k\right); k = 0,1,\ldots \\ p_{k+1} = p_k; k = 0,1,\ldots \\ w_{k+1}^{(2)} = \psi_k\left(w_k^{(2)}, \beta\right) + \varepsilon_k; k = 0,1,\ldots \\ y_k = h_k(x_k) + v_k; k = 0,1,\ldots \end{cases} \tag{22}$$

$$\Sigma_k^{Aug}: \begin{cases} x_{k+1}^{Aug} = \underbrace{\begin{bmatrix} f_k\left(\Delta t, x_k, u_k, w_k^{(1)}, w_k^{(2)}, p_k\right) \\ p_k \\ \psi_k\left(w_k^{(2)}, \beta\right) \end{bmatrix}}_{f_k^{Aug}\left(\Delta t, x_k^{Aug}, u_k, w_k^{(1)}\right)} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}}_{L_k} \varepsilon_k; k = 0,1,\ldots \\ y_k = h_k^{Aug}\left(x_k^{Aug}\right) + v_k; k = 0,1,\ldots \end{cases} \tag{23}$$

The elements of x, u, $w^{(1)}$, $w^{(2)}$, $w^{(1)}$, p, and y are scaled with respect to their nominal values. For example,

$$\rho_{ed,H_2SO_4} = \rho^0_{ed,H_2SO_4} . \rho^{sc}_{ed,H_2SO_4}$$

where ρ^0_{ed,H_2SO_4} is the nominal value of ρ_{ed,H_2SO_4} and such scaling makes ρ^{sc}_{ed,H_2SO_4} varies close to 1. It is advisable to use scaling if the variables vary in vastly different scales. Table 1 gives variables' units.

Variable	Units	Description
t	$[h]$	Time
V	$[m^3]$	Volume
$\dot{V}$	$[m^3/h]$	Volumetric flow rate
ρ	$[g/l]$	Mass concentration
M	$[g/mol]$	Molecular mass
z	$[-]$	Number of valence electrons
I	$[A]$	Electric current
$\bar{\eta}$	$[-]$	Currency efficiency

Table 1: Variables' units used in this paper.

III. State Estimation and Optimal Control

Consider the discrete stochastic system [13] in Eq. 22. The augmented state,

$$x_k^{Aug} = \begin{bmatrix} x_k \\ w_k^{(2)} \\ p_k \end{bmatrix}$$

may be estimated from available measurements (if the augmented system is observable) y_i for $i \leq k$; $w_i^{(1)}$ for $i \leq k-1$; and u_i for $i \leq k-1$. Various nonlinear state estimators are available, such as the Extended Kalman Filter (EKF), the Unscented Kalman Filter (UKF) [10], the Particle Filter (PF) [19][20][21], and the Moving Horizon Estimate (MHE). A combination of the MHE and the EKF (MHE-EKF combo) is used in this paper — EKF may be replaced by, for example, PF (i.e., MHE-PF combo). The success of a state

[1] A notation: n_a gives the number of elements of the vector a.

estimation process demands three things: (1) state observability, (2) state stochastic observability, (3) and stochastic controllability. Stochastic controllability associates with the excitation of state variables by process noises variables — more precisely, unstable state variables should be controllable with respect to process noise variables to achieve stochastic controllability. Note that some choices of ψ may cause the augmented system unobservable, hence it should be defined carefully. The augmented noisy system which is used in state-parameter-disturbance estimation is given in Eq. 23.[2]

Fig. 2 gives an EKF implementation. z_k^{Aug+} is an estimate for x_k^{Aug}. A_{k-1} matrix is adjusted by an instability term αI ($\alpha \geq 0$) and this improves the stability of EKF [22][25]. If $\alpha = 0$, then we have the conventional EKF. It is seen that $\alpha \neq 0$ is advisable especially for parameter estimation.

Notice that ε excites the system state x via $w^{(2)}$. Even though the unstable modes of the augmented system is controllable with respect to ε, it is still appropriate to add additional fictitious noise [10] variables to every state equation in the augmented system. Fictitious noise variables improve the stability of EKF — i.e., Eq. 21 becomes $\dot{x} = f\left(x, u, w^{(1)}, w^{(2)}, p\right) + \varepsilon_x$ and $\dot{p} = \varepsilon_p$, where ε_x and ε_p are fictitious noise vectors.

$$A_{k-1} = \frac{\partial}{\partial x_{k-1}^{Aug}} f_{k-1}^{Aug}\left(\Delta t, x_{k-1}^{Aug}, u_{k-1}, w_{k-1}^{(1)}\right)\Big|_{z_{k-1}^{Aug+}}$$
$$A_{k-1}^{\alpha} = A_{k-1} + \alpha I;\ \alpha \geq 0$$
$$Q_{k-1} \geq 0 \text{ is given.}$$
$$P_k^- = A_{k-1}^{\alpha} P_{k-1}^+ A_{k-1}^{\alpha T} + L_{k-1} Q_{k-1} L_{k-1}^T$$
$$P_k^- = \left(P_k^- + P_k^{-T}\right)/2$$
$$z_k^{Aug-} = f_{k-1}^{Aug}\left(\Delta t, z_{k-1}^{Aug+}, u_{k-1}, w_{k-1}^{(1)}\right)$$
$$C_k = \frac{\partial}{\partial x_k^{Aug}} h(x_k^{Aug})\Big|_{z_k^{Aug-}}$$
$$R_k > 0 \text{ is given.}$$
$$K_k = P_k^- C_k^T \left(C_k P_k^- C_k^T + R_k\right)^{-1}$$
$$z_k^{Aug+} = z_k^{Aug-} + K_k\left[y_k - C_k z_k^{Aug-}\right]$$
$$P_k^+ = (I - K_k C_k) P_k^-$$
$$P_k^+ = \left(P_k^+ + P_k^{+T}\right)/2$$

Figure 2: An EKF implementation.

Consider $y(t)$ for $t_k - T \leq t \leq t_k$, where T is a time horizon to the past — $T = N_{mhe}.\Delta t$ and N_{mhe} is a positive integer. $x(t_k)$ and $w^{(2)}(t_k)$ are estimated such that J_{mhe} (in Eq. 24) is minimized subjected to Eq. 23 and constraints given in Eqs. 15, 18 and 19. $\hat{z}_{k-N_{mhe}}$ should be known and it can be estimated using EKF or PF or other estimator. At the end of each sample time, the horizon is shifted one time step forward — i.e., $t \in [t_k + \Delta t - \mathrm{T}, t_k + \Delta t]$ — J_{mhe} is optimized again and continue. Weighting matrices Q_1^i, Q_2^i, and Q_3^i are given. See Fig. 2.

$$J_{mhe} = \sum_{i=k-N_{mhe}}^{k-1} \varepsilon_i^T Q_1^i \varepsilon_i + \sum_{i=k-N_{mhe}}^{k} \left(y_i - h_i^{Aug}(z_i)\right)^T Q_2^i \left(y_i - h_i^{Aug}(z_i)\right) + (z_{i-N_{mhe}} - \hat{z}_{i-N_{mhe}})^T Q_3^i (z_{i-N_{mhe}} - \hat{z}_{i-N_{mhe}}) \tag{24}$$

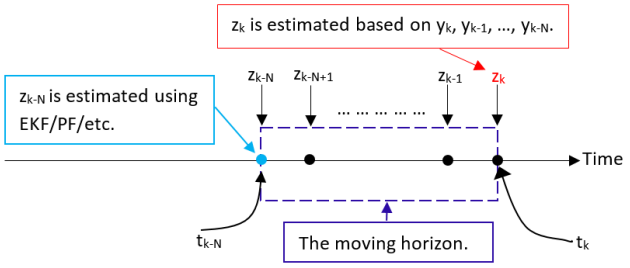


Figure 3: MHE implementation with EKF/PF/etc.

