NTNU
Norwegian University of
Science and Technology

Master's thesis

2019

David Pérez Piñeiro

Master's thesis

NTNU
Norwegian University of
Science and Technology
Faculty of Engineering
Department of Energy and Process Engineering

David Pérez Piñeiro

Self-optimizing control of a continuous-flow pharmaceutical manufacturing plant

August 2019

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Self-optimizing control of a continuous-flow pharmaceutical manufacturing plant

David Pérez Piñeiro

Natural Gas Technology
Submission date: August 2019
Supervisor: Truls Gundersen, EPT
Co-supervisor: Johannes Jäschke, IKP
Richard D. Braatz, MIT

Norwegian University of Science and Technology
Department of Energy and Process Engineering

Abstract

The production of pharmaceuticals is typically done using batch processing at multiple facilities. Drawbacks of the current approach include long production times and the potential for drug shortages due to supply chain interruptions and quality control issues. Other major challenges facing the pharmaceutical industry today include the need to respond to sudden changes in demand due to epidemics or pandemics, lower drug production costs, and reduce waste generation. To address these issues, research efforts in academia and industry over the last decade have been oriented towards the development of end-to-end continuous-flow processes for drug production.

This work considers the plantwide control design for a portable pharmaceutical platform for the on-demand continuous-flow production of atropine, which is an active pharmaceutical ingredient used in the treatment of heart rhythm problems. The existing physical platform developed at the Massachusetts Institute of Technology has a regulatory control layer that controls key states such as reactor temperatures to specified setpoints. However, the design of an upper supervisory control layer to meet plantwide control objectives such as minimization of the environmental factor is needed. In this work, we explore the application of self-optimizing control in the design of the supervisory control layer. The idea is to achieve near-optimal operation by tracking carefully selected controlled variables to constant setpoints without the need to re-optimize online the system when disturbances occur. Screening methodologies for selection of self-optimizing controlled variables based on local and global approaches were developed. Furthermore, decentralized PID control and model predictive control strategies were developed for the optimal operation and control of the system.

It was found that a constant setpoint control strategy of linear combinations of 6 concentration measurements results in near-optimal operation in the face of uncertainty. The uncertainty considered in this work includes process disturbances, parametric model uncertainty and sensor noise.

Acknowledgements

I am deeply grateful to my thesis advisors at the Norwegian University of Science and Technology, Truls Gundersen and Johannes Jäschke, for providing me with a good balance of direction and freedom while conducting my research. Furthermore, I want to thank Richard D. Braatz for accepting me as a visiting student and guiding my work during my research stay at the Massachusetts Institute of Technology. Our many conversations gave me perspective on my research and invaluable career advice.

Special thanks go to my project collaborator at the Braatz lab, Anastasia Nikolakopoulou. Your understanding of the atropine process and academic advice have been instrumental in the completion of this work. I want to extend this acknowledgement to all my labmates, for creating an inspiring environment during my stay at MIT. I am specially thankful to my officemates during this period: Domenico, Fabian and Max. On a less serious note, I want to thank my friends from MIT VISTA for all of the evenings spent together at the Muddy Charles Pub, the days sailing on the Charles River, the BBQs and much more.

Moving now to Norway, I want to thank my friends Roberto and Jairo for being a sort of a family away from home for the past three years, and for the many more that are to come.

Looking back on my days as an undergraduate student at the University of Vigo, I want to thank Eduardo Liz, who was an inspiring professor of calculus and linear algebra. I am sure that part of the motivation that has propelled me to continue further in my studies is due to professors like you.

Chitti, thank you for being a constant source of love, support and inspiration everyday.

Mom and dad, for all the unyielding support you have giving me over the years, I am forever grateful.

I gratefully acknowledge the financial support received from “La Caixa” Foundation during my master studies at NTNU and my research stay at MIT.

Boston, August 2019

David Pérez Piñeiro

Table of Contents

Abstract i

Acknowledgements iii

List of Tables vii

List of Figures x

Nomenclature xiv

1 Introduction 1

1.1 Project background 1
1.2 Motivation 1
1.3 Objectives 2
1.4 Organization 2

2 Technical background 3

2.1 Introduction to control system design 3
2.2 Principles of hierarchical control 4
2.3 Plantwide control 5
2.4 Self-optimizing control 7
2.4.1 Problem formulation 8
2.4.2 Brute force methods 11
2.4.3 Local methods 12
2.4.4 Global methods 18
2.4.5 The problem of measurement subset selection 22
2.4.6 The problem of handling active set changes 25
2.5 Controller design in multivariable plants 26
2.5.1 Decentralized PID control 27
2.5.2 Model predictive control 30
2.6 Continuous-flow production of pharmaceuticals 31

3 Case study and methodology 33
3.1 The case study . . . 33
3.1.1 Process description . . . 33
3.1.2 Mathematical model . . . 35
3.1.3 Operational objective . . . 39
3.1.4 Challenges of optimal operation . . . 39
3.2 Controlled variable selection methodology . . . 40
3.2.1 Preliminary controlled variable selection . . . 41
3.2.2 Steady-state validation . . . 43
3.2.3 Dynamic validation . . . 44
3.3 Controller design methodology . . . 45
3.3.1 Decentralized PI control . . . 45
3.3.2 Quadratic dynamic matrix control (QDMC) . . . 45

4 Results and conclusions 47
4.1 Optimal operation for nominal conditions . . . 47
4.2 Preliminary controlled variable selection . . . 48
4.3 Steady-state validation . . . 53
4.4 Dynamic validation . . . 59
4.5 Overall conclusions . . . 62
4.6 Evaluation of objectives . . . 63
4.7 Directions for further research . . . 63

Bibliography 63

A MATLAB code 69
A.1 MIQP solver for local self-optimizing control . . . 69
A.2 MIQP solver for global self-optimizing control . . . 71
A.3 Computing the steady-state RGA matrix . . . 73
A.4 Computing PI settings . . . 74
A.5 Computing steady-state losses . . . 76
A.6 NLP solver . . . 80
A.7 Closed-loop simulator . . . 82
A.8 Local self-optimizing control (main) . . . 85
A.9 Global self-optimizing control (main) . . . 93

List of Tables

3.1 Chemical process components. 35
3.2 Nominal values of the disturbances and their standard deviations. 40
3.3 Nominal operating conditions and disturbance scenarios considered in the dynamic closed-loop simulations. 44

4.1 Some key process variables at the nominally optimal operating point. . . . 47
4.2 Optimal sets of measurements selected by the local method and corresponding steady-state local average losses. 50
4.3 Optimal sets of measurements selected by the global method and corresponding steady-state global average losses. 52
4.4 Optimal measurement sets from local and global methods validated using the nonlinear model. The steady-state average losses were computed from 100 normally distributed scenarios of disturbances and sensor noise realizations generated by Monte Carlo simulations. 56

List of Figures

2.1 Generalized control system. 3
2.2 Typical control hierarchy in a chemical plant (Skogestad, 2000). 5
2.3 Block diagram of a general self-optimizing control structure. 10
2.4 Pareto frontier representing the optimal set of measurements for a given control structure (Verheyleweghen and Jäschke, 2019). 23
2.5 Illustration of the effect of disturbances in active set changes. For disturbance d_1, the optimum lies at the constraint, while for disturbance d_2 the optimum is unconstrained (Krishnamoorthy and Skogestad, 2019). 26
2.6 Decentralized control of a 2×2 plant. (Skogestad and Postlethwaite, 2007). 27
2.7 Illustration of the model predictive control (MPC) principle (Foss and Heirung, 2013). 30

3.1 Atropine molecule ($C_{17}H_{23}NO_3$) 33
3.2 Process flowsheet. Adapted from (Nikolakopoulou et al., 2019). 34

4.1 Pareto frontier showing the steady-state local average loss versus the number of measurements for the control structures selected by the local method. 49
4.2 Pareto frontier showing the steady-state global average loss versus the number of measurements for the control structures selected by the global method. 53
4.3 Steady-state nonlinear average loss versus the number of measurements for control structures selected by global (red) and local (blue) methods, and for open-loop nominal operation (black). The average losses were computed from 100 normally distributed scenarios of disturbances and sensor noise realizations generated by Monte Carlo simulations. 55
4.4 Steady-state losses for 100 scenarios of normally distributed disturbances and sensor noise realizations resulting from open-loop nominal operation (black), and constant setpoint control of the optimal set of 6 measurements found by the global (red) and local (blue) methods. 57

4.5 Sensitivity analysis of the steady-state loss for the different disturbances considered in this work. The losses shown correspond to open-loop nominal operation (black), and constant setpoint control of the optimal set of 6 measurements found by the global (red) and local (blue) methods. 58
4.6 Profile of the cost function (E-factor) for nominal operation and three disturbance scenarios. 60
4.7 Profiles of the controlled variables for nominal operation and three disturbances scenarios. 60
4.8 Profile of the cost function (E-factor) for nominal operation and three disturbance scenarios. Sensor noise is present in the control system. 61
4.9 Profiles of the controlled variables for nominal operation and three disturbances scenarios. Sensor noise is present in the control system. 61

Nomenclature

Abbreviations

CV	Controlled variable
DAE	Differential algebraic equation
DMC	Dynamic matrix control
DOF	Degree of freedom
LQG	Linear-quadratic-Gaussian control
MIMO	Multiple-input-multiple-output
MPC	Model predictive control
MV	Manipulated variable
ODE	Ordinary differential equation
PID	Proportional-integral-derivative controller
QDMC	Quadratic dynamic matrix control
RGA	Relative gain array
RTO	Real-time optimization
SIMC	Simple internal model control
SISO	Single-input-single-output
SOC	Self-optimizing control

Control nomenclature

$\mathbf{c}$	Controlled variables
$\mathbf{c}_\mathrm{s}$	Controlled variable setpoints
$\mathbf{d}$	Disturbances
$\mathbf{F}$	Sensitivity matrix
$\mathbf{f}$	Vector of model equations
$\mathbf{g}$	Vector of operational constraints
$\mathbf{G}_y$	Linearized process model from $\mathbf{u}$ to $\mathbf{y}$
$\mathbf{G}_{yd}$	Linearized process model from $\mathbf{d}$ to $\mathbf{y}$
$\mathbf{H}$	Combination matrix
$\mathbf{n}$	Sensor noise
$\mathbf{u}$	Manipulated variables
$\mathbf{W}_d$	Diagonal scaling matrix containing standard deviations of disturbances
$\mathbf{W}_n$	Diagonal scaling matrix containing standard deviations of sensor noise
$\mathbf{x}$	States
$\mathbf{y}$	Measurements
$\mathcal{D}$	Set of all possible disturbances
$\mathcal{N}$	Set of all possible sensor noise realizations
τ	Time constant
τ_D	Derivative time constant
τ_I	Integral time constant
θ	Time delay
J	Cost function
k	Process gain

K_c	Controller gain	
L	Loss of optimality	
n_c	Number of controlled variables	
n_d	Number of disturbances	
n_u	Number of manipulated variables	
n_y	Number of measurements	

Process model nomenclature

$\bar{c_i}$	Average concentration of species i inside the liquid-liquid separator	mol/L
$\boldsymbol{R}$	Reaction rate matrix in $\mathbb{R}^{n_r \times n_d}$	mol/mL·min
$\boldsymbol{r}$	Component reaction rate matrix in $\mathbb{R}^{n_c \times n_d}$	mol/mL·min
$\boldsymbol{S}$	Stoichiometric matrix in $\mathbb{R}^{n_c \times n_r}$	
$\dot{m}_{\text{in},i,k}$	Mass flowrate of species i in stream k entering a mixer	g/min
$\dot{m}_{\text{out,tot}}$	Total mass flow coming out of a unit	g/min
$\dot{m}_{\text{out},i}$	Mass flowrate of species i exiting a mixer	g/min
ρ	Density inside of a unit operation	g/mL
ρ_i	Density of pure species i	g/mL
$c_{\text{AQ},i}$	Concentration of species i in the aqueous phase	mol/mL
$c_{\text{in},i}$	Concentration of species i at the inlet of the liquid-liquid separator	mol/mL
$c_{\text{in},s}$	Concentration of the solvent in the entrance of the liquid-liquid separator	mol/L
$c_{\text{OR},i}$	Concentration of species i in the organic phase	mol/mL
$c_{i,l}$	Concentration of the ith species at the lth discretization point in the tubular reactor	mol/mL
D_i	Distribution coefficient of species i in the liquid-liquid separator	
$E_{\text{A},j}$	Activation energy of the jth reaction	J/mol

$k_{j,0}$	Pre-exponential factor of the jth reaction	
M_i	Molecular mass of species i	g/mol
n_c	Number of species in the process	
n_d	Number of discretization points in the partial differential equation describing a tubular reactor	
n_s	Number of streams entering a mixer	
Q_{AQ}	Volumetric flowrate of the aqueous stream in the liquid-liquid separator	mL/min
Q_{OR}	Volumetric flowrate of the organic stream in the liquid-liquid separator	mL/min
Q_{tot}	Total volumetric flowrate in a unit	mL/min
R	Gas constant	J/mol·K
T	Reaction temperature	K
V_l	Volume from the entrance of the reactor to discretization point l	mL
x_i	Mass fraction of species i in a stream	

Convention

$(\cdot)^{\text{opt}}$	Variable at the optimal point
$(\cdot)_\delta$	Vectorized variable
$(\cdot)_{\text{m}}$	Measured variable, corrupted by sensor noise
$(\cdot)_{\text{nom}}$	Variable at the nominal point
$(\cdot)_{(i)}$	Variable at the ith disturbance scenario
Δ	Deviation of a variable from the nominal point

Chapter 1

Introduction

1.1 Project background

This master thesis is submitted in partial fulfillment of the requirements for the degree of Master of Science in Natural Gas Technology at the Norwegian University of Science and Technology. The work has been supervised by Professor Truls Gundersen and Associate Professor Johannes Jäschke at the Norwegian University of Science and Technology, and Professor Richard D. Braatz at the Massachusetts Institute of Technology.

1.2 Motivation

The production of pharmaceuticals is typically done using batch processing. The entire process chain, from raw materials to the final drug form, can take up to a total of 12 months. This long production time together with drug shortages due to supply chain interruptions and quality control issues are among the most important challenges facing the pharmaceutical industry today. Other challenges include the need to respond to sudden changes in demand (e.g., due to epidemics or pandemics), high drug production costs, and high waste generation.

To address these issues, important research efforts in academia and industry over the last decade have been oriented towards the development of continuous-flow processes for drug production. The case study considered in this work is a compact modular reconfigurable system for continuous-flow production of atropine recently developed at the Massachusetts Institute of Technology. Atropine is an active pharmaceutical ingredient with a variety of therapeutic uses, including the treatment of heart rhythm problems.

The stable and optimal operation of these pharmaceutical platforms requires the implementation of plantwide control strategies to guarantee short-term stability and long-term optimal economic performance in the face of uncertainty. This shall be the focus of this work.

1.3 Objectives

The objectives of this master thesis are:

1. Perform a literature study on self-optimizing control and controller design methodologies in multivariable plants.
2. Develop a screening methodology for control structure selection based on recent global self-optimizing control approaches, and apply it to the atropine process. Compare these results with the ones obtained by traditional local approaches.
3. Design a multiple-input-multiple-output controller for the atropine process.
4. Validate the resulting control system via dynamic closed-loop simulations.

1.4 Organization

This report comprises four chapters, including this introduction, and one appendix.

Chapter 2 contains the technical background. It begins by presenting some basic concepts on control system design and hierarchical plantwide control. The central part of the chapter is devoted to self-optimizing control, covering the general problem formulation as well as local and global methods for controlled variable selection. Then, controller design methodologies in multivariable plants are discussed, covering both decentralized PID control and model predictive control. The final section of the chapter is the most applied one and it motivates some recent research efforts in continuous-flow pharmaceutical manufacturing.

Chapter 3 presents the case study considered in this work, which is the continuous-flow production of atropine. A first-principles model of the process is derived. Furthermore, the methodologies for controlled variable selection and controller design used in this work are explained.

Chapter 4 presents the results of the case study, including the selection and validation of the self-optimizing control structure proposed for the atropine process. The chapter ends with a summary of the overall conclusions, an evaluation of the objectives set out in the previous section, and some directions for further research.

Appendix A contains some important pieces of source code developed in this project.

Chapter 2

Technical background

2.1 Introduction to control system design

From a systems engineering perspective, every problem can be considered a system composed of several interactive components. For some systems, like chemical plants, these components are processing units (reactors, separators, crystallizers, heat exchangers, etc.) and the interactions are material and energy flows. Other biological systems, like living cells, are composed of molecules and macromolecular structures interacting through covalent and non-covalent bondings. The list may be endless. However, no matter what type of system we are considering, in systems engineering the interest is always on studying how the components and their interactions determine the behavior of the whole system.

Control has been defined by Findeisen et al. (1980) as influencing a system to behave in a desired way. A block diagram of a generalized control system is shown in Fig. 2.1, where

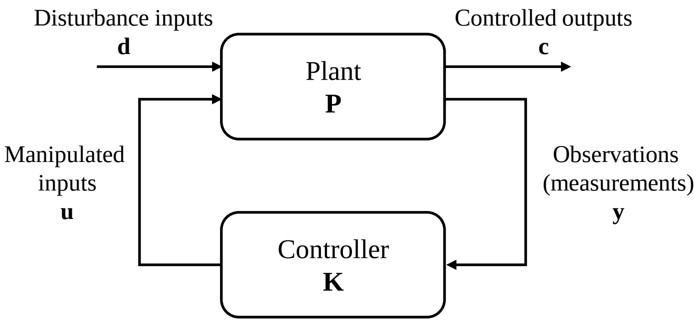

Figure 2.1: Generalized control system.

the plant $\mathbf{P}$ and the controller $\mathbf{K}$ are linked via actuator signals and sensor in a feedback structure.

The general *controller design* problem is to find the controller $\mathbf{K}$ that based on the information provided by the measurements $\mathbf{y}$, generates the control signals $\mathbf{u}$ such that the controlled outputs $\mathbf{c}$ are kept close to their desired setpoints despite disturbances $\mathbf{d}$. This problem will be the topic of Section 2.5.

However, this problem formulation assumes that we know which variables to measure, manipulate and control. Furthermore, specifications about the control configuration (local decomposition of the controller) and the control law (PID, MPC, LQG, H-infinity, etc.) may also be assumed. In other words, a given control structure is given at the outset.

The problem of determining the above structural decisions is called *control structure selection* or *control structure design*. In the field of process control, the term *plantwide control* is used as a synonym for referring to the same problem (Skogestad and Postlethwaite, 2007).

2.2 Principles of hierarchical control

In practice, complex systems such as complete chemical plants are controlled using hierarchical structures. The overall control problem is decomposed into a series of simple control problems. This decomposition follows two basic principles:

- *Hierarchical control*, based on the multilayer concept proposed by Lefkowitz (1966). This vertical decomposition divides the control system into a set of algorithms, or layers, that act on different time scales.

- *Decentralized control*, based on the multilevel concept proposed by Mesarovic et al. (1970). This horizontal decomposition divides the control system into a set of local control units, whose actions are coordinated by a master unit.

There are fundamental reasons for decomposing the overall control problem instead of using a centralized optimizing controller. One is robustness. In case of failure of a control unit, the overall system will survive. Furthermore, the use of decentralized (local) control operating on a fast time-scale will make the overall system less sensitive to disturbances. Another important reason is that decentralized control can be used effectively with limited knowledge about the system and therefore without much need for models.

The theoretical foundations of control and coordination in hierarchical systems go back to the classic work by Mesarovic et al. (1970). In the field of process control, these ideas gained wide recognition through the work of Findeisen et al. (1980).

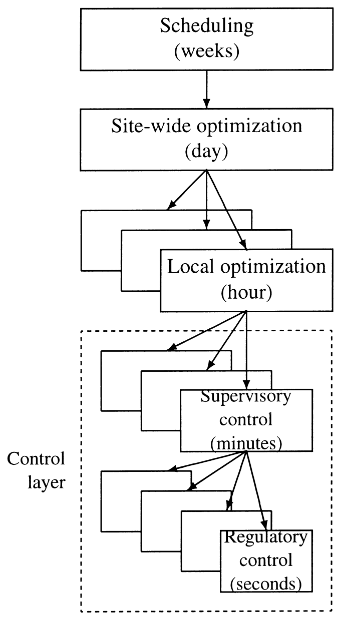

Figure 2.2: Typical control hierarchy in a chemical plant (Skogestad, 2000).

2.3 Plantwide control

Plantwide control refers to the problem of control structure selection discussed in Section 2.1, but applied to chemical plants. This involves all the structural decisions to design the control system with focus on the overall control philosophy of the plant. These decisions include the following tasks (Skogestad, 2000):

1. Selection of controlled variables and setpoints.
2. Selection of manipulated variables.
3. Selection of measurements for control purposes, including stabilization.
4. Selection of a control configuration (this refers to the structure of the overall controller, which can be decomposed into a series of decentralized controllers).
5. Selection of the type of controller (PID, MPC, LQG, H-infinity, etc.).

The plantwide control system for a chemical plant is organized in a hierarchical structure as the one shown in Fig. 2.2, where layers operate on different time scales. This decomposition is based on the decomposition principles discussed in Section 2.2. Typical layers in the control hierarchy of a chemical plant include:

- Scheduling (weeks).

- Site-wide optimization (day).
- Local optimization (hour).
- Supervisory/predictive control (minutes).
- Stabilization and regulatory control (seconds).

Optimal operation is achieved by cascading the economic plant objectives down from the upper optimization layers to the lower control layers. The different layers are linked via controlled variables and setpoints. Due to time-scale separation, the optimization and the control objectives are decoupled. This means that the control layer immediately implements a given setpoint from the optimization layer above without any knowledge of the optimality criterion used to compute it.

The operational objectives of a plantwide control system are to achieve short-term stability and long-term economic profitability. Skogestad (2004) outlines the following plantwide control procedure to achieve these objectives.

I. **Top-down analysis** (focus on plant economics)

  Step 1. Define optimal operation in terms of a scalar cost function to be minimized and a set of constraints to be satisfied.

  Step 2. Identify steady-state degrees of freedom and determine the steady-state optimal operating point for nominal operation, including active constraints.

  Step 3. Identify measurement candidates and select primary controlled variables for the supervisory control layer based on the principles of self-optimizing control.

  Step 4. Select the location of the throughput manipulator (TPM), which will determine where to set the production rate and the structure of the remaining inventory control system.

II. **Bottom-down analysis** (focus on robustness and stability)

  Step 5. Select secondary controlled variables for the regulatory (stabilizing) control layer and determine the pairing with the manipulated variables.

  Step 6. Select the structure of the supervisory control layer (decentralized or multivariable control).

  Step 7. Determine the need for an optimization layer (RTO).

  Step 7. Validation through nonlinear dynamic simulation.

This work is mainly concerned with the top-down analysis of the procedure, which focuses on the selection of controlled variables and setpoints for achieving near-optimal operation in the presence of disturbances. This will be discussed more in detailed in the following section.

2.4 Self-optimizing control

The paradigm of hierarchical control is effective in handling the complexity of real process systems, e.g., when controlling a complete chemical plant. However, it results in a loss of optimality with respect to a centralized optimizing controller. This loss comes from the time-scale separation between the optimization and the control layers. The optimization layer typically operates on a slower time-scale where setpoints are only recomputed at low-frequency time intervals. Therefore, disturbances impacting the process between setpoint updates are not optimally rejected by the lower control layer until the next update.

It turns out that the loss of optimality from low-frequency setpoint adaptation strongly depends on the selected controlled variables. This is an important structural decision with consequences for the overall profitability of the plant. This has been recognized by Foss (1973) in his "Critique of chemical process control theory":

> Which variables should be measured, which inputs should be manipulated, and what links should be made between these two sets? This problem is considered by many to be the most important problem encountered by designers of chemical process control systems (Foss, 1973).

Since then, important research efforts have been focused on finding optimal control structures in terms of economics. Early work in this direction includes Morari et al. (1980); Morari (1981); Narraway and Perkins (1994). In the first part of the series of papers "Studies in the synthesis of control structures for chemical processes", Morari et al. (1980) introduced the concept of "feedback optimizing control":

> In attempting to synthesize a feedback optimizing control structure, our main objective is to translate the economic objective into process control objectives. In other words we want to find a function $\mathbf{c}$ of the process variables [...] which when held constant, leads automatically to the optimal adjustment of the manipulated variables, and with it, the optimal operating conditions (Morari et al., 1980).

The idea of Morari et al. (1980) was to achieve optimal operation not by solving optimization problems online, but through feedback control of optimal invariants (i.e., process variables that remain constant whenever the process is operated optimally). However, the idea of "feedback optimizing control" was largely forgotten for about two decades, but it laid the groundwork for the concept of "self-optimizing control" presented by Skogestad (2000). Closely related, the idea here is to achieve near-optimal operation by selecting controlled variables whose optimal setpoints are near-insensitive to disturbances and sensor noise, so that the loss from time-scale separation is minimized. Skogestad (2000) writes:

> Self-optimizing control is when we can achieve an acceptable loss with constant setpoint values for the controlled variables (without the need to reoptimize when disturbances occur) (Skogestad, 2000).

A control layer designed by the principles of self-optimizing control can reject disturbances near-optimally on a fast time-scale without having to wait for the real-time optimizer to update the setpoints. Meanwhile, the upper optimization layer can still correct for plant-model mismatches or unmodeled disturbances on a slower time-scale.

Since the publication of the seminal paper by Skogestad (2000), research in this area has focused on providing systematic methods to select self-optimizing controlled variables. The developments since then are collected in the recent review paper "Self-optimizing control - A survey" by Jäschke et al. (2017).

2.4.1 Problem formulation

We assume that quasi-steady-state optimal operation of continuous processes with parametric model uncertainty represented as disturbances can be mathematically formulated as a static optimization problem,[1]

$$\begin{aligned} \min_{\mathbf{u}} \quad & J(\mathbf{u},\mathbf{x},\mathbf{d}) \\ \text{s.t.} \quad & \mathbf{f}(\mathbf{u},\mathbf{x},\mathbf{d}) = 0, \\ & \mathbf{g}(\mathbf{u},\mathbf{x},\mathbf{d}) \leq 0, \end{aligned} \tag{2.1}$$

where $\mathbf{u} \in \mathbb{R}^{n_u}$ are the manipulated variables, $\mathbf{x} \in \mathbb{R}^{n_x}$ are the states, $\mathbf{d} \in \mathbb{R}^{n_d}$ are the disturbances, $J : \mathbb{R}^{n_u \times n_x \times n_d} \mapsto \mathbb{R}$ is a scalar objective function representing a given optimality criterion (typically, the operating cost of the plant to be minimized), $\mathbf{f} : \mathbb{R}^{n_u \times n_x \times n_d} \mapsto \mathbb{R}^{n_f}$ are the model equations, and $\mathbf{g} : \mathbb{R}^{n_u \times n_x \times n_d} \mapsto \mathbb{R}^{n_g}$ are the operational constraints.

By using the model equations $\mathbf{f}(\mathbf{u},\mathbf{x},\mathbf{d}) = 0$ to formally eliminate the states $\mathbf{x}$, the optimization problem above becomes

$$\begin{aligned} \min_{\mathbf{u}} \quad & J(\mathbf{u},\mathbf{d}) \\ \text{s.t.} \quad & \mathbf{g}(\mathbf{u},\mathbf{d}) \leq 0. \end{aligned} \tag{2.2}$$

In general, a subset of the operational constraints $\mathbf{g_a} \subset \mathbf{g}$ will be active at the optimal solution, i.e., $\mathbf{g_a} = 0$. In self-optimizing control, we assume that the active constraints can be identified and controlled at the active boundaries using an equal number of degrees of freedom (manipulated variables).[2] By formally eliminating the active constraints and their corresponding degrees of freedom in (2.2), we obtain the following unconstrained problem:

$$\min_{\mathbf{u}} \quad J(\mathbf{u},\mathbf{d}), \tag{2.3}$$

[1]Economic plant performance in most continuous processes is mainly determined by steady-state operation. In the case of continuous processes with frequent grade changes or batch processes, optimal operation should be formulated as a dynamic optimization problem.

[2]Theory in self-optimizing control has been originally derived under the assumption that active constraints do not change under operation. If this is not the case, Section 2.4.6 will give a brief overview of methods to handle active set changes within the self-optimizing control framework.

where, for simplicity, we also use $\mathbf{u}$ here to denote the remaining degrees of freedom in the reduced unconstrained space after satisfying all the active constraints.

To achieve optimal operation, the necessary conditions of optimality must be satisfied. This is, the reduced gradient must be zero, $\mathbf{J}_u = 0$. Self-optimizing control achieves this objective indirectly by using the unconstrained degrees of freedom $\mathbf{u}$ to control carefully selected controlled variables $\mathbf{c}$ at constant setpoints $\mathbf{c}_{\mathrm{s}}$. By tracking the right variables using a feedback controller, the system is driven towards the optimal operating point and $\mathbf{u}$ approaches $\mathbf{u}^{\mathrm{opt}}$, despite disturbances and sensor noise.

The ideal self-optimizing controlled variable would be the reduced gradient of the cost function, $\mathbf{J}_u$. In this case, optimal operation is achieved by simply controlling the gradient to a constant setpoint of zero. The problem is that a direct evaluation of the gradient in real processes is often challenging. Instead, we use the available plant measurements for which we assume we have a model,

$$\mathbf{y} = \mathbf{m}(\mathbf{u}, \mathbf{x}, \mathbf{d}), \tag{2.4}$$

where the measurements $\mathbf{y}$ are a function of the inputs $\mathbf{u}$, the states $\mathbf{x}$, and the disturbances $\mathbf{d}$. These measurements can include process variables, manipulated variables (inputs) and also measured disturbances (parametric model uncertainty that can be measured in a feed-forward manner before affecting the process, e.g., knowledge about prices).

In a real plant, measurements $\mathbf{y}$ are corrupted by sensor noise $\mathbf{n}$, such that

$$\mathbf{y}_{\mathrm{m}} = \mathbf{y} + \mathbf{n}. \tag{2.5}$$

In general, controlled variables can be selected as any function $\mathbf{h}$ of the available plant measurements, such that

$$\mathbf{c} = \mathbf{h}(\mathbf{y}_{\mathrm{m}}). \tag{2.6}$$

However, we often restrict this function to linear combination of measurements on the form

$$\mathbf{c} = \mathbf{H}\mathbf{y}_{\mathrm{m}}. \tag{2.7}$$

The matrix $\mathbf{H} \in \mathbb{R}^{n_c \times n_y}$ is called selection or combination matrix. A full matrix will result in measurement combinations, while rows containing one ”1” and zeros otherwise will result in the control of single measurements. Research has shown that controlling combination of measurements enhances the economic performance of the control scheme, achieving lower steady-state losses. (Alstad and Skogestad, 2007; Kariwala, 2007; Alstad et al., 2009). A block diagram of a general self-optimizing control structure is shown in Fig. 2.3.

