Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Chemical Engineering

Haakon Løvdokken

Combining Multistage Model Predictive Control with Global Sensitivity Analysis

Master's thesis in Chemical Engineering and Biotechnology
Supervisor: Johannes Jäschke
Co-supervisor: Halvor Aarnes Krog
July 2023

NTNU
Norwegian University of Science and Technology

Haakon Løvdokken

Combining Multistage Model Predictive Control with Global Sensitivity Analysis

Master's thesis in Chemical Engineering and Biotechnology
Supervisor: Johannes Jäschke
Co-supervisor: Halvor Aarnes Krog
July 2023

Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Chemical Engineering

Norwegian University of
Science and Technology

Abstract

In this thesis, the objective was to improve the multistage model predictive control (MS-MPC) by combining with sensitivity analysis (SA). The MS-MPC can better deal with the uncertainty in a given system than the widely used closed-loop model predictive control (CL-MPC, MPC). The uncertainty evolution is modeled by a so-called scenario-tree, in which increases with the number of uncertain parameters. Hence, only a few uncertain parameters should be considered. Which parameters should that be? One may use the most sensitive parameters to an important constraint of the system, in which can be found from SA. The most sensitive parameters might vary during the process. The solution is to include an online SA-based switch for the MS-MPC, that can switch the uncertain parameters in the scenario-tree.

In this thesis' theoretical case study, it was produced penicillin from a fermentation process in an isothermal fed-batch bioreactor. In order to produce the highest penicillin concentration, it is crucial to respect the constraint on the biomass concentration ($X \leqslant 3.7\mathrm{g/l}$). It was found that each of the MS-MPCs with switch performed better than the MS-MPC that always had the Y_x and S_in as the uncertain parameters. It was used three SA methods, i.e., the Sobol' method, Morris screening, and Modified Morris screening. These performed similarly, and it was found that the increase in the number of base samples from $\mathrm{N} = 2^{10}$ to $\mathrm{N} = 2^{12}$ had very little effect. Applying SA every time-step ($t_\mathrm{SA} = 1$) performed better than every fifth time-step ($t_\mathrm{SA} = 5$), and considering two uncertain parameters in the scenario-tree performed better than for three. The best MS-MPC with switch used the Modified Morris screening, two uncertain parameters, $\mathrm{N} = 2^{10}$ and $t_\mathrm{SA} = 1$. It was carried out 25 runs of all the MPCs. The CL-MPC, the MS-MPC without switch, and the MS-MPCs with switch, averaged 7.4, 2.0 and 0.0 constraint violations per run, respectively. The CL-MPC, the MS-MPC without switch, and the best MS-MPC with SA-based switch, averaged $1.63401\mathrm{g/l}$, $1.62107\mathrm{g/l}$ and $1.62412\mathrm{g/l}$ penicillin concentration, respectively. The MS-MPCs with switch perform the best in terms of the constraint handling, and provides higher penicillin concentrations than for the MS-MPC without switch. However, the standard CL-MPC produces the most penicillin. It provides faster increase in the biomass, which compensates for its constraint violations. Thus, the best control is to use the MPC until the biomass approaches its constraint, and then use the best MS-MPC with switch.

Sammendrag

I denne masteravhandlingen var målet å forbedre flerstegs modell prediktiv kontroll (MS-MPC) ved å kombinere det med sensitivitetsanalyse (SA). MS-MPC håndterer usikkerhetene i ett gitt system til større grad enn den vanlige lukket-løkke modell prediktiv kontroll (CL-MPC, MPC). Utviklingen av usikkerhetene modelleres ved et såkalt scenario-tre, og treet øker i størrelse med antallet usikre parametere. Derfor bør det kun være et par usikre parametere som tas hensyn til. Hvilke parametere bør det være? En mulighet er å kun bruke de mest sensitive parameterne til en viktig begrensning på systemet, som kan finnes fra SA. De mest sensitive parameterne kan variere under prosessen. Løsningen er å inkludere en sann-tid SA-basert bryter for MS-MPC, som kan bytte om de usikre parametrene i scenario-treet.

I denne avhandlingens case-studie ble det produsert penicillin ved en fermenteringsprosess i en isotermisk og matet parti-bioreaktor. For å oppnå den største konsentrasjonen av penicillin, er det avgjørende å overholde begrensningen på konsentrasjonen av biomassen ($X \leqslant 3.7\mathrm{g/l}$). Det ble funnet at alle MS-MPC-ene med bryter presterte bedre enn den MS-MPC-en som alltid hadde de usikre parameterene Y_x og S_in. Det ble brukt tre SA-metoder, nemlig Sobol' metoden, Morris skjerming og Modifisert Morris skjerming. Disse presterte svært likt, og det ble funnet at en økning i antallet grunnprøver fra $\mathrm{N} = 2^{10}$ til $\mathrm{N} = 2^{12}$ ikke førte til noen bedre resultater. Å bruke SA ved hvert tidssteg ($t_\mathrm{SA} = 1$) presterte bedre enn for hvert femte tidssteg ($t_\mathrm{SA} = 5$), samt å bruke to usikre parametere i scenario-treet presterte bedre enn for tre usikre parametere. Den beste MS-MPC-en med bryter brukte Modifisert Morris skjerming, to usikre parametere, $\mathrm{N} = 2^{10}$ og $t_\mathrm{SA} = 1$. Alle MPC-ene ble kjørt 25 ganger. CL-MPC-en, MS-MPC-en uten bryter, og de MS-MPC-ene med bryter, hadde gjennomsnittlig 7.4, 2.0 og 0.0 brudd på begrensningen per kjøring, henholdsvis. CL-MPC-en, MS-MPC-en uten bryter, og denne beste MS-MPC-en med bryter, hadde gjennomsnittlig $1.63401\mathrm{g/l}$, $1.62107\mathrm{g/l}$ og $1.62412\mathrm{g/l}$ med konsentrasjon av penicillin, henholdsvis. MS-MPC-ene med bryter presterte best når det gjaldt håndtering av biomasse begrensningen, med høyere konsentrasjon av penicillin enn MS-MPC-en uten bryter. Imidlertid produserer den vanlige CL-MPC mest penicillin. Den gir raskere økning i biomasse, som kompenserer for brudd på begrensningen. Den beste kontrollstrategien er å bruke MPC-en til biomassen nærmer seg begrensningen, og deretter bruke den beste MS-MPC-en med bryter.

Preface

This thesis was written during the spring of 2023 for the course TKP4900 - Chemical Process Technology, Master's Thesis, at the Norwegian University of Science and Technology (NTNU). It concludes the 5-year master's degree in Chemical Engineering and Biotechnology (MTKJ), in which I have spent the last three years at the Department of Chemical Engineering and the last year in the research group of Process Systems Engineering. The thesis is a continuation of the specialization project I worked on during the autumn of 2022.

I would like to express gratitude to my supervisor, Professor Johannes Jäschke, for allowing me the opportunity to work on the topic. I want to give the sincerest thanks to my co-supervisor, Ph.D. candidate Halvor Aarnes Krog, for providing great assistance and feedback this spring.

I am grateful for the friends that I made along the way. Attending lectures and exercise hours with you were always a joy, and I will never forget. We even moved in together for three years. I want to thank my friends and family back home. We kept in contact as if the five years were more like five days. I want to thank my girlfriend, Maren, for the great emotional support.

Declaration of Compliance

I, Haakon Løvdokken, hereby declare that this is an independent work according to the exam regulations of the Norwegian University of Science and Technology (NTNU).

Haakon Løvdokken

Trondheim, Norway
July 10, 2023

Contents

Abstract i

Sammendrag ii

Preface iii

Table of Contents v

List of Figures vii

List of Tables viii

Nomenclature ix

1 Introduction 1
1.1 Motivation . . . 1
1.2 Previous work . . . 2
1.3 Scope of the thesis . . . 2
1.4 Thesis structure . . . 3

2 Model predictive control 4
2.1 Model predictive control . . . 4
2.1.1 Orthogonal collocation . . . 8
2.2 Robust model predictive control . . . 10
2.2.1 Multistage model predictive control . . . 11
2.2.2 Establishing scenario switching rules . . . 14

3 Sensitivity analysis 16
3.1 Sensitivity analysis . . . 16
3.1.1 Local sensitivity analysis . . . 17
3.1.2 Settings for sensitivity analysis . . . 17
3.1.3 Global sensitivity analysis . . . 18
3.2 Sobol' method . . . 19
3.2.1 Saltelli's modification . . . 21

3.3 Morris screening . . . 23
3.3.1 Modified Morris screening . . . 25
3.4 Other GSA methods . . . 27

4 The case study **28**
4.1 The case study . . . 28
4.1.1 Model predictive control . . . 31
4.1.2 Sensitivity analysis . . . 32
4.1.3 Multistage model predictive control . . . 32
4.2 Methodology . . . 34

5 Results and discussion **35**
5.1 Open-loop MPC . . . 36
5.2 Closed-loop MPC . . . 38
5.3 MS-MPC without SA-based switch . . . 40
5.4 Sensitivity Analysis . . . 42
5.4.1 Sobol' method . . . 42
5.4.2 Morris screening . . . 44
5.4.3 Modified Morris screening . . . 46
5.5 MS-MPC with SA-based switch . . . 48
5.6 Comparing all the MPCs . . . 52

6 Conclusion **57**
6.1 Conclusion . . . 57
6.2 Further work . . . 58
6.3 Source criticism . . . 58

Bibliography **59**

Appendix **63**

List of Figures

2.1 Simplified block diagram of the closed-loop MPC[31] . . . 5
2.2 Single-input single output (SISO) optimization-loop[4] . . . 6
2.3 Lagrange polynomials as an approximation to the ODE solution on the finite elements[13] . . . 9
2.4 The uncertainty evolution represented by the scenario-tree for MS-MPC[11] . . . 12
2.5 Simplified block diagram of the multistage MPC combined with sensitivity analysis. . . . 15

3.1 Random sampling, LH sampling and Morris sampling of the input factors, X_1 and X_2. . . . 23

4.1 Simplified flowsheet of the penicillin production in a fed-batch reactor[11] . . . 29

5.1 Ideal input and outputs of OL-MPC. . . . 36
5.2 Mean input and outputs of OL-MPC. . . . 36
5.3 Mean biomass of OL-MPC. . . . 36
5.4 Worst-case biomass of OL-MPC. . . . 36
5.5 Ideal input and outputs of CL-MPC. . . . 38
5.6 Mean input and outputs of CL-MPC. . . . 38
5.7 Mean biomass of CL-MPC. . . . 38
5.8 Worst-case biomass of CL-MPC. . . . 38
5.9 Ideal input and outputs of MS-MPC. . . . 40
5.10 Mean input and outputs of MS-MPC. . . . 40
5.11 Mean biomass of MS-MPC. . . . 40
5.12 Worst-case biomass of MS-MPC. . . . 40
5.13 First-order Sobol' indices (N = 2^{12}). . . . 42
5.14 Total-order Sobol' indices (N = 2^{12}). . . . 42
5.15 First-order Sobol' indices (N = 2^{15}). . . . 42
5.16 Total-order Sobol' indices (N = 2^{15}). . . . 42
5.17 Elementary effects (N = 2^{12}). . . . 44
5.18 EE with one and two SD (N = 2^{12}). . . . 44
5.19 Elementary effects (N = 2^{15}). . . . 44
5.20 EE with one and two SD (N = 2^{15}). . . . 44

5.21 The partial effects (N = 2^{12}). 46
5.22 PE with one and two SD (N = 2^{12}). 46
5.23 The partial effects (N = 2^{15}). 46
5.24 PE with one and two SD (N = 2^{15}). 46
5.25 Worst-case biomass of the MS-MPC of two uncertain parameters (N = 2^{10}, $t_{\mathrm{SA}} = 1$). 48
5.26 Worst-case biomass of the MS-MPC of two uncertain parameters (N = 2^{10}, $t_{\mathrm{SA}} = 5$). 48
5.27 Worst-case biomass of the MS-MPC of two uncertain parameters (N = 2^{12}, $t_{\mathrm{SA}} = 1$). 48
5.28 Worst-case biomass of the MS-MPC of two uncertain parameters (N = 2^{12}, $t_{\mathrm{SA}} = 5$). 48
5.29 Worst-case biomass of the MS-MPC of three uncertain parameters (N = 2^{10}, $t_{\mathrm{SA}} = 1$). 50
5.30 Worst-case biomass of the MS-MPC of three uncertain parameters (N = 2^{10}, $t_{\mathrm{SA}} = 5$). 50
5.31 Worst-case biomass of the MS-MPC of three uncertain parameters (N = 2^{12}, $t_{\mathrm{SA}} = 1$). 50
5.32 Worst-case biomass of the MS-MPC of three uncertain parameters (N = 2^{12}, $t_{\mathrm{SA}} = 5$). 50
5.33 Penicillin concentration at the end of the process, of all the 25 runs, for each of the MPCs. 53
5.34 Mean versus standard deviation (1/2). 56
5.35 Mean versus standard deviation (2/2). 56

List of Tables

2.1 Gauss–Legendre roots and Gauss-Radau roots as the collocation points[1] 9

4.1 Initial or nominal values for the states, inputs and parameters[11] 30
4.2 Lower and upper constraints the outputs, inputs and input changes[11] 31

5.1 Information on each of the MPCs for the 25 runs with respect to the biomass constraint. 52
5.2 Average penicillin concentration at the end of the process, for each of the MPCs. 54

Nomenclature

Abbreviations

Abbreviation	Description
CDF	Cumulative distribution function
CL-MPC	Closed-loop model predictive control
CNST	Constant
CV	Controlled variable
DOF	Degree of freedom
DV	Disturbance variable
EMPC	Economic model predictive control
FF	Factor Fixing
FM	Factor Mapping
FORM	First-order reliability method
FP	Factor Prioritization
GSA	Global sensitivity analysis
LH	Latin Hypercube
LHS	Latin Hypercube sampling
LH-OAT	Latin Hypercube one-at-a-time
LSA	Local sensitivity analysis
MCF	Monte Carlo filtering
MMOR	Modified Morris screening
MORR	Morris screening
MPC	Model predictive control
MS-MPC	Multistage model predictive control
MV	Manipulated variable
NLP	Non-linear program
NMPC	Non-linear model predictive control
OAT	One-at-a-time
ODE	Ordinary differential equation
OL-MPC	Open-loop model predictive control
PID	Proportional-integral-derivative
PPF	Percent-point function
RA	Reliability analysis
RMPC	Robust model predictive control
SA	Sensitivity analysis
SOBO	Sobol' method
SORM	Second-order reliability method
UA	Variance Cutting
UP	Uncertain parameters
VC	Uncertainty analysis

List of symbols - Model predictive control

Latin symbol	Description	Unit
a	Collocation coefficients	-
d	Continuity coefficients	-
f	Process model	-
g	Nonlinear constraints	-
J	Cost function	-
l	Lagrangian polynomial basis	-
$\mathrm{N_m}$	Length of the control horizon	-
$\mathrm{N_p}$	Length of the process horizon	-
$\mathrm{N_{params}}$	Number of uncertain parameters	-
$\mathrm{N_{robust}}$	Length of the robust horizon	-
$\mathrm{N_s}$	Number of scenarios	-
N_θ	Number of uncertainty levels	-
p	Model parameters	-
Q	States importance matrix	-
R	Input movements penalization matrix	-
S	Scenario in the scenario-tree	-
s	Sensitivity indices	-
t	Time element	-
Δt	Time difference	-
u	Computed inputs	-
u_{min}	Minimum inputs	-
u_{max}	Maximum inputs	-
Δu	Changes in the input movements	-
Δu_{min}	Minimum input movements	-
Δu_{max}	Maximum input movements	-
x	Predicted outputs	-
x_{min}	Minimum outputs	-
x_{max}	Maximum outputs	-
$\hat{x}$	Estimated outputs	-
Greek symbol	**Description**	**Unit**
α	Polynomial coefficient	-
θ_{low}	Lower level of uncertainty	-
θ_{nom}	Nominal level of uncertainty	-
θ_{high}	Higher level of uncertainty	-
τ	Dimensionless time	-
ϕ	Uncertain parameters switch	-
ω	Probability of the scenario	-

List of symbols - Sensitivity analysis

Latin symbol	Description	Unit
A	Sampling matrix of the Sobol' method	-
B	Sampling matrix of the Sobol' method	-
C	Sampling matrix of the Sobol' method	-
E	Expectation of the output	-
EE	Elementary effects	-
f	Model function	-
N	Number of base samples	-
$\mathrm{N_X}$	Number of input factors	-
$\mathrm{N_Y}$	Number of model outputs	-
p	Number of levels in the grid	-
PE	Partial effects	-
r	Number of trajectories	-
S	First-order sensitivity index	-
S_{T}	Total-order sensitivity index	-
S^{p}	Non-normalized derivative-based sensitivity	-
S^{σ}	Sigma-normalized derivative-based sensitivity	-
V	Variance of the output	-
X	Input factor(s)	-
Y	Model output(s)	-
Y_{crit}	Critical value of the output	-
y_A	Model outputs of the matrix A	-
y_B	Model outputs of the matrix B	-
y_C	Model outputs of the matrix C	-
$\mathbf{B}$	Lower triangular matrix of only 1's below diagonal	-
$\mathbf{B}$*	Randomized sampling matrix of the input factors	-
$\mathbf{D}$*	Random matrix of +1's and -1's	-
$\mathbf{J}$	Matrix of only 1's	-
$\mathbf{P}$	Random permutation matrix of 1's and 0's	-
$\mathbf{x}$ ($\mathbf{X}$)	Vector of the input factors	-
$\mathbf{x}$* ($\mathbf{X}$*)	Vector of the input factors' random values	-
Greek symbol	**Description**	**Unit**
β	Minimum distance between $\mathbf{X}^*$ and Ω	-
Δ	Increments of the inputs	-
μ	Morris' estimated mean	-
μ^*	Campolongo's estimated mean	-
σ	Morris' estimated standard deviation	-
σ_X	Standard deviations of the inputs	-
σ_Y	Standard deviations of the outputs	-
Ω	Unit hypercube (Sobol' method)	-
Ω	Grid in the unit cube (Morris screening)	-
Ω	Hypersurface in the input space (FORM / SORM)	-

List of symbols - The case study

Latin symbol	Description	Unit
K_{i}	Inhibition constant	$\mathrm{g/l}$
K_{m}	Saturation constant	$\mathrm{g/l}$
P	Product concentration	$\mathrm{g/l}$
S	Substrate concentration	$\mathrm{g/l}$
S_{in}	Inlet substrate concentration	$\mathrm{g/l}$
u	Substrate feed	$\mathrm{m^3/hr}$
V	Reactor volume	$\mathrm{m^3}$
X	Biomass concentration	$\mathrm{g/l}$
Y_{p}	Product yield	$\mathrm{g}(P)/(\mathrm{g}(S))$
Y_{x}	Biomass yield	$\mathrm{g}(P)/(\mathrm{g}(S))$
Greek symbol	**Description**	**Unit**
μ	Specific growth rate	$\mathrm{hr^{-1}}$
μ_{m}	Maximum specific growth rate	$\mathrm{hr^{-1}}$
ν	Specific rate of the product formation	$\mathrm{g}(P)/(\mathrm{g}(X)\cdot\mathrm{hr})$

Chapter 1

Introduction

In this chapter, it is presented motivation, previous work, scope of the thesis and thesis structure.

1.1 Motivation

In non-linear chemical or biochemical processes, such as a fermentation process in an isothermal fed-batch bioreactor, it is expected that a controlled variable of interest may be sensitive to some parameters during one part of the process, but more sensitive to other parameters during another part of the process[33]. As in this thesis' case study, penicillin is being produced from biomass, in which the latter needs to be carefully controlled to not violate its constraint. Exceeding this biomass constraint will lead to less amount of penicillin. In today's practice, the biomass would be controlled by a PID-controller and/or a model predictive controller (MPC)[31]. In the case of using MPC, the uncertainties of the parameters in the process plant are managed with feedback. The MPC calculates the optimal control actions based on the prediction of the process model, and with feeding new measurements back to the MPC, feedback counteracts the uncertainties that the model cannot predict. The model sees the world without disturbances and uncertainties. However, all systems have uncertainty, and it should be accounted for. Say that some parameter that is sensitive to the biomass is uncertain. This could prove problematic for the constraint on the biomass if not the uncertainty is accounted for. Even though there is feedback in the MPC, it may not be enough to respect the constraint. The robust model predictive controller (RMPC) provides better handling of the uncertainty, in which the multistage-MPC has been a promising approach since the early 2010s[11]. Here, the uncertainty evolution is modeled with a so-called scenario-tree. The size of the scenario-tree increases exponentially with how many uncertain parameters that are considered. Thus, only a few of the parameters can be considered at once in order to reduce the optimization problem. Which of the parameters should be selected?

Until now, there have been few attempts at answering this question for non-linear systems. One may use local sensitivity analysis (LSA)[10], but its sensitivity measures are unwarranted when the model is non-linear. Global sensitivity analysis (GSA) is preferred for such models, and is carried out by apportioning the ouputs' uncertainty to the inputs' uncertainty, by the use of probability distributions for the inputs' range. With combining MS-MPC and GSA one may improve robustness of the control in regard to the uncertainty. In today's practice of MS-MPC, a few parameters are chosen uncertain and they are kept uncertain over the whole horizon[11].

This may be an unreasonable assumption, as in processes such as the fermentation of penicillin, the most sensitive parameters can vary over the horizon. Thus, we need an SA-based switch for this MS-MPC that can determine which parameters that are the most sensitive to the biomass. Implementing SA in the MS-MPC comes with computational costs, but should produce higher penicillin concentrations than without SA. Penicillin is a group of antibiotics that are used to treat many different bacterial infections [36]. If we could produce higher amounts of penicillin, it would have become cheaper and many lives could have been saved in poor countries [36].

1.2 Previous work

The work presented in this master's thesis is a continuation of the specialization project I worked on during the autumn of 2022 [15]. By coding in Python, it was conducted an uncertainty analysis and a sensitivity analysis on the open-loop and closed-loop MPC. Here, we used the same case study as for this thesis, i.e., the fermentation process in an isothermal fed-batch bioreactor that produces penicillin. As it is expected, the closed-loop MPC had less constraint violations than the open-loop MPC, but there were still a lot of violations. It was found that the most sensitive parameters to the biomass concentration varied over the horizon, where the Sobol' method was used as the SA. Sobol' method required a lot of samples in order to give reasonable sensitivity measures of the parameters. However, it was later found that we can assume the model output to be normalized, lowering the required sample size. In the specialization project, the output was non-normalized, and the Sobol' method was concluded too computationally expensive for use in the MS-MPC. The sensitivity measures were unreasonable even with the 2^{17} base samples.

The case study was proposed by Srinivasan [33], and later adapted by Lucia [11] for his study on the MS-MPC. The idea of MS-MPC inherited from the multistage stochastic optimization for non-linear model predictive control (NMPC) in the late 2000s. Lucia created the MS-MPC as an approach where the uncertainty evolution is represented by a scenario-tree. If we assume the scenario-tree can describe the uncertainties perfectly, then the MS-MPC strategy represents the control problem exactly, and provides the best solution. The main drawback of the MS-MPC is that it might lead to large optimization problems [11]. However, as an approach to the RMPC, it is quite efficient compared to many of the other methods.

1.3 Scope of the thesis

In this master's thesis, the objective was to figure out whether combining the MS-MPC with SA provides a better solution to the control problem or not. The success was based on three criteria: (i) how much violations there are of the biomass constraint, (ii) how much penicillin is produced, and (iii) the computational costs. It was implemented three different methods of GSA, that is, the Sobol' method, Morris screening and Modified Morris screening. It was studied which of these gave the best success, and whether using SA every time-step or every fifth time-step gave better performance. It was also studied if increasing the sample size had any effect. Moreover, it was studied if it was better to consider two or three uncertain parameters in the scenario-tree. All these MS-MPCs with SA-based switches were compared to each other, as well as with the open-loop MPC, closed-loop MPC and multistage MPC without SA-based switch.

1.4 Thesis structure

In Chapter 2, it is presented theory on the model predictive control. Here, Section 2.1 shows the general theory on MPC, while orthogonal collocation as an approach to the optimization solver is shown in Section 2.2.1. The Robust MPC is presented in Section 2.2 and the multistage MPC is presented in Section 2.2.1. The MS-MPC with SA-based switch is shown in Section 2.2.2.

In Chapter 3, it is presented theory on the sensitivity analysis. Here, Section 3.1 shows the general theory on SA. The local sensitivity analysis is presented in Section 3.1.1, while settings of the GSA are presented in Section 3.1.2, with the global sensitivity analysis in Section 3.1.3. The Sobol' method is described in Section 3.2, with the Saltelli's modification in Section 3.2.1. The Morris screening is described in Section 3.3, with the Modified Morris screening presented in Section 3.3.1. Other relevant GSA methods are shown in Section 3.4.

In Chapter 4, the thesis' case study is presented. Here, Section 4.1 introduces the case study, while its model predictive control is described in Section 4.1.1. Its sensitivity analysis is shown in Section 4.1.2, and then its multistage model predictive control is described in Section 4.1.3. The methodology of the thesis' case study is presented in Section 4.2.

In Chapter 5, the results of the case study are presented and discussed. Section 5.1 shows the open-loop MPC, Section 5.2 shows the closed-loop MPC, and the multistage MPC without the SA-based switch is shown in Section 5.3. The sensitivity analysis is presented in Section 5.4, in which the Sobol' method, Morris screening and Modified Morris screening are presented in Section 5.4.1, 5.4.2 and 5.4.3, respectively. The multistage MPCs with the SA-based switches are shown in Section 5.5, and all the different MPCs are compared in Section 5.6.

In Chapter 6, the conclusions of the master's thesis are shown. Here, Section 6.1 shows the conclusions on the thesis' case study, and Section 6.2 demonstrates the possible future research. In Section 6.3, it is presented source criticism.

The code listings for the thesis are presented in Appendix - code listings.

Chapter 2

Model predictive control

The concept of control is far from new. The origin of control systems can be traced thousands of years back to the ancient times. One of the earliest inventions is the water clock, also known as the *clepsydra*. Although nobody quite knows when or where the first water clock was made, experts estimate as far back as 1500 BCE in the ancient Egypt. The Greeks began using this device around 325 BCE, and named it clepsydra (water thief)[40]. These clocks used regulated water flow in order to measure the time, and there were two types of them; outflow and inflow. The outflow clock was filled with water and drained evenly out of the device, where an observer would look at the lines inside the device to tell the time that had passed. The inflow clock was quite alike, but the device was instead empty at the start and then filled. In the ancient Greece, the water clock was useful for keeping track of the time in law courts[39].

It was not until the 1920s, that the first formal control law of the PID control was presented, by the engineer Nicolas Minorsky[42]. This law was initially designed for ship steering, but has been applied in various other fields since. It was in the 1950s and 1960s, that implementation of PID-control for the chemical industry became popular, as the electronic systems had become cheaper and more reliable. In the 1970s, the first generations of the MPC were designed by the research groups Shell Oil and ADERSA[31]. Since then, MPCs have become common in the process industry, and it is still a popular topic within academic and industrial research[11].

2.1 Model predictive control

Model predictive control (MPC) is a widely used control strategy, in which a system model is the main component for predicting future behavior[22]. MPC solves an optimization problem, that is, it computes optimal control actions from either maximizing or minimizing an objective function over some finite time horizon. MPC comes with several advantages compared to the classic PID-control[31]: (i) the ability to manage multi-input multi-output (MIMO) systems that might involve interactions between the inputs, outputs and disturbances, (ii) offers a systematic approach of managing constraints on the inputs and outputs, (iii), enables coordination of the control calculations with the optimization of the desired objective, and (iv) the ability to offer early warnings of potential issues, given an accurate process model. However, there are also disadvantages with MPC: (i) the requirement of an accurate model, (ii) the online complexity can be computationally expensive, (iii) the performance and the stability can be sensitive to the

tuning parameters, (iv) commissioning costs of the modeling and expenses of the maintenance, (v) lesser transparent control strategy than the classic PID-control [31].

In general control theory, the outputs, inputs and disturbances are commonly referred to as the controlled variables (CVs), manipulated variables (MVs) and disturbance variables (DVs), respectively. The number of independently controlled process variables is represented with the degrees of freedom (DOF). With this terminology in mind, the general objectives of the MPC can be ranked by their importance as follows [21]:

1. Avoid violations of the input and output constraints.
2. Direct the CVs to their optimal steady-state values.
3. Direct the MVs to their optimal steady-state values using the remaining DOFs.
4. Avoid excessive movements of the MVs.
5. If signals and actuators fail, then control as much of the plant as possible.

If the optimization is done offline, meaning the computations are not executed in real-time, it is called open-loop MPC (OL-MPC). This is often used for simple control systems, such as a traditional toaster, or during design of complex systems [19]. On the other hand, if done online with real-time computations, it is called closed-loop MPC (CL-MPC). This feature is of great importance for the MPC, as it allows for feedback from the process plant to the optimization. The feedback provides robustness for the MPC with respect to the uncertainties in the system. A simplified block diagram of the CL-MPC is shown in Figure 2.1.

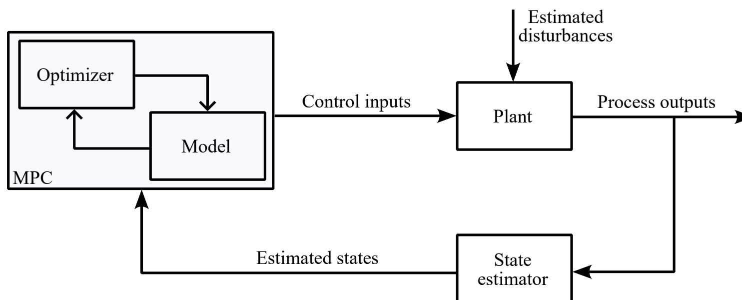


Figure 2.1: Simplified block diagram of the closed-loop MPC [31].

The MPC receives information about the current system variables, which its process model utilizes to predict the future behavior of the system. Here, the optimizer takes these predictions into account when computing the next control actions, and these are given back to the model, creating an inner loop of prediction and optimization [31]. These computations are allowed over the prediction horizon and control horizon, respectively. The computed control inputs are given to the process plant, in which the previous inputs are adjusted with respect to the first of these. Typically, this is done with valves, pumps or motors. Additionally, the plant might also receive estimated (or not estimated) disturbances. The plant responds accordingly to these new values, in which the process outputs are measured after having changed over the pre-defined time-step. Assuming that the process outputs correspond to the states, these are fed to the state estimator, where estimation algorithms (e.g., Kalman Filter) are utilized on the available measurements of the process. However, assuming process outputs are measured perfectly, the state estimator can be neglected. That is, the estimated states are set equal to the process outputs, and are fed

to the MPC block, completing the feedback-loop. The above steps continue in a cyclic manner along the time-horizon of the process. The feedback-loop is also illustrated in Figure 2.2.

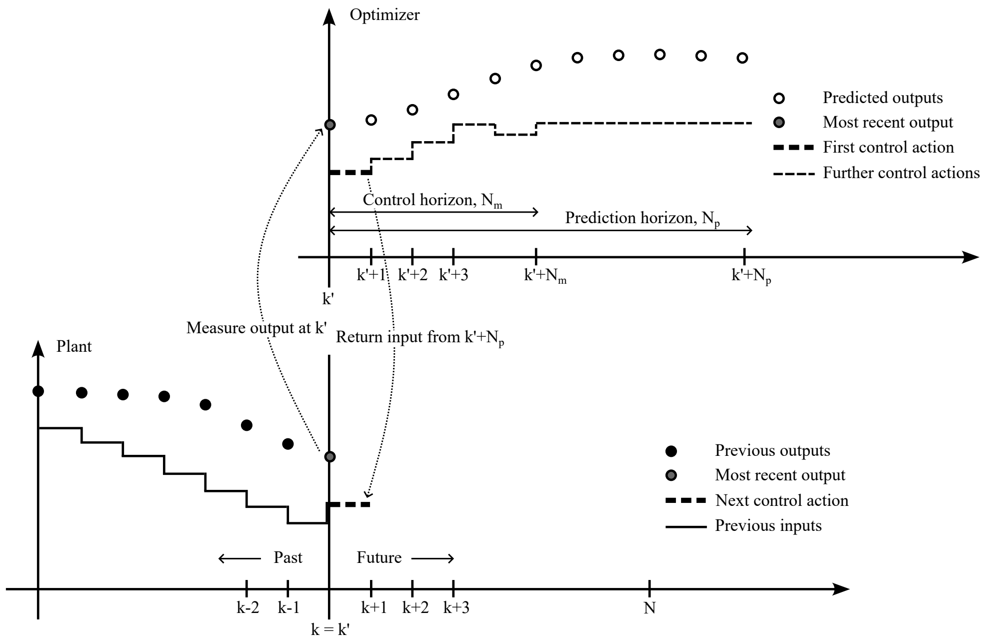


Figure 2.2: Single-input single output (SISO) optimization-loop [4].

Here, in the optimizer, the predicted outputs and the optimal inputs are calculated along the prediction horizon $(\mathrm{k}', \ldots, \mathrm{N_p})$ and control horizon $(\mathrm{k}', \ldots, \mathrm{N_m})$, respectively. It is required that the control horizon cannot exceed the prediction horizon, and both are positive integers, i.e., $1 \leqslant \mathrm{N_m} \leqslant \mathrm{N_p} < \infty$ [31]. Only the first computed control action is fed to the process plant, where the process is carried out at the instance k, giving rise to the new current outputs at the instance $\mathrm{k}+1$. This is fed to the optimizer, completing the feedback-loop, and this continues along the time-horizon $(\mathrm{k}, \ldots, \mathrm{N})$.

For the sake of simplicity, the discrete time k' in the optimizer and k in the plant are now used interchangeably. The typical objective function for the MPC strategy can be written as [4]

$$\min_{x_\mathrm{k}, u_\mathrm{k}} \sum_{\mathrm{k}=0}^{\mathrm{N_p}} \underbrace{J(x_\mathrm{k}, u_\mathrm{k})}_{\text{cost function}} + \sum_{\mathrm{k}=1}^{\mathrm{N_m}} \underbrace{\Delta u_\mathrm{k}^\mathrm{T} \mathrm{R} \Delta u_\mathrm{k}}_{\text{input usage penalties}} \quad , \tag{2.1}$$

where x_k is the predicted outputs, u_k is the computed inputs, and Δu_k denotes the changes in the input movements. The cost function, $J(x_\mathrm{k}, u_\mathrm{k})$, can either be a set-point tracking objective or an economic objective. Both of these are often used, but this thesis focuses explicitly on the economic objective. The input usage penalties, $\Delta u_\mathrm{k}^\mathrm{T} \mathrm{R} \Delta u_\mathrm{k}$, represent the only regularization term of the objective function. It is desirable to have minimal weights in the input movements penalization matrix, R, but weights large enough to ensure there is stability in the controller [4]. The cost function for the economic MPC (EMPC) aims at maximizing profit, minimizing costs (e.g., energy, materials), or maximizing production. The objective function may be written as

$$\min_{x_k, u_k} \sum_{k=0}^{N_p} x_k^T Q x_k + \sum_{k=1}^{N_m} \Delta u_k^T R \Delta u_k, \tag{2.2}$$

where the importance of the outputs are represented with the diagonal weights in the matrix Q. This makes up an unconstrained optimization problem. However, the inclusion of inequality constraints on the outputs and inputs, as well as on the input movements, is an important feature of the MPC. For example, a given flow rate (MV) can only take values between zero and some upper limit determined by the pumps, valves and piping. The product quality (CV) in a given distillation column can only take values between zero and some upper limit determined by the process dynamics [31]. It can also be desirable to include soft-constraints on the product quality, with regard to the customers' demand. The constrained optimization problem can be written

$$\min_{x_k, u_k} \sum_{k=0}^{N_p} x_k^T Q x_k + \sum_{k=1}^{N_m} \Delta u_k^T R \Delta u_k \tag{2.3a}$$

subject to

$$x_{k+1} = f(x_k, u_k, p_k), \qquad \forall\, k = 0, \ldots, N_p - 1 \tag{2.3b}$$

$$g(x_k, u_k, p_k) \leqslant 0, \qquad \forall\, k = 1, \ldots, N_p \tag{2.3c}$$

$$x_{min} \leqslant x_k \leqslant x_{max}, \qquad \forall\, k = 1, \ldots, N_p \tag{2.3d}$$

$$u_{min} \leqslant u_k \leqslant u_{max}, \qquad \forall\, k = 1, \ldots, N_m \tag{2.3e}$$

$$-\Delta u_{max} \leqslant \Delta u_k \leqslant \Delta u_{max}, \qquad \forall\, k = 1, \ldots, N_m \tag{2.3f}$$

where

$$x_0 = \hat{x}_0, \tag{2.3g}$$

$$\Delta u_k = u_k - u_{k-1}, \qquad \forall\, k = 1, \ldots, N_m \tag{2.3h}$$

$$\Delta u_k = 0, \qquad \forall\, k = N_m + 1, \ldots, N_p \tag{2.3i}$$

where the outputs x_k are allowed to vary between the limits x_{min} and x_{max}, and the inputs u_k are allowed to vary between the limits u_{min} and u_{max}, and the input movements Δu_k can only vary between $-\Delta u_{max}$ and Δu_{max}. The model parameters are denoted p_k, and the predicted states are represented with x_{k+1}, which are found from integrating the model, $f(x_k, u_k, p_k)$, using the current process variables. The nonlinear inequality constraints upon the system are given by $g(x_k, u_k, p_k)$. The initial outputs, x_0, equals to the estimated outputs from plant, $\hat{x}_0$.

The algorithm of the CL-MPC can be summarized through Algorithm 1 [4],

Algorithm 1: CL-MPC

for $k = 0, \ldots, N$ **do**
- Obtain the current state, x_k, from the plant.
- Get the control actions, $u_k, \ldots, u_{k+N_p}$, from eq. (2.3) with x_k as the initial state.
- Apply the first control action, u_k, to the plant.

end

Process models are commonly written as ODEs, i.e., $\dot{x} = f(x, u, p)$, but can be discretized, i.e., $x_{k+1} = f(x_k, u_k, p_k)$, and solved as NLPs with an approach as the *orthogonal collocation*.

2.1.1 Orthogonal collocation

Orthogonal collocation is an often used numerical method for MPC, where the system model is fully discretized for the optimization. In orthogonal collocation, the time-horizon of the MPC is broken down into a set of finite elements. Within each of these finite elements, the dynamics of the system are approximated using polynomial interpolation. This is made possible by the enforcement of the system dynamics at the so-called collocation points, through satisfying the orthogonality condition of the polynomial[13]. This results in a set of algebraic equations that can be solved together with the optimization problem. In other words, one can "write out all the integrator equations" and solve them together with the objective and the constraints in the nonlinear program (NLP). This causes large NLPs, but with sparse structures that are exploited by the solver. In summary, orthogonal collocation is a direct transcription method that allows for a simultaneous approach of an optimization problem. For example, consider the ODE[1]:

$$\dot{x} = f(x), \qquad x_0 = \hat{x}_0. \tag{2.4}$$

Assume that the ODE solution, $x(t)$, can be approximated by the J'th order polynomial,

$$x_{\mathrm{k}}^{\mathrm{J}}(t) = \alpha_0 t^0 + \alpha_1 t^1 + \alpha_2 t^2 + \cdots + \alpha_{\mathrm{J}} t^{\mathrm{J}}, \tag{2.5}$$

valid on the finite elements $t \in [t_{\mathrm{k}}, t_{\mathrm{k+1}}]$. The Lagrange interpolation polynomials are utilized on the $\mathrm{i} = 0, \ldots, \mathrm{J}$ interpolation points $(t_{\mathrm{i}}, x_{\mathrm{k,i}})$ in the interval $[t_{\mathrm{k}}, t_{\mathrm{k+1}}]$, which results in[1]

$$x_{\mathrm{k}}^{\mathrm{J}}(t) = \sum_{\mathrm{i}=0}^{\mathrm{J}} l_{\mathrm{i}}(\tau) x_{\mathrm{k,i}}, \tag{2.6}$$

where $l_{\mathrm{i}}(\tau)$ is the Lagrangian polynomial basis with $\tau \in [0, 1]$ as the dimensionless time[1],

$$l_{\mathrm{i}}(\tau) = \prod_{\mathrm{j}=0,\mathrm{j}\neq \mathrm{i}}^{\mathrm{J}} \frac{\tau - \tau_{\mathrm{j}}}{\tau_{\mathrm{i}} - \tau_{\mathrm{j}}}, \qquad \tau = \frac{t - t_{\mathrm{k}}}{\Delta t_{\mathrm{k}}}, \qquad \Delta t_{\mathrm{k}} = t_{\mathrm{k+1}} - t_{\mathrm{k}}. \tag{2.7}$$

It should be noted that the basis polynomial, $l_{\mathrm{i}}(\tau)$, is designed to make sure that $l_{\mathrm{i}}(\tau_{\mathrm{i}}) = 1$ and that $l_{\mathrm{i}}(\tau_{\mathrm{k}}) = 0$ for the interpolation points where $\mathrm{k} \neq \mathrm{i}$. This polynomial ensures that $x_{\mathrm{k}}^{\mathrm{J}}(t_{\mathrm{k+1,i}}) = x_{\mathrm{k+1,i}}$, and it is fitted to all the finite elements, which is shown in Figure 2.3. Hence, all the integration equations to be solved together in the optimizer can be written as[1]

$$\sum_{\mathrm{i}=0}^{\mathrm{J}} \underbrace{\left.\frac{dl_{\mathrm{i}}}{d\tau}\right|_{\tau_{\mathrm{j}}}}_{a_{\mathrm{i,j}}} \frac{x_{\mathrm{k,i}}}{\Delta t_{\mathrm{k}}} = f(x_{\mathrm{k,j}}), \qquad \mathrm{j} = 1, \ldots, \mathrm{J}, \tag{2.8}$$

where the terms $a_{\mathrm{i,j}}$ are all pre-computed constants. Still, there is one last equation remaining, which ensures continuity between the finite elements. This equation can be formulated as[1]

$$x_{\mathrm{k+1,0}} = x_{\mathrm{k}}^{\mathrm{J}}(t_{\mathrm{k+1}}) = \sum_{\mathrm{i}=0}^{\mathrm{J}} \underbrace{l_{\mathrm{i}}(1)}_{d_{\mathrm{i}}} x_{\mathrm{k,i}}, \tag{2.9}$$

in which, likewise for the collocation coefficients, $a_{\mathrm{i,j}}$, the continuity coefficients, d_{i}, are also pre-computed. With regards to the polynomial approximation, there are several options for the orthogonal collocation, in which all have different numbers of collocation points and positions. The most often used are the Gauss-Lobatto, Gauss-Legendre and Gauss-Radau polynomials. The roots τ_{i} of the last two approaches are shown in Table 2.1.

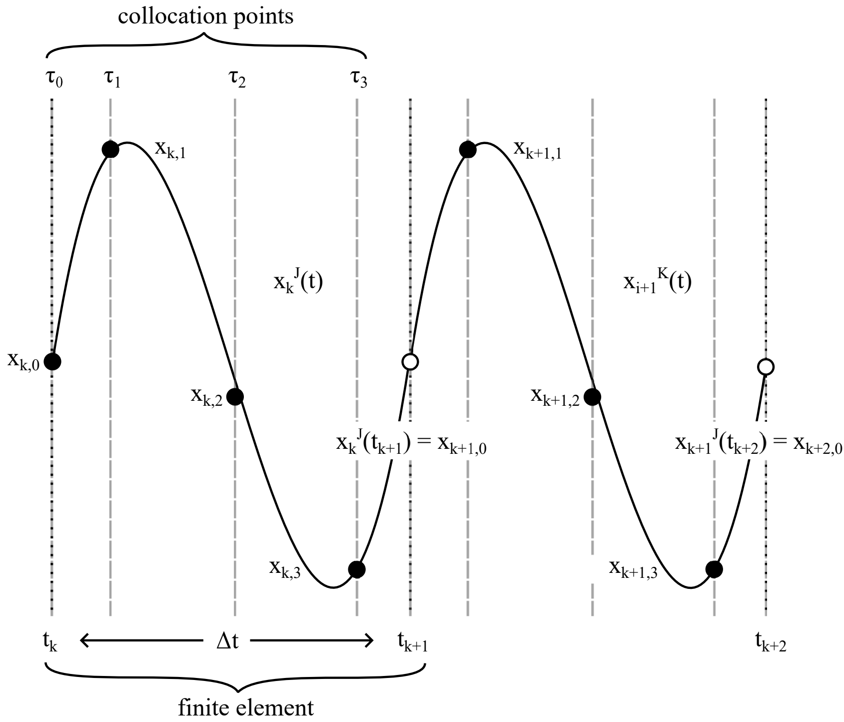


Figure 2.3: Lagrange polynomials as an approximation to the ODE solution on the finite elements [13].

Table 2.1: Gauss–Legendre roots and Gauss-Radau roots as the collocation points [1].

Degree J	Gauss–Legendre roots	Gauss-Radau roots
1	0.500000	1.000000
2	0.211325	0.333333
	0.788675	1.000000
3	0.112702	0.155051
	0.500000	0.644949
	0.887298	1.000000
4	0.069432	0.088588
	0.330009	0.409467
	0.669991	0.787659
	0.930568	1.000000
5	0.046910	0.057104
	0.230765	0.276843
	0.500000	0.583590
	0.769235	0.860240
	0.953090	1.000000

Finally, one can re-write the MPC (i.e., eq. (2.3)) in terms of the orthogonal collocation[1],

$$\min_{x_{\mathrm{k,j}},u_{\mathrm{k,j}}} \sum_{\mathrm{k=0}}^{\mathrm{N_p}} x_{\mathrm{k,j}}^{\mathrm{T}} \mathrm{Q} x_{\mathrm{k,j}} + \sum_{\mathrm{k=1}}^{\mathrm{N_m}} \Delta u_{\mathrm{k,j}}^{\mathrm{T}} \mathrm{R} \Delta u_{\mathrm{k,j}} \tag{2.10a}$$

subject to

$$\sum_{\mathrm{i=0}}^{\mathrm{J}} a_{\mathrm{i,j}} \frac{x_{\mathrm{k,i}}}{\Delta t} = f(x_{\mathrm{k,j}}), \qquad \forall\,(\mathrm{k,j}) \in \mathrm{I} \tag{2.10b}$$

$$x_{\mathrm{k+1,0}} = \sum_{\mathrm{i=0}}^{\mathrm{J}} d_{\mathrm{i}} x_{\mathrm{k,i}}, \qquad \forall\,(\mathrm{k,j}) \in \mathrm{I} \tag{2.10c}$$

$$g(x_{\mathrm{k,j}}, u_{\mathrm{k,j}}, p_{\mathrm{k,j}}) \leqslant 0, \qquad \forall\,(\mathrm{k,j}) \in \mathrm{I} \tag{2.10d}$$

$$x_{\min} \leqslant x_{\mathrm{k,j}} \leqslant x_{\max}, \qquad \forall\,(\mathrm{k,j}) \in \mathrm{I} \tag{2.10e}$$

$$u_{\min} \leqslant u_{\mathrm{k,j}} \leqslant u_{\max}, \qquad \forall\,(\mathrm{k,j}) \in \mathrm{I} \tag{2.10f}$$

$$-\Delta u_{\max} \leqslant \Delta u_{\mathrm{k,j}} \leqslant \Delta u_{\max}, \qquad \forall\,(\mathrm{k,j}) \in \mathrm{I} \tag{2.10g}$$

where

$$x_{0,j} = \hat{x}_0, \qquad \forall\,(\mathrm{k,j}) \in \mathrm{I} \tag{2.10h}$$

$$\Delta u_{\mathrm{k,j}} = u_{\mathrm{k,j}} - u_{\mathrm{k-1,j}}, \qquad \forall\,(\mathrm{k,j}) \in \mathrm{I} \tag{2.10i}$$

$$\Delta u_{\mathrm{k,j}} = 0, \qquad \forall\,(\mathrm{k,j}) \in \mathrm{I} \tag{2.10j}$$

2.2 Robust model predictive control

The standard CL-MPC is based on the nominal system model, that is, possible disturbances and uncertainties are not accounted for. The model views the process as being perfect, and thus, the predictions of the model are deterministic. This is not an issue for systems that are little affected by the uncertainties, but in the real world and for most processes, uncertainties are an important factor. If these are not dealt with, it can lead to plant-model mismatch, which yields worse economic performance and might give constraint violations. One approach to the issue is to incorporate *robustness* into the controller[22]. The robust model predictive control (RMPC) serves this purpose, where it seeks to ensure optimal performance and satisfy constraints while dealing with uncertainties. There are several RMPC approaches, where the most common ones are the min-max MPC, the tube-based MPC and the multistage MPC (MS-MPC)[14].

The min-max MPC was amongst the first attempts at the RMPC. In the open-loop strategy, the controller obtains the control inputs that minimize the objective function of the worst-case realization of the uncertainty, while satisfying the constraints for all the cases. This might give conservative control and infeasibility, due to the lack of feedback. In the closed-loop strategy, the controller instead receives so-called control policies that minimize the objective function. This gives an infinite-dimension optimization problem, that is difficult to solve. It is simplified by optimizing just over a few control policies (e.g., affine policies), but gives suboptimality[14].

The tube-based MPC is a more modern alternative to the min-max approach of the RMPC, that guarantees stability and feasibility [14]. This control strategy is based on the solution of the nominal control problem, and the design of an ancillary controller that ensures the trajectory of the real uncertain system stays within an invariant tube, where the cross-section is centered around the nominal trajectory. This tube-based MPC has been tried modified and improved, mainly in how the cross-sections and ancillary controller are calculated, which yields different computational complexity and conservativeness. However, in order to satisfy the constraints of the real uncertain system, the cross-section for the nominal control problem must be tightened for all the cases. This gives conservative control for nonlinear systems with active constraints. Thus, the method ensures stability and feasibility for uncertain systems, but not optimality [14].

2.2.1 Multistage model predictive control

This thesis focuses explicitly on the multistage MPC (MS-MPC) as an approach of the RMPC. The main idea behind the MS-MPC approach is to model the evolution of the uncertainties by so-called scenario-trees [11]. Thereby, feedback is taken explicitly into account, and the future control inputs are adjusted with respect to this feature in order to counteract the uncertainties. Assuming that the scenario-tree can describe the uncertainties perfectly, the MS-MPC strategy represents the real-time decision problem exactly, and provides the best solution [11]. However, the major drawback of the MS-MPC is that it gives possibly very large optimization problems. The size of the control problem increases exponentially with the prediction horizon and with the number of uncertain parameters and their respective levels [14]. However, robustness is only guaranteed for the uncertainty values that are accounted for in the scenario-tree. When the plant receives parameter values outside of the MS-MPC accounted range, it may result in constraint violations and suboptimality. However, assuming that the scenario-tree can perfectly describe the uncertainty, MS-MPC should excel. The main idea of the method is shown in Figure 2.4.

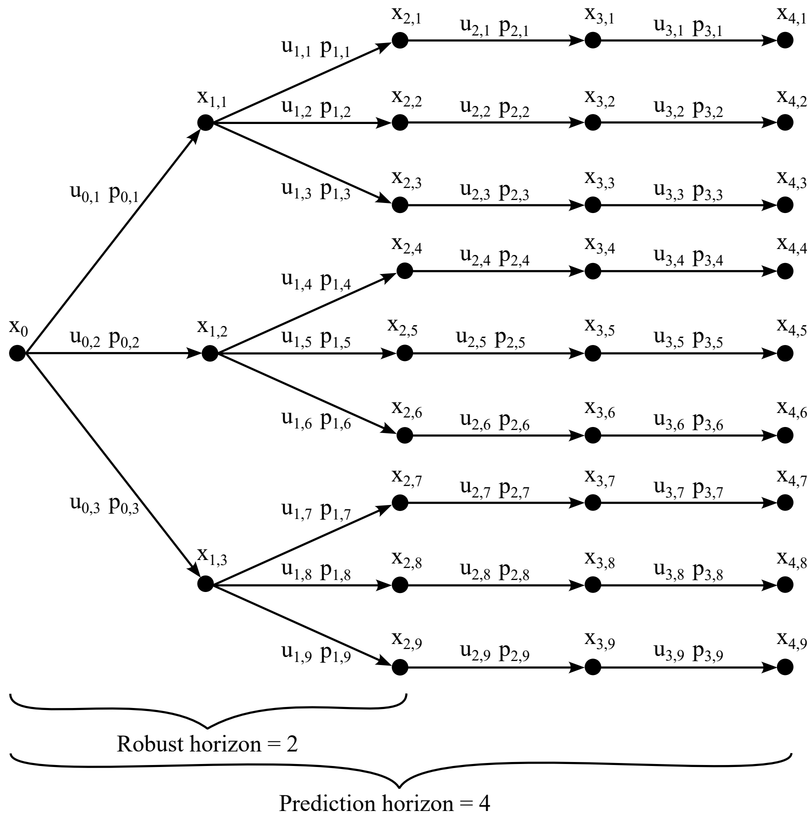


Figure 2.4: The uncertainty evolution represented by the scenario-tree for MS-MPC[11].

Here, the scenario-tree considers different scenarios, where each scenario is defined as a possible realization of all the uncertain parameters at every control instant within the horizon. In the figure, each scenario can be seen as one path from the root, x_0, to the leaf node at the right hand side. For example, the state evolution for the first scenario, S_1, can be written $x_0 \rightarrow x_{1,1} \rightarrow x_{3,1} \rightarrow x_{4,1}$. At every instant, the optimization problem is solved at the root node while explicitly accounting for the uncertainty evolution, and the future control actions, that exploit the information obtained over the branches. With this design, feedback is included into the open-loop control problem, reducing conservativeness[11]. With this feedback feature, control inputs branching from the same node are equal, as they originate from the same values. It is called *non-anticipativity constraints*, and for example, it implies that $u_{1,1} = u_{1,2} = u_{1,3}$, and $u_{1,4} = u_{1,5} = u_{1,6}$, and so on. The objective function for MS-MPC can be written as[11]

$$\min_{x_{\mathrm{k,j}}, u_{\mathrm{k,j}}} \sum_{\mathrm{i}=1}^{\mathrm{N_s}} \underbrace{\omega_{\mathrm{i}} J_{\mathrm{i}}(x_{\mathrm{i}}, u_{\mathrm{i}})}_{\text{scenario objective}}, \tag{2.11}$$

in which x_{i} denotes the predicted outputs and u_{i} denotes the computed inputs for the scenario S_{i} in the scenario-tree. The objective function is the weighted average of the scenario objective, $\omega_{\mathrm{i}} J_{\mathrm{i}}$, where ω_{i} represents the probability of S_{i}, while J_{i} is its respective objective function[11]. Furthermore, the total number of scenarios, $\mathrm{N_s}$, can be obtained through the formula,

$$\mathrm{N_s} = \left(\prod_{\mathrm{i}=1}^{\mathrm{N_{params}}} \mathrm{N}_{\theta \mathrm{i}} \right)^{\mathrm{N_{robust}}}, \tag{2.12}$$

where $\mathrm{N_{params}}$ represents the number of uncertain parameters, and $\mathrm{N_{robust}}$ represents the length of the robust horizon. The number of uncertainty levels, $\mathrm{N}_{\theta\mathrm{i}}$, for the i'th uncertain parameter, normally consists of the three levels, θ_{low}, θ_{nom} and θ_{high}, representing the cases of low, nominal and high values, respectively. The three uncertain cases are combined with certain parameters, e.g., $p_{1,1}$, $p_{1,2}$ and $p_{1,3}$, respectively. It is clear by the eq. (2.12) that the number of scenarios, $\mathrm{N_s}$, increases exponentially with the size of the robust horizon. Thus, with regards to the higher computational expenses, it is generally a good idea to have this tuning low[12]. In most cases, increasing the robust horizon is seen as redundant due to the feedback. The branching is only done for the first $\mathrm{N_{robust}}$ time-steps, and then the uncertain parameters are kept constant to the end of the prediction horizon. If the uncertainties are bounded, unknown, and time-invariant, then $\mathrm{N_{robust}} = 1$ is the suggested choice. Extending eq. (2.11) with respect to eq. (2.1), yields

$$\min_{x_{\mathrm{k,j}}, u_{\mathrm{k,j}}} \sum_{\mathrm{i}=1}^{\mathrm{N_s}} \omega_{\mathrm{i}} \Big(\sum_{\mathrm{k}=0}^{\mathrm{N_p}} \underbrace{J(x_{\mathrm{k,j}}, u_{\mathrm{k,j}})}_{\text{cost function}} + \sum_{\mathrm{k}=1}^{\mathrm{N_m}} \underbrace{\Delta u_{\mathrm{k,j}}^{\mathrm{T}} \mathrm{R} \Delta u_{\mathrm{k,j}}}_{\text{input usage penalties}} \Big), \tag{2.13}$$

where, at the instance, k, and for the scenario, S_j, the predicted outputs are $x_{\mathrm{k,j}}$, the calculated inputs are $u_{\mathrm{k,j}}$, and the input movements are $\Delta u_{\mathrm{k,j}}$. From now on, all the recurring index pairs (k, j) are denoted I. Reformulating this equation with respect to the eq. (2.2), results in[11]

$$\min_{x_{\mathrm{k,j}}, u_{\mathrm{k,j}}} \sum_{\mathrm{i}=1}^{\mathrm{N_s}} \omega_{\mathrm{i}} \Big(\sum_{\mathrm{k}=0}^{\mathrm{N_p}} x_{\mathrm{k,j}}^{\mathrm{T}} \mathrm{Q} x_{\mathrm{k,j}} + \sum_{\mathrm{k}=1}^{\mathrm{N_m}} \Delta u_{\mathrm{k,j}}^{\mathrm{T}} \mathrm{R} \Delta u_{\mathrm{k,j}} \Big), \tag{2.14}$$

which is the unconstrained optimization problem. Having inequality constraints on the system is still vital, and with respect to eq. (2.3), the constrained optimization problem becomes[11]

$$\min_{x_{\mathrm{k,j}}, u_{\mathrm{k,j}}} \sum_{\mathrm{i}=1}^{\mathrm{N_s}} \omega_{\mathrm{i}} \Big(\sum_{\mathrm{k}=0}^{\mathrm{N_p}} x_{\mathrm{k,j}}^{\mathrm{T}} \mathrm{Q} x_{\mathrm{k,j}} + \sum_{\mathrm{k}=1}^{\mathrm{N_m}} \Delta u_{\mathrm{k,j}}^{\mathrm{T}} \mathrm{R} \Delta u_{\mathrm{k,j}} \Big) \tag{2.15a}$$

subject to

$$x_{\mathrm{k+1,j}} = f(x_{\mathrm{k,p(j)}}, u_{\mathrm{k,j}}, p_{\mathrm{k,r(j)}}), \quad \forall\, (\mathrm{k,j}) \in \mathrm{I} \tag{2.15b}$$

$$u_{\mathrm{k,i}} = u_{\mathrm{k,j}} \text{ if } x_{\mathrm{k,p(i)}} = x_{\mathrm{k,p(j)}}, \quad \forall\, (\mathrm{k,j}), (\mathrm{k,i}) \in \mathrm{I} \tag{2.15c}$$

$$g(x_{\mathrm{k,p(j)}}, u_{\mathrm{k,j}}, p_{\mathrm{k,r(j)}}) \leqslant 0, \quad \forall\, (\mathrm{k,j}) \in \mathrm{I} \tag{2.15d}$$

$$x_{\mathrm{min}} \leqslant x_{\mathrm{k,j}} \leqslant x_{\mathrm{max}}, \quad \forall\, (\mathrm{k,j}) \in \mathrm{I} \tag{2.15e}$$

$$u_{\mathrm{min}} \leqslant u_{\mathrm{k,j}} \leqslant u_{\mathrm{max}}, \quad \forall\, (\mathrm{k,j}) \in \mathrm{I} \tag{2.15f}$$

$$-\Delta u_{\mathrm{max}} \leqslant \Delta u_{\mathrm{k,j}} \leqslant \Delta u_{\mathrm{max}}, \quad \forall\, (\mathrm{k,j}) \in \mathrm{I} \tag{2.15g}$$

where

$$x_{0,j} = \hat{x}_0, \quad \forall\, (\mathrm{k,j}) \in \mathrm{I} \tag{2.15h}$$

$$\Delta u_{\mathrm{k,j}} = u_{\mathrm{k,j}} - u_{\mathrm{k-1,j}}, \quad \forall\, (\mathrm{k,j}) \in \mathrm{I} \tag{2.15i}$$

$$\Delta u_{\mathrm{k,j}} = 0, \quad \forall\, (\mathrm{k,j}) \in \mathrm{I} \tag{2.15j}$$

Here, the indices $\mathrm{p(j)}$ and $\mathrm{r(j)}$ denote the parent node and next realized node, respectively. Each of the states, $x_{\mathrm{k+1,j}}$, depend on the parent states, $x_{\mathrm{k,p(j)}}$, the corresponding control inputs, $u_{\mathrm{k,j}}$, and the corresponding realization of the parameters, $p_{\mathrm{k,r(j)}}$ [14]. This regards the nonlinear inequality constraints, g, as well. The non-anticipativity constraints are presented through the so-called if-equality, $u_{\mathrm{k,i}} = u_{\mathrm{k,j}}$ if $x_{\mathrm{k,p(i)}} = x_{\mathrm{k,p(j)}}$, where $\mathrm{i} = 0, \dots, \mathrm{N_s}$ and $\mathrm{j} = 0, \dots, \mathrm{N_s}$. Thus, the control inputs are bounded by the non-anticipativity constraint if they originate from the same nodes. In summary, given that the scenario-trees can perfectly address the uncertainty, there are no constraints violations and the MS-MPC strategy represents the best solution [11].