Assume the system given Eq. 21 is output controllable and there exists an equilibrium point such that $\dot{x} = f\left(x^0, u^0, w^{(1),0}, w^{(2),0}, p^0\right) = 0$ and $y^0 = h(x^0) + 0$, $\forall t \geq 0$. Eq. 25 gives the linear model in deviation form. One possibility is to use the linear model to design an optimal controller.

$$\Sigma^{Linear}: \begin{cases} \frac{d}{dt}\Delta x = A\Delta x + B\Delta u + L_1\Delta w^{(1)} + L_2\Delta w^{(2)} \\ \Delta y = C\Delta x + \Delta v \\ A = \frac{\partial}{\partial x} f\Big|_{\left(x^0, u^0, w^{(1),0}, w^{(2),0}, p^0\right)} \\ B = \frac{\partial}{\partial u} f\Big|_{\left(x^0, u^0, w^{(1),0}, w^{(2),0}, p^0\right)} \\ C = \frac{\partial}{\partial x} h\Big|_{\left(x^0\right)} \\ L_1 = \frac{\partial}{\partial w^{(1)}} f\Big|_{\left(x^0, u^0, w^{(1),0}, w^{(2),0}, p^0\right)} \\ L_2 = \frac{\partial}{\partial w^{(2)}} f\Big|_{\left(x^0, u^0, w^{(1),0}, w^{(2),0}, p^0\right)} \\ \Delta x = x - x^0 \\ \Delta u = u - u^0 \\ \Delta w^{(1)} = w^{(1)} - w^{(1),0} \\ \Delta w^{(2)} = w^{(2)} - w^{(2),0} \\ \Delta y = y - y^0 \\ \Delta v = v - 0 \end{cases} \tag{25}$$

$$\frac{d}{dt} z_i = \Delta r_i - \Delta y_i; k = 0,1,\ldots,n_y$$
$$\Delta r_i = r_i - r_i^0$$
$$\Delta y_i = y_i^{filt} - y_i^0$$
$$z = \begin{bmatrix} z_0 \\ z_1 \\ \ldots \\ z_{n_y} \end{bmatrix} \tag{26}$$

[2] 0 and 1 are used both as scalars and (zero and unit) matrices where necessary and reader is advised to pick the correct one.

It is necessary to include integral actions in order to achieve an offset-free reference tracking (the Magic of Integral Control [23]) and better disturbance rejection. See Eq. 26, where r_i is the reference of y_i and y_i^{filt} is the (lowpass) filtered signal of y_i. Refer [24] and [25] for more details. $\Delta w^{(2)}$ is augmented appropriately — see Eq. 28. $\Delta w^{(2)}$ is not controllable by Δu, hence a model for $\Delta w^{(2)}$ is defined such that $\Delta w^{(2)}$ is stable. Also some rate control is included (see Eq. 27).

$$\frac{d}{dt}\Delta u = \tilde{u} \tag{27}$$

$$\frac{d}{dt}\Delta w^{(2)} = \begin{bmatrix} -\underbrace{\beta_0}_{0<\tilde{\beta}_0<1} & 0 \\ 0 & -\underbrace{\beta_0}_{0<\tilde{\beta}_1<1} \end{bmatrix}\Delta w^{(2)} \tag{28}$$

Now, the cost function given in Eq. 29 is minimized subjected to Eqs. 25, 26, 27 and 28. $Q_{\Delta w^{(2)}}$ could be set to zero since it may not be necessary to optimize $\Delta w^{(2)}$.

$$\begin{aligned}
&J_\infty = \underset{t_f\to\infty}{limit} \int_0^{t_f} J(t)dt \\
&J(t) = \tilde{x}Q_{\tilde{x}}\tilde{x}^T + \tilde{u}Q_{\tilde{u}}\tilde{u}^T \\
&\tilde{x} = \begin{bmatrix} \Delta x \\ \Delta w^{(2)} \\ z \\ \Delta u \end{bmatrix} \\
&Q_{\tilde{x}} = \begin{bmatrix} Q_{\Delta x} & 0 & 0 & 0 \\ 0 & Q_{\Delta w^{(2)}} & 0 & 0 \\ 0 & 0 & Q_z & 0 \\ 0 & 0 & 0 & Q_{\Delta u} \end{bmatrix}
\end{aligned} \tag{29}$$

$$\begin{aligned}
&RA + A^TR - RBQ_{\tilde{u}}^{-1}B^TR + Q_{\tilde{x}} = 0 \\
&K_\infty = Q_{\tilde{u}}^{-1}B^TR \\
&\tilde{u} = -K_\infty\tilde{x} = -\begin{bmatrix} K_\infty^1 & K_\infty^2 & K_\infty^3 & K_\infty^4 \end{bmatrix}\begin{bmatrix} \Delta x \\ \Delta w^{(2)} \\ z \\ \Delta u \end{bmatrix} \\
&\quad = -K_\infty^1\Delta x - K_\infty^2\Delta w^{(2)} - K_\infty^3 z - K_\infty^4\Delta u \\
&\Delta u = \int_0^t \tilde{u}(\tau)d\tau
\end{aligned} \tag{30}$$

The analytical solution to unconstrained OCP is given by Eq. 30. On the other hand, it is possible to consider a finite horizon OCP — i.e.,

$$J_{t_f} = \int_0^{t_f<\infty} J(t)dt\ .$$

Unconstrained infinite horizon problem is much faster to implement. $Q_{\tilde{x}}$ and $Q_{\Delta u}$ are carefully tuned. Δx and $\Delta w^{(2)}$ are replaced by their estimates. Alternatively, linear or nonlinear Model Predictive Controller (LMPC/NMPC) can be considered. Assume the current state x_k is known (or estimated). Let, it be necessary to steer output from y_k to y_{k+M_y} ($M_y < \infty$ is the prediction horizon and it is a positive integer) in finite time by manipulating a sequence of control variables $\{u_k, u_{k+1}, \cdots, u_{k+M_u-1}\}$, where M_u is control horizon ($M_u \le M_y$). $\Delta u_i = u_i - u_{i-1}$ for $i = k, k+1, \cdots, k+M_u-1$, $\Delta u_i = 0$ for $i \ge k+M_u$ and u_{-1} is given. There could be many possibilities of choosing a control sequence to achieve the same output transition. In order to pick one of the input trajectories, additional constraints can be imposed on input variables. This is the basic idea in optimal control, where a cost function is defined and minimized with respect to $u(t)$ for $t \in \left[t_k, t_{k+M_y}\right]$. The cost function can be defined in many flavors. Eq. 31 gives an example.

$w^{(1)}$ and $w^{(2)}$ affect the performance of the controller. One way to tackle the problem is to assume $w^{(2)}$ is a random process and $w^{(1)}(t) = w_k^{(1)}$ for $t \in \left[t_k, t_k + M_y.\Delta t\right]$. This makes J_{nmpc} random. We may, among other options, optimize the expected value of the cost function — i.e., $E\{J_{nmpc}\}$. Stochastic optimal control strategies are discussed in [7]. Instead, we consider some estimate of $w_k^{(2)}$ — say $\hat{w}_k^{(2)}$ — and $w^{(2)}(t) = w_k^{(2)}$ for $t \in \left[t_k, t_k + M_y.\Delta t\right]$, in order to obtain a deterministic cost function. Once the OCP is solved (for given x_k or an estimate of it x_k), the optimal control sequence $\{u_k^*, u_{k+1}^*, \cdots, u_{k+M_u-1}^*\}$ as well as $\left\{x_{k+1}^*, x_{k+2}^*, \cdots, x_{k+M_y}^*\right\}$ are found. We set the current control action to be $u_k = u_k^*$ for $t \in [t_k, t_k + \Delta t]$. An estimate of p_k used in the OCP implementation as well.