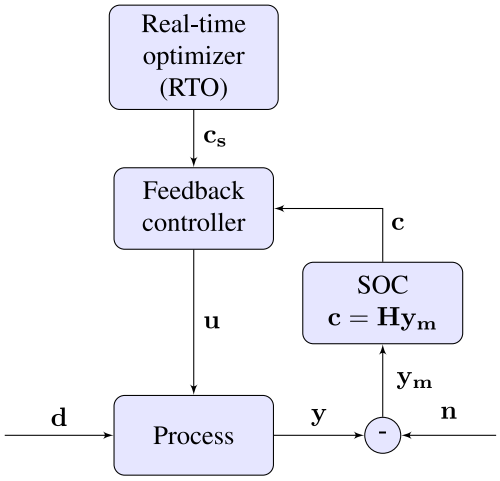

Figure 2.3: Block diagram of a general self-optimizing control structure.

Skogestad and Postlethwaite (2007) state some requirements for selecting good controlled variables:

1. The inputs $\mathbf{u}$ should have a significant effect (gain) on the controlled variables $\mathbf{c}$ (i.e., they should be easy to control).
2. Their optimal setpoints $\mathbf{c}_\mathrm{s}$ should be insensitive to disturbances and sensor noise.
3. In case of several controlled variables, these should not be closely correlated.

The criterion for evaluating the performance of a given control structure $\mathbf{c} = \mathbf{H}\mathbf{y}_\mathrm{m}$ is the economic loss, which is defined as the difference between the cost resulting from that control structure and the cost resulting from truly optimal operation. Mathematically,

$$\begin{aligned} L &= J(\mathbf{u}, \mathbf{d}) - J(\mathbf{u}^\mathrm{opt}, \mathbf{d}), \\ &= J(\mathbf{u}, \mathbf{d}) - J^\mathrm{opt}(\mathbf{d}). \end{aligned} \tag{2.8}$$

Here, the loss is written in terms of the steady-state inputs, which may be generated by using either a closed or an open-loop policy.

The central issue addressed in the framework of self-optimizing control is the selection of appropriate controlled variables $\mathbf{c} = \mathbf{H}\mathbf{y}_\mathrm{m}$ and setpoints $\mathbf{c}_\mathrm{s}$. More specifically, for a given set of possible disturbances $\mathbf{d} \in \mathcal{D}$ and sensor noise realizations $\mathbf{n} \in \mathcal{N}$, we want to find the optimal combination matrix $\mathbf{H}$ and the setpoints $\mathbf{c}_\mathrm{s}$ which minimize the average (or worst-case) loss for the entire operating region. To solve this problem, different methods have been proposed in the literature. The next sections will give a brief overview over some of these methods, including early brute force approaches, local methods and some recent global methods.

2.4.2 Brute force methods

Early methods to find self-optimizing controlled variables relied on the brute force evaluation of all possible control structures. The idea is simply to evaluate the loss L corresponding to each possible combination matrix $\mathbf{H}$ for all possible values of disturbances $\mathbf{d} \in \mathcal{D}$ and measurement noise realizations $\mathbf{n} \in \mathcal{N}$, and then rank the different candidates in terms of the worst-case loss

$$L_{\text{wc}}(\mathbf{H}) = \max_{\mathbf{d}\in\mathcal{D},\, \mathbf{n}\in\mathcal{N}} L(\mathbf{n}, \mathbf{d}, \mathbf{H}), \tag{2.9}$$

or the average loss

$$L_{\text{av}}(\mathbf{H}) = \mathop{\mathbb{E}}_{\mathbf{d}\in\mathcal{D},\, \mathbf{n}\in\mathcal{N}} \left[L(\mathbf{n}, \mathbf{d}, \mathbf{H})\right], \tag{2.10}$$

where $\mathbb{E}\left[\,\cdot\,\right]$ is the expectation operator. The candidate with the lowest average or worst-case loss is finally selected.

Brute force approaches can only be realized in small-scale problems with only a few sets of possible controlled variables. Otherwise, they lead to a combinatorial explosion. When selecting n_u single measurements as controlled variables out of a total of n_y measurements, there are

$$\mathrm{C}_{n_y}^{n_u} = \binom{n_y}{n_u} = \frac{n_y!}{(n_y - n_u)!\, n_u!} \tag{2.11}$$

possible combinations. In complete chemical plants, with hundreds or thousands of possible measurements, this approach is clearly intractable. Furthermore, for each loss evaluation a nonlinear programming problem has to be solved, which is generally large-dimensional and non-convex.

If we allow linear combination of measurements to be chosen as controlled variables, instead of single measurements, the problem becomes even more difficult. Here, the problems are generally large-scale nonlinear bilevel optimization problems.

One common strategy in brute force approaches is to neglect the sensor noise, $\mathbf{n} = 0$, and therefore reduce the number of optimization problems to be solved. However, this can lead to poor control structures when noise is present in the real plant and if the number of disturbances and potential controlled variables is still high, the problem will remain intractable. The use of heuristics can also be used to reduce the number of candidate controlled variables.

However, it is clear that without systematic methods to find optimal controlled variables in real large-scale processes, the search for self-optimizing control structures is hopeless.

2.4.3 Local methods

The first systematic methods proposed in the literature were based on the linearization of the plant model and a quadratic approximation of the loss around the nominal operating point. The idea is to select locally optimal control structures that behave well around the nominal point, where the process is expected to operate most of the time. The control structures resulting from this local screening can then be validated using the original nonlinear model over the entire operating region.

Local approximation of the loss

The nonlinear cost function J is approximated locally by a second order Taylor expansion around the moving optimal point $(\mathbf{u}^{\text{opt}}, \mathbf{d})$. For a given disturbance $\mathbf{d}$, this expansion gives

$$J(\mathbf{u},\mathbf{d}) = J(\mathbf{u}^{\text{opt}},\mathbf{d}) + \mathbf{J}_u^{\mathrm{T}}(\mathbf{u}-\mathbf{u}^{\text{opt}}) + \frac{1}{2}(\mathbf{u}-\mathbf{u}^{\text{opt}})^{\mathrm{T}}\mathbf{J}_{uu}(\mathbf{u}-\mathbf{u}^{\text{opt}}) + \mathcal{O}^3 \quad (2.12)$$

where $\mathbf{J}_u$ and $\mathbf{J}_{uu}$ are the Jacobian and the Hessian matrices of the cost function J evaluated at the optimal point, respectively. Subtracting $J(\mathbf{u}^{\text{opt}}, \mathbf{d})$ from both sides of (2.12) and taking into account that at the optimum $\mathbf{J}_u^{\mathrm{T}}(\mathbf{u} - \mathbf{u}^{\text{opt}}) = 0$, which corresponds with the necessary condition of optimality, we get the following approximation of the loss:

$$\begin{aligned} L &= J(\mathbf{u},\mathbf{d}) - J(\mathbf{u}^{\text{opt}},\mathbf{d}), \\ &\approx \frac{1}{2}\mathbf{e}_u^{\mathrm{T}}\mathbf{J}_{uu}\mathbf{e}_u. \end{aligned} \quad (2.13)$$

Here, $\mathbf{e}_u \triangleq \mathbf{u} - \mathbf{u}^{\text{opt}}$ is the deviation of the inputs with respect to their optimal value.

Introducing the loss variable $\mathbf{z} = \mathbf{J}_{uu}^{1/2}\mathbf{e}_u$, we can write the loss as

$$L = \frac{1}{2}\mathbf{z}^{\mathrm{T}}\mathbf{z}, \quad (2.14)$$

or alternatively

$$L = \frac{1}{2}\|\mathbf{z}\|_2^2, \quad (2.15)$$

where $\|\cdot\|$ denotes the two-norm of the vector.

Local approximation of the plant and linear measurement combination

For small deviations with respect to the nominally optimal operating point, we can linearize the measurement model (2.4) as

$$\Delta \mathbf{y}_{\mathrm{m}} = \mathbf{G}_y \Delta \mathbf{u} + \mathbf{G}_{yd} \Delta \mathbf{d} + \mathbf{n}, \tag{2.16}$$

where $\Delta \mathbf{y}_{\mathrm{m}} = \mathbf{y}_{\mathrm{m}} - \mathbf{y}_{\mathrm{m}}^{\mathrm{opt}}$, $\Delta \mathbf{u} = \mathbf{u} - \mathbf{u}^{\mathrm{opt}}$, and $\Delta \mathbf{d} = \mathbf{d} - \mathbf{d}_{\mathrm{nom}}$. Furthermore, $\mathbf{G}_y = (\partial \mathbf{y}/\partial \mathbf{u})$, and $\mathbf{G}_{yd} = (\partial \mathbf{y}/\partial \mathbf{d})$ are the gains from the inputs and disturbances to the outputs evaluated at the nominally optimal point, respectively. The additive sensor noise is represented by the vector $\mathbf{n}$.

Assuming that the controlled variables are linear combinations of measurements, the controlled variables in deviation variables are given by

$$\Delta \mathbf{c}_{\mathrm{m}} = \mathbf{H} \Delta \mathbf{y}_{\mathrm{m}}. \tag{2.17}$$

Inserting (2.16) into (2.17), we get an expression for the controlled variables as a linear function of the inputs, disturbances and measurement noise,

$$\Delta \mathbf{c}_{\mathrm{m}} = \mathbf{H}\mathbf{G}_y \Delta \mathbf{u} + \mathbf{H}\mathbf{G}_{yd} \Delta \mathbf{d} + \mathbf{H}\mathbf{n}, \tag{2.18}$$

where the matrix $\mathbf{H}\mathbf{G}_y$ must have full rank, i.e., $\mathrm{rank}(\mathbf{H}\mathbf{G}_y) = n_u$ in order to obtain linear independent controlled variables that fully specify the system.

Exact local method

The exact local method (Halvorsen et al., 2003; Alstad et al., 2009) is a method for evaluating the loss for a given control structure.

To derive an expression for the loss in terms of a given combination matrix $\mathbf{H}$, disturbance $\mathbf{d}$, and noise realization $\mathbf{n}$, we start by assuming perfect control. Therefore, the controlled variables will be kept at their nominal setpoints despite disturbances, $\mathbf{c} = \mathbf{c}_{\mathrm{nom}}$, which in deviation variables corresponds to $\Delta \mathbf{c} = 0$. Imposing this condition on (2.18), the input $\Delta \mathbf{u}$ generated by the controllers is

$$\begin{aligned} \Delta \mathbf{u} &= (\mathbf{H}\mathbf{G}_y)^{-1}(\underbrace{\Delta \mathbf{c}_{\mathrm{m}}}_{=0} - \mathbf{H}\mathbf{G}_{yd}\Delta \mathbf{d} - \mathbf{H}\mathbf{n}) \\ &= -(\mathbf{H}\mathbf{G}_y)^{-1}\mathbf{H}(\mathbf{G}_{yd}\Delta \mathbf{d} + \mathbf{n}). \end{aligned} \tag{2.19}$$

Using this result into the loss expression (2.15), Halvorsen et al. (2003) shows that the loss variable $\mathbf{z}$ for a given control structure $\mathbf{H}$, disturbance $\mathbf{d}$, and sensor noise value $\mathbf{n}$

becomes

$$\begin{aligned}\mathbf{z} &= -\mathbf{J}_{uu}^{1/2}(\mathbf{H}\mathbf{G}_y)^{-1}\mathbf{H}\Bigg[\underbrace{(\mathbf{G}_{yd} - \mathbf{G}_y\mathbf{J}_{uu}^{-1}\mathbf{J}_{ud})}_{\mathbf{F}}\Delta\mathbf{d} + \mathbf{n}\Bigg] \\ &= -\mathbf{J}_{uu}^{1/2}(\mathbf{H}\mathbf{G}_y)^{-1}\mathbf{H}\Bigg[\mathbf{F}\Delta\mathbf{d} + \mathbf{n}\Bigg].\end{aligned} \tag{2.20}$$

Here, $\mathbf{F}$ is the optimal sensitivity matrix,

$$\mathbf{F} = \mathbf{G}_{yd} - \mathbf{G}_y\mathbf{J}_{uu}^{-1}\mathbf{J}_{ud}, \tag{2.21}$$

which represents the sensitivity of the vector of optimal measurements with respect to the disturbances, i.e.,

$$\mathbf{F} = \frac{\partial \mathbf{y}^{\text{opt}}}{\partial \mathbf{d}}. \tag{2.22}$$

The sensitivity matrix $\mathbf{F}$ can be computed by evaluating (2.21), using re-optimization and finite differences, or nonlinear programming sensitivity using the inverse function theorem, see Fiacco et al. (1983); Pirnay et al. (2012).

We scale the disturbances and sensor noise realizations using diagonal scaling matrices, $\mathbf{W}_d$ and $\mathbf{W}_n$, respectively, of appropriate sizes such that

$$\Delta\mathbf{d} = \mathbf{W}_d\mathbf{d}', \tag{2.23}$$

$$\mathbf{n} = \mathbf{W}_n\mathbf{n}', \tag{2.24}$$

where $\mathbf{d}'$ and $\mathbf{n}'$ are the scaled disturbance and sensor noise vector.

The loss for a given value of $\mathbf{d}'$ and $\mathbf{n}'$ can be evaluated by

$$L = \frac{1}{2}\left\|\mathbf{J}_{uu}^{1/2}(\mathbf{H}\mathbf{G}_y)^{-1}\mathbf{H}\tilde{\mathbf{F}}\begin{bmatrix}\mathbf{d}' \\ \mathbf{n}'\end{bmatrix}\right\|_2^2, \tag{2.25}$$

where the augmented matrix $\tilde{\mathbf{F}}$ is defined as

$$\tilde{\mathbf{F}} \triangleq \begin{bmatrix}\mathbf{F}\mathbf{W}_d & \mathbf{W}_n\end{bmatrix}. \tag{2.26}$$

We can simplify (2.25) by introducing the matrix

$$\mathbf{M} = \mathbf{J}_{uu}^{1/2}(\mathbf{H}\mathbf{G}_y)^{-1}\mathbf{H}\tilde{\mathbf{F}}, \tag{2.27}$$

so that the loss becomes

$$L = \frac{1}{2}\left\|\mathbf{M}\begin{bmatrix}\mathbf{d}'\\ \mathbf{n}'\end{bmatrix}\right\|_2^2. \tag{2.28}$$

The steady-state loss is independent of the scaling chosen for the controlled variables. The reason is that the loss matrix $\mathbf{M}$ remains constant when we pre-multiply it by any non-singular matrix. To show this, consider that the controlled variables are rescaled using the scaling matrix $\mathbf{Q}$, such that (Jäschke et al., 2017)

$$\begin{aligned}\Delta\hat{\mathbf{c}} &= \mathbf{Q}\Delta\mathbf{c}\\ &= \mathbf{Q}(\mathbf{H}\Delta\mathbf{y})\\ &= \hat{\mathbf{H}}\Delta\mathbf{y},\end{aligned} \tag{2.29}$$

where $\hat{\mathbf{H}}$ is the rescaled selection matrix. We can see that the loss given by $\mathbf{M}$ is the same in both cases,

$$\begin{aligned}\mathbf{M} &= \mathbf{J}_{uu}^{1/2}(\hat{\mathbf{H}}\mathbf{G}_y)^{-1}\hat{\mathbf{H}}\tilde{\mathbf{F}}\\ &= \mathbf{J}_{uu}^{1/2}(\mathbf{Q}\mathbf{H}\mathbf{G}_y)^{-1}\mathbf{Q}\mathbf{H}\tilde{\mathbf{F}}\\ &= \mathbf{J}_{uu}^{1/2}(\mathbf{H}\mathbf{G}_y)^{-1}\mathbf{Q}^{-1}\mathbf{Q}\mathbf{H}\tilde{\mathbf{F}}\\ &= \mathbf{J}_{uu}^{1/2}(\mathbf{H}\mathbf{G}_y)^{-1}\mathbf{H}\tilde{\mathbf{F}}.\end{aligned} \tag{2.30}$$

Local average loss

Local expressions for both the average and worst-case loss have been derived under different assumptions on how disturbances and sensor noise realizations are distributed. We show here the most relevant ones.

- **Two-norm bounded disturbance and noise** (Halvorsen et al., 2003). Assuming that the disturbances and noise are independent and uniformly distributed over the set

$$\mathcal{DN}_2 = \left\{(\mathbf{d}',\mathbf{n}')\;\middle|\;\left\|\begin{bmatrix}\mathbf{d}' & \mathbf{n}'\end{bmatrix}^{\mathrm{T}}\right\|_2 \le 1\right\}, \tag{2.31}$$

the worst-case loss becomes

$$L_{\text{wc}} = \max_{\left\| \begin{bmatrix} \mathbf{d}' \\ \mathbf{n}' \end{bmatrix} \right\|_2 \leq 1} \frac{1}{2} \left\| \mathbf{M} \begin{bmatrix} \mathbf{d}' \\ \mathbf{n}' \end{bmatrix} \right\|_2^2 = \frac{1}{2} \|\mathbf{M}\|_2^2 = \frac{1}{2} \bar{\sigma}^2(\mathbf{M}), \tag{2.32}$$

where $\bar{\sigma}(\,\cdot\,)$ represents the largest singular value.

- **Infinity-norm bounded disturbance and noise** (Kariwala et al., 2008). Assuming that the disturbances and noise are independent and uniformly distributed over the set

$$\mathcal{DN}_\infty = \left\{ (\mathbf{d}', \mathbf{n}') \,\middle|\, \left\| \begin{bmatrix} \mathbf{d}' & \mathbf{n}' \end{bmatrix}^{\mathrm{T}} \right\|_\infty \leq 1 \right\}, \tag{2.33}$$

the average loss in given by

$$L_{\text{av}} = \mathop{\mathbb{E}}_{\mathbf{d}',\, \mathbf{n}' \in \mathcal{DN}_\infty} \left[\frac{1}{2} \left\| \mathbf{M} \begin{bmatrix} \mathbf{d}' \\ \mathbf{n}' \end{bmatrix} \right\|_2^2 \right] = \frac{1}{6} \|\mathbf{M}\|_{\mathrm{F}}^2, \tag{2.34}$$

where $\mathbb{E}[\cdot]$ is the expectation operator.

- **Normally distributed disturbance and noise** (Kariwala et al., 2008). Assuming that the disturbances and noise are normally distributed with zero mean and unit variance,

$$\mathcal{DN}_\mathcal{N} = \left\{ \mathbf{d}' \sim \mathcal{N}(0, \mathbf{I}),\, \mathbf{n}' \sim \mathcal{N}(0, \mathbf{I}) \right\}, \tag{2.35}$$

the average loss becomes

$$L_{\text{av}} = \mathop{\mathbb{E}}_{\mathbf{d}',\, \mathbf{n}' \in \mathcal{DN}_\mathcal{N}} \left[\frac{1}{2} \mathbf{M} \begin{bmatrix} \mathbf{d}' \\ \mathbf{n}' \end{bmatrix} \right] = \frac{1}{2} \|\mathbf{M}\|_{\mathrm{F}}^2, \tag{2.36}$$

and the worst-case loss results unbounded, i.e., $L_{\text{wc}} = \infty$, as the noise and disturbances can become arbitrarily large.

In this work, disturbances and sensor noise realizations are assumed to be normally distributed. Kariwala et al. (2008) showed that the combination matrix $\mathbf{H}$ that minimizes the average loss is superoptimal, meaning that it also minimizes the worst-case loss. For this reason and because it is more meaningful in real applications, we decided to use the average loss in this work.

Minimum loss method

The optimal linear combination of measurements $\mathbf{c}_\mathrm{m} = \mathbf{H}\mathbf{y}_\mathrm{m}$ can be found by minimizing any of the previous expressions for the average or worst-case loss. Considering the average loss minimization for normally distributed scenarios of disturbances and sensor noise,

$$\begin{aligned}\mathbf{H} &= \arg\min_{\mathbf{H}} \frac{1}{2}\|\mathbf{M}\|_\mathrm{F}^2 \\ &= \arg\min_{\mathbf{H}} \frac{1}{2}\|\mathbf{J_{uu}^{1/2}}(\mathbf{H}\mathbf{G_y})^{-1}\mathbf{H}\tilde{\mathbf{F}}\|_\mathrm{F}^2.\end{aligned} \tag{2.37}$$

This formulation results in a non-convex optimization problem difficult to solve. However, exploiting the non-uniqueness property of the combination matrix $\mathbf{H}$, Kariwala et al. (2008) derived an equivalent convex optimization problem:

$$\begin{aligned}\min_{\mathbf{H}} \quad & \|\mathbf{H}\tilde{\mathbf{F}}\|_\mathrm{F} \\ \text{s.t.} \quad & \mathbf{H}\mathbf{G}_y = \mathbf{J}_{uu}^{1/2}.\end{aligned} \tag{2.38}$$

The idea is to select a nonsingular matrix $\mathbf{Q}$ such that $\mathbf{H}\mathbf{G}_y = \mathbf{J}_{uu}^{1/2}$ so that the nonlinearity in the matrix $\mathbf{M}$ is canceled. This is possible because pre-multiplying $\mathbf{H}$ by any nonsingular matrix $\mathbf{Q}$ will not affect the value of the loss matrix $\mathbf{M}$, as shown previously.

An analytical solution to this problem was found by Alstad et al. (2009) and later simplified by Yelchuru and Skogestad (2012),

$$\mathbf{H} = (\mathbf{G}_y)^\mathrm{T}(\tilde{\mathbf{F}}\tilde{\mathbf{F}}^\mathrm{T})^{-1}. \tag{2.39}$$

The matrix $(\tilde{\mathbf{F}}\tilde{\mathbf{F}}^\mathrm{T})^{-1}$ is always invertible provided that all measurements are corrupted by noise.

Null-space method

The null-space method is a special case of the minimum loss method when no sensor noise is present, $\mathbf{W}_n = 0$. Alstad and Skogestad (2007) showed that if the number of measurements is greater or equal than the number of inputs and disturbances, $n_y \geq n_u + n_d$, it is always possible to find a nontrivial combination matrix $\mathbf{H}$ such that

$$\mathbf{H}\mathbf{F} = 0, \tag{2.40}$$

resulting in zero average (or worst-case) loss. The matrix $\mathbf{H}$ is therefore selected in the null-space of $\mathbf{F}$, and hence the name of the method.

However, neglecting measurement noise will not give optimal results in a real setting. Furthermore, the condition $n_y \geq n_u + n_d$ requires control structures with large number of measurements.

2.4.4 Global methods

Local methods introduced so far are based on linearization around the nominally optimal operating point, resulting in locally optimal control structures. However, if disturbances move the process far away from the nominal point, local control structures may perform poorly, resulting in unacceptable losses. Several global approaches have been derived to address this issue. In particular, this section will focus on a simplified global self-optimizing control (gSOC) algorithm proposed by Ye et al. (2015), that leads to an analytical solution of $\mathbf{H}$ similar to the one derived in the minimum loss method. However, the matrices involved in the global solution are constructed based on optimal data sampled from the whole operating region. For an overview of other global self-optimizing control methods, we refer the reader to Jäschke et al. (2017).

In the global approach by Ye et al. (2015), both the combination matrix $\mathbf{H}$ and the setpoints $\mathbf{c}_\mathrm{s}$ are degrees of freedom for optimization. The objective is no longer to achieve locally zero loss (in local methods, $\mathbf{c}_\mathrm{s}$ is typically selected to be $\mathbf{c}_\mathrm{s} = \mathbf{H}\mathbf{y}^\mathrm{opt}_\mathrm{nom}$), but to select economically robust setpoints that minimize the average loss over the entire operating range. Setpoints $\mathbf{c}_\mathrm{s}$ are included in the combination matrix $\mathbf{H}$ by introducing an augmented combination matrix $\mathbf{H}_\mathrm{aug} = \begin{bmatrix}-\mathbf{c}_\mathrm{s} & \mathbf{H}\end{bmatrix}$ and an augmented measurement vector $\mathbf{y}_\mathrm{aug} = \begin{bmatrix}1 & \mathbf{y}^\mathrm{T}\end{bmatrix}^\mathrm{T}$. This formulation generalizes the concept of arbitrary controlled variables $\mathbf{c}$ with nonzero setpoints $\mathbf{c}_\mathrm{s}$ to controlled variables $\mathbf{c}_\mathrm{aug}$ with zero setpoints. Both formulations are equivalent:

$$\begin{aligned}
\mathbf{c}_\mathrm{aug} = 0 &\Leftrightarrow \mathbf{H}_\mathrm{aug}\,\mathbf{y}_\mathrm{aug} = 0,\\
&\Leftrightarrow \begin{bmatrix}-\mathbf{c}_\mathrm{s} & \mathbf{H}\end{bmatrix}\begin{bmatrix}1\\ \mathbf{y}\end{bmatrix} = 0,\\
&\Leftrightarrow -\mathbf{c}_\mathrm{s} + \mathbf{H}\mathbf{y} = 0,\\
&\Leftrightarrow \mathbf{H}\mathbf{y} = \mathbf{c}_\mathrm{s},\\
&\Leftrightarrow \mathbf{c} = \mathbf{c}_\mathrm{s}.
\end{aligned} \tag{2.41}$$

For simplicity and to draw a parallel with the notation used in local methods, the subscript corresponding to the augmented matrices will be dropped in the remainder of this section.

Global average loss

The global average loss in the entire uncertain space spanned by all expected disturbances $\mathbf{d} \in \mathcal{D}$ and sensor noise realizations $\mathbf{n} \in \mathcal{N}$, resulting from a given control structure $\mathbf{H}$, can be expressed as

$$\begin{aligned}
L_\mathrm{av}(\mathbf{H}) &= \mathbb{E}\left[L(\mathbf{d},\mathbf{n},\mathbf{H})\right],\\
&= \int_{\mathbf{d}\in\mathcal{D},\,\mathbf{n}\in\mathcal{N}} \rho(\mathbf{d})\rho(\mathbf{n})L(\mathbf{d},\mathbf{n},\mathbf{H})\,\mathrm{d}\mathbf{n}\,\mathrm{d}\mathbf{d},
\end{aligned} \tag{2.42}$$

where $\mathbb{E}[\,\cdot\,]$ and $\rho(\,\cdot\,)$ denote the expected value and the probability density of a random variable, respectively.

The loss has been originally defined from an open-loop perspective in terms of $\mathbf{u}$ and $\mathbf{d}$. Here, we consider the closed-loop performance of the system and the loss L is implicitly defined in terms of disturbances $\mathbf{d}$, sensor noise $\mathbf{n}$ and the selected control structure $\mathbf{H}$. The feedback result of $\mathbf{u}$ in closed-loop operation is governed by the following equations:

$$\mathbf{y} = \mathbf{f}(\mathbf{u}, \mathbf{d}), \tag{2.43}$$

$$\mathbf{H}\mathbf{y}_{\mathrm{m}} = \mathbf{H}(\mathbf{y} + \mathbf{n}) = 0, \tag{2.44}$$

which correspond to the nonlinear process model and the feedback control effects, respectively. The direct evaluation of the loss based on the formulation above requires one to solve a set of nonlinear equations. The calculations involved are cumbersome, rendering the approach intractable in practice.

Global approximation of the loss

To make the problem tractable, it is important to approximate the loss L as an explicit function of the combination matrix $\mathbf{H}$. Similar to local methods, the loss L is approximated using a second-order Taylor expansion around a chosen reference point. However, $\mathbf{c}$ is selected as the independent variable instead of $\mathbf{u}$, to directly use the feedback results from closed-loop operation. The simplified loss becomes (Halvorsen et al., 2003)

$$L \approx \frac{1}{2}\mathbf{e}_c^T \mathbf{J}_{cc} \mathbf{e}_c, \tag{2.45}$$

where $\mathbf{e}_c \triangleq \mathbf{c} - \mathbf{c}^{\mathrm{opt}}$ is the deviation of $\mathbf{c}$ from its optimal value $\mathbf{c}^{\mathrm{opt}}$, and $\mathbf{J}_{cc}$ is the Hessian of J with respect to $\mathbf{c}$. Using $\mathbf{c} = \mathbf{H}\mathbf{y} = \mathbf{H}(\mathbf{y}_{\mathrm{m}} - \mathbf{n})$ and the feedback result $\mathbf{H}\mathbf{y}_{\mathrm{m}} = 0$, we get $\mathbf{c} = -\mathbf{H}\mathbf{n}$. Furthermore, the optimal value of the controlled variables corresponds to $\mathbf{c}^{\mathrm{opt}} = \mathbf{H}\mathbf{y}^{\mathrm{opt}}$. Therefore, $\mathbf{e}_c$ can be expressed as

$$\mathbf{e}_c = -\mathbf{H}(\mathbf{y} + \mathbf{n}). \tag{2.46}$$

Inserting (2.46) into (2.45), we obtain an explicit expression of the loss L in terms of the combination matrix $\mathbf{H}$:

$$L \approx \frac{1}{2}(\mathbf{y}^{\mathrm{opt}} + \mathbf{n})^{\mathrm{T}} \mathbf{H}^{\mathrm{T}} \mathbf{J}_{cc} \mathbf{H} (\mathbf{y}^{\mathrm{opt}} + \mathbf{n}). \tag{2.47}$$

Here, the Hessian matrix $\mathbf{J}_{cc}$ is defined as in Halvorsen et al. (2003),

$$\mathbf{J}_{cc} = (\mathbf{H}\mathbf{G}_y)^{-T} \mathbf{J}_{uu} (\mathbf{H}\mathbf{G}_y)^{-1}, \tag{2.48}$$

where $\mathbf{G}_y$ is the gain matrix from the inputs $\mathbf{u}$ to the measurements $\mathbf{y}$, and $\mathbf{J}_{uu}$ is the reduced Hessian of J with respect to $\mathbf{u}$.