The algorithm of the MS-MPC can be summarized through Algorithm 2 [4],

Algorithm 2: MS-MPC

for $\mathrm{k} = 0, \dots, \mathrm{N}$ **do**
- Obtain the current state, x_{k}, from the plant.
- Get the control actions, $u_{\mathrm{k}}, \dots, u_{\mathrm{k+N_p}}$, from eq. (2.15) with x_{k} as the initial state.
- Apply the first control action, u_{k}, to the plant.

end

2.2.2 Establishing scenario switching rules

The dimension of the MS-MPC optimization problem increases exponentially with the number of uncertain parameters, $\mathrm{N_{params}}$ as seen by the eq. (2.12). Thus, for the purpose of reducing the computational expenses, not all of the system parameters are considered for the scenario-trees. However, what parameters should be accounted for? An approach is to only consider one, two or three of the most sensitive parameters to an important constraint or to the objective function. In order to decide on the most sensitive parameters, a method of sensitivity analysis is utilized. In literature, implementing local sensitivity analysis has proven successful, greatly reducing the computational expenses while producing the same performance as an all-uncertain-parameters MS-MPC [43]. This method is implemented in real-time. However, for many nonlinear systems, global sensitivity analysis gives a better understanding of the system behavior and interactions than the local sensitivity analysis. This topic is talked about in Chapter 3. For now, just know that sensitivity analysis can give information on what parameters are the most sensitive to an important constraint or the objective function. This information can be given to a switch-rule, and changes the structure of the scenario-tree corresponding to the most sensitive parameters.

The MS-MPC approach with an online switch-rule is not well discussed in literature. Thus, we have to propose a switch-rule ourselves, which generally can be written as

$$\phi_{\mathrm{k+1}} = \phi(s_{\mathrm{k+1}}) = \phi(s_{\mathrm{k+1}}(x_{\mathrm{k}}, u_{\mathrm{k+1}}, p_{\mathrm{k,i}})), \tag{2.16}$$

where $\phi_{\mathrm{k+1}}$ is called the *switch*, containing only the indices of the most sensitive parameters. The switch is a function of the switch-rule, $\phi(s_{\mathrm{k+1}})$, which depends on the sensitivity indices, $s_{\mathrm{k+1}}$, for each parameter from the sensitivity analysis. These sensitivity indices depend on the current outputs, x_{k}, the second control inputs, $u_{\mathrm{k+1}}$, and the random sampled parameters, $p_{\mathrm{k,i}}$, where $\mathrm{i} = 1, \dots, \mathrm{N}$, and where N is the number of base samples from the chosen distribution. The plant model is simulated over these arguments, i.e., $x_{\mathrm{k+1,i}} = f(x_{\mathrm{k}}, u_{\mathrm{k+1}}, p_{\mathrm{k,i}})$, where all outputs $x_{\mathrm{k+1,i}}$ are used in a method of global sensitivity analysis, giving the sensitivity indices,

s_{k+1}, and thus, the indices of the most sensitive parameters, ϕ_{k+1}. A simplified block diagram of the MS-MPC with switch is shown in Figure 2.5.

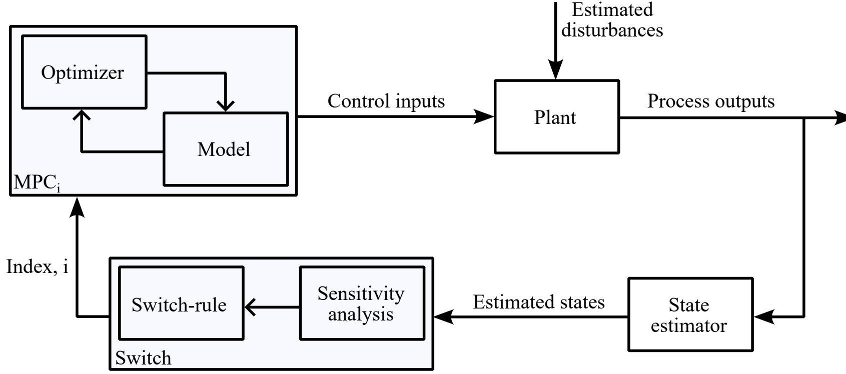


Figure 2.5: Simplified block diagram of the multistage MPC combined with sensitivity analysis.

As an addition to Figure 2.1, the purpose of the switch-block is to return an index, i, for the MPC-block. This index represents the $\mathrm{i} = 1, \ldots, \mathrm{N}$ combination of the uncertain parameters, which is obtained from the switch, ϕ_{k+1}, as shown in eq. (2.16). Here, the number of unique combinations of the uncertain parameters are denoted N, which yields N number of unique MPCs that can be pre-defined offline before online simulation. This saves computational effort, as well as being a systematic way of dealing with different scenario-trees along the horizon.

There are several advantages of using a switch with the MS-MPC: (i) lower computational costs than for the all-uncertain-parameters MS-MPC, (ii) the performance should be better than for not using the switch, (iii) the constraints could very likely be satisfied even though there are uncertainties in the least sensitive parameters that are not accounted for in the scenario-trees, and (iv) it offers a structured approach of choosing uncertain parameters in the scenario-tree.

The algorithm of the MS-MPC with switch can be summarized through Algorithm 3[4],

Algorithm 3: MS-MPC with switch

for $\mathrm{k} = 0, \ldots, \mathrm{N}$ **do**
- Obtain the current state, x_k, from the plant.
- Get the control actions, $u_k, \ldots, u_{k+N_p}$, from eq. (2.15) with x_k as the initial state.
- Obtain the sensitivity indices, s_{k+1}, using the second control action, u_{k+1}, and x_k.
- Get the indices of the most sensitive parameters, ϕ_{k+1}, based on s_{k+1}.
- Choose the correct pre-defined MS-MPC based on ϕ_{k+1}.
- Apply the first control action, u_k, to the plant.

end

Chapter 3

Sensitivity analysis

While the first control system was developed thousands of years ago, sensitivity analysis (SA) is a rather new concept, although its fundamental ideas can be viewed as older. For example, consider Leonardo da Vinci's laws of sliding friction from the 15th century. The friction force between two sliding surfaces should be proportional (i.e., linearly sensitive) to the applied load, while remaining unaffected (i.e., insensitive) to the area between the surfaces. These laws were developed by conducting experiments based on the SA fundamentals, changing only *one factor at a time* and evaluating the impact of the change [26].

It was not until the mid-20th century, that sensitivity analysis got its roots. It was need for an efficient design of physical and chemical experiments, in order to obtain information on the effect of some variables on the other variables in the system. This motivated the evolution of *design of experiments*, which is a vast family of statistical methods [26]. With the evolution of computers in terms of mathematical modeling, there came SA-related questions that required a new way of thinking. In the 1980s and 90s, SA formally began to take place. The SA-experts believe that it is already growing to be an independent and interdisciplinary area of science [23].

3.1 Sensitivity analysis

Sensitivity analysis (SA) is defined as *the study of how the uncertainty in the outputs of a model can be apportioned to the different sources of uncertainty in the model inputs*, as stated by the Italian professor and expert on the topic, Andrea Saltelli [28]. It should not be confused with the uncertainty analysis (UA), which rather seeks to quantify the uncertainty in the model output, given the uncertain inputs. Ideally, UA and SA are used together, with UA as the initial step. There are two major strategies within the area of SA: (i) The local sensitivity analysis (LSA), and (ii) the global sensitivity analysis (GSA). Methods of the LSA are typically based on the partial derivatives of the outputs with respect to the inputs, that is, within a local region around the nominal point. The details are discussed in Section 3.1.1. Methods of the GSA, however, are based on the apportioning of the ouputs' uncertainty to the inputs' uncertainty, with the use of probability distributions for the inputs' range [27]. The details are discussed in Section 3.1.3.

3.1.1 Local sensitivity analysis

In literature, it occurs quite frequently that the sensitivities are defined based on the derivatives. The partial derivative, $\partial Y_{\mathrm{j}}/\partial X_{\mathrm{i}}$, can indeed be thought of as the definition of the sensitivity of the outputs, Y_{j}, with respect to the inputs, X_{i} [27]. This approach has the appealing benefit of computational efficiency, due to its small amount of required executions. Hence, the sensitivity, $S^{\mathrm{p}}_{X_{\mathrm{i}}}$, where the index p denotes "partial derivative", can be defined as [27]

$$S^{\mathrm{p}}_{X_{\mathrm{i}}} = \frac{\partial Y_{\mathrm{j}}}{\partial X_{\mathrm{i}}}, \tag{3.1}$$

where the model outputs are denoted by Y_{j}, in which $\mathrm{j} = 1, \ldots, \mathrm{N_Y}$, and the model inputs are denoted by X_{i}, in which $\mathrm{i} = 1, \ldots, \mathrm{N}_X$. This derivative-based formulation can be problematic, as the uncertainties in the inputs are not considered. One method of improving the sensitivities, $S^{\mathrm{p}}_{X_{\mathrm{i}}}$, is to introduce the sigma-normalized derivatives, $S^{\sigma}_{X_{\mathrm{i}}}$, that can be defined as [27]

$$S^{\sigma}_{X_{\mathrm{i}}} = \frac{\sigma_{X_{\mathrm{i}}} \partial Y_{\mathrm{j}}}{\sigma_{Y_{\mathrm{j}}} \partial X_{\mathrm{i}}}, \tag{3.2}$$

in which $\sigma_{X_{\mathrm{i}}}$ and $\sigma_{Y_{\mathrm{j}}}$ are the standard deviations for the inputs X_{i} and outputs Y_{j}, respectively. This formulation performs well for linear systems. However, the derivative-based sensitivities are unwarranted when the model inputs have uncertainty and when the model is nonlinear [27]. Thus, as the case study in Chapter 4 is nonlinear, this thesis focuses on the GSA.

3.1.2 Settings for sensitivity analysis

When using methods of GSA, it is crucial to know the correct *setting* beforehand. In literature, there are examples where several SA methods have been used for the same problem, and with an unstructured manner [27]. This can give confusing results, as the methods can rank the input factors after importance differently. It can be difficult to really know which of the answers that is the true answer. To solve this issue, *setting* can be used as a way of framing the SA objective, such that the answers become trustworthy. To select the best suited setting, carefully consider: (i) the outputs of interest, and (ii) the concept of 'importance'. Some common settings are [28]:

- Factor Prioritization (FP) - this setting is used to identify input factors, that when fixed at their respective true values, yield the largest reductions of the output variance. Hence, these identified input factors are those which account the most for the output variance.
- Factor Fixing (FF) - this setting is used to identify input factors, that when varying freely within the uncertainty bounds, account little for the output variance. The identified input factors can be fixed at any value within the boundaries, not affecting the output variance.
- Variance Cutting (VC) - this setting is used for reducing the output variance below some given tolerance. In the reliability analysis (RA), this may be the most desired setting.
- Factor Mapping (FM) - this setting is used to identify input factors that produce model realizations within some given range of the output space. That is, for an industrial plant it may appeal to highlight model realizations that give outputs over the 95th percentile, as it could correspond to unsafe operation in terms of the input factors.

Out of these four common settings, the first three are susceptible to the variance-based SA, which is an approach of the GSA that has great utility. More of this is in the upcoming section.

3.1.3 Global sensitivity analysis

The various methods of the GSA are carried out by the apportioning of the ouputs' uncertainty to the inputs' uncertainty, with the use of probability distributions for the inputs' whole range. The ranges are an important aspect, as they portray the knowledge one has or might be lacking, with regards to the model and its parameterization [27]. Generally speaking, there are four main categories of the GSA methods [37]: (i) variance-based methods, (ii) screening-based methods, (iii) regression-based methods, and (iv) metamodel-based methods. In the scope of this thesis, the focus is on the first two categories. However, as the variance-based SA offer great utility and its theory trivializes the comparison with the screening-based SA, it is explained in detail.

The variance-based SA methods utilize variance as the basis to quantify the influence of the inputs on the overall output variance. This choice seems natural, due to the variance being a measure of the dispersion, or the variability, in the model output. This can indicate the model precision due to the input variations. Consider the generic model [27],

$$Y = f(X_1, X_2, \ldots, X_\mathrm{k}), \tag{3.3}$$

where f is the model function, Y is the model output, and $X_1, X_2, \ldots, X_\mathrm{k}$ are the input factors in which $\mathrm{i} = 1, \ldots, \mathrm{k}$. Each of the X_i has non-null bounded variation. If X_i is fixed at a value, x_i^*, and $V_{\mathbf{X}_{\sim\mathrm{i}}}(Y|X_\mathrm{i} = x_\mathrm{i}^*)$ is the variance of Y over the input factors $\mathbf{X}_{\sim\mathrm{i}}$ (i.e., all input factors but the X_i), then this is called the *conditional variance* [27]. However, the dependency on x_i^* disappears if the conditional variance is averaged by all the possible points x_i^*, meaning that $E_{X_\mathrm{i}}(V_{\mathbf{X}_{\sim\mathrm{i}}}(Y|X_\mathrm{i}))$, which is always lower or equal to $V(Y)$. In fact, we have the equality [27]:

$$E_{X_\mathrm{i}}(V_{\mathbf{X}_{\sim\mathrm{i}}}(Y|X_\mathrm{i})) + V_{X_\mathrm{i}}(E_{\mathbf{X}_{\sim\mathrm{i}}}(Y|X_\mathrm{i})) = V(Y). \tag{3.4}$$

In eq. (3.4), a low $E_{X_\mathrm{i}}(V_{\mathbf{X}_{\sim\mathrm{i}}}(Y|X_\mathrm{i}))$, or a high $V_{X_\mathrm{i}}(E_{\mathbf{X}_{\sim\mathrm{i}}}(Y|X_\mathrm{i}))$, indicates that X_i is a factor of high importance [27]. Another name for the conditional variance, $V_{X_\mathrm{i}}(E_{\mathbf{X}_{\sim\mathrm{i}}}(Y|X_\mathrm{i}))$, is the *first-order effect of* X_i *on* Y. Thus, the *first-order sensitivity index of* X_i *on* Y,

$$S_\mathrm{i} = \frac{V_{X_\mathrm{i}}(E_{\mathbf{X}_{\sim\mathrm{i}}}(Y|X_\mathrm{i}))}{V(Y)}, \tag{3.5}$$

is a sensitivity measure of X_i on Y, where $S_\mathrm{i} \in [0, 1]$. Now, if there are two factors X_i and X_j considered for the conditional variance instead of only X_i, then it can be reformulated as [27]

$$\frac{V(E(Y|X_\mathrm{i}, X_\mathrm{j}))}{V(Y)}, \tag{3.6}$$

in which $\mathrm{i} \neq \mathrm{j}$, and the indices on E and V are left out. The following equation holds [27]:

$$V(E(Y|X_\mathrm{i}, X_\mathrm{j})) = V_\mathrm{i} + V_\mathrm{j} + V_\mathrm{ij}. \tag{3.7}$$

Here, the terms V_i, V_j and V_ij are written as [27]

$$V_\mathrm{i} = V(E(Y|X_\mathrm{i})) \tag{3.8a}$$

$$V_\mathrm{j} = V(E(Y|X_\mathrm{j})) \tag{3.8b}$$

$$V_\mathrm{ij} = V(E(Y|X_\mathrm{i}, X_\mathrm{j})) - V_\mathrm{i} - V_\mathrm{j}. \tag{3.8c}$$

The interactions between the factors X_i and X_j are represented through V_ij. For example, the non-linear and additive model, $Y = \sum_\mathrm{i} X_\mathrm{i}^2$, leads to none V_ij terms, but the non-linear and non-additive model, $Y = \prod_\mathrm{i} X_\mathrm{i}$, leads to non-zero V_ij terms. Thus, we are able to recover all of the variance $V(Y)$, even for non-additive models. Given enough patience to compute all the interaction terms, we can fully understand the sensitivities of the model. Given k input factors, the full analysis of the model can be formulated as [27]

$$\sum_\mathrm{i} S_\mathrm{i} + \sum_\mathrm{i}\sum_\mathrm{j>i} S_\mathrm{ij} + \sum_\mathrm{i}\sum_\mathrm{j>i}\sum_\mathrm{l>j} S_\mathrm{ijl} + \cdots + S_\mathrm{123\ldots k} = 1. \tag{3.9}$$

Recall the (3.4), which can be reformulated in terms of the *total-order effect of* X_i *on* Y. The total-order effect accounts for the influence of the factor X_i to the overall output variation (i.e., the first-order effect), plus all higher-order effects due to interaction. Thus, the *total-order sensitivity index of* X_i *on* Y can be defined with [27]

$$S_{T\mathrm{i}} = \frac{E(V(Y|\mathbf{X}_{\sim\mathrm{i}}))}{V(Y)} = 1 - \frac{V(E(Y|\mathbf{X}_{\sim\mathrm{i}}))}{V(Y)}. \tag{3.10}$$

Thus far, all the input factors have been assumed independent of each other. The reason is quite simple; dependent input factor samples are more tedious to generate, and their required sample size is much greater [27]. Going forward, only uncorrelated samples are considered.

3.2 Sobol' method

One such method of variance-based SA is the Sobol' method, proposed by Ilya M. Sobol' [32]. Consider the square-integrable function f over Ω_k, that is, the k-dimensional unit hypercube,

$$\Omega^\mathrm{k} = (X \,|\, 0 \leqslant x_\mathrm{i} \leqslant 1 \qquad \forall\, \mathrm{i} = 1, \ldots, \mathrm{k}). \tag{3.11}$$

The Sobol' method considers an expansion of f into terms with increasing dimensions [27],

$$f = f_0 + \sum_\mathrm{i} f_\mathrm{i} + \sum_\mathrm{i}\sum_\mathrm{j>i} f_\mathrm{ij} + \cdots + f_\mathrm{12\ldots k}. \tag{3.12}$$

All of these terms are square-integrable over their domain, and these are only functions of the factors in their indices, i.e., $f_\mathrm{i} = f_\mathrm{i}(X_\mathrm{i})$ and $f_\mathrm{ij} = f_\mathrm{i}(X_\mathrm{i}, X_\mathrm{j})$, and so forth. However, it is not a series decomposition, as the number of terms is finite. The number of terms equals 2^k, where one term is constant (f_0), there are k number of first-order functions (f_i), there are $\binom{\mathrm{k}}{2}$ number of second-order functions (f_ij), and so forth. This expansion is not unique, such that for the function, f, the terms have an infinite number of choices. However, given that each of the terms has the mean of zero, i.e., $\int f(x_\mathrm{i})\mathrm{d}x_\mathrm{i} = 0$, Sobol' proved that these terms must be orthogonal in pairs, i.e., $\int f(x_\mathrm{i})f(x_\mathrm{j})\mathrm{d}x_\mathrm{i}\mathrm{d}x_\mathrm{j} = 0$. Thus, the terms can be calculated univocally through the conditional expectation of the output Y. That is, f_0, f_i, and f_ij can be found by [27]

$$f_0 = E(Y) \tag{3.13a}$$
$$f_\mathrm{i} = E(Y|X_\mathrm{i}) - E(Y) \tag{3.13b}$$
$$f_\mathrm{ij} = E(Y|X_\mathrm{i}, X_\mathrm{j}) - f_\mathrm{i} - f_\mathrm{j} - E(Y). \tag{3.13c}$$

The conditional expectation $E(Y|X_{\mathrm{i}})$ is found by slicing the X_{i} domain and averaging the $Y|X_{\mathrm{i}}$ values within each slice. The variance of the conditional expectation, i.e., $V(E(Y|X_{\mathrm{i}}))$, can be considered as a sensitivity measure. In fact, $V(f_{\mathrm{i}}(X_{\mathrm{i}}))$ corresponds with $V(E(Y|X_{\mathrm{i}}))$. Thus, dividing by the unconditional variance, $V(Y)$, gives the first-order sensitivity index[27],

$$S_{\mathrm{i}} = \frac{V(E(Y|X_{\mathrm{i}}))}{V(Y)}. \tag{3.14}$$

This first-order sensitivity index represents the main effect contribution of the input factors to the variance of the output. However, the interactions between the input factors have not been accounted for yet. Two input factors, X_{i} and X_{j}, interact if their effect on the output, Y, is not equal to the sum of their respective effects. Decomposition of the eq. (3.13), gives that[27]

$$V_{\mathrm{i}} = V(f_{\mathrm{i}}(X_{\mathrm{i}})) = V(E(Y|X_{\mathrm{i}})) \tag{3.15a}$$

$$V_{\mathrm{ij}} = V(f_{\mathrm{ij}}(X_{\mathrm{i}}, X_{\mathrm{j}})) = V(E(Y|X_{\mathrm{i}}, X_{\mathrm{j}})) - V(E(Y|X_{\mathrm{i}})) - V(E(Y|X_{\mathrm{j}})). \tag{3.15b}$$

The joint effect of the input factor pair, X_{i} and X_{j}, is represented with $V(E(Y|X_{\mathrm{i}}, X_{\mathrm{j}}))$, where $V(f_{\mathrm{ij}}(X_{\mathrm{i}}, X_{\mathrm{j}}))$ equals this joint effect minus the first-order effects for the input factors, that is, $V(E(Y|X_{\mathrm{i}}))$ and $V(E(Y|X_{\mathrm{j}}))$. Here, $V(f_{\mathrm{ij}}(X_{\mathrm{i}}, X_{\mathrm{j}}))$ is commonly referred to as the second-order effect. Similarly, higher-order effects can be addressed using the same approach. For simplicity, we write $V(f_{\mathrm{i}}) = V_{\mathrm{i}}$ and $V(f_{\mathrm{ij}}) = V_{\mathrm{ij}}$, and so forth. Square integrating each of these terms over the k-dimensional unit hypercube, Ω^{k}, we obtain the decomposition[27]

$$V(Y) = \sum_{\mathrm{i}} V_{\mathrm{i}} + \sum_{\mathrm{i}} \sum_{\mathrm{j>i}} V_{\mathrm{ij}} + \cdots + V_{12\ldots\mathrm{k}}. \tag{3.16}$$

Dividing both sides of the eq. (3.16) by the variance, $V(Y)$, gives that[27]

$$\sum_{\mathrm{i}} S_{\mathrm{i}} + \sum_{\mathrm{i}} \sum_{\mathrm{j>i}} \sum_{\mathrm{l>j}} S_{\mathrm{ijl}} + \cdots + S_{123\ldots\mathrm{k}} = 1, \tag{3.17}$$

which equals the eq. (3.9). The total-order effect results from Sobol' variance decomposition, and is defined as the first-order effect plus all the other higher-order effects due to interaction. Another way of acquiring the total-order sensitivity index, is to consider the decomposition of the unconditional variance, $V(Y)$, such that[27],

$$V(Y) = V(E(Y|X_{\mathrm{i}})) + E(V(Y|X_{\mathrm{i}})), \tag{3.18}$$

and decomposing this once more, with respect to each of the input factors but the X_{i}[27],

$$V(Y) = V(E(Y|\mathbf{X}_{\sim\mathrm{i}})) + E(V(Y|\mathbf{X}_{\sim\mathrm{i}})). \tag{3.19}$$

Here, the measure, $V(Y) - V(E(Y|\mathbf{X}_{\sim\mathrm{i}})) = E(V(Y|\mathbf{X}_{\sim\mathrm{i}}))$, represents the mean variance of the output, Y, that is left if the true values of $\mathbf{X}_{\sim\mathrm{i}}$ can be determined. Thus, dividing by the output variance, $V(Y)$, the total-order sensitivity index is defined as

$$S_{T\mathrm{i}} = \frac{E(V(Y|\mathbf{X}_{\sim\mathrm{i}}))}{V(Y)} = 1 - \frac{V(E(Y|\mathbf{X}_{\sim\mathrm{i}}))}{V(Y)}, \tag{3.20}$$

which is the same as in the eq. (3.10). However, this computation is tedious without a shortcut. In order to acquire $V(E(Y|X_{\mathrm{i}}))$, there are typically required around 1000 Monte Carlo points

to get an estimate of $E(Y|X_\mathrm{i})$, and the further 1000 points to get an estimate of $V(E(Y|X_\mathrm{i}))$. Consequently, $1\mathrm{e}^6$ number of points are required only for the sensitivity of the input factor X_i. Monte Carlo points, in the context of Monte Carlo simulations, are randomly generated values of the input factors from a probability distribution, in which are used for estimating the output. The computation can be accelerated with Saltelli's modification, as seen in the next section[27].

3.2.1 Saltelli's modification

In Saltelli's modification, we generate the $\mathrm{N} \times 2\mathrm{k}$ matrix of random values, and allocate half of this sample to each of the $\mathrm{N} \times \mathrm{k}$ matrices, A and B. Here, k is the number of input factors, and N is the number of *base samples*, typically ranging from a few hundred to the thousands.

The matrix A can be formulated as[27]

$$A = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_\mathrm{i}^{(1)} & \cdots & x_\mathrm{k}^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_\mathrm{i}^{(2)} & \cdots & x_\mathrm{k}^{(2)} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ x_1^{(\mathrm{N}-1)} & x_2^{(\mathrm{N}-1)} & \cdots & x_\mathrm{i}^{(\mathrm{N}-1)} & \cdots & x_\mathrm{k}^{(\mathrm{N}-1)} \\ x_1^{(\mathrm{N})} & x_2^{(\mathrm{N})} & \cdots & x_\mathrm{i}^{(\mathrm{N})} & \cdots & x_\mathrm{k}^{(\mathrm{N})} \end{bmatrix}, \tag{3.21}$$

and the matrix B can be formulated as[27]

$$B = \begin{bmatrix} x_{\mathrm{k}+1}^{(1)} & x_{\mathrm{k}+2}^{(1)} & \cdots & x_{\mathrm{k}+\mathrm{i}}^{(1)} & \cdots & x_{2\mathrm{k}}^{(1)} \\ x_{\mathrm{k}+1}^{(2)} & x_{\mathrm{k}+2}^{(2)} & \cdots & x_{\mathrm{k}+\mathrm{i}}^{(2)} & \cdots & x_{2\mathrm{k}}^{(2)} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{\mathrm{k}+1}^{(\mathrm{N}-1)} & x_{\mathrm{k}+2}^{(\mathrm{N}-1)} & \cdots & x_{\mathrm{k}+\mathrm{i}}^{(\mathrm{N}-1)} & \cdots & x_{2\mathrm{k}}^{(\mathrm{N}-1)} \\ x_{\mathrm{k}+1}^{(\mathrm{N})} & x_{\mathrm{k}+2}^{(\mathrm{N})} & \cdots & x_{\mathrm{k}+\mathrm{i}}^{(\mathrm{N})} & \cdots & x_{2\mathrm{k}}^{(\mathrm{N})} \end{bmatrix}. \tag{3.22}$$

A new $\mathrm{N} \times \mathrm{k}$ matrix, C_i, takes all columns of B except the i'th, which is taken from A[27],

$$C_\mathrm{i} = \begin{bmatrix} x_{\mathrm{k}+1}^{(1)} & x_{\mathrm{k}+2}^{(1)} & \cdots & x_\mathrm{i}^{(1)} & \cdots & x_{2\mathrm{k}}^{(1)} \\ x_{\mathrm{k}+1}^{(2)} & x_{\mathrm{k}+2}^{(2)} & \cdots & x_\mathrm{i}^{(2)} & \cdots & x_{2\mathrm{k}}^{(2)} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{\mathrm{k}+1}^{(\mathrm{N}-1)} & x_{\mathrm{k}+2}^{(\mathrm{N}-1)} & \cdots & x_\mathrm{i}^{(\mathrm{N}-1)} & \cdots & x_{2\mathrm{k}}^{(\mathrm{N}-1)} \\ x_{\mathrm{k}+1}^{(\mathrm{N})} & x_{\mathrm{k}+2}^{(\mathrm{N})} & \cdots & x_\mathrm{i}^{(\mathrm{N})} & \cdots & x_{2\mathrm{k}}^{(\mathrm{N})} \end{bmatrix}. \tag{3.23}$$

The sampling matrices, A, B and C, are used for computing the model output with respect to all the samples of the input factors. This gives three $\mathrm{N} \times 1$ vectors of the model output[27]

$$y_A = f(A), \qquad y_B = f(B), \qquad y_{C_\mathrm{i}} = f(C_\mathrm{i}), \tag{3.24}$$

where y_A, y_B and y_C are the model output vectors for the matrices A, B and C, respectively. Assuming that these vectors are everything needed for obtaining the first-order and total-order sensitivity indices, S_i and S_{Ti}, these indices can be calculated from[27]

$$S_\mathrm{i} = \frac{V(E(Y|X_\mathrm{i}))}{V(Y)} = \frac{y_A \cdot y_{C_\mathrm{i}} - f_0^2}{y_A \cdot y_A - f_0^2} = \frac{(1/\mathrm{N}) \sum_{\mathrm{j}=1}^{\mathrm{N}} y_A^{(\mathrm{j})} y_{C_\mathrm{i}}^{(\mathrm{j})} - f_0^2}{(1/\mathrm{N}) \sum_{\mathrm{j}=1}^{\mathrm{N}} y_A^{(\mathrm{j})} y_A^{(\mathrm{j})} - f_0^2}, \tag{3.25}$$

and

$$S_{T\mathrm{i}} = 1 - \frac{V(E(Y|\mathbf{X}_{\sim\mathrm{i}}))}{V(Y)} = 1 - \frac{y_B \cdot y_{C_\mathrm{i}} - f_0^2}{y_A \cdot y_A - f_0^2} = 1 - \frac{(1/\mathrm{N})\sum_{\mathrm{j}=1}^{\mathrm{N}} y_B^{(\mathrm{j})} y_{C_\mathrm{i}}^{(\mathrm{j})} - f_0^2}{(1/\mathrm{N})\sum_{\mathrm{j}=1}^{\mathrm{N}} y_A^{(\mathrm{j})} y_A^{(\mathrm{j})} - f_0^2}, \quad (3.26)$$

in which the mean, f_0, is calculated from

$$f_0^2 = \left(\frac{1}{\mathrm{N}} \sum_{\mathrm{j}=1}^{\mathrm{N}} y_A^{(\mathrm{j})}\right)^2. \quad (3.27)$$

Here, the scalar product of two vectors is symbolized with $(\cdot)$. The computational costs for Saltelli's modification are just $\mathrm{N}(\mathrm{k}+2)$, compared to the original cost of N^2 model evaluations. In the scalar product, $y_A \cdot y_{C\mathrm{i}}$, the model outputs from A are multiplied by the model outputs from B, in which all input factors but X_i are resampled. If the input factor X_i is non-influential, low and high values of y_A and $y_{C\mathrm{i}}$ are randomly associated. If the input factor X_i is influential, then high values of y_A are multiplied by high values of $y_{C\mathrm{i}}$, or, low values of y_A are multiplied by low values of $y_{C\mathrm{i}}$. This strategy increases the values of the scalar products [27].

It is by design, that the values of S_{Ti} are always greater or equal to the values of S_i. Hence, $S_{\mathrm{Ti}} - S_\mathrm{i}$ can be seen as a measure of how much interactions the input factor, X_i, has with the other input factors. If $S_{\mathrm{Ti}} = 0$, then this indicates that the input factor, X_i, is non-influential. Moreover, $\sum_\mathrm{i} S_\mathrm{i} = 1$ for additive models, $\sum_\mathrm{i} S_\mathrm{i} < 1$ for non-additive models, and $1 - \sum_\mathrm{i} S_\mathrm{i}$ can indicate how much interactions there are in the model. It is always true that $\sum_\mathrm{i} S_{\mathrm{Ti}} \geqslant 1$, but for additive models we have$\sum_\mathrm{i} S_{\mathrm{Ti}} = 1$ [27].

In Section 3.1.2, there were presented various *settings* in the context of sensitivity analysis. How do these first-order and total-order sensitivity indices, S_i, and S_{Ti}, relate to these settings? We could ask, *which of the input factors deserve a further analysis?* in which S_i can be used as an answer, and is related to the FP setting. Or we could ask, *which of the input factors can be fixed or simplified*? in which S_{Ti} can be used as an answer, and is related to the FF setting. It is important to consider these questions beforehand, such that the most fitting index is used, and in our case, that is in the switch-rule for the MS-MPC.

There are numerous available methods for generating the random $\mathrm{N} \times 2\mathrm{k}$ sampling matrix. The default method is the *random sampling*, where the values of the input factors are randomly and independently sampled from a distribution. As the method relies purely on the randomness, it can be inefficient if some spaces of the cumulative distribution function (CDF) are left empty, and other spaces are quite clustered. Another approach is the *Latin Hypercube sampling* (LHS), that spreads out the sample points more evenly across the inverse CDF. It is commonly used in Monte Carlo simulations as an alternative to the random sampling. The main idea is that more evenly distributed samples might result in fewer required samples for the sensitivity indices. Another approach is the *Morris sampling*, which is explained in more detail in the next section on the Morris screening. It is a one-at-a-time (OAT) sampling method, where the space of the input factors is discretized into grids in which they are systematically sampled. In Figure 3.1, two input factors, X_1 and X_2, are randomly sampled within the interval $[0, 1]$ using 500 points, each for the random sampling, Latin Hypercube sampling and Morris sampling, respectively. The script for producing these plots is shown in Appendix - code listings under 'xplots.py' [27].

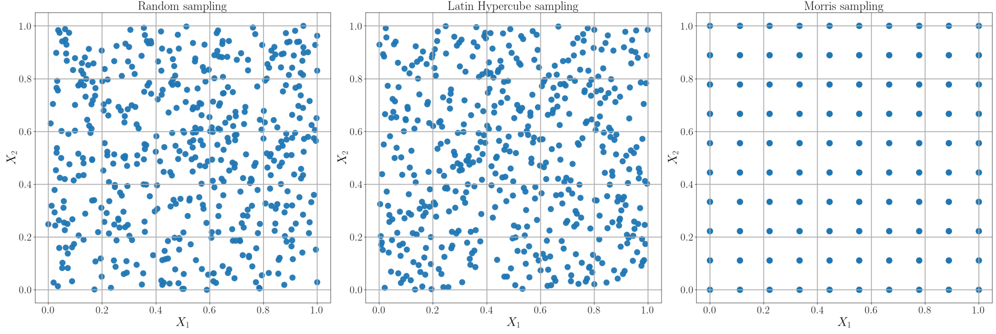


Figure 3.1: Random sampling, LH sampling and Morris sampling of the input factors, X_1 and X_2.

3.3 Morris screening

Another widely-used method of the GSA is the Morris screening. It uses a one-at-a-time (OAT) approach in order to identify the sensitivities. Here, the space of the input factors is discretized into grids in which they are systematically sampled. Only one input factor is permitted to vary over the grid-space at a time, while all the other input factors are fixed at their nominal values. In general, screening methods require fewer model evaluations than variance-based methods, and should be used prior to these methods such that non-influential input factors can be left out for further analysis. The Morris screening only provides semi-quantitative information on the sensitivity measures, and it struggles with identifying which of the input factors that experience non-linearity and interactions. The early ranking of the input factors is still very useful [20].

Consider the same model as in eq. (3.3), where Y is the model output, and X_i are the input factors in which $\mathrm{i} = 1, \ldots, \mathrm{k}$. Let the input factors vary within the k-dimensional unit cube across the p number of levels. The value of p is usually an even number, e.g., $\mathrm{p} = 4$ is used in this thesis' case study. Hence, the grid is denoted Ω, and is discretized into p number of levels. Given the values of the input factors, $\mathbf{X} = (X_1, X_2, \ldots, X_\mathrm{k})$, the *elementary effect* of the i'th input factor can be formulated as [27]

$$EE_\mathrm{i} = \frac{[Y(X_1, X_2, \ldots, X_\mathrm{i} + \Delta, \ldots, X_\mathrm{k}) - Y(X_1, X_2, \ldots, X_\mathrm{i}, \ldots, X_\mathrm{k})]}{\Delta}, \tag{3.28}$$

in which Δ equals one of the values, $\{1/(\mathrm{p}-1), \ldots, 1-1/(\mathrm{p}-1)\}$. These elementary effects, EE_i, are used for further computing the sensitivity measures of the screening-based methods. They have similar nature to the derivative-based sensitivities in the LSA, as these EE_i can be viewed as an extension to the eq. (3.1). The increment, or the step-size, Δ, is typically selected equal to $\mathrm{p}/(2(\mathrm{p}-1))$. Morris proposed the sensitivity measures, μ and σ, as the estimates of the mean and standard deviation of the distribution of the elementary effects, respectively [17]. The mean, μ, addresses the input factors' influence on the output. The standard deviation, σ, addresses the factors' effects ensemble, due to non-linearity or the interactions with each other. Campolongo proposed an alternative estimated mean, $\mu*$, which instead estimates the mean of the distribution of the elementary effects' absolute values [2]. It prevents possible type-II errors, that might occur if the elementary effects are negative and positive. This can possibly lead to important input factors not being fully accounted for, as the effects might cancel each other [27].

In general, it is recommended that all three of the measures, μ, μ^* and σ, are used together in order to get the most information on the sensitivities. Furthermore, μ^* can be viewed as an approximation to the total-order sensitivity index, S_{Ti}, shown in eq. (3.20). The measures have the common objective of identifying non-influential input factors, in which μ^* comes with less computational costs than S_{Ti}. Computing both measures μ and μ^* does not increase the costs, and comparing these two yields useful information about the effects' signs, whether they have any effect or not. In short, (i) a high value of μ_{i}^* indicates that the input factor X_{i} is influential on the output Y, (ii) a high value of σ_{i} indicates that the input factor X_{i} has interactions with the other input factors, and (iii) comparing μ_{i} and μ_{i}^* indicates whether the elementary effect's sign have any significance [27]. Now follows the sampling strategy for this three measures.

The full description of the Morris sampling is given by M. D. Morris [17] and A. Saltelli [27], whereas the following explanation focuses on the matrix calculations. Consider the $(\mathrm{k}+1)\times \mathrm{k}$ dimensional matrix $\mathbf{B}^*$, in which its rows consist of the inputs' effect vectors, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{\mathrm{k}}$. Here, the matrix $\mathbf{B}^*$ is used to describe one out of the r number of trajectories, that consists of $(\mathrm{k}+1)$ number of points in which each of the points provides k number of elementary effects. The index, k, still represents the number of input factors. In order to calculate the matrix $\mathbf{B}^*$, the initial step is to construct the $(\mathrm{k}+1)\times \mathrm{k}$ dimensional matrix, $\mathbf{B}$, made of only 0's and 1's. In fact, $\mathbf{B}$ is a lower triangular matrix with only 1's below the diagonal [27],

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1 & 0 & \cdots & 0 \\ 1 & 1 & 1 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix}. \tag{3.29}$$

Furthermore, the randomized sampling matrix $\mathbf{B}^*$ can be formulated as [27]

$$\mathbf{B}^* = (\mathbf{J}_{\mathrm{k}+1,1}\mathbf{x}^* + (\Delta/2)[(2\mathbf{B} - \mathbf{J}_{\mathrm{k}+1,\mathrm{k}})\mathbf{D}^* + \mathbf{J}_{\mathrm{k}+1,\mathrm{k}}])\mathbf{P}^*, \tag{3.30}$$

in which $\mathbf{J}_{\mathrm{k}+1,\mathrm{k}}$ is a $(\mathrm{k}+1)\times \mathrm{k}$ dimensional matrix of only 1's, and $\mathbf{x}^*$ is a vector of the random base values of $\mathbf{x} = \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{\mathrm{k}}$. Moreover, $\mathbf{D}^*$ is a k-dimensional diagonal matrix in which the elements are either $+1$ or -1 with the same probability, and $\mathbf{P}$ is a $\mathrm{k}\times\mathrm{k}$ random permutation matrix in which the rows contain one element equal to 1, while the other elements are equal to 0. None of the columns in $\mathbf{P}$ are allowed elements equal to 1 in the same position. The purpose of $\mathbf{P}$ is to supply the order in which the input factors are moved, and $\mathbf{D}^*$ states if the input factors are increased or decreased over the trajectory. Hence, the sampling matrix $\mathbf{B}^*$ provides k number of elementary effects, and this is repeated over the r number of trajectories within the grid Ω, in order to get the sensitivity measures, μ, μ^* and σ [27].

Consider the two random sampling points for the j'th trajectory, $\mathbf{x}_{\mathrm{l,j}}$ and $\mathbf{x}_{\mathrm{l+1,j}}$, in which $\mathrm{l} = 1, \ldots, \mathrm{k}$ and $\mathrm{j} = 1, \ldots, \mathrm{r}$. The elementary effect of the input factor, X_{i}, where $\mathrm{i} = 1, \ldots, \mathrm{k}$, can thus be formulated as [27]

$$EE_{\mathrm{i,j}}(\mathbf{x}_{\mathrm{l,j}}) = \frac{f(\mathbf{x}_{\mathrm{l+1,j}}) - f(\mathbf{x}_{\mathrm{l,j}})}{\Delta}, \tag{3.31}$$

given that there is an increase in the i'th component of $\mathbf{x}_{\mathrm{l,j}}$ with Δ, or

$$EE_{\mathrm{i,j}}(\mathbf{x}_{\mathrm{l+1,j}}) = \frac{f(\mathbf{x}_{\mathrm{l,j}}) - f(\mathbf{x}_{\mathrm{l+1,j}})}{\Delta}, \tag{3.32}$$

given that there is a decrease in the i'th component of $\mathbf{x}_{\mathrm{l,j}}$ with Δ. When all the $\mathrm{k} \cdot \mathrm{r}$ number of elementary effect are calculated, then the sensitivity measures, μ_{i}, μ_{i}^{*} and σ_{i}, can be computed with respect to the input factors' distribution. These measures can be formulated as [27]

$$\mu_{\mathrm{i}} = \frac{1}{\mathrm{r}} \sum_{\mathrm{j}=1}^{\mathrm{r}} EE_{\mathrm{i,j}}, \tag{3.33a}$$

$$\mu_{\mathrm{i}}^{*} = \frac{1}{\mathrm{r}} \sum_{\mathrm{j}=1}^{\mathrm{r}} |EE_{\mathrm{i,j}}|, \tag{3.33b}$$

$$\sigma_{\mathrm{i}}^{2} = \frac{1}{\mathrm{r}-1} \sum_{\mathrm{j}=1}^{\mathrm{r}} (EE_{\mathrm{i,j}} - \mu_{\mathrm{i}})^{2}, \tag{3.33c}$$

in which $EE_{\mathrm{i,j}}$ represents the elementary effects of the input factor X_{i} over the j'th trajectory. How do these sensitivity measures relate to the settings presented in Section 3.1.2? In general, μ^{*} is the most relevant measure with regard to the settings. It can be seen as an approximation of the total-order sensitivity index, S_{Ti}, and is therefore fitted to the FF setting. One could ask, *which input factors can be fixed or simplified*? and μ^{*} can provide semi-quantitative answers. The measures μ and σ do not relate better to the setting than μ^{*}, and are only used to acquire additional information on the input factors [27]. Thus, μ^{*} is the only sensitivity measure applied in the switch-rule for the MS-MPC.

3.3.1 Modified Morris screening

An alternative SA strategy to the Morris screening is the so-called *Modified Morris screening*, proposed by van Griensven [5]. The method combines the features of an OAT approach with an improved sampling method, i.e., Latin Hypercube sampling, thus giving the LH-OAT strategy. The LSA methodology is directly transformed into an approach with features of the GSA [7].

Consider now r number of Latin Hypercube samples in the space taken of the input factors, $X_1, X_2, \ldots, X_{\mathrm{k}}$, in which $\mathrm{i} = 1, \ldots, \mathrm{k}$ and $\mathrm{j} = 1, \ldots, \mathrm{r}$. All the j'th sample points are varied one-at-a-time with respect to the k number of input factors, resulting in r number of trajectories. Within all these trajectories, the so-called *partial effects* can be used as the sensitivity measure, similarly to the elementary effects in the last section. The partial effect of the input factor, X_{i}, at the j'th trajectory, is presented by $PE_{\mathrm{i,j}}$. It can be formulated as [5]

$$PE_{\mathrm{i,j}} = \frac{100}{\Delta_{\mathrm{i}}} \frac{f(x_{1,\mathrm{j}}, \ldots, x_{\mathrm{i,j}}(1+\Delta_{\mathrm{i}}), \ldots, x_{\mathrm{k,j}}) - f(x_{1,\mathrm{j}}, \ldots, x_{\mathrm{i,j}}, \ldots, x_{\mathrm{k,j}})}{(f(x_{1,\mathrm{j}}, \ldots, x_{\mathrm{i,j}}(1+\Delta_{\mathrm{i}}), \ldots, x_{\mathrm{k,j}}) + f(x_{1,\mathrm{j}}, \ldots, x_{\mathrm{i,j}}, \ldots, x_{\mathrm{k,j}}))/2}, \tag{3.34}$$

in which f is the model function, $x_{\mathrm{i,j}}$ is the randomly generated sample of the input factor, X_{i}, of the j'th trajectory, and Δ_{i} represents the percentage-wise fraction that X_{i} is increased with. The sensitivity measures, μ_{i}, μ_{i}^{*} and σ_{i}, can be formulated equally as in the eq. (3.33) [5],

$$\mu_{\mathrm{i}} = \frac{1}{\mathrm{r}} \sum_{\mathrm{j}=1}^{\mathrm{r}} PE_{\mathrm{i,j}}, \tag{3.35a}$$

$$\mu_{\mathrm{i}}^{*} = \frac{1}{\mathrm{r}} \sum_{\mathrm{j}=1}^{\mathrm{r}} |PE_{\mathrm{i,j}}|, \tag{3.35b}$$

$$\sigma_{\mathrm{i}}^{2} = \frac{1}{\mathrm{r}-1} \sum_{\mathrm{j}=1}^{\mathrm{r}} (PE_{\mathrm{i,j}} - \mu_{\mathrm{i}})^{2}. \tag{3.35c}$$

However, van Griensven focused explicitly on the absolute mean of the partial effects, μ_{i}^{*}, in his study. It is not mentioned whether the mean μ_{i} or the standard deviation σ_{i} provide any additional information. In the results of this thesis' case study, only the μ^{*} and σ are used in the plots of the sensitivities, as for both the Morris screening and the Modified Morris screening. However, the σ is left out for these plots of the Sobol' method, as it requires bootstrapping [27]. Comparison of the μ and μ^{*} is only briefly discussed, and the σ is provided in the plots as an additional information on the interactions between the input factors. As the primary objective of the screening-based methods is to rank the input factors' influence, it makes sense to focus on the μ^{*}, and only this sensitivity measure is used in the switch-rule for the MS-MPC [5].

The calculation of the partial effects is efficient, combining features of the OAT approach with the LHS, which can give lower required samples. In short, it is required 9N, 8N and 8N number of model evaluations for the Sobol' method, the Morris screening and the Modified Morris screening, respectively, where N is the number of base samples. It is presumed that the Sobol' method requires even further computational costs, due to the expenses of inverting the CDF to the percent-point function (PPF) [6]. Thus, Modified Morris screening can be viewed as the most efficient SA of these three, with the possible drawback of not assessing the μ and σ as accurately as the Morris screening. Another disadvantage of the screening-based methods, is that they struggle with identifying which of the inputs factors that experience non-linearity and interactions. These can only relate to the FF-setting, whereas the variance-based methods can relate to both the FP and FF-setting, giving them more flexibility [27].

Eq. (3.34) was tested for this thesis' case study, but the partial effects did not turn out well, and shared nothing in common with the total-order sensitivity index and elementary effects. By trial-and-error, a new proposed definition of the partial effect, $PE_{\mathrm{i,j}}$, was found better,

$$PE_{\mathrm{i,j}} = \frac{f(x_{1,\mathrm{j}}, \ldots, x_{\mathrm{i,j}}(1+\Delta_{\mathrm{i}}), \ldots, x_{\mathrm{k,j}}) - f(x_{1,\mathrm{j}}, \ldots, x_{\mathrm{i,j}}, \ldots, x_{\mathrm{k,j}})}{\sqrt{\Delta_{\mathrm{i}}}}, \tag{3.36}$$

in which it has more similarities with the definition of the elementary effects in the eq. (3.28), but it is instead divided by $\sqrt{\Delta_{\mathrm{i}}}$; the squared fraction that the input factor X_i is increased with. This definition gives reasonable μ^{*}, but μ and σ are more difficult to interpret, as intuited by van Griensven's definition where the focus is on the μ^{*}. It is also worth mentioning that the model outputs were normalized with respect to the mean and standard deviation, for all of the GSA methods. This was done in order to prevent bias of the sensitivity measures [24].

3.4 Other GSA methods

There are many other GSA methods that have not been mentioned yet. An interesting example is the Monte Carlo filtering (MCF). Here, the objective is not to identify the input factors, X_i, that are the most influential to the model output, Y. It rather focuses on mapping the randomly sampled values of the input factors into the space of the output, then filtering out input factors corresponding to the space of unacceptable output values [27]. The sample values that give good realizations of the output can be flagged as 'behavioral', while the sample values that give bad realizations of the output can be flagged as 'non-behavioral'. If an input factor is flagged as non-behavioral, then it is considered important for the output in terms of its defined threshold. Say that we are interested in preventing financial loss (i.e., the output), and want to highlight the realizations and their corresponding input factors that give loss beneath the 95th percentile. In this case, MCF can be a fitted method of SA, and it is linked with the FM-setting [27].

Another related practice is the *first-order reliability method* (FORM), or the *second-order reliability method* (SORM). In fact, they are neither methods of LSA or GSA, as the magnitude of the output, Y, and its potential variation, are not the interest, but rather the probability of Y exceeding a critical value. Assume that we have some constraint, $Y - Y_{\mathrm{crit}} \leqslant 0$, which gives a *hypersurface* in the space, Ω, of the input factors, $\mathbf{X}$. The quantity of interest is the minimum distance between a design point, $\mathbf{X}$*, and the hypersurface. This distance is denoted β for a joint distribution of $\mathbf{X}$, where its derivative with respect to $\mathbf{X}$ defines the sensitivity measure. In short, FORM aims at identifying the points that result in the highest possibility of failure, in which failure means violation of the constraint. The difference between FORM and SORM is that the former can only manage linear constraints, while the latter can manage non-linear. They are alike in that they both use an optimization algorithm in order to identify points that most likely lead to failure, with respect to the input factors [25].

Neither of the MCF, FORM and SORM were applied in this thesis' case study, even though they would be very relevant. In the next section, this case study is presented.

Chapter 4

The case study

In 1928, while at the St. Mary's Hospital in London, the Scottish physician and microbiologist Alexander Fleming made an important scientific discovery. He had just returned from holiday, and was greeted with the sight of mold growing on his bacteria cultivated Petri dish. It seemed that the mold had inhibited the growth of the bacteria, in which it had produced some chemical that could kill bacteria. This marked the discovery of the antibacterial substance *penicillin*[36].

It was first in 1940 that Howard Florey, Ernst Chain and their colleagues at the Sir William Dunn School of Pathology at the Oxford University were able to produce purified penicillin. However, their production was inefficient and there were demands for an industrial upscaling. Florey brought the penicillin to the US, where by the mid-1940s, pharmaceutical companies such as Pfizer and Merck played an important role in industrializing the penicillin production through fermentation processes. This scientific breakthrough helped the treatment of countless bacterial infections, and saved many lives. In 1945, the Nobel Prize in Physiology or Medicine was awarded Fleming, Florey and Chain for their invaluable research on the penicillin[36].

4.1 The case study

The selected study is a fermentation process of penicillin in an isothermal fed-batch bioreactor. Generally speaking, bioreactors are either batch, fed-batch or continuous: (i) In a batch reactor, all of the reactants are added at the start, and the reaction proceeds without any further inputs. (ii) In a fed-batch reactor, not all of the reactants are added at the start, but some are added as the reaction proceeds. (iii) In a continuous reactor, the reactants are continuously added as the reaction proceeds, and the products are continuously being removed (instead of at the end)[41]. In the case of industrial fermentation processes, batch and fed-batch reactors are mostly used. However, as the fed-batch reactors offer greater control over the different stages of the process, it is the chosen reactor for this case study. The requirement of control is driven by the growth conditions of the biomass. At too large biomass concentrations, it may occur limitations on the oxygen supply, lowering the amount of product that is produced[33]. The main objective of the fermentation process is to produce as much penicillin as possible. Hence, there is no question that the biomass needs controlling, and this is accomplished by adjusting the substrate feed[11].

The case study was proposed by Srinivasan[33], and later adapted by Lucia[11] for his study on the MS-MPC. A fed-batch reactor is used instead of a continuous reactor, as it is expected that the fed-batch reactor differs more in what parameters are the most sensitive to the biomass. At the beginning of the process, there is much sugar for the cells to digest, but as the reaction goes on, more and more sugar is consumed, increasing the amount of biomass and penicillin. It is expected that one or two parameters are the most sensitive at the beginning of the process, but another two parameters at the end. This was confirmed in the specialization project in prior to this thesis, where the Sobol' method was used as the sensitivity analysis on the OL-MPC[15]. In short, this case study aims at improving the MS-MPC by the inclusion of SA-based switch.

A simplified flowsheet of the case study is presented in Figure 4.1. There are four states (X, S, P, V), one input (u) and seven parameters ($\mu_\mathrm{m}, K_\mathrm{m}, K_\mathrm{i}, \nu, Y_\mathrm{p}, Y_\mathrm{x}, S_\mathrm{in}$). The only flow into the reactor is the substrate feed, and there are not any outflows. The bioreactor is assumed perfectly mixed and isothermal. The main objective of the process is to maximize penicillin.

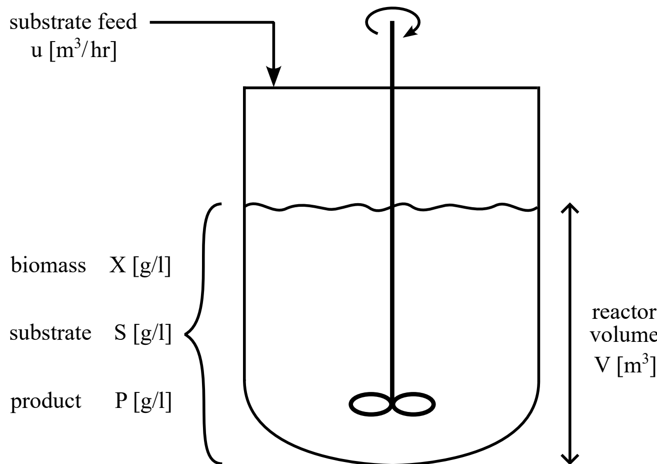


Figure 4.1: Simplified flowsheet of the penicillin production in a fed-batch reactor[11].

In short, there are main two reactions that happen in the bioreactor simultaneously,

$$S \xrightarrow{X} X, \qquad X \xrightarrow{X} P, \tag{4.1}$$

in which the substrate produces biomass and the biomass produces penicillin, with dependence on the amount of biomass in the reactor. There are also other dynamics that are not captured by the reaction, e.g., dependence on oxygen supply, which is neglected for the sake of simplicity. However, complex dynamics are captured through the ordinary differential equation (ODE)[11]:

$$\dot{X} = \mu(S)X - \frac{u}{V}X \tag{4.2a}$$

$$\dot{S} = -\frac{\mu(S)X}{Y_\mathrm{x}} - \frac{\nu X}{Y_\mathrm{p}} + \frac{u}{V}(S_\mathrm{in} - S) \tag{4.2b}$$

$$\dot{P} = \nu X - \frac{u}{V}P \tag{4.2c}$$

$$\dot{V} = u \tag{4.2d}$$

where

$$\mu(S) = \frac{\mu_\mathrm{m} S}{K_\mathrm{m} + S + (S^2/K_\mathrm{i})} \tag{4.2e}$$

Here, the four states, $X[\mathrm{g/l}]$, $S[\mathrm{g/l}]$, $P[\mathrm{g/l}]$ and $V[\mathrm{m}^3]$ define the biomass concentration, the substrate concentration, the product concentration, and the bioreactor volume, respectively. The only input, $u[\mathrm{m}^3/\mathrm{hr}]$, represents the flow rate of the substrate feed. The auxiliary term, $\mu[\mathrm{hr}^{-1}]$, represents the specific growth rate of the modified Monod equation[33]. Furthermore, the maximum specific growth rate is denoted $\mu_\mathrm{m}[\mathrm{hr}^{-1}]$, the saturation constant is $K_\mathrm{m}[\mathrm{g/l}]$, the inhibition constant is $K_\mathrm{i}[\mathrm{g/l}]$, the specific rate of the product formation is represented with $\nu[\mathrm{g(P)/(g(X)\cdot hr)}]$, the product yield is represented with $Y_\mathrm{p}[\mathrm{g(P)/g(S)}]$, the biomass yield is represented with $Y_\mathrm{x}[\mathrm{g(X)/g(S)}]$, and the inlet substrate concentration is defined as $S_\mathrm{in}[\mathrm{g/l}]$. The initial values for the states and the input, as well as the nominal values for the parameters, are presented in Table 4.1.

Table 4.1: Initial or nominal values for the states, inputs and parameters[11].