$$\begin{aligned}
&J_{nmpc} = \sum_{k=0}^{k=M_y-1} \left(y_{k+1} - y_{k+1}^{ref}\right)^T W^y \left(y_{k+1} - y_{k+1}^{ref}\right) \\
&\qquad + \sum_{k=0}^{k=M_u-1} \Delta u_k^T W^u \Delta u_k \\
&x_{k+i} - \\
&\quad f_{k+i-1}\left(\Delta t, x_{k+i-1}, \Delta u_{k+i-1} + u_{k+i-2}, w_k^{(1)}, \hat{w}_k^{(2)}, \hat{p}_k\right) = 0; i = 1,2,\ldots,M_y \\
&u_{k+i} = \Delta u_{k+i} + u_{k+i-1}; i = 0,1,\ldots,M_y - 1 \\
&\Delta u_{k+i} = 0; i = M_u, M_u+1,\ldots,M_y - 1 \\
&x_{k+i} \ge 0; i = 1,2,\ldots,M_y \\
&u_{k+i}^{min} \le u_{k+i} \le u_{k+i}^{max}; i = 0,1,\ldots,M_y - 1
\end{aligned} \tag{31}$$

Analysis and synthesis of OCPs involves controllability and observability analysis. In this paper, we will not discuss algebraic observability and controllability [26][27], instead we exploit the structural observability and controllability [28][29][6]. Observability of the noise-free system must be fulfilled first. Consider the noise-free form of Eq. 21 (by setting $\varepsilon = v = 0$) given in Eq. 35.

$$\Sigma_{nf}: \begin{cases} \dot{x}_{nf} = f\left(x_{nf}, u, w^{(1)}, w_{nf}^{(2)}, p_{nf}\right) \\ \dot{p}_{nf} = 0 \\ \dot{w}_{nf}^{(2)} = \psi\left(w_{nf}^{(2)}, \beta\right) \\ y_{nf} = h\left(x_{nf}\right) \end{cases} \tag{32}$$

The augmented state $x_{nf}^{Aug}(t)$ moves on its state space, starting at $x_{nf}^{Aug}(t=0)$. The trajectory of $x_{nf}^{Aug}(t)$ is shaped by $u(t)$ and $w^{(1)}(t)$. The output trajectory is given by $y_{nf}(t) = h\left(x_{nf}(t)\right)$. If (locally) no two distinct initial conditions give identical output trajectories for any admissible $u_{nf}(t)$ and $w_{nf}^{(1)}(t)$, then we say that the state are (locally) distinguishable. In other words:

$$h\left(x_{nf}^1\right) \neq h\left(x_{nf}^2\right) \Rightarrow x_{nf}^1 \neq x_{nf}^2, w_{nf}^{(2),1} \neq w_{nf}^{(2),2}, p_{nf}^1 \neq p_{nf}^2$$

State distinguishability property implies state observability. Note that $u_{nf}(t)$ and $w_{nf}^{(1)}(t)$ affect observability. If there exists an admissible control $u_{nf}(t)$ such that the state transition from any x_{nf}^1 to any state x_{nf}^2 is possible, then we say that the state is controllable ($w_{nf}^{(1)}(t)$ and $w_{nf}^{(2)}(t)$ affect controllability). The reader is refereed to [27][30] for a detailed discussion on nonlinear observability and controllability. Often, implementing algebraic tests for observability and controllability analysis on large-scale complex systems are tedious or even practically impossible. A practical solution is first to analyze large-scale systems for structural observability and controllability, followed by algebraic tests for structurally decomposed subsystems as necessary. Structural observability (controllability) gives a necessary condition for observability (controllability). Interestingly, there is a graph-theoretic analogy for structural analysis [6].[3]

Structural observable analysis involves structural dependencies among state and measurement variables and those dependencies are encoded into a directed graph G_o (directed graph = digraph). Nodes are the elements of the vectors x_{nf}, $w_{nf}^{(2)}$, p_{nf} and p_{nf}. E.g., $x_{nf,i}$ is the i-th element of x_{nf} and $x_{nf,i}$ is a node. Edges are defined in the following way:

- $x_{nf,i} \rightarrow x_{nf,j}$ exists if $\partial f_i / \partial x_{nf,j} \neq 0$;
- $x_{nf,i} \rightarrow w_{nf,j}^{(2)}$ exists if $\partial f_i / w_{nf,j}^{(2)} \neq 0$;
- $x_{nf,i} \rightarrow p_{nf,j}$ exists if $\partial f_i / \partial p_{nf,j} \neq 0$;
- $x_{nf,i} \rightarrow p_{nf,j}$ exists if $\partial f_i / \partial p_{nf,j} \neq 0$;
- $w_{nf,i}^{(2)} \rightarrow w_{nf,j}^{(2)}$ exists if $\partial \psi_i / \partial w_{nf,j}^{(2)} \neq 0$;
- $y_{nf,i} \rightarrow x_{nf,j}$ exists if $\partial h_i / \partial x_{nf,j} \neq 0$.

Note that $a \rightarrow b$ denotes a directed edge from a to b. Once the digraph is created, we can analyze for structural observability. It is a must that all state variables are reachable from outputs (output connectivity) – i.e. if there is at least one state variable such that there is no directed path from any of the output variables to that state's node, then the system is not structurally observable. However, this is a necessary condition for structural observability. A necessary and sufficient condition is that iff G_o is spanned by cacti [28], then the system is structurally observable. A cactus contains a directed path, called a stem, starting from an output the root and ends at a state variable the top and cycles, called buds, which are attached to the stem and/or to other buds. See Fig. 4 for an example.

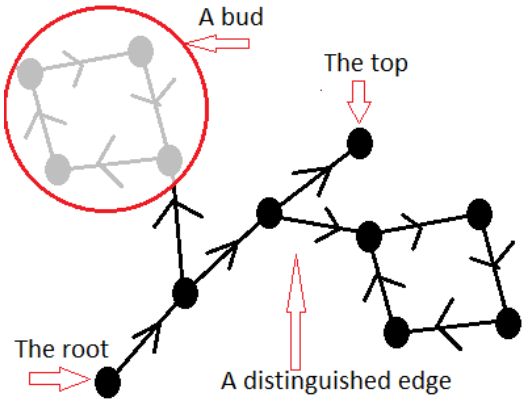


Figure 4: A cactus with two buds.

Digraphs for structural controllability (with respect to control variables) analysis are created in a similar fashion. Nodes are the elements of the vectors x_{nf}, u and y_{nf}. Let G_c be the digraph for structural controllability analysis. The edges are defined in the following way:

- $x_{nf,j} \rightarrow x_{nf,i}$ exists if $\partial f_i / \partial x_{nf,j} \neq 0$;
- $u_j \rightarrow x_{nf,i}$ exists if $\partial f_i / u_j \neq 0$;
- $x_{nf,i} \rightarrow y_{nf,j}$ exists if $\partial h_j / \partial x_{nf,i} \neq 0$.

Iff G_c is spanned by cacti, then the system is structurally output controllable. Similarly, state controllability can be defined. It is also possible to define structural controllability with respect to process noise $w_{nf}^{(2)}$ for stochastic controllability. Stochastic controllability affects convergence characteristic of the state estimation process [16][32]. In some cases, structural observability (controllability) is sufficient for algebraic observability (controllability). For example, the system we consider in this paper is structurally observability and controllability and also structural properties provides sufficiency.

IV. Software Implementation

In this paper, usage of free software tools are emphasized. Python which is a powerful tool for technical computing is used as the scripting language. There are many Python packages which support, among others, various aspects systems and control engineering applications such as scipy, numpy, matplotlib, assimulo, casadi, pygraphviz, networkx, etc. For modeling of large-scale complex dynamic systems, Modelica specifications provide a better systematic approach. Several tools offer interfaces between Modelica and Python, for examples OpenModelica and JModelica.org. Modelica standards mainly focuses on modeling and simulation, however there exist an extension to Modelica, Optimica [33], which can handle OCPs. OpenModelica and JModelica.org support Optimica standards as well. CasADi — available as a Python package — is a symbolic framework for numerical optimization and it is conveniently possible to cast a general OCP using CasADi. Also, it is always possible to solve a general OCP via SciPy.