Global average loss minimization

A central idea in the gSOC approach of Ye et al. (2015) is the decomposition of the loss L into two contributions: L^d due to disturbances and L^n due to sensor noise, such that the total loss is $L = L^d + L^n$. By substituting (2.47) into (2.42), the global average loss can be derived as

$$L_{\text{av}}(\mathbf{H}) = \mathbb{E}[L^d] + \mathbb{E}[L^n], \tag{2.49}$$

where

$$L^d = \frac{1}{2}(\mathbf{y}^{\text{opt}})^{\text{T}}\mathbf{H}^{\text{T}}\mathbf{J}_{cc}\mathbf{H}\mathbf{y}^{\text{opt}}, \quad L^n = \frac{1}{2}\text{tr}(\mathbf{W}^2\mathbf{H}^{\text{T}}\mathbf{J}_{cc}\mathbf{H}). \tag{2.50}$$

A proof of the proposition above is given in the original paper by Ye et al. (2015). Here, $\text{tr}(\,\cdot\,)$ represents the trace of a matrix and $\mathbf{W}^2 \triangleq E(\mathbf{n}\mathbf{n}^{\text{T}})$ is the covariance matrix of measurement noise. Introducing the diagonal matrix $\mathbf{W}_n$ containing the expected magnitudes of the measurement errors, it can be shown that $\mathbf{W}^2 = \mathbf{W}_n^2$ for normally distributed sensor noise realizations, while $\mathbf{W}^2 = 1/3\mathbf{W}_n^2$ for uniform distributions.

To further simplify the problem, $\mathbf{J}_{cc}$ is relaxed as a constant matrix so that the loss $L \approx \frac{1}{2}\mathbf{e}_c^{\text{T}}\mathbf{J}_{cc}\mathbf{e}_c$ is approximated by a quadratic function in terms of the controlled variables, as in local approaches. However, the linearization here is based on data from the whole operating region such that the Euclidean norm of $\mathbf{e}_c$ is minimized. Based on this assumption, the condition $\mathbf{J}_{cc} \approx \mathbf{I}$ is enforced by imposing an equivalent constraint $\mathbf{H}\mathbf{G}_{y,\text{ref}} = \mathbf{J}_{uu,\text{ref}}^{1/2}$ at a chosen reference point. The equivalence between both constraints follows from (2.48). The trick of adding a constraint does not change the problem because it exploits the non-uniqueness property of the optimal $\mathbf{H}$, whose structure is preserved upon left-multiplication with any non-singular matrix. Under this assumption, the expected value of L^n becomes

$$\begin{aligned}\mathbb{E}[L^n] &= \frac{1}{2}E[\text{tr}(\mathbf{W}^2\mathbf{H}^{\text{T}}\mathbf{H})],\\ &= \frac{1}{2}E[\text{tr}(\mathbf{H}\mathbf{W}^2\mathbf{H}^{\text{T}})],\\ &= \frac{1}{2}\left\|\mathbf{W}\mathbf{H}^{\text{T}}\right\|_{\text{F}}^2.\end{aligned} \tag{2.51}$$

The expected value of L^d with $\mathbf{J}_{cc} = \mathbf{I}$ is estimated by sampling the disturbance space into N disturbance scenarios, using Monte Carlo simulations:

$$\begin{aligned}\mathbb{E}[L^d] &= \frac{1}{2}E[(\mathbf{y}^{\text{opt}})^{\mathrm{T}}\mathbf{H}^{\mathrm{T}}\mathbf{H}\mathbf{y}^{\text{opt}}],\\ &\approx \frac{1}{2N}\sum_{i=1}^{N}(\mathbf{y}_{(i)}^{\text{opt}})^{\mathrm{T}}\mathbf{H}^{\mathrm{T}}\mathbf{H}\mathbf{y}_{(i)}^{\text{opt}}.\end{aligned} \tag{2.52}$$

The subscript $(\,\cdot\,)_{(i)}$ denotes variables corresponding to the ith disturbance scenario $\mathbf{d}_{(i)}$. The expression above for $\mathbb{E}[L^d]$ can be rewritten by introducing the matrix $\mathbf{Y}$,

$$\mathbf{Y} = \begin{bmatrix}(\mathbf{y}_{(1)}^{\text{opt}})^{\mathrm{T}}\\ (\mathbf{y}_{(2)}^{\text{opt}})^{\mathrm{T}}\\ \vdots\\ (\mathbf{y}_{(N)}^{\text{opt}})^{\mathrm{T}}\end{bmatrix} = \begin{bmatrix}1 & y_{1,(1)}^{\text{opt}} & \cdots & y_{n_y,(1)}^{\text{opt}}\\ 1 & y_{1,(2)}^{\text{opt}} & \cdots & y_{n_y,(2)}^{\text{opt}}\\ \vdots & \vdots & & \vdots\\ 1 & y_{1,(N)}^{\text{opt}} & \cdots & y_{n_y,(N)}^{\text{opt}}\end{bmatrix}, \tag{2.53}$$

whose ith row vector contains the optimal noise-free measurements for a given disturbance scenario $\mathbf{d}_{(i)}$. Therefore, rearranging (2.52) in terms of (2.53), the expectation value of L^d becomes

$$\begin{aligned}\mathbb{E}[L^d] &= \frac{1}{2N}\operatorname{tr}(\mathbf{H}\mathbf{Y}^{\mathrm{T}}\mathbf{Y}\mathbf{H}^{\mathrm{T}}),\\ &= \frac{1}{2N}\left\|\mathbf{Y}\mathbf{H}^{\mathrm{T}}\right\|_{\mathrm{F}}^{2}.\end{aligned} \tag{2.54}$$

An approximation of the global average loss is derived by combining the expressions above, (2.54) and (2.51), corresponding to the expected values of L^d and L^n, respectively.

$$\begin{aligned}L_{\text{av}}(\mathbf{H}) &= \mathbb{E}(L^d) + \mathbb{E}(L^n),\\ &= \frac{1}{2N}\left\|\mathbf{Y}\mathbf{H}^{\mathrm{T}}\right\|_{\mathrm{F}}^{2} + \frac{1}{2}\left\|\mathbf{W}\mathbf{H}^{\mathrm{T}}\right\|_{\mathrm{F}}^{2},\\ &= \frac{1}{2}\left\|\tilde{\mathbf{Y}}\mathbf{H}^{\mathrm{T}}\right\|_{\mathrm{F}}^{2},\end{aligned} \tag{2.55}$$

where the intermediate matrix $\tilde{\mathbf{Y}}$ is constructed as

$$\tilde{\mathbf{Y}} = \begin{bmatrix}\frac{1}{\sqrt{N}}\mathbf{Y}\\ \mathbf{W}\end{bmatrix}. \tag{2.56}$$

Finally, by combining the approximated global average loss (2.55) together with the constraint $\mathbf{H}\mathbf{G}_{y,\text{ref}} = \mathbf{J}_{uu,\text{ref}}^{1/2}$ introduced before, the following convex optimization problem for global average loss minimization is formulated:

$$\begin{aligned} \min_{\mathbf{H}} \quad & L_{\text{av}}(\mathbf{H}) = \frac{1}{2}\left\|\tilde{\mathbf{Y}}\mathbf{H}^{\text{T}}\right\|_{\text{F}}^{2} \\ \text{s.t.} \quad & \mathbf{H}\mathbf{G}_{y,\text{ref}} = \mathbf{J}_{uu,\text{ref}}^{1/2}. \end{aligned} \tag{2.57}$$

An analytical solution to this problem is given as

$$\mathbf{H}^{\text{T}} = (\tilde{\mathbf{Y}}^{\text{T}}\tilde{\mathbf{Y}})^{-1}\mathbf{G}_{y,\text{ref}}(\mathbf{G}_{y,\text{ref}}^{\text{T}}(\tilde{\mathbf{Y}}^{\text{T}}\tilde{\mathbf{Y}})^{-1}\mathbf{G}_{y,\text{ref}})^{-1}\mathbf{J}_{uu,\text{ref}}^{1/2}. \tag{2.58}$$

To sum up, the main steps involved in the gSOC short-cut algorithm for selecting global controlled variables are presented below:

Algorithm 1 Global controlled variable selection methodology by Ye et al. (2015)

```
1: Sample $N$ finite disturbance scenarios through Monte Carlo simulation
2: For each $\mathbf{d}_{(i)}$, calculate $\mathbf{y}_{(i)}^{\text{opt}}$ via off-line optimization (minimizing $J$)
3: Evaluate $\mathbf{G}_{y,\text{ref}}$ and $\mathbf{J}_{uu,\text{ref}}$ at a reference point
4: $\mathbf{W}^2 = \mathbf{W}_n^2$ for Gaussian and $\mathbf{W}^2 = \frac{1}{3}\mathbf{W}_n^2$ for uniform distributions
5: Construct $\mathbf{Y}$ and $\tilde{\mathbf{Y}}$ from (2.53) and (2.56), respectively
6: Compute optimal $\mathbf{H}$ using (2.58)
```

2.4.5 The problem of measurement subset selection

This section discusses the selection of optimal subsets of measurements, which is of great importance in practical applications. Local and global methods presented before address the following problem:

- Given a set of available measurements $\mathbf{y}$, find the optimal combination matrix $\mathbf{H}$ such that, by controlling $\mathbf{c} = \mathbf{H}\mathbf{y}$ to a constant setpoint of $\mathbf{c}_{\text{s}}$, the average loss for all expected disturbances $\mathbf{d} \in \mathcal{D}$ and sensor noise realizations $\mathbf{n} \in \mathcal{N}$ is minimized.

In this section, we add another restriction to the problem:

- Given a set of available measurements $\mathbf{y}$, find the optimal subset of n_y measurements that results in minimum average loss applying the methods considered before.

In a complete chemical plant, the number of possible measurements can be in the order of hundreds or thousands and using all of them when designing the control structure is not desired nor required.

The problem is to find the optimal trade-off between the benefits of adding more sensors (better disturbance rejection and less impact of sensor noise) and the associated costs (increased investment and control complexity). If we plot the average loss of optimality as a

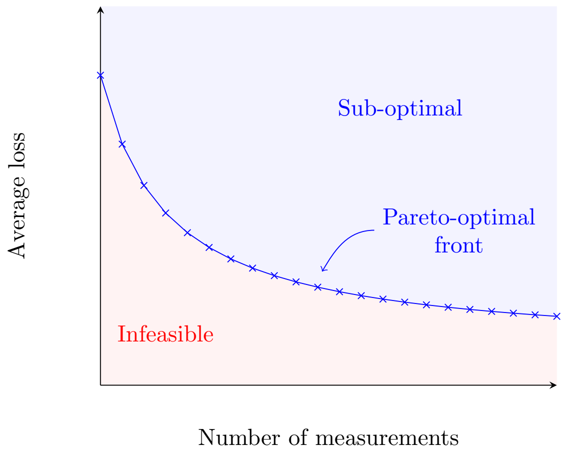

Figure 2.4: Pareto frontier representing the optimal set of measurements for a given control structure (Verheyleweghen and Jäschke, 2019).

function of the number of measurements selected in the control structure, the trend follows a Pareto frontier (Kariwala and Cao, 2009), as the one shown in Fig. 2.4. Above certain threshold, the reduction in loss becomes insignificant with the number of measurements and therefore the addition of a new sensor does not offset the investment cost and the increased complexity of the control structure.

The problem of selecting subsets of measurements is combinatorial in nature. To solve the problem, two approaches are currently used in the literature. One option is to use tailor-made branch and bound algorithms (Kariwala and Cao, 2009). The second approach proposed by Yelchuru and Skogestad (2012) consists of formulating the problem as a mixed integer quadratic programming (MIQP) problem and use standard MIQP solvers to find the solution. This second approach will be the one used in this work.

Mixed integer quadratic programming (MIQP) formulation

The first step in the method proposed by Yelchuru and Skogestad (2012) is to transform the quadratic programming (QP) problem considered in the minimum loss method,

$$\begin{aligned} \min_{\mathbf{H}} \quad & \|\mathbf{H}\tilde{\mathbf{F}}\|_{\mathrm{F}} \\ \text{s.t.} \quad & \mathbf{H}\mathbf{G}_y = \mathbf{J}_{uu}^{1/2}, \end{aligned} \tag{2.59}$$

into an equivalent vectorized problem,

$$\begin{aligned} \min_{\mathbf{h}_\delta} \quad & \mathbf{h}_\delta^{\mathrm{T}} \mathbf{F}_\delta \mathbf{h}_\delta \\ \text{s.t.} \quad & \mathbf{G}_{y\delta}^{\mathrm{T}} \mathbf{h}_\delta = \mathbf{j}_\delta . \end{aligned} \tag{2.60}$$

This is done for practical reasons, because most standard solvers work with vectors and not matrix formulations. In the formulation above, the matrix $\mathbf{H} \in \mathbb{R}^{n_u \times n_y}$

$$\mathbf{H} = \begin{bmatrix} h_{11} & \cdots & h_{1n_y} \\ \vdots & \ddots & \vdots \\ h_{n_u 1} & \cdots & h_{n_u n_y} \end{bmatrix} \tag{2.61}$$

is vectorized along the rows of $\mathbf{H}$ to form the vector

$$\mathbf{h}_\delta = [\underbrace{h_{11} \ldots h_{1n_y}}_{\text{row 1}} \quad \underbrace{h_{21} \ldots h_{2n_y}}_{\text{row 2}} \quad \cdots \quad \underbrace{h_{n_u 1} \ldots h_{n_u n_y}}_{\text{row } n_u}]^{\mathrm{T}} \in \mathbb{R}^{n_u n_y \times 1}. \tag{2.62}$$

A similar vectorization procedure is applied along the columns of $\mathbf{J}_{uu}^{1/2} \in \mathbb{R}^{n_u \times n_u}$, giving $\mathbf{j}_\delta \in \mathbb{R}^{n_u n_u \times 1}$.

The matrix $\mathbf{G}_{y\delta}^{\mathrm{T}} \in \mathbb{R}^{n_u n_u \times n_y n_u}$ is a block diagonal matrix, where the matrix $\mathbf{G}_y^{\mathrm{T}}$ is repeated along the diagonal n_u times,

$$\mathbf{G}_{y\delta}^{\mathrm{T}} = \begin{bmatrix} \mathbf{G}_y^{\mathrm{T}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \mathbf{G}_y^{\mathrm{T}} \end{bmatrix} \tag{2.63}$$

The matrix $\mathbf{F}_\delta \in \mathbb{R}^{n_u n_y \times n_u n_u}$ is computed as $\mathbf{F}_\delta = \tilde{\mathbf{F}}_\delta \tilde{\mathbf{F}}_\delta^{\mathrm{T}}$, where

$$\tilde{\mathbf{F}}_\delta = \begin{bmatrix} \tilde{\mathbf{F}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \tilde{\mathbf{F}} \end{bmatrix}. \tag{2.64}$$

After this vectorization procedure, we introduce the vector of binary variables σ

$$\sigma = [\sigma_1 \quad \sigma_2 \quad \cdots \quad \sigma_{n_y}], \quad \sigma_j = \{0, 1\}, \tag{2.65}$$

where $\sigma_j = 0$ corresponds to a measurement that is excluded from the subset and $\sigma_j = 1$ corresponds to a selected measurement. The dimension of the subset can be specified by the constraint

$$\mathbf{P}\sigma = s, \tag{2.66}$$

where $\mathbf{P} = \underline{1}^{\mathrm{T}}_{1\times n_y}$ is a n_y dimensional vector of ones, and s is the number of measurements included in the subset.

Finally, the mixed integer quadratic programming (MIQP) problem to find optimal subsets of measurements corresponds to

$$\begin{array}{lrcll} \min\limits_{\mathbf{h}_\delta,\sigma} & \mathbf{h}_\delta^{\mathrm{T}}\mathbf{F}_\delta\mathbf{h}_\delta & & & \\ \text{s.t.} & \mathbf{G}_{y\delta}^{\mathrm{T}}\mathbf{h}_\delta & = & \mathbf{j}_\delta, & \\ & \mathbf{P}\sigma & = & s, & \\ & -\mathbf{M}\sigma_j \leq h_{i,j} & \leq & \mathbf{M}\sigma_j, & j=1,\dots,n_y, \quad i=1,\dots,n_u, \\ & \sigma & \in & \{0,1\}. & \end{array} \tag{2.67}$$

The constraints $-\mathbf{M}\sigma_j \leq h_{i,j} \leq \mathbf{M}\sigma_j$ are known as big-M constraints and their mission is to ensure that the elements in $\mathbf{H}$ are zeros whenever $\sigma_j = 0$. $\mathbf{M} \in \mathbb{R}_+^{n_u}$ is a vector of positive constants that act as upper bounds on the elements in $\mathbf{H}$. Selecting appropriate values for $\mathbf{M}$ is not straightforward and it is often done on a trial and error basis. Low values can result in a false active constraint and a suboptimal solution, while large values can become very computationally demanding. One strategy is to reduce iteratively the values in $\mathbf{M}$ until no changes are seen in the solution. Examples of standard MIQP commercial solvers are CPLEX (CPLEX, 2009) and GUROBI (Gurobi, 2014).

2.4.6 The problem of handling active set changes

In general, the set of active constraints may change during operation due to disturbances. This situation is illustrated in Fig. 2.5. To handle this problem within the framework of self-optimizing control, different approaches have been proposed in the literature. The purpose of this section is not an extensive review of these methods, but to provide references to the reader.

One approach is to decompose the disturbance space into different active set regions and design self-optimizing control structures for each region based on the methods presented so far. Then, implement switching laws to change between control structures when disturbances make the active set change. This is the idea of the multi-parametric programming approach by Manum and Skogestad (2012). The applicability of this approach is limited by the size of the problem because the number of potential active set regions grows exponentially with the number of constraints.

Another approach, more in line with the spirit of self-optimizing control due to its simplicity, is the integrated approach by Hu et al. (2012). Here, the goal is to find a single control structure that is feasible and minimized the average loss for all disturbances.

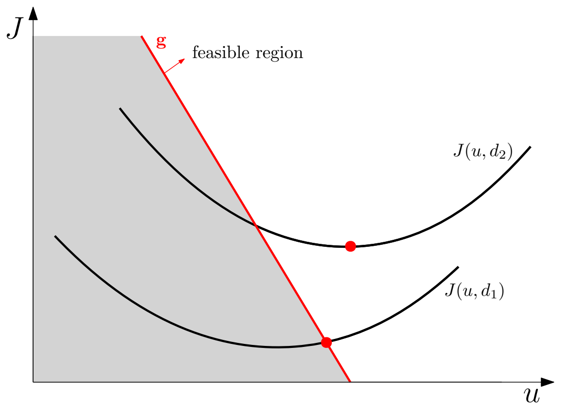

Figure 2.5: Illustration of the effect of disturbances in active set changes. For disturbance d_1, the optimum lies at the constraint, while for disturbance d_2 the optimum is unconstrained (Krishnamoorthy and Skogestad, 2019).

For cases when the set of active constraints is not likely to change frequently and the number of constraints is less than the number of controlled variables, the cascade approach by Cao (2004) may be implemented.

However, handling active set changes within the framework of self-optimizing control can still be challenging and it is one of the open issues for further research in this field (Jäschke et al., 2017).

2.5 Controller design in multivariable plants

This section covers some basic concepts on the design of two important types of controllers used in multivariable plants: decentralized proportional-integral-derivative (PID) controllers and model predictive controllers (MPC). The regulatory control layer is usually composed of single-loop PID controllers to stabilize the plant, giving robust performance without the need for a detailed plant model. In the upper supervisory control layer, advanced control structures involving PID controllers and logic elements such as selectors have been traditionally used. However, in the last decades, model predictive control has become the most widespread alternative due to its ability to explicitly handle process constraints and interactions in multivariable systems. The choice of one approach over the other will mainly depend on the specific process. The improvement in performance resulting from a multivariable controller needs to offset the cost of modeling and controller design.

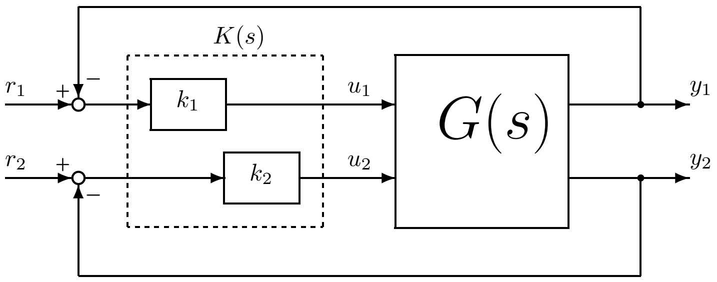

Figure 2.6: Decentralized control of a 2×2 plant. (Skogestad and Postlethwaite, 2007).

2.5.1 Decentralized PID control

The simplest approach to control multiple-input-multiple-output (MIMO) plants is to decompose the plant into a series of subplants controlled by local control units. An example of such a decentralized or diagonal controller for a 2×2 plant is shown in Figure 2.6. The performance of a decentralized controller is limited by the plant interactions. Although the use of decouplers can minimize the interactions between the control loops, for highly interactive plants the use of multivariable controllers is recommended. Model predictive control (MPC) will be the topic of the next section.

For plants where a decentralized control strategy can be realized, the controller design typically involves two steps:

- The choice of pairings (control configuration selection).
- The design (tuning) of each controller.

Input-output pairing

Inputs (manipulated variables, MVs) and outputs (controlled variables, CVs) should be paired such that interactions between the control loops are minimized. A tool used in control configuration selection to quantify these interactions is the relative gain array (RGA) matrix. It is defined as

$$\text{RGA}(\mathbf{G}) \equiv \mathbf{\Lambda}(\mathbf{G}) \triangleq \mathbf{G} \times (\mathbf{G}^{-1})^{\text{T}}, \tag{2.68}$$

where $\mathbf{G}$ is a non-singular square matrix that represents the plant gain from inputs to outputs, and $\times$ denotes the element wise multiplication operator or Hadamard product. Some important properties of the RGA matrix are that columns and rows always sum to 1 and that the matrix is independent from the scaling chosen for the inputs and outputs.

A derivation of the RGA matrix is given in Skogestad and Postlethwaite (2007). For a given plant $\mathbf{G}(s)$ and a control loop defined by the input-output pair u_j-y_i, we consider the following two extreme cases:

- All other loops are open:

$$u_k = 0 \quad \forall \quad k \neq j \quad , \quad g_{ij} = \left(\frac{\partial y_i}{\partial u_j}\right)_{u_{k\neq j}=0} \tag{2.69}$$

- All other loops are perfectly controlled:

$$y_k = 0 \quad \forall \quad k \neq i \quad , \quad \hat{g}_{ij} = \left(\frac{\partial y_i}{\partial u_j}\right)_{y_{k\neq i}=0} \tag{2.70}$$

where g_{ij} and $\hat{g}_{ij}$ are the open and closed-loop gains of the pair u_j-y_i. The elements in the RGA matrix correspond to the ratio between these gains,

$$\lambda_{ij} = \frac{g_{ij}}{\hat{g}_{ij}}. \tag{2.71}$$

The value of λ_{ij} provides a measure of the interactions between the pair and the rest of the control loops. RGA-elements greater than one, $\lambda_{ij} > 1$, imply a gain reduction when other loops are closed in the plant, making control slower and more difficult due to “fighting loops”. On the other side, RGA-elements smaller than one, $\lambda_{ij} < 1$ imply a gain increase when other loops are closed, which can result in a dangerous situation. Furthermore, negative RGA-elements, $\lambda_{ij} < 0$, result in gain changes when other loops are closed, causing instability.

Skogestad and Postlethwaite (2007) suggest two rules for input-output selection based on the RGA matrix. The first rule is to avoid pairing on negative steady-state RGA-elements to not get instability if one of the loops becomes inactive (e.g., due to saturation). The second rule is to select pairings corresponding to RGA-elements close to 1, so that interactions between control loops are minimized.

Controller tuning

The PID controller equation in the time domain can be written as

$$\mathbf{u}(t) = \mathbf{u}_0 + K_c\left(\mathbf{e}(t) + \frac{1}{\tau_I}\int_0^t \mathbf{e}(\tau)\,\mathrm{d}\tau + \tau_D\,\frac{\mathrm{d}}{\mathrm{d}t}\,\mathbf{e}(t)\right), \tag{2.72}$$

where $\mathbf{u}_0$ is the bias term; $\mathbf{e}(t) = \mathbf{y}_\mathrm{s}(t) - \mathbf{y}(t)$ is the error vector, which measures the deviation of the controlled variables from their desired setpoints; K_c is the controller gain; τ_I is the integral time; and τ_D is the derivative time.

The tuning problem is to find appropriate values for the controller settings K_c, τ_I and τ_D. Several model-based PID tuning methods have been proposed, including the classical method by Ziegler and Nichols (1942), the internal model control (IMC-PID) tuning method by Rivera et al. (1986), the direct synthesis tuning method by Smith and Corripio

(1985) and the simple internal model control (SIMC-PID) method by Skogestad (2003). The SIMC-PID tuning method by Skogestad (2003) will be used in this work because it results in simple PI/PID settings; works well for a wide range of processes, including integrating and pure time delay processes; and provides satisfactory performance both for setpoint tracking and disturbance rejection.

The first step of the SIMC method is to approximate the process by a first or second-order plus delay model. On transfer function form, a first-order model corresponds to

$$g_1(s) = \frac{k}{(\tau_1 s + 1)} e^{-\theta s}, \tag{2.73}$$

where k is the process gain, τ_1 is the dominant time constant, and θ is the effective time delay. For a second-order model,

$$g_2(s) = \frac{k}{(\tau_1 s + 1)(\tau_2 s + 1)} e^{-\theta s}, \tag{2.74}$$

where τ_2 is the second-order time constant (used to describe second-order processes where $\tau_2 > \theta$, approximately). These process parameters can be obtained from open-loop step responses, closed-loop setpoint responses, or by approximating a detailed model using the so-called "half-rule". For a detailed explanation of these methods, the reader is referred to Skogestad (2003); Skogestad and Grimholt (2012).

Based on the model formulations above, the SIMC method for a first-order model results in a PI controller with settings

$$K_c = \frac{1}{k} \frac{\tau_1}{(\tau_c + \theta)}, \tag{2.75}$$

$$\tau_I = \min\{\tau_1, 4(\tau_c + \theta)\}. \tag{2.76}$$

For a second-order model, the SIMC method results in a PID controller with settings

$$K_c = \frac{1}{k} \frac{\tau_1}{(\tau_c + \theta)}, \tag{2.77}$$

$$\tau_I = \min\{\tau_1, 4(\tau_c + \theta)\}, \tag{2.78}$$

$$\tau_D = \tau_2. \tag{2.79}$$

These PID settings are derived for the cascade (series) form of the PID controller equation in the Laplace domain:

$$c(s) = K_c \left(\frac{\tau_I s + 1}{\tau_I s} \right) (\tau_D s + 1) \,. \tag{2.80}$$

The SIMC method reduces the tuning problem to only one tuning parameter: the closed-loop time constant τ_c. A small value of τ_c favors fast speed of response and good disturbance rejection; while a large value of τ_c favors stability, robustness, and small input variation. Skogestad (2003) recommends to select τ_c equal to the time delay, $\tau_c = \theta$, to achieve a good trade-off between fast response, moderate input variation, and good robustness margins.

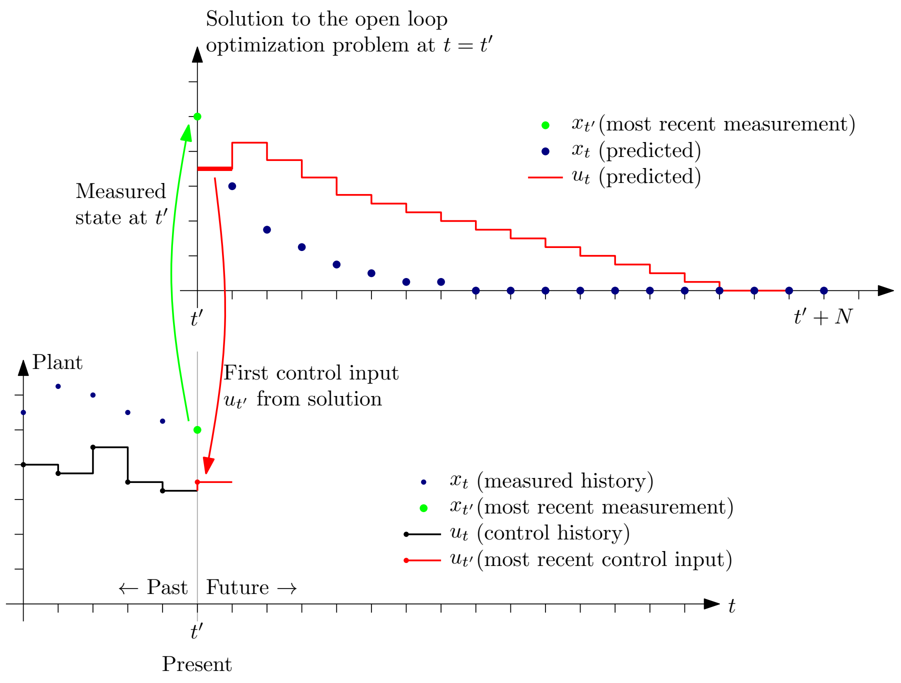

Figure 2.7: Illustration of the model predictive control (MPC) principle (Foss and Heirung, 2013).

2.5.2 Model predictive control

Model predictive control (MPC) is a widely used control technology with industrial applications due to its ability to handle constraints in the manipulated and controlled variables. An online optimization is solved iteratively to minimize an objective function with respect to current and future control moves. A plant model is used to predict the future plant behavior over a time period called prediction horizon p given these control moves over a time period called control horizon c. The goal of the optimization is to compute the current and future control moves in a way that the future output is driven as close to a desired setpoint or trajectory as possible while maintaining a penalty on the control moves. The optimization problem solved by the MPC algorithm is typically formulated as

$$\begin{aligned} \min_{\Delta\mathbf{u}} \quad & (\mathbf{y}_{\text{sp}} - \hat{\mathbf{y}})^{\mathrm{T}}\mathbf{W}_y(\mathbf{y}_{\text{sp}} - \hat{\mathbf{y}}) + \Delta\mathbf{u}^{\mathrm{T}}\mathbf{W}_u\Delta\mathbf{u} \\ \text{s.t.} \quad & \mathbf{y}_{\min} \le \hat{\mathbf{y}} \le \mathbf{y}_{\max}, \\ & \mathbf{u}_{\min} \le \mathbf{u} \le \mathbf{u}_{\max}, \\ & \Delta\mathbf{u}_{\min} \le \Delta\mathbf{u} \le \Delta\mathbf{u}_{\max}, \end{aligned} \tag{2.81}$$

where $\mathbf{y}_{\text{sp}}$ is the vector of output setpoint values, $\hat{\mathbf{y}}$ is the vector of predicted outputs, $\Delta\mathbf{u}$ are the future control moves given as deviations from the inputs of the previous time instance, and the matrices $\mathbf{W}_y$ and $\mathbf{W}_u$ are the relative weights in the objective function. Only the first control move is implemented and the problem is solved iteratively in a receding horizon fashion. The MPC principle is illustrated in Figure 2.7.