Symbol	Initial or nominal value	Unit
X_0	1.0	$[\mathrm{g/l}]$
S_0	0.5	$[\mathrm{g/l}]$
P_0	0.0	$[\mathrm{g/l}]$
V_0	120.0	$[\mathrm{m}^3]$
u_0	0.0	$[\mathrm{m}^3/\mathrm{hr}]$
μ_m	0.02	$[\mathrm{hr}^{-1}]$
K_m	0.05	$[\mathrm{g/l}]$
K_i	5.0	$[\mathrm{g/l}]$
ν	0.004	$[\mathrm{g(P)/(g(X)\cdot hr)}]$
Y_p	1.2	$[\mathrm{g(P)/g(S)}]$
Y_x	0.4	$[\mathrm{g(X)/g(S)}]$
S_in	200.0	$[\mathrm{g/l}]$

4.1.1 Model predictive control

Recall the eq. (2.3), which applied to this case study, can be formulated as

$$\min_{x_k, u_k} \sum_{k=0}^{N_p} x_k^T Q x_k + \sum_{k=1}^{N_m} \Delta u_k^T R \Delta u_k \tag{4.3a}$$

subject to

$$x_{k+1} = f(x_k, u_k, p_k), \qquad \forall\, k = 0, \dots, N_p - 1 \tag{4.3b}$$

$$g(x_k, u_k, p_k) \leqslant 0, \qquad \forall\, k = 1, \dots, N_p \tag{4.3c}$$

$$x_{min} \leqslant x_k \leqslant x_{max}, \qquad \forall\, k = 1, \dots, N_p \tag{4.3d}$$

$$u_{min} \leqslant u_k \leqslant u_{max}, \qquad \forall\, k = 1, \dots, N_m \tag{4.3e}$$

$$-\Delta u_{max} \leqslant \Delta u_k \leqslant \Delta u_{max}, \qquad \forall\, k = 1, \dots, N_m \tag{4.3f}$$

where

$$x_0 = \hat{x}_0, \tag{4.3g}$$

$$\Delta u_k = u_k - u_{k-1}, \qquad \forall\, k = 1, \dots, N_m \tag{4.3h}$$

$$\Delta u_k = 0, \qquad \forall\, k = N_m + 1, \dots, N_p \tag{4.3i}$$

where $\hat{x}_0 = [X_0, S_0, P_0, V_0]$, and the weighted matrices Q and R are defined as

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad R = \begin{bmatrix} 0 \end{bmatrix}. \tag{4.4}$$

The only cell in the matrix Q that is non-zero represents the importance of the product P. It is negative due to the optimization is being based on minimization of the objective function. As seen from the input movements penalization matrix R, the input movements are chosen not to be penalized in the objective function. These are rather included as hard constraints, as this seemed to yield better control of this particular system. All the hard constraints on the outputs, input and input movement are presented in Table 4.2.

Table 4.2: Lower and upper constraints the outputs, inputs and input changes [11].

Symbol	Lower constraint	Upper constraint	Unit
X	0.0	3.7	[g/l]
S	0.0	∞	[g/l]
P	0.0	3.0	[g/l]
V	0.0	∞	[m^3]
u	0.0	0.2	[m^3/hr]
Δu	-0.0035	0.0035	[m^3/hr]

Optimally, the constraint on the biomass X should be active, as this gives the most amount of the product P. Exceeding this constraint, or having an unnecessarily conservative back-off from the constraint, results in a lower concentration of penicillin at the end of the time-horizon. Furthermore, the prediction horizon, $\mathrm{N_p}$, and the control horizon, $\mathrm{N_m}$, were assigned different values from the open-loop implementation to the closed-loop. For the OL-MPC, it was given $\mathrm{N_p} = 150$ and $\mathrm{N_m} = 150$, while for the CL-MPC, these values were $\mathrm{N_p} = 20$ and $\mathrm{N_m} = 3$. In the orthogonal collocation, three Gauss-Radau roots were used for each finite element.

4.1.2 Sensitivity analysis

Out of all the sensitivity analysis methods presented in Chapter 4, this thesis was restricted to the implementation of the Sobol' method, Morris screening and Modified Morris screening. For this case study, the methods served the purpose of identifying the most sensitive parameters to the important constraint on the biomass, i.e., $X \leqslant 3.7$. Using the Latin Hypercube sampling for the Sobol' method and Modified Morris screening, and the one-at-a-time Morris sampling for the Morris screening, the samples were drawn quasi-randomly from a uniform distribution. The samples, θ_i, were taken from $U(85\%\, E[\theta_\mathrm{i}], E[\theta_\mathrm{i}], 115\%\, E[\theta_\mathrm{i}])$, in which U is the uniform parameter distribution. The computational expenses were identified as 9N, 8N and 8N for each of the Sobol' method, Morris screening and the Modified Morris screening, where N represents the number of base samples. In plots of the sensitivity indices, $\mathrm{N} = 2^{12}$ and $\mathrm{N} = 2^{15}$ was used, but in the online implementation of the MS-MPC with switch, $\mathrm{N} = 2^{10}$ and $\mathrm{N} = 2^{12}$ was rather used due to the greater computational time.

4.1.3 Multistage model predictive control

Recall the eq. (2.10), which applied to this case study, can be formulated as

$$\min_{x_{\mathrm{k,j}},u_{\mathrm{k,j}}} \sum_{\mathrm{i}=1}^{\mathrm{N_s}} \omega_\mathrm{i} \Big(\sum_{\mathrm{k}=0}^{\mathrm{N_p}} x_{\mathrm{k,j}}^{\mathrm{T}} \mathrm{Q} x_{\mathrm{k,j}} + \sum_{\mathrm{k}=1}^{\mathrm{N_m}} \Delta u_{\mathrm{k,j}}^{\mathrm{T}} \mathrm{R} \Delta u_{\mathrm{k,j}} \Big) \tag{4.5a}$$

subject to

$$x_{\mathrm{k+1,j}} = f(x_{\mathrm{k,p(j)}}, u_{\mathrm{k,j}}, p_{\mathrm{k,r(j)}}), \qquad \forall\,(\mathrm{k,j}) \in \mathrm{I} \tag{4.5b}$$

$$u_{\mathrm{k,i}} = u_{\mathrm{k,j}} \text{ if } x_{\mathrm{k,p(i)}} = x_{\mathrm{k,p(j)}}, \qquad \forall\,(\mathrm{k,j}),(\mathrm{k,i}) \in \mathrm{I} \tag{4.5c}$$

$$g(x_{\mathrm{k,p(j)}}, u_{\mathrm{k,j}}, p_{\mathrm{k,r(j)}}) \leqslant 0, \qquad \forall\,(\mathrm{k,j}) \in \mathrm{I} \tag{4.5d}$$

$$x_{\min} \leqslant x_{\mathrm{k,j}} \leqslant x_{\max}, \qquad \forall\,(\mathrm{k,j}) \in \mathrm{I} \tag{4.5e}$$

$$u_{\min} \leqslant u_{\mathrm{k,j}} \leqslant u_{\max}, \qquad \forall\,(\mathrm{k,j}) \in \mathrm{I} \tag{4.5f}$$

$$-\Delta u_{\max} \leqslant \Delta u_{\mathrm{k,j}} \leqslant \Delta u_{\max}, \qquad \forall\,(\mathrm{k,j}) \in \mathrm{I} \tag{4.5g}$$

where

$$x_{0,j} = \hat{x}_0, \qquad \forall\,(\mathrm{k,j}) \in \mathrm{I} \tag{4.5h}$$

$$\Delta u_{\mathrm{k,j}} = u_{\mathrm{k,j}} - u_{\mathrm{k-1,j}}, \qquad \forall\,(\mathrm{k,j}) \in \mathrm{I} \tag{4.5i}$$

$$\Delta u_{\mathrm{k,j}} = 0, \qquad \forall\,(\mathrm{k,j}) \in \mathrm{I} \tag{4.5j}$$

where $\hat{x}_0$, Q and R are defined equally as for the basic model predictive control, as well as the constraints on the outputs, inputs and input movements. The prediction horizon and the control horizon are equal to those of the CL-MPC, i.e., $\mathrm{N_p} = 20$ and $\mathrm{N_m} = 3$. The robust horizon, $\mathrm{N_s}$, is set equal to 1, as this gives good enough results while also avoiding the exponential growth of the scenario-tree[11]. This is a reasonable assumption when the parameters are unknown but constant over the time-horizon[12]. In that case, the parameters are constant over the prediction horizon too, meaning it is only necessary to branch the scenario-tree once. The probability of the scenarios, ω_i, are assumed uniform ($\omega_\mathrm{i} = 1/\mathrm{N_s}$), and all realizations have equal probability.

In this case study, it is investigated whether increasing the amount of uncertain parameters in the scenario-tree has an effect or not. Out of the seven parameters in the system, either two or three uncertain parameters are used for the scenario-trees. It is also investigated whether it may be better to conduct SA every 5th time-step instead of at every time-step. This is expressed with $t_\mathrm{SA} = 1$ and $t_\mathrm{SA} = 5$, respectively. Recalling the switch from eq. (2.16),

$$\phi_\mathrm{k+1} = \phi(s_\mathrm{k+1}) = \phi(s_\mathrm{k+1}(x_\mathrm{k}, u_\mathrm{k+1}, p_\mathrm{k,i})), \tag{4.6}$$

where $s_\mathrm{k+1}$ and $u_\mathrm{k+1}$ each have dimensions 7×1 and 1×1 when $t_\mathrm{SA} = 1$, but when $t_\mathrm{SA} = 5$ they each have dimensions 7×5 and 1×5. However, $\phi_\mathrm{k+1}$ has either the dimensions 2×1 or 3×1 depending on the amount of uncertain parameters in the scenario-tree. In order to get correct dimensions of the switch $\phi_\mathrm{k+1}$, the weighted average of the sensitivities $s_\mathrm{k+1}$ over the t_SA time-steps can be used. The new sensitivities $\bar{s}_\mathrm{k+1}$ of dimension 7×1 can be found from

$$\bar{s}_\mathrm{k+1} = \sum_{\mathrm{j}=0}^{t_\mathrm{SA}-1} \frac{w_\mathrm{j} s_\mathrm{k+1,j}}{\sum_{\mathrm{i}=0}^{t_\mathrm{SA}-1} w_\mathrm{i}}, \tag{4.7}$$

in which the weights, w_j, e.g., can form a geometric series, giving the new formula,

$$\bar{s}_\mathrm{k+1} = \sum_{\mathrm{j}=0}^{t_\mathrm{SA}-1} \frac{0.65^\mathrm{j} s_\mathrm{k+1,j}}{\sum_{\mathrm{i}=0}^{t_\mathrm{SA}-1} 0.65^\mathrm{i}}, \tag{4.8}$$

If $t_\mathrm{SA} = 1$, the only weight w_j equals to 1, but if $t_\mathrm{SA} = 5$, these weights are smaller with the increasing time-horizon, and they sum to 1. The idea of conducting SA every 5th time-step originates from the intent of not overreacting the control actions with respect to the parameters, if two or three parameters are equally as sensitive over some time-interval. The weight, 0.65^j, indicates that the closest future predictions are more important to the weighted sensitivities $\bar{s}_\mathrm{k+1}$, which makes w_j a tuning parameter. However, only $w_\mathrm{j} = 0.65^\mathrm{j}$ is used in this thesis.

Thus, if we define $\mathrm{j} = 1, \ldots, t_\mathrm{SA}$, the eq. (4.6) can be rewritten as

$$\phi_\mathrm{k+1} = \phi(\bar{s}_\mathrm{k+1}) = \phi(x_\mathrm{k}, u_\mathrm{k+1}, \ldots, u_{\mathrm{k}+t_\mathrm{SA}}, p_\mathrm{k,i}). \tag{4.9}$$

4.2 Methodology

The case study was carried out in Python with the libraries NumPy [18], SciPy [29] and CasADi [3]. The two former libraries are well-known in the Python community, and provide efficient tools for scientific computation. The latter library is also quite well-known, providing efficient tools for nonlinear optimization and algorithmic differentiation. CasADi is supported by a symbolic framework that can implement forward and backward algorithmic differentiation on complex mathematical expressions, in order to construct large Jacobians and Hessians with sparsity that can be exploited by the solver [3].

The Python scripts that were used for this case study are shown in Appendix - code listings. Here, the optimization problems for all the different MPCs were implemented in 'optimiz.py'. In order to solve these large NLPs, the 'Interior Point OPTimizer', or Ipopt, is a natural choice as the solver. In short, it uses an interior point line search filter method that seeks to find a local solution of the NLP [8]. The maximum iterations allowed for the solver were set equal to 200. The convergence tolerance and acceptable convergence tolerance were set as $1\mathrm{e}^{-6}$ and $1\mathrm{e}^{-4}$, respectively. These values gave constraint satisfaction and feasibility of all the nominal MPCs, while lower or higher tolerances could result in minimal constraint violations and infeasibility.

The process plant for all the MPCs was implemented in 'process.py'. In order to calculate solutions of the ODE, CVODES is a great choice as the solver for stiff and non-stiff systems [34]. The maximum iterations allowed for the solver were assigned to 200, and the absolute tolerance and the relative tolerance were set as $1\mathrm{e}^{-6}$ and $1\mathrm{e}^{-4}$, respectively.

The sensitivity analysis methods and their switch-rules were implemented in 'sensitiv.py'. In order to generate Latin Hypercube samples, the function 'scipy.stats.qmc.LatinHypercube' was used from the SciPy library [30]. The main-loops for all the MPCs are shown in 'main.py', along with plots of the inputs and outputs, as well as plots of the different sensitivity indices. The remaining plots are obtained from 'utilities.py', where the packages 'matplotlib.pyplot' [16] and 'seaborn.swarmplot' [38] were helpful. These scripts were run with an Intel Core i7-8550U CPU at the estimated 1.80GHz clock-speed with 16GB of RAM using Microsoft Windows.

Chapter 5

Results and discussion

In this chapter, the results of the thesis' case study are presented and discussed. In Section 5.1, the results are of the OL-MPC. In Section 5.2, the results are of the CL-MPC. In Section 5.3, the results are of the MS-MPC without the SA-based switch. Here, the parameters Y_x and S_in are always the uncertain parameters considered in the scenario-tree, as was stated by Lucia[11]. In Section 5.4, the results are of the SA methods on the nominal (i.e., no uncertainty) CL-MPC. Here, in each of Section 5.4.1, 5.4.2 and 5.4.3, SA used is the Sobol' method, Morris screening and Modified Morris screening, respectively. In Section 5.5, the results are of the MS-MPCs with the SA-based switches, and it is compared to the MS-MPC without the SA-based switch. In Section 5.6, all the different MPCs are compared to each other.

5.1 Open-loop MPC

The nominal OL-MPC is only executed once, but the uncertain OL-MPC is executed 25 times with 15% parametric uncertainty. Figure 5.1 and 5.2 show input and output trajectories for the nominal and the uncertain OL-MPC, respectively. In Figure 5.3, the mean biomass trajectory with both one and two standard deviations is shown for the uncertain OL-MPC. In Figure 5.4, the worst-case biomass trajectory is shown for the uncertain OL-MPC.

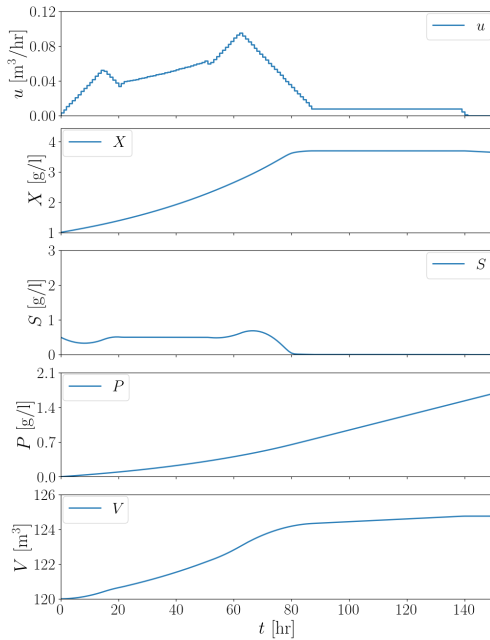


Figure 5.1: Ideal input and outputs of OL-MPC.

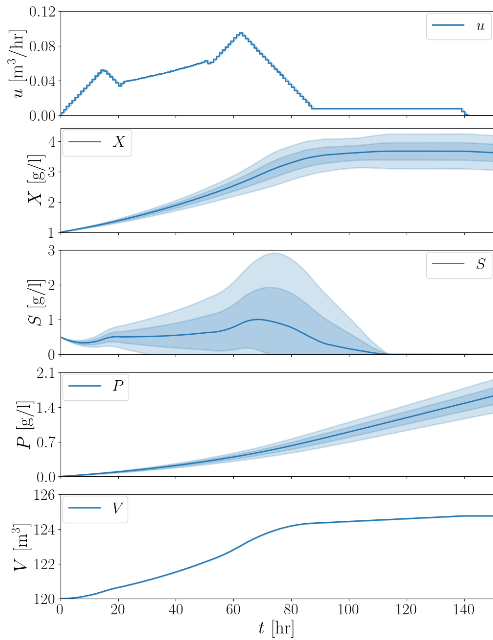


Figure 5.2: Mean input and outputs of OL-MPC.

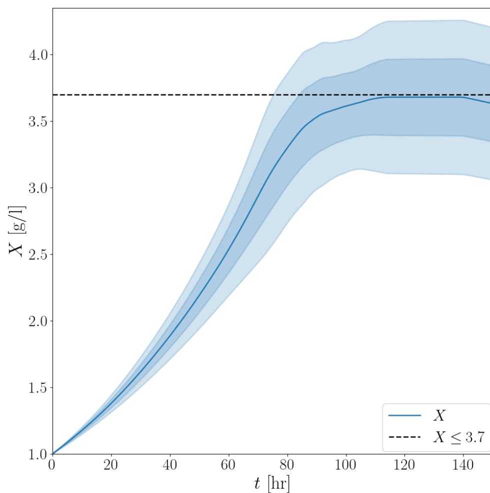


Figure 5.3: Mean biomass of OL-MPC.

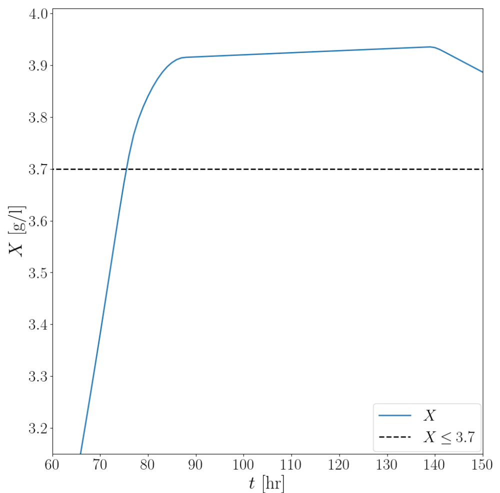


Figure 5.4: Worst-case biomass of OL-MPC.

Comparing Figure 5.1 and 5.2, it is clear for the OL-MPC, that including uncertainty in the plant gives large variation in the biomass X, substrate S and product P. The substrate feed u and the volume V are exactly the same for both figures. This is due to the OL-MPC not having feedback implemented. As seen from the process model in eq. (4.2), V is only a function of u, in which for the OL-MPC, u is only based on the initial states. Thus, u and V for the uncertain OL-MPC are equal to those of the nominal OL-MPC. However, the absence of feedback gives large variations in X, S and P. This is unwanted, as exceeding $X \leqslant 3.7$ give less penicillin.

In Figure 5.3, the mean biomass trajectory does not violate the constraint, but it comes with large standard deviations. This is unwanted, not only with respect to the constraint ($X \leqslant 3.7$), but also with regards to producing same amount of product each run [31]. As seen in Figure 5.4, the worst-case biomass trajectory gives large violations of the constraint. This is far from the optimal operation, and produced $1.59195\,\mathrm{g/l}$ penicillin instead of the optimal $1.67647\,\mathrm{g/l}$.

Here, the 'worst-case' was defined as the run with the most amount of constraint violations, not accounting for how much the constraint was violated. The minimal product concentration was as small as $1.31767\,\mathrm{g/l}$. Thus, it is clear that feedback has to be implemented in order to provide robustness against uncertainty in the plant. This is accomplished with the CL-MPC.

5.2 Closed-loop MPC

The nominal CL-MPC is only executed once, but the uncertain CL-MPC is executed 25 times with 15% parametric uncertainty. Figure 5.5 and 5.6 show input and output trajectories for the nominal and the uncertain CL-MPC, respectively. In Figure 5.7, the mean biomass trajectory with both one and two standard deviations is shown for the uncertain CL-MPC. In Figure 5.8, the worst-case biomass trajectory is shown for the uncertain CL-MPC.

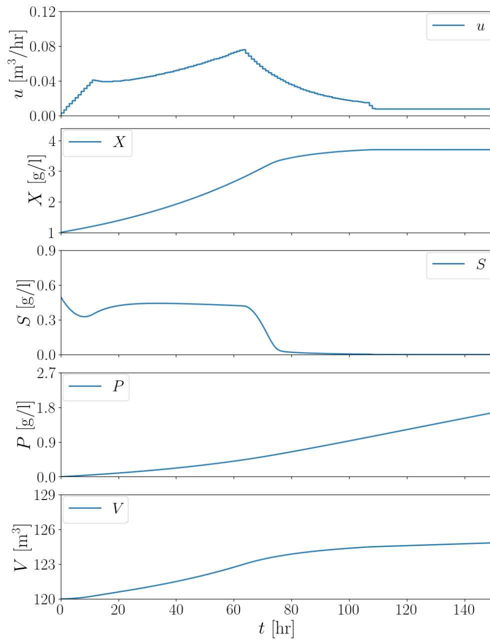


Figure 5.5: Ideal input and outputs of CL-MPC.

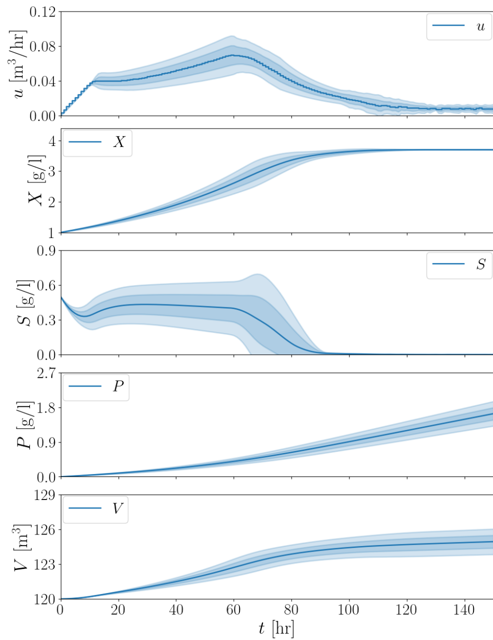


Figure 5.6: Mean input and outputs of CL-MPC.

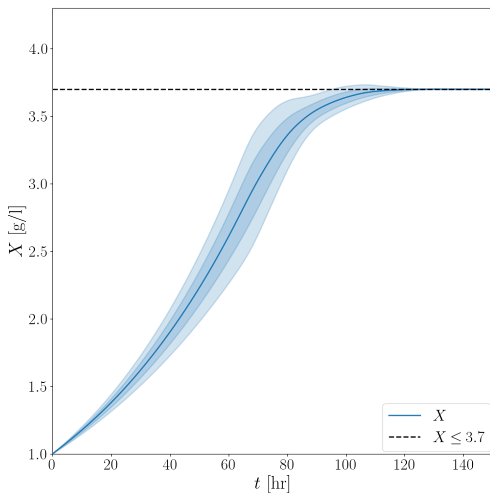


Figure 5.7: Mean biomass of CL-MPC.

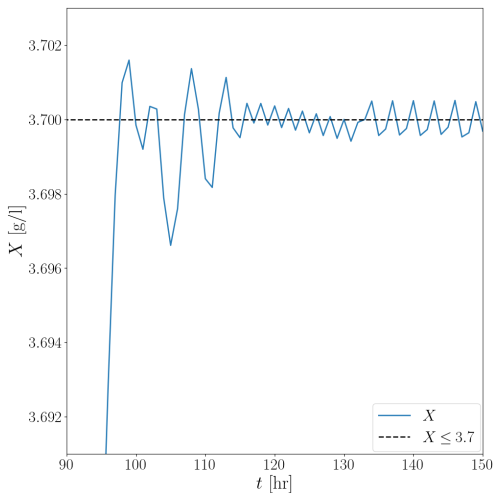


Figure 5.8: Worst-case biomass of CL-MPC.

With comparing Figure 5.5 and 5.6 of the CL-MPC to Figure 5.1 and 5.2 of the OL-MPC, it is clear that the uncertainties in the biomass X, substrate S and product P has been lowered. The substrate feed u and the volume V are not equal in the nominal to the uncertain CL-MPC, as seen in Figure 5.5 and 5.6. This is due to the inclusion of feedback, as the control actions are now based on the current states of the plant. Feedback provides robustness to the uncertainties in the plant, thus lowering the overall variance of X, S and P.

Figure 5.7 of the CL-MPC compared to Figure 5.3 of the OL-MPC, shows that the biomass constraint is violated much less. The mean of the biomass concentration is not much different, but their standard deviations differ a lot. This is much beneficial, as we can expect more of the same penicillin concentration for each run. Comparing Figure 5.8 of the CL-MPC to Figure 5.4 of the OL-MPC, in which of the worst-case biomass trajectory, show improved control of the biomass with respect to its constraint ($X \leqslant 3.7$). It is expected that the CL-MPC yields more product than the OL-MPC. However, the worst-case gave $1.52662 \mathrm{g/l}$ penicillin concentration compared to the optimal $1.67647\,\mathrm{g/l}$, which is actually less than the worst-case of OL-MPC. This is related to how quickly the biomass increases. In Figure 5.4, the biomass is already at $3.7 \mathrm{g/l}$ before the 80 hours, but in Figure 5.8, the biomass reaches $3.7 \mathrm{g/l}$ close to the 100 hours. It is a trade-off between how quickly the biomass concentration increases and how frequently it leads to constraint violation. The definition of 'worst-case' as the run with the most amount of constraint violations can be found misleading, in regards to the worst-case of the OL-MPC giving more penicillin than the worst-case of the CL-MPC. However, over the 25 runs it is still expected that the CL-MPC gives more product. The worst-case has its definition since we are interested in how much the biomass constraint is violated. The lowest penicillin concentration was found as $1.35005\,\mathrm{g/l}$, that is, more than for the OL-MPC. The average concentration was found as $1.63401\,\mathrm{g/l}$, compared to $1.62045\,\mathrm{g/l}$ of the OL-MPC, which is a great improvement.

The number of runs that violated constraint was identified as 12 for both the OL-MPC and CL-MPC, of the 25 runs, where they averaged 29.24 and 7.44 violations per run, respectively. The number of constraint violations was decreased, but there is still room for improvements. The MS-MPC is an answer to this, either without or with the SA-based switch. It might provide more robustness to the uncertainties in the plant, over the already present feedback.

5.3 MS-MPC without SA-based switch

The MS-MPC without the SA-based switch is just referred to as the MS-MPC in this section. The nominal MS-MPC is only executed once, but the uncertain MS-MPC is executed 25 times with 15% parametric uncertainty. Figure 5.9 and 5.10 show input and output trajectories for the nominal and the uncertain MS-MPC, respectively. In Figure 5.11, the mean biomass trajectory with both one and two standard deviations is shown for the uncertain MS-MPC. In Figure 5.12, the worst-case biomass trajectory is shown for the uncertain MS-MPC.

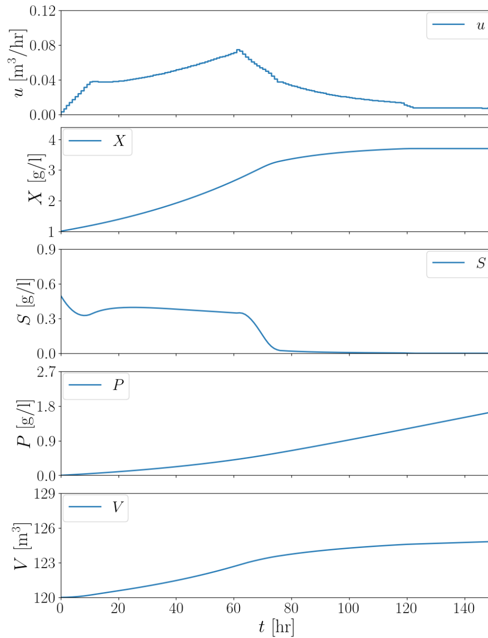


Figure 5.9: Ideal input and outputs of MS-MPC.

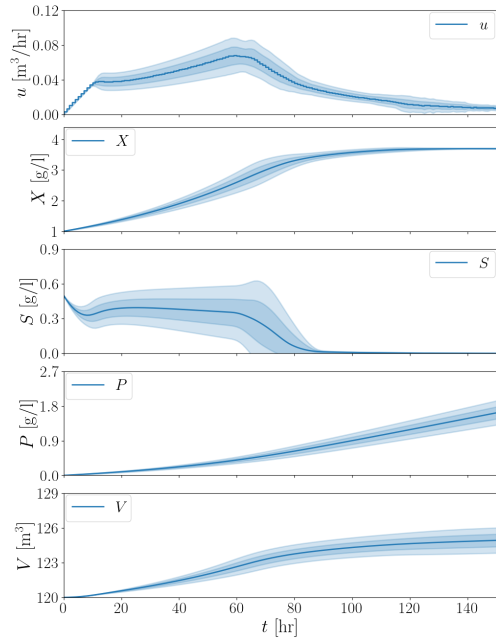


Figure 5.10: Mean input and outputs of MS-MPC.

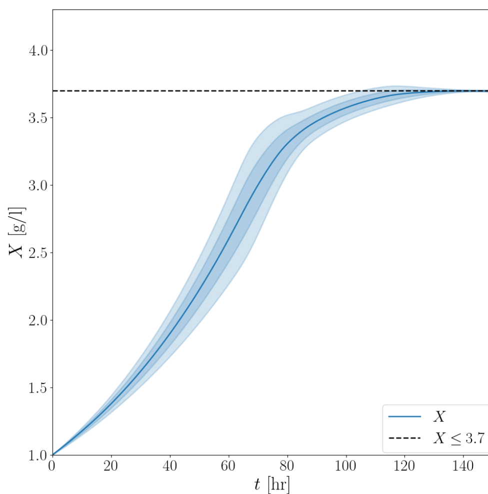


Figure 5.11: Mean biomass of MS-MPC.

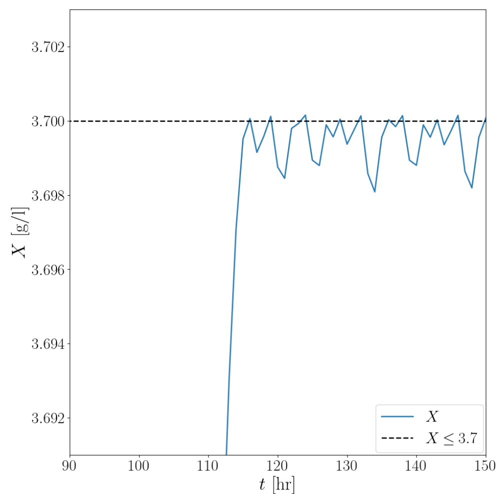


Figure 5.12: Worst-case biomass of MS-MPC.

With comparing Figure 5.9 and 5.10 of the MS-MPC to Figure 5.5 and 5.6 of the CL-MPC, the uncertainties in the biomass X, substrate S and product P have been slightly decreased. Figure 5.11 of the MS-MPC compared to Figure 5.7 of the CL-MPC, shows this slight decrease of the uncertainty with respect to the biomass. With comparing Figure 5.12 of the MS-MPC to Figure 5.8 of the CL-MPC, in which of the worst-case biomass trajectory, show better control of the biomass in regard to its constraint ($X \leqslant 3.7$). There are a lot fewer constraint violations, and the extent of the violations is smaller. Thus, it is expected that MS-MPC gives more product than the CL-MPC. The worst-case gave $1.53098\,\mathrm{g/l}$ penicillin concentration compared to the optimal $1.67647\,\mathrm{g/l}$, which is higher than the worst-case for the CL-MPC.

Here, 'worst-case' is still defined as the run with the most amount of constraint violations. The lowest penicillin concentration was found as $1.34354\,\mathrm{g/l}$, that is, less than the CL-MPC. The average concentration was found as $1.62107\,\mathrm{g/l}$, compared to $1.63401\,\mathrm{g/l}$ of the CL-MPC. This is a significant difference, meaning that the MS-MPC produces less product than the basic CL-MPC. Even though the biomass constraint is more respected, there is a trade-off between how quickly the biomass increases and how much constraint violations there are. In Figure 5.8, the biomass is already at $3.7\,\mathrm{g/l}$ before the 100 hours, but in Figure 5.12, the biomass reaches $3.7\,\mathrm{g/l}$ after the 110 hours. Thus, the MS-MPC with the two uncertain parameters Y_x and S_in, gives *conservative* control. This is contrary to Lucia [11], where the MS-MPC performed better than the CL-MPC. However, in his case study there were only uncertainties in the parameters Y_x and S_in. Here, the parameter Y_x was uniformly distributed with constant 25% uncertainty, and S_in was normally distributed with 12.5% uncertainty that varied each hour. In this thesis' case study, there was 15% uncertainty in all the parameters, which makes the main reason for the differences in the results. It was chosen an equal amount of uncertainty for the parameters, as we do not know the actual uncertainties in the plant, thus giving a more objective view.

The number of runs that violated constraint was decreased from 12 to 6, of the 25 runs, when comparing the MS-MPC with the CL-MPC. The average violations per run were lowered from 7.44 to 2.00, meaning that the MS-MPC provides great improvements with respect to the biomass constraint. However, it comes at the cost of less penicillin, i.e., conservative control. There are still constraint violations and low amounts of penicillin are produced. How can this issue be resolved? The proposed solution is to include the SA-based switch for the MS-MPC. This could lower the constraint violations even further, thus giving more product. It should be mentioned that in Figure 5.12, the constraint violations are minimal. However, if the biomass exceeds the constraint ($X \leqslant 3.7$), control actions are constantly made in order to adjust the biomass below the constraint. Optimally, we want the biomass to stay just below the constraint at all times. In the next section, the different SA methods are investigated for the later inclusion in the MS-MPC with the SA-based switch.

5.4 Sensitivity Analysis

In this section, it is shown plots of the different SA methods applied on the nominal CL-MPC. It was decided to show the results of the CL-MPC instead of the MS-MPC, as the idea was that using SA on the CL-MPC could give more objective answers than using SA on the MS-MPC, which already has assumed Y_{x} and S_{in} as the uncertain parameters in the scenario-tree.

5.4.1 Sobol' method

In Figure 5.13 and 5.14, the first-order and the total-order Sobol' indices are presented for the $\mathrm{N} = 2^{12}$ base samples, respectively. In Figure 5.15 and 5.16, the first-order and the total-order Sobol' indices are presented for the $\mathrm{N} = 2^{15}$ base samples, respectively.

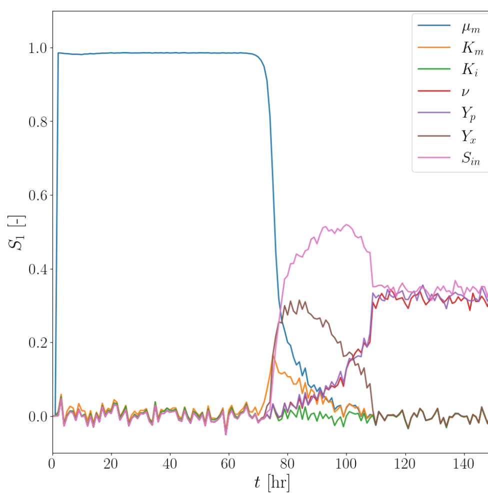


Figure 5.13: First-order Sobol' indices ($\mathrm{N} = 2^{12}$).

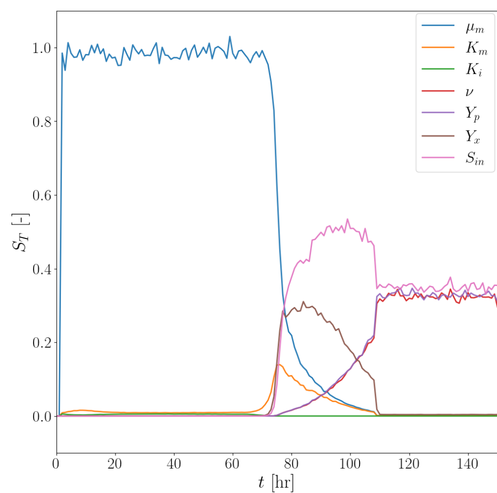


Figure 5.14: Total-order Sobol' indices ($\mathrm{N} = 2^{12}$).

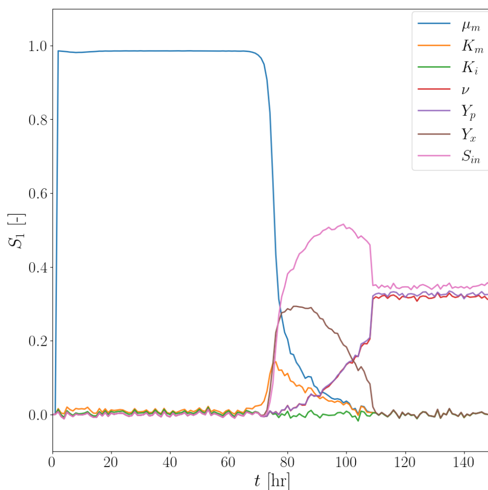


Figure 5.15: First-order Sobol' indices ($\mathrm{N} = 2^{15}$).

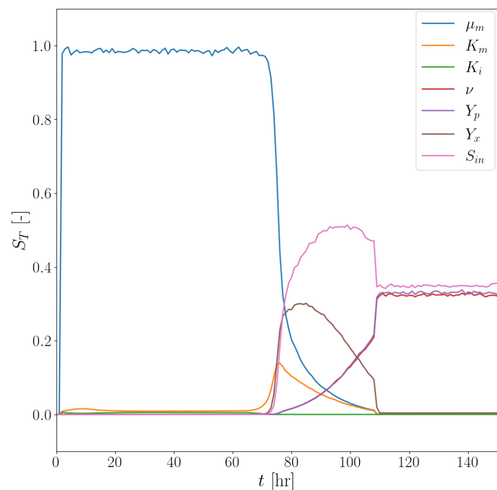


Figure 5.16: Total-order Sobol' indices ($\mathrm{N} = 2^{15}$).

Recall the settings shown in Section 3.1.2. The first-order Sobol' indices are related to the FP setting, i.e., identifying input factors that give the largest reductions of the output variance. The total-order Sobol' indices are related with the FF setting, i.e., identifying input factors that contribute little to the overall output variance. As seen in Figure 5.13 to the 5.16, the first-order Sobol' indices, S_1, better describe the influence of the most sensitive parameters. Meanwhile, the total-order Sobol' indices, S_{T}, better describe the other parameters with smaller influence. As there are seven parameters in the system, only the total-order Sobol' indices are used in the SA-based switch for the MS-MPC. The use of the switch becomes more relevant after the first 70 hours have passed, as the biomass starts to approach the constraint. After 70 hours and out, there are a lot of different parameters that are influential, and thus the FF setting seems more reasonable than using the FP setting.

Figure 5.13 to the 5.16 show that μ_{m} is the most sensitive parameter in the first half of the fermentation process. As seen from the ODE model in the eq. (4.2), this is due to that there is produced a lot of biomass from all the available sugar in the bioreactor. Here, μ_{m} denotes the maximum specific growth rate, and sets the limit of how much biomass that can be produced. As the reaction proceeds, the influence of μ_{m} decreases and the other parameters become more influential on the biomass concentration. However, the parameter K_{i} remains little influential during the whole process, and could have been disregarded in further SA. The parameter K_{m} is also little influential on the biomass, but at the same time, it is the second most influential parameter in the first half of the process. Its values are rather low due to μ_m being much more influential than any other parameter halfway into the process. The other parameters that are of significant importance are the ν, Y_{p}, Y_{x} and S_{in}. Here, Y_{x} become influential after 70 hours have passed, and its influence decreases greatly after 110 hours. Also the ν, Y_{p} and S_{in} become influential after 70 hours, but remain influential to the end of the process. Thus, it is clear that μ_m, ν, Y_{p}, Y_{x} and S_{in} must be accounted for as the uncertain parameters in the scenario-tree. Also K_{m} can be considered, but K_{i} can be left out. In the work of Srinivasan [33] and Lucia [11], only Y_{x} and S_{in} are used as the uncertain parameters. Figure 5.13 to the 5.16 show that it is not reasonable to consider only these two parameters. In fact, Y_{x} is non-influential on the biomass after 110 hours. This is a strong argument for using the SA-based switch for the MS-MPC.

As mentioned in Chapter 3, the variance-based GSA generally requires a lot of samples to provide reasonable results. Comparing Figure 5.13 and 5.14 to Figure 5.15 and 5.16, it is clear that $\mathrm{N} = 2^{15}$ base samples gives better results than $\mathrm{N} = 2^{12}$ base samples. If the sample size was increased even further, the first-order and total-order Sobol' indices would approach the perfect description of the sensitivities. However, due to the computational costs of evaluating the model many times, the MS-MPC with the SA-based switch considers $\mathrm{N} = 2^{10}$ and $\mathrm{N} = 2^{12}$ number of base samples. These sample sizes are quite low for the Sobol' method [27], but maybe the screening-based methods perform better for the same sample sizes?

5.4.2 Morris screening

In Figure 5.17 and 5.18, elementary effects with one and two standard deviations are presented for the $\mathrm{N} = 2^{12}$ base samples, respectively. In Figure 5.19 and 5.20, elementary effects with one and two standard deviations are presented for the $\mathrm{N} = 2^{15}$ base samples, respectively.

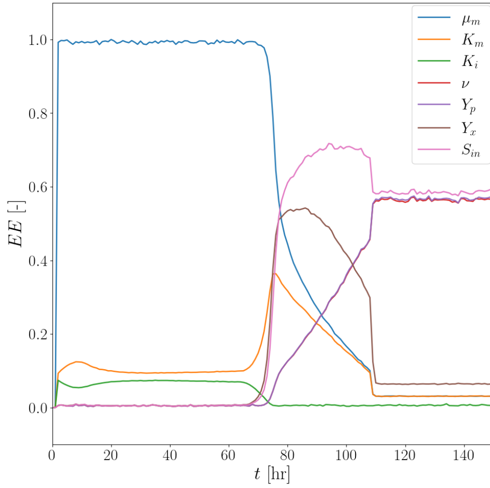


Figure 5.17: Elementary effects ($\mathrm{N} = 2^{12}$).

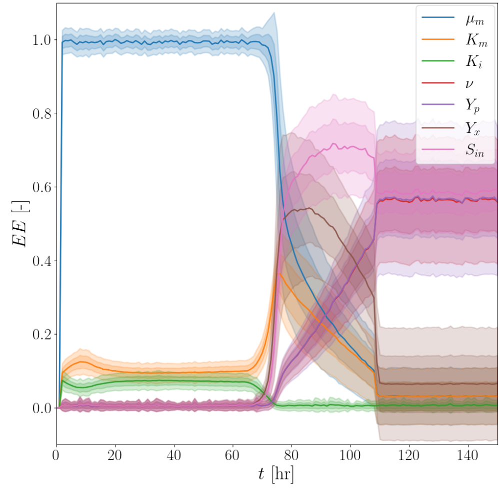


Figure 5.18: EE with one and two SD ($\mathrm{N} = 2^{12}$).

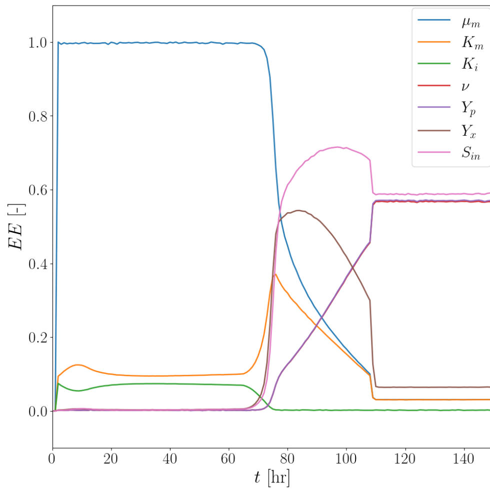


Figure 5.19: Elementary effects ($\mathrm{N} = 2^{15}$).

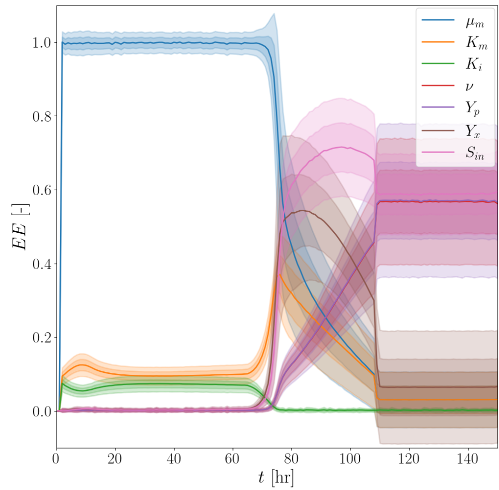


Figure 5.20: EE with one and two SD ($\mathrm{N} = 2^{15}$).

Recalling the settings presented in Section 3.1.2, the elementary effects are related with the FF setting, i.e., identifying the input factors that contribute little to the overall output variance. Figure 5.17 to the 5.20 are scaled so that the maximum value is equal to one, which makes the comparison with the Sobol' method easier. The elementary effects do not sum to one, however. With comparing Figure 5.17 and 5.19 of the elementary effects to Figure 5.14 and 5.16 of the total-order Sobol' indices, there are many similarities. Here, μ_{m} is the most sensitive parameter in the first half of the process, then μ, Y_{p}, Y_{x} and S_{in} are the most sensitive parameters in the second half of the process. However, μ_{m} is not as dominant as for the Sobol' method, and the parameters K_{m} and K_{i} are more influential. The Morris screening provides semi-quantitative information on the sensitivity measures. It struggles with identifying which of the input factors that experience non-linearity and interactions. This may be the main reason for the differences between the two GSA methods.

In Figure 5.18 and 5.20, the standard deviations are measures of how much the parameters interact with each other. It is clear that all of the parameters have interactions with the others. Here, there are little interactions in the first half of the process, but the parameters experience more interactions in the second half of the process. These interactions are not accounted for in the elementary effects, and it explains the differences between Figure 5.17 and 5.19 to Figure 5.14 and 5.16. Is the Morris screening worth using as it struggles to describe the interactions? The general answer is yes. Even though only the elementary effects were used in the SA-based switch for the MS-MPC, it is possible to create a new sensitivity measure of both EE and σ. It was not done in this thesis, and often it is enough with only EE[27]. The Morris screening provides quite simple sensitivity measures but is more cost effective than the Sobol' method.

Comparing Figure 5.17 and 5.19 with Figure 5.14 and 5.16, it is clear that Morris screening requires lesser samples than Sobol' method. It is seen by how smooth the different graphs are. Increasing the number of base samples increases the graph smoothness, and for Sobol' method it is required more samples to obtain smoothness. In real plants, MPCs calculate control actions as fast as possible, so that the inputs can adjust quickly and drive the controlled variables to the predicted values. It is particularly important for processes that measure states and disturbances every second or every minute. The time-interval for this case study was one hour, but the idea still remains. It is desired to adjust the inputs as fast as possible, in which the Morris screening outperforms the Sobol' method. On the other hand, Sobol' indices are more accurate measures.

What if we combined the OAT approach of the Morris screening with the LH sampling of the Sobol' method? Would this provide a great middle ground between the two GSA methods?

5.4.3 Modified Morris screening

In Figure 5.21 and 5.22, the partial effects with one and two standard deviations are presented for the $\mathrm{N} = 2^{12}$ base samples, respectively. In Figure 5.23 and 5.24, the partial effects with one and two standard deviations are presented for the $\mathrm{N} = 2^{15}$ base samples, respectively.

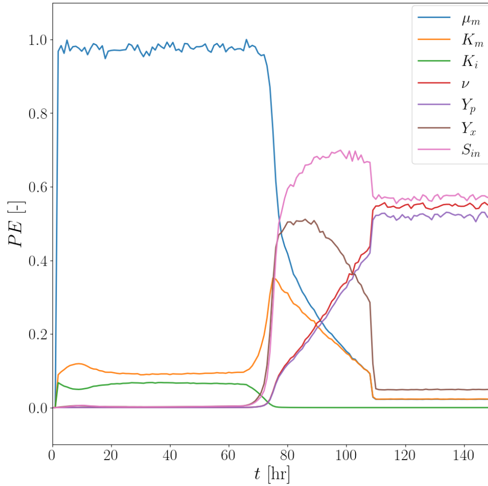


Figure 5.21: The partial effects ($\mathrm{N} = 2^{12}$).

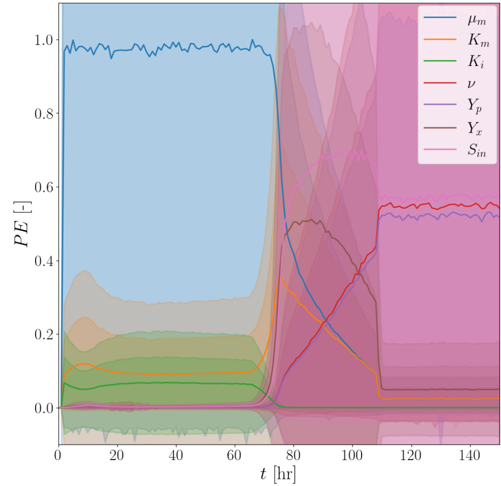


Figure 5.22: PE with one and two SD ($\mathrm{N} = 2^{12}$).

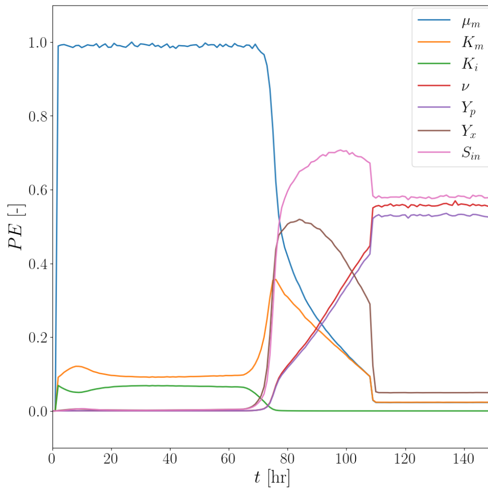


Figure 5.23: The partial effects ($\mathrm{N} = 2^{15}$).

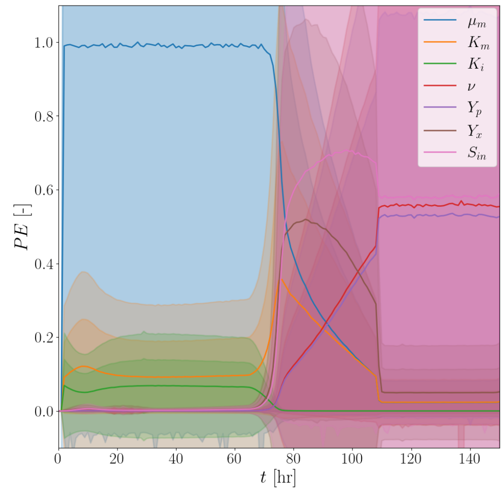


Figure 5.24: PE with one and two SD ($\mathrm{N} = 2^{15}$).

Recalling all the settings introduced in Section 3.1.2, the partial effects are related with the FF setting, i.e., identifying the input factors that contribute little to the overall output variance. Figure 5.21 to the 5.24 are scaled so that the maximum value is equal to one, which makes the comparison with the Sobol' method and Morris screening easier. As for the elementary effects, neither do the partial effects sum to one. Comparing Figure 5.21 and 5.23 of the partial effects with Figure 5.14 and 5.16 of the total-order Sobol' indices and to Figure 5.17 and 5.19 of the elementary effects, there are many similarities. Here, μ_m is the most sensitive parameter in the first half of the process, then μ, Y_p, Y_x and S_in are the most sensitive parameters in the second half of the process. However, similarly to the Morris screening, μ_m is not as dominant as for the Sobol' method, and the parameters K_m and K_i are more influential. Hence, it is clear that the partial effects inherit similarities to the elementary effects.

Similarities with the Sobol' method are seen by the smoothness of the partial effect graphs. The LH sampling requires more samples than the Morris sampling, where Figure 5.21 and 5.23 present these partial effects as a middle ground between the total-order Sobol' indices and the elementary effects, in terms of the required sample size. The partial effects describe the whole range of the parameters, while having effectiveness of the OAT approach. Figure 5.21 and 5.23 show that ν is greater than Y_p after 110 hours, which is not the case for the other SA methods. The Sobol' method has Y_p slightly higher than ν, while the Morris screening has ν equal to Y_p through the whole process. It can be beneficial to not change the scenario-tree constantly if the sensitivities are very close, and in that regards the Modified Morris screening performs better. This is also the reason why it was experimented with using SA every time-step and every fifth time-step. In this case study, the partial effects were better at differentiating ν and Y_p because of the LH sampling can describe the full range of the parameters. As for the elementary effects, the partial effects also struggle with identifying which of the parameters that have non-linearity and interactions. However, the standard deviations of the partial effects yield less reasonable results than the standard deviations of the elementary effects, as seen in Figure 5.22 and 5.24.

Using samples of the full parameter space could have proven troublesome for the definition of the standard deviation, shown in the eq. (3.35), and it would explain why van Griensven [5] only considered the partial effects as the sensitivity measure. On the other hand, it may be that Figure 5.22 and 5.24 provide more true results than Figure 5.18 and 5.20, as the full parameter space is used. In that case, the parameters experience much interaction, and the Sobol' method would give better sensitivity measures, as its sensitivity measures can capture interactions.

Increasing the sample size provides better smoothness of the partial effects graphs, but it is required more samples than with the elementary effects. The partial effects are more accurate sensitivity measures, but not as accurate as the total-order Sobol' indices [5]. We will not look at which of the SA methods provide the best results for the MS-MPC with the SA-based switch.

5.5 MS-MPC with SA-based switch

In this section, the worst-case biomass trajectory of the 25 runs, that is, for the MS-MPC with SA-based switch, is plotted for each of the SA methods and compared to the MS-MPC without the SA-based switch. These switches are denoted CNST-switch, SOBO-switch, MORR-switch and MMOR-switch for each of the constant switch, Sobol' based switch, Morris based switch and Modified Morris based switch. Figure 5.25 to the 5.28 consider two uncertain parameters in the scenario-tree. Figure 5.25 and 5.26 show worst-case biomass trajectories for the $\mathrm{N} = 2^{10}$ base samples with $t_{\mathrm{SA}} = 1$ and $t_{\mathrm{SA}} = 5$, respectively. Figure 5.27 and 5.28 show worst-case biomass trajectories for the $\mathrm{N} = 2^{12}$ base samples with $t_{\mathrm{SA}} = 1$ and $t_{\mathrm{SA}} = 5$, respectively.

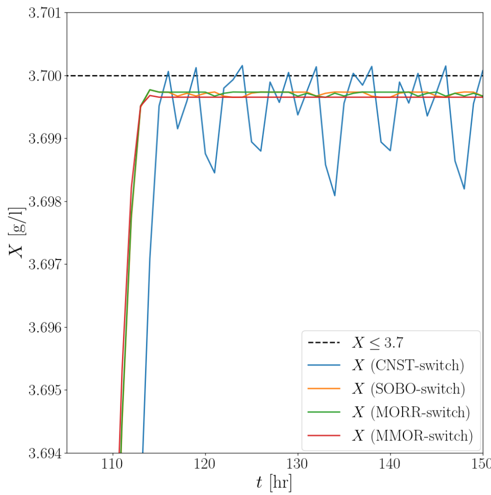


Figure 5.25: Worst-case biomass of the MS-MPC of two uncertain parameters ($\mathrm{N} = 2^{10}$, $t_{\mathrm{SA}} = 1$).

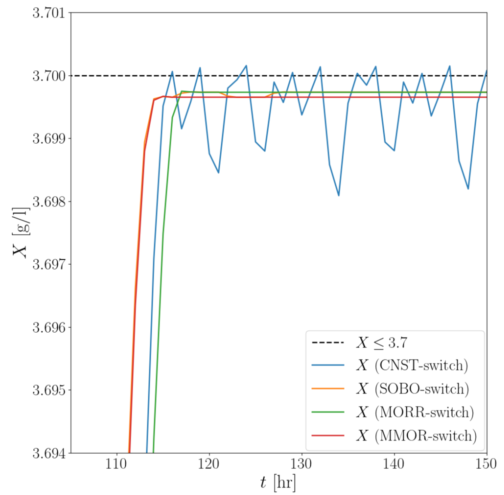


Figure 5.26: Worst-case biomass of the MS-MPC of two uncertain parameters ($\mathrm{N} = 2^{10}$, $t_{\mathrm{SA}} = 5$).

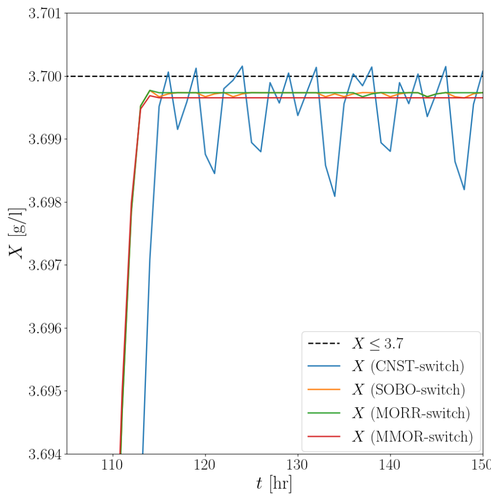


Figure 5.27: Worst-case biomass of the MS-MPC of two uncertain parameters ($\mathrm{N} = 2^{12}$, $t_{\mathrm{SA}} = 1$).

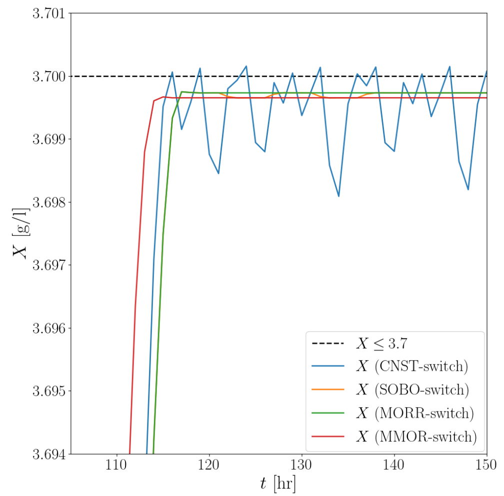


Figure 5.28: Worst-case biomass of the MS-MPC of two uncertain parameters ($\mathrm{N} = 2^{12}$, $t_{\mathrm{SA}} = 5$).

Figure 5.25 to the 5.28 show that the worst-case biomass trajectory of the SOBO-switch, MORR-switch and MMOR-switch MS-MPC do not violate the biomass constraint ($X \leqslant 3.7$). Here, the worst-case is defined as the run with the highest biomass concentration at any hour, as there are no constraint violations. The worst-case of the CNST-switch MS-MPC is still the run with the most amount of constraint violations. It is clear that the MS-MPCs with SA-based switches have biomass trajectories closer to the constraint, thus resulting in more penicillin.