First, a Modelica[4] package is created which contains the copper leaching process model EW and EW_Init (the steady

[3] A way of automating structural analysis in Python, using Modelica-JModelica.org-CasADi, is given in [31].

[4] See more about Modelica http://book.xogeny.com/.

state model) which extends EW.[5] The key point here is that we can encode large-scale complex systems systematically using Modelica standards — in other words exploiting the modeling power in Modelica. To calculate a steady state, EW_Init is compiled using JModelica.org — the pymodelica package is used for compiling — and the compiled model is imported back, as a JMUModel object and called it init_model, to Python through pyjmi. Calling the method initialize() initializes EW_Init and thereby, a steady state is found. The method initialize() uses IPOPT solver [34]. There are many other alternatives which can be used in steady state calculation for instances CasADi and SciPy. EW is also used to generate necessary digraphs for structural observability and controllability analysis. This is done via CasADi by importing Modelica model into Python as symbolic DAEs, processing them (e.g. index reduction, etc.), generating and analyzing digraphs using networkx package.

The module scipy.optimize is used in the implementations of optimal control and moving horizon estimate. For example, scipy.optimize.fmin_cobyla() is a nonlinear optimizer which can be used for constrained optimization. An alternatively is IPOPT optimizer (through CasADi). IPOPT-CasADi combination needs lesser computational time as compared to fmin_cobyla()-Python. The EKF is straight forward to automate.

V. Results

The volume of electrowinning tanks V_{ew} determines the rate of change of the chemical compositions in electrowinning tanks. V_{ew} is not precisely known. However, the experience shows that the time constant is around 2 hours, hence V_{ew} is chosen accordingly. Note that variables are scaled with respect to an equilibrium point, thereby scaled variables have their nominal values equal to ones. The entire analysis is based on scaled variables.

Two of the eigenvalues of matrix A are zeros and others are complex with real negative parts. Zero eigenvalues corresponds to V_{ed} and V_{em}. Due to the presence of zero eigenvalues, it is necessary to have shorter sampling time. $\Delta t = 0.25\ [min]$ is selected.

A. Structural analysis

Based on available measurements ($n_y = 4$) it is expected to estimate all state variables ($n_x = 8$) and additionally, if possible, disturbance variables ($n_{w^{(2)}} = 2$) and parameters ($n_p = 2$). Fig. 5 depicts the digraph for structural observability analysis. There exists a spanning cacti covering all vertices, hence the system is structurally observable. There could be more than one spanning cacti. Note that even though $\beta_{\dot{V}_{ew2m}} = \beta_{\dot{V}_{ed2w}} = 0$ (in Eqs. 13 and 14), the augmented model maintains structural observability.

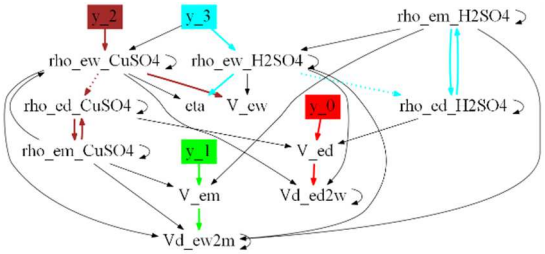


Figure 5: Structural observability analysis: digraph is spanned by cacti covering all nodes.

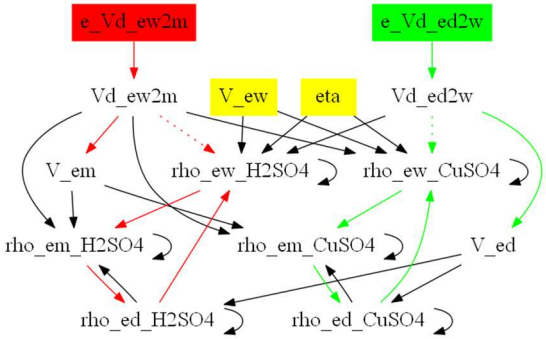


Figure 6: Structural controllability analysis w.r.t. process noise variables. Nodes related to parameters are colored in yellow.

The digraph given in Fig. 6 is used to analyze for the controllability of state variables with respect to process noise variables. The objective is to check whether unstable modes are excited enough by process noise variables $\varepsilon_{\dot{V}_{ew2m}}$ and $\varepsilon_{\dot{V}_{ed2w}}$. It is seen that augmented state variables, except the state variables related to parameters, are controllable by process noise variables. If there is any unstable mode which is not controllable by process noise, then it is necessary to add some fictitious noise to it. It is always advisable to add fictitious noise processes to all state equations including Eqs. 11 and 12 (these equations links with model parameters). According to Fig. 7, it is clear that the system is structurally output controllable. In this case, the digraph can be divided into two sets S_1 and S_2:

$$S_1 = \{y_0, y_1, V_{ed}, V_{em}, u_0, u_1\}$$

$$S_2 = \{y_2, y_3, \rho_{ed,CuSO_4}, \rho_{ew,CuSO_4}, \rho_{em,CuSO_4}, \rho_{ed,H_2SO_4}, \rho_{ew,H_2SO_4}, \rho_{em,H_2SO_4}, u_2, u_3\}$$

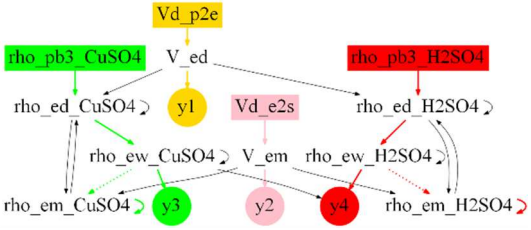


Figure 7: Structural (output) controllability analysis w.r.t. input variables.

There are no directed edges starting from S_2 and ending at S_1, which means the OCP problem can be considered as two

[5] In EW, variable are scaled with respect to their nominal values and consequently, scaled variables vary around 1.

sub-OCPs, say OCP1 and OCP2, and such partitioning of state variables is not always possible. The OCP1 related to $y_0 - y_1$ control using $u_0 - u_1$ and OCP1 can be solved completely independent from OCP2. Since there are directed edges from S_1 to S_2, OCP2 depends on OCP1. The subsystem which corresponds to OCP1 is a linear time invariant system with has two zero eigenvalues. On the other hand, OCP2 deals with a sluggish system of ODEs as compared to OCP1.

B. *State estimation and optimal control*

The model (Eq. 21) is simulated for given $w^{(1)}$ (see Fig. 22), $w^{(2)}$, and p with the optimal controller given in Eq. 29. EKF and MHE is implemented for simulated data. Figs. 8 and 9 show estimates for η and V_{ew} using a shorter moving horizon $N_{mhe} = 2$.

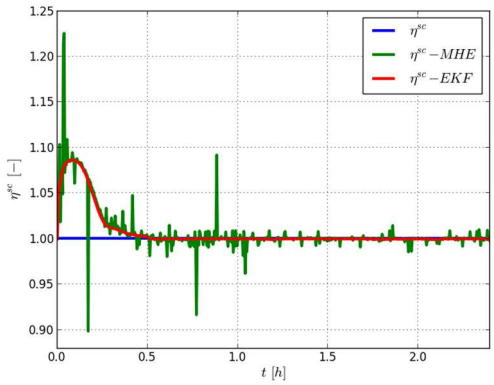


Figure 8: Estimation of scaled η with $N_{mhe} = 2$.

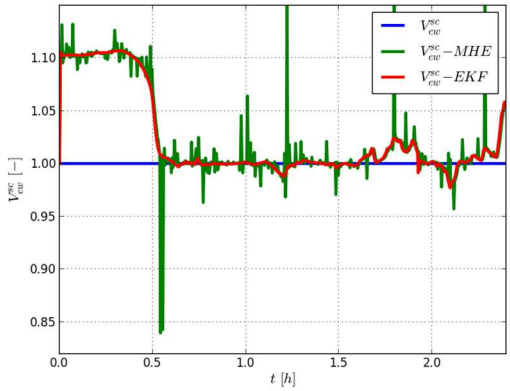


Figure 9: Estimation of scaled V_{ew} with $N_{mhe} = 2$.