Dynamic matrix control (DMC) (Cutler and Ramaker, 1979) is an MPC algorithm that provides an approach to obtain the predictions $\hat{\mathbf{y}}$. A linear input-output model is constructed by performing step changes in the inputs or manipulated variables (MVs) of the plant and recording the response of the plant outputs or controlled variables (CVs). Using an input-output model offers the advantage of decoupling the model size from the actual system size for large scale distributed systems such as the one examined in this work resulting in low computational costs, and using an input-output model in (2.81) yields a quadratic program. A more sophisticated version of DMC, quadratic dynamic matrix control (QDMC) is used in this work. For details regarding these methodologies, the reader is referred to Cutler and Ramaker (1979); Garcia and Morshedi (1986); Ikonen (2017).

2.6 Continuous-flow production of pharmaceuticals

This last section is the most applied of the chapter and gives some basic background for the case study considered in this work. In particular, it motivates some recent research efforts in continuous-flow pharmaceutical manufacturing.

The production of pharmaceuticals is typically done using batch processing at multiple sites. Most of the molecular fragments involved in the process synthesis are provided by different sources and only the final synthesis steps are carried out at the company site. The entire process chain, from raw materials to the final drug form, can take up to a total of 12 months. This long production time is one of the main limitations of the current batch-wise approach. Other important challenges facing the pharmaceutical industry today include drug shortages due to supply chain interruptions and quality control issues, the need to respond to sudden changes in demand (e.g., due to epidemics or pandemics), high drug production costs, and high waste generation (Adamo et al., 2016).

To address these issues, important research efforts in academia and industry over the last decade have been oriented towards the development of continuous-flow processes for drug production (Mascia et al., 2013; Baxendale et al., 2015; Myerson et al., 2015). Furthermore, compact reconfigurable platforms for on-demand pharmaceutical manufacturing have been designed, see Adamo et al. (2016). On them, end-to-end synthesis from simple inexpensive molecules to the final active pharmaceutical ingredient is carried out in one single platform. Such systems have the potential to reduce manufacturing times and overcome the major drawbacks of batch-wise production listed above.

Chapter 3

Case study and methodology

3.1 The case study

3.1.1 Process description

The case study for this work is the continuous flow synthesis of atropine. Atropine is an active pharmaceutical ingredient (API) with a variety of therapeutic uses, including the treatment of heart rhythm problems (North and Kelly, 1987). A two dimensional view of the atropine molecule is shown in Fig. 3.1.

A flow synthetic route for atropine was reported in 2015 by Dai et al. (2015). Unfortunately, the process suffered from the production of large amounts of undesired byproducts, resulting in a low overall yield of atropine obtained (8%).

A way to quantify the efficiency in a chemical manufacturing process accounting for the waste production is through the so-called environmental factor, often abbreviated as E-factor. This metric is defined as the ratio of the mass of waste produced (excluding water,

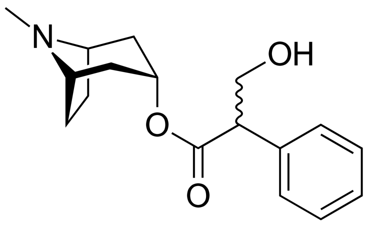

Figure 3.1: Atropine molecule ($C_{17}H_{23}NO_3$)

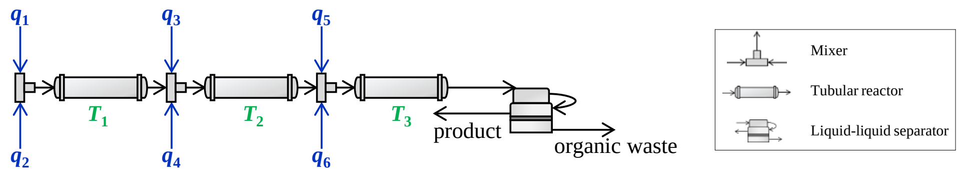

Figure 3.2: Process flowsheet. Adapted from (Nikolakopoulou et al., 2019).

which is not environmentally harmful) to that of product obtained, i.e.,

$$\text{E-factor} = \frac{\text{mass of waste produced (excluding water)}}{\text{mass of product obtained}}. \tag{3.1}$$

In the previous process formulation reported by Dai et al. (2015), the E-factor was 2245, meaning that for each kilogram of atropine obtained, 2245 kilograms of undesired byproducts were produced. Two years later, in 2017, a new process synthesis was proposed by Bédard et al. (2017), giving an E-factor of 24. This represents a reduction of two orders of magnitude in waste generation.

This study builds on the experimental results reported by Bédard et al. (2017) to construct an improved version of a recently published first-principles model of the continuous flow synthesis of atropine (Nikolakopoulou et al., 2019). A flowsheet of the process is shown in Fig. 3.2.

The process is composed by three mixers, three tubular reactors and a liquid-liquid separator unit. First, streams q_1 (tropine in dimethylformamide) and q_2 (phenylacetylchloride) are mixed and then flow to the first tubular reactor, which is kept at temperature T_1 by a lower regulatory control layer. Streams q_3 (formaldehyde in methanol and water) and q_4 (sodium hydroxide in water) are mixed with the outlet stream of the first tubular reactor before flowing to the second tubular reactor with temperature T_2. Subsequently, streams q_5 (buffer) and q_6 (toluene) are mixed with the the outlet stream from the second tubular reactor and flow to the third tubular reactor with temperature T_3. Finally, the atropine is extracted in the aqueous phase of the liquid-liquid separator.

The overall process involves a total of 14 chemical components, which are listed in Table 3.1. Atropine is produced according to the following reaction set (Eq. 3.2):

$$\mathrm{C_8H_{15}NO + C_8H_7ClO \rightarrow C_{16}H_{21}O_2NHCl} \tag{3.2a}$$

$$\mathrm{C_{16}H_{21}O_2NHCl + NaOH \rightarrow H_2O + C_{16}H_{21}O_2N + NaCl} \tag{3.2b}$$

$$\mathrm{CH_2O + C_{16}H_{21}O_2N \rightarrow \underbrace{C_{17}H_{23}NO_3}_{\text{atropine}}} \tag{3.2c}$$

Table 3.1: Chemical process components.

Component	**Chemical formula**	**Ref.**
Tropine	$C_8H_{15}NO$	1
Dimethylformamide	C_3H_7NO	2
Phenylacetyl chloride	C_8H_7ClO	3
Intermediate	$C_{16}H_{21}O_2NHCl$	4
Formaldehyde	CH_2O	5
Methanol	CH_3OH	6
Sodium hydroxide	$NaOH$	7
Water	H_2O	8
Atropine	$C_{17}H_{23}NO_3$	9
Apoatropine	$C_{17}H_{21}NO_2$	10
Tropine ester	$C_{16}H_{21}O_2N$	11
Sodium chloride	$NaCl$	12
Buffer	NH_4Cl	13
Toluene	C_7H_8	14

$$CH_2O + C_{16}H_{21}O_2N \rightarrow C_{17}H_{21}NO_2 + H_2O \tag{3.2d}$$

In the first tubular reactor, only reaction (3.2a) occurs due to the absence of $NaOH$, CH_2O and $C_{16}H_{21}O_2N$ to start the remaining reactions. Another interesting observation is that reactions (3.2c) and (3.2d) are competing to produce atropine (desired product) and apoatropine (undesired byproduct), respectively.

3.1.2 Mathematical model

Each unit operation in the system is described by first-principles model equations, namely mass balances, and chemical reaction kinetics. The energy balances are not modeled since the reactors can be kept at the desired temperature setpoint using regulatory level controllers. The setpoint temperature can be achieved rapidly due to the high surface-to-volume ratio of the tubular reactors. Below is a description of the mathematical models of each unit operation.

Mixer

The mixers are in-line and are assumed to achieve perfect mixing instantaneously. The model equations represent mass conservation without accumulation for each species

$$\dot{m}_{\text{out},i} = \sum_{k=1}^{n_s} \dot{m}_{\text{in},i,k} \quad \text{for } i = 1, 2, \ldots, n_c, \tag{3.3}$$

where $\dot{m}_{\text{out},i}$ is the outlet mass flowrate of species i, $\dot{m}_{\text{in},i,k}$ is the inlet mass flowrate of species i in the inlet stream k, n_s is the number of feed streams, and n_c is the number of species. The overall mass balance within each mixer unit is

$$\dot{m}_{\text{out,tot}} = \sum_{i=1}^{n_c} \dot{m}_{\text{out},i}, \tag{3.4}$$

and the concentration of species i,

$$c_{\text{out},i} = \frac{\dot{m}_{\text{out},i}\rho}{\dot{m}_{\text{out,tot}} M_i} \quad \text{for } i = 1, 2, \ldots, n_c, \tag{3.5}$$

where ρ is density of the solution and M_i is the molecular mass of species i.

Tubular Reactor

The tubular reactors are modeled using partial differential equations (PDEs), which are discretized in space using the method of lines with n_d discretization points. The mass conservation equations are described by

$$\begin{aligned} &\frac{\partial c_{i,l}}{\partial t} = -Q_{\text{tot}} \frac{\partial c}{\partial V}\bigg|_{i,l} + r_{i,l} \\ &\frac{\partial c}{\partial V}\bigg|_{i,l} = \frac{c_{i,l} - c_{i,l-1}}{V_l - V_{l-1}} \\ &\text{for } i = 1, 2, \ldots, n_c,\ l = 1, 2, \ldots, n_d, \end{aligned} \tag{3.6}$$

where $c_{i,l}$ is the concentration of species i at discretization point l, Q_{tot} is the total volumetric flowrate, $r_{i,l}$ is the reaction rate of species i at discretization point l, and V_l is the volume from the entrance of the reactor to discretization point l. The second equation approaches $\partial c/\partial V$ with a first-order spatial discretization. The concentration at the inlet of the reactor, which is represented as the 0th discretization, is known and can be obtained from the outlet of the unit upstream

$$c_{i,0} = c_{\text{in},i} = \frac{\dot{m}_{\text{in},i}}{Q_{\text{tot}} M_i} \quad \text{for } i = 1, 2, \ldots, n_c. \tag{3.7}$$

The total volumetric flowrate in the reactor is

$$Q_{\text{tot}} = \frac{\dot{m}_{\text{in,tot}}}{\rho}, \tag{3.8}$$

where $\dot{m}_{\text{in,tot}}$ is the total mass flowrate. The volumetric mass density is calculated assuming additive volumes since most species are in dilute concentration in the solution,

$$\frac{1}{\rho} = \sum_{i=1}^{nc} \frac{x_i}{\rho_i}, \tag{3.9}$$

where x_i is the mass fraction of species i, and ρ_i the density of pure species i. The density and consequently the flowrate do not vary across the axial dimension of the reactor.

The reaction rate matrix $\mathbf{r} \in \mathbb{R}^{n_c \times n_d}$ returns the reaction rate of each species at each discretization point and is given by

$$\mathbf{r} = \mathbf{SR}, \tag{3.10}$$

where $\mathbf{S} \in \mathbb{R}^{n_c \times n_r}$ is the stoichiometric matrix, $\mathbf{R} \in \mathbb{R}^{n_r \times n_d}$ is the reaction matrix, and n_r is the number of reactions. The reaction rates are assumed to follow the Arrhenius law. The reaction rate of the jth reaction at discretization point l is

$$\begin{aligned} &R_{j,l} = k_j \exp\left(\frac{-E_{\mathrm{A},j}}{RT}\right) \prod_{m \in \mathcal{M}_j} c_{m,l}^{o_{m,j}} \\ &\text{for } j = 1, 2, \ldots, n_r,\ l = 1, 2, \ldots, n_d, \end{aligned} \tag{3.11}$$

where k_j is the pre-exponential factor, a constant for each reaction j, $E_{\mathrm{A},j}$ is the activation energy, R is the ideal gas constant, T is the temperature, $\mathcal{M}_j$ is the set of reactant species for reaction j, $c_{m,l}$ is the concentration of reactant species m at the discretization point l, and $o_{m,j}$ is the reaction order of reactant species m in reaction j.

The parameters k_j and $E_{\mathrm{A},j}$ for each reaction are estimated from the experimental data from Bédard et al. (2017). First order reaction kinetics with respect to each species are assumed for all four reactions in the atropine synthesis process

$$\begin{aligned} R_{1,l} &= k_1 \exp\left(\frac{-E_{\mathrm{A},1}}{RT}\right) c_{1,l} c_{3,l} \\ R_{2,l} &= k_2 \exp\left(\frac{-E_{\mathrm{A},2}}{RT}\right) c_{4,l} c_{7,l} \\ R_{3,l} &= k_3 \exp\left(\frac{-E_{\mathrm{A},3}}{RT}\right) c_{5,l} c_{11,l} \\ R_{4,l} &= k_4 \exp\left(\frac{-E_{\mathrm{A},4}}{RT}\right) c_{5,l} c_{11,l}. \end{aligned} \tag{3.12}$$

The reaction network is given in Bédard et al. (2017) and the numbering of the species follows their order of appearance in these reactions. These numbers can also be checked in Table 3.1.

The mass flowrates at the outlet of the reactor are

$$\begin{aligned}
&\dot{m}_{\text{out},i} = c_{i,n_d} Q_{\text{tot}} M_i \quad \text{for } i = 1, 2, \ldots, n_c \\
&\dot{m}_{\text{out,tot}} = \sum_{i=1}^{nc} \dot{m}_{\text{out},i}.
\end{aligned} \tag{3.13}$$

Liquid-Liquid Separator

To describe the dynamics of the liquid-liquid separator unit an effective average molar concentration $\bar{c}_i$ for each species i is introduced, such that

$$Q_{\text{tot}} \bar{c}_i = F_{\text{OR},i} + F_{\text{AQ},i} \quad \text{for } i = 1, 2, \ldots, n_c, \tag{3.14}$$

$$V \frac{\mathrm{d}\bar{c}_i}{\mathrm{d}t} = Q_{\text{tot}} \big(c_{\text{in},i} - \bar{c}_i \big) \quad \text{for } i = 1, 2, \ldots, n_c, \tag{3.15}$$

where V is the volume of the liquid-liquid separator, F_i is the molar flowrate of species i, and the subscripts 'OR' and 'AQ' refer to the organic and aqueous phase respectively. The volume of the separator is assumed to be constant over time. An additional assumption of constant density inside the separator results in

$$Q_{\text{tot}} = Q_{\text{OR}} + Q_{\text{AQ}}. \tag{3.16}$$

For solutes that can be found the organic stream in trace quantities, it follows that

$$\frac{x_{\text{in},s}}{x_{\text{in},s} + x_{\text{in},w}} Q_{\text{tot}} = Q_{\text{OR}}, \tag{3.17}$$

where $x_{\text{in},s}$ is the mass fraction of the solvent in the inlet of the unit and $x_{\text{in},w}$ is the mass fraction of water.

Phase equilibrium that is achieved instantaneously is modeled by

$$c_{\text{OR},i} = D_i c_{\text{AQ},i} \quad \text{for } i = 1, 2, \ldots, n_c, \tag{3.18}$$

The dynamics of the mass transfer can be identified given enough data and can be included in the model as a time-delay. The separation partition coefficients D_i are obtained from Bédard et al. (2017), assuming negligible variation with temperature and concentration.

Overall the equations describing the mass conservation of the process form an index-1 DAE.

3.1.3 Operational objective

The operational objective of the process is to minimize the environmental factor (E-factor) [kg waste/kg product], while keeping the volume flowrates of the feed streams (inputs) within the specified bounds:

$$0 \text{ mL/min} \leq q_{1-4} \leq 5 \text{ mL/min} \tag{3.19}$$

The process is controlled using the volume flowrates of the feed streams that contain reagents, q_{1-4}, which are the manipulated variables. The feed streams in the third mixer, q_5 and q_6, contain a buffer solution and toluene (solvents). These variables have relatively small steady-state gains and so are not effective manipulated variables. Therefore, we assume that they are kept at their nominal values during operation, which corresponds to:

$$q_5 = 0.2 \text{ mL/min} \tag{3.20}$$

$$q_6 = 0.5 \text{ mL/min} \tag{3.21}$$

In the calculation of the E-factor, only the mass flowrate of atropine recovered in the aqueous phase of the liquid-liquid separator is regarded as product. The rest of components in both phases (aqueous and organic) are counted as waste, excluding water which is not considered harmful for the environment.

Based on this, optimal operation of the atropine process can be formulated as the following optimization problem:

$$\begin{aligned} \min_{q} \quad & J := \text{E-factor} \\ \text{s.t.} \quad & \text{Process model: (3.3)} - \text{(3.18)} \\ & \text{Operational constraints: (3.19)} - \text{(3.21)} \end{aligned} \tag{3.22}$$

To solved it, a nonlinear programming problem (NLP) was formulated and implemented in MATLAB using Casadi 3.0.0 (Andersson et al., 2012) and solved using IPOPT 3.12.3 (Biegler and Zavala, 2009).

3.1.4 Challenges of optimal operation

We consider the following challenges when designing the control structure for this process:

- *Parametric model uncertainty* in the kinetic parameters of the Arrhenius equation (pre-exponential factors k_{1-4} and activation energies $E_{A,1-4}$), and in the separation partition coefficient of atropine D_9.[1]

[1]The numbering of the components follows the one in Table 3.1.

Table 3.2: Nominal values of the disturbances and their standard deviations.

Variable	**Unit**	**Nominal value**	**Standard deviation**
M_1	mol/L	2	1% nom.
M_7	mol/L	4	1% nom.
T_1	°C	100	1
T_2	°C	100	1
T_3	°C	50	1
k_1	mol/(mL min)	34206	5% nom.
k_2	mol/(mL min)	10000	5% nom.
k_3	mol/(mL min)	24	5% nom.
k_4	mol/(mL min)	43599	5% nom.
E_{A1}	J/mol	1000	5% nom.
E_{A2}	J/mol	100	5% nom.
E_{A3}	J/mol	1819	5% nom.
E_{A4}	J/mol	26207	5% nom.
$\log(D_9)$	-	-2	0.5

- *Process perturbations* in the temperatures of the tubular reactors, T_{1-3}, and in the molarity of tropine and sodium hydroxide, M_1 and M_7, at the feed streams.
- *Sensor noise* affecting the value of concentrations, volume flowrates and temperature measurements. We assume that sensor noise is normally distributed with zero mean and standard deviation of 2.5% of the nominal value for concentrations and volume flowrates, and $1\,\mathrm{K}$ for temperatures.

In the remainder of this work, both parametric model uncertainty and process perturbations will be considered as “disturbances”, following the notation introduced in Section 2.4.1, when framing the self-optimizing control problem. Their nominal values and standard deviations are given in Table 3.2.

The goal is to design a control structure able to achieve near-optimal operation in the face of uncertainty. This uncertainty can either come from process perturbations or from an inaccurate model of the system. Plant-model mismatch is not explicitly addressed in this work, i.e., we assume that the model is structurally correct, even though the model parameters are uncertain.

3.2 Controlled variable selection methodology

This section describes the methodology used in this work to select self-optimizing controlled variables. Two different methods were used for comparison: a local method based on the ideas presented in Section 2.4.3 and a global method based on the ideas presented in Section 2.4.4. Both approaches approximate the nonlinear loss function quadratically around the nominal operating point, to make the problem tractable. Therefore, they do not provide the rigorous optimal solution, but promising controlled variables that need to be further validated before making the final selection. This final step involves a steady-state validation, where the nonlinar model is used to check if the controlled variables give good economic performance in the entire operating region; and a dynamic validation, to assess the controllability of the control structures. Details of the different steps in this methodology are given in the remainder of this section.

3.2.1 Preliminary controlled variable selection

We start by assuming that the following measurements are available for control purposes:

- Concentrations at the outlet of the tubular reactors and the liquid-liquid separator, and in the feed streams.
- Volume flowrates of the feed streams, tubular reactors, and liquid-liquid separator.
- Temperatures of the tubular reactors.

The measurement set has a total of 39 variables. The objective now is to investigate which combination of measurements should be selected as controlled variables, such that the resulting control structure minimizes the loss of optimality due to the impact of disturbances and sensor noise between setpoint updates in a hierarchical control architecture, where the optimization and control layers operate on different time scales.

Local method

Using the mixed integer quadratic programming (MIQP) formulation presented in Section 2.4.5 in combination with the exact local method presented in Section 2.4.3, we solve the measurement subset selection problem for locally optimal control structures. To select m sets of controlled variables which give the least losses in increasing order with s number of measurements, we formulate the following optimization problem:

$$\min_{\mathbf{h}_\delta,\sigma} \quad \mathbf{h}_\delta^{\mathrm{T}}\mathbf{F}_\delta\mathbf{h}_\delta \tag{3.23a}$$

$$\text{s.t.} \quad \mathbf{G}_{y\delta}^{\mathrm{T}}\mathbf{h}_\delta = \mathbf{j}_\delta, \tag{3.23b}$$

$$\mathbf{P}\sigma = s, \tag{3.23c}$$

$$\sigma_j = 0 \Rightarrow \begin{bmatrix} h_{1j} \\ h_{2j} \\ \vdots \\ h_{n_u j} \end{bmatrix} = \underline{0}_{n_u\times 1}, \quad \text{for} \quad j = 1,\dots,n_y, \tag{3.23d}$$

$$(\sigma^{l-1})^{\mathrm{T}}\sigma^l \leq s-1, \quad \text{for} \quad l = 2,\dots,m, \tag{3.23e}$$

$$\sigma \in \{0,1\}. \tag{3.23f}$$

The difference between this formulation and the one presented in Section 2.4.5 is the use of indicator constraints (3.23d) to select subsets of measurements, instead of the so-called big-M constraints; and the addition of the constraint (3.23d) to select the top m solutions to the problem.

The idea behind the indicator constraints is that whenever the binary variable σ associated to a particular measurement j is zero, the column of the combination matrix $\mathbf{H}$ associated to that measurement will also be zero. Indicator constraints have the advantage of making the formulation problem-independent, making the optimization more robust. This avoids the problem of selecting appropriate values for the big-M constraints on a trial and error basis, as discussed in Section 2.4.5.

The constraint (3.23d) guarantees that the lth solution is different from the $l-1$ previous solutions. Therefore, by solving the optimization problem with increasing values of l, we obtain sequentially the m solutions which give the least losses in increasing order. The final control structure can be selected from this set of m candidates through further validation using the nonlinear model.

The problem was implemented in MATLAB and solved using the MIQP solver *cplexmiqp* from IBM ILOG CPLEX Optimization Studio (CPLEX, 2009). CPLEX and most optimization packages, require to formulate the MIQP in the following standard form:

$$\min_{\mathbf{x}} \quad \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{H}\mathbf{x} + \mathbf{x}^{\mathrm{T}}\mathbf{f} \tag{3.24a}$$

$$\text{s.t.} \quad \mathbf{A}_{\mathrm{eq}}\mathbf{x} = \mathbf{b}_{\mathrm{eq}}, \tag{3.24b}$$

$$\mathbf{A}_{\mathrm{ineq}}\mathbf{x} \leq \mathbf{b}_{\mathrm{ineq}}, \tag{3.24c}$$

$$\mathbf{l}_b \leq \mathbf{x} \leq \mathbf{u}_b. \tag{3.24d}$$

The translation between both formulations is as follows. The solution vector and its upper and lower bound correspond to

$$\mathbf{x} = \left[\begin{array}{c}\mathbf{h}_\delta \\ \hline \sigma\end{array}\right] \tag{3.25}$$

$$\mathbf{l}_b = \left[\begin{array}{c}-\infty \\ \hline \mathbf{0}\end{array}\right] \tag{3.26}$$

$$\mathbf{u}_b = \left[\begin{array}{c}\infty \\ \hline \mathbf{1}\end{array}\right] \tag{3.27}$$

where $\mathbf{h}_\delta \in \mathbb{R}^{n_u n_y \times 1}$ and $\sigma \in \mathbb{R}^{n_y \times 1}$. The equality and inequality constraints corresponds to

$$\mathbf{A}_{\mathrm{eq}} = \left[\begin{array}{c|c}\mathbf{G}_{y\delta}^{\mathrm{T}} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{P}\end{array}\right] \tag{3.28}$$

$$\mathbf{b}_{\mathrm{eq}} = \left[\begin{array}{c}\mathbf{j}_\delta \\ \hline s\end{array}\right] \tag{3.29}$$

$$\mathbf{A}_{\mathrm{ineq}} = \left[\mathbf{0} \mid (\sigma^{l-1})^{\mathrm{T}}\right] \tag{3.30}$$

$$\mathbf{b}_{\mathrm{ineq}} = s - 1 \tag{3.31}$$

where $\mathbf{G}_{y\delta}^{\mathrm{T}} \in \mathbb{R}^{n_u n_u \times n_y n_u}$, $\mathbf{P} \in \mathbb{R}^{1 \times n_y}$, $\mathbf{j}_\delta \in \mathbb{R}^{n_u n_u \times 1}$, $s \in \mathbb{N}$, and $(\sigma^{l-1})^{\mathrm{T}} \in \mathbb{R}^{1 \times n_y}$. The construction of these matrices is explained in Section 2.4.5. Finally, the matrices in the objective function are

$$\mathbf{H} = 2\left[\begin{array}{c|c}\mathbf{F}_\delta & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0}_{n_y \times n_y}\end{array}\right] \tag{3.32}$$

$$\mathbf{f} = \underline{0}_{1 \times (n_u n_y + n_y)} \tag{3.33}$$

where $\mathbf{F}_\delta = \tilde{\mathbf{F}}_\delta \tilde{\mathbf{F}}_\delta^{\mathrm{T}}$, and $\tilde{\mathbf{F}}_\delta$ is constructed as in Section 2.4.5.

The problem formulation above assumes that the following gain and Hessian matrices evaluated at the nominally optimal point are available beforehand:

$$\mathbf{G}_y = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{u}}\right)_{\text{nom}} \tag{3.34}$$

$$\mathbf{G}_{yd} = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{d}}\right)_{\text{nom}} \tag{3.35}$$

$$\mathbf{J}_{uu} = \left(\frac{\partial^2 J}{\partial \mathbf{u}^2}\right)_{\text{nom}} \tag{3.36}$$

$$\mathbf{J}_{ud} = \left(\frac{\partial^2 J}{\partial \mathbf{u} \partial \mathbf{d}}\right)_{\text{nom}} \tag{3.37}$$

These matrices were computed using automatic differentiation in Casadi 3.0.0 (Andersson et al., 2012).

The diagonal scale matrices for the disturbances and sensor noise realizations are constructed based on the standard deviation values reported in Section 3.1.4.

$$\mathbf{W}_d = \begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_{n_d} \end{bmatrix} \tag{3.38}$$

$$\mathbf{W}_n = \begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_{n_y} \end{bmatrix} \tag{3.39}$$

In this work, the local method described so far was used to find the best $m = 4$ control structures for each number of measurements, ranging from $s = 4$, which is the number of inputs (to have a square control problem) and $s = 39$, which is the total number of available measurements.

Global method

For the case of the global method, we combine the mixed integer quadratic programming (MIQP) formulation presented in Section 2.4.5 with the gSOC method presented in Section 2.4.4. The problem formulation is the same as in the local case, with the only difference being that the matrix $\tilde{\mathbf{F}}^{\mathrm{T}}$ in the minimum loss method is replaced by the matrix $\tilde{\mathbf{Y}}$. This augmented matrix is constructed based on optimal data from the entire operating region using Monte Carlo simulations, as explained in Section 2.4.4. In this work, 1000 Monte Carlo simulations were used to generate normally distributed scenarios of disturbances with standard deviations given in Section 3.1.4.

3.2.2 Steady-state validation

The candidate controlled variables found by the pre-screening methods need to be further evaluated using the nonlinear model. Each candidate control structure is tested in closed-loop simulations under 100 scenarios of normally distributed disturbances and sensor noise realizations. These uncertainty scenarios are generated using Monte Carlo simulations based on the standard deviations given in Section 3.1.4. For each uncertainty scenario, the loss of optimality from using a given control structure is computed as the difference between the cost function resulting from using that control structure and the cost function resulting from truly optimal operation, which is obtained by re-optimizing the system for the new operating conditions. When the closed-loop simulations are performed for all of the uncertainty scenarios, the nonlinear average loss for each candidate control structure

Table 3.3: Nominal operating conditions and disturbance scenarios considered in the dynamic closed-loop simulations.

Disturbance	**Unit**	**Nominal conditions**	**Scenario** 1	**Scenario** 2	**Scenario** 3
M_1	mol/L	2	2.01451	2.01454	1.99796
M_7	mol/L	4	3.99748	3.98786	3.99034
T_1	°C	100	113.8	108.2	96.2
T_2	°C	100	93.3	102.4	106.9
T_3	°C	50	65.2	55.2	41.4
k_1	mol/(mL min)	34206	37342.5	33855.4	32859.5
k_2	mol/(mL min)	10000	8870.6	9937.9	10444.2
k_3	mol/(mL min)	24	25.0	25.8	22.6
k_4	mol/(mL min)	43599	44293.9	46670.6	41268.9
E_{A1}	J/mol	1000	934.6	1070.9	959.5
E_{A2}	J/mol	100	97.8	103.4	85.3
E_{A3}	J/mol	1819	1850.2	1709.2	1949.8
E_{A4}	J/mol	26207	30896.0	27146.8	26633.1
$\log(D_9)$	-	-2	-1.73	-1.64	-1.85

can then be computed. For any given number of measurements, the control structure with the lowest nonlinear average loss is selected. Furthermore, to select the optimal number of measurements to be included in the control structure, we search for a good trade-off between the reduction in the average loss (favored by more measurements) and the reduction in structure complexity and investment cost (favored by less measurements). A detailed analysis of this Pareto front curve is given in Section 2.4.5.

3.2.3 Dynamic validation

The analysis so far was done from a steady-state perspective, assuming that the controlled variables can be well controlled within one transient phase. To demonstrate that the control structure can be realized in practice, closed-loop dynamic simulations are performed according to the following set-up. Initially, the system is operated at the nominal conditions. Subsequently, for every 1000 min, the operating conditions in the plant are changed according to the disturbance scenarios given in Table 3.3. These scenarios were generated by Monte Carlo simulations of normally distributed disturbances with standard deviations given in Section 3.1.4.

For the closed-loop simulations, decentralized PI controllers were used to track the self-optimizing controlled variables. For comparison, a quadratic dynamic matrix controller (QDMC), which is a form of model predictive control, was tested under the same disturbance scenarios by tracking the steady-state optimal operating conditions computed by re-optimizing the system for each uncertainty scenario. In practice, the optimal setpoints are assumed to be given by an upper real-time optimization layer.

3.3 Controller design methodology

This section discusses the design of the decentralized PI controllers and the quadratic dynamic matrix controller used in the dynamic closed-loop simulations.