With comparing Figure 5.25 and 5.27 to Figure 5.26 and 5.28, it seems that using SA every time-step ($t_{\mathrm{SA}} = 1$) performs better than using SA every fifth time-step ($t_{\mathrm{SA}} = 5$). There is no difference in how close one gets to the constraint using either of these, but $t_{\mathrm{SA}} = 1$ results in quicker increases of the biomass towards the constraint. It is clear from Figure 5.25 and 5.27, that the SA-based switches perform similarly. It can be said that the MORR-switch results in higher biomass, but the differences are minimal between the SA methods when using $t_{\mathrm{SA}} = 1$. However, when using $t_{\mathrm{SA}} = 5$ as in Figure 5.26 and 5.28, the MORR-switch performs worse than the other switches. Its biomass increases rather slowly, giving less amount of penicillin.

By comparing Figure 5.25 and 5.26 to Figure 5.27 and 5.28, it was found that increasing the sample size has close to none effect on the worst-case biomass trajectories. This is beneficial, as the computational costs are lower for less base samples. With the $\mathrm{N} = 2^{10}$ number of base samples instead of the $\mathrm{N} = 2^{12}$ number of base samples, the computational costs are close to four times lower. In real plants, it is desired to adjust the manipulated inputs as fast as possible, even when the time-interval between each measurement is one hour. Hence, the best choice of the SA-based switch is shown in Figure 5.25, in which $\mathrm{N} = 2^{10}$ base samples are used and SA is used every time-step ($t_{\mathrm{SA}} = 1$). Here, the MORR-switch performed slightly better than the SOBO-switch and the MMOR-switch, in which each of them outperformed the CNST-switch. Hence, the natural choice is the MS-MPC with the MORR-switch using $\mathrm{N} = 2^{10}$ and $t_{\mathrm{SA}} = 1$.

The average penicillin concentration at the end of the process, and its standard deviation, with respect to the 25 runs, for each of these different MS-MPCs, are shown later in Table 5.2.

Figure 5.29 to the 5.32 consider three uncertain parameters in the scenario-tree. Figure 5.29 and 5.30 show worst-case biomass trajectories for the $\mathrm{N} = 2^{10}$ number of base samples with $t_{\mathrm{SA}} = 1$ and $t_{\mathrm{SA}} = 5$, respectively. Figure 5.31 and 5.32 show worst-case biomass trajectories for the $\mathrm{N} = 2^{12}$ number of base samples with $t_{\mathrm{SA}} = 1$ and $t_{\mathrm{SA}} = 5$, respectively.

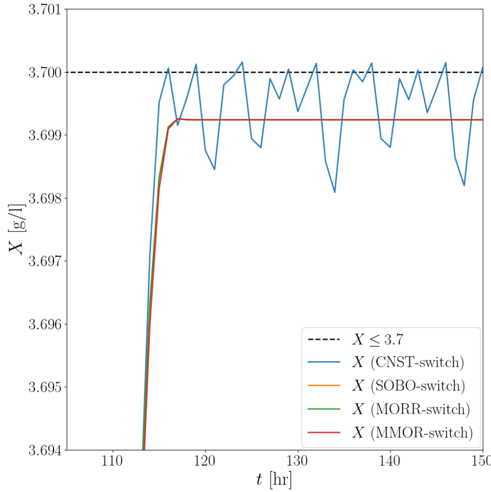


Figure 5.29: Worst-case biomass of the MS-MPC of three uncertain parameters ($\mathrm{N} = 2^{10}$, $t_{\mathrm{SA}} = 1$).

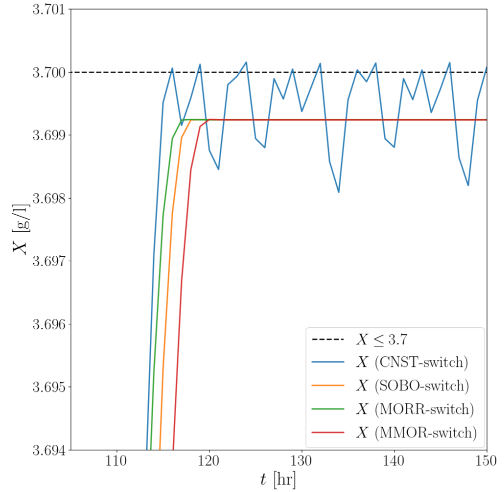


Figure 5.30: Worst-case biomass of the MS-MPC of three uncertain parameters ($\mathrm{N} = 2^{10}$, $t_{\mathrm{SA}} = 5$).

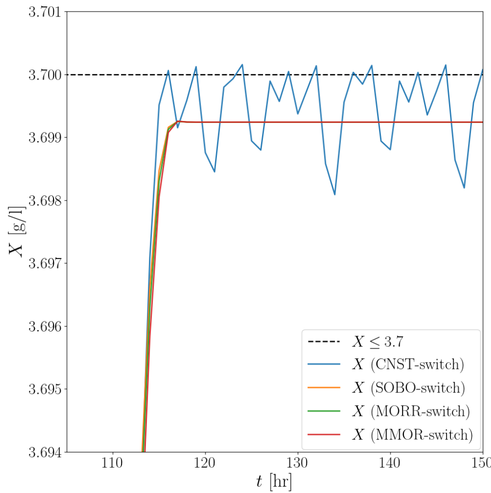


Figure 5.31: Worst-case biomass of the MS-MPC of three uncertain parameters ($\mathrm{N} = 2^{12}$, $t_{\mathrm{SA}} = 1$).

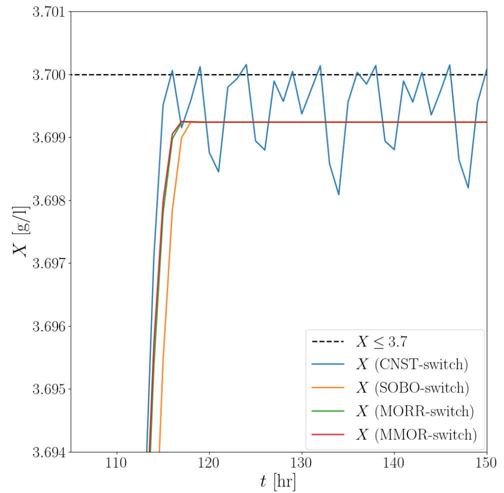


Figure 5.32: Worst-case biomass of the MS-MPC of three uncertain parameters ($\mathrm{N} = 2^{12}$, $t_{\mathrm{SA}} = 5$).

Figure 5.29 to the 5.32 show that the worst-case biomass trajectory of the SOBO-switch, MORR-switch and MMOR-switch MS-MPC do not violate the biomass constraint ($X \leqslant 3.7$). Here, the worst-case is defined as the run with the highest biomass concentration at any hour, as there are no constraint violations. The worst-case of the CNST-switch MS-MPC is still the run with the most constraint violations. In Figure 5.25 to the 5.28, the MS-MPCs consider two uncertain parameters in the scenario-tree. The MS-MPCs with SA-based switch have biomass trajectories closer to the constraint. This is not the case for the MS-MPCs that consider three uncertain parameters in the scenario-tree. As seen from Figure 5.29 to the 5.32, the worst-case biomass trajectories of the SOBO-switch, MORR-switch and MMOR-switch MS-MPC are not closer to the constraint than for the CNST-switch MS-MPC. These trajectories seem to mimic the behavior of a constant back-off. With using more uncertain parameters in the scenario-tree, there is less need for switching along the horizon. In particular, this is relevant after 110 hours into the process, as the μ, Y_{p} and S_{in} are selected between for the MS-MPCs that consider two uncertain parameters, but the MS-MPCs that consider three uncertain parameters can include all of them. By considering three uncertain parameters instead of two, conservativeness of the MS-MPC might increase, which can be seen by comparing Figure 5.29 to the 5.32 with Figure 5.25 to the 5.28. The worst-case biomass trajectories are further from the biomass constraint, giving less product. It is due to that these MS-MPCs account for more uncertainty than needed.

With comparing Figure 5.29 and 5.31 to Figure 5.30 and 5.32, it seems that using SA every time-step ($t_{\mathrm{SA}} = 1$) performs better than using SA every fifth time-step ($t_{\mathrm{SA}} = 5$). There is no difference in how close one gets to the constraint using either of these, but $t_{\mathrm{SA}} = 1$ results in quicker increases of the biomass towards the constraint. It is clear from Figure 5.29 and 5.31, that all the SA-based switches perform similarly for $t_{\mathrm{SA}} = 1$. However, when $t_{\mathrm{SA}} = 5$ as in Figure 5.30 and 5.32, the SOBO-switch, MORR-switch and MMOR-switch do not lead to the same performance. Regardless, it is hard to find a pattern in the switches between these plots.

Comparing Figure 5.29 and 5.30 to Figure 5.31 and 5.32, it was found that increasing the sampling size has close to none effect on the worst-case biomass trajectories when $t_{\mathrm{SA}} = 1$, but some effect when $t_{\mathrm{SA}} = 5$. In the latter case, the MMOR-switch performed better for the $\mathrm{N} = 2^{12}$ base samples than for the $\mathrm{N} = 2^{10}$ base samples. Otherwise, there was no effect with increasing the sample size. In summary, Figure 5.25 to the 5.32 only has one case in which the greater sample size gave better results. With using $t_{\mathrm{SA}} = 5$ instead of $t_{\mathrm{SA}} = 1$, it led to worse results for all the cases. Having two uncertain parameters instead of three in the scenario-tree, gave better results for all the cases. Of the MS-MPCs that consider three uncertain parameters, the best choice would be in Figure 5.33, in which $\mathrm{N} = 2^{10}$ and $t_{\mathrm{SA}} = 1$. Here, it makes sense to use either the MORR-switch or MMOR-switch, as they have less computational costs than the SOBO-switch. On the other hand, considering two uncertain parameters should always be used over three uncertain parameters in this case study, as the latter gave conservative control.

The average penicillin concentration at the end of the process, and its standard deviation, with respect to the 25 runs, for each of these different MS-MPCs, are shown later in Table 5.2.

5.6 Comparing all the MPCs

In this section, Figure 5.33, 5.34 and 5.35, as well as Table 5.1 and 5.2, provide the comparisons of these different MPCs, in which are 27 MPCs in total. The OL-MPC, CL-MPC and MS-MPC without the SA-based switch (i.e., Y_x and S_in are always used in the scenario-tree) count as the first three MPCs. Furthermore, MS-MPCs with the SA-based switch can consider two or three uncertain parameters, using either $\mathrm{N} = 2^{10}$ or $\mathrm{N} = 2^{12}$ base samples, as well as using either $t_\mathrm{SA} = 1$ or $t_\mathrm{SA} = 5$ (i.e., whether to conduct SA every time-step or only every fifth time-step). Moreover, the MS-MPCs can use either the SOBO-switch, MORR-switch or MMOR-switch. This counts for 24 MPCs, giving the total of 27 MPCs. Table 5.1 presents information on each of these with respect to the biomass constraint. That is, the number of runs out of the 25 runs which produced constraint violations, and the average number of constraint violations per run. 2-UP and 3-UP state whether the scenario-tree considered two or three uncertain parameters.

Table 5.1: Information on each of the MPCs for the 25 runs with respect to the biomass constraint.

Controller	Number of runs that violated constraint [-]	Average constraint violations per run [-]
OL-MPC	12	29.24
CL-MPC	12	7.44
2-UP-MS-MPC, CNST-switch	6	2.00
2-UP-MS-MPC, SOBO-switch, $\mathrm{N} = [2^{10}, 2^{12}]$, $t_\mathrm{SA} = [1, 5]$	0	0.00
2-UP-MS-MPC, MORR-switch, $\mathrm{N} = [2^{10}, 2^{12}]$, $t_\mathrm{SA} = [1, 5]$	0	0.00
2-UP-MS-MPC, MMOR-switch, $\mathrm{N} = [2^{10}, 2^{12}]$, $t_\mathrm{SA} = [1, 5]$	0	0.00
3-UP-MS-MPC, SOBO-switch, $\mathrm{N} = [2^{10}, 2^{12}]$, $t_\mathrm{SA} = [1, 5]$	0	0.00
3-UP-MS-MPC, MORR-switch, $\mathrm{N} = [2^{10}, 2^{12}]$, $t_\mathrm{SA} = [1, 5]$	0	0.00
3-UP-MS-MPC, MMOR-switch, $\mathrm{N} = [2^{10}, 2^{12}]$, $t_\mathrm{SA} = [1, 5]$	0	0.00

From Table 5.1, it is clear that none of the MS-MPCs with SA-based switches resulted in constraint violations. As it is expected, the OL-MPC results in the most constraint violations, which is improved by using feedback with the CL-MPC, giving robustness to the uncertainties. The amount of constraint violations is lowered even further with the CNST-switch MS-MPC, in which the evolution of the uncertainties is modeled by the scenario-tree. This controller only accounts for the uncertainty evolution of the parameters Y_x and S_in, which was found to be an unreasonable assumption. As seen in Section 5.4, there are other parameters that are sensitive to the biomass concentration. The MS-MPCs with SA-based switches take the most sensitive parameters into account for the scenario-tree, which led to none constraint violations for each of the cases. Whether it was considered two or three uncertain parameters in the scenario-tree, or the number of base samples was $\mathrm{N} = 2^{10}$ or $\mathrm{N} = 2^{12}$, or if it was used $t_\mathrm{SA} = 1$ or $t_\mathrm{SA} = 5$, the SOBO-switch, MORR-switch and MMOR-switch MS-MPC did not violate the constraint.

Figure 5.33 presents the so-called swarmplots of the penicillin concentration at the end of the process, of all the 25 runs, for each of the MPCs. The concentrations that correspond to the violation of the biomass constraint are shown as the orange points, otherwise, these are blue.

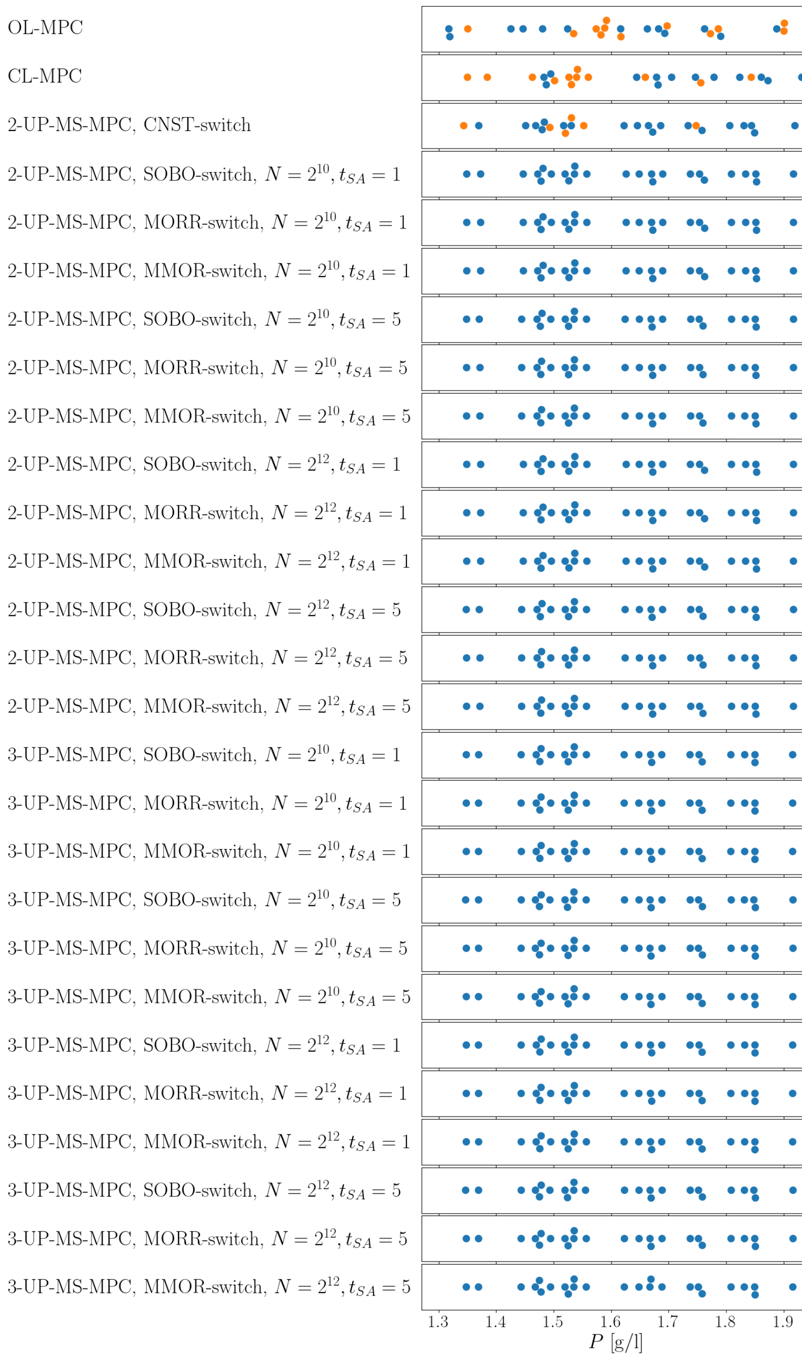


Figure 5.33: Penicillin concentration at the end of the process, of all the 25 runs, for each of the MPCs.

As seen in Figure 5.33, the MS-MPCs with SA-based switch perform similarly with regard to producing penicillin. The OL-MPC produces the least amount of penicillin over the 25 runs, while the CL-MPC produces the most. The CNST-switch MS-MPC yields less penicillin than the CL-MPC, which may surprise. It is expected that lesser violations of the biomass constraint result in more product, but as mentioned earlier, there is a trade-off between how quickly the biomass increases and how much its constraint is violated. Hence, the CNST-switch MS-MPC provides conservative control. The MS-MPCs with SA-based switches provide better control, in which for all the cases of the SOBO-switch, MORR-switch and MMOR-switch MS-MPC. However, the CL-MPC still results in the highest penicillin concentration, indicating that these MS-MPCs also are conservative. In order to get a better view of these penicillin concentrations, Table 5.2 provides the average penicillin concentration at the end, and its standard deviation, with respect to the 25 runs, for each of the different MPCs.

Table 5.2: Average penicillin concentration at the end of the process, for each of the MPCs.

Controller	Average product concentration [g/l]
OL-MPC	1.62045 ± 0.16865
CL-MPC	1.63401 ± 0.15934
2-UP-MS-MPC, CNST-switch	1.62107 ± 0.15700
2-UP-MS-MPC, SOBO-switch, N $= 2^{10}$, $t_{\mathrm{SA}} = 1$	1.62410 ± 0.15730
2-UP-MS-MPC, MORR-switch, N $= 2^{10}$, $t_{\mathrm{SA}} = 1$	1.62412 ± 0.15731
2-UP-MS-MPC, MMOR-switch, N $= 2^{10}$, $t_{\mathrm{SA}} = 1$	1.62412 ± 0.15726
2-UP-MS-MPC, SOBO-switch, N $= 2^{10}$, $t_{\mathrm{SA}} = 5$	1.62326 ± 0.15760
2-UP-MS-MPC, MORR-switch, N $= 2^{10}$, $t_{\mathrm{SA}} = 5$	1.62335 ± 0.15749
2-UP-MS-MPC, MMOR-switch, N $= 2^{10}$, $t_{\mathrm{SA}} = 5$	1.62332 ± 0.15750
2-UP-MS-MPC, SOBO-switch, N $= 2^{12}$, $t_{\mathrm{SA}} = 1$	1.62413 ± 0.15730
2-UP-MS-MPC, MORR-switch, N $= 2^{12}$, $t_{\mathrm{SA}} = 1$	1.62411 ± 0.15730
2-UP-MS-MPC, MMOR-switch, N $= 2^{12}$, $t_{\mathrm{SA}} = 1$	1.62409 ± 0.15728
2-UP-MS-MPC, SOBO-switch, N $= 2^{12}$, $t_{\mathrm{SA}} = 5$	1.62319 ± 0.15759
2-UP-MS-MPC, MORR-switch, N $= 2^{12}$, $t_{\mathrm{SA}} = 5$	1.62336 ± 0.15749
2-UP-MS-MPC, MMOR-switch, N $= 2^{12}$, $t_{\mathrm{SA}} = 5$	1.62332 ± 0.15749
3-UP-MS-MPC, SOBO-switch, N $= 2^{10}$, $t_{\mathrm{SA}} = 1$	1.62230 ± 0.15741
3-UP-MS-MPC, MORR-switch, N $= 2^{10}$, $t_{\mathrm{SA}} = 1$	1.62228 ± 0.15742
3-UP-MS-MPC, MMOR-switch, N $= 2^{10}$, $t_{\mathrm{SA}} = 1$	1.62224 ± 0.15739
3-UP-MS-MPC, SOBO-switch, N $= 2^{10}$, $t_{\mathrm{SA}} = 5$	1.62199 ± 0.15762
3-UP-MS-MPC, MORR-switch, N $= 2^{10}$, $t_{\mathrm{SA}} = 5$	1.62217 ± 0.15756
3-UP-MS-MPC, MMOR-switch, N $= 2^{10}$, $t_{\mathrm{SA}} = 5$	1.62206 ± 0.15749
3-UP-MS-MPC, SOBO-switch, N $= 2^{12}$, $t_{\mathrm{SA}} = 1$	1.62229 ± 0.15742
3-UP-MS-MPC, MORR-switch, N $= 2^{12}$, $t_{\mathrm{SA}} = 1$	1.62228 ± 0.15741
3-UP-MS-MPC, MMOR-switch, N $= 2^{12}$, $t_{\mathrm{SA}} = 1$	1.62225 ± 0.15738
3-UP-MS-MPC, SOBO-switch, N $= 2^{12}$, $t_{\mathrm{SA}} = 5$	1.62197 ± 0.15765
3-UP-MS-MPC, MORR-switch, N $= 2^{12}$, $t_{\mathrm{SA}} = 5$	1.62218 ± 0.15755
3-UP-MS-MPC, MMOR-switch, N $= 2^{12}$, $t_{\mathrm{SA}} = 5$	1.62206 ± 0.15747

As seen in Table 5.2, the OL-MPC yields the lowest penicillin concentration ($1.62045\,\mathrm{g/l}$) and with the highest deviation ($0.16865\,\mathrm{g/l}$). However, this open-loop approach is rarely used for this kind of process system. The far more common approach is the CL-MPC, which gives the highest concentration ($1.63401\,\mathrm{g/l}$) of the MPCs, and with lower deviation ($0.15934\,\mathrm{g/l}$). This is the standard approach, in which feedback is used in order to counteract the disturbances.

The MS-MPC without SA-based switch (i.e., the parameters Y_{x} and S_{in} are always used in the scenario-tree) proposed by Lucia[11], produces lower penicillin concentration ($1.62107\,\mathrm{g/l}$) than the CL-MPC, but with lower deviation ($0.15700\,\mathrm{g/l}$). Both these values matter for control, but the difference in the deviations is not large enough to compensate for the loss of penicillin. Hence, the CL-MPC outperforms the MS-MPC without the SA-based switch.

All of the MS-MPCs with SA-based switches gave higher penicillin concentration than the MS-MPC without SA-based switch, but with higher deviations. Again, it is beneficial with low deviations so that we can expect the same product concentration each run, but these differences in the deviations do not compensate for the loss of penicillin. Here, the two best cases are the MMOR-switch MS-MPC with $\mathrm{N} = 2^{10}$ and $t_{\mathrm{SA}} = 1$, and the SOBO-switch MS-MPC with $\mathrm{N} = 2^{12}$ and $t_{\mathrm{SA}} = 1$. The former produced $1.62412\,\mathrm{g/l}$ penicillin with $0.15726\,\mathrm{g/l}$ deviation, and the latter produced $1.62413\,\mathrm{g/l}$ penicillin with $0.15730\,\mathrm{g/l}$ deviation. As the latter control comes with more computational costs, i.e., close to four times more, the best case is the former. The Modified Morris screening is a cheap method of GSA, meaning that this MMOR-switch MS-MPC with $\mathrm{N} = 2^{10}$ and $t_{\mathrm{SA}} = 1$ outperforms the MS-MPC without the SA-based switch. It comes at the cost of more model evaluations, but as was proven, it is enough with $\mathrm{N} = 2^{10}$ number of base samples. Hence, combining SA with the MS-MPC might improve the control. In this case study, it gave better results, which was the primary objective for investigation.

However, the standard CL-MPC still outperforms the best MS-MPC with SA-based switch. It gives greater penicillin concentration ($1.63401\,\mathrm{g/l}$ v. $1.62412\,\mathrm{g/l}$), but with higher deviation ($0.15934\,\mathrm{g/l}$ v. $0.15726\,\mathrm{g/l}$). Here, the difference in the deviations is not enough to compensate for the loss of penicillin. Why is it that the MS-MPC gives conservative control? There is not one single answer to this question. However, it seems likely that decreasing the values of the uncertainties in the scenario-tree could give higher penicillin concentration. This might lead to constraint violations, as the plant would have higher uncertainties than the scenario-tree could account for, but it would reduce conservativeness of the control. Another approach that seems more promising, is to use the MS-MPC only when the biomass reaches closer to the constraint. As seen by comparing Figure 5.8 and 5.12, where their axes are equal, the worst-case biomass trajectory of the CL-MPC increases more quickly than for the MS-MPC. Say that we used the CL-MPC until 90 hours of the process, but used the MS-MPC after 90 hours. This would give the highest penicillin concentration, as when using the MS-MPC with SA-based switch there are none constraint violations. In this way, the conservativeness of the MS-MPC vanishes.

Figure 5.34 shows the clear trends in the number of uncertain parameters considered for the scenario-tree, but also in whether applying SA every time-step or every fifth time-step is better. Here, the average product concentrations are plotted against the respective standard deviations, using the data from Table 5.2. A regression line is fitted to the data to visualize the differences. Figure 5.35 is a copy of Figure 5.34, but focuses only on considering two uncertain parameters in the scenario-tree, in which a regression line is fitted to the data to visualize the differences.

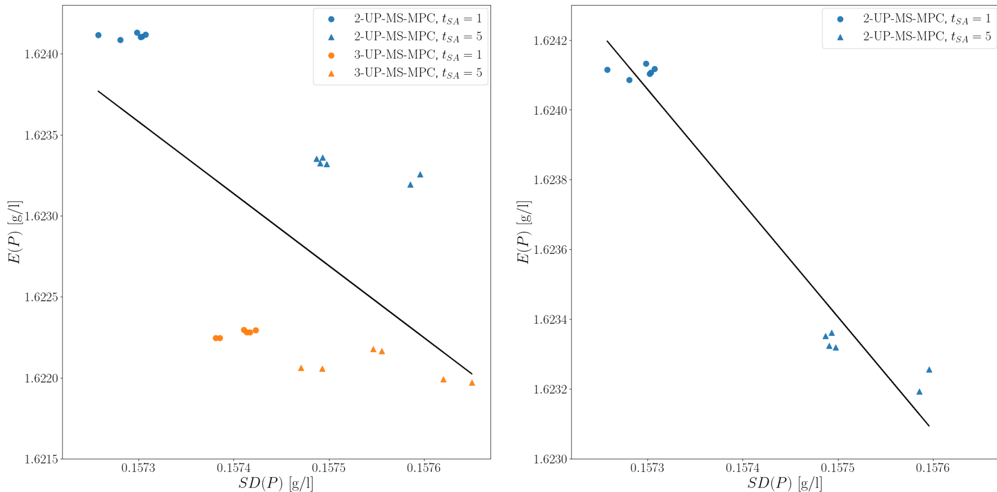


Figure 5.34: Mean versus standard deviation (1/2). **Figure 5.35:** Mean versus standard deviation (2/2).

As seen from Figure 5.34, using two uncertain parameters in the scenario-tree gives higher penicillin concentrations than for using three uncertain parameters, as well as lower deviations. Thus, there are no reasons for using three uncertain parameters. Figure 5.35 proves that using SA every time-step ($t_{\mathrm{SA}} = 1$) gives higher penicillin concentrations than using SA every fifth time-step ($t_{\mathrm{SA}} = 5$), as well as lower deviations. Thus, there are no reasons for using $t_{\mathrm{SA}} = 5$.

Moreover, it was determined that increasing the number of base samples from $\mathrm{N} = 2^{10}$ to $\mathrm{N} = 2^{12}$ has little effect. There was only one case that benefited from this greater sample size. It was rather difficult to find a pattern between the different SA methods. Hence, it makes sense to use either the Morris screening or the Modified Morris screening, as these SA methods offer less computational costs than the Sobol' method.

Are the results presented in this chapter reasonable? A possible source of error is that Ipopt was used as the solver in the NLPs, but CVODES was used as the ODE solver in the integrator. This can result in mismatch between the predicted outputs and simulated outputs, giving small numerical differences. The convergence tolerance and the acceptable tolerance were chosen as $1\mathrm{e}^{-6}$ and $1\mathrm{e}^{-4}$, respectively, for Ipopt, while the absolute tolerance and relative tolerance were $1\mathrm{e}^{-6}$ and $1\mathrm{e}^{-4}$, respectively, for CVODES. These were chosen such that none of the nominal MPCs violated the biomass constraint. However, these tolerances are rather low [35] [9], and can give numerical inaccurate solutions, especially when many decimals matter, which it does for this case study. However, the tolerances were chosen so that we got feasible solutions.

Chapter 6

Conclusion

In this chapter, we leave the concluding remarks on the thesis' work and possible future work.

6.1 Conclusion

In this thesis' theoretical case study, it was produced penicillin of a fermentation process in an isothermal fed-batch bioreactor. The case study was proposed by Srinivasan[33] and then later adapted by Lucia[11] for his studies on the MS-MPC. The main objective of this thesis was to improve Lucia's MS-MPC, in which the parameters Y_x and S_in always were considered as the uncertain parameters in the scenario-tree. However, this assumption was found unreasonable, as there are other parameters sensitive to the biomass concentration. The proposed solution is to use MS-MPC with an SA-based switch, so that the scenario-tree considers the most sensitive parameters in its uncertainty evolution, thus increasing its robustness to the uncertainty.

Three methods of GSA were investigated, that is, the Sobol' method, the Morris screening, and the Modified Morris screening. These methods performed similarly, and it was found that increasing the number of base samples from $\mathrm{N} = 2^{10}$ to $\mathrm{N} = 2^{12}$ had very little effect on how much penicillin that was produced. This is positive, as SA of $\mathrm{N} = 2^{10}$ base samples has close to four times less computational costs than SA of $\mathrm{N} = 2^{12}$ base samples. Moreover, using SA every time-step ($t_\mathrm{SA} = 1$) gave more penicillin than using SA every fifth time-step ($t_\mathrm{SA} = 5$). It was also found that considering two uncertain parameters in the scenario-tree provides more product than considering three uncertain parameters, where the latter gave conservative control. On the other hand, all the cases of the MS-MPC with SA-based switch performed better than the MS-MPC without SA-based switch. The biomass constraint ($\mathrm{X} \leqslant 3.7$) was never violated when using the SA-based switch, but it was for without the switch. Here, it was also produced more penicillin, concluding that combining GSA with MS-MPC offers better control.

However, for the penicillin production there is a trade-off between how quickly the biomass increases and how much its constraint is respected. We want both of the conditions in order to produce the highest penicillin concentration. Out of all the MS-MPCs with SA-based switches, the best performing used the Modified Morris screening with $\mathrm{N} = 2^{10}$ number of base samples every time-step ($t_\mathrm{SA} = 1$), and with two uncertain parameters in the scenario-tree. It was found that the standard CL-MPC produced more penicillin ($1.63401\,\mathrm{g/l}$ v. $1.62412\,\mathrm{g/l}$) than this best

performing MS-MPC, but has higher deviation ($0.15934\,\mathrm{g/l}$ v. $0.15726\,\mathrm{g/l}$). This difference in the deviations is not large enough to compensate for the loss of penicillin. The most promising strategy to resolve this problem, is to use the CL-MPC until the biomass closes on its constraint, and then use the best performing MS-MPC. This would give the fastest increase in the biomass while also satisfying the biomass constraint, thus giving the highest penicillin concentration.

The final conclusion is that using the CL-MPC until the biomass approaches its constraint, then using the best performing MS-MPC with SA-based switch (i.e., two uncertain parameters in the scenario-tree and Modified Morris screening every time-step of $\mathrm{N} = 2^{10}$ base samples), gives the best possible control of the biomass and provides the highest penicillin concentration. This combination of the CL-MPC and MS-MPC with SA-based switch was not implemented in this case study. However, it seems most likely as the best approach based on the results.

6.2 Further work

First and foremost, further work could consider the combination of the CL-MPC and MS-MPC with SA-based switch. This would have been the main priority if time did reach for this thesis. This seems like the most promising approach based on the results of the case study, and it may be applicable for other non-linear systems as well. Further research should also consider other methods of the GSA. As mentioned in Section 3.4, the MCF and FORM/SORM were not used for this case study, but these are promising SA methods with regard to the computational costs. Although the Sobol' method provides more accurate sensitivity measures than compared to the screening-based methods, it did not provide better results. Thus, future research could consider using cost-effective methods of GSA, that do not necessarily have the most accurate measures. Another topic that future work should consider is the tolerances that were given in the solvers. The convergence tolerance and acceptable tolerance were chosen as $1\mathrm{e}^{-6}$ and $1\mathrm{e}^{-4}$ for Ipopt, and the absolute tolerance and relative tolerance were $1\mathrm{e}^{-6}$ and $1\mathrm{e}^{-4}$ for CVODES. However, it is uncertain whether these tolerances provided enough precision for the case study.

6.3 Source criticism

As the final words of the thesis, we evaluate the more questionable sources that were utilized. This includes planeus-solutions [19] on the different applications of the OL-MPC and CL-MPC, Bob's Watches [39] on the water-clock, and Wikipedia on the history of control engineering [40], fermentation [42] and PID-control [41]. However, these sources were mainly for the introductory paragraphs in Chapter 2, 3 and 4, and do not affect the results of the case study. The Wikipedia articles are well cited, but I was not able to obtain the original sources and validate for myself, and thus, Wikipedia was cited instead.

Bibliography

[1] Biegler, L.T., 2010. Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes. Society for Industrial and Applied Mathematics, Philadelphia.

[2] Campolongo, F., Cariboni, J., Saltelli, A., 2007. An effective screening design for sensitivity analysis of large models. Environmental Modelling and Software.

[3] CasADi, 2023. Build efficient optimal control software, with minimal effort. `https://web.casadi.org/`. Last accessed 23 June 2023.

[4] Foss, B., Heirung, T.A.N., 2016. Merging Optimization and Control. Norwegian University of Science and Technology.

[5] van Griensven, A., Meixner, T., Grunwald, S., Bishop, T., Diluzio, M., 2005. A global sensitivity analysis tool for the parameters of multi-variable catchment models. Elsevier.

[6] Grzelak, L.A., Witteveen, J.A.S., Suárez-Taboada, M., Oosterlee, C.W., 2015. The Stochastic Collocation Monte Carlo Sampler: Highly efficient sampling from "expensive" distributions. Delft University of Technology.

[7] van Hoey, S., 2016. Development and application of a framework for model structure evaluation in environmental modelling. Ghent University, Faculty of Bioscience Engineering.

[8] Ipopt, 2023a. Ipopt documentation. `https://coin-or.github.io/Ipopt/`. Last accessed 23 June 2023.

[9] Ipopt, 2023b. Ipopt options. `https://coin-or.github.io/Ipopt/OPTIONS.html`. Last accessed 04 July 2023.

[10] Krishnamoorthy, D., Suwartadi, E., Foss, B., Skogestad, S., Jäschke, J., 2018. Improving Scenario Decomposition for Multistage MPC using a Sensitivity-based Path following Algorithm. Norwegian University of Science and Technology (NTNU).

[11] Lucia, S., Engell, S., 2013. Robust Nonlinear Model Predictive Control of a Batch Bioreactor Using Multi-stage Stochastic Programming. European Control Conference (ECC).

[12] Lucia, S., Fiedler, F., 2023a. Basics of model predictive control. `https://www.do-mpc.com/en/latest/theory_mpc.html`. Last accessed 12 June 2023.

[13] Lucia, S., Fiedler, F., 2023b. Orthogonal collocation on finite elements. https://www.do-mpc.com/en/latest/theory_orthogonal_collocation.html. Last accessed 12 June 2023.

[14] Lucia, S., Finkler, T., Engell, S., 2013. Multi-stage nonlinear model predictive control applied to a semi-batch polymerization reactor under uncertainty. Elsevier.

[15] Løvdokken, H., 2022. Combining robust model predictive control with sensitivity analysis. Norwegian University of Science and Technology.

[16] Matplotlib, 2023. matplotlib.pyplot. https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html. Last accessed 24 June 2023.

[17] Morris, M.D., 1991. Factorial Sampling Plans for Preliminary Computational Experiments. American Statistical Association and the American Society of Quality Control.

[18] NumPy, 2023. The fundamental package for scientific computing with python. https://numpy.org/. Last accessed 23 June 2023.

[19] Planeus, 2021. Closed-loop vs. open-loop production control: Examples and differences. https://planeus-solutions.com/blog/en/closed-loop-vs-open-loop-production-control-system/. Last accessed 09 June 2023.

[20] Qian, G., Mahdi, A., 2020. Sensitivity analysis methods in the biomedical sciences. Elsevier.

[21] Qin, S.J., Badgwell, T.A., 2003. A survey of industrial model predictive control technology. Elsevier.

[22] Rawlings, J.B., Mayne, D.Q., Diehl, M.M., 2022. Model Predictive Control: Theory, Computation, and Design. Nob Hill Publishing.

[23] Razavi, S., Jakeman, A., Saltelli, A., Prieur, C., Iooss, B., Borgonovo, E., Plischke, E., Piano, S., Iwanaga, T., Beckeri, W., Tarantola, S., Guillaume, J., Jakeman, J., Gupta, H., Melillo, N., Rabitti, G., Chabridon, V., Duan, Q., Sun, X., Smith, S., Sheikholeslami, R., Hosseini, N., Asadzadeh, M., Puy, A., Kucherenko, S., Maier, H., 2020. The Future of Sensitivity Analysis: An essential discipline for systems modeling and policy support. Elsevier.

[24] SALib, 2023. Frequently asked questions. https://salib.readthedocs.io/en/latest/user_guide/faq.html. Last accessed 29 June 2023.

[25] Saltelli, A., Chan, K., Scott, E.M., 2000. Sensitivity Analysis: Gauging the Worth of Scientific Models. Wiley.

[26] Saltelli, A., Jakeman, A., Razavi, S., Wu, Q., 2021. Sensitivity analysis: A discipline coming of age. Elsevier.

[27] Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., Tarantola, S., 2008. Global Sensitivity Analysis. Wiley.

[28] Saltelli, A., Tarantola, S., Campolongo, F., Ratto, M., 2004. Sensitivity Analysis in Practise: A Guide to Assessing Scientific Models. Wiley.

[29] SciPy, 2023a. Fundamental algorithms for scientific computing in python. https://scipy.org/. Last accessed 23 June 2023.

[30] SciPy, 2023b. scipy.stats.qmc.latinhypercube. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.qmc.LatinHypercube.html. Last accessed 24 June 2023.

[31] Seborg, D.E., Edgar, T.F., Mellichamp, D.A., III, F.J.D., 2017. Process Dynamics and Control. Wiley.

[32] Sobol, I.M., 1990. Sensitivity estimates for nonlinear mathematical models. Matematicheskoe Modelirovanie.

[33] Srinivasan, B., Bonvin, D., Visser, E., Palanki, S., 2002. Dynamic optimization of batch processes II. Role of measurements in handling uncertainty. Matematicheskoe Modelirovanie.

[34] SUNDIALS, 2023a. Cvodes. https://computing.llnl.gov/projects/sundials/cvodes. Last accessed 24 June 2023.

[35] SUNDIALS, 2023b. Using cvode for ivp solution. https://sundials.readthedocs.io/en/latest/cvode/Usage/index.html. Last accessed 24 June 2023.

[36] The Alexander Fleming Laboratory Museum, London, U., 1999. The discovery and development of penicillin. American Chemical Society and Royal Society of Chemistry.

[37] Wang, Z., Ierapetritou, M., 2018. Global sensitivity, feasibility, and flexibility analysis of continuous pharmaceutical manufacturing processes. Elsevier.

[38] Waskom, M., 2023. seaborn.swarmplot. https://seaborn.pydata.org/generated/seaborn.swarmplot.html. Last accessed 24 June 2023.

[39] Watches, B., 2023. History of watches: The water clock. https://www.bobswatches.com/water-clock.html. Last accessed 16 June 2023.

[40] Wikipedia, 2023a. Control engineering. https://en.wikipedia.org/wiki/Control_engineering. Last accessed 16 June 2023.

[41] Wikipedia, 2023b. Fermentation. https://en.wikipedia.org/wiki/Fermentation. Last accessed 20 June 2023.

[42] Wikipedia, 2023c. Pid controller. https://en.wikipedia.org/wiki/PID_controller. Last accessed 17 June 2023.

[43] Yu, Z., Biegler, L.T., 2020. Sensitivity-assisted Robust Nonlinear Model Predictive Control with Scenario Generation. Preprints of the 21st IFAC World Congress.