Note that a modified EKF with a stability correction is used here. Estimates for chemical compositions and dilution and mixing tanks' volumes are given in Figs. 11, 12, 13, 14, 15, 16, and 17 while Figs. 18 and 19 are for the estimation of disturbance variables. The controller makes outputs to follow the reference trajectories in the presence of both known and unknown disturbances — see Figs. 20 and 21.

In general, initial state estimate is unknown. However, we know that all state variables must be positive. It is important to check the stability characteristic of the estimates with respect to initial state estimate. A Monte Carlo simulation is done by choosing initial state randomly by means of uniform probability distributions. It is observed from Monte Carlo simulations — by picking initial state estimates within $\pm 50\%$ of their nominal values —, the estimates always converging for true values. See Figs. 24 and 25. Similar Monte Carlo simulation is done to check the effect of α. See Figs. 26 and 27. In Fig. 27, all curves converge to true value except one which corresponds to $\alpha = 0$. This is an important observation. The conventional EKF fails (i.e. $\alpha = 0$) even though the augmented state is observable. A properly tuned Moving Horizon Estimate gives comparatively better stable estimates. One of the disadvantage with the Moving Horizon Estimate is that it demand more computer resources. For $N_{mhe} = 2$, the time taken per each sample time is around 1 $[min]$ while 20 $[min]$ for $N_{mhe} = 20$. See Figs. 30 and 31 for the results when $N_{mhe} = 20$.

VI. Conclusion and Future Work

We have demonstrated how to use available free software and analytical tools in automating control and estimation problems related to large-scale complex dynamic systems by considering the copper leaching process at Glencore Nikkelverk, Kristiansand as a case study. The case study is a success story of exploiting structural analysis, graph-theory, Modelica and Python in handling large-scale complex control systems. The case study mainly concern optimal control and state-parameter-disturbance estimation problems and these problems connect with controllability and observability analysis. By mapping system structure appropriately into directed graphs (so-called digraphs) makes easy to analyze for structural controllability/observability. We have suggested a simple way of creating system digraphs and structural analysis using Modelica and Python.

The electrowinning process is a subsystem of the copper leaching process. A mechanistic model for the electrowinning process is available and it is considered in our analysis. The model has 8 state variables, 2 unknown disturbances and 2 unknown parameters and all of them are estimated based on available 2 level and 2 composition measurements. Two state estimators are used: the Extended Kalman Filter and the Moving Horizon Estimate. The Moving Horizon Estimator can handle constraints (e.g. chemical composition is always positive) while the Extended Kalman Filter cannot. Also, the Extended Kalman Filter may give poor stability properties. We implemented a modified Extended Kalman Filter, by replacing system matrix A by $A + \alpha I$ where α is small positive number, which gives improved results as the Moving Horizon Estimate's results. We have shown that the Moving Horizon Estimate gives better results in sense of estimator's stability and robustness proving that the estimator is tuned properly as compared to conventional Extended Kalman Filter without including stability correction.

Before implementing any state estimator in order to estimate disturbance variables, it is essential to define dynamic models for them. There could be many possibilities of doing so. Disturbance models should be defined in such way that augmented model is structurally observable. We have shown an easy and efficient procedure to pick suitable disturbance models using a graph-theocratic approach. On the other hand, to solve the control problem, an optimal controller with integral action is implemented. The model used in controller analysis and synthesis is an augmented model and its state variables consists of both state variables in the original system and unknown disturbance variables. Here, disturbance models are defined such that unstable modes of the augmented system is controllable with respect to input

variables and again, we have used the help of structural controllability analysis.

In the software implementation, we only concerned free software tools. Modeling is done in Modelica and Modelica models are used in structural observability and controllability analysis by importing Modelica models into Python via the JModelica.org-CasADi interface. Python packages NetworkX and PyGraphviz are used for graph-theocratic analysis and visualization respectively. To solve nonlinear optimization problems — which encounter in optimal control and state estimation using the Moving Horizon Estimate — constrained optimizers available in SciPy, such as fmin_cobyla, can be used. Alternatively, Ipopt (Interior Point OPTimizer) optimizer may be used and for instance, it can be accessed through CasADi.

Several suggestions are made for possible future work. An immediate extension to our work is to validate the results based on real process data. For example, state-parameter-disturbance estimation using real process data should be done. It is seen that chemical compositions in electrowinning tanks and the mixing tank are more or less equal according to model simulations, hence it is possible to consider some reduced order model in the analysis. We have considered the chemical compositions in the dilution tank as two of the control signals to control chemical compositions in the electrowinning tanks. In reality, chemical compositions in the dilution tank are manipulated by adding copper oxide and sulfuric acid at two remote locations within the copper plant and this adds imminent delays in the control action. It is necessary to include delays in the analysis. Finally, it is of interest to integrate an optimal controller to the real process and to implement an online state-parameter-disturbance estimator.

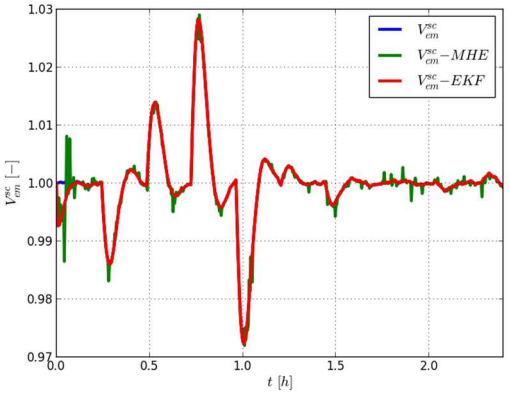


Figure 10: Estimation of scaled V_{em} with $N_{mhe} = 2$.

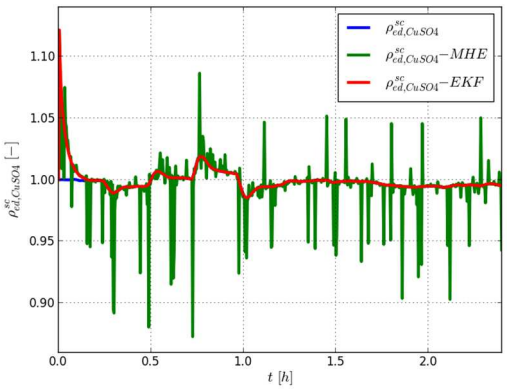


Figure 11: Estimation of scaled $\rho_{ed,CuSO4}$ with $N_{mhe} = 2$.

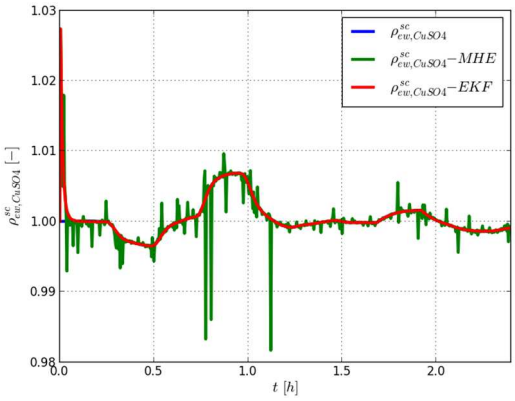


Figure 12: Estimation of scaled $\rho_{ew,CuSO4}$ with $N_{mhe} = 2$.