3.3.1 Decentralized PI control

As discussed in Section 2.5.1, the design of decentralized PI controllers involves two steps: the choice of pairings and the controller tuning. The pairings between controlled and manipulated variables was done using the steady-state relative gain array (RGA) matrix by selecting positive RGA-elements close to 1. This avoids unstable loops and minimize the interactions with other control loops.

Regarding the controller tuning, the SIMC tuning method presented in Section 2.5.1 was applied. This method requires a first-order plus delay model of the system, on the form

$$g_1(s) = \frac{k}{(\tau_1 s + 1)} e^{-\theta s}, \tag{3.40}$$

where k is the process gain, τ_1 is the dominant time constant, and θ is the effective time delay. These process parameters were obtained from open-loop responses of the nonlinear system to a step change of 1% on the inputs. The resulting nonlinear response was then fitted to a first-order plus delay model. The PI controller settings were then calculated according to

$$K_c = \frac{1}{k} \frac{\tau_1}{(\tau_c + \theta)}, \tag{3.41}$$

$$\tau_I = \min\{\tau_1, 4(\theta_c + \theta)\}, \tag{3.42}$$

where the closed-loop time constant τ_c is the only tuning parameter (it is a degree of freedom for controller design, as it does not come from model information). The SIMC recommendation of selecting the closed-loop time constant equal to the time delay was not follow in this case because for most of the simulations the time delay was close to zero. This would lead to a large controller gain that could lead to unstable closed-loop performance. Instead, a value of $\tau_c = 70$ was found on a trial and error based to give a good trade-off between fast response, moderate input variation and good robustness margins.

3.3.2 Quadratic dynamic matrix control (QDMC)

Quadratic dynamic matrix control (QDMC) is a more sophisticated version of dynamic matrix control (DMC), which was discussed in Section 2.5.2. Linear step response models that relate the inputs to the outputs of the plant were generated and the linear predictive model of the E-factor with respect to the plant inputs was constructed using the methodology described in Nikolakopoulou et al. (2019). The weights used were

$$W_u = \text{diag}(10^4, \dots, 10^4) \in \mathbb{R}^{n_u c \times n_u c}, \tag{3.43}$$

where n_u is the number of inputs, and

$$W_y = \text{diag}(1, \dots, 1) \in \mathbb{R}^{p \times n_y}, \tag{3.44}$$

where n_y is the number of outputs. In this problem, $n_u = 6$, $n_y = 1$, $c = 30$ mins, and $p = 300$ mins. The constraints on the manipulated variables were $u_{\text{min}} = 0.02$ mL/min and $u_{\text{max}} = 5$ mL/min. Constraints on the rate of change of the control moves were also implemented to avoid aggressive changes in the inputs, by restricting the moves to a rate of change of no more than 10% of the input flowrates in the nominal plant operation.

Chapter 4

Results and conclusions

4.1 Optimal operation for nominal conditions

The optimal operating point of the plant for the nominal conditions given in Table 3.2 was found by solving the nonlinear programming problem (NLP) formulated in Section 3.1.3. The nominally optimal value of some important process variables is shown in Table 4.1. These variables include the environmental factor (E-factor), which was the objective function minimized in this work; the production rate of atropine and the reaction yield, which are key performance indicators of the process; and the volume flowrate of the feed streams, which are the process inputs (manipulated variables).

Regarding the process operational objectives, we seek to maximize the mass flow rate of atropine obtained at the outlet of the liquid-liquid separator, which is the desired active pharmaceutical ingredient; as well as the efficiency of the overall atropine reaction.

Table 4.1: Some key process variables at the nominally optimal operating point.

Process variable	**Symbol**	**Unit**	**Optimal value**
Environmental factor	E-factor	-	13.2997
Production rate	Production rate	mg/min	96.4325
Yield	Yield	-	0.4085
Volume flowrate stream 1	q_1	mL/min	0.4078
Volume flowrate stream 2	q_2	mL/min	0.1089
Volume flowrate stream 3	q_3	mL/min	0.3888
Volume flowrate stream 4	q_4	mL/min	0.2126
Volume flowrate stream 5	q_5	mL/min	0.2000
Volume flowrate stream 6	q_6	mL/min	0.5000

These objectives correspond to the maximization of production rate and yield, respectively. Therefore, the problem of achieving optimal operation presented in Section 3.1.3 could have been formulated using any of these variables as the objective function. In this work, we chose the environmental factor (E-factor) [kg waste/kg product] as the objective function to take into account the waste generation in the problem formulation. The E-factor gives the optimal trade-off between achieving maximum production rate and minimum waste. However, these metrics are closely related and by minimizing the E-factor, the production rate of atropine and the yield are also indirectly optimized.

Regarding the optimal values of the manipulated variables presented above, these are only optimal for the nominal operating conditions. Therefore, an open-loop operation policy based on keeping these inputs constant will result in suboptimal operation in the face of uncertainty and process disturbances. This motivates the search for controlled variables that, when kept at constant setpoints, drive the process towards its optimum despite disturbances. Results concerning selection and validation of such self-optimizing control structures for the atropine process are presented in the remainder of this chapter.

4.2 Preliminary controlled variable selection

This section presents the control structures derived from the local and global pre-screening methods formulated in Section 3.2.1 and Section 3.2.1, respectively. These methods provide promising sets of controlled variables that can be further validated using the nonlinear model and dynamic simulations to make the final selection.

The use of these control structures to generate actuator signals (process inputs) in the presence of uncertainty (disturbances and sensor noise) results in a loss of optimality compared to the truly optimal inputs. The aim of this work is to find control structures that result in acceptable losses for all disturbances and sensor noise realizations, achieving therefore near-optimal operation in the presence of uncertainty without the need to re-optimize the system every time a disturbance occurs.

The local method formulated in Section 3.2.1 gives the m sets of s measurements that provide the least average losses in increasing order. In this work, the top $m = 4$ sets were selected for each number of measurements, ranging from $s = 4$ (equal to the number of inputs, to form a square control problem)[1] to $s = 39$ (which is the total number of available process measurements). The average loss computed by the local method as a function of the number of measurements included in the control structure follows a Pareto frontier, which is shown in Fig. 4.1. Here, the average loss is expressed in the same units as the objective function (E-factor). Therefore, the average loss for a given control structure corresponds to the average amount of extra waste produced in the process (in mg) for every

[1]In decentralized control, $n_c = n_u$, i.e., the number of inputs is equal to the number of controlled variables, making the control problem square. This is because each control loop composed of a decentralized controller connects a single input with a single output. Furthermore, we also require that $n_c \leq n_y$, i.e., the number of measurements has to be equal or greater than the number of controlled variables. Otherwise, the system will be overdetermined and perfect control could not be achieved.

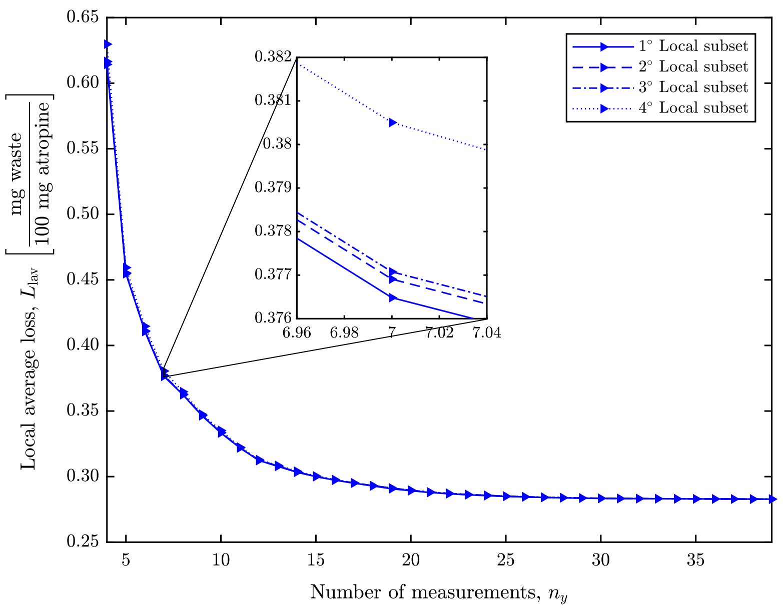

Figure 4.1: Pareto frontier showing the steady-state local average loss versus the number of measurements for the control structures selected by the local method.

100 mg of atropine obtained as a result of using that control structure in the presence of uncertainty instead of online process re-optimization.

As expected from the theory discussed in Section 2.4.5, with more sensors the average loss decreases. This is because more sensors give better disturbance rejection and make the control structure less sensitive to sensor noise. However, the addition of every new sensor increases the capital cost and the complexity of the control system. As a result, there is an optimal trade-off between these two objectives and the goal is to find the number of measurements above which the reduction in loss does not compensate for the addition of an extra sensor. Based on the Pareto frontier in Fig. 4.1, this threshold can be located in the range of 10-15 measurements, where the addition of a new sensor will not decrease the average loss more than 0.025 mg of waste per 100 mg of atropine obtained, approximately.

However, the loss value predicted by the local method is just a local approximation of the true loss. In Section 2.4.3, we discussed how local methods relied on the quadratic approximation of the loss function and the linearization of the process model around the nominal operating point. Therefore, these loss values are only valid in the vicinity of the nominal point and may result in poor approximations of the true loss for disturbances with large standard deviation values (in this work, the uncertainty is assumed to be normally distributed). A further validation using the nonlinear model is necessary to evaluate the accuracy of the predictions made by the local method. The results of this validation are presented in the next section.

Table 4.2: Optimal sets of measurements selected by the local method and corresponding steady-state local average losses.

n_y	**Local subset**	L_{lav}
4	$[c5_{\text{r3}}\ c1_{\text{s,aq}}\ c7_{\text{s,aq}}\ c11_{\text{s,or}}]$	0.6143
	$[c1_{\text{r3}}\ c5_{\text{r3}}\ c7_{\text{s,aq}}\ c11_{\text{s,or}}]$	0.6165
	$[c1_{\text{r2}}\ c5_{\text{r3}}\ c7_{\text{s,aq}}\ c11_{\text{s,or}}]$	0.6165
	$[c3_{\text{r2}}\ c5_{\text{r3}}\ c7_{\text{s,aq}}\ c11_{\text{s,or}}]$	0.6297
5	$[c5_{\text{r2}}\ c1_{\text{s,aq}}\ c7_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}]$	0.4547
	$[c5_{\text{r2}}\ c1_{\text{r3}}\ c7_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}]$	0.4552
	$[c1_{\text{r2}}\ c5_{\text{r2}}\ c7_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}]$	0.4553
	$[c5_{\text{r2}}\ c7_{\text{r2}}\ c1_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}]$	0.4594
6	$[c5_{\text{r2}}\ c11_{\text{r3}}\ c1_{\text{s,aq}}\ c7_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}]$	0.4106
	$[c5_{\text{r2}}\ c1_{\text{r3}}\ c11_{\text{r3}}\ c7_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}]$	0.4110
	$[c1_{\text{r2}}\ c5_{\text{r2}}\ c11_{\text{r3}}\ c7_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}]$	0.4112
	$[c5_{\text{r2}}\ c7_{\text{r2}}\ c11_{\text{r3}}\ c1_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}]$	0.4146
7	$[c5_{\text{r2}}\ c5_{\text{r3}}\ c11_{\text{r3}}\ c1_{\text{s,aq}}\ c7_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}]$	0.3765
	$[c5_{\text{r2}}\ c1_{\text{r3}}\ c5_{\text{r3}}\ c11_{\text{r3}}\ c7_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}]$	0.3769
	$[c1_{\text{r2}}\ c5_{\text{r2}}\ c5_{\text{r3}}\ c11_{\text{r3}}\ c7_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}]$	0.3771
	$[c5_{\text{r2}}\ c7_{\text{r2}}\ c5_{\text{r3}}\ c11_{\text{r3}}\ c1_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}]$	0.3805
8	$[c5_{\text{r2}}\ c5_{\text{r3}}\ c11_{\text{r3}}\ c1_{\text{s,aq}}\ c7_{\text{s,aq}}\ c9_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}]$	0.3624
	$[c5_{\text{r2}}\ c1_{\text{r3}}\ c5_{\text{r3}}\ c11_{\text{r3}}\ c7_{\text{s,aq}}\ c9_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}]$	0.3627
	$[c1_{\text{r2}}\ c5_{\text{r2}}\ c5_{\text{r3}}\ c11_{\text{r3}}\ c7_{\text{s,aq}}\ c9_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}]$	0.3629
	$[c5_{\text{r2}}\ c10_{\text{r2}}\ c5_{\text{r3}}\ c11_{\text{r3}}\ c1_{\text{s,aq}}\ c7_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}]$	0.3647
9	$[c5_{\text{r2}}\ c5_{\text{r3}}\ c11_{\text{r3}}\ c1_{\text{s,aq}}\ c7_{\text{s,aq}}\ c9_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}\ q3]$	0.3465
	$[c5_{\text{r2}}\ c1_{\text{r3}}\ c5_{\text{r3}}\ c11_{\text{r3}}\ c7_{\text{s,aq}}\ c9_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}\ q3]$	0.3466
	$[c1_{\text{r2}}\ c5_{\text{r2}}\ c5_{\text{r3}}\ c11_{\text{r3}}\ c7_{\text{s,aq}}\ c9_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}\ q3]$	0.3468
	$[c5_{\text{r2}}\ c9_{\text{r2}}\ c5_{\text{r3}}\ c11_{\text{r3}}\ c1_{\text{s,aq}}\ c7_{\text{s,aq}}\ c9_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}]$	0.3475
10	$[c5_{\text{r2}}\ c9_{\text{r2}}\ c5_{\text{r3}}\ c11_{\text{r3}}\ c1_{\text{s,aq}}\ c7_{\text{s,aq}}\ c9_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}\ q3]$	0.3333
	$[c5_{\text{r2}}\ c9_{\text{r2}}\ c1_{\text{r3}}\ c5_{\text{r3}}\ c11_{\text{r3}}\ c7_{\text{s,aq}}\ c9_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}\ q3]$	0.3335
	$[c1_{\text{r2}}\ c5_{\text{r2}}\ c9_{\text{r2}}\ c5_{\text{r3}}\ c11_{\text{r3}}\ c7_{\text{s,aq}}\ c9_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}\ q3]$	0.3337
	$[c5_{\text{r2}}\ c5_{\text{r3}}\ c9_{\text{r3}}\ c11_{\text{r3}}\ c1_{\text{s,aq}}\ c7_{\text{s,aq}}\ c9_{\text{s,aq}}\ c5_{\text{s,or}}\ c11_{\text{s,or}}\ q3]$	0.3352
⋮	⋮	⋮
39	all measurements	0.2828

More important in this preliminary analysis are the optimal set of measurements selected by the local method, which are shown in Table 4.2. Each set corresponds to a point in the Pareto frontier in Fig. 4.1. For the sake of brevity, we only present here sets containing between 4 and 10 measurements. The sets in this table should be read as follows. The volume flowrate of the feed streams, which are the manipulated variables, are denoted by q_{1-4}. The rest of the process variables are given by the following three identification elements:

- *Letter* indicating the type of process variable; namely, c for concentration, q for volume flow rate, and T for temperature.
- *Number* between 1 and 14 indicating the process component. These numbers follow the order established in Table 3.1.
- *Subscript* indicating the unit operation in the system; namely, $(\cdot)_{\mathrm{r}1-3}$ for the three tubular reactors, $(\cdot)_{\mathrm{s,aq}}$ for the aqueous phase of the liquid-liquid separator, and $(\cdot)_{\mathrm{s,or}}$ for the organic phase of the liquid-liquid separator.

As an example, the best set of 4 measurements found by the local method corresponds to

$$[c_{5_{\mathrm{r3}}}\ c_{1_{\mathrm{s,aq}}}\ c_{7_{\mathrm{s,aq}}}\ c_{11_{\mathrm{s,or}}}], \tag{4.1}$$

which is composed of the following four measurements: the concentration of formaldehyde at the outlet of the third tubular reactor ($c_{5_{\mathrm{r3}}}$), the concentration of tropine in the aqueous phase of the liquid-liquid separator ($c_{1_{\mathrm{s,aq}}}$), the concentration of sodium hydroxide in the aqueous phase of the liquid-liquid separator ($c_{7_{\mathrm{s,aq}}}$), and the concentration of tropine ester in the organic phase of the liquid-liquid separator ($c_{11_{\mathrm{s,or}}}$). This set derived from the local method agrees with the intuitive dominant variables in the process that one may expect. Both formaldehyde (CH_2O, component 5) and tropine ester ($C_{16}H_{21}O_2N$, component 11) are the reactants in reactions (3.2c) and (3.2d), that compete to produce atropine (desired product) and apoatropine (undesired byproduct), respectively. Furthermore, tropine ($C_8H_{15}NO$, component 1) and sodium hydroxide ($NaOH$, component 7) are the reactants in reactions (3.2a) and (3.2b), respectively, that trigger the overall reaction set. These variables appear repeatedly in the rest of the selected sets of measurements.

Similar results found by the global method formulated in Section 3.2.1 are shown in Fig. 4.2 and Table 4.3. The global method uses optimal data sampled from the entire operating region, giving average loss predictions closer to the true values. Compared with the local method, the loss predictions are larger (more than the double). Regarding the selection of optimal measurement sets, there are several similarities with the sets proposed by the local method. One example is the optimal set of 4 measurements, where the same process variables were selected by both methods. Among some differences, it is remarkable the early inclusion of the concentration of atropine in the aqueous phase of the liquid-liquid separator ($c_{9_{\mathrm{s,aq}}}$) into the measurement sets, as well as the volume flowrate of the first feed stream (q_1).

Table 4.3: Optimal sets of measurements selected by the global method and corresponding steady-state global average losses.

n_y	**Global subset**	L_gav
4	$[c_{5_\mathrm{r3}}\ c_{1_\mathrm{s,aq}}\ c_{7_\mathrm{s,aq}}\ c_{11_\mathrm{s,or}}]$	1.4427
	$[c_{1_\mathrm{r2}}\ c_{5_\mathrm{r3}}\ c_{7_\mathrm{s,aq}}\ c_{11_\mathrm{s,or}}]$	1.4456
	$[c_{1_\mathrm{r3}}\ c_{5_\mathrm{r3}}\ c_{7_\mathrm{s,aq}}\ c_{11_\mathrm{s,or}}]$	1.4457
	$[c_{3_\mathrm{r2}}\ c_{5_\mathrm{r3}}\ c_{7_\mathrm{s,aq}}\ c_{11_\mathrm{s,or}}]$	1.4600
5	$[c_{5_\mathrm{r3}}\ c_{11_\mathrm{r3}}\ c_{1_\mathrm{s,aq}}\ c_{7_\mathrm{s,aq}}\ q_1]$	1.2990
	$[c_{1_\mathrm{r3}}\ c_{5_\mathrm{r3}}\ c_{11_\mathrm{r3}}\ c_{7_\mathrm{s,aq}}\ q_1]$	1.2992
	$[c_{1_\mathrm{r2}}\ c_{5_\mathrm{r3}}\ c_{11_\mathrm{r3}}\ c_{7_\mathrm{s,aq}}\ q_1]$	1.2994
	$[c_{5_\mathrm{r3}}\ c_{11_\mathrm{r3}}\ c_{1_\mathrm{s,aq}}\ c_{7_\mathrm{s,aq}}\ q_{r1}]$	1.2999
6	$[c_{9_\mathrm{r2}}\ c_{5_\mathrm{r3}}\ c_{1_\mathrm{s,aq}}\ c_{7_\mathrm{s,aq}}\ c_{9_\mathrm{s,aq}}\ c_{11_\mathrm{s,or}}]$	1.1725
	$[c_{9_\mathrm{r2}}\ c_{1_\mathrm{r3}}\ c_{5_\mathrm{r3}}\ c_{7_\mathrm{s,aq}}\ c_{9_\mathrm{s,aq}}\ c_{11_\mathrm{s,or}}]$	1.1733
	$[c_{1_\mathrm{r2}}\ c_{9_\mathrm{r2}}\ c_{5_\mathrm{r3}}\ c_{7_\mathrm{s,aq}}\ c_{9_\mathrm{s,aq}}\ c_{11_\mathrm{s,or}}]$	1.1734
	$[c_{3_\mathrm{r2}}\ c_{9_\mathrm{r2}}\ c_{5_\mathrm{r3}}\ c_{7_\mathrm{s,aq}}\ c_{9_\mathrm{s,aq}}\ c_{11_\mathrm{s,or}}]$	1.1805
7	$[c_{9_\mathrm{r2}}\ c_{5_\mathrm{r3}}\ c_{1_\mathrm{s,aq}}\ c_{7_\mathrm{s,aq}}\ c_{9_\mathrm{s,aq}}\ c_{5_\mathrm{s,or}}\ c_{11_\mathrm{s,or}}]$	1.0690
	$[c_{9_\mathrm{r2}}\ c_{1_\mathrm{r3}}\ c_{5_\mathrm{r3}}\ c_{7_\mathrm{s,aq}}\ c_{9_\mathrm{s,aq}}\ c_{5_\mathrm{s,or}}\ c_{11_\mathrm{s,or}}]$	1.0696
	$[c_{1_\mathrm{r2}}\ c_{9_\mathrm{r2}}\ c_{5_\mathrm{r3}}\ c_{7_\mathrm{s,aq}}\ c_{9_\mathrm{s,aq}}\ c_{5_\mathrm{s,or}}\ c_{11_\mathrm{s,or}}]$	1.0698
	$[c_{7_\mathrm{r2}}\ c_{9_\mathrm{r2}}\ c_{5_\mathrm{r3}}\ c_{1_\mathrm{s,aq}}\ c_{9_\mathrm{s,aq}}\ c_{5_\mathrm{s,or}}\ c_{11_\mathrm{s,or}}]$	1.0757
8	$[c_{9_\mathrm{r2}}\ c_{5_\mathrm{r3}}\ c_{11_\mathrm{r3}}\ c_{1_\mathrm{s,aq}}\ c_{7_\mathrm{s,aq}}\ c_{9_\mathrm{s,aq}}\ c_{5_\mathrm{s,or}}\ q_1]$	1.0090
	$[c_{9_\mathrm{r2}}\ c_{1_\mathrm{r3}}\ c_{5_\mathrm{r3}}\ c_{11_\mathrm{r3}}\ c_{7_\mathrm{s,aq}}\ c_{9_\mathrm{s,aq}}\ c_{5_\mathrm{s,or}}\ q_1]$	1.0092
	$[c_{5_\mathrm{r2}}\ c_{9_\mathrm{r2}}\ c_{5_\mathrm{r3}}\ c_{11_\mathrm{r3}}\ c_{1_\mathrm{s,aq}}\ c_{7_\mathrm{s,aq}}\ c_{9_\mathrm{s,aq}}\ q_1]$	1.0092
	$[c_{1_\mathrm{r2}}\ c_{9_\mathrm{r2}}\ c_{5_\mathrm{r3}}\ c_{11_\mathrm{r3}}\ c_{7_\mathrm{s,aq}}\ c_{9_\mathrm{s,aq}}\ c_{5_\mathrm{s,or}}\ q_1]$	1.0094
9	$[c_{5_\mathrm{r2}}\ c_{9_\mathrm{r2}}\ c_{5_\mathrm{r3}}\ c_{11_\mathrm{r3}}\ c_{1_\mathrm{s,aq}}\ c_{7_\mathrm{s,aq}}\ c_{9_\mathrm{s,aq}}\ c_{5_\mathrm{s,or}}\ q_1]$	0.9734
	$[c_{9_\mathrm{r2}}\ c_{3_\mathrm{r3}}\ c_{5_\mathrm{r3}}\ c_{11_\mathrm{r3}}\ c_{7_\mathrm{s,aq}}\ c_{9_\mathrm{s,aq}}\ c_{3_\mathrm{s,or}}\ c_{5_\mathrm{s,or}}\ q_1]$	0.9735
	$[c_{5_\mathrm{r2}}\ c_{9_\mathrm{r2}}\ c_{1_\mathrm{r3}}\ c_{5_\mathrm{r3}}\ c_{11_\mathrm{r3}}\ c_{7_\mathrm{s,aq}}\ c_{9_\mathrm{s,aq}}\ c_{5_\mathrm{s,or}}\ q_1]$	0.9735
	$[c_{1_\mathrm{r2}}\ c_{5_\mathrm{r2}}\ c_{9_\mathrm{r2}}\ c_{5_\mathrm{r3}}\ c_{11_\mathrm{r3}}\ c_{7_\mathrm{s,aq}}\ c_{9_\mathrm{s,aq}}\ c_{5_\mathrm{s,or}}\ q_1]$	0.9737
10	$[c_{5_\mathrm{r2}}\ c_{9_\mathrm{r2}}\ c_{3_\mathrm{r3}}\ c_{5_\mathrm{r3}}\ c_{11_\mathrm{r3}}\ c_{7_\mathrm{s,aq}}\ c_{9_\mathrm{s,aq}}\ c_{3_\mathrm{s,or}}\ c_{5_\mathrm{s,or}}\ q_1]$	0.9392
	$[c_{5_\mathrm{r2}}\ c_{9_\mathrm{r2}}\ c_{3_\mathrm{r3}}\ c_{5_\mathrm{r3}}\ c_{11_\mathrm{r3}}\ c_{7_\mathrm{s,aq}}\ c_{9_\mathrm{s,aq}}\ c_{3_\mathrm{s,or}}\ c_{5_\mathrm{s,or}}\ q_1]$	0.9405
	$[c_{9_\mathrm{r2}}\ c_{3_\mathrm{r3}}\ c_{5_\mathrm{r3}}\ c_{11_\mathrm{r3}}\ c_{7_\mathrm{s,aq}}\ c_{9_\mathrm{s,aq}}\ c_{12_\mathrm{s,aq}}\ c_{3_\mathrm{s,or}}\ c_{5_\mathrm{s,or}}\ q_1]$	0.9413
	$[c_{9_\mathrm{r2}}\ c_{3_\mathrm{r3}}\ c_{5_\mathrm{r3}}\ c_{11_\mathrm{r3}}\ c_{7_\mathrm{s,aq}}\ c_{9_\mathrm{s,aq}}\ c_{12_\mathrm{s,aq}}\ c_{3_\mathrm{s,or}}\ c_{5_\mathrm{s,or}}\ q_{r1}]$	0.94241
⋮	⋮	⋮
39	all measurements	0.7706

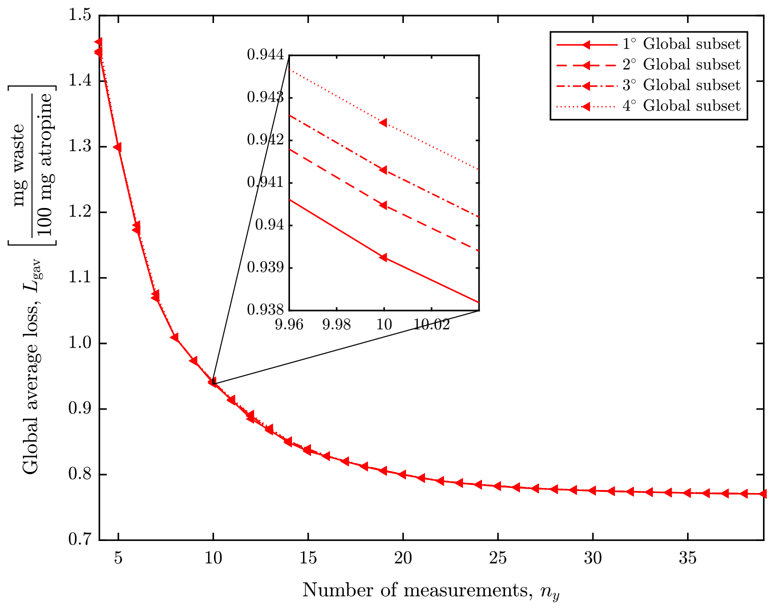

Figure 4.2: Pareto frontier showing the steady-state global average loss versus the number of measurements for the control structures selected by the global method.

4.3 Steady-state validation

The candidate controlled variables found in the previous section are now evaluated using the nonlinear model to check if the average losses predicted by the local and global methods are accurate with respect to the true average loss. For each control structure, the steady-state average loss was computed by performing closed-loop simulations under 100 scenarios of normally distributed disturbances and sensor noise realizations generated by Monte Carlo simulations with standard deviations given in Section 3.1.4. The results of these simulations are illustrated in Fig. 4.3. The shape and the average loss values of the Pareto frontier curves have changed with respect to the ones showed in the previous section, indicating that the true average loss is larger than the predicted values. The nonlinear average losses corresponding to the control structures selected by the local and global methods are depicted in blue and red lines, respectively. Furthermore, the nonlinear average loss resulting from open-loop nominal operation (black line) has been included for the sake of comparison. The optimal measurement sets from local and global methods are given in Table 4.4.

The control structures resulting from the global method outperform the ones from the local method, giving lower steady-state average losses. Furthermore, tracking controlled variables derived from the global method reduces in about 15 mg the waste generated per 100 mg of atropine obtained, compared with open-loop nominal operation. The inclusion of 6 measurements into the control structure gives a good trade-off between the reduction in the average loss and the number of measurements. Above this point, the controlled variables found by the global method do not significantly reduce the steady-state average loss when new sensors are added to the control structure.

A more detailed look into the control structures containing 6 measurements is given in Fig. 4.4. Here, the steady-state losses for the 100 normally distributed uncertainty scenarios are shown. The performance of global self-optimizing control structures is superior to that of local self-optimizing control structures and nominal open-loop operation.

A sensitivity analysis of the steady-state loss resulting from control structures of 6 measurements for the different disturbances considered in this work is presented in Fig. 4.5. The loss of optimality is shown as a function of the standard deviation of the different disturbances. At the nominal operating point ($\sigma = 0$), the locally optimal control structures give zero loss. This is not the case for globally optimal control structures, where the objective is to minimize the average loss over the entire operating envelope rather than obtaining locally zero loss. The disturbances with more influence over the steady-state loss are the value of the distribution coefficient of atropine in the liquid-liquid separator $\log(D_{\mathrm{atrop}})$ and the activation energy of reaction (3.2d) E_{A4}. Both disturbances correspond to parametric model uncertainty.