Appendix - code listings

The main-loop for all the MPCs

```python
import os as os
import time as tm
import numpy as np
import pathlib as pathlib
import matplotlib.pyplot as plt
from matplotlib import rcParams
from optimiz import optimzr_nmpc
from optimiz import optimzr_rmpc2
from optimiz import optimzr_rmpc3
from optimiz import solver_nmpc
from optimiz import solver_rmpc2
from optimiz import solver_rmpc3
from process import integrate
from process import simulator
from sensitiv import unifrm_sample
from sensitiv import modify_method
from sensitiv import morris_method
from sensitiv import sobols_method
from sensitiv import switch_rule2
from sensitiv import switch_rule3

u0 = np.array([0.000])  # -- u0; Inlet flow-rate [(m^{3}/hr)]

x0 = np.array([1.000,  # -- X0; Biomass concentration [(g/l)]
               0.500,  # -- S0; Substr. concentration [(g/l)]
               0.000,  # -- P0; Product concentration [(g/l)]
               120.0])  # - V0; Volume inside reactor [(m^3)]

p0 = np.array([0.020,  # -- mu_m0; Kinetic parameter [(unit)]
               0.050,  # -- K_m_0; Kinetic parameter [(unit)]
               5.000,  # -- K_i_0; Kinetic parameter [(unit)]
               0.004,  # -- nu__0; Kinetic parameter [(unit)]
               1.200,  # -- Y_p_0; Prod.-yield coef. [(unit)]
               0.400,  # -- Y_x_0; Biom.-yield coef. [(unit)]
               200.0])  # - S_in0; Inlet subs. conc. [(unit)]

ts = np.linspace(0., 150., 151)  # Time-axis MPC-plot [(hr)]
tz = np.linspace(0., 1., 2)  # Time-axis SA-true-plot [(hr)]

dim_x = x0.shape[0]  # x0-state dimension
dim_p = p0.shape[0]  # p0-param dimension
dim_u = u0.shape[0]  # u0-input dimension
dim_ts = ts.shape[0]  # ts-time dimension
dim_tz = tz.shape[0]  # tz-time dimension

N = 1  # how many times the loop runs
f = integrate()  # obtain integrate()
x_opts = np.zeros((dim_x, dim_ts, N))
u_opts = np.zeros((dim_u, dim_ts, N))
t_calc = np.zeros((dim_u, dim_ts, N))

# Sensitivity with the Sobol' method
s1_inds = np.zeros((dim_p, dim_ts, N))
st_inds = np.zeros((dim_p, dim_ts, N))

# Sensitivity with the Morris method
ee_abss = np.zeros((dim_p, dim_ts, N))
ee_stds = np.zeros((dim_p, dim_ts, N))

# Sensitivity with the Modify method
pe_abss = np.zeros((dim_p, dim_ts, N))
pe_stds = np.zeros((dim_p, dim_ts, N))

# Save/load parameters for the loops
p0_save, p0_load = False, True
p_cnst = np.zeros((dim_p, dim_ts, N))
p_flct = np.zeros((dim_p, dim_ts, N))

# Save/load parameters for the loops
proj_dir = pathlib.Path(__file__).parent.parent
data_dir = os.path.join(proj_dir, "data/params")
fpath1 = os.path.join(data_dir, "p_cnst.npy")
fpath2 = os.path.join(data_dir, "p_flct.npy")

if p0_load:
    p_cnst = np.load(fpath1)
    p_flct = np.load(fpath2)
else:
    for i in range(N):
        p_cnst[:, 0, i] = unifrm_sample(p0)
        p_flct[:, 0, i] = unifrm_sample(p0)
```

```
82          for k in range(1, dim_ts):
83              p_cnst[:, k, i] = p_cnst[:, 0, i]
84              p_flct[:, k, i] = unifrm_sample(p0)
85      if p0_save:
86          np.save(fpath1, p_cnst)
87          np.save(fpath2, p_flct)
88
89  # Boolean deciding if we should run the Sobol' method
90  run_SA1 = False
91  # Boolean deciding if we should run the Morris method
92  run_SA2 = False
93  # Boolean deciding if we should run the Modify method
94  run_SA3 = False
95
96  # Boolean deciding if we should plot the figure 'fig0'
97  plot_figure_0 = False
98  # Boolean deciding if we should plot the figure 'fig1'
99  plot_figure_1 = False
100 # Boolean deciding if we should plot the figure 'fig2'
101 plot_figure_2 = False
102 # Boolean deciding if we should plot the figure 'fig3'
103 plot_figure_3 = False
104 # Boolean deciding if we should plot the figure 'fig4'
105 plot_figure_4 = False
106 # Boolean deciding if we should plot the figure 'fig5'
107 plot_figure_5 = False
108 # Boolean deciding if we should plot the figure 'fig6'
109 plot_figure_6 = False
110 # Boolean deciding if we should plot the figure 'fig7'
111 plot_figure_7 = False
112 # Boolean deciding if we should plot the figure 'fig8'
113 plot_figure_8 = False
114 # Boolean deciding if we should plot the figure 'fig9'
115 plot_figure_9 = False
116
117 # Open loop normal mpc with 150 predictions 150 controls
118 open_loop_nmpc = True
119 # Closed loop normal mpc with 20 predictions 03 controls
120 clsd_loop_nmpc = False
121 # Closed loop robust mpc with Y_x S_in as the parameters
122 clsd_loop_rmpc1 = False
123 # Closed loop robust mpc with SA; 2 uncertain parameters
124 clsd_loop_rmpc2 = False
125 # Closed loop robust mpc with SA; 3 uncertain parameters
126 clsd_loop_rmpc3 = False
127
128 if open_loop_nmpc:
129     trajects01, solver01, w0_conv01, lbw_conv01, ubw_conv01, lbg01, ubg01 = solver_nmpc(150, 150)
130     nlp01 = (trajects01, solver01, w0_conv01, lbw_conv01, ubw_conv01, lbg01, ubg01)  # (150, 150)
131
132 elif clsd_loop_nmpc:
133     trajects01, solver01, w0_conv01, lbw_conv01, ubw_conv01, lbg01, ubg01 = solver_nmpc(20, 3)
134     nlp01 = (trajects01, solver01, w0_conv01, lbw_conv01, ubw_conv01, lbg01, ubg01)  # (20, 3)
135
136 elif clsd_loop_rmpc1:
137     # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "Y_x":
138     trajects21, solver21, w0_conv21, lbw_conv21, ubw_conv21, lbg21, ubg21 = solver_rmpc2("Y_x", "S_in")
139     nlp21 = (trajects21, solver21, w0_conv21, lbw_conv21, ubw_conv21, lbg21, ubg21)  # ("Y_x", "S_in")
140
141 elif clsd_loop_rmpc2:
142     # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "mu_m":
143     trajects01, solver01, w0_conv01, lbw_conv01, ubw_conv01, lbg01, ubg01 = solver_rmpc2("mu_m", "K_m")
144     nlp01 = (trajects01, solver01, w0_conv01, lbw_conv01, ubw_conv01, lbg01, ubg01)  # ("mu_m", "K_m")
145     trajects02, solver02, w0_conv02, lbw_conv02, ubw_conv02, lbg02, ubg02 = solver_rmpc2("mu_m", "K_i")
146     nlp02 = (trajects02, solver02, w0_conv02, lbw_conv02, ubw_conv02, lbg02, ubg02)  # ("mu_m", "K_i")
147     trajects03, solver03, w0_conv03, lbw_conv03, ubw_conv03, lbg03, ubg03 = solver_rmpc2("mu_m", "nu")
148     nlp03 = (trajects03, solver03, w0_conv03, lbw_conv03, ubw_conv03, lbg03, ubg03)  # ("mu_m", "nu")
149     trajects04, solver04, w0_conv04, lbw_conv04, ubw_conv04, lbg04, ubg04 = solver_rmpc2("mu_m", "Y_p")
150     nlp04 = (trajects04, solver04, w0_conv04, lbw_conv04, ubw_conv04, lbg04, ubg04)  # ("mu_m", "Y_p")
151     trajects05, solver05, w0_conv05, lbw_conv05, ubw_conv05, lbg05, ubg05 = solver_rmpc2("mu_m", "Y_x")
152     nlp05 = (trajects05, solver05, w0_conv05, lbw_conv05, ubw_conv05, lbg05, ubg05)  # ("mu_m", "Y_x")
153     trajects06, solver06, w0_conv06, lbw_conv06, ubw_conv06, lbg06, ubg06 = solver_rmpc2("mu_m", "S_in")
154     nlp06 = (trajects06, solver06, w0_conv06, lbw_conv06, ubw_conv06, lbg06, ubg06)  # ("mu_m", "S_in")
155
156     # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "K_m":
157     trajects07, solver07, w0_conv07, lbw_conv07, ubw_conv07, lbg07, ubg07 = solver_rmpc2("K_m", "K_i")
158     nlp07 = (trajects07, solver07, w0_conv07, lbw_conv07, ubw_conv07, lbg07, ubg07)  # ("K_m", "K_i")
159     trajects08, solver08, w0_conv08, lbw_conv08, ubw_conv08, lbg08, ubg08 = solver_rmpc2("K_m", "nu")
160     nlp08 = (trajects08, solver08, w0_conv08, lbw_conv08, ubw_conv08, lbg08, ubg08)  # ("K_m", "nu")
161     trajects09, solver09, w0_conv09, lbw_conv09, ubw_conv09, lbg09, ubg09 = solver_rmpc2("K_m", "Y_p")
162     nlp09 = (trajects09, solver09, w0_conv09, lbw_conv09, ubw_conv09, lbg09, ubg09)  # ("K_m", "Y_p")
163     trajects10, solver10, w0_conv10, lbw_conv10, ubw_conv10, lbg10, ubg10 = solver_rmpc2("K_m", "Y_x")
164     nlp10 = (trajects10, solver10, w0_conv10, lbw_conv10, ubw_conv10, lbg10, ubg10)  # ("K_m", "Y_x")
165     trajects11, solver11, w0_conv11, lbw_conv11, ubw_conv11, lbg11, ubg11 = solver_rmpc2("K_m", "S_in")
166     nlp11 = (trajects11, solver11, w0_conv11, lbw_conv11, ubw_conv11, lbg11, ubg11)  # ("K_m", "S_in")
167
168     # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "K_i":
169     trajects12, solver12, w0_conv12, lbw_conv12, ubw_conv12, lbg12, ubg12 = solver_rmpc2("K_i", "nu")
170     nlp12 = (trajects12, solver12, w0_conv12, lbw_conv12, ubw_conv12, lbg12, ubg12)  # ("K_i", "nu")
171     trajects13, solver13, w0_conv13, lbw_conv13, ubw_conv13, lbg13, ubg13 = solver_rmpc2("K_i", "Y_p")
172     nlp13 = (trajects13, solver13, w0_conv13, lbw_conv13, ubw_conv13, lbg13, ubg13)  # ("K_i", "Y_p")
173     trajects14, solver14, w0_conv14, lbw_conv14, ubw_conv14, lbg14, ubg14 = solver_rmpc2("K_i", "Y_x")
174     nlp14 = (trajects14, solver14, w0_conv14, lbw_conv14, ubw_conv14, lbg14, ubg14)  # ("K_i", "Y_x")
175     trajects15, solver15, w0_conv15, lbw_conv15, ubw_conv15, lbg15, ubg15 = solver_rmpc2("K_i", "S_in")
176     nlp15 = (trajects15, solver15, w0_conv15, lbw_conv15, ubw_conv15, lbg15, ubg15)  # ("K_i", "S_in")
```

```
177
178     # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "nu":
179     trajects16, solver16, w0_conv16, lbw_conv16, ubw_conv16, lbg16, ubg16 = solver_rmpc2("nu", "Y_p")
180     nlp16 = (trajects16, solver16, w0_conv16, lbw_conv16, ubw_conv16, lbg16, ubg16)  # ("nu", "Y_p")
181     trajects17, solver17, w0_conv17, lbw_conv17, ubw_conv17, lbg17, ubg17 = solver_rmpc2("nu", "Y_x")
182     nlp17 = (trajects17, solver17, w0_conv17, lbw_conv17, ubw_conv17, lbg17, ubg17)  # ("nu", "Y_x")
183     trajects18, solver18, w0_conv18, lbw_conv18, ubw_conv18, lbg18, ubg18 = solver_rmpc2("nu", "S_in")
184     nlp18 = (trajects18, solver18, w0_conv18, lbw_conv18, ubw_conv18, lbg18, ubg18)  # ("nu", "S_in")
185
186     # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "Y_p":
187     trajects19, solver19, w0_conv19, lbw_conv19, ubw_conv19, lbg19, ubg19 = solver_rmpc2("Y_p", "Y_x")
188     nlp19 = (trajects19, solver19, w0_conv19, lbw_conv19, ubw_conv19, lbg19, ubg19)  # ("Y_p", "Y_x")
189     trajects20, solver20, w0_conv20, lbw_conv20, ubw_conv20, lbg20, ubg20 = solver_rmpc2("Y_p", "S_in")
190     nlp20 = (trajects20, solver20, w0_conv20, lbw_conv20, ubw_conv20, lbg20, ubg20)  # ("Y_p", "S_in")
191
192     # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "Y_x":
193     trajects21, solver21, w0_conv21, lbw_conv21, ubw_conv21, lbg21, ubg21 = solver_rmpc2("Y_x", "S_in")
194     nlp21 = (trajects21, solver21, w0_conv21, lbw_conv21, ubw_conv21, lbg21, ubg21)  # ("Y_x", "S_in")
195
196 elif clsd_loop_rmpc3:
197     # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "mu_m", "K_m":
198     trajects01, solver01, w0_conv01, lbw_conv01, ubw_conv01, lbg01, ubg01 = solver_rmpc3("mu_m", "K_m", "K_i")
199     nlp01 = (trajects01, solver01, w0_conv01, lbw_conv01, ubw_conv01, lbg01, ubg01)  # ("mu_m", "K_m", "K_i")
200     trajects02, solver02, w0_conv02, lbw_conv02, ubw_conv02, lbg02, ubg02 = solver_rmpc3("mu_m", "K_m", "nu")
201     nlp02 = (trajects02, solver02, w0_conv02, lbw_conv02, ubw_conv02, lbg02, ubg02)  # ("mu_m", "K_m", "nu")
202     trajects03, solver03, w0_conv03, lbw_conv03, ubw_conv03, lbg03, ubg03 = solver_rmpc3("mu_m", "K_m", "Y_p")
203     nlp03 = (trajects03, solver03, w0_conv03, lbw_conv03, ubw_conv03, lbg03, ubg03)  # ("mu_m", "K_m", "Y_p")
204     trajects04, solver04, w0_conv04, lbw_conv04, ubw_conv04, lbg04, ubg04 = solver_rmpc3("mu_m", "K_m", "Y_x")
205     nlp04 = (trajects04, solver04, w0_conv04, lbw_conv04, ubw_conv04, lbg04, ubg04)  # ("mu_m", "K_m", "Y_x")
206     trajects05, solver05, w0_conv05, lbw_conv05, ubw_conv05, lbg05, ubg05 = solver_rmpc3("mu_m", "K_m", "S_in")
207     nlp05 = (trajects05, solver05, w0_conv05, lbw_conv05, ubw_conv05, lbg05, ubg05)  # ("mu_m", "K_m", "S_in")
208
209     # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "mu_m", "K_i":
210     trajects06, solver06, w0_conv06, lbw_conv06, ubw_conv06, lbg06, ubg06 = solver_rmpc3("mu_m", "K_i", "nu")
211     nlp06 = (trajects06, solver06, w0_conv06, lbw_conv06, ubw_conv06, lbg06, ubg06)  # ("mu_m", "K_i", "nu")
212     trajects07, solver07, w0_conv07, lbw_conv07, ubw_conv07, lbg07, ubg07 = solver_rmpc3("mu_m", "K_i", "Y_p")
213     nlp07 = (trajects07, solver07, w0_conv07, lbw_conv07, ubw_conv07, lbg07, ubg07)  # ("mu_m", "K_i", "Y_p")
214     trajects08, solver08, w0_conv08, lbw_conv08, ubw_conv08, lbg08, ubg08 = solver_rmpc3("mu_m", "K_i", "Y_x")
215     nlp08 = (trajects08, solver08, w0_conv08, lbw_conv08, ubw_conv08, lbg08, ubg08)  # ("mu_m", "K_i", "Y_x")
216     trajects09, solver09, w0_conv09, lbw_conv09, ubw_conv09, lbg09, ubg09 = solver_rmpc3("mu_m", "K_i", "S_in")
217     nlp09 = (trajects09, solver09, w0_conv09, lbw_conv09, ubw_conv09, lbg09, ubg09)  # ("mu_m", "K_i", "S_in")
218
219     # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "mu_m", "nu":
220     trajects10, solver10, w0_conv10, lbw_conv10, ubw_conv10, lbg10, ubg10 = solver_rmpc3("mu_m", "nu", "Y_p")
221     nlp10 = (trajects10, solver10, w0_conv10, lbw_conv10, ubw_conv10, lbg10, ubg10)  # ("mu_m", "nu", "Y_p")
222     trajects11, solver11, w0_conv11, lbw_conv11, ubw_conv11, lbg11, ubg11 = solver_rmpc3("mu_m", "nu", "Y_x")
223     nlp11 = (trajects11, solver11, w0_conv11, lbw_conv11, ubw_conv11, lbg11, ubg11)  # ("mu_m", "nu", "Y_x")
224     trajects12, solver12, w0_conv12, lbw_conv12, ubw_conv12, lbg12, ubg12 = solver_rmpc3("mu_m", "nu", "S_in")
225     nlp12 = (trajects12, solver12, w0_conv12, lbw_conv12, ubw_conv12, lbg12, ubg12)  # ("mu_m", "nu", "S_in")
226
227     # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "mu_m", "Y_p":
228     trajects13, solver13, w0_conv13, lbw_conv13, ubw_conv13, lbg13, ubg13 = solver_rmpc3("mu_m", "Y_p", "Y_x")
229     nlp13 = (trajects13, solver13, w0_conv13, lbw_conv13, ubw_conv13, lbg13, ubg13)  # ("mu_m", "Y_p", "Y_x")
230     trajects14, solver14, w0_conv14, lbw_conv14, ubw_conv14, lbg14, ubg14 = solver_rmpc3("mu_m", "Y_p", "S_in")
231     nlp14 = (trajects14, solver14, w0_conv14, lbw_conv14, ubw_conv14, lbg14, ubg14)  # ("mu_m", "Y_p", "S_in")
232
233     # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "mu_m", "Y_x":
234     trajects15, solver15, w0_conv15, lbw_conv15, ubw_conv15, lbg15, ubg15 = solver_rmpc3("mu_m", "Y_x", "S_in")
235     nlp15 = (trajects15, solver15, w0_conv15, lbw_conv15, ubw_conv15, lbg15, ubg15)  # ("mu_m", "Y_x", "S_in")
236
237     # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "K_m", "K_i":
238     trajects16, solver16, w0_conv16, lbw_conv16, ubw_conv16, lbg16, ubg16 = solver_rmpc3("K_m", "K_i", "nu")
239     nlp16 = (trajects16, solver16, w0_conv16, lbw_conv16, ubw_conv16, lbg16, ubg16)  # ("K_m", "K_i", "nu")
240     trajects17, solver17, w0_conv17, lbw_conv17, ubw_conv17, lbg17, ubg17 = solver_rmpc3("K_m", "K_i", "Y_p")
241     nlp17 = (trajects17, solver17, w0_conv17, lbw_conv17, ubw_conv17, lbg17, ubg17)  # ("K_m", "K_i", "Y_p")
242     trajects18, solver18, w0_conv18, lbw_conv18, ubw_conv18, lbg18, ubg18 = solver_rmpc3("K_m", "K_i", "Y_x")
243     nlp18 = (trajects18, solver18, w0_conv18, lbw_conv18, ubw_conv18, lbg18, ubg18)  # ("K_m", "K_i", "Y_x")
244     trajects19, solver19, w0_conv19, lbw_conv19, ubw_conv19, lbg19, ubg19 = solver_rmpc3("K_m", "K_i", "S_in")
245     nlp19 = (trajects19, solver19, w0_conv19, lbw_conv19, ubw_conv19, lbg19, ubg19)  # ("K_m", "K_i", "S_in")
246
247     # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "K_m", "nu":
248     trajects20, solver20, w0_conv20, lbw_conv20, ubw_conv20, lbg20, ubg20 = solver_rmpc3("K_m", "nu", "Y_p")
249     nlp20 = (trajects20, solver20, w0_conv20, lbw_conv20, ubw_conv20, lbg20, ubg20)  # ("K_m", "nu", "Y_p")
250     trajects21, solver21, w0_conv21, lbw_conv21, ubw_conv21, lbg21, ubg21 = solver_rmpc3("K_m", "nu", "Y_x")
251     nlp21 = (trajects21, solver21, w0_conv21, lbw_conv21, ubw_conv21, lbg21, ubg21)  # ("K_m", "nu", "Y_x")
252     trajects22, solver22, w0_conv22, lbw_conv22, ubw_conv22, lbg22, ubg22 = solver_rmpc3("K_m", "nu", "S_in")
253     nlp22 = (trajects22, solver22, w0_conv22, lbw_conv22, ubw_conv22, lbg22, ubg22)  # ("K_m", "nu", "S_in")
254
255     # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "K_m", "Y_p":
256     trajects23, solver23, w0_conv23, lbw_conv23, ubw_conv23, lbg23, ubg23 = solver_rmpc3("K_m", "Y_p", "Y_x")
257     nlp23 = (trajects23, solver23, w0_conv23, lbw_conv23, ubw_conv23, lbg23, ubg23)  # ("K_m", "Y_p", "Y_x")
258     trajects24, solver24, w0_conv24, lbw_conv24, ubw_conv24, lbg24, ubg24 = solver_rmpc3("K_m", "Y_p", "S_in")
259     nlp24 = (trajects24, solver24, w0_conv24, lbw_conv24, ubw_conv24, lbg24, ubg24)  # ("K_m", "Y_p", "S_in")
260
261     # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "K_m", "Y_x":
262     trajects25, solver25, w0_conv25, lbw_conv25, ubw_conv25, lbg25, ubg25 = solver_rmpc3("K_m", "Y_x", "S_in")
263     nlp25 = (trajects25, solver25, w0_conv25, lbw_conv25, ubw_conv25, lbg25, ubg25)  # ("K_m", "Y_x", "S_in")
264
265     # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "K_i", "nu":
266     trajects26, solver26, w0_conv26, lbw_conv26, ubw_conv26, lbg26, ubg26 = solver_rmpc3("K_i", "nu", "Y_p")
267     nlp26 = (trajects26, solver26, w0_conv26, lbw_conv26, ubw_conv26, lbg26, ubg26)  # ("K_i", "nu", "Y_p")
268     trajects27, solver27, w0_conv27, lbw_conv27, ubw_conv27, lbg27, ubg27 = solver_rmpc3("K_i", "nu", "Y_x")
269     nlp27 = (trajects27, solver27, w0_conv27, lbw_conv27, ubw_conv27, lbg27, ubg27)  # ("K_i", "nu", "Y_x")
270     trajects28, solver28, w0_conv28, lbw_conv28, ubw_conv28, lbg28, ubg28 = solver_rmpc3("K_i", "nu", "S_in")
271     nlp28 = (trajects28, solver28, w0_conv28, lbw_conv28, ubw_conv28, lbg28, ubg28)  # ("K_i", "nu", "S_in")
```

```python

    # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "K_i", "Y_p":
    trajects29, solver29, w0_conv29, lbw_conv29, ubw_conv29, lbg29, ubg29 = solver_rmpc3("K_i", "Y_p", "Y_x")
    nlp29 = (trajects29, solver29, w0_conv29, lbw_conv29, ubw_conv29, lbg29, ubg29)  # ("K_i", "Y_p", "Y_x")
    trajects30, solver30, w0_conv30, lbw_conv30, ubw_conv30, lbg30, ubg30 = solver_rmpc3("K_i", "Y_p", "S_in")
    nlp30 = (trajects30, solver30, w0_conv30, lbw_conv30, ubw_conv30, lbg30, ubg30)  # ("K_i", "Y_p", "S_in")

    # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "K_i", "Y_x":
    trajects31, solver31, w0_conv31, lbw_conv31, ubw_conv31, lbg31, ubg31 = solver_rmpc3("K_i", "Y_x", "S_in")
    nlp31 = (trajects31, solver31, w0_conv31, lbw_conv31, ubw_conv31, lbg31, ubg31)  # ("K_i", "Y_x", "S_in")

    # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "nu", "Y_p":
    trajects32, solver32, w0_conv32, lbw_conv32, ubw_conv32, lbg32, ubg32 = solver_rmpc3("nu", "Y_p", "Y_x")
    nlp32 = (trajects32, solver32, w0_conv32, lbw_conv32, ubw_conv32, lbg32, ubg32)  # ("nu", "Y_p", "Y_x")
    trajects33, solver33, w0_conv33, lbw_conv33, ubw_conv33, lbg33, ubg33 = solver_rmpc3("nu", "Y_p", "S_in")
    nlp33 = (trajects33, solver33, w0_conv33, lbw_conv33, ubw_conv33, lbg33, ubg33)  # ("nu", "Y_p", "S_in")

    # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "nu", "Y_x":
    trajects34, solver34, w0_conv34, lbw_conv34, ubw_conv34, lbg34, ubg34 = solver_rmpc3("nu", "Y_x", "S_in")
    nlp34 = (trajects34, solver34, w0_conv34, lbw_conv34, ubw_conv34, lbg34, ubg34)  # ("nu", "Y_x", "S_in")

    # Obtain {trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg} for the ms-mpc's with "Y_p", "Y_x":
    trajects35, solver35, w0_conv35, lbw_conv35, ubw_conv35, lbg35, ubg35 = solver_rmpc3("Y_p", "Y_x", "S_in")
    nlp35 = (trajects35, solver35, w0_conv35, lbw_conv35, ubw_conv35, lbg35, ubg35)  # ("Y_p", "Y_x", "S_in")

if open_loop_nmpc:
    for i in range(N):
        w_opt = np.array([])
        switch = np.array([0, 0])
        x_opts[:, 0, i] = x0.flatten()
        u_opts[:, 0, i] = u0.flatten()
        for k in range(1, dim_ts):
            t_iter = tm.time()
            tk = np.array([ts[k - 1], ts[k]])
            xk = np.array(x_opts[:, k - 1, i])
            uk = np.array(u_opts[:, k - 1, i])
            pk = np.array(p_cnst[:, k - 1, i])
            if k == 1:  # Open-loop-mpc; we run the optimization only once
                u_opt, w_opt = optimzr_nmpc(*nlp01, xk, tk, uk, p0, w_opt)
            u_optk = np.array([u_opt[0, k - 1]]).flatten()
            u_optz = np.array([u_opt[0, k - 1:]]).flatten()
            x_opt = simulator(f, xk, tk, u_optk, p0)
            x_optk = np.array([x_opt]).flatten()
            t_iter = tm.time() - t_iter
            x_opts[:, k, i] = x_optk
            u_opts[:, k, i] = u_optk
            t_calc[:, k, i] = t_iter
            print(f"Computed iteration: {k}")
            print(f"Computer-time: {t_iter}")
            print(f"Switch-rule:   {switch}")

elif clsd_loop_nmpc:
    for i in range(N):
        w_opt = np.array([])
        switch = np.array([0, 0])
        x_opts[:, 0, i] = x0.flatten()
        u_opts[:, 0, i] = u0.flatten()
        for k in range(1, dim_ts):
            t_iter = tm.time()
            tk = np.array([ts[k - 1], ts[k]])
            xk = np.array(x_opts[:, k - 1, i])
            uk = np.array(u_opts[:, k - 1, i])
            pk = np.array(p_cnst[:, k - 1, i])
            if k == k:  # Closed-loop-mpc; we run the optimizer every loop
                u_opt, w_opt = optimzr_nmpc(*nlp01, xk, tk, uk, p0, w_opt)
            u_optk = np.array([u_opt[0, 0]]).flatten()
            u_optz = np.array([u_opt[0, 1:]]).flatten()
            x_opt = simulator(f, xk, tk, u_optk, p0)
            x_optk = np.array([x_opt]).flatten()

            if (k - 1) % (dim_tz - 1) == 0:
                s1_ind = np.zeros((dim_p, dim_tz - 1))
                st_ind = np.zeros((dim_p, dim_tz - 1))
                ee_abs = np.zeros((dim_p, dim_tz - 1))
                ee_std = np.zeros((dim_p, dim_tz - 1))
                pe_abs = np.zeros((dim_p, dim_tz - 1))
                pe_std = np.zeros((dim_p, dim_tz - 1))

                if k >= 1 and run_SA1:
                    s1_ind, st_ind = sobols_method(f, x_optk, tz, u_optz[:(dim_tz - 1)], p0)
                if k >= 1 and run_SA2:
                    ee_abs, ee_std = morris_method(f, x_optk, tz, u_optz[:(dim_tz - 1)], p0)
                if k >= 1 and run_SA3:
                    pe_abs, pe_std = modify_method(f, x_optk, tz, u_optz[:(dim_tz - 1)], p0)

                for j in range(dim_tz - 1):
                    if (k + 1 + j) > (dim_ts - 1):
                        break
                    s1_inds[:, k + 1 + j, i] = s1_ind[:, j].flatten()
                    st_inds[:, k + 1 + j, i] = st_ind[:, j].flatten()
                    ee_abss[:, k + 1 + j, i] = ee_abs[:, j].flatten()
                    ee_stds[:, k + 1 + j, i] = ee_std[:, j].flatten()
                    pe_abss[:, k + 1 + j, i] = pe_abs[:, j].flatten()
                    pe_stds[:, k + 1 + j, i] = pe_std[:, j].flatten()
            t_iter = tm.time() - t_iter
```

```
367                 x_opts[:, k, i] = x_optk
368                 u_opts[:, k, i] = u_optk
369                 t_calc[:, k, i] = t_iter
370                 print(f"Computed iteration: {k}")
371                 print(f"Computer-time: {t_iter}")
372                 print(f"Switch-rule:   {switch}")
373 
374 elif clsd_loop_rmpc1:
375     for i in range(N):
376         w_opt = np.array([])
377         switch = np.array([0, 0])
378         x_opts[:, 0, i] = x0.flatten()
379         u_opts[:, 0, i] = u0.flatten()
380         for k in range(1, dim_ts):
381             t_iter = tm.time()
382             tk = np.array([ts[k - 1], ts[k]])
383             xk = np.array(x_opts[:, k - 1, i])
384             uk = np.array(u_opts[:, k - 1, i])
385             pk = np.array(p_cnst[:, k - 1, i])
386             if k == k:  # Closed-loop-rmpc; we run the optimizer every 'k'
387                 u_opt, w_opt = optimzr_rmpc2(*nlp21, xk, tk, uk, p0, w_opt)
388             u_optk = np.array([u_opt[0]]).flatten()
389             u_optz = np.array([u_opt[1:]]).flatten()
390             x_opt = simulator(f, xk, tk, u_optk, p0)
391             x_optk = np.array([x_opt]).flatten()
392             t_iter = tm.time() - t_iter
393             x_opts[:, k, i] = x_optk
394             u_opts[:, k, i] = u_optk
395             t_calc[:, k, i] = t_iter
396             print(f"Computed iteration: {k}")
397             print(f"Computer-time: {t_iter}")
398             print(f"Switch-rule:   {switch}")
399 
400 elif clsd_loop_rmpc2:
401     for i in range(N):
402         w_opt = np.array([])
403         switch = np.array([0, 0])
404         x_opts[:, 0, i] = x0.flatten()
405         u_opts[:, 0, i] = u0.flatten()
406         for k in range(1, dim_ts):
407             t_iter = tm.time()
408             tk = np.array([ts[k - 1], ts[k]])
409             xk = np.array(x_opts[:, k - 1, i])
410             uk = np.array(u_opts[:, k - 1, i])
411             pk = np.array(p_cnst[:, k - 1, i])
412             if k <= 69 or (np.all(switch) == 0):
413                 u_opt, w_opt = optimzr_rmpc2(*nlp01, xk, tk, uk, p0, w_opt)
414             elif np.all(np.sort(switch) == np.array([1, 2])):
415                 u_opt, w_opt = optimzr_rmpc2(*nlp01, xk, tk, uk, p0, w_opt)
416             elif np.all(np.sort(switch) == np.array([1, 3])):
417                 u_opt, w_opt = optimzr_rmpc2(*nlp02, xk, tk, uk, p0, w_opt)
418             elif np.all(np.sort(switch) == np.array([1, 4])):
419                 u_opt, w_opt = optimzr_rmpc2(*nlp03, xk, tk, uk, p0, w_opt)
420             elif np.all(np.sort(switch) == np.array([1, 5])):
421                 u_opt, w_opt = optimzr_rmpc2(*nlp04, xk, tk, uk, p0, w_opt)
422             elif np.all(np.sort(switch) == np.array([1, 6])):
423                 u_opt, w_opt = optimzr_rmpc2(*nlp05, xk, tk, uk, p0, w_opt)
424             elif np.all(np.sort(switch) == np.array([1, 7])):
425                 u_opt, w_opt = optimzr_rmpc2(*nlp06, xk, tk, uk, p0, w_opt)
426             elif np.all(np.sort(switch) == np.array([2, 3])):
427                 u_opt, w_opt = optimzr_rmpc2(*nlp07, xk, tk, uk, p0, w_opt)
428             elif np.all(np.sort(switch) == np.array([2, 4])):
429                 u_opt, w_opt = optimzr_rmpc2(*nlp08, xk, tk, uk, p0, w_opt)
430             elif np.all(np.sort(switch) == np.array([2, 5])):
431                 u_opt, w_opt = optimzr_rmpc2(*nlp09, xk, tk, uk, p0, w_opt)
432             elif np.all(np.sort(switch) == np.array([2, 6])):
433                 u_opt, w_opt = optimzr_rmpc2(*nlp10, xk, tk, uk, p0, w_opt)
434             elif np.all(np.sort(switch) == np.array([2, 7])):
435                 u_opt, w_opt = optimzr_rmpc2(*nlp11, xk, tk, uk, p0, w_opt)
436             elif np.all(np.sort(switch) == np.array([3, 4])):
437                 u_opt, w_opt = optimzr_rmpc2(*nlp12, xk, tk, uk, p0, w_opt)
438             elif np.all(np.sort(switch) == np.array([3, 5])):
439                 u_opt, w_opt = optimzr_rmpc2(*nlp13, xk, tk, uk, p0, w_opt)
440             elif np.all(np.sort(switch) == np.array([3, 6])):
441                 u_opt, w_opt = optimzr_rmpc2(*nlp14, xk, tk, uk, p0, w_opt)
442             elif np.all(np.sort(switch) == np.array([3, 7])):
443                 u_opt, w_opt = optimzr_rmpc2(*nlp15, xk, tk, uk, p0, w_opt)
444             elif np.all(np.sort(switch) == np.array([4, 5])):
445                 u_opt, w_opt = optimzr_rmpc2(*nlp16, xk, tk, uk, p0, w_opt)
446             elif np.all(np.sort(switch) == np.array([4, 6])):
447                 u_opt, w_opt = optimzr_rmpc2(*nlp17, xk, tk, uk, p0, w_opt)
448             elif np.all(np.sort(switch) == np.array([4, 7])):
449                 u_opt, w_opt = optimzr_rmpc2(*nlp18, xk, tk, uk, p0, w_opt)
450             elif np.all(np.sort(switch) == np.array([5, 6])):
451                 u_opt, w_opt = optimzr_rmpc2(*nlp19, xk, tk, uk, p0, w_opt)
452             elif np.all(np.sort(switch) == np.array([5, 7])):
453                 u_opt, w_opt = optimzr_rmpc2(*nlp20, xk, tk, uk, p0, w_opt)
454             elif np.all(np.sort(switch) == np.array([6, 7])):
455                 u_opt, w_opt = optimzr_rmpc2(*nlp21, xk, tk, uk, p0, w_opt)
456             else:  # if there is given an invalid parameter combination
457                 raise IndexError("The given combination doesn't exist")
458 
459             u_optk = np.array([u_opt[0]]).flatten()
460             u_optz = np.array([u_opt[1:]]).flatten()
461             x_opt = simulator(f, xk, tk, u_optk, p0)
```

```
462             x_optk = np.array([x_opt]).flatten()
463
464             if (k - 1) % (dim_tz - 1) == 0:
465                 s1_ind = np.zeros((dim_p, dim_tz - 1))
466                 st_ind = np.zeros((dim_p, dim_tz - 1))
467                 ee_abs = np.zeros((dim_p, dim_tz - 1))
468                 ee_std = np.zeros((dim_p, dim_tz - 1))
469                 pe_abs = np.zeros((dim_p, dim_tz - 1))
470                 pe_std = np.zeros((dim_p, dim_tz - 1))
471
472                 if k >= 69 and run_SA1:
473                     s1_ind, st_ind = sobols_method(f, x_optk, tz, u_optz[:(dim_tz - 1)], p0)
474                 if k >= 69 and run_SA2:
475                     ee_abs, ee_std = morris_method(f, x_optk, tz, u_optz[:(dim_tz - 1)], p0)
476                 if k >= 69 and run_SA3:
477                     pe_abs, pe_std = modify_method(f, x_optk, tz, u_optz[:(dim_tz - 1)], p0)
478
479                 if k >= 69 and (run_SA1 or run_SA2 or run_SA3):
480                     switch = switch_rule2(switch, st_ind, ee_abs, pe_abs)
481                 else:
482                     switch = np.array([0, 0], dtype=int)
483
484                 for j in range(dim_tz - 1):
485                     if (k + 1 + j) > (dim_ts - 1):
486                         break
487                     s1_inds[:, k + 1 + j, i] = s1_ind[:, j].flatten()
488                     st_inds[:, k + 1 + j, i] = st_ind[:, j].flatten()
489                     ee_abss[:, k + 1 + j, i] = ee_abs[:, j].flatten()
490                     ee_stds[:, k + 1 + j, i] = ee_std[:, j].flatten()
491                     pe_abss[:, k + 1 + j, i] = pe_abs[:, j].flatten()
492                     pe_stds[:, k + 1 + j, i] = pe_std[:, j].flatten()
493             t_iter = tm.time() - t_iter
494             x_opts[:, k, i] = x_optk
495             u_opts[:, k, i] = u_optk
496             t_calc[:, k, i] = t_iter
497             print(f"Computed iteration: {k}")
498             print(f"Computer-time: {t_iter}")
499             print(f"Switch-rule:   {switch}")
500
501 elif clsd_loop_rmpc3:
502     for i in range(N):
503         w_opt = np.array([])
504         switch = np.array([0, 0, 0])
505         x_opts[:, 0, i] = x0.flatten()
506         u_opts[:, 0, i] = u0.flatten()
507         for k in range(1, dim_ts):
508             t_iter = tm.time()
509             tk = np.array([ts[k - 1], ts[k]])
510             xk = np.array(x_opts[:, k - 1, i])
511             uk = np.array(u_opts[:, k - 1, i])
512             pk = np.array(p_cnst[:, k - 1, i])
513             if k <= 69 or (np.all(switch) == 0):
514                 u_opt, w_opt = optimzr_rmpc3(*nlp01, xk, tk, uk, p0, w_opt)
515             elif np.all(np.sort(switch) == np.array([1, 2, 3])):
516                 u_opt, w_opt = optimzr_rmpc3(*nlp01, xk, tk, uk, p0, w_opt)
517             elif np.all(np.sort(switch) == np.array([1, 2, 4])):
518                 u_opt, w_opt = optimzr_rmpc3(*nlp02, xk, tk, uk, p0, w_opt)
519             elif np.all(np.sort(switch) == np.array([1, 2, 5])):
520                 u_opt, w_opt = optimzr_rmpc3(*nlp03, xk, tk, uk, p0, w_opt)
521             elif np.all(np.sort(switch) == np.array([1, 2, 6])):
522                 u_opt, w_opt = optimzr_rmpc3(*nlp04, xk, tk, uk, p0, w_opt)
523             elif np.all(np.sort(switch) == np.array([1, 2, 7])):
524                 u_opt, w_opt = optimzr_rmpc3(*nlp05, xk, tk, uk, p0, w_opt)
525             elif np.all(np.sort(switch) == np.array([1, 3, 4])):
526                 u_opt, w_opt = optimzr_rmpc3(*nlp06, xk, tk, uk, p0, w_opt)
527             elif np.all(np.sort(switch) == np.array([1, 3, 5])):
528                 u_opt, w_opt = optimzr_rmpc3(*nlp07, xk, tk, uk, p0, w_opt)
529             elif np.all(np.sort(switch) == np.array([1, 3, 6])):
530                 u_opt, w_opt = optimzr_rmpc3(*nlp08, xk, tk, uk, p0, w_opt)
531             elif np.all(np.sort(switch) == np.array([1, 3, 7])):
532                 u_opt, w_opt = optimzr_rmpc3(*nlp09, xk, tk, uk, p0, w_opt)
533             elif np.all(np.sort(switch) == np.array([1, 4, 5])):
534                 u_opt, w_opt = optimzr_rmpc3(*nlp10, xk, tk, uk, p0, w_opt)
535             elif np.all(np.sort(switch) == np.array([1, 4, 6])):
536                 u_opt, w_opt = optimzr_rmpc3(*nlp11, xk, tk, uk, p0, w_opt)
537             elif np.all(np.sort(switch) == np.array([1, 4, 7])):
538                 u_opt, w_opt = optimzr_rmpc3(*nlp12, xk, tk, uk, p0, w_opt)
539             elif np.all(np.sort(switch) == np.array([1, 5, 6])):
540                 u_opt, w_opt = optimzr_rmpc3(*nlp13, xk, tk, uk, p0, w_opt)
541             elif np.all(np.sort(switch) == np.array([1, 5, 7])):
542                 u_opt, w_opt = optimzr_rmpc3(*nlp14, xk, tk, uk, p0, w_opt)
543             elif np.all(np.sort(switch) == np.array([1, 6, 7])):
544                 u_opt, w_opt = optimzr_rmpc3(*nlp15, xk, tk, uk, p0, w_opt)
545             elif np.all(np.sort(switch) == np.array([2, 3, 4])):
546                 u_opt, w_opt = optimzr_rmpc3(*nlp16, xk, tk, uk, p0, w_opt)
547             elif np.all(np.sort(switch) == np.array([2, 3, 5])):
548                 u_opt, w_opt = optimzr_rmpc3(*nlp17, xk, tk, uk, p0, w_opt)
549             elif np.all(np.sort(switch) == np.array([2, 3, 6])):
550                 u_opt, w_opt = optimzr_rmpc3(*nlp18, xk, tk, uk, p0, w_opt)
551             elif np.all(np.sort(switch) == np.array([2, 3, 7])):
552                 u_opt, w_opt = optimzr_rmpc3(*nlp19, xk, tk, uk, p0, w_opt)
553             elif np.all(np.sort(switch) == np.array([2, 4, 5])):
554                 u_opt, w_opt = optimzr_rmpc3(*nlp20, xk, tk, uk, p0, w_opt)
555             elif np.all(np.sort(switch) == np.array([2, 4, 6])):
556                 u_opt, w_opt = optimzr_rmpc3(*nlp21, xk, tk, uk, p0, w_opt)
```

```
557             elif np.all(np.sort(switch) == np.array([2, 4, 7])):
558                 u_opt, w_opt = optimzr_rmpc3(*nlp22, xk, tk, uk, p0, w_opt)
559             elif np.all(np.sort(switch) == np.array([2, 5, 6])):
560                 u_opt, w_opt = optimzr_rmpc3(*nlp23, xk, tk, uk, p0, w_opt)
561             elif np.all(np.sort(switch) == np.array([2, 5, 7])):
562                 u_opt, w_opt = optimzr_rmpc3(*nlp24, xk, tk, uk, p0, w_opt)
563             elif np.all(np.sort(switch) == np.array([2, 6, 7])):
564                 u_opt, w_opt = optimzr_rmpc3(*nlp25, xk, tk, uk, p0, w_opt)
565             elif np.all(np.sort(switch) == np.array([3, 4, 5])):
566                 u_opt, w_opt = optimzr_rmpc3(*nlp26, xk, tk, uk, p0, w_opt)
567             elif np.all(np.sort(switch) == np.array([3, 4, 6])):
568                 u_opt, w_opt = optimzr_rmpc3(*nlp27, xk, tk, uk, p0, w_opt)
569             elif np.all(np.sort(switch) == np.array([3, 4, 7])):
570                 u_opt, w_opt = optimzr_rmpc3(*nlp28, xk, tk, uk, p0, w_opt)
571             elif np.all(np.sort(switch) == np.array([3, 5, 6])):
572                 u_opt, w_opt = optimzr_rmpc3(*nlp29, xk, tk, uk, p0, w_opt)
573             elif np.all(np.sort(switch) == np.array([3, 5, 7])):
574                 u_opt, w_opt = optimzr_rmpc3(*nlp30, xk, tk, uk, p0, w_opt)
575             elif np.all(np.sort(switch) == np.array([3, 6, 7])):
576                 u_opt, w_opt = optimzr_rmpc3(*nlp31, xk, tk, uk, p0, w_opt)
577             elif np.all(np.sort(switch) == np.array([4, 5, 6])):
578                 u_opt, w_opt = optimzr_rmpc3(*nlp32, xk, tk, uk, p0, w_opt)
579             elif np.all(np.sort(switch) == np.array([4, 5, 7])):
580                 u_opt, w_opt = optimzr_rmpc3(*nlp33, xk, tk, uk, p0, w_opt)
581             elif np.all(np.sort(switch) == np.array([4, 6, 7])):
582                 u_opt, w_opt = optimzr_rmpc3(*nlp34, xk, tk, uk, p0, w_opt)
583             elif np.all(np.sort(switch) == np.array([5, 6, 7])):
584                 u_opt, w_opt = optimzr_rmpc3(*nlp35, xk, tk, uk, p0, w_opt)
585             else:  # if there is given an invalid parameter combination
586                 raise IndexError("The given combination doesn't exist")
587 
588             u_optk = np.array([u_opt[0]]).flatten()
589             u_optz = np.array([u_opt[1:]]).flatten()
590             x_opt = simulator(f, xk, tk, u_optk, p0)
591             x_optk = np.array([x_opt]).flatten()
592 
593             if (k - 1) % (dim_tz - 1) == 0:
594                 s1_ind = np.zeros((dim_p, dim_tz - 1))
595                 st_ind = np.zeros((dim_p, dim_tz - 1))
596                 ee_abs = np.zeros((dim_p, dim_tz - 1))
597                 ee_std = np.zeros((dim_p, dim_tz - 1))
598                 pe_abs = np.zeros((dim_p, dim_tz - 1))
599                 pe_std = np.zeros((dim_p, dim_tz - 1))
600 
601                 if k >= 69 and run_SA1:
602                     s1_ind, st_ind = sobols_method(f, x_optk, tz, u_optz[:(dim_tz - 1)], p0)
603                 if k >= 69 and run_SA2:
604                     ee_abs, ee_std = morris_method(f, x_optk, tz, u_optz[:(dim_tz - 1)], p0)
605                 if k >= 69 and run_SA3:
606                     pe_abs, pe_std = modify_method(f, x_optk, tz, u_optz[:(dim_tz - 1)], p0)
607 
608                 if k >= 69 and (run_SA1 or run_SA2 or run_SA3):
609                     switch = switch_rule3(switch, st_ind, ee_abs, pe_abs)
610                 else:
611                     switch = np.array([0, 0, 0], dtype=int)
612 
613                 for j in range(dim_tz - 1):
614                     if (k + 1 + j) > (dim_ts - 1):
615                         break
616                     s1_inds[:, k + 1 + j, i] = s1_ind[:, j].flatten()
617                     st_inds[:, k + 1 + j, i] = st_ind[:, j].flatten()
618                     ee_abss[:, k + 1 + j, i] = ee_abs[:, j].flatten()
619                     ee_stds[:, k + 1 + j, i] = ee_std[:, j].flatten()
620                     pe_abss[:, k + 1 + j, i] = pe_abs[:, j].flatten()
621                     pe_stds[:, k + 1 + j, i] = pe_std[:, j].flatten()
622             t_iter = tm.time() - t_iter
623             x_opts[:, k, i] = x_optk
624             u_opts[:, k, i] = u_optk
625             t_calc[:, k, i] = t_iter
626             print(f"Computed iteration: {k}")
627             print(f"Computer-time: {t_iter}")
628             print(f"Switch-rule:   {switch}")
629 
630 rcParams["axes.grid"] = False
631 rcParams["text.usetex"] = True
632 rcParams["axes.titlesize"] = 27.
633 rcParams["axes.labelsize"] = 27.
634 rcParams["xtick.labelsize"] = 23.
635 rcParams["ytick.labelsize"] = 23.
636 rcParams["legend.fontsize"] = 23.
637 
638 rcParams["figure.constrained_layout.use"] = True
639 rcParams["figure.constrained_layout.hspace"] = .0200
640 rcParams["figure.constrained_layout.wspace"] = .0200
641 rcParams["figure.constrained_layout.h_pad"] = .04167
642 rcParams["figure.constrained_layout.w_pad"] = .04167
643 cl = (rcParams["axes.prop_cycle"].by_key()["color"])
644 
645 if N >= 1:
646     x0_cvs = np.sum((x_opts[0, :, :] > 3.7), axis=0)
647     x0_worst_case_cv = x_opts[0, :, x0_cvs.argmax()]
648     if np.all(x0_cvs == 0):
649         mx = np.max(x_opts[0, :, :], axis=0)
650         x0_worst_case_cv = x_opts[0, :, mx.argmax()]
651 
```

```
652 if N >= 1:
653     u0_mean = np.mean(u_opts[0, :, :], axis=1)
654     x0_mean = np.mean(x_opts[0, :, :], axis=1)
655     x1_mean = np.mean(x_opts[1, :, :], axis=1)
656     x2_mean = np.mean(x_opts[2, :, :], axis=1)
657     x3_mean = np.mean(x_opts[3, :, :], axis=1)
658
659 if N >= 1:
660     u0_std = np.std(u_opts[0, :, :], axis=1)
661     x0_std = np.std(x_opts[0, :, :], axis=1)
662     x1_std = np.std(x_opts[1, :, :], axis=1)
663     x2_std = np.std(x_opts[2, :, :], axis=1)
664     x3_std = np.std(x_opts[3, :, :], axis=1)
665
666 if N >= 1:
667     s1_ind0 = s1_inds[0, :, :].flatten()
668     s1_ind1 = s1_inds[1, :, :].flatten()
669     s1_ind2 = s1_inds[2, :, :].flatten()
670     s1_ind3 = s1_inds[3, :, :].flatten()
671     s1_ind4 = s1_inds[4, :, :].flatten()
672     s1_ind5 = s1_inds[5, :, :].flatten()
673     s1_ind6 = s1_inds[6, :, :].flatten()
674
675 if N >= 1:
676     st_ind0 = st_inds[0, :, :].flatten()
677     st_ind1 = st_inds[1, :, :].flatten()
678     st_ind2 = st_inds[2, :, :].flatten()
679     st_ind3 = st_inds[3, :, :].flatten()
680     st_ind4 = st_inds[4, :, :].flatten()
681     st_ind5 = st_inds[5, :, :].flatten()
682     st_ind6 = st_inds[6, :, :].flatten()
683
684 if np.all(ee_abss == 0):
685     ee_abs0 = ee_abss[0, :, :].flatten()
686     ee_abs1 = ee_abss[1, :, :].flatten()
687     ee_abs2 = ee_abss[2, :, :].flatten()
688     ee_abs3 = ee_abss[3, :, :].flatten()
689     ee_abs4 = ee_abss[4, :, :].flatten()
690     ee_abs5 = ee_abss[5, :, :].flatten()
691     ee_abs6 = ee_abss[6, :, :].flatten()
692
693     ee_std0 = ee_stds[0, :, :].flatten()
694     ee_std1 = ee_stds[1, :, :].flatten()
695     ee_std2 = ee_stds[2, :, :].flatten()
696     ee_std3 = ee_stds[3, :, :].flatten()
697     ee_std4 = ee_stds[4, :, :].flatten()
698     ee_std5 = ee_stds[5, :, :].flatten()
699     ee_std6 = ee_stds[6, :, :].flatten()
700
701 if np.all(pe_abss == 0):
702     pe_abs0 = pe_abss[0, :, :].flatten()
703     pe_abs1 = pe_abss[1, :, :].flatten()
704     pe_abs2 = pe_abss[2, :, :].flatten()
705     pe_abs3 = pe_abss[3, :, :].flatten()
706     pe_abs4 = pe_abss[4, :, :].flatten()
707     pe_abs5 = pe_abss[5, :, :].flatten()
708     pe_abs6 = pe_abss[6, :, :].flatten()
709
710     pe_std0 = pe_stds[0, :, :].flatten()
711     pe_std1 = pe_stds[1, :, :].flatten()
712     pe_std2 = pe_stds[2, :, :].flatten()
713     pe_std3 = pe_stds[3, :, :].flatten()
714     pe_std4 = pe_stds[4, :, :].flatten()
715     pe_std5 = pe_stds[5, :, :].flatten()
716     pe_std6 = pe_stds[6, :, :].flatten()
717
718 if np.any(ee_abss != 0):
719     m1, m2 = np.max(ee_abss), np.max(ee_stds)
720     ee_abs0 = ee_abss[0, :, :].flatten() / m1
721     ee_abs1 = ee_abss[1, :, :].flatten() / m1
722     ee_abs2 = ee_abss[2, :, :].flatten() / m1
723     ee_abs3 = ee_abss[3, :, :].flatten() / m1
724     ee_abs4 = ee_abss[4, :, :].flatten() / m1
725     ee_abs5 = ee_abss[5, :, :].flatten() / m1
726     ee_abs6 = ee_abss[6, :, :].flatten() / m1
727
728     ee_std0 = ee_stds[0, :, :].flatten() / m1
729     ee_std1 = ee_stds[1, :, :].flatten() / m1
730     ee_std2 = ee_stds[2, :, :].flatten() / m1
731     ee_std3 = ee_stds[3, :, :].flatten() / m1
732     ee_std4 = ee_stds[4, :, :].flatten() / m1
733     ee_std5 = ee_stds[5, :, :].flatten() / m1
734     ee_std6 = ee_stds[6, :, :].flatten() / m1
735
736 if np.any(pe_abss != 0):
737     m1, m2 = np.max(pe_abss), np.max(pe_stds)
738     pe_abs0 = pe_abss[0, :, :].flatten() / m1
739     pe_abs1 = pe_abss[1, :, :].flatten() / m1
740     pe_abs2 = pe_abss[2, :, :].flatten() / m1
741     pe_abs3 = pe_abss[3, :, :].flatten() / m1
742     pe_abs4 = pe_abss[4, :, :].flatten() / m1
743     pe_abs5 = pe_abss[5, :, :].flatten() / m1
744     pe_abs6 = pe_abss[6, :, :].flatten() / m1
745
746     pe_std0 = pe_stds[0, :, :].flatten() / m1
```

```
747     pe_std1 = pe_stds[1, :, :].flatten() / ml
748     pe_std2 = pe_stds[2, :, :].flatten() / ml
749     pe_std3 = pe_stds[3, :, :].flatten() / ml
750     pe_std4 = pe_stds[4, :, :].flatten() / ml
751     pe_std5 = pe_stds[5, :, :].flatten() / ml
752     pe_std6 = pe_stds[6, :, :].flatten() / ml
753
754 if open_loop_nmpc:
755     u0_ax0_legend_bounds = tuple([0.845, 0.730])
756     u0_ax1_legend_bounds = tuple([0.845, 0.730])
757
758     x0_ax0_legend_bounds = tuple([0.005, 0.730])
759     x0_ax1_legend_bounds = tuple([0.005, 0.730])
760     x0_ax2_legend_bounds = tuple([0.750, 0.005])
761     x0_ax3_legend_bounds = tuple([0.745, 0.005])
762
763     x1_ax0_legend_bounds = tuple([0.845, 0.730])
764     x1_ax1_legend_bounds = tuple([0.845, 0.730])
765
766     x2_ax0_legend_bounds = tuple([0.005, 0.730])
767     x2_ax1_legend_bounds = tuple([0.005, 0.730])
768
769     x3_ax0_legend_bounds = tuple([0.005, 0.730])
770     x3_ax1_legend_bounds = tuple([0.005, 0.730])
771
772 elif clsd_loop_nmpc:
773     u0_ax0_legend_bounds = tuple([0.845, 0.730])
774     u0_ax1_legend_bounds = tuple([0.845, 0.730])
775
776     x0_ax0_legend_bounds = tuple([0.005, 0.730])
777     x0_ax1_legend_bounds = tuple([0.005, 0.730])
778     x0_ax2_legend_bounds = tuple([0.750, 0.005])
779     x0_ax3_legend_bounds = tuple([0.735, 0.005])
780
781     x1_ax0_legend_bounds = tuple([0.845, 0.730])
782     x1_ax1_legend_bounds = tuple([0.845, 0.730])
783
784     x2_ax0_legend_bounds = tuple([0.005, 0.730])
785     x2_ax1_legend_bounds = tuple([0.005, 0.730])
786
787     x3_ax0_legend_bounds = tuple([0.005, 0.730])
788     x3_ax1_legend_bounds = tuple([0.005, 0.730])
789
790 elif clsd_loop_rmpc1:
791     u0_ax0_legend_bounds = tuple([0.845, 0.730])
792     u0_ax1_legend_bounds = tuple([0.845, 0.730])
793
794     x0_ax0_legend_bounds = tuple([0.005, 0.730])
795     x0_ax1_legend_bounds = tuple([0.005, 0.730])
796     x0_ax2_legend_bounds = tuple([0.750, 0.005])
797     x0_ax3_legend_bounds = tuple([0.735, 0.005])
798
799     x1_ax0_legend_bounds = tuple([0.845, 0.730])
800     x1_ax1_legend_bounds = tuple([0.845, 0.730])
801
802     x2_ax0_legend_bounds = tuple([0.005, 0.730])
803     x2_ax1_legend_bounds = tuple([0.005, 0.730])
804
805     x3_ax0_legend_bounds = tuple([0.005, 0.730])
806     x3_ax1_legend_bounds = tuple([0.005, 0.730])
807
808 elif clsd_loop_rmpc2:
809     u0_ax0_legend_bounds = tuple([0.845, 0.730])
810     u0_ax1_legend_bounds = tuple([0.845, 0.730])
811
812     x0_ax0_legend_bounds = tuple([0.005, 0.730])
813     x0_ax1_legend_bounds = tuple([0.005, 0.730])
814     x0_ax2_legend_bounds = tuple([0.750, 0.005])
815     x0_ax3_legend_bounds = tuple([0.735, 0.005])
816
817     x1_ax0_legend_bounds = tuple([0.845, 0.730])
818     x1_ax1_legend_bounds = tuple([0.845, 0.730])
819
820     x2_ax0_legend_bounds = tuple([0.005, 0.730])
821     x2_ax1_legend_bounds = tuple([0.005, 0.730])
822
823     x3_ax0_legend_bounds = tuple([0.005, 0.730])
824     x3_ax1_legend_bounds = tuple([0.005, 0.730])
825
826 elif clsd_loop_rmpc3:
827     u0_ax0_legend_bounds = tuple([0.845, 0.730])
828     u0_ax1_legend_bounds = tuple([0.845, 0.730])
829
830     x0_ax0_legend_bounds = tuple([0.005, 0.730])
831     x0_ax1_legend_bounds = tuple([0.005, 0.730])
832     x0_ax2_legend_bounds = tuple([0.750, 0.005])
833     x0_ax3_legend_bounds = tuple([0.735, 0.005])
834
835     x1_ax0_legend_bounds = tuple([0.845, 0.730])
836     x1_ax1_legend_bounds = tuple([0.845, 0.730])
837
838     x2_ax0_legend_bounds = tuple([0.005, 0.730])
839     x2_ax1_legend_bounds = tuple([0.005, 0.730])
840
841     x3_ax0_legend_bounds = tuple([0.005, 0.730])
```

```
842         x3_ax1_legend_bounds = tuple([0.005, 0.730])
843
844     if N >= 1:
845         s1_ax4_legend_bounds = tuple([0.815, 0.635])
846         st_ax5_legend_bounds = tuple([0.815, 0.635])
847
848         ee_ax6_legend_bounds = tuple([0.815, 0.635])
849         ee_ax7_legend_bounds = tuple([0.815, 0.635])
850
851         pe_ax8_legend_bounds = tuple([0.815, 0.635])
852         pe_ax9_legend_bounds = tuple([0.815, 0.635])
853
854     if open_loop_nmpc:
855         u0_ax0_plotting_bounds = np.array([0.000, 0.120])
856         u0_ax1_plotting_bounds = np.array([0.000, 0.120])
857
858         x0_ax0_plotting_bounds = np.array([1.000, 4.450])
859         x0_ax1_plotting_bounds = np.array([1.000, 4.450])
860         x0_ax2_plotting_bounds = np.array([1.000, 4.350])
861         x0_ax3_plotting_bounds = np.array([3.150, 4.010])
862
863         x1_ax0_plotting_bounds = np.array([0.000, 3.000])
864         x1_ax1_plotting_bounds = np.array([0.000, 3.000])
865
866         x2_ax0_plotting_bounds = np.array([0.000, 2.100])
867         x2_ax1_plotting_bounds = np.array([0.000, 2.100])
868
869         x3_ax0_plotting_bounds = np.array([120.0, 126.0])
870         x3_ax1_plotting_bounds = np.array([120.0, 126.0])
871
872         ts_ax0_plotting_bounds = np.array([0.000, 150.0])
873         ts_ax1_plotting_bounds = np.array([0.000, 150.0])
874         ts_ax2_plotting_bounds = np.array([0.000, 150.0])
875         ts_ax3_plotting_bounds = np.array([60.00, 150.0])
876
877     elif clsd_loop_nmpc:
878         u0_ax0_plotting_bounds = np.array([0.000, 0.120])
879         u0_ax1_plotting_bounds = np.array([0.000, 0.120])
880
881         x0_ax0_plotting_bounds = np.array([1.000, 4.400])
882         x0_ax1_plotting_bounds = np.array([1.000, 4.400])
883         x0_ax2_plotting_bounds = np.array([1.000, 4.300])
884         x0_ax3_plotting_bounds = np.array([3.691, 3.703])
885
886         x1_ax0_plotting_bounds = np.array([0.000, 0.900])
887         x1_ax1_plotting_bounds = np.array([0.000, 0.900])
888
889         x2_ax0_plotting_bounds = np.array([0.000, 2.700])
890         x2_ax1_plotting_bounds = np.array([0.000, 2.700])
891
892         x3_ax0_plotting_bounds = np.array([120.0, 129.0])
893         x3_ax1_plotting_bounds = np.array([120.0, 129.0])
894
895         ts_ax0_plotting_bounds = np.array([0.000, 150.0])
896         ts_ax1_plotting_bounds = np.array([0.000, 150.0])
897         ts_ax2_plotting_bounds = np.array([0.000, 150.0])
898         ts_ax3_plotting_bounds = np.array([90.00, 150.0])
899
900     elif clsd_loop_rmpc1:
901         u0_ax0_plotting_bounds = np.array([0.000, 0.120])
902         u0_ax1_plotting_bounds = np.array([0.000, 0.120])
903
904         x0_ax0_plotting_bounds = np.array([1.000, 4.400])
905         x0_ax1_plotting_bounds = np.array([1.000, 4.400])
906         x0_ax2_plotting_bounds = np.array([1.000, 4.300])
907         x0_ax3_plotting_bounds = np.array([3.691, 3.703])
908
909         x1_ax0_plotting_bounds = np.array([0.000, 0.900])
910         x1_ax1_plotting_bounds = np.array([0.000, 0.900])
911
912         x2_ax0_plotting_bounds = np.array([0.000, 2.700])
913         x2_ax1_plotting_bounds = np.array([0.000, 2.700])
914
915         x3_ax0_plotting_bounds = np.array([120.0, 129.0])
916         x3_ax1_plotting_bounds = np.array([120.0, 129.0])
917
918         ts_ax0_plotting_bounds = np.array([0.000, 150.0])
919         ts_ax1_plotting_bounds = np.array([0.000, 150.0])
920         ts_ax2_plotting_bounds = np.array([0.000, 150.0])
921         ts_ax3_plotting_bounds = np.array([90.00, 150.0])
922
923     elif clsd_loop_rmpc2:
924         u0_ax0_plotting_bounds = np.array([0.000, 0.120])
925         u0_ax1_plotting_bounds = np.array([0.000, 0.120])
926
927         x0_ax0_plotting_bounds = np.array([1.000, 4.400])
928         x0_ax1_plotting_bounds = np.array([1.000, 4.400])
929         x0_ax2_plotting_bounds = np.array([1.000, 4.300])
930         x0_ax3_plotting_bounds = np.array([3.691, 3.703])
931
932         x1_ax0_plotting_bounds = np.array([0.000, 0.900])
933         x1_ax1_plotting_bounds = np.array([0.000, 0.900])
934
935         x2_ax0_plotting_bounds = np.array([0.000, 2.700])
936         x2_ax1_plotting_bounds = np.array([0.000, 2.700])
```

```
937
938     x3_ax0_plotting_bounds = np.array([120.0, 129.0])
939     x3_ax1_plotting_bounds = np.array([120.0, 129.0])
940
941     ts_ax0_plotting_bounds = np.array([0.000, 150.0])
942     ts_ax1_plotting_bounds = np.array([0.000, 150.0])
943     ts_ax2_plotting_bounds = np.array([0.000, 150.0])
944     ts_ax3_plotting_bounds = np.array([90.00, 150.0])
945
946 elif clsd_loop_rmpc3:
947     u0_ax0_plotting_bounds = np.array([0.000, 0.120])
948     u0_ax1_plotting_bounds = np.array([0.000, 0.120])
949
950     x0_ax0_plotting_bounds = np.array([1.000, 4.400])
951     x0_ax1_plotting_bounds = np.array([1.000, 4.400])
952     x0_ax2_plotting_bounds = np.array([1.000, 4.300])
953     x0_ax3_plotting_bounds = np.array([3.691, 3.703])
954
955     x1_ax0_plotting_bounds = np.array([0.000, 0.900])
956     x1_ax1_plotting_bounds = np.array([0.000, 0.900])
957
958     x2_ax0_plotting_bounds = np.array([0.000, 2.700])
959     x2_ax1_plotting_bounds = np.array([0.000, 2.700])
960
961     x3_ax0_plotting_bounds = np.array([120.0, 129.0])
962     x3_ax1_plotting_bounds = np.array([120.0, 129.0])
963
964     ts_ax0_plotting_bounds = np.array([0.000, 150.0])
965     ts_ax1_plotting_bounds = np.array([0.000, 150.0])
966     ts_ax2_plotting_bounds = np.array([0.000, 150.0])
967     ts_ax3_plotting_bounds = np.array([90.00, 150.0])
968
969 if N >= 1:
970     s1_ax4_plotting_bounds = np.array([-0.10, 1.100])
971     st_ax5_plotting_bounds = np.array([-0.10, 1.100])
972
973     ee_ax6_plotting_bounds = np.array([-0.10, 1.100])
974     ee_ax7_plotting_bounds = np.array([-0.10, 1.100])
975
976     pe_ax8_plotting_bounds = np.array([-0.10, 1.100])
977     pe_ax9_plotting_bounds = np.array([-0.10, 1.100])
978
979     ts_ax4_plotting_bounds = np.array([0.000, 150.0])
980     ts_ax5_plotting_bounds = np.array([0.000, 150.0])
981     ts_ax6_plotting_bounds = np.array([0.000, 150.0])
982     ts_ax7_plotting_bounds = np.array([0.000, 150.0])
983     ts_ax8_plotting_bounds = np.array([0.000, 150.0])
984     ts_ax9_plotting_bounds = np.array([0.000, 150.0])
985
986 if open_loop_nmpc and N == 25:
987     np.save("open_loop_nmpc_N25_u_opts.npy", u_opts)
988     np.save("open_loop_nmpc_N25_x_opts.npy", x_opts)
989     np.save("open_loop_nmpc_N25_t_calc.npy", t_calc)
990
991 elif clsd_loop_nmpc and N == 25:
992     np.save("clsd_loop_nmpc_N25_u_opts.npy", u_opts)
993     np.save("clsd_loop_nmpc_N25_x_opts.npy", x_opts)
994     np.save("clsd_loop_nmpc_N25_t_calc.npy", t_calc)
995
996 elif clsd_loop_rmpc1 and N == 25:
997     np.save("clsd_loop_rmpc1_N25_u_opts.npy", u_opts)
998     np.save("clsd_loop_rmpc1_N25_x_opts.npy", x_opts)
999     np.save("clsd_loop_rmpc1_N25_t_calc.npy", t_calc)
1000
1001 elif clsd_loop_nmpc and run_SA1 and N == 1:
1002     np.save("clsd_loop_nmpc_N01_N2e12_SA1_s1_inds.npy", s1_inds)
1003     np.save("clsd_loop_nmpc_N01_N2e12_SA1_st_inds.npy", st_inds)
1004
1005 elif clsd_loop_nmpc and run_SA2 and N == 1:
1006     np.save("clsd_loop_nmpc_N01_N2e12_SA2_ee_abss.npy", ee_abss)
1007     np.save("clsd_loop_nmpc_N01_N2e12_SA2_ee_stds.npy", ee_stds)
1008
1009 elif clsd_loop_nmpc and run_SA3 and N == 1:
1010     np.save("clsd_loop_nmpc_N01_N2e12_SA3_pe_abss.npy", pe_abss)
1011     np.save("clsd_loop_nmpc_N01_N2e12_SA3_pe_stds.npy", pe_stds)
1012
1013 elif clsd_loop_rmpc2 and run_SA1 and N == 25 and dim_tz == 2:
1014     np.save("clsd_loop_rmpc2_N25_N2e10_tz01_SA1_u_opts.npy", u_opts)
1015     np.save("clsd_loop_rmpc2_N25_N2e10_tz01_SA1_x_opts.npy", x_opts)
1016     np.save("clsd_loop_rmpc2_N25_N2e10_tz01_SA1_t_calc.npy", t_calc)
1017
1018 elif clsd_loop_rmpc2 and run_SA2 and N == 25 and dim_tz == 2:
1019     np.save("clsd_loop_rmpc2_N25_N2e10_tz01_SA2_u_opts.npy", u_opts)
1020     np.save("clsd_loop_rmpc2_N25_N2e10_tz01_SA2_x_opts.npy", x_opts)
1021     np.save("clsd_loop_rmpc2_N25_N2e10_tz01_SA2_t_calc.npy", t_calc)
1022
1023 elif clsd_loop_rmpc2 and run_SA3 and N == 25 and dim_tz == 2:
1024     np.save("clsd_loop_rmpc2_N25_N2e10_tz01_SA3_u_opts.npy", u_opts)
1025     np.save("clsd_loop_rmpc2_N25_N2e10_tz01_SA3_x_opts.npy", x_opts)
1026     np.save("clsd_loop_rmpc2_N25_N2e10_tz01_SA3_t_calc.npy", t_calc)
1027
1028 elif clsd_loop_rmpc2 and run_SA1 and N == 25 and dim_tz == 6:
1029     np.save("clsd_loop_rmpc2_N25_N2e10_tz05_SA1_u_opts.npy", u_opts)
1030     np.save("clsd_loop_rmpc2_N25_N2e10_tz05_SA1_x_opts.npy", x_opts)
1031     np.save("clsd_loop_rmpc2_N25_N2e10_tz05_SA1_t_calc.npy", t_calc)
```

```python
elif clsd_loop_rmpc2 and run_SA2 and N == 25 and dim_tz == 6:
    np.save("clsd_loop_rmpc2_N25_N2e10_tz05_SA2_u_opts.npy", u_opts)
    np.save("clsd_loop_rmpc2_N25_N2e10_tz05_SA2_x_opts.npy", x_opts)
    np.save("clsd_loop_rmpc2_N25_N2e10_tz05_SA2_t_calc.npy", t_calc)

elif clsd_loop_rmpc2 and run_SA3 and N == 25 and dim_tz == 6:
    np.save("clsd_loop_rmpc2_N25_N2e10_tz05_SA3_u_opts.npy", u_opts)
    np.save("clsd_loop_rmpc2_N25_N2e10_tz05_SA3_x_opts.npy", x_opts)
    np.save("clsd_loop_rmpc2_N25_N2e10_tz05_SA3_t_calc.npy", t_calc)

elif clsd_loop_rmpc3 and run_SA1 and N == 25 and dim_tz == 2:
    np.save("clsd_loop_rmpc3_N25_N2e10_tz01_SA1_u_opts.npy", u_opts)
    np.save("clsd_loop_rmpc3_N25_N2e10_tz01_SA1_x_opts.npy", x_opts)
    np.save("clsd_loop_rmpc3_N25_N2e10_tz01_SA1_t_calc.npy", t_calc)

elif clsd_loop_rmpc3 and run_SA2 and N == 25 and dim_tz == 2:
    np.save("clsd_loop_rmpc3_N25_N2e10_tz01_SA2_u_opts.npy", u_opts)
    np.save("clsd_loop_rmpc3_N25_N2e10_tz01_SA2_x_opts.npy", x_opts)
    np.save("clsd_loop_rmpc3_N25_N2e10_tz01_SA2_t_calc.npy", t_calc)

elif clsd_loop_rmpc3 and run_SA3 and N == 25 and dim_tz == 2:
    np.save("clsd_loop_rmpc3_N25_N2e10_tz01_SA3_u_opts.npy", u_opts)
    np.save("clsd_loop_rmpc3_N25_N2e10_tz01_SA3_x_opts.npy", x_opts)
    np.save("clsd_loop_rmpc3_N25_N2e10_tz01_SA3_t_calc.npy", t_calc)

elif clsd_loop_rmpc3 and run_SA1 and N == 25 and dim_tz == 6:
    np.save("clsd_loop_rmpc3_N25_N2e10_tz05_SA1_u_opts.npy", u_opts)
    np.save("clsd_loop_rmpc3_N25_N2e10_tz05_SA1_x_opts.npy", x_opts)
    np.save("clsd_loop_rmpc3_N25_N2e10_tz05_SA1_t_calc.npy", t_calc)

elif clsd_loop_rmpc3 and run_SA2 and N == 25 and dim_tz == 6:
    np.save("clsd_loop_rmpc3_N25_N2e10_tz05_SA2_u_opts.npy", u_opts)
    np.save("clsd_loop_rmpc3_N25_N2e10_tz05_SA2_x_opts.npy", x_opts)
    np.save("clsd_loop_rmpc3_N25_N2e10_tz05_SA2_t_calc.npy", t_calc)

elif clsd_loop_rmpc3 and run_SA3 and N == 25 and dim_tz == 6:
    np.save("clsd_loop_rmpc3_N25_N2e10_tz05_SA3_u_opts.npy", u_opts)
    np.save("clsd_loop_rmpc3_N25_N2e10_tz05_SA3_x_opts.npy", x_opts)
    np.save("clsd_loop_rmpc3_N25_N2e10_tz05_SA3_t_calc.npy", t_calc)

if plot_figure_0 and N == 1:
    fig0, ax0 = plt.subplots(nrows=5, ncols=1, sharex="all", figsize=(10, 13))
    ax0[0].step(ts, u0_mean, linewidth=2., alpha=.95, color=cl[0], label=r"$u$")
    ax0[1].plot(ts, x0_mean, linewidth=2., alpha=.95, color=cl[0], label=r"$X$")
    ax0[2].plot(ts, x1_mean, linewidth=2., alpha=.95, color=cl[0], label=r"$S$")
    ax0[3].plot(ts, x2_mean, linewidth=2., alpha=.95, color=cl[0], label=r"$P$")
    ax0[4].plot(ts, x3_mean, linewidth=2., alpha=.95, color=cl[0], label=r"$V$")

    ax0[0].set_yticks(np.linspace(*u0_ax0_plotting_bounds, 4).round(2))
    ax0[1].set_yticks(np.linspace(*x0_ax0_plotting_bounds, 4).round(0))
    ax0[2].set_yticks(np.linspace(*x1_ax0_plotting_bounds, 4).round(1))
    ax0[3].set_yticks(np.linspace(*x2_ax0_plotting_bounds, 4).round(1))
    ax0[4].set_yticks(np.linspace(*x3_ax0_plotting_bounds, 4).round(0))

    ax0[0].set_ylabel(r"$u~[\textrm{m}^{3}/\textrm{hr}]$")
    ax0[1].set_ylabel(r"$X~      [\textrm{g}/\textrm{l}]$")
    ax0[2].set_ylabel(r"$S~      [\textrm{g}/\textrm{l}]$")
    ax0[3].set_ylabel(r"$P~      [\textrm{g}/\textrm{l}]$")
    ax0[4].set_ylabel(r"$V~             [\textrm{m}^{3}]$")
    ax0[4].set_xlabel(r"$t~                [\textrm{hr}]$")

    ax0[0].set_ylim(u0_ax0_plotting_bounds)
    ax0[1].set_ylim(x0_ax0_plotting_bounds)
    ax0[2].set_ylim(x1_ax0_plotting_bounds)
    ax0[3].set_ylim(x2_ax0_plotting_bounds)
    ax0[4].set_ylim(x3_ax0_plotting_bounds)
    ax0[4].set_xlim(ts_ax0_plotting_bounds)

    ax0[0].legend(loc=u0_ax0_legend_bounds)
    ax0[1].legend(loc=x0_ax0_legend_bounds)
    ax0[2].legend(loc=x1_ax0_legend_bounds)
    ax0[3].legend(loc=x2_ax0_legend_bounds)
    ax0[4].legend(loc=x3_ax0_legend_bounds)
    fig0.show()  # ---------------------- #

if plot_figure_1 and N != 1:
    fig1, ax1 = plt.subplots(nrows=5, ncols=1, sharex="all", figsize=(10, 13))
    ax1[0].step(ts, u0_mean, linewidth=2., alpha=.95, color=cl[0], label=r"$u$")
    ax1[0].fill_between(ts, u0_mean - 1 * u0_std, u0_mean + 1 * u0_std, linewidth=2., alpha=.2, color=cl[0])
    ax1[0].fill_between(ts, u0_mean - 2 * u0_std, u0_mean + 2 * u0_std, linewidth=2., alpha=.2, color=cl[0])

    ax1[1].plot(ts, x0_mean, linewidth=2., alpha=.95, color=cl[0], label=r"$X$")
    ax1[1].fill_between(ts, x0_mean - 1 * x0_std, x0_mean + 1 * x0_std, linewidth=2., alpha=.2, color=cl[0])
    ax1[1].fill_between(ts, x0_mean - 2 * x0_std, x0_mean + 2 * x0_std, linewidth=2., alpha=.2, color=cl[0])

    ax1[2].plot(ts, x1_mean, linewidth=2., alpha=.95, color=cl[0], label=r"$S$")
    ax1[2].fill_between(ts, x1_mean - 1 * x1_std, x1_mean + 1 * x1_std, linewidth=2., alpha=.2, color=cl[0])
    ax1[2].fill_between(ts, x1_mean - 2 * x1_std, x1_mean + 2 * x1_std, linewidth=2., alpha=.2, color=cl[0])

    ax1[3].plot(ts, x2_mean, linewidth=2., alpha=.95, color=cl[0], label=r"$P$")
    ax1[3].fill_between(ts, x2_mean - 1 * x2_std, x2_mean + 1 * x2_std, linewidth=2., alpha=.2, color=cl[0])
    ax1[3].fill_between(ts, x2_mean - 2 * x2_std, x2_mean + 2 * x2_std, linewidth=2., alpha=.2, color=cl[0])

    ax1[4].plot(ts, x3_mean, linewidth=2., alpha=.95, color=cl[0], label=r"$V$")
```

```
1127        ax1[4].fill_between(ts, x3_mean - 1 * x3_std, x3_mean + 1 * x3_std, linewidth=2., alpha=.2, color=cl[0])
1128        ax1[4].fill_between(ts, x3_mean - 2 * x3_std, x3_mean + 2 * x3_std, linewidth=2., alpha=.2, color=cl[0])
1129
1130        ax1[0].set_yticks(np.linspace(*u0_ax1_plotting_bounds, 4).round(3))
1131        ax1[1].set_yticks(np.linspace(*x0_ax1_plotting_bounds, 4).round(0))
1132        ax1[2].set_yticks(np.linspace(*x1_ax1_plotting_bounds, 4).round(1))
1133        ax1[3].set_yticks(np.linspace(*x2_ax1_plotting_bounds, 4).round(1))
1134        ax1[4].set_yticks(np.linspace(*x3_ax1_plotting_bounds, 4).round(0))
1135
1136        ax1[0].set_ylabel(r"$u~[\textrm{m}^{3}/\textrm{hr}]$")
1137        ax1[1].set_ylabel(r"$X~      [\textrm{g}/\textrm{l}]$")
1138        ax1[2].set_ylabel(r"$S~      [\textrm{g}/\textrm{l}]$")
1139        ax1[3].set_ylabel(r"$P~      [\textrm{g}/\textrm{l}]$")
1140        ax1[4].set_ylabel(r"$V~            [\textrm{m}^{3}]$")
1141        ax1[4].set_xlabel(r"$t~              [\textrm{hr}]$")
1142
1143        ax1[0].set_ylim(u0_ax1_plotting_bounds)
1144        ax1[1].set_ylim(x0_ax1_plotting_bounds)
1145        ax1[2].set_ylim(x1_ax1_plotting_bounds)
1146        ax1[3].set_ylim(x2_ax1_plotting_bounds)
1147        ax1[4].set_ylim(x3_ax1_plotting_bounds)
1148        ax1[4].set_xlim(ts_ax1_plotting_bounds)
1149
1150        ax1[0].legend(loc=u0_ax1_legend_bounds)
1151        ax1[1].legend(loc=x0_ax1_legend_bounds)
1152        ax1[2].legend(loc=x1_ax1_legend_bounds)
1153        ax1[3].legend(loc=x2_ax1_legend_bounds)
1154        ax1[4].legend(loc=x3_ax1_legend_bounds)
1155        fig1.show()  # ---------------------- #
1156
1157    if plot_figure_2 and N != 1:
1158        fig2, ax2 = plt.subplots(nrows=1, ncols=1, sharex="all", figsize=(10, 10))
1159        ax2.plot(ts, x0_mean, linewidth=2., alpha=.95, color=cl[0], label=r"$X~$")
1160        ax2.fill_between(ts, x0_mean - 1 * x0_std, x0_mean + 1 * x0_std, linewidth=2., alpha=.2, color=cl[0])
1161        ax2.fill_between(ts, x0_mean - 2 * x0_std, x0_mean + 2 * x0_std, linewidth=2., alpha=.2, color=cl[0])
1162        ax2.hlines(3.7, ts[0], ts[-1], linestyle="--", linewidth=2., alpha=.95, color="k", label=r"$X\:{\leq}\:{3.7}$")
1163
1164        ax2.set_ylabel(r"$X~[\textrm{g}/\textrm{l}]$")
1165        ax2.set_xlabel(r"$t~          [\textrm{hr}]$")
1166
1167        ax2.set_ylim(x0_ax2_plotting_bounds)
1168        ax2.set_xlim(ts_ax2_plotting_bounds)
1169
1170        ax2.legend(loc=x0_ax2_legend_bounds)
1171        fig2.show()  # ------------------- #
1172
1173    if plot_figure_3 and N != 1:
1174        fig3, ax3 = plt.subplots(nrows=1, ncols=1, sharex="all", figsize=(10, 10))
1175        ax3.plot(ts, x0_worst_case_cv, linewidth=2., alpha=.95, color=cl[0], label=r"$X$")
1176        ax3.hlines(3.7, ts[0], ts[-1], linestyle="--", linewidth=2., alpha=.95, color="k", label=r"$X\:{\leq}\:{3.7}$")
1177
1178        ax3.set_ylabel(r"$X~[\textrm{g}/\textrm{l}]$")
1179        ax3.set_xlabel(r"$t~          [\textrm{hr}]$")
1180
1181        ax3.set_ylim(x0_ax3_plotting_bounds)
1182        ax3.set_xlim(ts_ax3_plotting_bounds)
1183
1184        ax3.legend(loc=x0_ax3_legend_bounds)
1185        fig3.show()  # ------------------- #
1186
1187    if plot_figure_4 and N == 1:
1188        fig4, ax4 = plt.subplots(nrows=1, ncols=1, sharex="all", figsize=(10, 10))
1189        ax4.plot(ts, s1_ind0, linewidth=2., alpha=.95, color=cl[0], label=r"$\mu_{m}$")
1190        ax4.plot(ts, s1_ind1, linewidth=2., alpha=.95, color=cl[1], label=r"  $K_{m}$")
1191        ax4.plot(ts, s1_ind2, linewidth=2., alpha=.95, color=cl[2], label=r"  $K_{i}$")
1192        ax4.plot(ts, s1_ind3, linewidth=2., alpha=.95, color=cl[3], label=r"     $\nu$")
1193        ax4.plot(ts, s1_ind4, linewidth=2., alpha=.95, color=cl[4], label=r"  $Y_{p}$")
1194        ax4.plot(ts, s1_ind5, linewidth=2., alpha=.95, color=cl[5], label=r"  $Y_{x}$")
1195        ax4.plot(ts, s1_ind6, linewidth=2., alpha=.95, color=cl[6], label=r" $S_{in}$")
1196
1197        ax4.set_ylabel(r"$S_{1}~[\textrm{-}]$")
1198        ax4.set_xlabel(r"$t~   [\textrm{hr}]$")
1199
1200        ax4.set_ylim(s1_ax4_plotting_bounds)
1201        ax4.set_xlim(ts_ax4_plotting_bounds)
1202
1203        ax4.legend(loc=s1_ax4_legend_bounds)
1204        fig4.show()  # ------------------- #
1205
1206    if plot_figure_5 and N == 1:
1207        fig5, ax5 = plt.subplots(nrows=1, ncols=1, sharex="all", figsize=(10, 10))
1208        ax5.plot(ts, st_ind0, linewidth=2., alpha=.95, color=cl[0], label=r"$\mu_{m}$")
1209        ax5.plot(ts, st_ind1, linewidth=2., alpha=.95, color=cl[1], label=r"  $K_{m}$")
1210        ax5.plot(ts, st_ind2, linewidth=2., alpha=.95, color=cl[2], label=r"  $K_{i}$")
1211        ax5.plot(ts, st_ind3, linewidth=2., alpha=.95, color=cl[3], label=r"     $\nu$")
1212        ax5.plot(ts, st_ind4, linewidth=2., alpha=.95, color=cl[4], label=r"  $Y_{p}$")
1213        ax5.plot(ts, st_ind5, linewidth=2., alpha=.95, color=cl[5], label=r"  $Y_{x}$")
1214        ax5.plot(ts, st_ind6, linewidth=2., alpha=.95, color=cl[6], label=r" $S_{in}$")
1215
1216        ax5.set_ylabel(r"$S_{T}~[\textrm{-}]$")
1217        ax5.set_xlabel(r"$t~   [\textrm{hr}]$")
1218
1219        ax5.set_ylim(st_ax5_plotting_bounds)
1220        ax5.set_xlim(ts_ax5_plotting_bounds)
1221
```

```
1222     ax5.legend(loc=st_ax5_legend_bounds)
1223     fig5.show()  # ------------------- #
1224
1225 if plot_figure_6 and N == 1:
1226     fig6, ax6 = plt.subplots(nrows=1, ncols=1, sharex="all", figsize=(10, 10))
1227     ax6.plot(ts, ee_abs0, linewidth=2., alpha=.95, color=cl[0], label=r"$\mu_{m}$")
1228     ax6.plot(ts, ee_abs1, linewidth=2., alpha=.95, color=cl[1], label=r"  $K_{m}$")
1229     ax6.plot(ts, ee_abs2, linewidth=2., alpha=.95, color=cl[2], label=r"  $K_{i}$")
1230     ax6.plot(ts, ee_abs3, linewidth=2., alpha=.95, color=cl[3], label=r"    $\nu$")
1231     ax6.plot(ts, ee_abs4, linewidth=2., alpha=.95, color=cl[4], label=r"  $Y_{p}$")
1232     ax6.plot(ts, ee_abs5, linewidth=2., alpha=.95, color=cl[5], label=r"  $Y_{x}$")
1233     ax6.plot(ts, ee_abs6, linewidth=2., alpha=.95, color=cl[6], label=r" $S_{in}$")
1234
1235     ax6.set_ylabel(r"$EE~[\textrm{-}]$")
1236     ax6.set_xlabel(r"$t~[\textrm{hr}]$")
1237
1238     ax6.set_ylim(ee_ax6_plotting_bounds)
1239     ax6.set_xlim(ts_ax6_plotting_bounds)
1240
1241     ax6.legend(loc=ee_ax6_legend_bounds)
1242     fig6.show()  # ------------------- #
1243
1244 if plot_figure_7 and N == 1:
1245     fig7, ax7 = plt.subplots(nrows=1, ncols=1, sharex="all", figsize=(10, 10))
1246     ax7.plot(ts, ee_abs0, linewidth=2., alpha=.95, color=cl[0], label=r"$\mu_{m}$")
1247     ax7.fill_between(ts, ee_abs0 - 1 * ee_std0, ee_abs0 + 1 * ee_std0, linewidth=2., alpha=.2, color=cl[0])
1248     ax7.fill_between(ts, ee_abs0 - 2 * ee_std0, ee_abs0 + 2 * ee_std0, linewidth=2., alpha=.2, color=cl[0])
1249
1250     ax7.plot(ts, ee_abs1, linewidth=2., alpha=.95, color=cl[1], label=r"  $K_{m}$")
1251     ax7.fill_between(ts, ee_abs1 - 1 * ee_std1, ee_abs1 + 1 * ee_std1, linewidth=2., alpha=.2, color=cl[1])
1252     ax7.fill_between(ts, ee_abs1 - 2 * ee_std1, ee_abs1 + 2 * ee_std1, linewidth=2., alpha=.2, color=cl[1])
1253
1254     ax7.plot(ts, ee_abs2, linewidth=2., alpha=.95, color=cl[2], label=r"  $K_{i}$")
1255     ax7.fill_between(ts, ee_abs2 - 1 * ee_std2, ee_abs2 + 1 * ee_std2, linewidth=2., alpha=.2, color=cl[2])
1256     ax7.fill_between(ts, ee_abs2 - 2 * ee_std2, ee_abs2 + 2 * ee_std2, linewidth=2., alpha=.2, color=cl[2])
1257
1258     ax7.plot(ts, ee_abs3, linewidth=2., alpha=.95, color=cl[3], label=r"    $\nu$")
1259     ax7.fill_between(ts, ee_abs3 - 1 * ee_std3, ee_abs3 + 1 * ee_std3, linewidth=2., alpha=.2, color=cl[3])
1260     ax7.fill_between(ts, ee_abs3 - 2 * ee_std3, ee_abs3 + 2 * ee_std3, linewidth=2., alpha=.2, color=cl[3])
1261
1262     ax7.plot(ts, ee_abs4, linewidth=2., alpha=.95, color=cl[4], label=r"  $Y_{p}$")
1263     ax7.fill_between(ts, ee_abs4 - 1 * ee_std4, ee_abs4 + 1 * ee_std4, linewidth=2., alpha=.2, color=cl[4])
1264     ax7.fill_between(ts, ee_abs4 - 2 * ee_std4, ee_abs4 + 2 * ee_std4, linewidth=2., alpha=.2, color=cl[4])
1265
1266     ax7.plot(ts, ee_abs5, linewidth=2., alpha=.95, color=cl[5], label=r"  $Y_{x}$")
1267     ax7.fill_between(ts, ee_abs5 - 1 * ee_std5, ee_abs5 + 1 * ee_std5, linewidth=2., alpha=.2, color=cl[5])
1268     ax7.fill_between(ts, ee_abs5 - 2 * ee_std5, ee_abs5 + 2 * ee_std5, linewidth=2., alpha=.2, color=cl[5])
1269
1270     ax7.plot(ts, ee_abs6, linewidth=2., alpha=.95, color=cl[6], label=r" $S_{in}$")
1271     ax7.fill_between(ts, ee_abs6 - 1 * ee_std6, ee_abs6 + 1 * ee_std6, linewidth=2., alpha=.2, color=cl[6])
1272     ax7.fill_between(ts, ee_abs6 - 2 * ee_std6, ee_abs6 + 2 * ee_std6, linewidth=2., alpha=.2, color=cl[6])
1273
1274     ax7.set_ylabel(r"$EE~[\textrm{-}]$")
1275     ax7.set_xlabel(r"$t~[\textrm{hr}]$")
1276
1277     ax7.set_ylim(ee_ax7_plotting_bounds)
1278     ax7.set_xlim(ts_ax7_plotting_bounds)
1279
1280     ax7.legend(loc=ee_ax7_legend_bounds)
1281     fig7.show()  # ------------------- #
1282
1283 if plot_figure_8 and N == 1:
1284     fig8, ax8 = plt.subplots(nrows=1, ncols=1, sharex="all", figsize=(10, 10))
1285     ax8.plot(ts, pe_abs0, linewidth=2., alpha=.95, color=cl[0], label=r"$\mu_{m}$")
1286     ax8.plot(ts, pe_abs1, linewidth=2., alpha=.95, color=cl[1], label=r"  $K_{m}$")
1287     ax8.plot(ts, pe_abs2, linewidth=2., alpha=.95, color=cl[2], label=r"  $K_{i}$")
1288     ax8.plot(ts, pe_abs3, linewidth=2., alpha=.95, color=cl[3], label=r"    $\nu$")
1289     ax8.plot(ts, pe_abs4, linewidth=2., alpha=.95, color=cl[4], label=r"  $Y_{p}$")
1290     ax8.plot(ts, pe_abs5, linewidth=2., alpha=.95, color=cl[5], label=r"  $Y_{x}$")
1291     ax8.plot(ts, pe_abs6, linewidth=2., alpha=.95, color=cl[6], label=r" $S_{in}$")
1292
1293     ax8.set_ylabel(r"$PE~[\textrm{-}]$")
1294     ax8.set_xlabel(r"$t~[\textrm{hr}]$")
1295
1296     ax8.set_ylim(pe_ax8_plotting_bounds)
1297     ax8.set_xlim(ts_ax8_plotting_bounds)
1298
1299     ax8.legend(loc=pe_ax8_legend_bounds)
1300     fig8.show()  # ------------------- #
1301
1302 if plot_figure_9 and N == 1:
1303     fig9, ax9 = plt.subplots(nrows=1, ncols=1, sharex="all", figsize=(10, 10))
1304     ax9.plot(ts, pe_abs0, linewidth=2., alpha=.95, color=cl[0], label=r"$\mu_{m}$")
1305     ax9.fill_between(ts, pe_abs0 - 1 * pe_std0, pe_abs0 + 1 * pe_std0, linewidth=2., alpha=.2, color=cl[0])
1306     ax9.fill_between(ts, pe_abs0 - 2 * pe_std0, pe_abs0 + 2 * pe_std0, linewidth=2., alpha=.2, color=cl[0])
1307
1308     ax9.plot(ts, pe_abs1, linewidth=2., alpha=.95, color=cl[1], label=r"  $K_{m}$")
1309     ax9.fill_between(ts, pe_abs1 - 1 * pe_std1, pe_abs1 + 1 * pe_std1, linewidth=2., alpha=.2, color=cl[1])
1310     ax9.fill_between(ts, pe_abs1 - 2 * pe_std1, pe_abs1 + 2 * pe_std1, linewidth=2., alpha=.2, color=cl[1])
1311
1312     ax9.plot(ts, pe_abs2, linewidth=2., alpha=.95, color=cl[2], label=r"  $K_{i}$")
1313     ax9.fill_between(ts, pe_abs2 - 1 * pe_std2, pe_abs2 + 1 * pe_std2, linewidth=2., alpha=.2, color=cl[2])
1314     ax9.fill_between(ts, pe_abs2 - 2 * pe_std2, pe_abs2 + 2 * pe_std2, linewidth=2., alpha=.2, color=cl[2])
1315
1316     ax9.plot(ts, pe_abs3, linewidth=2., alpha=.95, color=cl[3], label=r"    $\nu$")
```

```
    ax9.fill_between(ts, pe_abs3 - 1 * pe_std3, pe_abs3 + 1 * pe_std3, linewidth=2., alpha=.2, color=cl[3])
    ax9.fill_between(ts, pe_abs3 - 2 * pe_std3, pe_abs3 + 2 * pe_std3, linewidth=2., alpha=.2, color=cl[3])

    ax9.plot(ts, pe_abs4, linewidth=2., alpha=.95, color=cl[4], label=r"  $Y_{p}$")
    ax9.fill_between(ts, pe_abs4 - 1 * pe_std4, pe_abs4 + 1 * pe_std4, linewidth=2., alpha=.2, color=cl[4])
    ax9.fill_between(ts, pe_abs4 - 2 * pe_std4, pe_abs4 + 2 * pe_std4, linewidth=2., alpha=.2, color=cl[4])

    ax9.plot(ts, pe_abs5, linewidth=2., alpha=.95, color=cl[5], label=r"  $Y_{x}$")
    ax9.fill_between(ts, pe_abs5 - 1 * pe_std5, pe_abs5 + 1 * pe_std5, linewidth=2., alpha=.2, color=cl[5])
    ax9.fill_between(ts, pe_abs5 - 2 * pe_std5, pe_abs5 + 2 * pe_std5, linewidth=2., alpha=.2, color=cl[5])

    ax9.plot(ts, pe_abs6, linewidth=2., alpha=.95, color=cl[6], label=r" $S_{in}$")
    ax9.fill_between(ts, pe_abs6 - 1 * pe_std6, pe_abs6 + 1 * pe_std6, linewidth=2., alpha=.2, color=cl[6])
    ax9.fill_between(ts, pe_abs6 - 2 * pe_std6, pe_abs6 + 2 * pe_std6, linewidth=2., alpha=.2, color=cl[6])

    ax9.set_ylabel(r"$PE~[\textrm{-}]$")
    ax9.set_xlabel(r"$t~[\textrm{hr}]$")

    ax9.set_ylim(pe_ax9_plotting_bounds)
    ax9.set_xlim(ts_ax9_plotting_bounds)

    ax9.legend(loc=pe_ax9_legend_bounds)
    fig9.show()  # ------------------ #
```