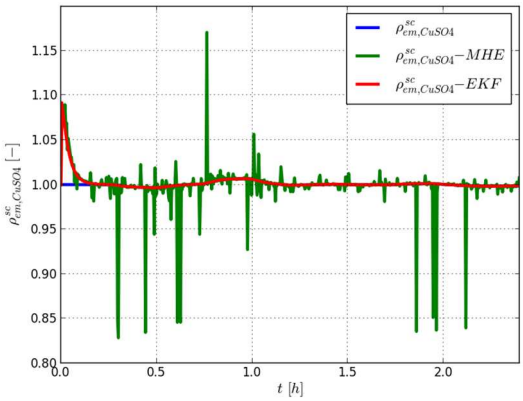


Figure 13: Estimation of scaled $\rho_{em,CuSO4}$ with $N_{mhe} = 2$.

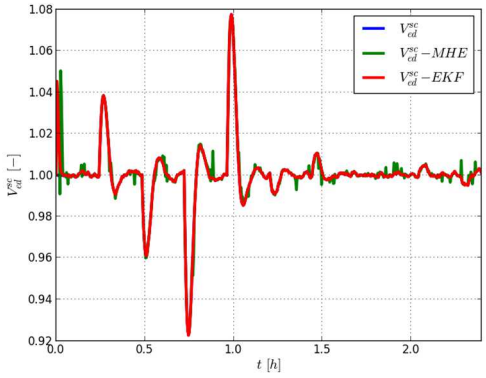


Figure 14: Estimation of scaled V_{ed} with $N_{mhe} = 2$.

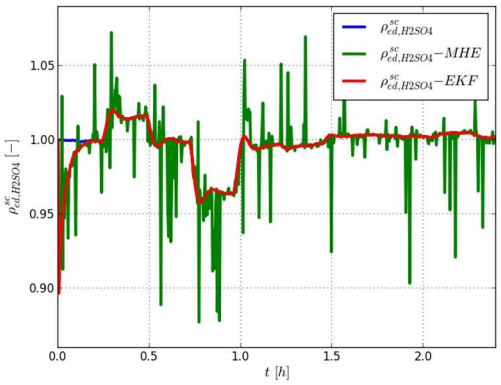


Figure 15: Estimation of scaled $\rho_{ed,H2SO4}$ with $N_{mhe} = 2$.

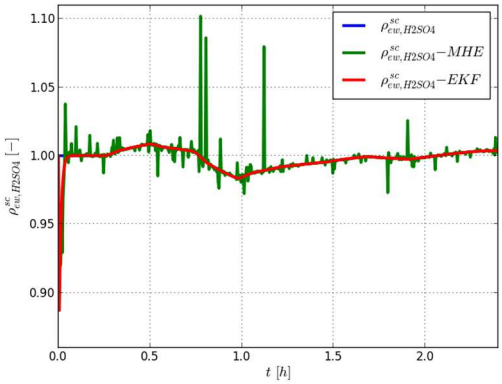


Figure 16: Estimation of scaled $\rho_{ew,H2SO4}$ with $N_{mhe} = 2$.

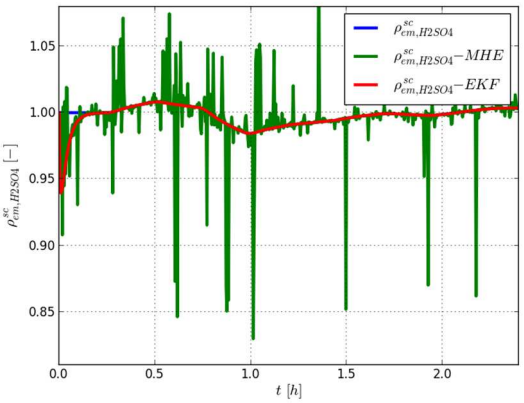


Figure 17: Estimation of scaled $\rho_{em,H2SO4}$ with $N_{mhe} = 2$.

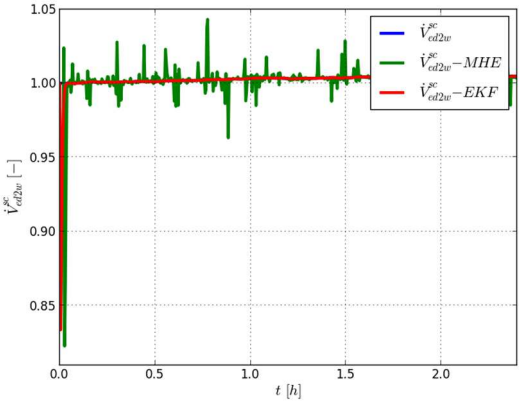


Figure 18: Estimation of scaled $\dot{V}_{ed2w}$ with $N_{mhe} = 2$.

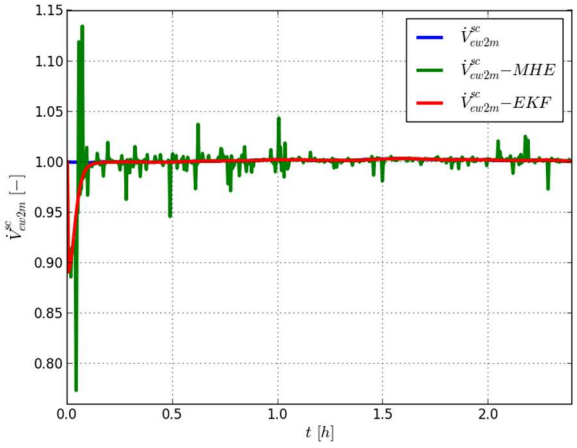


Figure 19: Estimation of scaled $\dot{V}_{ew2m}$ with $N_{mhe} = 2$.

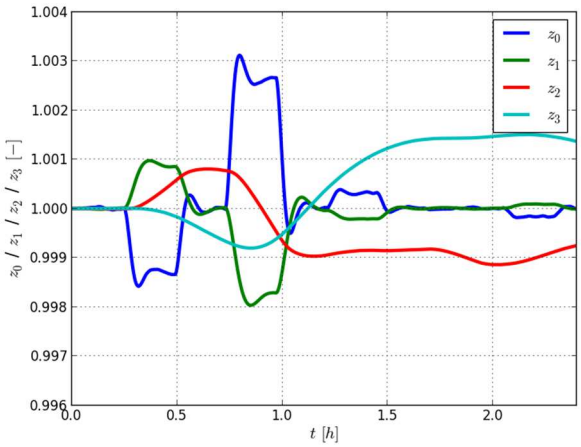


Figure 20: Inputs.

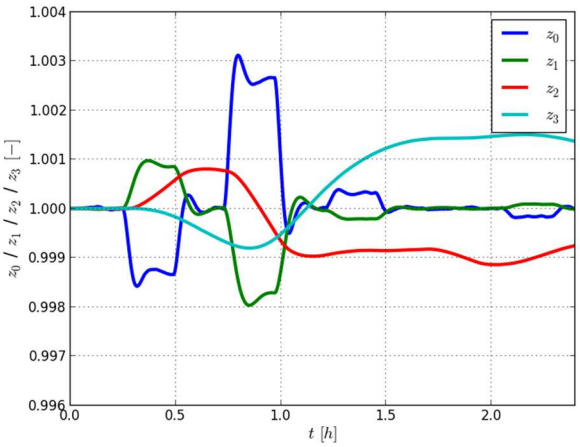


Figure 23: Integral actions.

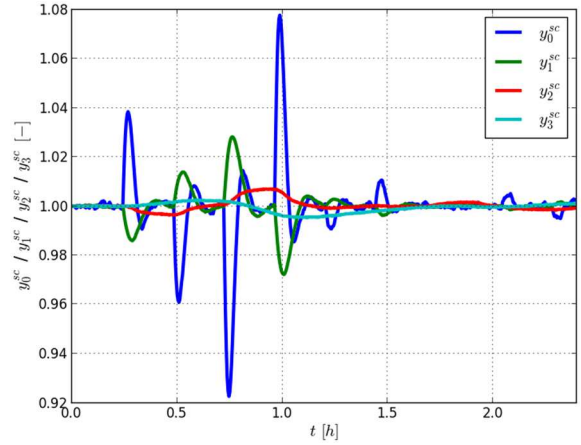


Figure 21: Outputs.