Based on the results from the steady-state validation, the global self-optimizing (gsoc) control structure of 6 measurements was finally selected for further dynamic validation. Writing the controlled variables as linear combination of the available measurements on the form $\mathbf{c} = \mathbf{H}\mathbf{y}$, we get

$$\mathbf{c}_{\mathrm{gsoc}} = \mathbf{H}_{\mathrm{gsoc}} \begin{bmatrix} c_{9_{\mathrm{r2}}} \\ c_{5_{\mathrm{r3}}} \\ c_{1_{\mathrm{s,aq}}} \\ c_{7_{\mathrm{s,aq}}} \\ c_{9_{\mathrm{s,aq}}} \\ c_{11_{\mathrm{s,or}}} \end{bmatrix}, \tag{4.2}$$

where the combination matrix $\mathbf{H}_{\mathrm{gsoc}}$ is

$$\mathbf{H}_{\mathrm{gsoc}} = \begin{bmatrix} -0.011 & -0.003 & 0.319 & -0.091 & 0.008 & 0.022 & 2.594 \cdot 10^{-15} & 1.463 \cdot 10^{-14} \\ -0.003 & -8.737 \cdot 10^{-4} & -1.096 & -0.034 & 0.002 & 0.007 & 5.151 \cdot 10^{-16} & 3.560 \cdot 10^{-15} \\ -0.015 & 0.038 & -0.007 & 0.005 & 0.011 & -2.452 \cdot 10^{-4} & 8.243 \cdot 10^{-15} & 4.038 \cdot 10^{-14} \\ -0.007 & -0.002 & -0.030 & 0.225 & 0.008 & 0.010 & 2.353 \cdot 10^{-15} & 1.465 \cdot 10^{-14} \end{bmatrix} \tag{4.3}$$

and the setpoints $\mathbf{c}_{\mathrm{s,gsoc}}$ are

$$\mathbf{c}_{\mathrm{s,gsoc}} = \begin{bmatrix} -1.085 \\ -0.202 \\ -2.738 \\ -1.078 \end{bmatrix}. \tag{4.4}$$

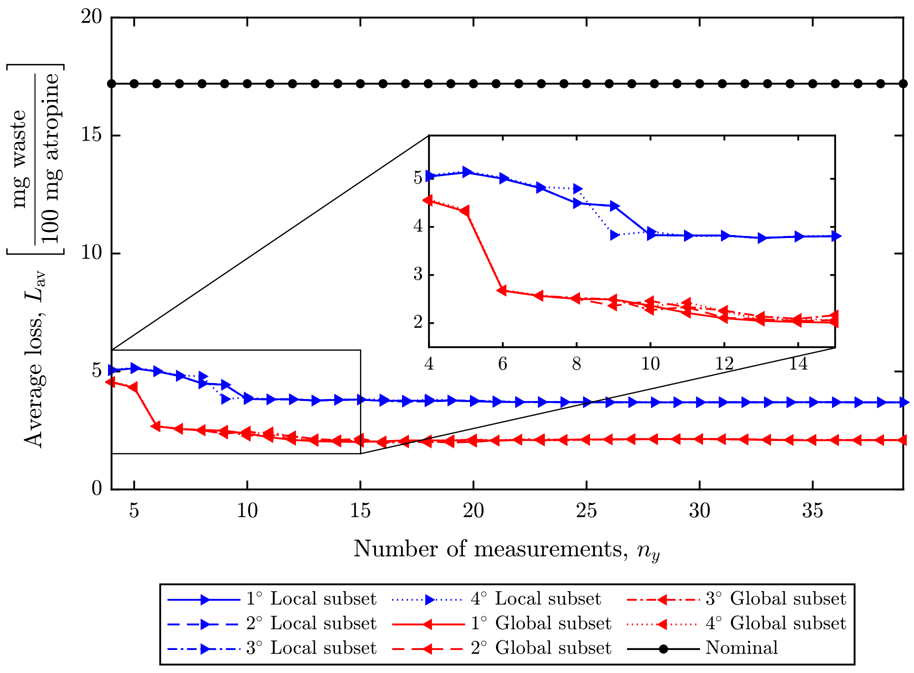

Figure 4.3: Steady-state nonlinear average loss versus the number of measurements for control structures selected by global (red) and local (blue) methods, and for open-loop nominal operation (black). The average losses were computed from 100 normally distributed scenarios of disturbances and sensor noise realizations generated by Monte Carlo simulations.

Table 4.4: Optimal measurement sets from local and global methods validated using the nonlinear model. The steady-state average losses were computed from 100 normally distributed scenarios of disturbances and sensor noise realizations generated by Monte Carlo simulations.

n_y	**Best local subset**	L_{av}	**Best global subset**	L_{av}
4	$[c_{5_{\mathrm{r3}}}\ c_{1_{\mathrm{s,aq}}}\ c_{7_{\mathrm{s,aq}}}\ c_{11_{\mathrm{s,or}}}]$	5.0471	$[c_{5_{\mathrm{r3}}}\ c_{1_{\mathrm{s,aq}}}\ c_{7_{\mathrm{s,aq}}}\ c_{11_{\mathrm{s,or}}}]$	4.5428
5	$[c_{5_{\mathrm{r2}}}\ c_{1_{\mathrm{s,aq}}}\ c_{7_{\mathrm{s,aq}}}\ c_{5_{\mathrm{s,or}}}\ c_{11_{\mathrm{s,or}}}]$	5.1331	$[c_{5_{\mathrm{r3}}}\ c_{11_{\mathrm{r3}}}\ c_{1_{\mathrm{s,aq}}}\ c_{7_{\mathrm{s,aq}}}\ q_1]$	4.3177
6	$[c_{5_{\mathrm{r2}}}\ c_{11_{\mathrm{r2}}}\ c_{1_{\mathrm{s,aq}}}\ c_{7_{\mathrm{s,aq}}}\ c_{5_{\mathrm{s,or}}}\ c_{11_{\mathrm{s,or}}}]$	4.9944	$[c_{9_{\mathrm{r2}}}\ c_{5_{\mathrm{r3}}}\ c_{1_{\mathrm{s,aq}}}\ c_{7_{\mathrm{s,aq}}}\ c_{9_{\mathrm{s,aq}}}\ c_{11_{\mathrm{s,or}}}]$	2.6753
7	$[c_{5_{\mathrm{r2}}}\ c_{11_{\mathrm{r2}}}\ c_{5_{\mathrm{r3}}}\ c_{1_{\mathrm{s,aq}}}\ c_{7_{\mathrm{s,aq}}}\ c_{5_{\mathrm{s,or}}}\ c_{11_{\mathrm{s,or}}}]$	4.8052	$[c_{9_{\mathrm{r2}}}\ c_{5_{\mathrm{r3}}}\ c_{1_{\mathrm{s,aq}}}\ c_{7_{\mathrm{s,aq}}}\ c_{9_{\mathrm{s,aq}}}\ c_{5_{\mathrm{s,or}}}\ c_{11_{\mathrm{s,or}}}]$	2.5654
8	$[c_{5_{\mathrm{r2}}}\ c_{5_{\mathrm{r3}}}\ c_{11_{\mathrm{r3}}}\ c_{1_{\mathrm{s,aq}}}\ c_{7_{\mathrm{s,aq}}}\ c_{9_{\mathrm{s,aq}}}\ c_{5_{\mathrm{s,or}}}\ c_{11_{\mathrm{s,or}}}]$	4.4934	$[c_{9_{\mathrm{r2}}}\ c_{5_{\mathrm{r3}}}\ c_{11_{\mathrm{r3}}}\ c_{1_{\mathrm{s,aq}}}\ c_{7_{\mathrm{s,aq}}}\ c_{9_{\mathrm{s,aq}}}\ c_{5_{\mathrm{s,or}}}\ q_1]$	2.5033
9	$[c_{5_{\mathrm{r2}}}\ c_{9_{\mathrm{r2}}}\ c_{5_{\mathrm{r3}}}\ c_{11_{\mathrm{r3}}}\ c_{1_{\mathrm{s,aq}}}\ c_{7_{\mathrm{s,aq}}}\ c_{9_{\mathrm{s,aq}}}\ c_{5_{\mathrm{s,or}}}\ c_{11_{\mathrm{s,or}}}]$	3.8306	$[c_{9_{\mathrm{r2}}}\ c_{3_{\mathrm{r3}}}\ c_{5_{\mathrm{r3}}}\ c_{11_{\mathrm{r3}}}\ c_{7_{\mathrm{s,aq}}}\ c_{9_{\mathrm{s,aq}}}\ c_{3_{\mathrm{s,or}}}\ c_{5_{\mathrm{s,or}}}\ q_1]$	2.3614
10	$[c_{5_{\mathrm{r2}}}\ c_{9_{\mathrm{r2}}}\ c_{5_{\mathrm{r3}}}\ c_{11_{\mathrm{r3}}}\ c_{1_{\mathrm{s,aq}}}\ c_{7_{\mathrm{s,aq}}}\ c_{9_{\mathrm{s,aq}}}\ c_{5_{\mathrm{s,or}}}\ c_{11_{\mathrm{s,or}}}\ q_3]$	3.8292	$[c_{9_{\mathrm{r2}}}\ c_{3_{\mathrm{r3}}}\ c_{5_{\mathrm{r3}}}\ c_{11_{\mathrm{r3}}}\ c_{7_{\mathrm{s,aq}}}\ c_{9_{\mathrm{s,aq}}}\ c_{12_{\mathrm{s,aq}}}\ c_{3_{\mathrm{s,or}}}\ c_{5_{\mathrm{s,or}}}\ q_1]$	2.2716
⋮	⋮	⋮	⋮	⋮
39	all measurements	3.6909	all measurements	2.0855

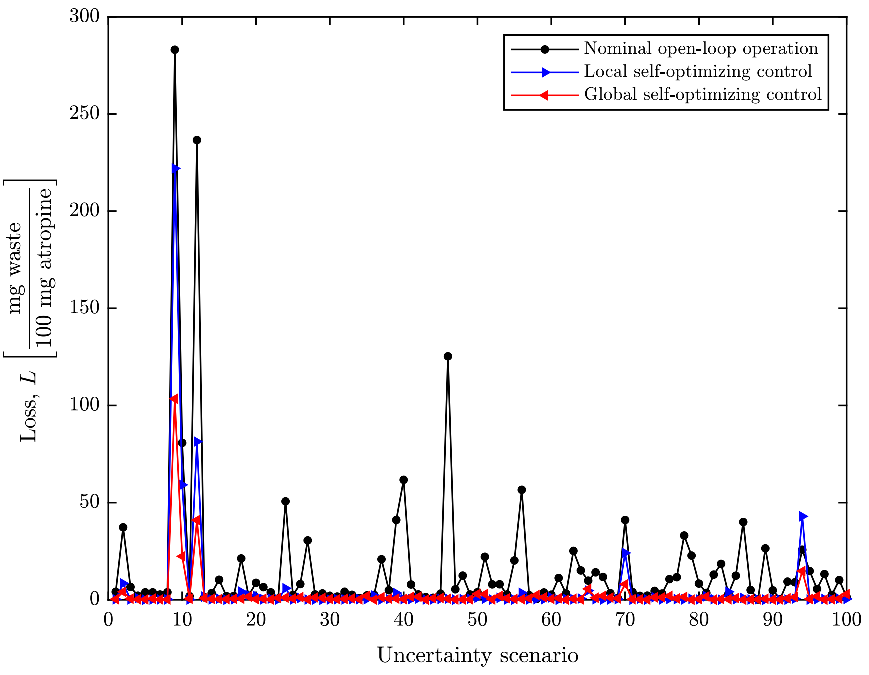

Figure 4.4: Steady-state losses for 100 scenarios of normally distributed disturbances and sensor noise realizations resulting from open-loop nominal operation (black), and constant setpoint control of the optimal set of 6 measurements found by the global (red) and local (blue) methods.

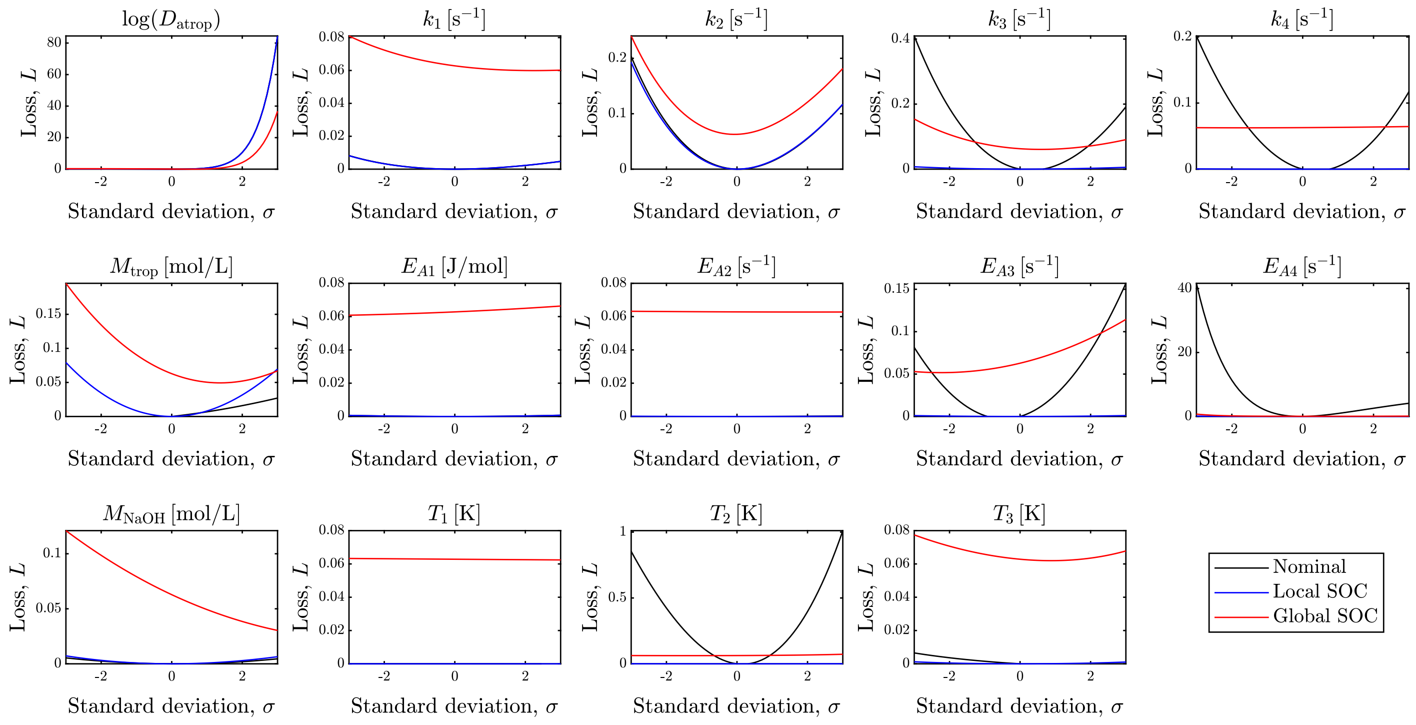

Figure 4.5: Sensitivity analysis of the steady-state loss for the different disturbances considered in this work. The losses shown correspond to open-loop nominal operation (black), and constant setpoint control of the optimal set of 6 measurements found by the global (red) and local (blue) methods.

4.4 Dynamic validation

The final step of the methodology for controlled variable selection is to test the dynamic behaviour of the control structure to check if it can be realized in practice. This is done by performing closed-loop step responses to selected disturbances as described in Section 3.2.3.

Self-optimizing control is realized by controlling the set of controlled variables selected from the global method to some optimally robust constant setpoint (also given by the global method) when disturbances occur. As the process is non highly interactive[2], the simplest approach is to use decentralized PI controllers for tracking the controlled variables to their given setpoints.

First, a dynamic closed-loop simulation was performed assuming no sensor noise. The trajectories of the different disturbances during the simulation are given in Section 3.2.3. The resulting profiles of the controlled variables are shown in Fig. 4.7. The controlled variables are given in deviations with respect to their setpoints. Therefore, tracking the controlled variables $\Delta\mathbf{c}$ (in deviation variables) to zero is equivalent to tracking the controlled variables $\mathbf{c}$ to their given setpoints $\mathbf{c}_\mathrm{s}$. By using feedback control to track these carefully selected controlled variables, the process is automatically driven towards its optimal point despite disturbances. This can be seen from the profile of the economic cost function (here, the E-factor) during the dynamic closed-loop simulation, shown in Fig. 4.6. The optimal steady-state value of the cost function is also depicted with a black dashed line for every disturbance scenario.

To compare the dynamic response of the decentralized PI controllers with a multivariable controller, a model predictive controller (MPC) was implemented. In particular, a quadratic dynamic matrix controller (QDMC) was designed to track the steady-state optimal value of the cost function (E-factor) for all uncertainty scenarios. The optimal setpoints were assumed to be readily available by an upper real-time optimization layer.

Both control strategies result in satisfactory dynamic performance. Some differences among them include the following. The response from the QDMC controller exhibits overshoots when disturbances impact the process, while the response from the decentralized PI controllers tracking self-optimizing controlled variables is rather slow and it takes about 500 min to converge towards the steady-state optimal operating point. One can play around with the PI settings to achieve a faster response. Here, we used the SIMC tuning method, which gives a good trade-off between fast response, moderate input variation and good robustness margins.

The addition of sensor noise to the control system was also studied. The closed-loop simulation results for this scenario are shown in Fig. 4.8 and Fig. 4.9. We observe that both controllers give good dynamic performance. Despite larger variations in the controlled variable profiles, the effect of sensor noise in the cost function is small.

[2]For interactive multiple-input-multiple-output (MIMO) systems, a multivariable controller such as model predictive control (MPC) should be used. To reduce the interactions in decentralized controllers, one can also implement decouplers.

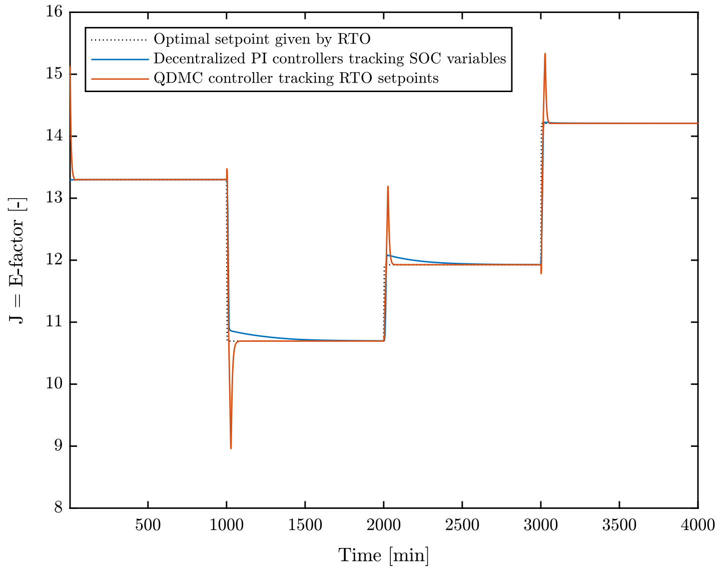

Figure 4.6: Profile of the cost function (E-factor) for nominal operation and three disturbance scenarios.

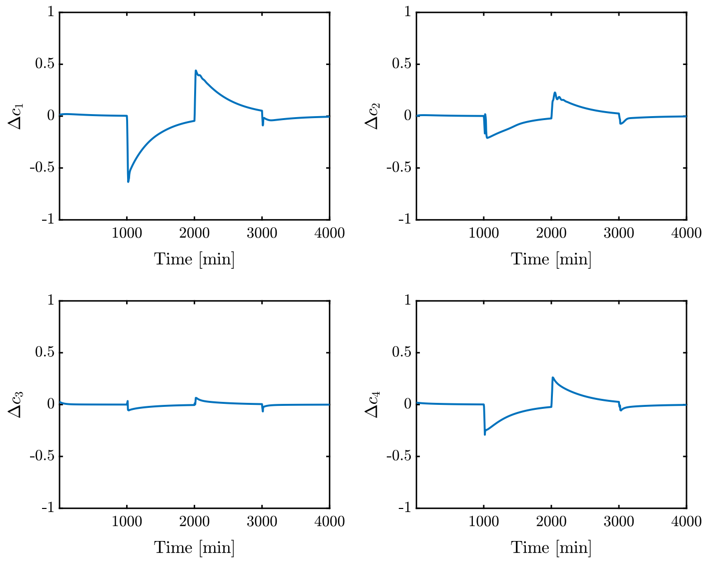

Figure 4.7: Profiles of the controlled variables for nominal operation and three disturbances scenarios.

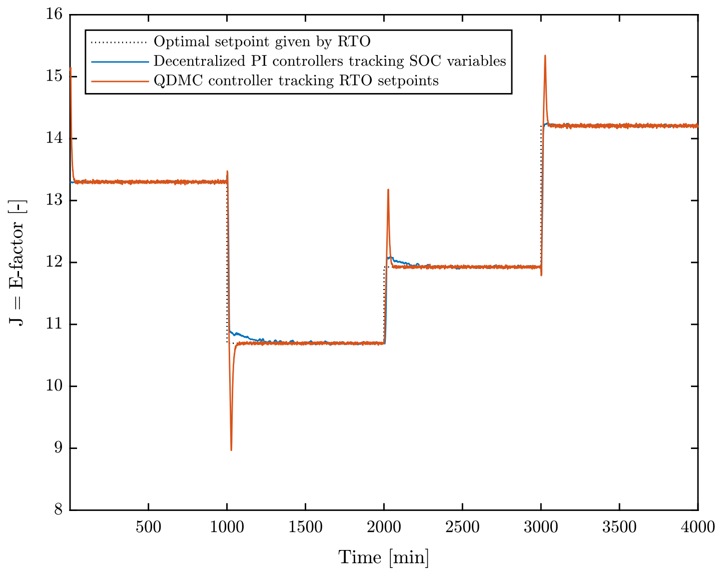

Figure 4.8: Profile of the cost function (E-factor) for nominal operation and three disturbance scenarios. Sensor noise is present in the control system.

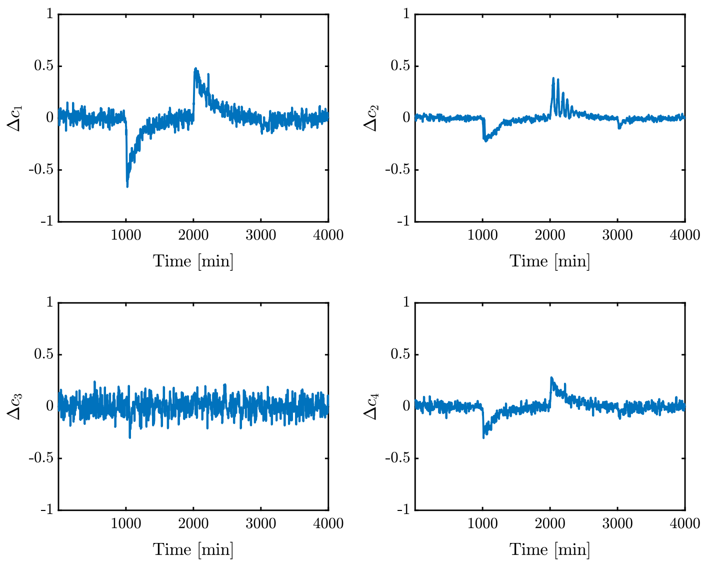

Figure 4.9: Profiles of the controlled variables for nominal operation and three disturbances scenarios. Sensor noise is present in the control system.

4.5 Overall conclusions

This section summarizes the main conclusions drawn from this work.

- A first-principles model of the atropine process is presented in Section 3.1.2. Due to the lack of experimental data, there is parametric model uncertainty in the kinetic parameters of the Arrhenius equation (pre-exponential factors and activation energies) and the separation partition coefficients. This is effectively tackled by designing a control layer selecting controlled variables whose optimal setpoints are near-insensitive to the value of these parameters and other disturbances, making the control system optimally robust to parametric uncertainty. Such optimal invariants are called self-optimizing controlled variables because by tracking them to constant setpoints, the plant is operated near-optimally despite parametric uncertainty and without the need to re-optimize the system when disturbances occur.

- The system was optimized for nominal conditions. At the optimum, all of the 4 degrees of freedom were unconstrained, which correspond with the volume flowrates of the feed streams q_{1-4}. These degrees of freedom were used to find self-optimizing controlled variables.

- To select such self-optimizing controlled variables and their optimal setpoints, a local and a global approach were used. Both methods resulted in minimum losses at steady-state, compared with open-loop nominal operation in the presence of disturbances. The global approach outperforms the local method, giving lower average losses. This was expected because the global method uses optimal data from the entire operating region to approximate the loss quadratically around the nominal operating point, and therefore linearization errors are minimized far away from the chosen linearization point.

- A set of 6 concentration measurements found by the global approach was selected as controlled variables for giving a good trade-off between the number of measurements and the loss reduction. This trade-off balances the benefits of adding more sensors (better disturbance rejection and a control structure less sensitive to sensor noise) and the associated cost (increase investment and control complexity).

- Two controllers were designed: namely, decentralized PI controllers tracking self-optimizing control variables and a quadratic dynamic matrix controller (QDMC) tracking optimal setpoints. Both control strategies resulted in good dynamic performance, also in the presence of sensor noise.

- Therefore, using decentralized PI controllers to track self-optimizing controlled variables is a simple and optimally robust way to operate the plant under parametric uncertainty.

4.6 Evaluation of objectives

The purpose of this section is to evaluate if the objectives set in Section 1.3 were accomplished and to what extent.

1. An extensive literature study on control structure selection methods based on self-optimizing control (including local and global approaches) and controller design methods in multivariable plants (including decentralized PID control and model predictive control) was conducted in Chapter 2.
2. Screening methodologies for selection of self-optimizing controlled variables based on local and global approaches were developed in Section 3.2.1. These methods select optimal subsets of measurements by solving a mixed integer quadratic programming (MIQP) reformulation of the problem. The resulting local and global methods were applied to the atropine process in Section 4.2.
3. Decentralized PI controllers were designed in Section 3.3.1 by using the steady-state relative gain array of the plant to select the pairings and the SIMC tuning method for finding the controller settings. Furthermore, a quadratic dynamic matrix controller (QDMC), which is a form of model predictive controller (MPC), was designed in Section 3.3.2.
4. A steady-state validation of the selected control structures using the nonlinear model was performed in Section 4.3. Furthermore, a dynamic validation using closed-loop step responses to selected disturbances was performed in Section 4.4.

4.7 Directions for further research

Further research directions in this project may include:

- Combine the proposed self-optimizing control layer with an upper real-time optimization (RTO) layer in a hierarchical control architecture. Among different RTO strategies, the authors advocate for the use of modifier adaptation, which can correct for plant-model mismatch. In the combined control hierarchy, the lower self-optimizing control layer can account for parametric model uncertainties on a fast time-scale, while the upper modifier adaptation layer can adjust the setpoints of the controlled variables below to correct for structurally unknown uncertainties and converge to the true optimum on a slower time-scale.
- Test the proposed plantwide control approach in the real plant and validate the simulation results.
- Develop approaches to automate start-up and shutdown procedures in the plant.

Bibliography

Adamo, A., Beingessner, R. L., Behnam, M., Chen, J., Jamison, T. F., Jensen, K. F., Monbaliu, J.-C. M., Myerson, A. S., Revalor, E. M., Snead, D. R., et al., 2016. On-demand continuous-flow production of pharmaceuticals in a compact, reconfigurable system. Science 352 (6281), 61–67.

Alstad, V., Skogestad, S., 2007. Null space method for selecting optimal measurement combinations as controlled variables. Industrial & engineering chemistry research 46 (3), 846–853.

Alstad, V., Skogestad, S., Hori, E. S., 2009. Optimal measurement combinations as controlled variables. Journal of Process Control 19 (1), 138–148.

Andersson, J., Åkesson, J., Diehl, M., 2012. Casadi: A symbolic package for automatic differentiation and optimal control. In: Recent advances in algorithmic differentiation. Springer, pp. 297–307.

Baxendale, I. R., Braatz, R. D., Hodnett, B. K., Jensen, K. F., Johnson, M. D., Sharratt, P., Sherlock, J.-P., Florence, A. J., 2015. Achieving continuous manufacturing: Technologies and approaches for synthesis, workup, and isolation of drug substance. may 20–21, 2014 continuous manufacturing symposium. Journal of pharmaceutical sciences 104 (3), 781–791.

Bédard, A.-C., Longstreet, A. R., Britton, J., Wang, Y., Moriguchi, H., Hicklin, R. W., Green, W. H., Jamison, T. F., 2017. Minimizing E-factor in the continuous-flow synthesis of diazepam and atropine. Bioorganic & Medicinal Chemistry 25 (23), 6233 – 6241, organic Synthesis in Flow for Medicinal Chemistry.

Biegler, L. T., Zavala, V. M., 2009. Large-scale nonlinear programming using ipopt: An integrating framework for enterprise-wide dynamic optimization. Computers & Chemical Engineering 33 (3), 575–582.

Cao, Y., 2004. Constrained self-optimizing control via differentiation. IFAC Proceedings Volumes 37 (1), 63–70.

CPLEX, I. I., 2009. V12. 1: Users manual for cplex. International Business Machines Corporation 46 (53), 157.

Cutler, C. R., Ramaker, B. L., 1979. Dynamic Matrix Control – A computer control algorithm. In: AIChE 86th National Meeting. (Houston, TX).

Dai, C., Snead, D. R., Zhang, P., Jamison, T. F., 2015. Continuous-flow synthesis and purification of atropine with sequential in-line separations of structurally similar impurities. Journal of Flow Chemistry 5 (3), 133–138.

Fiacco, A. V., et al., 1983. Introduction to sensitivity and stability analysis in nonlinear programming. Vol. 90075. Academic press New York.

Findeisen, W., Bailey, F. N., Brdys, M., Malinowski, K., Tatjewski, P., Wozniak, A., 1980. Control and coordination in hierarchical systems. John Wiley & Sons.

Foss, A. S., 1973. Critique of chemical process control theory. AIChE Journal 19 (2), 209–214.

Foss, B., Heirung, T. A. N., 2013. Merging optimization and control. Lecture Notes.

Garcia, C. E., Morshedi, A. M., 1986. Quadratic programming solution of dynamic matrix control (QDMC). Chemical Engineering Communications 46 (1-3), 73–87.

Gurobi, O., 2014. Inc.,gurobi optimizer reference manual, 2015. URL: http://www. gurobi. com.

Halvorsen, I. J., Skogestad, S., Morud, J. C., Alstad, V., 2003. Optimal selection of controlled variables. Industrial & Engineering Chemistry Research 42 (14), 3273–3284.

Hu, W., Umar, L. M., Xiao, G., Kariwala, V., 2012. Local self-optimizing control of constrained processes. Journal of Process Control 22 (2), 488–493.

Ikonen, E., 2017. Model Predictive Control and State Estimation. tech. rep., University of Oulu, Finland.