Listing 6.1: main.py - The main-loop for all the MPCs

The optimization for all the MPCs

```
import numpy as np
import casadi as cd

# Declaring 1st state symbolic
X = cd.MX.sym("X", 1)
# Declaring 2nd state symbolic
S = cd.MX.sym("S", 1)
# Declaring 3rd state symbolic
P = cd.MX.sym("P", 1)
# Declaring 4th state symbolic
V = cd.MX.sym("V", 1)

# Declaring 1st parameter symbolic
mu_m = cd.MX.sym("mu_m", 1)
# Declaring 2nd parameter symbolic
K_m = cd.MX.sym("K_m", 1)
# Declaring 3rd parameter symbolic
K_i = cd.MX.sym("K_i", 1)
# Declaring 4th parameter symbolic
nu = cd.MX.sym("nu", 1)
# Declaring 5th parameter symbolic
Y_p = cd.MX.sym("Y_p", 1)
# Declaring 6th parameter symbolic
Y_x = cd.MX.sym("Y_x", 1)
# Declaring 7th parameter symbolic
S_in = cd.MX.sym("S_in", 1)

# Concatenate symbolic states
x = cd.vertcat(X, S, P, V)
# Concatenate symbolic parameters
p = cd.vertcat(mu_m, K_m, K_i, nu, Y_p, Y_x, S_in)
# Declaring the input symbolic
u = cd.MX.sym("u", 1)
# Declaring MV-change symbolic
du = cd.MX.sym("du", 1)
# Declaring time-span symbolic
t = cd.MX.sym("t", 1)


# Defining ode_model()
def ode_model():
    # Declaring the kinetic model
    mu = (mu_m * S) / (K_m + S + ((S ** 2) / K_i))
    # Declaring the biomass equation
    dXdt = mu * X - (u / V) * X
    # Declaring the substrate equation
    dSdt = -(mu * X) / Y_x - (nu * X) / Y_p + (u / V) * (S_in - S)
    # Declaring the product equation
    dPdt = nu * X - (u / V) * P
    # Declaring the volume equation
    dVdt = u
    # Returning these ODEs together
    return cd.vertcat(dXdt, dSdt, dPdt, dVdt)


# Defining scenarios2()
def scenarios2(par_a, par_b):
    scens = cd.vertcat([])  # params combinations
    scens_count = par_a.shape[0] * par_b.shape[0]
    for i in range(par_a.shape[0]):
        for j in range(par_b.shape[0]):
            scens = cd.vertcat(scens, cd.horzcat(par_a[i], par_b[j]))
    return scens, scens_count

```

```
65
66 # Defining scenarios3()
67 def scenarios3(par_a, par_b, par_c):
68     scens = cd.vertcat([])  # three uncertain params combinations
69     scens_count = par_a.shape[0] * par_b.shape[0] * par_c.shape[0]
70     for i in range(par_a.shape[0]):
71         for j in range(par_b.shape[0]):
72             for k in range(par_b.shape[0]):
73                 scens = cd.vertcat(scens, cd.horzcat(par_a[i], par_b[j], par_c[k]))
74     return scens, scens_count
75
76
77 # Defining solver_nmpc()
78 def solver_nmpc(n_prd, n_ctr):
79     x0 = cd.MX.sym("x0_sym", 4)
80     tk = cd.MX.sym("tk_sym", 2)
81     u0 = cd.MX.sym("u0_sym", 1)
82     p0 = cd.MX.sym("p0_sym", 7)
83     p_var = cd.vertcat(x0, tk, u0, p0)
84
85     dg = 3  # orthogonal collocation with 3 points each element
86     tau_root = np.append(0, cd.collocation_points(dg, "radau"))
87     b = np.zeros((dg + 1, 1))
88     c = np.zeros((dg + 1, dg + 1))
89     d = np.zeros((dg + 1, 1))
90
91     for i in range(dg + 1):
92         coeff = 1
93         # Construct Lagrange polynomials to get the
94         # polynomial basis at the collocation point.
95         for r in range(dg + 1):
96             if r != i:
97                 coeff = np.convolve(coeff, [1., -tau_root[r]])
98                 coeff = coeff / (tau_root[i] - tau_root[r])
99
100         # Evaluate the polynomial at the final time to
101         # get coefficients of the continuity equation.
102         d[i] = np.polyval(coeff, 1.)
103
104         # Evaluate time derivative of the polynomial at all collocation
105         # points to obtain the coefficients of the continuity equation.
106         pder = np.polyder(coeff)
107         for r in range(dg + 1):
108             c[i][r] = np.polyval(pder, tau_root[r])
109
110         # Evaluate the integral of the polynomial to
111         # get coefficients of the quadrature function.
112         pint = np.polyint(coeff)
113         b[i] = np.polyval(pint, 1.)
114
115     # Declare matrix for the economic objective
116     Q = np.array([[0., 0., 0., 0.],
117                   [0., 0., 0., 0.],
118                   [0., 0., -1., 0.],
119                   [0., 0., 0., 0.]])
120
121     # Declare matrix for penalizing MV-changes
122     R = np.array([0.])  # same as do-mpc used!
123
124     # Declaring the horizons; npr,nct
125     npr, nct = n_prd, n_ctr
126
127     # Obtaining the defined ODE-model
128     sys = ode_model()
129
130     # Declare the economic objective function
131     J = (x.T @ Q @ x + du.T @ R @ du)
132
133     # Declare cd.Function "f" for later application
134     f = cd.Function("f", [x, u, p, t, du], [sys, J])
135
136     # Declare an empty NLP, i.e., empty vertcats + bounds
137     w, w0 = cd.vertcat([]), cd.vertcat([])  # states, inputs
138     lbw, ubw = cd.vertcat([]), cd.vertcat([])  # states, inputs
139     g1, lbg1, ubg1 = cd.vertcat([]), cd.vertcat([]), cd.vertcat([])
140     g2, lbg2, ubg2 = cd.vertcat([]), cd.vertcat([]), cd.vertcat([])
141
142     # Declare state constraints; X_min, S_min, P_min, V_min
143     X_min, S_min, P_min, V_min = 0., -.01, 0., 0.
144     # Declare state constraints; X_max, S_max, P_max, V_max
145     X_max, S_max, P_max, V_max = 3.7, np.inf, 3.0, np.inf
146
147     # Concatenate these minimum constraints on the states
148     x_min = cd.vertcat(X_min, S_min, P_min, V_min)
149     # Concatenate these maximum constraints on the states
150     x_max = cd.vertcat(X_max, S_max, P_max, V_max)
151
152     # Declaring the constraints; u_min, u_max, du_max
153     u_min, u_max, du_max = 0., .2, .0035
154
155     # Declare x_plt, u_plt for the trajectory split
156     x_plt, u_plt = cd.horzcat([]), cd.horzcat([])
157
158     # Initialize the economic objective function
159     J = 0
```

```
160
161     for k in range(npr):
162         if k == 0:  # at the initial point
163             # 'Lift' the initial conditions
164             xk = cd.MX.sym("x_" + str(k), 4)
165             w = cd.vertcat(w, xk)
166             w0 = cd.vertcat(w0, x0)
167             lbw = cd.vertcat(lbw, x0)
168             ubw = cd.vertcat(ubw, x0)
169             x_plt = cd.horzcat(x_plt, xk)
170
171         # New NLP variable for control
172         uk = cd.MX.sym("u_" + str(k), 1)
173         w = cd.vertcat(w, uk)
174         w0 = cd.vertcat(w0, u0)
175         lbw = cd.vertcat(lbw, u_min)
176         ubw = cd.vertcat(ubw, u_max)
177         u_plt = cd.horzcat(u_plt, uk)
178
179         # If-sentence for finding duk
180         if k <= (nct - 1):
181             if k == 0:
182                 duk = uk - u0
183             else:
184                 duk = uk - uk0
185             g2 = cd.vertcat(g2, duk)
186             lbg2 = cd.vertcat(lbg2, -du_max)
187             ubg2 = cd.vertcat(ubg2, du_max)
188         else:
189             duk = uk - uk0
190             g2 = cd.vertcat(g2, duk)
191             lbg2 = cd.vertcat(lbg2, 0.)
192             ubg2 = cd.vertcat(ubg2, 0.)
193
194         # If-sentence for finding uk0
195         if k != (npr - 1):
196             uk0 = uk
197         else:
198             uk0 = uk0
199
200         # States at the collocation points
201         xki = [[] * i for i in range(dg)]
202         for i in range(dg):
203             xki[i] = cd.MX.sym("x_" + str(k) + "_" + str(i), 4)
204             w = cd.vertcat(w, xki[i])
205             w0 = cd.vertcat(w0, x0)
206             lbw = cd.vertcat(lbw, x_min)
207             ubw = cd.vertcat(ubw, x_max)
208
209         # Loop over the collocation points
210         xk_end = d[0] * xk
211         for i in range(dg):
212             # State derivative collocation points
213             xp = c[0, i + 1] * xk
214             for r in range(dg):
215                 xp += c[r + 1, i + 1] * xki[r]
216
217             # Append collocation equations
218             dt = tk[1] - tk[0]  # the time-step
219             fi, qi = f(xki[i], uk, p0, dt, duk)
220             g1 = cd.vertcat(g1, dt * fi - xp)
221             lbg1 = cd.vertcat(lbg1, 0., 0., 0., 0.)
222             ubg1 = cd.vertcat(ubg1, 0., 0., 0., 0.)
223
224             # Add contribution to the end state
225             xk_end += d[i + 1] * xki[i]
226
227             # Add contribution to quad function
228             J += b[i + 1] * qi * dt
229
230         # New NLP variable for states at end
231         xk = cd.MX.sym("x_" + str(k + 1), 4)
232         w = cd.vertcat(w, xk)
233         w0 = cd.vertcat(w0, x0)
234         lbw = cd.vertcat(lbw, x_min)
235         ubw = cd.vertcat(ubw, x_max)
236         x_plt = cd.horzcat(x_plt, xk)
237
238         # Add the equality constraints
239         g1 = cd.vertcat(g1, xk_end - xk)
240         lbg1 = cd.vertcat(lbg1, 0., 0., 0., 0.)
241         ubg1 = cd.vertcat(ubg1, 0., 0., 0., 0.)
242
243     # Obtaining the trajectories from 'w'
244     trajects = cd.Function("trajectory", [w], [x_plt, u_plt], ["w"], ["x", "u"])
245
246     # Formalize this into the NLP problem
247     prob = {"x": cd.vertcat(w), "g": cd.vertcat(g1, g2), "f": J, "p": p_var}
248
249     # We may create the option dictionary
250     opts = {"ipopt.max_iter": 200, "ipopt.print_level": 0, "print_time": 0,
251             "ipopt.tol": 1.0e-6, "ipopt.acceptable_tol": 1.0e-4}
252
253     # Assign solver - 'IPOPT' in our case
254     solver = cd.nlpsol("solver", "ipopt", prob, opts)
```

```python

    # Converting from cd.MX into ndarray
    w0_conv = cd.Function("w0_conv", [x0, tk, u0, p0], [w0])
    lbw_conv = cd.Function("lbw_conv", [x0, tk, u0, p0], [lbw])
    ubw_conv = cd.Function("ubw_conv", [x0, tk, u0, p0], [ubw])

    # Converting from cd.MX into ndarray
    lbg1, ubg1 = np.array(lbg1).flatten(), np.array(ubg1).flatten()
    lbg2, ubg2 = np.array(lbg2).flatten(), np.array(ubg2).flatten()
    lbg = np.concatenate((lbg1, lbg2))  # concatenate lower constraints
    ubg = np.concatenate((ubg1, ubg2))  # concatenate upper constraints

    # Return trajects+solver together with the symbolic containers
    return trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg


# Defining solver_rmpc2()
def solver_rmpc2(par_a, par_b):
    x0 = cd.MX.sym("x0_sym", 4)
    tk = cd.MX.sym("tk_sym", 2)
    u0 = cd.MX.sym("u0_sym", 1)
    p0 = cd.MX.sym("p0_sym", 7)
    p_var = cd.vertcat(x0, tk, u0, p0)

    # possible uncertain parameters considered for MS-MPC
    unc_mu_m = cd.vertcat(0.85 * p0[0], p0[0], 1.15 * p0[0])
    unc__K_m = cd.vertcat(0.85 * p0[1], p0[1], 1.15 * p0[1])
    unc__K_i = cd.vertcat(0.85 * p0[2], p0[2], 1.15 * p0[2])
    unc___nu = cd.vertcat(0.85 * p0[3], p0[3], 1.15 * p0[3])
    unc__Y_p = cd.vertcat(0.85 * p0[4], p0[4], 1.15 * p0[4])
    unc__Y_x = cd.vertcat(0.85 * p0[5], p0[5], 1.15 * p0[5])
    unc_S_in = cd.vertcat(0.85 * p0[6], p0[6], 1.15 * p0[6])

    if par_a == "mu_m" and par_b == "K_m":
        scenario, scenario_count = scenarios2(unc_mu_m, unc__K_m)
    elif par_a == "mu_m" and par_b == "K_i":
        scenario, scenario_count = scenarios2(unc_mu_m, unc__K_i)
    elif par_a == "mu_m" and par_b == "nu":
        scenario, scenario_count = scenarios2(unc_mu_m, unc___nu)
    elif par_a == "mu_m" and par_b == "Y_p":
        scenario, scenario_count = scenarios2(unc_mu_m, unc__Y_p)
    elif par_a == "mu_m" and par_b == "Y_x":
        scenario, scenario_count = scenarios2(unc_mu_m, unc__Y_x)
    elif par_a == "mu_m" and par_b == "S_in":
        scenario, scenario_count = scenarios2(unc_mu_m, unc_S_in)
    elif par_a == "K_m" and par_b == "K_i":
        scenario, scenario_count = scenarios2(unc__K_m, unc__K_i)
    elif par_a == "K_m" and par_b == "nu":
        scenario, scenario_count = scenarios2(unc__K_m, unc___nu)
    elif par_a == "K_m" and par_b == "Y_p":
        scenario, scenario_count = scenarios2(unc__K_m, unc__Y_p)
    elif par_a == "K_m" and par_b == "Y_x":
        scenario, scenario_count = scenarios2(unc__K_m, unc__Y_x)
    elif par_a == "K_m" and par_b == "S_in":
        scenario, scenario_count = scenarios2(unc__K_m, unc_S_in)
    elif par_a == "K_i" and par_b == "nu":
        scenario, scenario_count = scenarios2(unc__K_i, unc___nu)
    elif par_a == "K_i" and par_b == "Y_p":
        scenario, scenario_count = scenarios2(unc__K_i, unc__Y_p)
    elif par_a == "K_i" and par_b == "Y_x":
        scenario, scenario_count = scenarios2(unc__K_i, unc__Y_x)
    elif par_a == "K_i" and par_b == "S_in":
        scenario, scenario_count = scenarios2(unc__K_i, unc_S_in)
    elif par_a == "nu" and par_b == "Y_p":
        scenario, scenario_count = scenarios2(unc___nu, unc__Y_p)
    elif par_a == "nu" and par_b == "Y_x":
        scenario, scenario_count = scenarios2(unc___nu, unc__Y_x)
    elif par_a == "nu" and par_b == "S_in":
        scenario, scenario_count = scenarios2(unc___nu, unc_S_in)
    elif par_a == "Y_p" and par_b == "Y_x":
        scenario, scenario_count = scenarios2(unc__Y_p, unc__Y_x)
    elif par_a == "Y_p" and par_b == "S_in":
        scenario, scenario_count = scenarios2(unc__Y_p, unc_S_in)
    elif par_a == "Y_x" and par_b == "S_in":
        scenario, scenario_count = scenarios2(unc__Y_x, unc_S_in)
    else:  # if there is given an invalid parameter combination
        raise IndexError("The given combination doesn't exist")

    dg = 3  # orthogonal collocation with 3 points each element
    tau_root = np.append(0, cd.collocation_points(dg, "radau"))
    b = np.zeros((dg + 1, 1))
    c = np.zeros((dg + 1, dg + 1))
    d = np.zeros((dg + 1, 1))

    for i in range(dg + 1):
        coeff = 1
        # Construct Lagrange polynomials to get the
        # polynomial basis at the collocation point.
        for r in range(dg + 1):
            if r != i:
                coeff = np.convolve(coeff, [1., -tau_root[r]])
                coeff = coeff / (tau_root[i] - tau_root[r])

        # Evaluate the polynomial at the final time to
        # get coefficients of the continuity equation.
```

```
350         d[i] = np.polyval(coeff, 1.)
351 
352         # Evaluate time derivative of the polynomial at all collocation
353         # points to obtain the coefficients of the continuity equation.
354         pder = np.polyder(coeff)
355         for r in range(dg + 1):
356             c[i][r] = np.polyval(pder, tau_root[r])
357 
358         # Evaluate the integral of the polynomial to
359         # get coefficients of the quadrature function.
360         pint = np.polyint(coeff)
361         b[i] = np.polyval(pint, 1.)
362 
363     # Declare matrix for the economic objective
364     Q = np.array([[0., 0., 0., 0.],
365                   [0., 0., 0., 0.],
366                   [0., 0., -1., 0.],
367                   [0., 0., 0., 0.]])
368 
369     # Declare matrix for penalizing MV-changes
370     R = np.array([0.])  # same as do-mpc used!
371 
372     # Declare the horizons; npr,nct,nrb
373     npr, nct, nrb = 20, 3, 1
374 
375     # Finding the number of scenarios
376     levels = scenario_count
377     nsc = levels ** nrb
378 
379     # Assuming scenario weights equal
380     omega = 1. / nsc
381 
382     # Obtaining the defined ODE-model
383     sys = ode_model()
384 
385     # Declare the economic objective function
386     J = omega * (x.T @ Q @ x + du.T @ R @ du)
387 
388     # Declare cd.Function "f" for later application
389     f = cd.Function("f", [x, u, p, t, du], [sys, J])
390 
391     # Declare an empty NLP, i.e., empty vertcats + bounds
392     w, w0 = cd.vertcat([]), cd.vertcat([])  # states, inputs
393     lbw, ubw = cd.vertcat([]), cd.vertcat([])  # states, inputs
394     g1, lbg1, ubg1 = cd.vertcat([]), cd.vertcat([]), cd.vertcat([])
395     g2, lbg2, ubg2 = cd.vertcat([]), cd.vertcat([]), cd.vertcat([])
396     g3, lbg3, ubg3 = cd.vertcat([]), cd.vertcat([]), cd.vertcat([])
397 
398     # Declare state constraints; X_min, S_min, P_min, V_min
399     X_min, S_min, P_min, V_min = 0., -.01, 0., 0.
400     # Declare state constraints; X_max, S_max, P_max, V_max
401     X_max, S_max, P_max, V_max = 3.7, np.inf, 3.0, np.inf
402 
403     # Concatenate these minimum constraints on the states
404     x_min = cd.vertcat(X_min, S_min, P_min, V_min)
405     # Concatenate these maximum constraints on the states
406     x_max = cd.vertcat(X_max, S_max, P_max, V_max)
407 
408     # Declaring the constraints; u_min, u_max, du_max
409     u_min, u_max, du_max = 0., .2, .0035
410 
411     # Declare x_plt, u_plt for the trajectory split
412     x_plt, u_plt = cd.horzcat([]), cd.horzcat([])
413 
414     # Initialize the economic objective function
415     J = 0
416 
417     for j in range(nsc):
418         if par_a == "mu_m" and par_b == "K_m":
419             pj = cd.vertcat(cd.horzcat(  # assumed 5 certain 2 uncertain parameters
420                 scenario[j, 0], scenario[j, 1], p0[2], p0[3], p0[4], p0[5], p0[6]))
421         elif par_a == "mu_m" and par_b == "K_i":
422             pj = cd.vertcat(cd.horzcat(  # assumed 5 certain 2 uncertain parameters
423                 scenario[j, 0], p0[1], scenario[j, 1], p0[3], p0[4], p0[5], p0[6]))
424         elif par_a == "mu_m" and par_b == "nu":
425             pj = cd.vertcat(cd.horzcat(  # assumed 5 certain 2 uncertain parameters
426                 scenario[j, 0], p0[1], p0[2], scenario[j, 1], p0[4], p0[5], p0[6]))
427         elif par_a == "mu_m" and par_b == "Y_p":
428             pj = cd.vertcat(cd.horzcat(  # assumed 5 certain 2 uncertain parameters
429                 scenario[j, 0], p0[1], p0[2], p0[3], scenario[j, 1], p0[5], p0[6]))
430         elif par_a == "mu_m" and par_b == "Y_x":
431             pj = cd.vertcat(cd.horzcat(  # assumed 5 certain 2 uncertain parameters
432                 scenario[j, 0], p0[1], p0[2], p0[3], p0[4], scenario[j, 1], p0[6]))
433         elif par_a == "mu_m" and par_b == "S_in":
434             pj = cd.vertcat(cd.horzcat(  # assumed 5 certain 2 uncertain parameters
435                 scenario[j, 0], p0[1], p0[2], p0[3], p0[4], p0[5], scenario[j, 1]))
436         elif par_a == "K_m" and par_b == "K_i":
437             pj = cd.vertcat(cd.horzcat(  # assumed 5 certain 2 uncertain parameters
438                 p0[0], scenario[j, 0], scenario[j, 1], p0[3], p0[4], p0[5], p0[6]))
439         elif par_a == "K_m" and par_b == "nu":
440             pj = cd.vertcat(cd.horzcat(  # assumed 5 certain 2 uncertain parameters
441                 p0[0], scenario[j, 0], p0[2], scenario[j, 1], p0[4], p0[5], p0[6]))
442         elif par_a == "K_m" and par_b == "Y_p":
443             pj = cd.vertcat(cd.horzcat(  # assumed 5 certain 2 uncertain parameters
444                 p0[0], scenario[j, 0], p0[2], p0[3], scenario[j, 1], p0[5], p0[6]))
```

```
445         elif par_a == "K_m" and par_b == "Y_x":
446             pj = cd.vertcat(cd.horzcat(  # assumed 5 certain 2 uncertain parameters
447                 p0[0], scenario[j, 0], p0[2], p0[3], p0[4], scenario[j, 1], p0[6]))
448         elif par_a == "K_m" and par_b == "S_in":
449             pj = cd.vertcat(cd.horzcat(  # assumed 5 certain 2 uncertain parameters
450                 p0[0], scenario[j, 0], p0[2], p0[3], p0[4], p0[5], scenario[j, 1]))
451         elif par_a == "K_i" and par_b == "nu":
452             pj = cd.vertcat(cd.horzcat(  # assumed 5 certain 2 uncertain parameters
453                 p0[0], p0[1], scenario[j, 0], scenario[j, 1], p0[4], p0[5], p0[6]))
454         elif par_a == "K_i" and par_b == "Y_p":
455             pj = cd.vertcat(cd.horzcat(  # assumed 5 certain 2 uncertain parameters
456                 p0[0], p0[1], scenario[j, 0], p0[3], scenario[j, 1], p0[5], p0[6]))
457         elif par_a == "K_i" and par_b == "Y_x":
458             pj = cd.vertcat(cd.horzcat(  # assumed 5 certain 2 uncertain parameters
459                 p0[0], p0[1], scenario[j, 0], p0[3], p0[4], scenario[j, 1], p0[6]))
460         elif par_a == "K_i" and par_b == "S_in":
461             pj = cd.vertcat(cd.horzcat(  # assumed 5 certain 2 uncertain parameters
462                 p0[0], p0[1], scenario[j, 0], p0[3], p0[4], p0[5], scenario[j, 1]))
463         elif par_a == "nu" and par_b == "Y_p":
464             pj = cd.vertcat(cd.horzcat(  # assumed 5 certain 2 uncertain parameters
465                 p0[0], p0[1], p0[2], scenario[j, 0], scenario[j, 1], p0[5], p0[6]))
466         elif par_a == "nu" and par_b == "Y_x":
467             pj = cd.vertcat(cd.horzcat(  # assumed 5 certain 2 uncertain parameters
468                 p0[0], p0[1], p0[2], scenario[j, 0], p0[4], scenario[j, 1], p0[6]))
469         elif par_a == "nu" and par_b == "S_in":
470             pj = cd.vertcat(cd.horzcat(  # assumed 5 certain 2 uncertain parameters
471                 p0[0], p0[1], p0[2], scenario[j, 0], p0[4], p0[5], scenario[j, 1]))
472         elif par_a == "Y_p" and par_b == "Y_x":
473             pj = cd.vertcat(cd.horzcat(  # assumed 5 certain 2 uncertain parameters
474                 p0[0], p0[1], p0[2], p0[3], scenario[j, 0], scenario[j, 1], p0[6]))
475         elif par_a == "Y_p" and par_b == "S_in":
476             pj = cd.vertcat(cd.horzcat(  # assumed 5 certain 2 uncertain parameters
477                 p0[0], p0[1], p0[2], p0[3], scenario[j, 0], p0[5], scenario[j, 1]))
478         elif par_a == "Y_x" and par_b == "S_in":
479             pj = cd.vertcat(cd.horzcat(  # assumed 5 certain 2 uncertain parameters
480                 p0[0], p0[1], p0[2], p0[3], p0[4], scenario[j, 0], scenario[j, 1]))
481         else:  # if there is given an invalid parameter combination
482             raise IndexError("The given combination doesn't exist")
483 
484         for k in range(npr):
485             if k == 0:  # at the initial point
486                 # 'Lift' the initial conditions
487                 xkj = cd.MX.sym("x_" + str(k) + "_" + str(j), 4)
488                 w = cd.vertcat(w, xkj)
489                 w0 = cd.vertcat(w0, x0)
490                 lbw = cd.vertcat(lbw, x0)
491                 ubw = cd.vertcat(ubw, x0)
492                 x_plt = cd.horzcat(x_plt, xkj)
493 
494             if k == 0 and j == 0:
495                 # New NLP variable for control
496                 u00 = cd.MX.sym("u_" + str(k) + "_" + str(j), 1)
497                 w = cd.vertcat(w, u00)
498                 w0 = cd.vertcat(w0, u0)
499                 lbw = cd.vertcat(lbw, u_min)
500                 ubw = cd.vertcat(ubw, u_max)
501                 u_plt = cd.horzcat(u_plt, u00)
502             else:
503                 # New NLP variable for control
504                 ukj = cd.MX.sym("u_" + str(k) + "_" + str(j), 1)
505                 w = cd.vertcat(w, ukj)
506                 w0 = cd.vertcat(w0, u0)
507                 lbw = cd.vertcat(lbw, u_min)
508                 ubw = cd.vertcat(ubw, u_max)
509                 u_plt = cd.horzcat(u_plt, ukj)
510 
511             # If-sentence for finding dukj
512             if k <= (nct - 1):
513                 if k == 0 and j == 0:
514                     dukj = u00 - u0
515                 elif k == 0 and j != 0:
516                     dukj = ukj - u0
517                 else:
518                     dukj = ukj - ukj0
519                 g2 = cd.vertcat(g2, dukj)
520                 lbg2 = cd.vertcat(lbg2, -du_max)
521                 ubg2 = cd.vertcat(ubg2, du_max)
522             else:
523                 dukj = ukj - ukj0
524                 g2 = cd.vertcat(g2, dukj)
525                 lbg2 = cd.vertcat(lbg2, 0.)
526                 ubg2 = cd.vertcat(ubg2, 0.)
527 
528             # If-sentence for finding ukj0
529             if k != (npr - 1):
530                 if k == 0 and j == 0:
531                     ukj0 = u00
532                 else:
533                     ukj0 = ukj
534             else:
535                 ukj0 = ukj0
536 
537             # States at the collocation points
538             xkji = [[] * i for i in range(dg)]
539             for i in range(dg):
```

```
540                 xkji[i] = cd.MX.sym("x_" + str(k) + "_" + str(j) + "_" + str(i), 4)
541                 w = cd.vertcat(w, xkji[i])
542                 w0 = cd.vertcat(w0, x0)
543                 lbw = cd.vertcat(lbw, x_min)
544                 ubw = cd.vertcat(ubw, x_max)
545 
546             # Loop over the collocation points
547             xkj_end = d[0] * xkj
548             for i in range(dg):
549                 # State derivative collocation points
550                 xp = c[0, i + 1] * xkj
551                 for r in range(dg):
552                     xp += c[r + 1, i + 1] * xkji[r]
553 
554                 # Append collocation equations
555                 dt = tk[1] - tk[0]  # current time-step
556                 if k == 0 and j == 0:
557                     fi, qi = f(xkji[i], u00, pj, dt, dukj)
558                 else:
559                     fi, qi = f(xkji[i], ukj, pj, dt, dukj)
560                 g1 = cd.vertcat(g1, dt * fi - xp)
561                 lbg1 = cd.vertcat(lbg1, 0., 0., 0., 0.)
562                 ubg1 = cd.vertcat(ubg1, 0., 0., 0., 0.)
563 
564                 # Add contribution to the end state
565                 xkj_end += d[i + 1] * xkji[i]
566 
567                 # Add contribution to quad function
568                 J += b[i + 1] * qi * dt
569 
570             # New NLP variable for states at the end of interval
571             xkj = cd.MX.sym("x_" + str(k + 1) + "_" + str(j), 4)
572             w = cd.vertcat(w, xkj)
573             w0 = cd.vertcat(w0, x0)
574             lbw = cd.vertcat(lbw, x_min)
575             ubw = cd.vertcat(ubw, x_max)
576             x_plt = cd.horzcat(x_plt, xkj)
577 
578             # Add the equality constraints
579             g1 = cd.vertcat(g1, xkj_end - xkj)
580             lbg1 = cd.vertcat(lbg1, 0., 0., 0., 0.)
581             ubg1 = cd.vertcat(ubg1, 0., 0., 0., 0.)
582 
583             # Add the non-anticipativity constraints
584             if k == 0 and j == 0:  # only 1st scenario
585                 g3 = cd.vertcat(g3, u00 - u00)
586                 lbg3 = cd.vertcat(lbg3, 0.)
587                 ubg3 = cd.vertcat(ubg3, 0.)
588             elif k == 0 and j != 0:  # other scenarios
589                 g3 = cd.vertcat(g3, ukj - u00)
590                 lbg3 = cd.vertcat(lbg3, 0.)
591                 ubg3 = cd.vertcat(ubg3, 0.)
592 
593     # Obtaining the trajectories from 'w'
594     trajects = cd.Function("trajectory", [w], [x_plt, u_plt], ["w"], ["x", "u"])
595 
596     # Formalize this into the NLP problem
597     prob = {"x": cd.vertcat(w), "g": cd.vertcat(g1, g2, g3), "f": J, "p": p_var}
598 
599     # We may create the option dictionary
600     opts = {"ipopt.max_iter": 200, "ipopt.print_level": 0, "print_time": 0,
601             "ipopt.tol": 1.0e-6, "ipopt.acceptable_tol": 1.0e-4}
602 
603     # Assign solver - 'IPOPT' in our case
604     solver = cd.nlpsol("solver", "ipopt", prob, opts)
605 
606     # Converting from cd.MX into ndarray
607     w0_conv = cd.Function("w0_conv", [x0, tk, u0, p0], [w0])
608     lbw_conv = cd.Function("lbw_conv", [x0, tk, u0, p0], [lbw])
609     ubw_conv = cd.Function("ubw_conv", [x0, tk, u0, p0], [ubw])
610 
611     # Converting from cd.MX into ndarray
612     lbg1, ubg1 = np.array(lbg1).flatten(), np.array(ubg1).flatten()
613     lbg2, ubg2 = np.array(lbg2).flatten(), np.array(ubg2).flatten()
614     lbg3, ubg3 = np.array(lbg3).flatten(), np.array(ubg3).flatten()
615     lbg = np.concatenate((lbg1, lbg2, lbg3))  # concatenate lower constraints
616     ubg = np.concatenate((ubg1, ubg2, ubg3))  # concatenate upper constraints
617 
618     # Return trajects+solver together with the symbolic containers
619     return trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg
620 
621 
622 # Defining solver_rmpc3()
623 def solver_rmpc3(par_a, par_b, par_c):
624     x0 = cd.MX.sym("x0_sym", 4)
625     tk = cd.MX.sym("tk_sym", 2)
626     u0 = cd.MX.sym("u0_sym", 1)
627     p0 = cd.MX.sym("p0_sym", 7)
628     p_var = cd.vertcat(x0, tk, u0, p0)
629 
630     # possible uncertain parameters considered for MS-MPC
631     unc_mu_m = cd.vertcat(0.85 * p0[0], p0[0], 1.15 * p0[0])
632     unc__K_m = cd.vertcat(0.85 * p0[1], p0[1], 1.15 * p0[1])
633     unc__K_i = cd.vertcat(0.85 * p0[2], p0[2], 1.15 * p0[2])
634     unc___nu = cd.vertcat(0.85 * p0[3], p0[3], 1.15 * p0[3])
```

```
635     unc__Y_p = cd.vertcat(0.85 * p0[4], p0[4], 1.15 * p0[4])
636     unc__Y_x = cd.vertcat(0.85 * p0[5], p0[5], 1.15 * p0[5])
637     unc_S_in = cd.vertcat(0.85 * p0[6], p0[6], 1.15 * p0[6])
638 
639     if par_a == "mu_m" and par_b == "K_m" and par_c == "K_i":
640         scenario, scenario_count = scenarios3(unc_mu_m, unc__K_m, unc__K_i)
641     elif par_a == "mu_m" and par_b == "K_m" and par_c == "nu":
642         scenario, scenario_count = scenarios3(unc_mu_m, unc__K_m, unc___nu)
643     elif par_a == "mu_m" and par_b == "K_m" and par_c == "Y_p":
644         scenario, scenario_count = scenarios3(unc_mu_m, unc__K_m, unc__Y_p)
645     elif par_a == "mu_m" and par_b == "K_m" and par_c == "Y_x":
646         scenario, scenario_count = scenarios3(unc_mu_m, unc__K_m, unc__Y_x)
647     elif par_a == "mu_m" and par_b == "K_m" and par_c == "S_in":
648         scenario, scenario_count = scenarios3(unc_mu_m, unc__K_m, unc_S_in)
649     elif par_a == "mu_m" and par_b == "K_i" and par_c == "nu":
650         scenario, scenario_count = scenarios3(unc_mu_m, unc__K_i, unc___nu)
651     elif par_a == "mu_m" and par_b == "K_i" and par_c == "Y_p":
652         scenario, scenario_count = scenarios3(unc_mu_m, unc__K_i, unc__Y_p)
653     elif par_a == "mu_m" and par_b == "K_i" and par_c == "Y_x":
654         scenario, scenario_count = scenarios3(unc_mu_m, unc__K_i, unc__Y_x)
655     elif par_a == "mu_m" and par_b == "K_i" and par_c == "S_in":
656         scenario, scenario_count = scenarios3(unc_mu_m, unc__K_i, unc_S_in)
657     elif par_a == "mu_m" and par_b == "nu" and par_c == "Y_p":
658         scenario, scenario_count = scenarios3(unc_mu_m, unc___nu, unc__Y_p)
659     elif par_a == "mu_m" and par_b == "nu" and par_c == "Y_x":
660         scenario, scenario_count = scenarios3(unc_mu_m, unc___nu, unc__Y_x)
661     elif par_a == "mu_m" and par_b == "nu" and par_c == "S_in":
662         scenario, scenario_count = scenarios3(unc_mu_m, unc___nu, unc_S_in)
663     elif par_a == "mu_m" and par_b == "Y_p" and par_c == "Y_x":
664         scenario, scenario_count = scenarios3(unc_mu_m, unc__Y_p, unc__Y_x)
665     elif par_a == "mu_m" and par_b == "Y_p" and par_c == "S_in":
666         scenario, scenario_count = scenarios3(unc_mu_m, unc__Y_p, unc_S_in)
667     elif par_a == "mu_m" and par_b == "Y_x" and par_c == "S_in":
668         scenario, scenario_count = scenarios3(unc_mu_m, unc__Y_x, unc_S_in)
669     elif par_a == "K_m" and par_b == "K_i" and par_c == "nu":
670         scenario, scenario_count = scenarios3(unc__K_m, unc__K_i, unc___nu)
671     elif par_a == "K_m" and par_b == "K_i" and par_c == "Y_p":
672         scenario, scenario_count = scenarios3(unc__K_m, unc__K_i, unc__Y_p)
673     elif par_a == "K_m" and par_b == "K_i" and par_c == "Y_x":
674         scenario, scenario_count = scenarios3(unc__K_m, unc__K_i, unc__Y_x)
675     elif par_a == "K_m" and par_b == "K_i" and par_c == "S_in":
676         scenario, scenario_count = scenarios3(unc__K_m, unc__K_i, unc_S_in)
677     elif par_a == "K_m" and par_b == "nu" and par_c == "Y_p":
678         scenario, scenario_count = scenarios3(unc__K_m, unc___nu, unc__Y_p)
679     elif par_a == "K_m" and par_b == "nu" and par_c == "Y_x":
680         scenario, scenario_count = scenarios3(unc__K_m, unc___nu, unc__Y_x)
681     elif par_a == "K_m" and par_b == "nu" and par_c == "S_in":
682         scenario, scenario_count = scenarios3(unc__K_m, unc___nu, unc_S_in)
683     elif par_a == "K_m" and par_b == "Y_p" and par_c == "Y_x":
684         scenario, scenario_count = scenarios3(unc__K_m, unc__Y_p, unc__Y_x)
685     elif par_a == "K_m" and par_b == "Y_p" and par_c == "S_in":
686         scenario, scenario_count = scenarios3(unc__K_m, unc__Y_p, unc_S_in)
687     elif par_a == "K_m" and par_b == "Y_x" and par_c == "S_in":
688         scenario, scenario_count = scenarios3(unc__K_m, unc__Y_x, unc_S_in)
689     elif par_a == "K_i" and par_b == "nu" and par_c == "Y_p":
690         scenario, scenario_count = scenarios3(unc__K_i, unc___nu, unc__Y_p)
691     elif par_a == "K_i" and par_b == "nu" and par_c == "Y_x":
692         scenario, scenario_count = scenarios3(unc__K_i, unc___nu, unc__Y_x)
693     elif par_a == "K_i" and par_b == "nu" and par_c == "S_in":
694         scenario, scenario_count = scenarios3(unc__K_i, unc___nu, unc_S_in)
695     elif par_a == "K_i" and par_b == "Y_p" and par_c == "Y_x":
696         scenario, scenario_count = scenarios3(unc__K_i, unc__Y_p, unc__Y_x)
697     elif par_a == "K_i" and par_b == "Y_p" and par_c == "S_in":
698         scenario, scenario_count = scenarios3(unc__K_i, unc__Y_p, unc_S_in)
699     elif par_a == "K_i" and par_b == "Y_x" and par_c == "S_in":
700         scenario, scenario_count = scenarios3(unc__K_i, unc__Y_x, unc_S_in)
701     elif par_a == "nu" and par_b == "Y_p" and par_c == "Y_x":
702         scenario, scenario_count = scenarios3(unc___nu, unc__Y_p, unc__Y_x)
703     elif par_a == "nu" and par_b == "Y_p" and par_c == "S_in":
704         scenario, scenario_count = scenarios3(unc___nu, unc__Y_p, unc_S_in)
705     elif par_a == "nu" and par_b == "Y_x" and par_c == "S_in":
706         scenario, scenario_count = scenarios3(unc___nu, unc__Y_x, unc_S_in)
707     elif par_a == "Y_p" and par_b == "Y_x" and par_c == "S_in":
708         scenario, scenario_count = scenarios3(unc__Y_p, unc__Y_x, unc_S_in)
709     else:  # if there is given an invalid parameter combination
710         raise IndexError("The given combination doesn't exist")
711 
712     dg = 3  # orthogonal collocation with 3 points each element
713     tau_root = np.append(0, cd.collocation_points(dg, "radau"))
714     b = np.zeros((dg + 1, 1))
715     c = np.zeros((dg + 1, dg + 1))
716     d = np.zeros((dg + 1, 1))
717 
718     for i in range(dg + 1):
719         coeff = 1
720         # Construct Lagrange polynomials to get the
721         # polynomial basis at the collocation point.
722         for r in range(dg + 1):
723             if r != i:
724                 coeff = np.convolve(coeff, [1., -tau_root[r]])
725                 coeff = coeff / (tau_root[i] - tau_root[r])
726 
727         # Evaluate the polynomial at the final time to
728         # get coefficients of the continuity equation.
729         d[i] = np.polyval(coeff, 1.)
```

```
730
731         # Evaluate time derivative of the polynomial at all collocation
732         # points to obtain the coefficients of the continuity equation.
733         pder = np.polyder(coeff)
734         for r in range(dg + 1):
735             c[i][r] = np.polyval(pder, tau_root[r])
736
737         # Evaluate the integral of the polynomial to
738         # get coefficients of the quadrature function.
739         pint = np.polyint(coeff)
740         b[i] = np.polyval(pint, 1.)
741
742     # Declare matrix for the economic objective
743     Q = np.array([[0., 0., 0., 0.],
744                   [0., 0., 0., 0.],
745                   [0., 0., -1., 0.],
746                   [0., 0., 0., 0.]])
747
748     # Declare matrix for penalizing MV-changes
749     R = np.array([0.])  # same as do-mpc used!
750
751     # Declare the horizons; npr,nct,nrb
752     npr, nct, nrb = 20, 3, 1
753
754     # Finding the number of scenarios
755     levels = scenario_count
756     nsc = levels ** nrb
757
758     # Assuming scenario weights equal
759     omega = 1. / nsc
760
761     # Obtaining the defined ODE-model
762     sys = ode_model()
763
764     # Declare the economic objective function
765     J = omega * (x.T @ Q @ x + du.T @ R @ du)
766
767     # Declare cd.Function "f" for later application
768     f = cd.Function("f", [x, u, p, t, du], [sys, J])
769
770     # Declare an empty NLP, i.e., empty vertcats + bounds
771     w, w0 = cd.vertcat([]), cd.vertcat([])  # states, inputs
772     lbw, ubw = cd.vertcat([]), cd.vertcat([])  # states, inputs
773     g1, lbg1, ubg1 = cd.vertcat([]), cd.vertcat([]), cd.vertcat([])
774     g2, lbg2, ubg2 = cd.vertcat([]), cd.vertcat([]), cd.vertcat([])
775     g3, lbg3, ubg3 = cd.vertcat([]), cd.vertcat([]), cd.vertcat([])
776
777     # Declare state constraints; X_min, S_min, P_min, V_min
778     X_min, S_min, P_min, V_min = 0., -.01, 0., 0.
779     # Declare state constraints; X_max, S_max, P_max, V_max
780     X_max, S_max, P_max, V_max = 3.7, np.inf, 3.0, np.inf
781
782     # Concatenate these minimum constraints on the states
783     x_min = cd.vertcat(X_min, S_min, P_min, V_min)
784     # Concatenate these maximum constraints on the states
785     x_max = cd.vertcat(X_max, S_max, P_max, V_max)
786
787     # Declaring the constraints; u_min, u_max, du_max
788     u_min, u_max, du_max = 0., .2, .0035
789
790     # Declare x_plt, u_plt for the trajectory split
791     x_plt, u_plt = cd.horzcat([]), cd.horzcat([])
792
793     # Initialize the economic objective function
794     J = 0
795
796     for j in range(nsc):
797         if par_a == "mu_m" and par_b == "K_m" and par_c == "K_i":
798             pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
799                 scenario[j, 0], scenario[j, 1], scenario[j, 2], p0[3], p0[4], p0[5], p0[6]))
800         elif par_a == "mu_m" and par_b == "K_m" and par_c == "nu":
801             pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
802                 scenario[j, 0], scenario[j, 1], p0[2], scenario[j, 2], p0[4], p0[5], p0[6]))
803         elif par_a == "mu_m" and par_b == "K_m" and par_c == "Y_p":
804             pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
805                 scenario[j, 0], scenario[j, 1], p0[2], p0[3], scenario[j, 2], p0[5], p0[6]))
806         elif par_a == "mu_m" and par_b == "K_m" and par_c == "Y_x":
807             pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
808                 scenario[j, 0], scenario[j, 1], p0[2], p0[3], p0[4], scenario[j, 2], p0[6]))
809         elif par_a == "mu_m" and par_b == "K_m" and par_c == "S_in":
810             pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
811                 scenario[j, 0], scenario[j, 1], p0[2], p0[3], p0[4], p0[5], scenario[j, 2]))
812         elif par_a == "mu_m" and par_b == "K_i" and par_c == "nu":
813             pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
814                 scenario[j, 0], p0[1], scenario[j, 1], scenario[j, 2], p0[4], p0[5], p0[6]))
815         elif par_a == "mu_m" and par_b == "K_i" and par_c == "Y_p":
816             pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
817                 scenario[j, 0], p0[1], scenario[j, 1], p0[3], scenario[j, 2], p0[5], p0[6]))
818         elif par_a == "mu_m" and par_b == "K_i" and par_c == "Y_x":
819             pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
820                 scenario[j, 0], p0[1], scenario[j, 1], p0[3], p0[4], scenario[j, 2], p0[6]))
821         elif par_a == "mu_m" and par_b == "K_i" and par_c == "S_in":
822             pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
823                 scenario[j, 0], p0[1], scenario[j, 1], p0[3], p0[4], p0[5], scenario[j, 2]))
824         elif par_a == "mu_m" and par_b == "nu" and par_c == "Y_p":
```

```python
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                scenario[j, 0], p0[1], p0[2], scenario[j, 1], scenario[j, 2], p0[5], p0[6]))
        elif par_a == "mu_m" and par_b == "nu" and par_c == "Y_x":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                scenario[j, 0], p0[1], p0[2], scenario[j, 1], p0[4], scenario[j, 2], p0[6]))
        elif par_a == "mu_m" and par_b == "nu" and par_c == "S_in":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                scenario[j, 0], p0[1], p0[2], scenario[j, 1], p0[4], p0[5], scenario[j, 2]))
        elif par_a == "mu_m" and par_b == "Y_p" and par_c == "Y_x":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                scenario[j, 0], p0[1], p0[2], p0[3], scenario[j, 1], scenario[j, 2], p0[6]))
        elif par_a == "mu_m" and par_b == "Y_p" and par_c == "S_in":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                scenario[j, 0], p0[1], p0[2], p0[3], scenario[j, 1], p0[5], scenario[j, 2]))
        elif par_a == "mu_m" and par_b == "Y_x" and par_c == "S_in":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                scenario[j, 0], p0[1], p0[2], p0[3], p0[4], scenario[j, 1], scenario[j, 2]))
        elif par_a == "K_m" and par_b == "K_i" and par_c == "nu":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                p0[0], scenario[j, 0], scenario[j, 1], scenario[j, 2], p0[4], p0[5], p0[6]))
        elif par_a == "K_m" and par_b == "K_i" and par_c == "Y_p":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                p0[0], scenario[j, 0], scenario[j, 1], p0[3], scenario[j, 2], p0[5], p0[6]))
        elif par_a == "K_m" and par_b == "K_i" and par_c == "Y_x":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                p0[0], scenario[j, 0], scenario[j, 1], p0[3], p0[4], scenario[j, 2], p0[6]))
        elif par_a == "K_m" and par_b == "K_i" and par_c == "S_in":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                p0[0], scenario[j, 0], scenario[j, 1], p0[3], p0[4], p0[5], scenario[j, 2]))
        elif par_a == "K_m" and par_b == "nu" and par_c == "Y_p":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                p0[0], scenario[j, 0], p0[2], scenario[j, 1], scenario[j, 2], p0[5], p0[6]))
        elif par_a == "K_m" and par_b == "nu" and par_c == "Y_x":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                p0[0], scenario[j, 0], p0[2], scenario[j, 1], p0[4], scenario[j, 2], p0[6]))
        elif par_a == "K_m" and par_b == "nu" and par_c == "S_in":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                p0[0], scenario[j, 0], p0[2], scenario[j, 1], p0[4], p0[5], scenario[j, 2]))
        elif par_a == "K_m" and par_b == "Y_p" and par_c == "Y_x":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                p0[0], scenario[j, 0], p0[2], p0[3], scenario[j, 1], scenario[j, 2], p0[6]))
        elif par_a == "K_m" and par_b == "Y_p" and par_c == "S_in":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                p0[0], scenario[j, 0], p0[2], p0[3], scenario[j, 1], p0[5], scenario[j, 2]))
        elif par_a == "K_m" and par_b == "Y_x" and par_c == "S_in":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                p0[0], scenario[j, 0], p0[2], p0[3], p0[4], scenario[j, 1], scenario[j, 2]))
        elif par_a == "K_i" and par_b == "nu" and par_c == "Y_p":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                p0[0], p0[1], scenario[j, 0], scenario[j, 1], scenario[j, 2], p0[5], p0[6]))
        elif par_a == "K_i" and par_b == "nu" and par_c == "Y_x":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                p0[0], p0[1], scenario[j, 0], scenario[j, 1], p0[4], scenario[j, 2], p0[6]))
        elif par_a == "K_i" and par_b == "nu" and par_c == "S_in":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                p0[0], p0[1], scenario[j, 0], scenario[j, 1], p0[4], p0[5], scenario[j, 2]))
        elif par_a == "K_i" and par_b == "Y_p" and par_c == "Y_x":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                p0[0], p0[1], scenario[j, 0], p0[3], scenario[j, 1], scenario[j, 2], p0[6]))
        elif par_a == "K_i" and par_b == "Y_p" and par_c == "S_in":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                p0[0], p0[1], scenario[j, 0], p0[3], scenario[j, 1], p0[5], scenario[j, 2]))
        elif par_a == "K_i" and par_b == "Y_x" and par_c == "S_in":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                p0[0], p0[1], scenario[j, 0], p0[3], p0[4], scenario[j, 1], scenario[j, 2]))
        elif par_a == "nu" and par_b == "Y_p" and par_c == "Y_x":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                p0[0], p0[1], p0[2], scenario[j, 0], scenario[j, 1], scenario[j, 2], p0[6]))
        elif par_a == "nu" and par_b == "Y_p" and par_c == "S_in":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                p0[0], p0[1], p0[2], scenario[j, 0], scenario[j, 1], p0[5], scenario[j, 2]))
        elif par_a == "nu" and par_b == "Y_x" and par_c == "S_in":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                p0[0], p0[1], p0[2], scenario[j, 0], p0[4], scenario[j, 1], scenario[j, 2]))
        elif par_a == "Y_p" and par_b == "Y_x" and par_c == "S_in":
            pj = cd.vertcat(cd.horzcat(  # assuming; 4 certain params and 3 uncertain params
                p0[0], p0[1], p0[2], p0[3], scenario[j, 0], scenario[j, 1], scenario[j, 2]))
        else:  # if there is given an invalid parameter combination
            raise IndexError("The given combination doesn't exist")

        for k in range(npr):
            if k == 0:  # at the initial point
                # 'Lift' the initial conditions
                xkj = cd.MX.sym("x_" + str(k) + "_" + str(j), 4)
                w = cd.vertcat(w, xkj)
                w0 = cd.vertcat(w0, x0)
                lbw = cd.vertcat(lbw, x0)
                ubw = cd.vertcat(ubw, x0)
                x_plt = cd.horzcat(x_plt, xkj)

            if k == 0 and j == 0:
                # New NLP variable for control
                u00 = cd.MX.sym("u_" + str(k) + "_" + str(j), 1)
                w = cd.vertcat(w, u00)
                w0 = cd.vertcat(w0, u0)
```

```
920                 lbw = cd.vertcat(lbw, u_min)
921                 ubw = cd.vertcat(ubw, u_max)
922                 u_plt = cd.horzcat(u_plt, u00)
923             else:
924                 # New NLP variable for control
925                 ukj = cd.MX.sym("u_" + str(k) + "_" + str(j), 1)
926                 w = cd.vertcat(w, ukj)
927                 w0 = cd.vertcat(w0, u0)
928                 lbw = cd.vertcat(lbw, u_min)
929                 ubw = cd.vertcat(ubw, u_max)
930                 u_plt = cd.horzcat(u_plt, ukj)
931 
932             # If-sentence for finding dukj
933             if k <= (nct - 1):
934                 if k == 0 and j == 0:
935                     dukj = u00 - u0
936                 elif k == 0 and j != 0:
937                     dukj = ukj - u0
938                 else:
939                     dukj = ukj - ukj0
940                 g2 = cd.vertcat(g2, dukj)
941                 lbg2 = cd.vertcat(lbg2, -du_max)
942                 ubg2 = cd.vertcat(ubg2, du_max)
943             else:
944                 dukj = ukj - ukj0
945                 g2 = cd.vertcat(g2, dukj)
946                 lbg2 = cd.vertcat(lbg2, 0.)
947                 ubg2 = cd.vertcat(ubg2, 0.)
948 
949             # If-sentence for finding ukj0
950             if k != (npr - 1):
951                 if k == 0 and j == 0:
952                     ukj0 = u00
953                 else:
954                     ukj0 = ukj
955             else:
956                 ukj0 = ukj0
957 
958             # States at the collocation points
959             xkji = [[] * i for i in range(dg)]
960             for i in range(dg):
961                 xkji[i] = cd.MX.sym("x_" + str(k) + "_" + str(j) + "_" + str(i), 4)
962                 w = cd.vertcat(w, xkji[i])
963                 w0 = cd.vertcat(w0, x0)
964                 lbw = cd.vertcat(lbw, x_min)
965                 ubw = cd.vertcat(ubw, x_max)
966 
967             # Loop over the collocation points
968             xkj_end = d[0] * xkj
969             for i in range(dg):
970                 # State derivative collocation points
971                 xp = c[0, i + 1] * xkj
972                 for r in range(dg):
973                     xp += c[r + 1, i + 1] * xkji[r]
974 
975                 # Append collocation equations
976                 dt = tk[1] - tk[0]  # current time-step
977                 if k == 0 and j == 0:
978                     fi, qi = f(xkji[i], u00, pj, dt, dukj)
979                 else:
980                     fi, qi = f(xkji[i], ukj, pj, dt, dukj)
981                 g1 = cd.vertcat(g1, dt * fi - xp)
982                 lbg1 = cd.vertcat(lbg1, 0., 0., 0., 0.)
983                 ubg1 = cd.vertcat(ubg1, 0., 0., 0., 0.)
984 
985                 # Add contribution to the end state
986                 xkj_end += d[i + 1] * xkji[i]
987 
988                 # Add contribution to quad function
989                 J += b[i + 1] * qi * dt
990 
991             # New NLP variable for states at the end of interval
992             xkj = cd.MX.sym("x_" + str(k + 1) + "_" + str(j), 4)
993             w = cd.vertcat(w, xkj)
994             w0 = cd.vertcat(w0, x0)
995             lbw = cd.vertcat(lbw, x_min)
996             ubw = cd.vertcat(ubw, x_max)
997             x_plt = cd.horzcat(x_plt, xkj)
998 
999             # Add the equality constraints
1000            g1 = cd.vertcat(g1, xkj_end - xkj)
1001            lbg1 = cd.vertcat(lbg1, 0., 0., 0., 0.)
1002            ubg1 = cd.vertcat(ubg1, 0., 0., 0., 0.)
1003
1004            # Add the non-anticipativity constraints
1005            if k == 0 and j == 0:  # only 1st scenario
1006                g3 = cd.vertcat(g3, u00 - u00)
1007                lbg3 = cd.vertcat(lbg3, 0.)
1008                ubg3 = cd.vertcat(ubg3, 0.)
1009            elif k == 0 and j != 0:  # other scenarios
1010                g3 = cd.vertcat(g3, ukj - u00)
1011                lbg3 = cd.vertcat(lbg3, 0.)
1012                ubg3 = cd.vertcat(ubg3, 0.)
1013
1014    # Obtaining the trajectories from 'w'
```

```
1015    trajects = cd.Function("trajectory", [w], [x_plt, u_plt], ["w"], ["x", "u"])
1016
1017    # Formalize this into the NLP problem
1018    prob = {"x": cd.vertcat(w), "g": cd.vertcat(g1, g2, g3), "f": J, "p": p_var}
1019
1020    # We may create the option dictionary
1021    opts = {"ipopt.max_iter": 200, "ipopt.print_level": 0, "print_time": 0,
1022            "ipopt.tol": 1.0e-6, "ipopt.acceptable_tol": 1.0e-4}
1023
1024    # Assign solver - 'IPOPT' in our case
1025    solver = cd.nlpsol("solver", "ipopt", prob, opts)
1026
1027    # Converting from cd.MX into ndarray
1028    w0_conv = cd.Function("w0_conv", [x0, tk, u0, p0], [w0])
1029    lbw_conv = cd.Function("lbw_conv", [x0, tk, u0, p0], [lbw])
1030    ubw_conv = cd.Function("ubw_conv", [x0, tk, u0, p0], [ubw])
1031
1032    # Converting from cd.MX into ndarray
1033    lbg1, ubg1 = np.array(lbg1).flatten(), np.array(ubg1).flatten()
1034    lbg2, ubg2 = np.array(lbg2).flatten(), np.array(ubg2).flatten()
1035    lbg3, ubg3 = np.array(lbg3).flatten(), np.array(ubg3).flatten()
1036    lbg = np.concatenate((lbg1, lbg2, lbg3))  # concatenate lower constraints
1037    ubg = np.concatenate((ubg1, ubg2, ubg3))  # concatenate upper constraints
1038
1039    # Return trajects+solver together with the symbolic containers
1040    return trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg
1041
1042
1043 # Defining optimzr_nmpc()
1044 def optimzr_nmpc(trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg, x0, tk, u0, p0, w_opt):
1045    # Converting from cd.MX into ndarray
1046    w0, lbw, ubw = w0_conv(x0, tk, u0, p0), lbw_conv(x0, tk, u0, p0), ubw_conv(x0, tk, u0, p0)
1047
1048    # Converting from cd.MX into ndarray
1049    if tk[0] == 0. and tk[1] == 1.:
1050        w0 = np.array(w0).flatten()
1051        lbw = np.array(lbw).flatten()
1052        ubw = np.array(ubw).flatten()
1053    else:
1054        w0 = np.array([w_opt]).flatten()
1055        lbw = np.array(lbw).flatten()
1056        ubw = np.array(ubw).flatten()
1057        w0[:x0.shape[0]] = x0  # update
1058        lbw[:x0.shape[0]] = x0  # update
1059        ubw[:x0.shape[0]] = x0  # update
1060
1061    # Must include the extra arguments
1062    p_var = np.hstack((x0, tk, u0, p0))
1063    # Solve using the defined initial guesses and bounds for solver()
1064    solv = solver(x0=w0, lbx=lbw, ubx=ubw, lbg=lbg, ubg=ubg, p=p_var)
1065    x_opt, u_opt = trajects(solv["x"])  # obtaining x_opt, u_opt
1066    w_opt = solv["x"]  # obtain w_opt required for optimizer()
1067    x_opt = x_opt.full()  # converting from cd.MX into ndarray
1068    u_opt = u_opt.full()  # converting from cd.MX into ndarray
1069    return u_opt, w_opt  # returning control inputs and w_opts
1070
1071
1072 # Defining optimzr_rmpc2()
1073 def optimzr_rmpc2(trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg, x0, tk, u0, p0, w_opt):
1074    # Converting from cd.MX into ndarray
1075    w0, lbw, ubw = w0_conv(x0, tk, u0, p0), lbw_conv(x0, tk, u0, p0), ubw_conv(x0, tk, u0, p0)
1076
1077    # Converting from cd.MX into ndarray
1078    if tk[0] == 0. and tk[1] == 1.:
1079        w0 = np.array(w0).flatten()
1080        lbw = np.array(lbw).flatten()
1081        ubw = np.array(ubw).flatten()
1082    else:
1083        w0 = np.array([w_opt]).flatten()
1084        lbw = np.array(lbw).flatten()
1085        ubw = np.array(ubw).flatten()
1086        for i in range(9):
1087            w0[344 * i:344 * i + 4] = x0
1088            lbw[344 * i:344 * i + 4] = x0
1089            ubw[344 * i:344 * i + 4] = x0
1090
1091    # Must include the extra arguments
1092    p_var = np.hstack((x0, tk, u0, p0))
1093    # Solve using the defined initial guesses and bounds for solver()
1094    solv = solver(x0=w0, lbx=lbw, ubx=ubw, lbg=lbg, ubg=ubg, p=p_var)
1095    x_opt, u_opt = trajects(solv["x"])  # obtaining x_opt, u_opt
1096    w_opt = solv["x"]  # obtain w_opt required for optimizer()
1097    x_opt = x_opt.full()  # converting from cd.MX into ndarray
1098    u_opt = u_opt.full()  # converting from cd.MX into ndarray
1099
1100    u01_opts = u_opt[0][0:-1:20]  # 1st controls for every scenario
1101    np.all(np.isclose(u01_opts, u01_opts[4]))  # asserting equality
1102    u02_opts = u_opt[0][1:-1:20]  # 2nd controls for every scenario
1103    u03_opts = u_opt[0][2:-1:20]  # 3rd controls for every scenario
1104    u04_opts = u_opt[0][3:-1:20]  # 4th controls for every scenario
1105    u05_opts = u_opt[0][4:-1:20]  # 5th controls for every scenario
1106    u06_opts = u_opt[0][5:-1:20]  # 6th controls for every scenario
1107    u07_opts = u_opt[0][6:-1:20]  # 7th controls for every scenario
1108    u08_opts = u_opt[0][7:-1:20]  # 8th controls for every scenario
1109    u09_opts = u_opt[0][8:-1:20]  # 9th controls for every scenario
```

```
    u10_opts = u_opt[0][9:-1:20]  # 10th controls for every scenario
    u11_opts = u_opt[0][10:-1:20]  # 11th controls for every scenario
    u12_opts = u_opt[0][11:-1:20]  # 12th controls for every scenario
    u13_opts = u_opt[0][12:-1:20]  # 13th controls for every scenario
    u14_opts = u_opt[0][13:-1:20]  # 14th controls for every scenario
    u15_opts = u_opt[0][14:-1:20]  # 15th controls for every scenario
    u16_opts = u_opt[0][15:-1:20]  # 16th controls for every scenario
    u17_opts = u_opt[0][16:-1:20]  # 17th controls for every scenario
    u18_opts = u_opt[0][17:-1:20]  # 18th controls for every scenario
    u19_opts = u_opt[0][18:-1:20]  # 19th controls for every scenario
    u20_opts = u_opt[0][19:-1:20]  # 20th controls for every scenario

    u_opts = np.array([u01_opts[4], u02_opts[4], u03_opts[4], u04_opts[4],
                       u05_opts[4], u06_opts[4], u07_opts[4], u08_opts[4],
                       u09_opts[4], u10_opts[4], u11_opts[4], u12_opts[4],
                       u13_opts[4], u14_opts[4], u15_opts[4], u16_opts[4],
                       u17_opts[4], u18_opts[4], u19_opts[4], u20_opts[4]])
    return u_opts, w_opt  # returning the nominal control inputs and w_opts


# Defining optimzr_rmpc3()
def optimzr_rmpc3(trajects, solver, w0_conv, lbw_conv, ubw_conv, lbg, ubg, x0, tk, u0, p0, w_opt):
    # Converting from cd.MX into ndarray
    w0, lbw, ubw = w0_conv(x0, tk, u0, p0), lbw_conv(x0, tk, u0, p0), ubw_conv(x0, tk, u0, p0)

    # Converting from cd.MX into ndarray
    if tk[0] == 0. and tk[1] == 1.:
        w0 = np.array(w0).flatten()
        lbw = np.array(lbw).flatten()
        ubw = np.array(ubw).flatten()
    else:
        w0 = np.array([w_opt]).flatten()
        lbw = np.array(lbw).flatten()
        ubw = np.array(ubw).flatten()
        for i in range(27):
            w0[344 * i:344 * i + 4] = x0
            lbw[344 * i:344 * i + 4] = x0
            ubw[344 * i:344 * i + 4] = x0

    # Must include the extra arguments
    p_var = np.hstack((x0, tk, u0, p0))
    # Solve using the defined initial guesses and bounds for solver()
    solv = solver(x0=w0, lbx=lbw, ubx=ubw, lbg=lbg, ubg=ubg, p=p_var)
    x_opt, u_opt = trajects(solv["x"])  # obtaining x_opt, u_opt
    w_opt = solv["x"]  # obtain w_opt required for optimizer()
    x_opt = x_opt.full()  # converting from cd.MX into ndarray
    u_opt = u_opt.full()  # converting from cd.MX into ndarray

    u01_opts = u_opt[0][0:-1:20]  # 1st controls for every scenario
    np.all(np.isclose(u01_opts, u01_opts[13]))  # asserting equality
    u02_opts = u_opt[0][1:-1:20]  # 2nd controls for every scenario
    u03_opts = u_opt[0][2:-1:20]  # 3rd controls for every scenario
    u04_opts = u_opt[0][3:-1:20]  # 4th controls for every scenario
    u05_opts = u_opt[0][4:-1:20]  # 5th controls for every scenario
    u06_opts = u_opt[0][5:-1:20]  # 6th controls for every scenario
    u07_opts = u_opt[0][6:-1:20]  # 7th controls for every scenario
    u08_opts = u_opt[0][7:-1:20]  # 8th controls for every scenario
    u09_opts = u_opt[0][8:-1:20]  # 9th controls for every scenario
    u10_opts = u_opt[0][9:-1:20]  # 10th controls for every scenario
    u11_opts = u_opt[0][10:-1:20]  # 11th controls for every scenario
    u12_opts = u_opt[0][11:-1:20]  # 12th controls for every scenario
    u13_opts = u_opt[0][12:-1:20]  # 13th controls for every scenario
    u14_opts = u_opt[0][13:-1:20]  # 14th controls for every scenario
    u15_opts = u_opt[0][14:-1:20]  # 15th controls for every scenario
    u16_opts = u_opt[0][15:-1:20]  # 16th controls for every scenario
    u17_opts = u_opt[0][16:-1:20]  # 17th controls for every scenario
    u18_opts = u_opt[0][17:-1:20]  # 18th controls for every scenario
    u19_opts = u_opt[0][18:-1:20]  # 19th controls for every scenario
    u20_opts = u_opt[0][19:-1:20]  # 20th controls for every scenario

    u_opts = np.array([u01_opts[13], u02_opts[13], u03_opts[13], u04_opts[13],
                       u05_opts[13], u06_opts[13], u07_opts[13], u08_opts[13],
                       u09_opts[13], u10_opts[13], u11_opts[13], u12_opts[13],
                       u13_opts[13], u14_opts[13], u15_opts[13], u16_opts[13],
                       u17_opts[13], u18_opts[13], u19_opts[13], u20_opts[13]])
    return u_opts, w_opt  # returning the nominal control inputs and w_opts
```