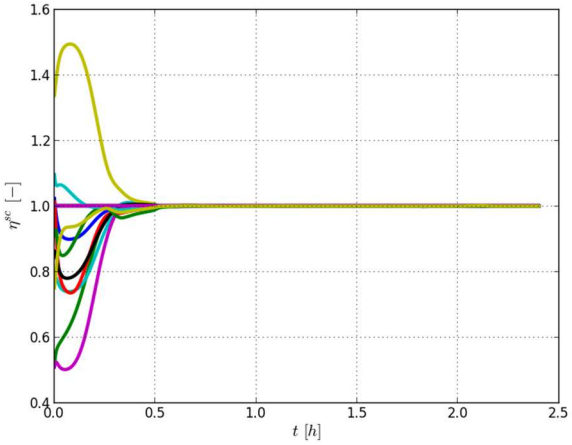


Figure 24: Sensitivity of initial state estimates for scaled η estimation using EKF.

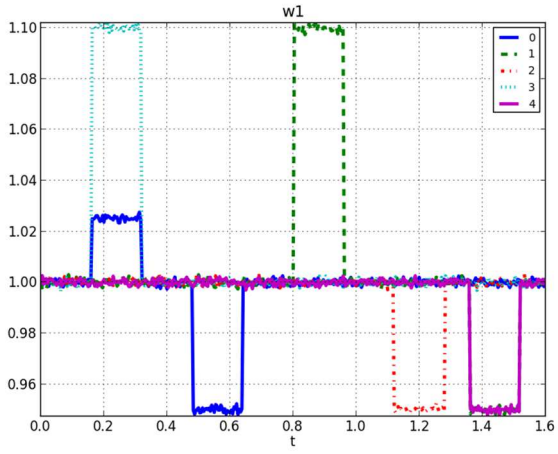


Figure 22: Known disturbances.

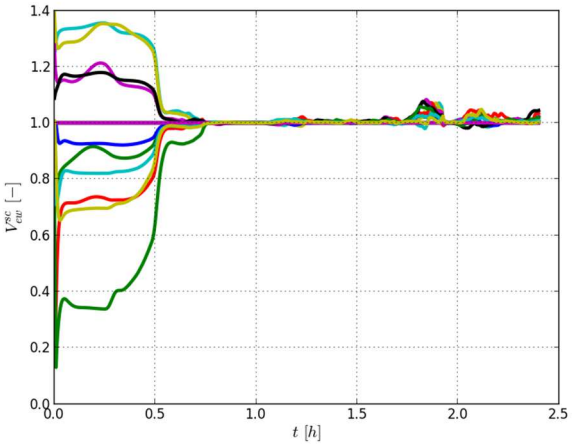


Figure 25: Sensitivity of initial state estimates for scaled V_{ew} estimation using EKF.

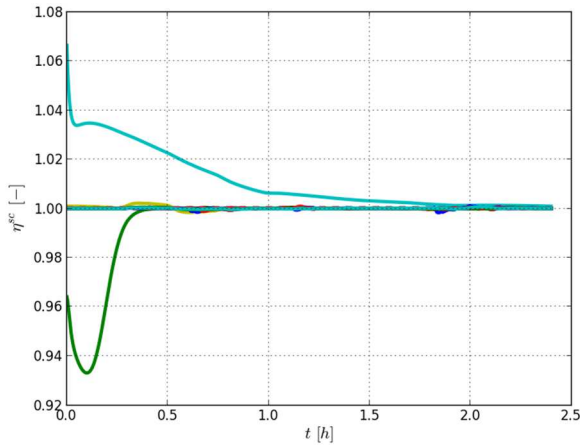


Figure 26: Sensitivity of α for scaled η estimation using EKF.

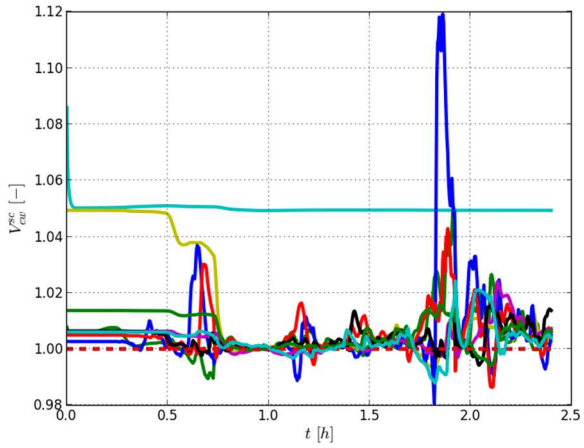


Figure 27: Sensitivity of α for scaled V_{ew} estimation using EKF.

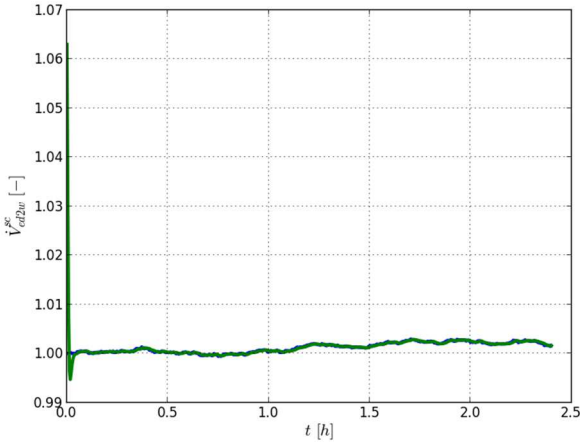


Figure 28: Sensitivity of β for scaled $\dot{V}_{ed2w}$ estimation using EKF.

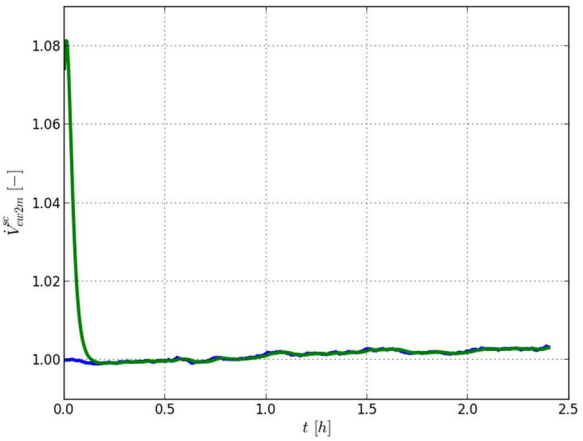


Figure 29: Sensitivity of β for scaled $\dot{V}_{ew2m}$ estimation using EKF.

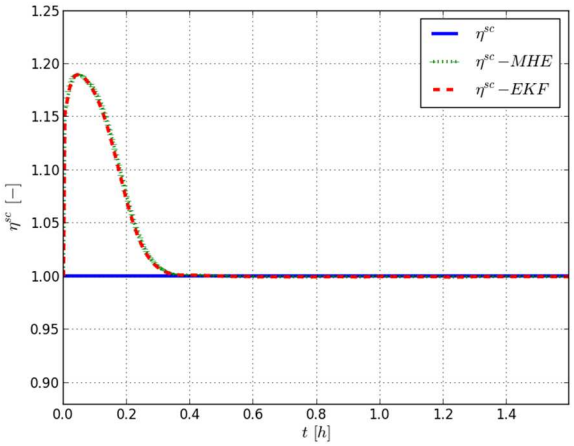


Figure 30: Estimation of η with $N_{mhe} = 20$.

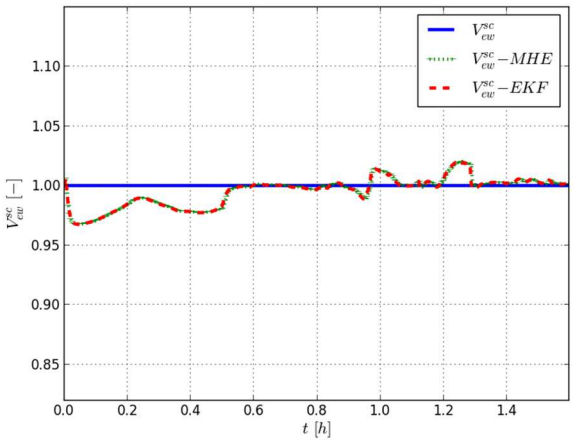


Figure 31: Estimation of V_{ew} with $N_{mhe} = 20$.