Jäschke, J., Cao, Y., Kariwala, V., 2017. Self-optimizing control–a survey. Annual Reviews in Control 43, 199–223.

Kariwala, V., 2007. Optimal measurement combination for local self-optimizing control. Industrial & Engineering Chemistry Research 46 (11), 3629–3634.

Kariwala, V., Cao, Y., 2009. Bidirectional branch and bound for controlled variable selection. part ii: Exact local method for self-optimizing control. Computers & chemical engineering 33 (8), 1402–1412.

Kariwala, V., Cao, Y., Janardhanan, S., 2008. Local self-optimizing control with average loss minimization. Industrial & Engineering Chemistry Research 47 (4), 1150–1158.

Krishnamoorthy, D., Skogestad, S., 2019. Online process optimization with active constraint set changes using simple control structures. Industrial & Engineering Chemistry Research.

Lefkowitz, I., 1966. Multilevel approach applied to control system design. Journal of Basic Engineering 88 (2), 392–398.

Manum, H., Skogestad, S., 2012. Self-optimizing control with active set changes. Journal of Process Control 22 (5), 873–883.

Mascia, S., Heider, P. L., Zhang, H., Lakerveld, R., Benyahia, B., Barton, P. I., Braatz, R. D., Cooney, C. L., Evans, J. M., Jamison, T. F., et al., 2013. End-to-end continuous manufacturing of pharmaceuticals: integrated synthesis, purification, and final dosage formation. Angewandte Chemie International Edition 52 (47), 12359–12363.

Mesarovic, M., Macko, D., Takahara, Y., 1970. Theory of hierarchical, multilevel systems. 1970. New York 640.

Morari, M., 1981. Integrated plant control: A solution at hand or a research topic for the next decade.

Morari, M., Arkun, Y., Stephanopoulos, G., 1980. Studies in the synthesis of control structures for chemical processes: Part i: Formulation of the problem. process decomposition and the classification of the control tasks. analysis of the optimizing control structures. AIChE Journal 26 (2), 220–232.

Myerson, A. S., Krumme, M., Nasr, M., Thomas, H., Braatz, R. D., 2015. Control systems engineering in continuous pharmaceutical manufacturing. may 20–21, 2014 continuous manufacturing symposium. Journal of pharmaceutical sciences 104 (3), 832–839.

Narraway, L., Perkins, J., 1994. Selection of process control structure based on economics. Computers & chemical engineering 18, S511–S515.

Nikolakopoulou, A., von Andrian, M., Braatz, R. D., 2019. Supervisory control of a compact modular reconfigurable system for continuous-flow pharmaceutical manufacturing. In: 2019 American Control Conference. Philadelphia, PA.

North, R., Kelly, M., 1987. A review of the uses and adverse effects of topical administration of atropine. Ophthalmic and Physiological Optics 7 (2), 109–114.

Pirnay, H., López-Negrete, R., Biegler, L. T., 2012. Optimal sensitivity based on ipopt. Mathematical Programming Computation 4 (4), 307–331.

Rivera, D. E., Morari, M., Skogestad, S., 1986. Internal model control: Pid controller design. Industrial & engineering chemistry process design and development 25 (1), 252–265.

Skogestad, S., 2000. Plantwide control: The search for the self-optimizing control structure. Journal of process control 10 (5), 487–507.

Skogestad, S., 2003. Simple analytic rules for model reduction and pid controller tuning. Journal of process control 13 (4), 291–309.

Skogestad, S., 2004. Control structure design for complete chemical plants. Computers & Chemical Engineering 28 (1-2), 219–234.

Skogestad, S., Grimholt, C., 2012. The simc method for smooth pid controller tuning. In: PID Control in the Third Millennium. Springer, pp. 147–175.

Skogestad, S., Postlethwaite, I., 2007. Multivariable feedback control: analysis and design. Vol. 2. Wiley New York.

Smith, C. A., Corripio, A. B., 1985. Principles and practice of automatic process control. Vol. 2. Wiley New York.

Verheyleweghen, A., Jäschke, J., 2019. Self-optimizing control of an lng liquefaction plant. Journal of Process Control 74, 63–75.

Ye, L., Cao, Y., Yuan, X., 2015. Global approximation of self-optimizing controlled variables with average loss minimization. Industrial & Engineering Chemistry Research 54 (48), 12040–12053.

Yelchuru, R., Skogestad, S., 2012. Convex formulations for optimal selection of controlled variables and measurements using mixed integer quadratic programming. Journal of Process control 22 (6), 995–1007.

Ziegler, J. G., Nichols, N. B., 1942. Optimum settings for automatic controllers. trans. ASME 64 (11).

Appendix A

MATLAB code

This section contains some important pieces of source code developed in this project. The code is written in MATLAB together with CPLEX (CPLEX, 2009) and Casadi (Andersson et al., 2012).

A.1 MIQP solver for local self-optimizing control

The function **miqp_solver_lsoc** solves the mixed integer quadratic programming (MIQP) problem for selecting the top ns local self-optimizing control structures of nm measurements.

```
function [H,loss,sset]= miqp_solver_lsoc(Gy,Gd,Juu,Jud,Wd,Wn,nm,ns)

    [ny,nu]      = size(Gy);
    nd           = size(Gd,2);
    sigma        = zeros(1,ny);

    if nm == ny
        ns = 1;
    end

    for k = 1:ns

try
    % Initialize CPLEX object
    cplex = Cplex('miqp_solver_lsoc');
    cplex.DisplayFunc = [];

    % Fill in the data for the problem using populatebyrow
    populatebyrow(cplex,Gy,Gd,Juu,Jud,Wd,Wn,nm,sigma);

    % Optimize the problem
    cplex.solve();

    xn = cplex.Solution.x;
    H(:,:,k) = reshape(xn(1:ny*nu),ny,nu)';

    % Compute average loss
    F = Gd-Gy/Juu*Jud;
    Y = [F*Wd Wn];
    M = Juu^(1/2)/(H(:,:,k)*Gy)*(H(:,:,k)*Y);
    loss(k) = 1/2*norm(M,'fro')^2;

    % Select best subset
    sigma(k,:) = xn(nu*ny+1:end)';
    sset(k,:) = find(sigma(k,:)==1);

catch m
    disp(m.message);
    throw (m);
end
    end

    function populatebyrow(cplex,Gy,Gd,Juu,Jud,Wd,Wn,nm,sigma)

        F            = Gd-Gy/Juu*Jud;
        Y            = [F*Wd Wn];
        Juu12        = sqrtm(Juu);
        [ny,nu]      = size(Gy);
        nd           = size(Gd,2);

        % Vectorization
        Jn = []; Yn = []; GnyT = [];
        for i = 1:nu
            GnyT     = blkdiag(GnyT,Gy');
            Jn       = [Jn;Juu12(i,:)'];
            Yn       = blkdiag(Yn,Y);
        end
        Fn           = Yn*Yn';

        % Objective function
        Q            = 2*blkdiag(Fn,zeros(ny,ny));
        c            = zeros(size(Q,1),1);

        % Constraint matrix and bounds
        GnyT_aug     = [GnyT zeros(nu*nu,ny)];
        P_aug        = [zeros(1,ny*nu) ones(1,ny)];
        sigma_aug    = [zeros(size(sigma,1),ny*nu) sigma];
        A            = [GnyT_aug;P_aug;sigma_aug];
        lhs          = [Jn;nm;zeros(size(sigma,1),1)];
        rhs          = [Jn;nm;(nm-1)*ones(size(sigma,1),1)];

        % Bounds
        x_L          = [-Inf*ones(nu*ny,1);zeros(ny,1)];
        x_U          = [Inf*ones(nu*ny,1);ones(ny,1)];

        % Define variable type
        str(1:ny*nu) = 'C';
        str(ny*nu+1:ny*nu+ny) = 'B';

        % Add columns
        cplex.addCols(c, [], x_L, x_U, char(str));

        % Add indicator constraints on columns
        for i = 1:ny
        counter = 0;
        for j = 1:nu
            a = zeros(nu*ny+ny,1);
            a(i+counter) = 1;
            cplex.addIndicators(nu*ny+i,1,a,'E',0);
            counter = counter+ny;
         end
        end

        % Update CPLEX class
        cplex.Model.sense = 'minimize';
        cplex.Model.ctype = char(str);
        cplex.addRows(lhs, A, rhs);
        cplex.Model.Q = Q;
    end
end
```

A.2 MIQP solver for global self-optimizing control

The function **miqp_solver_gsoc** solves the mixed integer quadratic programming (MIQP) problem for selecting the top ns global self-optimizing control structures of nm measurements.

```
function [H,loss,sset] = miqp_solver_gsoc(YY,Gy,Juu,nm,ns)

    [ny,nu]      = size(Gy);
    sigma        = zeros(1,ny);

    if nm == ny
        ns = 1;
    end

    for k = 1:ns

try
    % Initialize CPLEX object
    cplex = Cplex('miqp_solver_gsoc');
    cplex.DisplayFunc = [];

    % Fill in the data for the problem using populatebyrow
    populatebyrow(cplex,YY,Gy,Juu,nm,sigma);

    % Optimize the problem
    cplex.solve();

    xn = cplex.Solution.x;
    H(:,:,k) = reshape(xn(1:ny*nu),ny,nu)';

    % Compute average loss
    loss(k) = 1/2*norm(YY*H(:,:,k)','fro')^2;

    % Select best subset
    sigma(k,:) = xn(nu*ny+1:end)';
    sset(k,:)  = find(sigma(k,:)==1);

catch m
    disp(m.message);
    throw (m);
end
    end

    function populatebyrow(cplex,YY,Gy,Juu,nm,sigma)

        Y               = (YY)';
        Juu12           = sqrtm(Juu);
        [ny,nu]         = size(Gy);

        % Vectorization
        Jn = []; Yn = []; GnyT = [];
        for i = 1:nu
            GnyT        = blkdiag(GnyT,Gy');
            Jn          = [Jn;Juu12(i,:)'];
            Yn          = blkdiag(Yn,Y);
        end
        Fn              = Yn*Yn';

        % Objective function
        Q               = 2*blkdiag(Fn,zeros(ny,ny));
        c               = zeros(size(Q,1),1);

        % Constraint matrix and bounds
        GnyT_aug        = [GnyT zeros(nu*nu,ny)];
        P_aug           = [zeros(1,ny*nu) ones(1,ny)];
        sigma_aug       = [zeros(size(sigma,1),ny*nu) sigma];
        sigma1          = zeros(1,length(sigma_aug));
        sigma1(nu*ny+1) = 1;
        A               = [GnyT_aug;P_aug;sigma_aug;sigma1];
        lhs             = [Jn;nm;zeros(size(sigma,1),1);1];
        rhs             = [Jn;nm;(nm-1)*ones(size(sigma,1),1);1];

        % Bounds
        x_L             = [-Inf*ones(nu*ny,1);zeros(ny,1)];
        x_U             = [Inf*ones(nu*ny,1);ones(ny,1)];

        % Define variable type
        str(1:ny*nu) = 'C';
        str(ny*nu+1:ny*nu+ny) = 'B';

        % Add columns
        cplex.addCols(c, [], x_L, x_U, char(str));

        % Add indicator constraints on columns
        for i = 1:ny
        counter = 0;
        for j = 1:nu
```

```
            a = zeros(nu*ny+ny,1);
            a(i+counter) = 1;
            cplex.addIndicators(nu*ny+i,1,a,'E',0);
            counter = counter+ny;
         end
        end

        % Update CPLEX class
        cplex.Model.sense = 'minimize';
        cplex.Model.ctype = char(str);
        cplex.addRows(lhs, A, rhs);
        cplex.Model.Q = Q;
    end
end
```

A.3 Computing the steady-state RGA matrix

The function **rga** computes the steady-state relative gain array (RGA) matrix of the plant. This is used for selecting the input-output pairings in the decentralized PI controllers.

```
function [R,S]=rga(G)

% By Yi Cao at Cranfield University on 7th March 2008
%
% References:
%
% 1. Bristol, E.H., On a new measure of interactions for multivariable
% process control. IEEE Trans. Automatic Control, AC-11:133-134, 1966.
% 2. Cao, Y and Rossiter, D, An input pre-screening technique for control
% structure selection, Computers and Chemical Engineering, 21(6), pp.
% 563-569, 1997.

% Check input and output
error(nargchk(1,1,nargin));
error(nargoutchk(0,2,nargout));

% The RGA
R=pinv(G.').*G;
if nargout<2
    return
end

[m,n]=size(G);
% The square case
if n==m
    S=R;
    while norm(round(S)-S)>1e-9
        S(S<0)=0;
        S = inv(S.').*S;
    end
    return
end

% The nonsquare case
```

```
S = sqrt(sum(R))';
if m>n
    S = sqrt(sum(R,2));
end
```

A.4 Computing PI settings

The function **pi_settings** computes the PI controller settings based on the SIMC tuning method.

```
function [kp,ki] = pi_settings(H,Gy,tauc)
%% General settings: add paths and packages

% Add paths
CurrentPath = mfilename('fullpath');
MainFolderPath = fileparts(fileparts(CurrentPath));
addpath(genpath(MainFolderPath));

% Add specific packages
import casadi.*

%% RGA pairing

[R,S]=rga(H*Gy);

%% SIMC-PID tuning

m = model_main();
get_y = Function('get_y',{m.x,m.u,m.d},{m.y});
load('nominal.mat')

for nu = 1:size(H,1)

    nc = find(S(:,nu)>0);

    % Define struct for the integrator
    int_struct     = struct();
    int_struct.x   = vertcat(m.t,m.x);
    int_struct.ode = vertcat(1,m.ode);
    int_struct.p   = vertcat(m.u,m.d);

    % Define initial conditions
    x0  = vertcat(-20,x_nom);
    u0  = u_nom;
    d0  = d_nom;
    p   = vertcat(u0,d0);

    xsol = [];

    for i = 1:2

        if i == 1
            % Options structure
            opts = struct('grid',linspace(0,20,100),...
                  'max_num_steps',1e5,'reltol',1e-8,'abstol',1e-12);

            % Define the integrator
            I = integrator('I','cvodes',int_struct,opts);

            % Call the integrator
            sol = I('x0',x0,'p',p);
            xsol = [xsol;full(sol.xf)'];
            x0 = xsol(end,:);
            dim_time_ss = size(xsol,1);

        elseif i == 2
            % Options structure
            opts = struct('grid',linspace(0,100,1000),...
                   'max_num_steps',1e5,'reltol',1e-8,'abstol',1e-12);

            % Step change in input
            u_step = u0;
            u_step(nu) = u_step(nu)*1.01;
            p  = vertcat(u_step,d0);

            % Define the integrator
            I = integrator('I','cvodes',int_struct,opts);

            % Call the integrator
            sol = I('x0',x0,'p',p);
            xsol = [xsol;full(sol.xf)'];
            time = xsol(:,1);

            % Select measurements from states
                % Steady-state part
                for n = 1:dim_time_ss
                    ysol(n,:) = full(get_y(xsol(n,2:end),u_nom,d_nom));
                end
                % Step change response
                for n = dim_time_ss+1:length(time)
                    ysol(n,:) = full(get_y(xsol(n,2:end),u_step,d_nom));
                end

            % Compute controlled variable from measurements
            for n = 1:length(time)
                csol(n) = (H(nc,:)*ysol(n,:)')';
            end

            % Compute controlled variable in deviation variables
            for n = 1:length(time)
                dc(n) = csol(n)-(H(nc,:)*ysol(1,:)')';
            end

            % Compute process gain, time delay and time constant
            du = u_step(nu)-u_nom(nu);
            k(nu) = abs(dc(end))/du;
            theta(nu) = max(time(find(abs(dc)<=1e-2*abs(dc(end)))));
            tau(nu) = 5;
        end
    end

    % Closed-loop time constant
    tau_c(nu) = tauc;

    % Proportinal and integral controller gains
    kp(nu,1) = (1/k(nu))*(tau(nu)/(tau_c(nu)+theta(nu)));
    taui(nu) = min(tau(nu),4*(tau_c(nu)+theta(nu)));
    ki(nu,1) = kp(nu)/taui(nu);
end
end
```

A.5 Computing steady-state losses

The function **ss_loss** computes the steady-state losses for a given control structure defined by the combination matrix $\mathbf{H}$ and the setpoints $\mathbf{c}_s$. The losses are computed for each disturbance in a given range of standard deviation values.

```
function [sigma,L,L_ol] = ss_loss(H,cs,Gy,sigma_limits)
%% General settings

% Determine the paths for this script and the main folder, respectively
script = mfilename('fullpath');
main = fileparts(fileparts(script));

% Add folders and subfolders to current path
addpath(genpath(main));

% Import CasADi
import casadi.*

% Initialize parallel pool
delete(gcp('nocreate'))
clc
parpool;

%% Load model

m   = model_main();

%% Create CasADi functions

J_f = Function('J_f',{m.x,m.u,m.d},{m.J});
y_f = Function('y_f',{m.x,m.u,m.d},{m.y});

%% Finding nominal operating point

disp('Finding optimally nominal operating point')
tic

% Load data if available
if isfile(fullfile(main,'Data','nominal.mat')) == 1
    disp('    Loading data')
    load('nominal.mat')

% Compute if data non available
```

```
else
    d_nom          = m.d_nom;
    u0             = m.u0;
    [u_nom,x_nom] = nlp_solver(d_nom,u0);
    save(fullfile('Data','nominal.mat'),'u_nom','x_nom','d_nom')
end

disp('   Elapsed time (seconds):')
disp(toc)

%% Finding degress of freedom for self-optimizing control

disp('Finding degress of freedom for self-optimizing control')
tic

% Load data if available
if isfile(fullfile(main,'Data','sset_soc.mat')) == 1
    disp('   Loading data')
    load('sset_soc.mat')

% Compute if data non available
else
    tol      = 1e-4;
    sset_soc = zeros(1,length(u_nom));
    % Check active constraints on inputs
    for i = 1:length(u_nom)
        if abs(u_nom(i)-m.umin(i))<=tol || abs(u_nom(i)-m.umax(i))<=tol
        sset_soc(1,i) = 1;
        else
        sset_soc(1,i) = 0;
        end
    end
    sset_soc = find(sset_soc == 0);
    save(fullfile('Data','sset_soc.mat'),'sset_soc')
end

disp('   Elapsed time (seconds):')
disp(toc)

%% Computing PI controller settings

disp('Computing PI controller settings')
tic

tauc    = 250;
[kp,ki] = pi_settings(H,Gy,tauc);

disp('   Elapsed time (minutes):')
disp(toc/60)

%% Preparing data for parallel computing

disp('Preparing data for parallel computing')
tic

% Grid with standard deviation values
s0      = sigma_limits(1);
```

```
sf       = sigma_limits(2);
ds       = 0.1;
N        = (sf-s0)/ds+1;
sigma    = linspace(s0,sf,N);

% Disturbance values and standard deviations
Wd = diag(m.Wd);
d_nom = m.d_nom;

% Prepare data for parallel computing
ID = []; Dnom = []; WD = []; SIGMA = [];
for nd = 1:length(Wd)
    ID      = [ID,nd*ones(1,N)];
    Dnom    = [Dnom,d_nom(nd)*ones(1,N)];
    WD      = [WD,Wd(nd)*ones(1,N)];
    SIGMA   = [SIGMA,sigma];
end

for i = 1:length(SIGMA)
    d = d_nom;
    d(ID(i)) = Dnom(i)+SIGMA(i)*WD(i);

    D(:,i) = d;
end

disp('   Elapsed time (seconds):')
disp(toc)

%% Optimization

disp('Optimization')
tic

% Load data if available
if isfile(fullfile(main,'Data','ss_optimization.mat')) == 1
    disp('   Loading data')
    load('ss_optimization.mat')

    disp('   Elapsed time (seconds):')
    disp(toc)

% Compute if data non available
else
    % Create progress monitor bar
    ppm = ParforProgMon('Progress: ',size(D,2));
    u0 = m.u0;
    parfor i = 1:size(D,2)
        d          = D(:,i);
        [u,x]      = nlp_solver(d,u0);
        u_opt(i,:) = u;
        x_opt(i,:) = x;
        % Update progress monitor bar
        ppm.increment();
    end

    for i = 1:size(D,2)
        d         = D(:,i);
        x         = x_opt(i,:)';
        u         = u_opt(i,:)';
        J_opt(i)  = full(J_f(x,u,d));
    end

    save(fullfile('Data','ss_optimization.mat'),'J_opt')

    disp('    Elapsed time (hours):')
    disp(toc/3600)
end

%% Closedloop simulation

disp('Closed-loop simulation')
tic

% Create progress monitor bar
ppm = ParforProgMon('Progress: ',size(D,2));

% Closedloop simulation settings
x0      = x_nom;
u0      = u_nom;
t0      = 0;
tf      = [];
n0      = zeros(size(Gy,1),1);
err_tol = 1e-10;

parfor i = 1:size(D,2)
    d = D(:,i)';
    J = J_opt(i);
    [loss,~,~,~,~,~,~,~]  = closedloop_sim(H,kp,ki,n0,d,J,cs,x0,...
                                u0,sset_soc,t0,tf,err_tol)
    L(i,1)                = loss;
    % Update progress monitor bar
    ppm.increment();
end

disp('    Elapsed time (minutes):')
disp(toc/60)

%% Openloop simulation

disp('Open-loop simulation')
tic

% Create progress monitor bar
ppm = ParforProgMon('Progress: ',size(D,2));

% Closedloop simulation settings
tf      = 1000;
err_tol = [];

parfor i = 1:size(D,2)
    d = D(:,i)';
    J = J_opt(i);
    [loss,~,~,~,~,~,~,~]  = closedloop_sim(H,kp,ki,n0,d,J,cs,x0,...
                                u0,sset_soc,t0,tf,err_tol)
    L_ol(i,1)           = loss;
    % Update progress monitor bar
    ppm.increment();
end

disp('   Elapsed time (minutes):')
disp(toc/60)


%% Reorganize data

L = reshape(L,length(sigma),length(d_nom))';
L_ol = reshape(L_ol,length(sigma),length(d_nom))';
end
```

A.6 NLP solver

The function **nlp_solver** solves the nonlinear programming problem (NLP) corresponding to optimal operation. In particular, it minimizes the E-factor while satisfying actuator constraints for a given disturbance scenario.

```
function [u,x] = nlp_solver(d0,u0);
%% General settings: add paths and packages

% Add paths
CurrentPath = mfilename('fullpath');
MainFolderPath = fileparts(fileparts(CurrentPath));
addpath(genpath(MainFolderPath));

% Add specific packages
import casadi.*

%% Optimization parameters

m = model_main();        % Load model
J = m.J;                 % Cost function
x0 = vertcat(1,m.x0);    % Initial states
umax = m.umax;           % Upper bounds for inputs
umin = m.umin;           % Lower bounds for inputs

%% Improve x0 by simulating the system up to steady-state

% Create integrator
int_struct     = struct();
int_struct.x   = vertcat(m.t,m.x);       % Time + Differential states
int_struct.ode = vertcat(1,m.ode);       % dt/dt + Differential equations
int_struct.p   = vertcat(m.u,m.d);       % Parametrize u and d
opts = struct('grid',linspace(0,100,100),...
'max_num_steps',1e5,'reltol',1e-8,'abstol',1e-12);
I = integrator('I','cvodes',int_struct,opts);

% Call the integrator
p  = vertcat(u0,d0);
sol = I('x0',x0,'p',p);
```

```matlab
xsol = full(sol.xf)';

% New initial guess
x0 = xsol(end,2:end)';

%% Optimization

active_artificial_bounds = 1;
counter = 0;

while active_artificial_bounds == 1
    counter = counter+1;

% Create NLP solver
prob_struct   = struct();
prob_struct.f = J;                      % Objective
prob_struct.x = vertcat(m.x,m.u);       % Decision variables
prob_struct.g = m.ode;                  % (In)equality constraints
prob_struct.p = m.d;                    % Parameters: the disturbances

% Initial guess
w0 = vertcat(x0,u0);                    % Initial guess: SS operating point
p  = d0;                                % Use d0 as disturbance realization

% Exploration step
step = 0.2;
% Artificial upper bounds
for k = 1:4
    if u0(k)+step >= umax(k)
        ubound(1,k)  = umax(k);
    elseif u0(k)+step < umax(k)
        ubound(1,k)   = u0(k)+step;
    end
end
% Artificial lower bounds
for k = 1:4
    if u0(k)-step<=umin(k)
        lbound(1,k)  = umin(k);
    elseif u0(k)-step>umin(k)
        lbound(1,k)   = u0(k)-step;
    end
end

lbw = [zeros(1,length(x0)), lbound, 0.2, 0.5];
ubw = [0.025*ones(1,length(x0)), ubound, 0.2 0.5];

opts = struct;
opts.ipopt.acceptable_tol = eps;
opts.ipopt.tol = eps;
opts.ipopt.max_cpu_time = 3*3600;
nlp = nlpsol('nlp','ipopt',prob_struct,opts);

% Call NLP solver
sol_nlp = nlp('p',p,'x0',w0,'lbx',lbw,'ubx',ubw,'lbg',0,'ubg',0);
x0    = full(sol_nlp.x(1:length(x0)));
u0    = full(sol_nlp.x(length(x0)+1:end));

```

```
InputBounds(:,1) = lbw(end-5:end)';
InputBounds(:,2) = u0;
InputBounds(:,3) = ubw(end-5:end)'

% Check artificial upper bound on the states
if max(x0)<0.025
else
   disp('WARNING: artificial bound on states active')
   pause
end

% Check if artificial bounds are active
for n = 1:4
        if abs(u0(n)-InputBounds(n,1))<1e-4 ||...
                abs(u0(n)-InputBounds(n,3))<1e-4;
        active_artificial_bounds(n) = 1;
        else
        active_artificial_bounds(n) = 0;
        end
end

% Remove real bounds from this set
for n = 1:4
        if abs(u0(n)-umin(n))<1e-4 || abs(u0(n)-umax(n))<1e-4;
        active_artificial_bounds(n) = 0;
        else
        active_artificial_bounds(n) = active_artificial_bounds(n);
        end
end


if sum(active_artificial_bounds)>0
    active_artificial_bounds = 1;
    disp('New iteration required: artificial bounds active')
else
    active_artificial_bounds = 0;
    disp('Local optimum found')
end

u = u0;
x = x0;
end
end
```

A.7 Closed-loop simulator

The function **closedloop_sim** simulates the system in closed-loop for a given disturbance and sensor noise realization. The termination criterion can either be given as the final simulation time or as a certain error tolerance.

```
function [loss,J_c,J_opt,time,uend,xend,u,err] = ...
    closedloop_sim(H,kp,ki,n,d,J_opt,cs,x0,u0,sset_soc,t0,tf,err_tol)
%% General settings: add paths and packages

```

```
% Add paths
CurrentPath = mfilename('fullpath');
MainFolderPath = fileparts(fileparts(CurrentPath));
addpath(genpath(MainFolderPath));

% Add specific packages
import casadi.*

%% Load model
m = model_main();

%% Create CasADi functions
J_f    = Function('J_f',{m.x,m.u,m.d},{m.J});
y_f    = Function('y_f',{m.x,m.u,m.d},{m.y});
Wn_f   = Function('Wn_f',{m.x,m.u,m.d},{m.Wn});

%% Closedloop simulation

    % Number of control inputs
    nu = size(H,1);

    % Define error and integrated error
    e   = cs - H*(m.y+n);
    ie = MX.sym('ie',nu,1);

    % Input classification
    sset_soc;                                   % Self-optimizing control
    sset_acc = setdiff(1:length(u0),sset_soc);  % Active constraint
    control

    % Structure for the integrator
    int_struct        = struct();
    int_struct.x      = vertcat(m.t,m.x,ie);
    int_struct.ode    = vertcat(1,m.ode,e);
    int_struct.p      = vertcat(m.d);
    int_struct.z      = vertcat(m.u);
    if isempty(sset_soc) == 1
        alg_cl = [];
    else
        alg_cl            = m.u(sset_soc)-(u0(sset_soc)+(kp.*e + ki.*ie));
    end
    alg_ol            = m.u(sset_acc)-u0(sset_acc);
    int_struct.alg    = vertcat(alg_cl,alg_ol);

    % Initial conditions
    x0       = vertcat(t0,x0,zeros(nu,1));
    z0       = u0;
    p        = d;

    % Empty vectors to store results
    xsol     = [];
    zsol     = [];

    % Case 1: terminate when error below certain tolerance
    if isempty(err_tol) == 0 && isempty(sset_soc) == 0

        error    = 1;
        counter = 0;

        while error>err_tol

        counter = counter+1;
        tf = 50;
        dt = 1;
        N = (tf-t0)/dt+1;

        if counter == 2
            tf = 10000;
            dt = 20;
            N = (tf-t0)/dt+1;
        elseif counter > 2
            tf = 50000;
            dt = 50;
            N = (tf-t0)/dt+1;
        end

        % Create integrator
        opts = struct('grid',linspace(t0,tf,N));
        I = integrator('I','idas',int_struct,opts);

        % Call the integrator
        sol = I('x0',x0,'z0',z0,'p',p);
        xsol = [xsol;full(sol.xf)'];
        zsol = [zsol;full(sol.zf)'];
        x0   = xsol(end,:);
        z0   = zsol(end,:);
        time = xsol(:,1);

        % Compute outputs
        for t = 1:length(time)
            J_c(t)   = full(J_f(xsol(t,2:end-nu),zsol(t,:),d));
            y(t,:)   = full(y_f(xsol(t,2:end-nu),zsol(t,:),d));
            err(t,:) = cs-H*(y(t,:)'+n);
        end
        error        = max(abs(err(end,:)));
        end

    % Case 2: terminate when final time is reached
    else

        % Create integrator
        opts = struct('grid',linspace(t0,tf,250),...
                      'reltol',1e-12,'abstol',1e-12);
        I = integrator('I','idas',int_struct,opts);

        % Call the integrator
        sol = I('x0',x0,'z0',z0,'p',p);
        xsol = [xsol;full(sol.xf)'];
        zsol = [zsol;full(sol.zf)'];
        x0   = xsol(end,:);
        z0   = zsol(end,:);
        time = xsol(:,1);

        % Compute outputs
        for t = 1:length(time)
            J_c(t)   = full(J_f(xsol(t,2:end-nu),zsol(t,:),d));
            y(t,:)   = full(y_f(xsol(t,2:end-nu),zsol(t,:),d));
            err(t,:) = cs-H*(y(t,:)'+n);
        end

    end

    J_opt        = repmat(J_opt,1,length(time));
    loss         = J_c(end)-J_opt(end);
    uend         = zsol(end,:);
    xend         = xsol(end,2:end-nu);
    u            = zsol;
end
```