Listing 6.2: optimiz.py - The optimization for all the MPCs

The process plant for all the MPCs

```
import numpy as np
import casadi as cd

# Declaring 1st state symbolic
X = cd.MX.sym("X", 1)
# Declaring 2nd state symbolic
S = cd.MX.sym("S", 1)
# Declaring 3rd state symbolic
P = cd.MX.sym("P", 1)
# Declaring 4th state symbolic
V = cd.MX.sym("V", 1)
```

```python

# Declaring 1st parameter symbolic
mu_m = cd.MX.sym("mu_m", 1)
# Declaring 2nd parameter symbolic
K_m = cd.MX.sym("K_m", 1)
# Declaring 3rd parameter symbolic
K_i = cd.MX.sym("K_i", 1)
# Declaring 4th parameter symbolic
nu = cd.MX.sym("nu", 1)
# Declaring 5th parameter symbolic
Y_p = cd.MX.sym("Y_p", 1)
# Declaring 6th parameter symbolic
Y_x = cd.MX.sym("Y_x", 1)
# Declaring 7th parameter symbolic
S_in = cd.MX.sym("S_in", 1)

# Concatenate symbolic states
x = cd.vertcat(X, S, P, V)
# Concatenate symbolic parameters
p = cd.vertcat(mu_m, K_m, K_i, nu, Y_p, Y_x, S_in)
# Declaring the input symbolic
u = cd.MX.sym("u", 1)
# Declaring MV-change symbolic
du = cd.MX.sym("du", 1)
# Declaring time-span symbolic
t = cd.MX.sym("t", 1)


# Defining ode_model()
def ode_model():
    # Declaring the kinetic model
    mu = (mu_m * S) / (K_m + S + ((S ** 2) / K_i))
    # Declaring the biomass equation
    dXdt = mu * X - (u / V) * X
    # Declaring the substrate equation
    dSdt = -(mu * X) / Y_x - (nu * X) / Y_p + (u / V) * (S_in - S)
    # Declaring the product equation
    dPdt = nu * X - (u / V) * P
    # Declaring the volume equation
    dVdt = u
    # Returning these ODEs together
    return cd.vertcat(dXdt, dSdt, dPdt, dVdt)


# Defining integrate()
def integrate():
    sys = ode_model()
    p_var = cd.vertcat(u, p, t)
    # Declaring the ODE-dictionary
    ode = {"x": x, "p": p_var, "ode": sys * t}
    # Declaring options dictionary
    opts = {"max_num_steps": 200,
            "abstol": 1.0e-6, "reltol": 1.0e-4}
    # Declaring the ODE-integrator
    f = cd.integrator("F", "cvodes", ode, opts)
    # Returning the ODE-integrator
    return f


# Defining simulator()
def simulator(f, x0, tk, u0, p0):
    x_dim = x0.shape[0]
    t_dim = tk.shape[0]
    u_dim = u0.shape[0]
    p_dim = p0.shape[0]
    assert u_dim == t_dim - 1

    x0 = cd.vertcat(x0)
    u0 = cd.vertcat(u0)
    p0 = cd.vertcat(p0)
    dt = np.zeros(t_dim - 1)
    for i in range(t_dim - 1):
        dt[i] = tk[i + 1] - tk[i]
    dt = cd.vertcat(dt)

    xfs = np.zeros((x_dim, t_dim - 1))
    for i in range(t_dim - 1):
        if i == 0:
            p_var = cd.vertcat(u0[i], p0, dt[i])
            f_end = f(x0=x0, p=p_var)
            xf = np.array(f_end["xf"]).flatten()
        else:
            p_var = cd.vertcat(u0[i], p0, dt[i])
            f_end = f(x0=xfs[:, i - 1], p=p_var)
            xf = np.array(f_end["xf"]).flatten()
        xfs[:, i] = xf
    return xfs
```

Listing 6.3: process.py - The process plant for all the MPCs

The sensitivity analysis for all the MPCs

```python
import numpy as np
import scipy.stats as sc
from process import simulator


def unifrm_sample(p0):
    p_dim = p0.shape[0]
    p_min = (1 - 0.15) * p0.copy()
    p_max = (1 + 0.15) * p0.copy()
    # Obtain sample from numpy's uniform distribution
    p_sample = np.random.uniform(p_min, p_max, p_dim)
    return p_sample


def unifrm_distrb(p0):
    p_dim = p0.shape[0]
    p_min = (1 - 0.15) * p0.copy()
    p_max = (1 + 0.15) * p0.copy()
    # Obtaining 'frozen random variable objects', from scipy's uniform distribution function
    p_distrb = [sc.uniform(loc=p_min[i], scale=(p_max[i] - p_min[i])) for i in range(p_dim)]
    return p_distrb


def sobols_method(f, x0, tk, u0, p0):
    x_dim = x0.shape[0]
    t_dim = tk.shape[0]
    u_dim = u0.shape[0]
    p_dim = p0.shape[0]
    p_dist = unifrm_distrb(p0)
    assert u_dim == t_dim - 1

    N = 2 ** 10  # base samples for Saltelli's modification
    p_samples = np.zeros((2 * p_dim, N))  # Saltelli samples
    samples = sc.qmc.LatinHypercube(d=(2 * p_dim)).random(n=N)
    samples = np.moveaxis(samples.reshape((2 * p_dim, N)), 1, 1)

    for j in range(2 * p_dim):
        if j < p_dim:
            # Using '.ppf' - percent point function (cdf inversed)
            p_samples[j, :] = (p_dist[j].ppf(samples[j, :]))
        else:
            # Using '.ppf' - percent point function (cdf inversed)
            p_samples[j, :] = p_dist[j - p_dim].ppf(samples[j, :])

    A = p_samples[:p_dim, :]
    B = p_samples[p_dim:, :]

    yA = np.zeros((t_dim - 1, N))
    yB = np.zeros((t_dim - 1, N))
    yC = np.zeros((t_dim - 1, N))

    s1s = np.zeros((p_dim, t_dim - 1))
    sts = np.zeros((p_dim, t_dim - 1))

    for j in range(p_dim):
        C = B.copy()
        C[j, :] = A[j, :]
        for i in range(N):
            if j == 0:
                yA[:, i] = simulator(f, x0, tk, u0, A[:, i])[0]
                yB[:, i] = simulator(f, x0, tk, u0, B[:, i])[0]
            yC[:, i] = simulator(f, x0, tk, u0, C[:, i])[0]

        for i in range(t_dim - 1):
            yA_mean, yA_std = yA[i, :].mean(), yA[i, :].std()
            yB_mean, yB_std = yB[i, :].mean(), yB[i, :].std()
            yC_mean, yC_std = yC[i, :].mean(), yC[i, :].std()

            yA_norm = (yA[i, :] - yA_mean) / yA_std
            yB_norm = (yB[i, :] - yB_mean) / yB_std
            yC_norm = (yC[i, :] - yC_mean) / yC_std

            f0 = (((1 / N) * np.sum(yA_norm)) ** 2)
            s1 = (((1 / N) * (yA_norm @ yC_norm)) - f0) / (
                    ((1 / N) * (yA_norm @ yA_norm)) - f0)
            st = 1 - (((1 / N) * (yB_norm @ yC_norm)) - f0) / (
                    ((1 / N) * (yA_norm @ yA_norm)) - f0)
            s1s[j, i], sts[j, i] = s1, st
    return s1s, sts


def morris_method(f, x0, tk, u0, p0):
    x_dim = x0.shape[0]
    t_dim = tk.shape[0]
    u_dim = u0.shape[0]
    p_dim = p0.shape[0]
    assert u_dim == t_dim - 1

    p = 4  # levels in the grid space 'omega'
    N = 2 ** 10  # base samples Morris method
    delta = p / (2 * (p - 1))  # +-increments
    omega = np.linspace(0, 1 - delta, p // 2)
```

```
93      samples = np.zeros((p_dim + 1, p_dim, N))
94      p_samples = np.zeros((p_dim + 1, p_dim, N))
95
96      for i in range(N):
97          # Finding starting points 'x_star' for the trajectories
98          x_star = np.random.choice(omega, size=p_dim)
99          # Obtaining strictly lower triangular matrix 'B' of 1's
100         B = np.tril(np.ones((p_dim + 1, p_dim)), -1)
101         # Obtaining matrix 'J' of 1's for the copying of x_star
102         J = np.ones((p_dim + 1, p_dim))
103         # Obtaining matrix 'D_star', factor moves with +-delta?
104         D_star = np.diag(np.random.choice([-1, 1], size=p_dim))
105         # Obtaining random permutation matrix 'P_star' of 0/1's
106         P_star = np.random.permutation(np.eye(p_dim, p_dim))
107         # Finally obtaining randomized sampling matrix 'B_star'
108         samples[:, :, i] = (((J[0, :] * x_star) + (delta / 2.) * ((2 * B - J) @ D_star + J)) @ P_star)
109         # Scaling acquired 'samples' using uniform distribution
110         p_samples[:, :, i] = samples[:, :, i] * (p0 * (1. + .15) - p0 * (1. - .15)) + p0 * (1. - .15)
111
112     y = np.zeros((p_dim + 1, t_dim - 1, N))
113     y_norm = np.zeros((p_dim + 1, t_dim - 1, N))
114
115     ee = np.zeros((p_dim, t_dim - 1, N))
116     ee_abs = np.zeros((p_dim, t_dim - 1))
117     ee_std = np.zeros((p_dim, t_dim - 1))
118
119     for j in range(p_dim + 1):
120         for i in range(N):
121             # Obtain biomass values using 'p_samples' for the integrator
122             y[j, :, i] = simulator(f, x0, tk, u0, p_samples[j, :, i])[0]
123         for i in range(t_dim - 1):
124             y_mean, y_std = y[j, i, :].mean(), y[j, i, :].std()
125             y_norm[j, i, :] = ((y[j, i, :] - y_mean) / y_std)
126
127     for j in range(p_dim):
128         for i in range(p_dim):
129             for k in range(N):
130                 if p_samples[i + 1, j, k] > p_samples[i, j, k]:
131                     ee[j, :, k] = ((y_norm[i + 1, :, k] - y_norm[i, :, k]) / delta)
132                 elif p_samples[i + 1, j, k] < p_samples[i, j, k]:
133                     ee[j, :, k] = ((y_norm[i, :, k] - y_norm[i + 1, :, k]) / delta)
134         for i in range(t_dim - 1):
135             ee_abs[j, i], ee_std[j, i] = np.mean(np.abs(ee[j, i])), np.std(ee[j, i])
136     return ee_abs, ee_std
137
138
139 def modify_method(f, x0, tk, u0, p0):
140     x_dim = x0.shape[0]
141     t_dim = tk.shape[0]
142     u_dim = u0.shape[0]
143     p_dim = p0.shape[0]
144     assert u_dim == t_dim - 1
145
146     N = 2 ** 10  # base samples for the modified method
147     delta = np.zeros((p_dim, N))  # percent-wise change
148     samples1 = np.zeros((p_dim, p_dim, N))  # sample-set
149     samples2 = np.zeros((p_dim, p_dim, N))  # sample-set
150     samples = sc.qmc.LatinHypercube(d=p_dim).random(n=N)
151     samples = np.moveaxis(samples.reshape(p_dim, N), 1, 1)
152     for j in range(p_dim):
153         for i in range(N):
154             params1 = samples[:, i].copy()
155             params2 = samples[:, i].copy()
156             delta[j, i] = .05 * params1[j]
157
158             # Increase/decrease with delta:
159             if np.random.rand(1) > .5:
160                 params2[j] = params1[j] + delta[j, i]
161             else:
162                 params2[j] = params1[j] - delta[j, i]
163
164             # Scaling parameters uniformly:
165             samples1[j, :, i] = params1 * (p0 * (1 + .15) - p0 * (1 - .15)) + (p0 * (1 - .15))
166             samples2[j, :, i] = params2 * (p0 * (1 + .15) - p0 * (1 - .15)) + (p0 * (1 - .15))
167
168     y1 = np.zeros((p_dim, t_dim - 1, N))
169     y2 = np.zeros((p_dim, t_dim - 1, N))
170     y1_norm = np.zeros((p_dim, t_dim - 1, N))
171     y2_norm = np.zeros((p_dim, t_dim - 1, N))
172
173     pe = np.zeros((p_dim, t_dim - 1, N))
174     pe_abs = np.zeros((p_dim, t_dim - 1))
175     pe_std = np.zeros((p_dim, t_dim - 1))
176
177     for j in range(p_dim):
178         for i in range(N):
179             y1[j, :, i] = simulator(f, x0, tk, u0, samples1[j, :, i])[0]
180             y2[j, :, i] = simulator(f, x0, tk, u0, samples2[j, :, i])[0]
181         for i in range(t_dim - 1):
182             y1_mean, y1_std = y1[j, i, :].mean(), y1[j, i, :].std()
183             y2_mean, y2_std = y2[j, i, :].mean(), y2[j, i, :].std()
184
185             y1_norm[j, i, :] = (y1[j, i, :] - y1_mean) / y1_std
186             y2_norm[j, i, :] = (y2[j, i, :] - y2_mean) / y2_std
187
```

```
188     for j in range(p_dim):
189         for i in range(t_dim - 1):
190             pe[j, i, :] = ((y2_norm[j, i, :] - y1_norm[j, i, :]) / (np.sqrt(delta[j, :])))
191             pe_abs[j, i], pe_std[j, i] = np.mean(np.abs(pe[j, i, :])), np.std(pe[j, i, :])
192     return pe_abs, pe_std
193
194
195 def switch_rule2(switch, st, ee, pe):
196     p_dim, t_dim = st.shape[0], st.shape[1]
197     st_sums = np.zeros((p_dim, t_dim))
198     ee_sums = np.zeros((p_dim, t_dim))
199     pe_sums = np.zeros((p_dim, t_dim))
200
201     if np.any(st != 0):
202         st_p0_sum = 0.
203         st_p1_sum = 0.
204         st_p2_sum = 0.
205         st_p3_sum = 0.
206         st_p4_sum = 0.
207         st_p5_sum = 0.
208         st_p6_sum = 0.
209
210         total_weight = 0.
211         for i in range(t_dim):
212             i_weight = .65 ** i
213             total_weight += i_weight
214             st_p0_sum += st[0, i] * i_weight
215             st_p1_sum += st[1, i] * i_weight
216             st_p2_sum += st[2, i] * i_weight
217             st_p3_sum += st[3, i] * i_weight
218             st_p4_sum += st[4, i] * i_weight
219             st_p5_sum += st[5, i] * i_weight
220             st_p6_sum += st[6, i] * i_weight
221
222         st_p0_sum = st_p0_sum / total_weight
223         st_p1_sum = st_p1_sum / total_weight
224         st_p2_sum = st_p2_sum / total_weight
225         st_p3_sum = st_p3_sum / total_weight
226         st_p4_sum = st_p4_sum / total_weight
227         st_p5_sum = st_p5_sum / total_weight
228         st_p6_sum = st_p6_sum / total_weight
229
230         st_sums = np.array([st_p0_sum,
231                             st_p1_sum,
232                             st_p2_sum,
233                             st_p3_sum,
234                             st_p4_sum,
235                             st_p5_sum,
236                             st_p6_sum])
237
238     if np.any(ee != 0):
239         ee_p0_sum = 0.
240         ee_p1_sum = 0.
241         ee_p2_sum = 0.
242         ee_p3_sum = 0.
243         ee_p4_sum = 0.
244         ee_p5_sum = 0.
245         ee_p6_sum = 0.
246
247         total_weight = 0.
248         for i in range(t_dim):
249             i_weight = .65 ** i
250             total_weight += i_weight
251             ee_p0_sum += ee[0, i] * i_weight
252             ee_p1_sum += ee[1, i] * i_weight
253             ee_p2_sum += ee[2, i] * i_weight
254             ee_p3_sum += ee[3, i] * i_weight
255             ee_p4_sum += ee[4, i] * i_weight
256             ee_p5_sum += ee[5, i] * i_weight
257             ee_p6_sum += ee[6, i] * i_weight
258
259         ee_p0_sum = ee_p0_sum / total_weight
260         ee_p1_sum = ee_p1_sum / total_weight
261         ee_p2_sum = ee_p2_sum / total_weight
262         ee_p3_sum = ee_p3_sum / total_weight
263         ee_p4_sum = ee_p4_sum / total_weight
264         ee_p5_sum = ee_p5_sum / total_weight
265         ee_p6_sum = ee_p6_sum / total_weight
266
267         ee_sums = np.array([ee_p0_sum,
268                             ee_p1_sum,
269                             ee_p2_sum,
270                             ee_p3_sum,
271                             ee_p4_sum,
272                             ee_p5_sum,
273                             ee_p6_sum])
274
275     if np.any(pe != 0):
276         pe_p0_sum = 0.
277         pe_p1_sum = 0.
278         pe_p2_sum = 0.
279         pe_p3_sum = 0.
280         pe_p4_sum = 0.
281         pe_p5_sum = 0.
282         pe_p6_sum = 0.
```

```
283
284         total_weight = 0.
285         for i in range(t_dim):
286             i_weight = .65 ** i
287             total_weight += i_weight
288             pe_p0_sum += pe[0, i] * i_weight
289             pe_p1_sum += pe[1, i] * i_weight
290             pe_p2_sum += pe[2, i] * i_weight
291             pe_p3_sum += pe[3, i] * i_weight
292             pe_p4_sum += pe[4, i] * i_weight
293             pe_p5_sum += pe[5, i] * i_weight
294             pe_p6_sum += pe[6, i] * i_weight
295
296         pe_p0_sum = pe_p0_sum / total_weight
297         pe_p1_sum = pe_p1_sum / total_weight
298         pe_p2_sum = pe_p2_sum / total_weight
299         pe_p3_sum = pe_p3_sum / total_weight
300         pe_p4_sum = pe_p4_sum / total_weight
301         pe_p5_sum = pe_p5_sum / total_weight
302         pe_p6_sum = pe_p6_sum / total_weight
303
304         pe_sums = np.array([pe_p0_sum,
305                             pe_p1_sum,
306                             pe_p2_sum,
307                             pe_p3_sum,
308                             pe_p4_sum,
309                             pe_p5_sum,
310                             pe_p6_sum])
311
312     if np.any(st_sums != 0):
313         new_switch = np.zeros(2)
314         st_sorted = np.argsort(-st_sums)
315         new_switch[0] = st_sorted[0] + 1
316         new_switch[1] = st_sorted[1] + 1
317
318     if np.any(ee_sums != 0):
319         new_switch = np.zeros(2)
320         ee_sorted = np.argsort(-ee_sums)
321         new_switch[0] = ee_sorted[0] + 1
322         new_switch[1] = ee_sorted[1] + 1
323
324     if np.any(pe_sums != 0):
325         new_switch = np.zeros(2)
326         pe_sorted = np.argsort(-pe_sums)
327         new_switch[0] = pe_sorted[0] + 1
328         new_switch[1] = pe_sorted[1] + 1
329     return new_switch
330
331
332 def switch_rule3(switch, st, ee, pe):
333     p_dim, t_dim = st.shape[0], st.shape[1]
334     st_sums = np.zeros((p_dim, t_dim))
335     ee_sums = np.zeros((p_dim, t_dim))
336     pe_sums = np.zeros((p_dim, t_dim))
337
338     if np.any(st != 0):
339         st_p0_sum = 0.
340         st_p1_sum = 0.
341         st_p2_sum = 0.
342         st_p3_sum = 0.
343         st_p4_sum = 0.
344         st_p5_sum = 0.
345         st_p6_sum = 0.
346
347         total_weight = 0.
348         for i in range(t_dim):
349             i_weight = .65 ** i
350             total_weight += i_weight
351             st_p0_sum += st[0, i] * i_weight
352             st_p1_sum += st[1, i] * i_weight
353             st_p2_sum += st[2, i] * i_weight
354             st_p3_sum += st[3, i] * i_weight
355             st_p4_sum += st[4, i] * i_weight
356             st_p5_sum += st[5, i] * i_weight
357             st_p6_sum += st[6, i] * i_weight
358
359         st_p0_sum = st_p0_sum / total_weight
360         st_p1_sum = st_p1_sum / total_weight
361         st_p2_sum = st_p2_sum / total_weight
362         st_p3_sum = st_p3_sum / total_weight
363         st_p4_sum = st_p4_sum / total_weight
364         st_p5_sum = st_p5_sum / total_weight
365         st_p6_sum = st_p6_sum / total_weight
366
367         st_sums = np.array([st_p0_sum,
368                             st_p1_sum,
369                             st_p2_sum,
370                             st_p3_sum,
371                             st_p4_sum,
372                             st_p5_sum,
373                             st_p6_sum])
374
375     if np.any(ee != 0):
376         ee_p0_sum = 0.
377         ee_p1_sum = 0.
```

```
378         ee_p2_sum = 0.
379         ee_p3_sum = 0.
380         ee_p4_sum = 0.
381         ee_p5_sum = 0.
382         ee_p6_sum = 0.
383 
384         total_weight = 0.
385         for i in range(t_dim):
386             i_weight = .65 ** i
387             total_weight += i_weight
388             ee_p0_sum += ee[0, i] * i_weight
389             ee_p1_sum += ee[1, i] * i_weight
390             ee_p2_sum += ee[2, i] * i_weight
391             ee_p3_sum += ee[3, i] * i_weight
392             ee_p4_sum += ee[4, i] * i_weight
393             ee_p5_sum += ee[5, i] * i_weight
394             ee_p6_sum += ee[6, i] * i_weight
395 
396         ee_p0_sum = ee_p0_sum / total_weight
397         ee_p1_sum = ee_p1_sum / total_weight
398         ee_p2_sum = ee_p2_sum / total_weight
399         ee_p3_sum = ee_p3_sum / total_weight
400         ee_p4_sum = ee_p4_sum / total_weight
401         ee_p5_sum = ee_p5_sum / total_weight
402         ee_p6_sum = ee_p6_sum / total_weight
403 
404         ee_sums = np.array([ee_p0_sum,
405                             ee_p1_sum,
406                             ee_p2_sum,
407                             ee_p3_sum,
408                             ee_p4_sum,
409                             ee_p5_sum,
410                             ee_p6_sum])
411 
412     if np.any(pe != 0):
413         pe_p0_sum = 0.
414         pe_p1_sum = 0.
415         pe_p2_sum = 0.
416         pe_p3_sum = 0.
417         pe_p4_sum = 0.
418         pe_p5_sum = 0.
419         pe_p6_sum = 0.
420 
421         total_weight = 0.
422         for i in range(t_dim):
423             i_weight = .65 ** i
424             total_weight += i_weight
425             pe_p0_sum += pe[0, i] * i_weight
426             pe_p1_sum += pe[1, i] * i_weight
427             pe_p2_sum += pe[2, i] * i_weight
428             pe_p3_sum += pe[3, i] * i_weight
429             pe_p4_sum += pe[4, i] * i_weight
430             pe_p5_sum += pe[5, i] * i_weight
431             pe_p6_sum += pe[6, i] * i_weight
432 
433         pe_p0_sum = pe_p0_sum / total_weight
434         pe_p1_sum = pe_p1_sum / total_weight
435         pe_p2_sum = pe_p2_sum / total_weight
436         pe_p3_sum = pe_p3_sum / total_weight
437         pe_p4_sum = pe_p4_sum / total_weight
438         pe_p5_sum = pe_p5_sum / total_weight
439         pe_p6_sum = pe_p6_sum / total_weight
440 
441         pe_sums = np.array([pe_p0_sum,
442                             pe_p1_sum,
443                             pe_p2_sum,
444                             pe_p3_sum,
445                             pe_p4_sum,
446                             pe_p5_sum,
447                             pe_p6_sum])
448 
449     if np.any(st_sums != 0):
450         new_switch = np.zeros(3)
451         st_sorted = np.argsort(-st_sums)
452         new_switch[0] = st_sorted[0] + 1
453         new_switch[1] = st_sorted[1] + 1
454         new_switch[2] = st_sorted[2] + 1
455 
456     if np.any(ee_sums != 0):
457         new_switch = np.zeros(3)
458         ee_sorted = np.argsort(-ee_sums)
459         new_switch[0] = ee_sorted[0] + 1
460         new_switch[1] = ee_sorted[1] + 1
461         new_switch[2] = ee_sorted[2] + 1
462 
463     if np.any(pe_sums != 0):
464         new_switch = np.zeros(3)
465         pe_sorted = np.argsort(-pe_sums)
466         new_switch[0] = pe_sorted[0] + 1
467         new_switch[1] = pe_sorted[1] + 1
468         new_switch[2] = pe_sorted[2] + 1
469     return new_switch
```