References

[1] A. E. Bryson, Applied optimal control: optimization, estimation and control, CRC Press, 1975.

[2] B. Lie, T. A. Hauge, Modeling of an industrial copper leaching and electrowinning process, with validation against experimental data, in: Proceedings SIMS, 2008, pp. 7–8.

[3] F. Allgöwer, R. Findeisen, C. Ebenbauer, Nonlinear model predictive control (2000). http://ifatwww.et.uni-magdeburg.de/syst/about_us/people/findeisen/papers/EOLSS.pdf

[4] P. Fritzson, Introduction to modeling and simulation of technical and physical systems with Modelica, John Wiley & Sons, 2011.

[5] J. Andersson, J. Åkesson, M. Diehl, Casadi: a symbolic package for automatic differentiation and optimal control, in: Recent Advances in Algorithmic Differentiation, Springer, 2012, pp. 297–307.

[6] K. J. Reinschke, Multivariable control: a graph theoretic approach, Springer-Verlag, 1988.

[7] D. P. Bertsekas, Dynamic programming and optimal control, Vol. 1, Athena Scientific Belmont, Massachusetts, 1996.

[8] L. T. Biegler, Nonlinear programming: concepts, algorithms, and applications to chemical processes, Vol. 10, SIAM, 2010.

[9] L. Magni, D. M. Raimondo, F. Allgöwer, Nonlinear model predictive control, Springer, 2009.

[10] D. Simon, Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches, Wiley-Interscience, 2006.

[11] R. Brown, P. Hwang, Introduction to Random Signals and Applied Kalman Filtering – with MATLAB exercises and solutions, John Wiley and Sons, 1997.

[12] A. Gelb, Applied optimal estimation, The M.I.T. press, 2001.

[13] A. H. Jazwinski, Stochastic processes and filtering theory, Courier Corporation, 2007.

[14] A. Jazwinski, Limited memory optimal filtering, IEEE Transactions on Automatic Control 13 (5) (1968) 558–563.

[15] K. Reif, F. Sonnemann, R. Unbehauen, Modification of the extended kalman filter with an additive term of instability, in: Decision and Control, 1996., Proceedings of the 35th IEEE Conference on, Vol. 4, IEEE, 1996, pp. 4058–4059.

[16] R. J. Fitzgerald, Divergence of the kalman filter, Automatic Control, IEEE Transactions on 16 (6) (1971) 736–747.

[17] L. Ljung, Asymptotic behavior of the extended kalman filter as a parameter estimator for linear systems, Automatic Control, IEEE Transactions on 24 (1) (1979) 36–50.

[18] B. Bona, R. J. Smay, Optimum reset of ship's inertial navigation system, Aerospace and Electronic Systems, IEEE Transactions on AES-2 (4) (1966) 409–414.

[19] A. Doucet, S. Godsill, C. Andrieu, On sequential monte carlo sampling methods for bayesian filtering, Statistics and computing 10 (3) (2000) 197–208.

[20] N. Gordon, B. Ristic, S. Arulampalam, Beyond the kalman filter: Particle filters for tracking applications, Artech House, London.

[21] A. Smith, A. Doucet, N. de Freitas, N. Gordon, Sequential Monte Carlo methods in practice, Springer Science & Business Media, 2013.

[22] D. F. Bizup, D. E. Brown, The over-extended kalman filter-don't use it!, in: Proceedings of the Sixth International Conference of Information Fusion, Vol. 1, 2003, pp. 40–46.

[23] K. J. Åström, Introduction to Control, Department of automatic control, Lund Institute of Technology, 2004.

[24] P. C. Young, J. Willems, An approach to the linear multivariable servomechanism problemâ€, International journal of control 15 (5) (1972) 961–979.

[25] Y.-P. Shih, Integral action in the optimal control of linear systems with quadratic performance index, Industrial & Engineering Chemistry Fundamentals 9 (1) (1970) 35–37.

[26] A. Isidori, Nonlinear control systems, Springer Science & Business Media, 1995.

[27] R. Hermann, A. J. Krener, Nonlinear controllability and observability, IEEE Transactions on automatic control 22 (5) (1977) 728–740.

[28] C. T. Lin, Structural controllability, Automatic Control, IEEE Transactions on 19 (3) (1974) 201–208.

[29] Y.-Y. Liu, J.-J. Slotine, A.-L. Barabási, Observability of complex systems, Proceedings of the National Academy of Sciences 110 (7) (2013) 2460–2465.

[30] M. R. James, Controllability and observability of nonlinear systems., Tech. rep., Mathematics Department and Systems Research Center, University of Maryland (1987).

[31] M. A. S. Perera, B. Lie, C. F. Pfeiffer, Structural Observability Analysis of Large Scale Systems Using Modelica and Python, Modeling, Identification and Control 36 (1) (2015) 53–65. doi:10.4173/mic.2015.1.4.

[32] J. E. Potter, A matrix equation arising in statistical filter theory, Tech. rep., National Aeronautics and Space Administration (1965).

[33] J. Åkesson, K.-E. Årzén, M. Gäfvert, T. Bergdahl, H. Tummescheit, Modeling and optimization with optimica and jmodelica.Org languages and tools for solving large-scale dynamic optimization problems, Computers & Chemical Engineering 34 (11) (2010) 1737–1749.

M. Anushka S. Perera was born in 1982 in Negombo, Sri Lanka. He received his bachelor's degree in Chemical and Process Engineering from University of Moratuwa, Sri Lanka in 2007 and master's degree in Systems and Control Engineering from Telemark University College, Porsgrunn, Norway (at time it was called Telemark University College) in 2012. He is currently a PhD candidate at the University College of Southeast Norway, Porsgrunn. His main research interests are estimation and control of large-scale complex control systems.

Tor Anders Hauge obtained his PhD from Telemark University College, Porsgrunn, Norway in 2003. He has worked as adjunct Professor at University of Agder's mechatronics group during the course of 2008-2011. He has been working as a control engineer from 2003 at Glencore Nikkelverk, Kristiansand, Norway and his current position is senior control engineer.

Carlos Fernando Pfeiffer obtained his bachelor's degree in Chemical and Systems Engineering from ITESM, Monterrey, Mexico in 1987, master's degree in Control Engineering degree from the same institution in 1993 and the PhD from the University of Texas at Austin in 1999. The topic of his dissertation was heterogeneous control laws for nonlinear systems. He worked as a Process Engineering at the Advanced Process Research and Development Center at Motorola, Austin, from 1998 to 2001, implementing advanced control and monitoring for the lithography process, etch process and chemical mechanical planarization (CMP). Carlos F. Pfeiffer was a Professor at the Computer Science Department at ITESM, Monterry, Mexico from 2001 to 2010. From 2011 to date, he is faculty at Telemark University College, now University College of Southeast Norway. His current fields of teaching and research are Advanced Process Control, Process Modeling and Optimization, Thermodynamics, Computer Vision, Pattern recognition. Present research projects: Human Behavior modeling for Smart House and Welfare technology, thermodynamic Characterization of CO2 in polymer solutions, nonlinear process control.

[34] A. Wächter, L. T. Biegler, On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming, Mathematical programming 106 (1) (2006) 25–57.

Doctoral dissertation no. 6
2016

—

State Estimation and Optimal Control of an Industrial Copper Electrowinning

Dissertation for the degree of Ph.D

—

Magamage Anushka Sampath Perera

—

ISBN: 978-82-7206-417-3 (print)
ISBN: 978-82-7206-418-0 (online)

usn.no