A.8 Local self-optimizing control (main)

The function **lsoc** is the main code for selecting self-optimizing controlled variables using a global method. It pre-screens promising candidate controlled variables and then performs a steady-state nonlinear validation based on closed-loop simulations of normally distributed scenarios of disturbances and sensor noise realizations generated by Monte Carlo simulations.

```
function lsoc(Nsubsets,Nscenarios)
%% General settings

% Determine the paths for this script and the main folder, respectively
script = mfilename('fullpath');
main = fileparts(fileparts(script));

% Add folders and subfolders to current path
addpath(genpath(main));

% Import CasADi
import casadi.*

% Initialize parallel pool
delete(gcp('nocreate'))
clc
parpool;

%% Load from model

m     = model_main();

%% Create CasADi functions

J_f   = Function('J_f',{m.x,m.u,m.d},{m.J});
y_f   = Function('y_f',{m.x,m.u,m.d},{m.y});
Wn_f = Function('Wn_f',{m.x,m.u,m.d},{m.Wn});

%% Optimization for nominal operation

```

```
disp('1. Finding optimally nominal operating point')
tic

% Load data if available
if isfile(fullfile(main,'Data','nominal.mat')) == 1
    disp('    Loading data')
    load('nominal.mat')

% Compute if data non available
else
    d_nom           = m.d_nom;
    u0              = m.u0;
    [u_nom,x_nom] = nlp_solver(d_nom,u0);
    save(fullfile('Data','nominal.mat'),'u_nom','x_nom','d_nom')
end

disp('    Elapsed time (seconds):')
disp(toc)

%% Finding degress of freedom for self-optimizing control

disp('2. Finding degress of freedom for self-optimizing control')
tic

% Load data if available
if isfile(fullfile(main,'Data','sset_soc.mat')) == 1
    disp('    Loading data')
    load('sset_soc.mat')

% Compute if data non available
else
    tol       = 1e-4;
    sset_soc = zeros(1,length(u_nom));
    % Check active constraints on inputs
    for i = 1:length(u_nom)
        if abs(u_nom(i)-m.umin(i))<=tol || abs(u_nom(i)-m.umax(i))<=tol
        sset_soc(1,i) = 1;
        else
        sset_soc(1,i) = 0;
        end
    end
    sset_soc = find(sset_soc == 0);
    save(fullfile('Data','sset_soc.mat'),'sset_soc')
end

disp('    Elapsed time (seconds):')
disp(toc)

%% Finding sensitivities at the nominal point: Gy, Gd, Juu, Jud

disp('3. Finding sensitivities at the nominal point: Gy, Gd, Juu, Jud')
tic

% Load data if available
if isfile(fullfile(main,'Data','sensitivities.mat')) == 1
    disp('    Loading data')
    load('sensitivities.mat')
```

```matlab

% Compute if data non available
else
    rf_f     = Function('rf_f',{m.x,m.u,m.d},{m.ode,m.J,m.y},...
                              {'x','u','d'},{'xdot','J','y'});
    rf       = rootfinder('rf','kinsol',rf_f);

    Gy_f     = rf.factory('Gy',{'x','u','d'},{'jac:y:u'});
    Gd_f     = rf.factory('Gd',{'x','u','d'},{'jac:y:d'});
    Ju_f     = rf.factory('Ju',{'x','u','d'},{'jac:J:u'});
    Jud_f    = Ju_f.factory('Jud',{'x','u','d'},{'jac:jac_J_u:d'});
    Juu_f    = Ju_f.factory('Juu',{'x','u','d'},{'jac:jac_J_u:u'});

    args     = {x_nom,u_nom,d_nom};
    Gy       = full(Gy_f(args{:}));
    Gd       = full(Gd_f(args{:}));
    Juu      = full(Juu_f(args{:}));
    Jud      = full(Jud_f(args{:}));

    % Select unconstrained degrees of freedom
    Gy       = Gy(:,sset_soc);
    Juu      = Juu(sset_soc,sset_soc);
    Jud      = Jud(sset_soc,:);

    save(fullfile('Data','sensitivities.mat'),'Gy','Gd','Juu','Jud')
end

% Get dimensions
[ny,nu] = size(Gy);      % Number of measurements and control inputs
nd      = size(Gd,2);    % Number of disturbances

disp('    Elapsed time (seconds):')
disp(toc)

%% Pre-screening controlled variables using local methods

disp('4. Pre-screening controlled variables using local methods')
tic

% Load data if available
if isfile(fullfile(main,'Data','lsoc_candidates.mat')) == 1
    disp('    Loading data')
    load('lsoc_candidates.mat')

    disp('    Elapsed time (seconds):')
    disp(toc)

% Compute if data non available
else
    % Create progress monitor bar
    ppm = ParforProgMon('Progress: ',ny-nu+1);

    % Standard deviation values of disturbances and sensor noise
    Wd = m.Wd;
    Wn = full(Wn_f(x_nom,u_nom,d_nom));

    % Solve minimum loss problem using the MIQP formulation
    parfor nm = nu:ny
        % Call MIQP solver
        [H,loss,sset] = miqp_solver_lsoc(Gy,Gd,Juu,Jud,Wd,Wn,nm,Nsubsets);
        % Store results
        local_loss{nm,1} = loss;
        local_H{nm,1}    = H;
        local_sset{nm,1} = sset;
        % Update progress monitor bar
        ppm.increment();
    end

    % Save results in folder 'Data'
    save(fullfile('Data','lsoc_candidates.mat'),...
        'local_loss',...    % Average losses (local methods)
        'local_H',...       % Selection matrices
        'local_sset')       % Optimal subsets of measurements

    disp('   Elapsed time (hours):')
    disp(toc/3600)

end

%% Computing PI controller settings

disp('5. Computing PI controller settings')
tic

% Load data if available
if isfile(fullfile(main,'Data','lsoc_pi_settings.mat')) == 1
    disp('   Loading data')
    load('lsoc_pi_settings.mat')

    disp('   Elapsed time (seconds):')
    disp(toc)

% Compute if data non available
else
    % Organize data in a matrix (ParData) suited for parallel computing
    meas = []; subsets = [];
    for nm = 1:ny-(nu-1)
        if nm<ny-(nu-1)
            meas    = [meas,(nm+nu-1)*ones(1,Nsubsets)];
            subsets = [subsets,1:Nsubsets];
        else
            meas    = [meas,(nm+nu-1)];
            subsets = [subsets,1];
        end
    end
    ParData      = [];
    ParData(1,:) = meas;
    ParData(2,:) = subsets;

    % Create progress monitor bar
    ppm = ParforProgMon('Progress: ',size(ParData,2));

    % Compute PI controller settings in parallel
    parfor i = 1:size(ParData,2)
```

```
        nm          = ParData(1,i);
        ns          = ParData(2,i);
        H           = local_H{nm,1}(:,:,ns);
        tauc        = 250;
        [kp,ki]     = pi_settings(H,Gy,tauc);
        PI_kp{1,i}  = kp;
        PI_ki{1,i}  = ki;
        % Update progress monitor bar
        ppm.increment();
    end

    % Reorganize data
    for i = 1:size(ParData,2)
        nm = ParData(1,i);
        ns = ParData(2,i);

        pi_kp{nm,1}(:,ns) = PI_kp{1,i};
        pi_ki{nm,1}(:,ns) = PI_ki{1,i};
    end

    % Save results in folder 'Data'
        save(fullfile('Data','lsoc_pi_settings.mat'),...
            'ParData',...    % Parallel data
            'pi_kp',...      % Proportinal controller gain
            'pi_ki')         % Integral controller gain

        disp('   Elapsed time (minutes):')
        disp(toc/60)
end

%% Generating and re-optimizing uncertainty scenarios

title1 = '6. Generating and re-optimizing ';
title2 = ' uncertainty scenarios';
disp([title1,num2str(Nscenarios),title2])
tic

% Load data if available
file = fullfile(main,'Data','uncertainty_scenarios.mat');
if isfile(file) == 1
    disp('   Loading data')
    load('uncertainty_scenarios.mat')

    disp('   Elapsed time (seconds):')
    disp(toc)

% Compute if data non available
else

    % Standard deviation values of disturbances and sensor noise
    Wd = diag(m.Wd);
    Wn = diag(full(Wn_f(x_nom,u_nom,d_nom)));

    % Generate normally distributed uncertainty scenarios
    for i = 1:nd
        D(:,i) = normrnd(d_nom(i),Wd(i),Nscenarios,1);
    end
    for i = 1:ny
        N(:,i) = normrnd(0,Wn(i),Nscenarios,1);
    end

    % Create progress monitor bar
    ppm = ParforProgMon('Progress: ',Nscenarios);
    u0 = m.u0;
    parfor scenario = 1:Nscenarios
        d                    = D(scenario,:)';
        [u,x]                = nlp_solver(d,u0);
        u_opt(scenario,:)    = u;
        x_opt(scenario,:)    = x;
        % Update progress monitor bar
        ppm.increment();
    end

    % Evaluate cost functions
    for scenario = 1:Nscenarios
        d                = D(scenario,:)';
        x                = x_opt(scenario,:)';
        u                = u_opt(scenario,:)';
        J_opt(scenario) = full(J_f(x,u,d));
    end

    % Save results in folder 'Data'
    save(fullfile('Data','uncertainty_scenarios.mat'),...
        'D',...     % Scenarios of normally distributed disturbances
        'N',...     % Scenarios of normally distributed sensor noise values
        'J_opt')    % Optimal cost functions

    disp('    Elapsed time (hours):')
    disp(toc/3600)
end

%% Monte Carlo simulations

disp('7. Monte Carlo simulations')
tic

% Load data if available
if isfile(fullfile(main,'Data','lsoc_validation.mat')) == 1
    disp('    Loading data')
    load('lsoc_validation.mat')

    disp('    Elapsed time (seconds):')
    disp(toc)

% Compute if data non available
else
    % Organize data in a matrix (ParData) suited for parallel computing
    meas = []; subsets = []; scenarios = [];
    for nm = 1:ny-(nu-1)
        if nm<ny-(nu-1)
            meas = [meas,(nm+nu-1)*ones(1,Nsubsets*Nscenarios)];
            for ns = 1:Nsubsets
                subsets   = [subsets,ns*ones(1,Nscenarios)];
                scenarios = [scenarios,1:Nscenarios];
            end
        else
            meas      = [meas,(nm+nu-1)*ones(1,Nscenarios)];
            subsets   = [subsets,ones(1,Nscenarios)];
            scenarios = [scenarios,1:Nscenarios];
        end
    end
    ParData      = [];
    ParData(1,:) = meas;
    ParData(2,:) = subsets;
    ParData(3,:) = scenarios;

    % Create progress monitor bar
    ppm = ParforProgMon('Progress: ',size(ParData,2));

    % Closedloop simulation settings
    x0      = x_nom;
    u0      = u_nom;
    t0      = 0;
    tf      = [];
    err_tol = 1e-6;
    y_nom   = full(y_f(x_nom,u_nom,d_nom));

    % Monte Carlo simulations in parallel
    parfor i = 1:size(ParData,2)
        try
        nm = ParData(1,i);
        ns = ParData(2,i);
        scenario = ParData(3,i)

        H   = local_H{nm,1}(:,:,ns);
        cs  = H*y_nom;
        kp  = pi_kp{nm,1}(:,ns);
        ki  = pi_ki{nm,1}(:,ns);
        d   = D(scenario,:);
        n   = N(scenario,:)';
        n0  = zeros(size(n));
        J   = J_opt(scenario);

        [loss,~,~,~,~,~,~,~]  = closedloop_sim(H,kp,ki,n,d,J_opt,cs,x0,...
                                u0,sset_soc,t0,tf,err_tol)
        Loss(i,1)             = loss;
        [loss0,~,~,~,~,~,~,~] = closedloop_sim(H,kp,ki,n0,d,J_opt,cs,x0
    ,...
                                u0,sset_soc,t0,tf,err_tol)
        Loss_d(i,1)           = loss0;
        Loss_n(i,1)           = loss-loss0;
        % Update progress monitor bar
        ppm.increment();
        catch
            warning(['Closed-loop simulation failed at nm = ',...
                     num2str(nm),', ns = ',num2str(ns),...
                     ' and nd = ',num2str(scenario)])
        end
    end

    % Reorganize data
    for i = 1:size(ParData,2)
        nm = ParData(1,i);
        ns = ParData(2,i);
        scenario = ParData(3,i);

        L{nm,ns}(scenario,1)  = Loss(i,1);
        Ld{nm,ns}(scenario,1) = Loss_d(i,1);
        Ln{nm,ns}(scenario,1) = Loss_n(i,1);
    end

    % Compute average losses
    for nm = nu:ny
        for ns = 1:Nsubsets
            if nm == ny
                ns = 1;
            end
            L_av(nm,ns)    = mean(L{nm,ns}(:,1));
            Ld_av(nm,ns)   = mean(Ld{nm,ns}(:,1));
            Ln_av(nm,ns)   = mean(Ln{nm,ns}(:,1));
        end
    end

    % Save results in folder 'Data'
        save(fullfile('Data','lsoc_validation.mat'),...
             'L',...         % Nonlinear losses (disturbances and noise)
             'Ld',...        % Nonlinear losses (only disturbances)
             'Ln',...        % Nonlinear losses (only noise present)
             'L_av',...      % Average nonlinear losses (dist. and noise)
             'Ld_av',...     % Average nonlinear losses (only dist.)
             'Ln_av')        % Average nonlinear losses (only noise)

    disp('   Elapsed time (hours):')
    disp(toc/3600)
end

%% Selecting top subset for each measurement

disp('8. Selecting top subset for each measurement')
tic

% Load data if available
if isfile(fullfile(main,'Data','lsoc_top_sets.mat')) == 1
    disp('   Loading data')
    load('lsoc_top_sets.mat')

% Compute if data non available
else

    for nm = nu:ny
        ns_top             = find(L_av(nm,:) == min(L_av(nm,:)));
        if nm == ny
            ns_top = 1;
        end
        top_L(nm,1)        = L_av(nm,ns_top);
        top_Ld(nm,1)       = Ld_av(nm,ns_top);
        top_Ln(nm,1)       = Ln_av(nm,ns_top);
        top_L_local(nm,1)  = local_loss{nm,1}(1,ns_top);
        top_H{nm,1}         = local_H{nm,1}(:,:,ns_top);
        top_sset{nm,1}      = local_sset{nm,1}(ns_top,:);
    end

    % Save results in folder 'Data'
    save(fullfile('Data','lsoc_top_sets.mat'),...
        'top_L',...         % Average losses (disturbances and sensor noise
    )
        'top_Ld',...        % Average losses (only disturbances)
        'top_Ln',...        % Average losses (only sensor noise)
        'top_L_local',...   % Average losses from local methods
        'top_H',...         % Selection matrices
        'top_sset')         % Sets of measurements
end

disp('    Elapsed time (seconds):')
disp(toc)
end
```

A.9 Global self-optimizing control (main)

The function **gsoc** is the main code for selecting self-optimizing controlled variables using a global method. It pre-screens promising candidate controlled variables and then performs a steady-state nonlinear validation based on closed-loop simulations of normally distributed scenarios of disturbances and sensor noise realizations generated by Monte Carlo simulations.

```
function gsoc(Nsamples,Nsubsets,Nscenarios)
%% General settings

% Determine the paths for this script and the main folder, respectively
script = mfilename('fullpath');
main = fileparts(fileparts(script));

% Add folders and subfolders to current path
addpath(genpath(main));

% Import CasADi
import casadi.*

% Initialize parallel pool
delete(gcp('nocreate'))
clc
parpool;

%% Load from model

m     = model_main();

%% Create CasADi functions

J_f   = Function('J_f',{m.x,m.u,m.d},{m.J});
y_f   = Function('y_f',{m.x,m.u,m.d},{m.y});
```

```
Wn_f = Function('Wn_f',{m.x,m.u,m.d},{m.Wn});

%% Optimization for nominal operation

disp('1. Finding optimally nominal operating point')
tic

% Load data if available
if isfile(fullfile(main,'Data','nominal.mat')) == 1
    disp('   Loading data')
    load('nominal.mat')

% Compute if data non available
else
    d_nom          = m.d_nom;
    u0             = m.u0;
    [u_nom,x_nom] = nlp_solver(d_nom,u0);
    save(fullfile('Data','nominal.mat'),'u_nom','x_nom','d_nom')
end

disp('   Elapsed time (seconds):')
disp(toc)

%% Finding degress of freedom for self-optimizing control

disp('2. Finding degress of freedom for self-optimizing control')
tic

% Load data if available
if isfile(fullfile(main,'Data','sset_soc.mat')) == 1
    disp('   Loading data')
    load('sset_soc.mat')

% Compute if data non available
else
    tol      = 1e-4;
    sset_soc = zeros(1,length(u_nom));
    % Check active constraints on inputs
    for i = 1:length(u_nom)
        if abs(u_nom(i)-m.umin(i))<=tol || abs(u_nom(i)-m.umax(i))<=tol
        sset_soc(1,i) = 1;
        else
        sset_soc(1,i) = 0;
        end
    end
    sset_soc = find(sset_soc == 0);
    save(fullfile('Data','sset_soc.mat'),'sset_soc')
end

disp('   Elapsed time (seconds):')
disp(toc)

%% Preparing matrices for gSOC

disp('3. Preparing matrices for gSOC')
tic

```

```
% Load data if available
file = fullfile(main,'Data','gsoc_matrices.mat');
if isfile(file) == 1
    disp('   Loading data')
    load('gsoc_matrices.mat')

    disp('   Elapsed time (seconds):')
    disp(toc)

% Compute if data non available
else

    rf_f     = Function('rf_f',{m.x,m.u,m.d},{m.ode,m.J,m.y},...
                          {'x','u','d'},{'xdot','J','y'});
    rf       = rootfinder('rf','kinsol',rf_f);

    Gy_f     = rf.factory('Gy',{'x','u','d'},{'jac:y:u'});
    Ju_f     = rf.factory('Ju',{'x','u','d'},{'jac:J:u'});
    Juu_f    = Ju_f.factory('Juu',{'x','u','d'},{'jac:jac_J_u:u'});

    args     = {x_nom,u_nom,d_nom};
    Gy       = full(Gy_f(args{:}));
    Juu      = full(Juu_f(args{:}));

    % Select unconstrained degrees of freedom
    Gy       = Gy(:,sset_soc);
    Gy       = [zeros(1,length(sset_soc)); Gy];
    Juu      = Juu(sset_soc,sset_soc);

    % Get dimensions
    [ny,nu] = size(Gy);       % Number of measurements and control inputs
    nd       = length(d_nom); % Number of disturbances

    % Standard deviation values of sensor noise
    Wd = diag(m.Wd);
    Wn = diag(full(Wn_f(x_nom,u_nom,d_nom)));
    Wn = diag([0; Wn]);

    % Generate normally distributed uncertainty scenarios
    for i = 1:nd
        D(:,i) = normrnd(d_nom(i),Wd(i),Nsamples,1);
    end

    % Create progress monitor bar
    ppm = ParforProgMon('Progress: ',Nsamples);
    u0 = m.u0;
    parfor scenario = 1:Nsamples
        d                   = D(scenario,:)';
        [u,x]               = nlp_solver(d,u0);
        u_opt(scenario,:)   = u;
        x_opt(scenario,:)   = x;
        % Update progress monitor bar
        ppm.increment();
    end

    % Evaluate cost functions
    for scenario = 1:Nsamples
```

```
        d                     = D(scenario,:)';
        x                     = x_opt(scenario,:)';
        u                     = u_opt(scenario,:)';
        y_opt(scenario,:)     = full(y_f(x,u,d));
    end

    % Build matrices Y and YY
    for scenario = 1:Nsamples
        Y(scenario,:) = [1, y_opt(scenario,:)];
    end
    YY = [1/sqrt(Nsamples)*Y; Wn];

    % Save results in folder 'Data'
    save(fullfile('Data','gsoc_matrices.mat'),'Gy','Juu','Y','YY','Wn')

    disp('    Elapsed time (hours):')
    disp(toc/3600)
end

%% Subset selection problem (MIQP formulation)

disp('4. Pre-screening controlled variables using gSOC method')
tic

% Load data if available
if isfile(fullfile(main,'Data','gsoc_candidates.mat')) == 1
    disp('    Loading data')
    load('gsoc_candidates.mat')

    disp('    Elapsed time (seconds):')
    disp(toc)

% Compute if data non available
else
    % Dimensions
    [ny,nu] = size(Gy);

    % Create progress monitor bar
    ppm = ParforProgMon('Progress: ',ny-nu);

    % Solve minimum loss problem using the MIQP formulation
    parfor nm = nu+1:ny
        % Call MIQP solver
        [H,loss,sset]       = miqp_solver_gsoc(YY,Gy,Juu,nm,Nsubsets);
        % Store results
        gsoc_loss{nm-1,1}    = loss;
        gsoc_H{nm-1,1}       = H;
        gsoc_sset{nm-1,1}    = sset;
        % Update progress monitor bar
        ppm.increment();
    end

    % Save results in folder 'Data'
    save(fullfile('Data','gsoc_candidates.mat'),...
        'gsoc_loss',...     % Average losses (local methods)
        'gsoc_H',...        % Selection matrices
        'gsoc_sset')        % Optimal subsets of measurements

    disp('    Elapsed time (hours):')
    disp(toc/3600)

end

% Dimensions
[ny,nu] = size(Gy(2:end,:));
nd      = length(d_nom);

%% Computing PI controller settings

disp('5. Computing PI controller settings')
tic

% Load data if available
if isfile(fullfile(main,'Data','gsoc_pi_settings.mat')) == 1
    disp('    Loading data')
    load('gsoc_pi_settings.mat')

    disp('    Elapsed time (seconds):')
    disp(toc)

% Compute if data non available
else

    % Organize data in a matrix (ParData) suited for parallel computing
    meas = []; subsets = [];
    for nm = 1:ny-(nu-1)
        if nm<ny-(nu-1)
            meas    = [meas,(nm+nu-1)*ones(1,Nsubsets)];
            subsets = [subsets,1:Nsubsets];
        else
            meas    = [meas,(nm+nu-1)];
            subsets = [subsets,1];
        end
    end
    ParData      = [];
    ParData(1,:) = meas;
    ParData(2,:) = subsets;

    % Create progress monitor bar
    ppm = ParforProgMon('Progress: ',size(ParData,2));

    % Compute PI controller settings in parallel
    parfor i = 1:size(ParData,2)
        nm           = ParData(1,i);
        ns           = ParData(2,i);
        H            = gsoc_H{nm,1}(:,:,ns);
        H            = H(:,2:end);
        tauc         = 250;
        [kp,ki]      = pi_settings(H,Gy(2:end,:),tauc);
        PI_kp{1,i}   = kp;
        PI_ki{1,i}   = ki;
        % Update progress monitor bar
        ppm.increment();
    end

    % Reorganize data
    for i = 1:size(ParData,2)
        nm = ParData(1,i);
        ns = ParData(2,i);

        pi_kp{nm,1}(:,ns) = PI_kp{1,i};
        pi_ki{nm,1}(:,ns) = PI_ki{1,i};
    end

    % Save results in folder 'Data'
        save(fullfile('Data','gsoc_pi_settings.mat'),...
            'ParData',...    % Parallel data
            'pi_kp',...      % Proportinal controller gain
            'pi_ki')         % Integral controller gain

        disp('   Elapsed time (minutes):')
        disp(toc/60)
end

%% Generating and re-optimizing uncertainty scenarios

title1 = '6. Generating and re-optimizing ';
title2 = ' uncertainty scenarios';
disp([title1,num2str(Nscenarios),title2])
tic

% Load data if available
file = fullfile(main,'Data','uncertainty_scenarios.mat');
if isfile(file) == 1
    disp('   Loading data')
    load('uncertainty_scenarios.mat')

    disp('   Elapsed time (seconds):')
    disp(toc)

% Compute if data non available
else

    % Standard deviation values of disturbances and sensor noise
    Wd = diag(m.Wd);
    Wn = diag(full(Wn_f(x_nom,u_nom,d_nom)));

    % Generate normally distributed uncertainty scenarios
    for i = 1:nd
        D(:,i) = normrnd(d_nom(i),Wd(i),Nscenarios,1);
    end
    for i = 1:ny
        N(:,i) = normrnd(0,Wn(i),Nscenarios,1);
    end

    % Create progress monitor bar
    ppm = ParforProgMon('Progress: ',Nscenarios);
    u0 = m.u0;
    parfor scenario = 1:Nscenarios
        d                     = D(scenario,:)';
        [u,x]                 = nlp_solver(d,u0);
        u_opt(scenario,:)    = u;
        x_opt(scenario,:)    = x;
        % Update progress monitor bar
        ppm.increment();
    end

    % Evaluate cost functions
    for scenario = 1:Nscenarios
        d                = D(scenario,:)';
        x                = x_opt(scenario,:)';
        u                = u_opt(scenario,:)';
        J_opt(scenario)  = full(J_f(x,u,d));
    end

    % Save results in folder 'Data'
    save(fullfile('Data','uncertainty_scenarios.mat'),...
        'D',...    % Scenarios of normally distributed disturbances
        'N',...    % Scenarios of normally distributed sensor noise values
        'J_opt')   % Optimal cost functions

    disp('    Elapsed time (hours):')
    disp(toc/3600)
end

%% Monte Carlo simulations

disp('7. Monte Carlo simulations')
tic

% Load data if available
if isfile(fullfile(main,'Data','gsoc_validation.mat')) == 1
    disp('    Loading data')
    load('gsoc_validation.mat')

    disp('    Elapsed time (seconds):')
    disp(toc)

% Compute if data non available
else

    % Organize data in a matrix (ParData) suited for parallel computing
    meas = []; subsets = []; scenarios = [];
    for nm = 1:ny-(nu-1)
        if nm<ny-(nu-1)
            meas = [meas,(nm+nu-1)*ones(1,Nsubsets*Nscenarios)];
            for ns = 1:Nsubsets
                subsets   = [subsets,ns*ones(1,Nscenarios)];
                scenarios = [scenarios,1:Nscenarios];
            end
        else
            meas      = [meas,(nm+nu-1)*ones(1,Nscenarios)];
            subsets   = [subsets,ones(1,Nscenarios)];
            scenarios = [scenarios,1:Nscenarios];
        end
    end
    ParData       = [];
    ParData(1,:)  = meas;
    ParData(2,:) = subsets;
    ParData(3,:) = scenarios;

    % Create progress monitor bar
    ppm = ParforProgMon('Progress: ',size(ParData,2));

    % Closedloop simulation settings
    x0      = x_nom;
    u0      = u_nom;
    t0      = 0;
    tf      = [];
    err_tol = 1e-6;

    % Monte Carlo simulations in parallel
    parfor i = 1:size(ParData,2)
        try
        nm = ParData(1,i);
        ns = ParData(2,i);
        scenario = ParData(3,i)

        H   = gsoc_H{nm,1}(:,:,ns);
        cs  = -H(:,1);
        H   = H(:,2:end);
        kp  = pi_kp{nm,1}(:,ns);
        ki  = pi_ki{nm,1}(:,ns);
        d   = D(scenario,:);
        n   = N(scenario,:)';
        n0  = zeros(size(n));
        J   = J_opt(scenario);

        [loss,~,~,~,~,~,~,~]     = closedloop_sim(H,kp,ki,n,d,J,...
                                     cs,x0,u0,sset_soc,t0,tf,err_tol);
        Loss(i,1)                = loss;
        [loss0,~,~,~,~,~,~,~]    = closedloop_sim(H,kp,ki,n0,d,J,...
                                     cs,x0,u0,sset_soc,t0,tf,err_tol);
        Loss_d(i,1)              = loss0;
        Loss_n(i,1)              = loss-loss0;
        % Update progress monitor bar
        ppm.increment();
        catch
            warning(['Closed-loop simulation failed at nm = ',...
                      num2str(nm),', ns = ',num2str(ns),...
                     ' and nd = ',num2str(scenario)])
        end
    end

    % Reorganize data
    for i = 1:size(ParData,2)
        nm = ParData(1,i);
        ns = ParData(2,i);
        scenario = ParData(3,i);

        L{nm,ns}(scenario,1)  = Loss(i,1);
        Ld{nm,ns}(scenario,1) = Loss_d(i,1);
        Ln{nm,ns}(scenario,1) = Loss_n(i,1);
    end

    % Compute average losses
    for nm = nu:ny
        for ns = 1:Nsubsets
            if nm == ny
                ns = 1;
            end
            L_av(nm,ns)     = mean(L{nm,ns}(:,1));
            Ld_av(nm,ns)    = mean(Ld{nm,ns}(:,1));
            Ln_av(nm,ns)    = mean(Ln{nm,ns}(:,1));
        end
    end

    % Save results in folder 'Data'
        save(fullfile('Data','gsoc_validation.mat'),...
             'L',...         % Nonlinear losses (disturbances and noise)
             'Ld',...        % Nonlinear losses (only disturbances)
             'Ln',...        % Nonlinear losses (only noise present)
             'L_av',...      % Average nonlinear losses (dist. and noise)
             'Ld_av',...     % Average nonlinear losses (only dist.)
             'Ln_av')        % Average nonlinear losses (only noise)

    disp('    Elapsed time (hours):')
    disp(toc/3600)
end

%% Selecting top subset for each measurement

disp('8. Selecting top subset for each measurement')
tic

% Load data if available
if isfile(fullfile(main,'Data','gsoc_top_sets.mat')) == 1
    disp('    Loading data')
    load('gsoc_top_sets.mat')

% Compute if data non available
else

    for nm = nu:ny
        ns_top              = find(L_av(nm,:) == min(L_av(nm,:)));
        if nm == ny
            ns_top = 1;
        end
        top_L(nm,1)         = L_av(nm,ns_top);
        top_Ld(nm,1)        = Ld_av(nm,ns_top);
        top_Ln(nm,1)        = Ln_av(nm,ns_top);
        top_L_local(nm,1)   = gsoc_loss{nm,1}(1,ns_top);
        top_H{nm,1}         = gsoc_H{nm,1}(:,:,ns_top);
        top_sset{nm,1}      = gsoc_sset{nm,1}(ns_top,:);
    end

    % Save results in folder 'Data'
    save(fullfile('Data','gsoc_top_sets.mat'),...
        'top_L',...         % Average losses (disturbances and sensor noise
    )
        'top_Ld',...        % Average losses (only disturbances)
        'top_Ln',...        % Average losses (only sensor noise)
        'top_L_local',...  % Average losses from local methods
        'top_H',...        % Selection matrices
        'top_sset')        % Sets of measurements
end

disp('   Elapsed time (seconds):')
disp(toc)
end
```