Listing 6.4: sensitiv.py - The sensitivity analysis for all the MPCs

The plot-file apart from those in the main-file

```python
import os as os
import numpy as np
import pathlib as pathlib
import matplotlib.pyplot as plt
from matplotlib import rcParams
from seaborn import swarmplot

ts = np.linspace(0., 150., 151)  # time-horizon
proj_dir = pathlib.Path(__file__).parent.parent

data_dir1 = os.path.join(proj_dir, "data/open_loop_nmpc_N25")
data_dir2 = os.path.join(proj_dir, "data/clsd_loop_nmpc_N25")
data_dir3 = os.path.join(proj_dir, "data/clsd_loop_rmpc1_N25")

data_dir4 = os.path.join(proj_dir, "data/clsd_loop_rmpc2_N25_N2e10")
data_dir5 = os.path.join(proj_dir, "data/clsd_loop_rmpc2_N25_N2e12")
data_dir6 = os.path.join(proj_dir, "data/clsd_loop_rmpc3_N25_N2e10")
data_dir7 = os.path.join(proj_dir, "data/clsd_loop_rmpc3_N25_N2e12")

filepath1_1 = os.path.join(data_dir1, "open_loop_nmpc_N25_x_opts.npy")
filepath2_1 = os.path.join(data_dir2, "clsd_loop_nmpc_N25_x_opts.npy")
filepath3_1 = os.path.join(data_dir3, "clsd_loop_rmpc1_N25_x_opts.npy")

filepath4_1 = os.path.join(data_dir4, "clsd_loop_rmpc2_N25_N2e10_tz01_SA1_x_opts.npy")
filepath4_2 = os.path.join(data_dir4, "clsd_loop_rmpc2_N25_N2e10_tz01_SA2_x_opts.npy")
filepath4_3 = os.path.join(data_dir4, "clsd_loop_rmpc2_N25_N2e10_tz01_SA3_x_opts.npy")

filepath4_4 = os.path.join(data_dir4, "clsd_loop_rmpc2_N25_N2e10_tz05_SA1_x_opts.npy")
filepath4_5 = os.path.join(data_dir4, "clsd_loop_rmpc2_N25_N2e10_tz05_SA2_x_opts.npy")
filepath4_6 = os.path.join(data_dir4, "clsd_loop_rmpc2_N25_N2e10_tz05_SA3_x_opts.npy")

filepath5_1 = os.path.join(data_dir5, "clsd_loop_rmpc2_N25_N2e12_tz01_SA1_x_opts.npy")
filepath5_2 = os.path.join(data_dir5, "clsd_loop_rmpc2_N25_N2e12_tz01_SA2_x_opts.npy")
filepath5_3 = os.path.join(data_dir5, "clsd_loop_rmpc2_N25_N2e12_tz01_SA3_x_opts.npy")

filepath5_4 = os.path.join(data_dir5, "clsd_loop_rmpc2_N25_N2e12_tz05_SA1_x_opts.npy")
filepath5_5 = os.path.join(data_dir5, "clsd_loop_rmpc2_N25_N2e12_tz05_SA2_x_opts.npy")
filepath5_6 = os.path.join(data_dir5, "clsd_loop_rmpc2_N25_N2e12_tz05_SA3_x_opts.npy")

filepath6_1 = os.path.join(data_dir6, "clsd_loop_rmpc3_N25_N2e10_tz01_SA1_x_opts.npy")
filepath6_2 = os.path.join(data_dir6, "clsd_loop_rmpc3_N25_N2e10_tz01_SA2_x_opts.npy")
filepath6_3 = os.path.join(data_dir6, "clsd_loop_rmpc3_N25_N2e10_tz01_SA3_x_opts.npy")

filepath6_4 = os.path.join(data_dir6, "clsd_loop_rmpc3_N25_N2e10_tz05_SA1_x_opts.npy")
filepath6_5 = os.path.join(data_dir6, "clsd_loop_rmpc3_N25_N2e10_tz05_SA2_x_opts.npy")
filepath6_6 = os.path.join(data_dir6, "clsd_loop_rmpc3_N25_N2e10_tz05_SA3_x_opts.npy")

filepath7_1 = os.path.join(data_dir7, "clsd_loop_rmpc3_N25_N2e12_tz01_SA1_x_opts.npy")
filepath7_2 = os.path.join(data_dir7, "clsd_loop_rmpc3_N25_N2e12_tz01_SA2_x_opts.npy")
filepath7_3 = os.path.join(data_dir7, "clsd_loop_rmpc3_N25_N2e12_tz01_SA3_x_opts.npy")

filepath7_4 = os.path.join(data_dir7, "clsd_loop_rmpc3_N25_N2e12_tz05_SA1_x_opts.npy")
filepath7_5 = os.path.join(data_dir7, "clsd_loop_rmpc3_N25_N2e12_tz05_SA2_x_opts.npy")
filepath7_6 = os.path.join(data_dir7, "clsd_loop_rmpc3_N25_N2e12_tz05_SA3_x_opts.npy")

x_opts_1_1 = np.load(filepath1_1)  # loading x_opts_1_1
x_opts_2_1 = np.load(filepath2_1)  # loading x_opts_2_1
x_opts_3_1 = np.load(filepath3_1)  # loading x_opts_3_1

x_opts_4_1 = np.load(filepath4_1)  # loading x_opts_4_1
x_opts_4_2 = np.load(filepath4_2)  # loading x_opts_4_2
x_opts_4_3 = np.load(filepath4_3)  # loading x_opts_4_3
x_opts_4_4 = np.load(filepath4_4)  # loading x_opts_4_4
x_opts_4_5 = np.load(filepath4_5)  # loading x_opts_4_5
x_opts_4_6 = np.load(filepath4_6)  # loading x_opts_4_6

x_opts_5_1 = np.load(filepath5_1)  # loading x_opts_5_1
x_opts_5_2 = np.load(filepath5_2)  # loading x_opts_5_2
x_opts_5_3 = np.load(filepath5_3)  # loading x_opts_5_3
x_opts_5_4 = np.load(filepath5_4)  # loading x_opts_5_4
x_opts_5_5 = np.load(filepath5_5)  # loading x_opts_5_5
x_opts_5_6 = np.load(filepath5_6)  # loading x_opts_5_6

x_opts_6_1 = np.load(filepath6_1)  # loading x_opts_6_1
x_opts_6_2 = np.load(filepath6_2)  # loading x_opts_6_2
x_opts_6_3 = np.load(filepath6_3)  # loading x_opts_6_3
x_opts_6_4 = np.load(filepath6_4)  # loading x_opts_6_4
x_opts_6_5 = np.load(filepath6_5)  # loading x_opts_6_5
x_opts_6_6 = np.load(filepath6_6)  # loading x_opts_6_6

x_opts_7_1 = np.load(filepath7_1)  # loading x_opts_7_1
x_opts_7_2 = np.load(filepath7_2)  # loading x_opts_7_2
x_opts_7_3 = np.load(filepath7_3)  # loading x_opts_7_3
x_opts_7_4 = np.load(filepath7_4)  # loading x_opts_7_4
x_opts_7_5 = np.load(filepath7_5)  # loading x_opts_7_5
x_opts_7_6 = np.load(filepath7_6)  # loading x_opts_7_6

x0_1_1_cvs = np.sum((x_opts_1_1[0, :, :] > 3.7), axis=0)
x0_1_1_worst_case_cv = x_opts_1_1[0, :, x0_1_1_cvs.argmax()]
if np.all(x0_1_1_cvs == 0):
    mx_1_1 = np.max(x_opts_1_1[0, :, :], axis=0)
    x0_1_1_worst_case_cv = x_opts_1_1[0, :, mx_1_1.argmax()]
```

```

x0_2_1_cvs = np.sum((x_opts_2_1[0, :, :] > 3.7), axis=0)
x0_2_1_worst_case_cv = x_opts_2_1[0, :, x0_2_1_cvs.argmax()]
if np.all(x0_2_1_cvs == 0):
    mx_2_1 = np.max(x_opts_2_1[0, :, :], axis=0)
    x0_2_1_worst_case_cv = x_opts_2_1[0, :, mx_2_1.argmax()]

x0_3_1_cvs = np.sum((x_opts_3_1[0, :, :] > 3.7), axis=0)
x0_3_1_worst_case_cv = x_opts_3_1[0, :, x0_3_1_cvs.argmax()]
if np.all(x0_3_1_cvs == 0):
    mx_3_1 = np.max(x_opts_3_1[0, :, :], axis=0)
    x0_3_1_worst_case_cv = x_opts_3_1[0, :, mx_3_1.argmax()]

x0_4_1_cvs = np.sum((x_opts_4_1[0, :, :] > 3.7), axis=0)
x0_4_1_worst_case_cv = x_opts_4_1[0, :, x0_4_1_cvs.argmax()]
if np.all(x0_4_1_cvs == 0):
    mx_4_1 = np.max(x_opts_4_1[0, :, :], axis=0)
    x0_4_1_worst_case_cv = x_opts_4_1[0, :, mx_4_1.argmax()]

x0_4_2_cvs = np.sum((x_opts_4_2[0, :, :] > 3.7), axis=0)
x0_4_2_worst_case_cv = x_opts_4_2[0, :, x0_4_2_cvs.argmax()]
if np.all(x0_4_2_cvs == 0):
    mx_4_2 = np.max(x_opts_4_2[0, :, :], axis=0)
    x0_4_2_worst_case_cv = x_opts_4_2[0, :, mx_4_2.argmax()]

x0_4_3_cvs = np.sum((x_opts_4_3[0, :, :] > 3.7), axis=0)
x0_4_3_worst_case_cv = x_opts_4_3[0, :, x0_4_3_cvs.argmax()]
if np.all(x0_4_3_cvs == 0):
    mx_4_3 = np.max(x_opts_4_3[0, :, :], axis=0)
    x0_4_3_worst_case_cv = x_opts_4_3[0, :, mx_4_3.argmax()]

x0_4_4_cvs = np.sum((x_opts_4_4[0, :, :] > 3.7), axis=0)
x0_4_4_worst_case_cv = x_opts_4_4[0, :, x0_4_4_cvs.argmax()]
if np.all(x0_4_4_cvs == 0):
    mx_4_4 = np.max(x_opts_4_4[0, :, :], axis=0)
    x0_4_4_worst_case_cv = x_opts_4_4[0, :, mx_4_4.argmax()]

x0_4_5_cvs = np.sum((x_opts_4_5[0, :, :] > 3.7), axis=0)
x0_4_5_worst_case_cv = x_opts_4_5[0, :, x0_4_5_cvs.argmax()]
if np.all(x0_4_5_cvs == 0):
    mx_4_5 = np.max(x_opts_4_5[0, :, :], axis=0)
    x0_4_5_worst_case_cv = x_opts_4_5[0, :, mx_4_5.argmax()]

x0_4_6_cvs = np.sum((x_opts_4_6[0, :, :] > 3.7), axis=0)
x0_4_6_worst_case_cv = x_opts_4_6[0, :, x0_4_6_cvs.argmax()]
if np.all(x0_4_6_cvs == 0):
    mx_4_6 = np.max(x_opts_4_6[0, :, :], axis=0)
    x0_4_6_worst_case_cv = x_opts_4_6[0, :, mx_4_6.argmax()]

x0_5_1_cvs = np.sum((x_opts_5_1[0, :, :] > 3.7), axis=0)
x0_5_1_worst_case_cv = x_opts_5_1[0, :, x0_5_1_cvs.argmax()]
if np.all(x0_5_1_cvs == 0):
    mx_5_1 = np.max(x_opts_5_1[0, :, :], axis=0)
    x0_5_1_worst_case_cv = x_opts_5_1[0, :, mx_5_1.argmax()]

x0_5_2_cvs = np.sum((x_opts_5_2[0, :, :] > 3.7), axis=0)
x0_5_2_worst_case_cv = x_opts_5_2[0, :, x0_5_2_cvs.argmax()]
if np.all(x0_5_2_cvs == 0):
    mx_5_2 = np.max(x_opts_5_2[0, :, :], axis=0)
    x0_5_2_worst_case_cv = x_opts_5_2[0, :, mx_5_2.argmax()]

x0_5_3_cvs = np.sum((x_opts_5_3[0, :, :] > 3.7), axis=0)
x0_5_3_worst_case_cv = x_opts_5_3[0, :, x0_5_3_cvs.argmax()]
if np.all(x0_5_3_cvs == 0):
    mx_5_3 = np.max(x_opts_5_3[0, :, :], axis=0)
    x0_5_3_worst_case_cv = x_opts_5_3[0, :, mx_5_3.argmax()]

x0_5_4_cvs = np.sum((x_opts_5_4[0, :, :] > 3.7), axis=0)
x0_5_4_worst_case_cv = x_opts_5_4[0, :, x0_5_4_cvs.argmax()]
if np.all(x0_5_4_cvs == 0):
    mx_5_4 = np.max(x_opts_5_4[0, :, :], axis=0)
    x0_5_4_worst_case_cv = x_opts_5_4[0, :, mx_5_4.argmax()]

x0_5_5_cvs = np.sum((x_opts_5_5[0, :, :] > 3.7), axis=0)
x0_5_5_worst_case_cv = x_opts_5_5[0, :, x0_5_5_cvs.argmax()]
if np.all(x0_5_5_cvs == 0):
    mx_5_5 = np.max(x_opts_5_5[0, :, :], axis=0)
    x0_5_5_worst_case_cv = x_opts_5_5[0, :, mx_5_5.argmax()]

x0_5_6_cvs = np.sum((x_opts_5_6[0, :, :] > 3.7), axis=0)
x0_5_6_worst_case_cv = x_opts_5_6[0, :, x0_5_6_cvs.argmax()]
if np.all(x0_5_6_cvs == 0):
    mx_5_6 = np.max(x_opts_5_6[0, :, :], axis=0)
    x0_5_6_worst_case_cv = x_opts_5_6[0, :, mx_5_6.argmax()]

x0_6_1_cvs = np.sum((x_opts_6_1[0, :, :] > 3.7), axis=0)
x0_6_1_worst_case_cv = x_opts_6_1[0, :, x0_6_1_cvs.argmax()]
if np.all(x0_6_1_cvs == 0):
    mx_6_1 = np.max(x_opts_6_1[0, :, :], axis=0)
    x0_6_1_worst_case_cv = x_opts_6_1[0, :, mx_6_1.argmax()]

x0_6_2_cvs = np.sum((x_opts_6_2[0, :, :] > 3.7), axis=0)
x0_6_2_worst_case_cv = x_opts_6_2[0, :, x0_6_2_cvs.argmax()]
if np.all(x0_6_2_cvs == 0):
    mx_6_2 = np.max(x_opts_6_2[0, :, :], axis=0)
```

```
188     x0_6_2_worst_case_cv = x_opts_6_2[0, :, mx_6_2.argmax()]
189 
190 x0_6_3_cvs = np.sum((x_opts_6_3[0, :, :] > 3.7), axis=0)
191 x0_6_3_worst_case_cv = x_opts_6_3[0, :, x0_6_3_cvs.argmax()]
192 if np.all(x0_6_3_cvs == 0):
193     mx_6_3 = np.max(x_opts_6_3[0, :, :], axis=0)
194     x0_6_3_worst_case_cv = x_opts_6_3[0, :, mx_6_3.argmax()]
195 
196 x0_6_4_cvs = np.sum((x_opts_6_4[0, :, :] > 3.7), axis=0)
197 x0_6_4_worst_case_cv = x_opts_6_4[0, :, x0_6_4_cvs.argmax()]
198 if np.all(x0_6_4_cvs == 0):
199     mx_6_4 = np.max(x_opts_6_4[0, :, :], axis=0)
200     x0_6_4_worst_case_cv = x_opts_6_4[0, :, mx_6_4.argmax()]
201 
202 x0_6_5_cvs = np.sum((x_opts_6_5[0, :, :] > 3.7), axis=0)
203 x0_6_5_worst_case_cv = x_opts_6_5[0, :, x0_6_5_cvs.argmax()]
204 if np.all(x0_6_5_cvs == 0):
205     mx_6_5 = np.max(x_opts_6_5[0, :, :], axis=0)
206     x0_6_5_worst_case_cv = x_opts_6_5[0, :, mx_6_5.argmax()]
207 
208 x0_6_6_cvs = np.sum((x_opts_6_6[0, :, :] > 3.7), axis=0)
209 x0_6_6_worst_case_cv = x_opts_6_6[0, :, x0_6_6_cvs.argmax()]
210 if np.all(x0_6_6_cvs == 0):
211     mx_6_6 = np.max(x_opts_6_6[0, :, :], axis=0)
212     x0_6_6_worst_case_cv = x_opts_6_6[0, :, mx_6_6.argmax()]
213 
214 x0_7_1_cvs = np.sum((x_opts_7_1[0, :, :] > 3.7), axis=0)
215 x0_7_1_worst_case_cv = x_opts_7_1[0, :, x0_7_1_cvs.argmax()]
216 if np.all(x0_7_1_cvs == 0):
217     mx_7_1 = np.max(x_opts_7_1[0, :, :], axis=0)
218     x0_7_1_worst_case_cv = x_opts_7_1[0, :, mx_7_1.argmax()]
219 
220 x0_7_2_cvs = np.sum((x_opts_7_2[0, :, :] > 3.7), axis=0)
221 x0_7_2_worst_case_cv = x_opts_7_2[0, :, x0_7_2_cvs.argmax()]
222 if np.all(x0_7_2_cvs == 0):
223     mx_7_2 = np.max(x_opts_7_2[0, :, :], axis=0)
224     x0_7_2_worst_case_cv = x_opts_7_2[0, :, mx_7_2.argmax()]
225 
226 x0_7_3_cvs = np.sum((x_opts_7_3[0, :, :] > 3.7), axis=0)
227 x0_7_3_worst_case_cv = x_opts_7_3[0, :, x0_7_3_cvs.argmax()]
228 if np.all(x0_7_3_cvs == 0):
229     mx_7_3 = np.max(x_opts_7_3[0, :, :], axis=0)
230     x0_7_3_worst_case_cv = x_opts_7_3[0, :, mx_7_3.argmax()]
231 
232 x0_7_4_cvs = np.sum((x_opts_7_4[0, :, :] > 3.7), axis=0)
233 x0_7_4_worst_case_cv = x_opts_7_4[0, :, x0_7_4_cvs.argmax()]
234 if np.all(x0_7_4_cvs == 0):
235     mx_7_4 = np.max(x_opts_7_4[0, :, :], axis=0)
236     x0_7_4_worst_case_cv = x_opts_7_4[0, :, mx_7_4.argmax()]
237 
238 x0_7_5_cvs = np.sum((x_opts_7_5[0, :, :] > 3.7), axis=0)
239 x0_7_5_worst_case_cv = x_opts_7_5[0, :, x0_7_5_cvs.argmax()]
240 if np.all(x0_7_5_cvs == 0):
241     mx_7_5 = np.max(x_opts_7_5[0, :, :], axis=0)
242     x0_7_5_worst_case_cv = x_opts_7_5[0, :, mx_7_5.argmax()]
243 
244 x0_7_6_cvs = np.sum((x_opts_7_6[0, :, :] > 3.7), axis=0)
245 x0_7_6_worst_case_cv = x_opts_7_6[0, :, x0_7_6_cvs.argmax()]
246 if np.all(x0_7_6_cvs == 0):
247     mx_7_6 = np.max(x_opts_7_6[0, :, :], axis=0)
248     x0_7_6_worst_case_cv = x_opts_7_6[0, :, mx_7_6.argmax()]
249 
250 x2_1_1 = x_opts_1_1[2, 150, :].flatten()
251 x2_2_1 = x_opts_2_1[2, 150, :].flatten()
252 x2_3_1 = x_opts_3_1[2, 150, :].flatten()
253 
254 x2_4_1 = x_opts_4_1[2, 150, :].flatten()
255 x2_4_2 = x_opts_4_2[2, 150, :].flatten()
256 x2_4_3 = x_opts_4_3[2, 150, :].flatten()
257 x2_4_4 = x_opts_4_4[2, 150, :].flatten()
258 x2_4_5 = x_opts_4_5[2, 150, :].flatten()
259 x2_4_6 = x_opts_4_6[2, 150, :].flatten()
260 
261 x2_5_1 = x_opts_5_1[2, 150, :].flatten()
262 x2_5_2 = x_opts_5_2[2, 150, :].flatten()
263 x2_5_3 = x_opts_5_3[2, 150, :].flatten()
264 x2_5_4 = x_opts_5_4[2, 150, :].flatten()
265 x2_5_5 = x_opts_5_5[2, 150, :].flatten()
266 x2_5_6 = x_opts_5_6[2, 150, :].flatten()
267 
268 x2_6_1 = x_opts_6_1[2, 150, :].flatten()
269 x2_6_2 = x_opts_6_2[2, 150, :].flatten()
270 x2_6_3 = x_opts_6_3[2, 150, :].flatten()
271 x2_6_4 = x_opts_6_4[2, 150, :].flatten()
272 x2_6_5 = x_opts_6_5[2, 150, :].flatten()
273 x2_6_6 = x_opts_6_6[2, 150, :].flatten()
274 
275 x2_7_1 = x_opts_7_1[2, 150, :].flatten()
276 x2_7_2 = x_opts_7_2[2, 150, :].flatten()
277 x2_7_3 = x_opts_7_3[2, 150, :].flatten()
278 x2_7_4 = x_opts_7_4[2, 150, :].flatten()
279 x2_7_5 = x_opts_7_5[2, 150, :].flatten()
280 x2_7_6 = x_opts_7_6[2, 150, :].flatten()
281 
282 x2_1_1_mean = x_opts_1_1[2, 150, :].mean()
```

```
283 x2_2_1_mean = x_opts_2_1[2, 150, :].mean()
284 x2_3_1_mean = x_opts_3_1[2, 150, :].mean()
285
286 x2_4_1_mean = x_opts_4_1[2, 150, :].mean()
287 x2_4_2_mean = x_opts_4_2[2, 150, :].mean()
288 x2_4_3_mean = x_opts_4_3[2, 150, :].mean()
289 x2_4_4_mean = x_opts_4_4[2, 150, :].mean()
290 x2_4_5_mean = x_opts_4_5[2, 150, :].mean()
291 x2_4_6_mean = x_opts_4_6[2, 150, :].mean()
292
293 x2_5_1_mean = x_opts_5_1[2, 150, :].mean()
294 x2_5_2_mean = x_opts_5_2[2, 150, :].mean()
295 x2_5_3_mean = x_opts_5_3[2, 150, :].mean()
296 x2_5_4_mean = x_opts_5_4[2, 150, :].mean()
297 x2_5_5_mean = x_opts_5_5[2, 150, :].mean()
298 x2_5_6_mean = x_opts_5_6[2, 150, :].mean()
299
300 x2_6_1_mean = x_opts_6_1[2, 150, :].mean()
301 x2_6_2_mean = x_opts_6_2[2, 150, :].mean()
302 x2_6_3_mean = x_opts_6_3[2, 150, :].mean()
303 x2_6_4_mean = x_opts_6_4[2, 150, :].mean()
304 x2_6_5_mean = x_opts_6_5[2, 150, :].mean()
305 x2_6_6_mean = x_opts_6_6[2, 150, :].mean()
306
307 x2_7_1_mean = x_opts_7_1[2, 150, :].mean()
308 x2_7_2_mean = x_opts_7_2[2, 150, :].mean()
309 x2_7_3_mean = x_opts_7_3[2, 150, :].mean()
310 x2_7_4_mean = x_opts_7_4[2, 150, :].mean()
311 x2_7_5_mean = x_opts_7_5[2, 150, :].mean()
312 x2_7_6_mean = x_opts_7_6[2, 150, :].mean()
313
314 x2_1_1_sigma = x_opts_1_1[2, 150, :].std()
315 x2_2_1_sigma = x_opts_2_1[2, 150, :].std()
316 x2_3_1_sigma = x_opts_3_1[2, 150, :].std()
317
318 x2_4_1_sigma = x_opts_4_1[2, 150, :].std()
319 x2_4_2_sigma = x_opts_4_2[2, 150, :].std()
320 x2_4_3_sigma = x_opts_4_3[2, 150, :].std()
321 x2_4_4_sigma = x_opts_4_4[2, 150, :].std()
322 x2_4_5_sigma = x_opts_4_5[2, 150, :].std()
323 x2_4_6_sigma = x_opts_4_6[2, 150, :].std()
324
325 x2_5_1_sigma = x_opts_5_1[2, 150, :].std()
326 x2_5_2_sigma = x_opts_5_2[2, 150, :].std()
327 x2_5_3_sigma = x_opts_5_3[2, 150, :].std()
328 x2_5_4_sigma = x_opts_5_4[2, 150, :].std()
329 x2_5_5_sigma = x_opts_5_5[2, 150, :].std()
330 x2_5_6_sigma = x_opts_5_6[2, 150, :].std()
331
332 x2_6_1_sigma = x_opts_6_1[2, 150, :].std()
333 x2_6_2_sigma = x_opts_6_2[2, 150, :].std()
334 x2_6_3_sigma = x_opts_6_3[2, 150, :].std()
335 x2_6_4_sigma = x_opts_6_4[2, 150, :].std()
336 x2_6_5_sigma = x_opts_6_5[2, 150, :].std()
337 x2_6_6_sigma = x_opts_6_6[2, 150, :].std()
338
339 x2_7_1_sigma = x_opts_7_1[2, 150, :].std()
340 x2_7_2_sigma = x_opts_7_2[2, 150, :].std()
341 x2_7_3_sigma = x_opts_7_3[2, 150, :].std()
342 x2_7_4_sigma = x_opts_7_4[2, 150, :].std()
343 x2_7_5_sigma = x_opts_7_5[2, 150, :].std()
344 x2_7_6_sigma = x_opts_7_6[2, 150, :].std()
345
346 x0_ax1_legend_bounds = tuple([0.565, 0.005])
347 x0_ax2_legend_bounds = tuple([0.565, 0.005])
348 x0_ax3_legend_bounds = tuple([0.565, 0.005])
349 x0_ax4_legend_bounds = tuple([0.565, 0.005])
350 x0_ax5_legend_bounds = tuple([0.565, 0.005])
351 x0_ax6_legend_bounds = tuple([0.565, 0.005])
352 x0_ax7_legend_bounds = tuple([0.565, 0.005])
353 x0_ax8_legend_bounds = tuple([0.565, 0.005])
354
355 mn_axA_legend_bounds = tuple([0.5850, 0.8250])
356 mn_axB_legend_bounds = tuple([0.5850, 0.9050])
357
358 x0_ax1_plotting_bounds = np.array([3.694, 3.701])
359 x0_ax2_plotting_bounds = np.array([3.694, 3.701])
360 x0_ax3_plotting_bounds = np.array([3.694, 3.701])
361 x0_ax4_plotting_bounds = np.array([3.694, 3.701])
362 x0_ax5_plotting_bounds = np.array([3.694, 3.701])
363 x0_ax6_plotting_bounds = np.array([3.694, 3.701])
364 x0_ax7_plotting_bounds = np.array([3.694, 3.701])
365 x0_ax8_plotting_bounds = np.array([3.694, 3.701])
366 x2_ax9_plotting_bounds = np.array([1.270, 1.950])
367
368 ts_ax1_plotting_bounds = np.array([105.0, 150.0])
369 ts_ax2_plotting_bounds = np.array([105.0, 150.0])
370 ts_ax3_plotting_bounds = np.array([105.0, 150.0])
371 ts_ax4_plotting_bounds = np.array([105.0, 150.0])
372 ts_ax5_plotting_bounds = np.array([105.0, 150.0])
373 ts_ax6_plotting_bounds = np.array([105.0, 150.0])
374 ts_ax7_plotting_bounds = np.array([105.0, 150.0])
375 ts_ax8_plotting_bounds = np.array([105.0, 150.0])
376
377 mn_axA_plotting_bounds = np.array([1.62150, 1.62430])
```

```
378 mn_axB_plotting_bounds = np.array([1.62300, 1.62430])
379
380 sg_axA_plotting_bounds = np.array([0.15722, 0.15767])
381 sg_axB_plotting_bounds = np.array([0.15722, 0.15767])
382
383 # Boolean deciding if we should plot the figure 'fig1'
384 plot_figure_1 = False
385 # Boolean deciding if we should plot the figure 'fig2'
386 plot_figure_2 = False
387 # Boolean deciding if we should plot the figure 'fig3'
388 plot_figure_3 = False
389 # Boolean deciding if we should plot the figure 'fig4'
390 plot_figure_4 = False
391 # Boolean deciding if we should plot the figure 'fig5'
392 plot_figure_5 = False
393 # Boolean deciding if we should plot the figure 'fig6'
394 plot_figure_6 = False
395 # Boolean deciding if we should plot the figure 'fig7'
396 plot_figure_7 = False
397 # Boolean deciding if we should plot the figure 'fig8'
398 plot_figure_8 = False
399 # Boolean deciding if we should plot the figure 'fig9'
400 plot_figure_9 = False
401 # Boolean deciding if we should plot the figure 'figA'
402 plot_figure_A = False
403 # Boolean deciding if we should plot the figure 'figB'
404 plot_figure_B = False
405
406 rcParams["axes.grid"] = False
407 rcParams["text.usetex"] = True
408 rcParams["axes.titlesize"] = 27.
409 rcParams["axes.labelsize"] = 27.
410 rcParams["xtick.labelsize"] = 23.
411 rcParams["ytick.labelsize"] = 23.
412 rcParams["legend.fontsize"] = 23.
413
414 rcParams["figure.constrained_layout.use"] = True
415 rcParams["figure.constrained_layout.hspace"] = .0200
416 rcParams["figure.constrained_layout.wspace"] = .0200
417 rcParams["figure.constrained_layout.h_pad"] = .04167
418 rcParams["figure.constrained_layout.w_pad"] = .04167
419 cl = (rcParams["axes.prop_cycle"].by_key()["color"])
420
421 if plot_figure_1:
422     fig1, ax1 = plt.subplots(nrows=1, ncols=1, sharex="all", figsize=(10, 10))
423     ax1.hlines(3.7, ts[0], ts[-1], linestyle="--", linewidth=2., alpha=.95, color="k", label=r"$X\:{\leq}\:{3.7}$")
424     ax1.plot(ts, x0_3_1_worst_case_cv, linewidth=2., alpha=.95, color=cl[0], label=r"$X~  (\textrm{CNST-switch})$")
425     ax1.plot(ts, x0_4_1_worst_case_cv, linewidth=2., alpha=.95, color=cl[1], label=r"$X~  (\textrm{SOBO-switch})$")
426     ax1.plot(ts, x0_4_2_worst_case_cv, linewidth=2., alpha=.95, color=cl[2], label=r"$X~  (\textrm{MORR-switch})$")
427     ax1.plot(ts, x0_4_3_worst_case_cv, linewidth=2., alpha=.95, color=cl[3], label=r"$X~  (\textrm{MMOR-switch})$")
428
429     ax1.set_ylabel(r"$X~[\textrm{g}/\textrm{l}]$")
430     ax1.set_xlabel(r"$t~          [\textrm{hr}]$")
431
432     ax1.set_ylim(x0_ax1_plotting_bounds)
433     ax1.set_xlim(ts_ax1_plotting_bounds)
434
435     ax1.legend(loc=x0_ax1_legend_bounds)
436     fig1.show()  # ------------------- #
437
438 if plot_figure_2:
439     fig2, ax2 = plt.subplots(nrows=1, ncols=1, sharex="all", figsize=(10, 10))
440     ax2.hlines(3.7, ts[0], ts[-1], linestyle="--", linewidth=2., alpha=.95, color="k", label=r"$X\:{\leq}\:{3.7}$")
441     ax2.plot(ts, x0_3_1_worst_case_cv, linewidth=2., alpha=.95, color=cl[0], label=r"$X~  (\textrm{CNST-switch})$")
442     ax2.plot(ts, x0_4_4_worst_case_cv, linewidth=2., alpha=.95, color=cl[1], label=r"$X~  (\textrm{SOBO-switch})$")
443     ax2.plot(ts, x0_4_5_worst_case_cv, linewidth=2., alpha=.95, color=cl[2], label=r"$X~  (\textrm{MORR-switch})$")
444     ax2.plot(ts, x0_4_6_worst_case_cv, linewidth=2., alpha=.95, color=cl[3], label=r"$X~  (\textrm{MMOR-switch})$")
445
446     ax2.set_ylabel(r"$X~[\textrm{g}/\textrm{l}]$")
447     ax2.set_xlabel(r"$t~          [\textrm{hr}]$")
448
449     ax2.set_ylim(x0_ax2_plotting_bounds)
450     ax2.set_xlim(ts_ax2_plotting_bounds)
451
452     ax2.legend(loc=x0_ax2_legend_bounds)
453     fig2.show()  # ------------------- #
454
455 if plot_figure_3:
456     fig3, ax3 = plt.subplots(nrows=1, ncols=1, sharex="all", figsize=(10, 10))
457     ax3.hlines(3.7, ts[0], ts[-1], linestyle="--", linewidth=2., alpha=.95, color="k", label=r"$X\:{\leq}\:{3.7}$")
458     ax3.plot(ts, x0_3_1_worst_case_cv, linewidth=2., alpha=.95, color=cl[0], label=r"$X~  (\textrm{CNST-switch})$")
459     ax3.plot(ts, x0_5_1_worst_case_cv, linewidth=2., alpha=.95, color=cl[1], label=r"$X~  (\textrm{SOBO-switch})$")
460     ax3.plot(ts, x0_5_2_worst_case_cv, linewidth=2., alpha=.95, color=cl[2], label=r"$X~  (\textrm{MORR-switch})$")
461     ax3.plot(ts, x0_5_3_worst_case_cv, linewidth=2., alpha=.95, color=cl[3], label=r"$X~  (\textrm{MMOR-switch})$")
462
463     ax3.set_ylabel(r"$X~[\textrm{g}/\textrm{l}]$")
464     ax3.set_xlabel(r"$t~          [\textrm{hr}]$")
465
466     ax3.set_ylim(x0_ax3_plotting_bounds)
467     ax3.set_xlim(ts_ax3_plotting_bounds)
468
469     ax3.legend(loc=x0_ax3_legend_bounds)
470     fig3.show()  # ------------------- #
471
472 if plot_figure_4:
```

```
473     fig4, ax4 = plt.subplots(nrows=1, ncols=1, sharex="all", figsize=(10, 10))
474     ax4.hlines(3.7, ts[0], ts[-1], linestyle="--", linewidth=2., alpha=.95, color="k", label=r"$X\:{\leq}\:{3.7}$")
475     ax4.plot(ts, x0_3_1_worst_case_cv, linewidth=2., alpha=.95, color=cl[0], label=r"$X~  (\textrm{CNST-switch})$")
476     ax4.plot(ts, x0_5_4_worst_case_cv, linewidth=2., alpha=.95, color=cl[1], label=r"$X~  (\textrm{SOBO-switch})$")
477     ax4.plot(ts, x0_5_5_worst_case_cv, linewidth=2., alpha=.95, color=cl[2], label=r"$X~  (\textrm{MORR-switch})$")
478     ax4.plot(ts, x0_5_6_worst_case_cv, linewidth=2., alpha=.95, color=cl[3], label=r"$X~  (\textrm{MMOR-switch})$")
479
480     ax4.set_ylabel(r"$X~[\textrm{g}/\textrm{l}]$")
481     ax4.set_xlabel(r"$t~           [\textrm{hr}]$")
482
483     ax4.set_ylim(x0_ax4_plotting_bounds)
484     ax4.set_xlim(ts_ax4_plotting_bounds)
485
486     ax4.legend(loc=x0_ax4_legend_bounds)
487     fig4.show()  # ------------------- #
488
489 if plot_figure_5:
490     fig5, ax5 = plt.subplots(nrows=1, ncols=1, sharex="all", figsize=(10, 10))
491     ax5.hlines(3.7, ts[0], ts[-1], linestyle="--", linewidth=2., alpha=.95, color="k", label=r"$X\:{\leq}\:{3.7}$")
492     ax5.plot(ts, x0_3_1_worst_case_cv, linewidth=2., alpha=.95, color=cl[0], label=r"$X~  (\textrm{CNST-switch})$")
493     ax5.plot(ts, x0_6_1_worst_case_cv, linewidth=2., alpha=.95, color=cl[1], label=r"$X~  (\textrm{SOBO-switch})$")
494     ax5.plot(ts, x0_6_2_worst_case_cv, linewidth=2., alpha=.95, color=cl[2], label=r"$X~  (\textrm{MORR-switch})$")
495     ax5.plot(ts, x0_6_3_worst_case_cv, linewidth=2., alpha=.95, color=cl[3], label=r"$X~  (\textrm{MMOR-switch})$")
496
497     ax5.set_ylabel(r"$X~[\textrm{g}/\textrm{l}]$")
498     ax5.set_xlabel(r"$t~           [\textrm{hr}]$")
499
500     ax5.set_ylim(x0_ax5_plotting_bounds)
501     ax5.set_xlim(ts_ax5_plotting_bounds)
502
503     ax5.legend(loc=x0_ax5_legend_bounds)
504     fig5.show()  # ------------------- #
505
506 if plot_figure_6:
507     fig6, ax6 = plt.subplots(nrows=1, ncols=1, sharex="all", figsize=(10, 10))
508     ax6.hlines(3.7, ts[0], ts[-1], linestyle="--", linewidth=2., alpha=.95, color="k", label=r"$X\:{\leq}\:{3.7}$")
509     ax6.plot(ts, x0_3_1_worst_case_cv, linewidth=2., alpha=.95, color=cl[0], label=r"$X~  (\textrm{CNST-switch})$")
510     ax6.plot(ts, x0_6_4_worst_case_cv, linewidth=2., alpha=.95, color=cl[1], label=r"$X~  (\textrm{SOBO-switch})$")
511     ax6.plot(ts, x0_6_5_worst_case_cv, linewidth=2., alpha=.95, color=cl[2], label=r"$X~  (\textrm{MORR-switch})$")
512     ax6.plot(ts, x0_6_6_worst_case_cv, linewidth=2., alpha=.95, color=cl[3], label=r"$X~  (\textrm{MMOR-switch})$")
513
514     ax6.set_ylabel(r"$X~[\textrm{g}/\textrm{l}]$")
515     ax6.set_xlabel(r"$t~           [\textrm{hr}]$")
516
517     ax6.set_ylim(x0_ax6_plotting_bounds)
518     ax6.set_xlim(ts_ax6_plotting_bounds)
519
520     ax6.legend(loc=x0_ax6_legend_bounds)
521     fig6.show()  # ------------------- #
522
523 if plot_figure_7:
524     fig7, ax7 = plt.subplots(nrows=1, ncols=1, sharex="all", figsize=(10, 10))
525     ax7.hlines(3.7, ts[0], ts[-1], linestyle="--", linewidth=2., alpha=.95, color="k", label=r"$X\:{\leq}\:{3.7}$")
526     ax7.plot(ts, x0_3_1_worst_case_cv, linewidth=2., alpha=.95, color=cl[0], label=r"$X~  (\textrm{CNST-switch})$")
527     ax7.plot(ts, x0_7_1_worst_case_cv, linewidth=2., alpha=.95, color=cl[1], label=r"$X~  (\textrm{SOBO-switch})$")
528     ax7.plot(ts, x0_7_2_worst_case_cv, linewidth=2., alpha=.95, color=cl[2], label=r"$X~  (\textrm{MORR-switch})$")
529     ax7.plot(ts, x0_7_3_worst_case_cv, linewidth=2., alpha=.95, color=cl[3], label=r"$X~  (\textrm{MMOR-switch})$")
530
531     ax7.set_ylabel(r"$X~[\textrm{g}/\textrm{l}]$")
532     ax7.set_xlabel(r"$t~           [\textrm{hr}]$")
533
534     ax7.set_ylim(x0_ax7_plotting_bounds)
535     ax7.set_xlim(ts_ax7_plotting_bounds)
536
537     ax7.legend(loc=x0_ax7_legend_bounds)
538     fig7.show()  # ------------------- #
539
540 if plot_figure_8:
541     fig8, ax8 = plt.subplots(nrows=1, ncols=1, sharex="all", figsize=(10, 10))
542     ax8.hlines(3.7, ts[0], ts[-1], linestyle="--", linewidth=2., alpha=.95, color="k", label=r"$X\:{\leq}\:{3.7}$")
543     ax8.plot(ts, x0_3_1_worst_case_cv, linewidth=2., alpha=.95, color=cl[0], label=r"$X~  (\textrm{CNST-switch})$")
544     ax8.plot(ts, x0_7_4_worst_case_cv, linewidth=2., alpha=.95, color=cl[1], label=r"$X~  (\textrm{SOBO-switch})$")
545     ax8.plot(ts, x0_7_5_worst_case_cv, linewidth=2., alpha=.95, color=cl[2], label=r"$X~  (\textrm{MORR-switch})$")
546     ax8.plot(ts, x0_7_6_worst_case_cv, linewidth=2., alpha=.95, color=cl[3], label=r"$X~  (\textrm{MMOR-switch})$")
547
548     ax8.set_ylabel(r"$X~[\textrm{g}/\textrm{l}]$")
549     ax8.set_xlabel(r"$t~           [\textrm{hr}]$")
550
551     ax8.set_ylim(x0_ax8_plotting_bounds)
552     ax8.set_xlim(ts_ax8_plotting_bounds)
553
554     ax8.legend(loc=x0_ax8_legend_bounds)
555     fig8.show()  # ------------------- #
556
557 rcParams["axes.grid"] = False
558 rcParams["text.usetex"] = True
559 rcParams["axes.titlesize"] = 27.
560 rcParams["axes.labelsize"] = 27.
561 rcParams["xtick.labelsize"] = 23.
562 rcParams["ytick.labelsize"] = 23.
563 rcParams["legend.fontsize"] = 23.
564
565 rcParams["figure.constrained_layout.use"] = True
566 rcParams["figure.constrained_layout.hspace"] = .0000
567 rcParams["figure.constrained_layout.wspace"] = .0000
```

```
568 rcParams["figure.constrained_layout.h_pad"] = .00000
569 rcParams["figure.constrained_layout.w_pad"] = .04167
570 cl = (rcParams["axes.prop_cycle"].by_key()["color"])
571
572 if plot_figure_9:
573     fig9, ax9 = plt.subplots(nrows=27, ncols=1, sharex="all", figsize=(15, 25))
574     swarmplot(data=x2_1_1, ax=ax9[0], size=10, orient="h", color=np.where(x0_1_1_cvs == 0, cl[0], cl[1]))
575     swarmplot(data=x2_2_1, ax=ax9[1], size=10, orient="h", color=np.where(x0_2_1_cvs == 0, cl[0], cl[1]))
576     swarmplot(data=x2_3_1, ax=ax9[2], size=10, orient="h", color=np.where(x0_3_1_cvs == 0, cl[0], cl[1]))
577
578     swarmplot(data=x2_4_1, ax=ax9[3], size=10, orient="h", color=np.where(x0_4_1_cvs == 0, cl[0], cl[1]))
579     swarmplot(data=x2_4_2, ax=ax9[4], size=10, orient="h", color=np.where(x0_4_2_cvs == 0, cl[0], cl[1]))
580     swarmplot(data=x2_4_3, ax=ax9[5], size=10, orient="h", color=np.where(x0_4_3_cvs == 0, cl[0], cl[1]))
581
582     swarmplot(data=x2_4_4, ax=ax9[6], size=10, orient="h", color=np.where(x0_4_4_cvs == 0, cl[0], cl[1]))
583     swarmplot(data=x2_4_5, ax=ax9[7], size=10, orient="h", color=np.where(x0_4_5_cvs == 0, cl[0], cl[1]))
584     swarmplot(data=x2_4_6, ax=ax9[8], size=10, orient="h", color=np.where(x0_4_6_cvs == 0, cl[0], cl[1]))
585
586     swarmplot(data=x2_5_1, ax=ax9[9], size=10, orient="h", color=np.where(x0_5_1_cvs == 0, cl[0], cl[1]))
587     swarmplot(data=x2_5_2, ax=ax9[10], size=10, orient="h", color=np.where(x0_5_2_cvs == 0, cl[0], cl[1]))
588     swarmplot(data=x2_5_3, ax=ax9[11], size=10, orient="h", color=np.where(x0_5_3_cvs == 0, cl[0], cl[1]))
589
590     swarmplot(data=x2_5_4, ax=ax9[12], size=10, orient="h", color=np.where(x0_5_4_cvs == 0, cl[0], cl[1]))
591     swarmplot(data=x2_5_5, ax=ax9[13], size=10, orient="h", color=np.where(x0_5_5_cvs == 0, cl[0], cl[1]))
592     swarmplot(data=x2_5_6, ax=ax9[14], size=10, orient="h", color=np.where(x0_5_6_cvs == 0, cl[0], cl[1]))
593
594     swarmplot(data=x2_6_1, ax=ax9[15], size=10, orient="h", color=np.where(x0_6_1_cvs == 0, cl[0], cl[1]))
595     swarmplot(data=x2_6_2, ax=ax9[16], size=10, orient="h", color=np.where(x0_6_2_cvs == 0, cl[0], cl[1]))
596     swarmplot(data=x2_6_3, ax=ax9[17], size=10, orient="h", color=np.where(x0_6_3_cvs == 0, cl[0], cl[1]))
597
598     swarmplot(data=x2_6_4, ax=ax9[18], size=10, orient="h", color=np.where(x0_6_4_cvs == 0, cl[0], cl[1]))
599     swarmplot(data=x2_6_5, ax=ax9[19], size=10, orient="h", color=np.where(x0_6_5_cvs == 0, cl[0], cl[1]))
600     swarmplot(data=x2_6_6, ax=ax9[20], size=10, orient="h", color=np.where(x0_6_6_cvs == 0, cl[0], cl[1]))
601
602     swarmplot(data=x2_7_1, ax=ax9[21], size=10, orient="h", color=np.where(x0_7_1_cvs == 0, cl[0], cl[1]))
603     swarmplot(data=x2_7_2, ax=ax9[22], size=10, orient="h", color=np.where(x0_7_2_cvs == 0, cl[0], cl[1]))
604     swarmplot(data=x2_7_3, ax=ax9[23], size=10, orient="h", color=np.where(x0_7_3_cvs == 0, cl[0], cl[1]))
605
606     swarmplot(data=x2_7_4, ax=ax9[24], size=10, orient="h", color=np.where(x0_7_4_cvs == 0, cl[0], cl[1]))
607     swarmplot(data=x2_7_5, ax=ax9[25], size=10, orient="h", color=np.where(x0_7_5_cvs == 0, cl[0], cl[1]))
608     swarmplot(data=x2_7_6, ax=ax9[26], size=10, orient="h", color=np.where(x0_7_6_cvs == 0, cl[0], cl[1]))
609
610     ax9[0].set_ylabel(r"\textrm{OL-MPC                                    }",
611                       rotation=0., va="center", ha="left", ma="left", labelpad=545)
612     ax9[1].set_ylabel(r"\textrm{CL-MPC                                    }",
613                       rotation=0., va="center", ha="left", ma="left", labelpad=545)
614     ax9[2].set_ylabel(r"\textrm{2-UP-MS-MPC, CNST-switch                  }",
615                       rotation=0., va="center", ha="left", ma="left", labelpad=545)
616
617     ax9[3].set_ylabel(r"\textrm{2-UP-MS-MPC, SOBO-switch,} $N=2^{10}, t_{SA}=1$",
618                       rotation=0., va="center", ha="left", ma="left", labelpad=545)
619     ax9[4].set_ylabel(r"\textrm{2-UP-MS-MPC, MORR-switch,} $N=2^{10}, t_{SA}=1$",
620                       rotation=0., va="center", ha="left", ma="left", labelpad=545)
621     ax9[5].set_ylabel(r"\textrm{2-UP-MS-MPC, MMOR-switch,} $N=2^{10}, t_{SA}=1$",
622                       rotation=0., va="center", ha="left", ma="left", labelpad=545)
623
624     ax9[6].set_ylabel(r"\textrm{2-UP-MS-MPC, SOBO-switch,} $N=2^{10}, t_{SA}=5$",
625                       rotation=0., va="center", ha="left", ma="left", labelpad=545)
626     ax9[7].set_ylabel(r"\textrm{2-UP-MS-MPC, MORR-switch,} $N=2^{10}, t_{SA}=5$",
627                       rotation=0., va="center", ha="left", ma="left", labelpad=545)
628     ax9[8].set_ylabel(r"\textrm{2-UP-MS-MPC, MMOR-switch,} $N=2^{10}, t_{SA}=5$",
629                       rotation=0., va="center", ha="left", ma="left", labelpad=545)
630
631     ax9[9].set_ylabel(r"\textrm{2-UP-MS-MPC, SOBO-switch,} $N=2^{12}, t_{SA}=1$",
632                       rotation=0., va="center", ha="left", ma="left", labelpad=545)
633     ax9[10].set_ylabel(r"\textrm{2-UP-MS-MPC, MORR-switch,} $N=2^{12}, t_{SA}=1$",
634                        rotation=0., va="center", ha="left", ma="left", labelpad=545)
635     ax9[11].set_ylabel(r"\textrm{2-UP-MS-MPC, MMOR-switch,} $N=2^{12}, t_{SA}=1$",
636                        rotation=0., va="center", ha="left", ma="left", labelpad=545)
637
638     ax9[12].set_ylabel(r"\textrm{2-UP-MS-MPC, SOBO-switch,} $N=2^{12}, t_{SA}=5$",
639                        rotation=0., va="center", ha="left", ma="left", labelpad=545)
640     ax9[13].set_ylabel(r"\textrm{2-UP-MS-MPC, MORR-switch,} $N=2^{12}, t_{SA}=5$",
641                        rotation=0., va="center", ha="left", ma="left", labelpad=545)
642     ax9[14].set_ylabel(r"\textrm{2-UP-MS-MPC, MMOR-switch,} $N=2^{12}, t_{SA}=5$",
643                        rotation=0., va="center", ha="left", ma="left", labelpad=545)
644
645     ax9[15].set_ylabel(r"\textrm{3-UP-MS-MPC, SOBO-switch,} $N=2^{10}, t_{SA}=1$",
646                        rotation=0., va="center", ha="left", ma="left", labelpad=545)
647     ax9[16].set_ylabel(r"\textrm{3-UP-MS-MPC, MORR-switch,} $N=2^{10}, t_{SA}=1$",
648                        rotation=0., va="center", ha="left", ma="left", labelpad=545)
649     ax9[17].set_ylabel(r"\textrm{3-UP-MS-MPC, MMOR-switch,} $N=2^{10}, t_{SA}=1$",
650                        rotation=0., va="center", ha="left", ma="left", labelpad=545)
651
652     ax9[18].set_ylabel(r"\textrm{3-UP-MS-MPC, SOBO-switch,} $N=2^{10}, t_{SA}=5$",
653                        rotation=0., va="center", ha="left", ma="left", labelpad=545)
654     ax9[19].set_ylabel(r"\textrm{3-UP-MS-MPC, MORR-switch,} $N=2^{10}, t_{SA}=5$",
655                        rotation=0., va="center", ha="left", ma="left", labelpad=545)
656     ax9[20].set_ylabel(r"\textrm{3-UP-MS-MPC, MMOR-switch,} $N=2^{10}, t_{SA}=5$",
657                        rotation=0., va="center", ha="left", ma="left", labelpad=545)
658
659     ax9[21].set_ylabel(r"\textrm{3-UP-MS-MPC, SOBO-switch,} $N=2^{12}, t_{SA}=1$",
660                        rotation=0., va="center", ha="left", ma="left", labelpad=545)
661     ax9[22].set_ylabel(r"\textrm{3-UP-MS-MPC, MORR-switch,} $N=2^{12}, t_{SA}=1$",
662                        rotation=0., va="center", ha="left", ma="left", labelpad=545)
```

```
663     ax9[23].set_ylabel(r"\textrm{3-UP-MS-MPC, MMOR-switch,} $N=2^{12}, t_{SA}=1$",
664                        rotation=0., va="center", ha="left", ma="left", labelpad=545)
665 
666     ax9[24].set_ylabel(r"\textrm{3-UP-MS-MPC, SOBO-switch,} $N=2^{12}, t_{SA}=5$",
667                        rotation=0., va="center", ha="left", ma="left", labelpad=545)
668     ax9[25].set_ylabel(r"\textrm{3-UP-MS-MPC, MORR-switch,} $N=2^{12}, t_{SA}=5$",
669                        rotation=0., va="center", ha="left", ma="left", labelpad=545)
670     ax9[26].set_ylabel(r"\textrm{3-UP-MS-MPC, MMOR-switch,} $N=2^{12}, t_{SA}=5$",
671                        rotation=0., va="center", ha="left", ma="left", labelpad=545)
672 
673     ax9[0].tick_params(left=False, bottom=True, top=False, right=False)
674     ax9[1].tick_params(left=False, bottom=True, top=False, right=False)
675     ax9[2].tick_params(left=False, bottom=True, top=False, right=False)
676 
677     ax9[3].tick_params(left=False, bottom=True, top=False, right=False)
678     ax9[4].tick_params(left=False, bottom=True, top=False, right=False)
679     ax9[5].tick_params(left=False, bottom=True, top=False, right=False)
680 
681     ax9[6].tick_params(left=False, bottom=True, top=False, right=False)
682     ax9[7].tick_params(left=False, bottom=True, top=False, right=False)
683     ax9[8].tick_params(left=False, bottom=True, top=False, right=False)
684 
685     ax9[9].tick_params(left=False, bottom=True, top=False, right=False)
686     ax9[10].tick_params(left=False, bottom=True, top=False, right=False)
687     ax9[11].tick_params(left=False, bottom=True, top=False, right=False)
688 
689     ax9[12].tick_params(left=False, bottom=True, top=False, right=False)
690     ax9[13].tick_params(left=False, bottom=True, top=False, right=False)
691     ax9[14].tick_params(left=False, bottom=True, top=False, right=False)
692 
693     ax9[15].tick_params(left=False, bottom=True, top=False, right=False)
694     ax9[16].tick_params(left=False, bottom=True, top=False, right=False)
695     ax9[17].tick_params(left=False, bottom=True, top=False, right=False)
696 
697     ax9[18].tick_params(left=False, bottom=True, top=False, right=False)
698     ax9[19].tick_params(left=False, bottom=True, top=False, right=False)
699     ax9[20].tick_params(left=False, bottom=True, top=False, right=False)
700 
701     ax9[21].tick_params(left=False, bottom=True, top=False, right=False)
702     ax9[22].tick_params(left=False, bottom=True, top=False, right=False)
703     ax9[23].tick_params(left=False, bottom=True, top=False, right=False)
704 
705     ax9[24].tick_params(left=False, bottom=True, top=False, right=False)
706     ax9[25].tick_params(left=False, bottom=True, top=False, right=False)
707     ax9[26].tick_params(left=False, bottom=True, top=False, right=False)
708 
709     ax9[26].set_xlabel(r"$P~[\textrm{g}/\textrm{l}]$")
710 
711     ax9[26].set_xlim(x2_ax9_plotting_bounds)
712     fig9.show()  # ---------------------- #
713 
714 rcParams["axes.grid"] = False
715 rcParams["text.usetex"] = True
716 rcParams["axes.titlesize"] = 27.
717 rcParams["axes.labelsize"] = 27.
718 rcParams["xtick.labelsize"] = 23.
719 rcParams["ytick.labelsize"] = 23.
720 rcParams["legend.fontsize"] = 23.
721 
722 rcParams["figure.constrained_layout.use"] = True
723 rcParams["figure.constrained_layout.hspace"] = .0200
724 rcParams["figure.constrained_layout.wspace"] = .0200
725 rcParams["figure.constrained_layout.h_pad"] = .04167
726 rcParams["figure.constrained_layout.w_pad"] = .04167
727 cl = (rcParams["axes.prop_cycle"].by_key()["color"])
728 
729 if plot_figure_A:
730     figA, axA = plt.subplots(1, 1, sharex="all", figsize=(12, 12))
731     sg_data = np.array([x2_4_1_sigma, x2_4_2_sigma, x2_4_3_sigma,
732                         x2_4_4_sigma, x2_4_5_sigma, x2_4_6_sigma,
733                         x2_5_1_sigma, x2_5_2_sigma, x2_5_3_sigma,
734                         x2_5_4_sigma, x2_5_5_sigma, x2_5_6_sigma,
735                         x2_6_1_sigma, x2_6_2_sigma, x2_6_3_sigma,
736                         x2_6_4_sigma, x2_6_5_sigma, x2_6_6_sigma,
737                         x2_7_1_sigma, x2_7_2_sigma, x2_7_3_sigma,
738                         x2_7_4_sigma, x2_7_5_sigma, x2_7_6_sigma])
739     mn_data = np.array([x2_4_1_mean, x2_4_2_mean, x2_4_3_mean,
740                         x2_4_4_mean, x2_4_5_mean, x2_4_6_mean,
741                         x2_5_1_mean, x2_5_2_mean, x2_5_3_mean,
742                         x2_5_4_mean, x2_5_5_mean, x2_5_6_mean,
743                         x2_6_1_mean, x2_6_2_mean, x2_6_3_mean,
744                         x2_6_4_mean, x2_6_5_mean, x2_6_6_mean,
745                         x2_7_1_mean, x2_7_2_mean, x2_7_3_mean,
746                         x2_7_4_mean, x2_7_5_mean, x2_7_6_mean])
747     first_degree_poly = np.polyfit(x=sg_data, y=mn_data, deg=1)
748     poly = (sg_data * first_degree_poly[0] + first_degree_poly[1])
749     axA.plot(sg_data, poly, linewidth=2., alpha=.95, color="k")
750 
751     axA.scatter(x2_4_1_sigma, x2_4_1_mean, marker="o", alpha=.95, s=95., c=cl[0],
752                 label=r"\textrm{2-UP-MS-MPC}, $t_{SA}=1$")
753     axA.scatter(x2_4_2_sigma, x2_4_2_mean, marker="o", alpha=.95, s=95., c=cl[0])
754     axA.scatter(x2_4_3_sigma, x2_4_3_mean, marker="o", alpha=.95, s=95., c=cl[0])
755 
756     axA.scatter(x2_4_4_sigma, x2_4_4_mean, marker="^", alpha=.95, s=95., c=cl[0],
757                 label=r"\textrm{2-UP-MS-MPC}, $t_{SA}=5$")
```

```
    axA.scatter(x2_4_5_sigma, x2_4_5_mean, marker="^", alpha=.95, s=95., c=cl[0])
    axA.scatter(x2_4_6_sigma, x2_4_6_mean, marker="^", alpha=.95, s=95., c=cl[0])

    axA.scatter(x2_5_1_sigma, x2_5_1_mean, marker="o", alpha=.95, s=95., c=cl[0])
    axA.scatter(x2_5_2_sigma, x2_5_2_mean, marker="o", alpha=.95, s=95., c=cl[0])
    axA.scatter(x2_5_3_sigma, x2_5_3_mean, marker="o", alpha=.95, s=95., c=cl[0])

    axA.scatter(x2_5_4_sigma, x2_5_4_mean, marker="^", alpha=.95, s=95., c=cl[0])
    axA.scatter(x2_5_5_sigma, x2_5_5_mean, marker="^", alpha=.95, s=95., c=cl[0])
    axA.scatter(x2_5_6_sigma, x2_5_6_mean, marker="^", alpha=.95, s=95., c=cl[0])

    axA.scatter(x2_6_1_sigma, x2_6_1_mean, marker="o", alpha=.95, s=95., c=cl[1],
                label=r"\textrm{3-UP-MS-MPC}, $t_{SA}=1$")
    axA.scatter(x2_6_2_sigma, x2_6_2_mean, marker="o", alpha=.95, s=95., c=cl[1])
    axA.scatter(x2_6_3_sigma, x2_6_3_mean, marker="o", alpha=.95, s=95., c=cl[1])

    axA.scatter(x2_6_4_sigma, x2_6_4_mean, marker="^", alpha=.95, s=95., c=cl[1],
                label=r"\textrm{3-UP-MS-MPC}, $t_{SA}=5$")
    axA.scatter(x2_6_5_sigma, x2_6_5_mean, marker="^", alpha=.95, s=95., c=cl[1])
    axA.scatter(x2_6_6_sigma, x2_6_6_mean, marker="^", alpha=.95, s=95., c=cl[1])

    axA.scatter(x2_7_1_sigma, x2_7_1_mean, marker="o", alpha=.95, s=95., c=cl[1])
    axA.scatter(x2_7_2_sigma, x2_7_2_mean, marker="o", alpha=.95, s=95., c=cl[1])
    axA.scatter(x2_7_3_sigma, x2_7_3_mean, marker="o", alpha=.95, s=95., c=cl[1])

    axA.scatter(x2_7_4_sigma, x2_7_4_mean, marker="^", alpha=.95, s=95., c=cl[1])
    axA.scatter(x2_7_5_sigma, x2_7_5_mean, marker="^", alpha=.95, s=95., c=cl[1])
    axA.scatter(x2_7_6_sigma, x2_7_6_mean, marker="^", alpha=.95, s=95., c=cl[1])

    axA.set_ylabel(r"$ E(P)~[\textrm{g}/\textrm{l}]$")
    axA.set_xlabel(r"$SD(P)~[\textrm{g}/\textrm{l}]$")

    axA.set_ylim(mn_axA_plotting_bounds)
    axA.set_xlim(sg_axA_plotting_bounds)

    axA.legend(loc=mn_axA_legend_bounds)
    figA.show()  # ------------------ #

if plot_figure_B:
    figB, axB = plt.subplots(1, 1, sharex="all", figsize=(12, 12))
    sg_data = np.array([x2_4_1_sigma, x2_4_2_sigma, x2_4_3_sigma,
                        x2_4_4_sigma, x2_4_5_sigma, x2_4_6_sigma,
                        x2_5_1_sigma, x2_5_2_sigma, x2_5_3_sigma,
                        x2_5_4_sigma, x2_5_5_sigma, x2_5_6_sigma])
    mn_data = np.array([x2_4_1_mean, x2_4_2_mean, x2_4_3_mean,
                        x2_4_4_mean, x2_4_5_mean, x2_4_6_mean,
                        x2_5_1_mean, x2_5_2_mean, x2_5_3_mean,
                        x2_5_4_mean, x2_5_5_mean, x2_5_6_mean])
    first_degree_poly = np.polyfit(x=sg_data, y=mn_data, deg=1)
    poly = (sg_data * first_degree_poly[0] + first_degree_poly[1])
    axB.plot(sg_data, poly, linewidth=2., alpha=.95, color="k")

    axB.scatter(x2_4_1_sigma, x2_4_1_mean, marker="o", alpha=.95, s=95., c=cl[0],
                label=r"\textrm{2-UP-MS-MPC}, $t_{SA}=1$")
    axB.scatter(x2_4_2_sigma, x2_4_2_mean, marker="o", alpha=.95, s=95., c=cl[0])
    axB.scatter(x2_4_3_sigma, x2_4_3_mean, marker="o", alpha=.95, s=95., c=cl[0])

    axB.scatter(x2_4_4_sigma, x2_4_4_mean, marker="^", alpha=.95, s=95., c=cl[0],
                label=r"\textrm{2-UP-MS-MPC}, $t_{SA}=5$")
    axB.scatter(x2_4_5_sigma, x2_4_5_mean, marker="^", alpha=.95, s=95., c=cl[0])
    axB.scatter(x2_4_6_sigma, x2_4_6_mean, marker="^", alpha=.95, s=95., c=cl[0])

    axB.scatter(x2_5_1_sigma, x2_5_1_mean, marker="o", alpha=.95, s=95., c=cl[0])
    axB.scatter(x2_5_2_sigma, x2_5_2_mean, marker="o", alpha=.95, s=95., c=cl[0])
    axB.scatter(x2_5_3_sigma, x2_5_3_mean, marker="o", alpha=.95, s=95., c=cl[0])

    axB.scatter(x2_5_4_sigma, x2_5_4_mean, marker="^", alpha=.95, s=95., c=cl[0])
    axB.scatter(x2_5_5_sigma, x2_5_5_mean, marker="^", alpha=.95, s=95., c=cl[0])
    axB.scatter(x2_5_6_sigma, x2_5_6_mean, marker="^", alpha=.95, s=95., c=cl[0])

    axB.set_ylabel(r"$ E(P)~[\textrm{g}/\textrm{l}]$")
    axB.set_xlabel(r"$SD(P)~[\textrm{g}/\textrm{l}]$")

    axB.set_ylim(mn_axB_plotting_bounds)
    axB.set_xlim(sg_axB_plotting_bounds)

    axB.legend(loc=mn_axB_legend_bounds)
    figB.show()  # ------------------ #
```

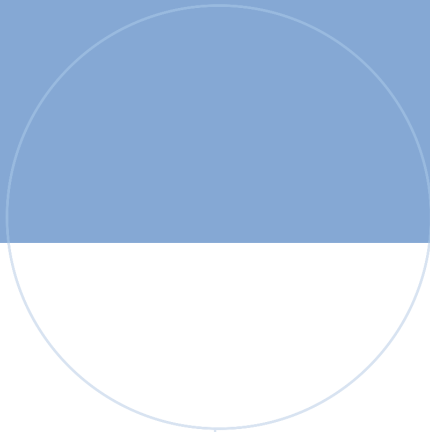
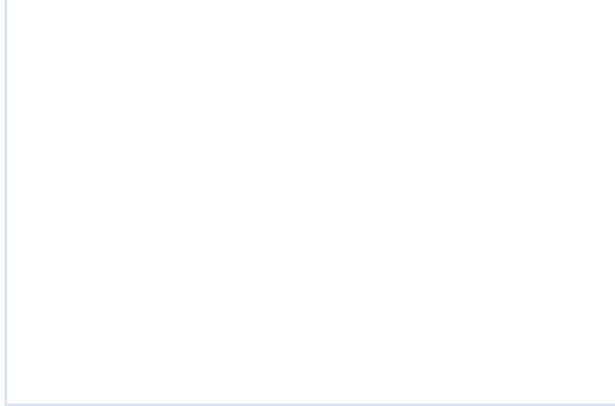
Listing 6.5: utilities.py - The plot-file apart from those in the main-file

The plot-file for visualizing the sampling methods

```
import numpy as np
import scipy.stats as sc
import matplotlib.pyplot as plt
from matplotlib import rcParams


def rnd_sampl(num_points):
    # Obtaining the random values of both 'X1' & 'X2'
    x = np.random.rand(num_points, 2)
```

```python
    return x[:, 0], x[:, 1]


def lhs_sampl(num_points):
    # Obtaining the random values of both 'X1' & 'X2'
    x = sc.qmc.LatinHypercube(d=2).random(num_points)
    return x[:, 0], x[:, 1]


def mor_sampl(num_points):
    p = 4  # levels in the grid space 'omega'
    delta = p / (2 * (p - 1))  # +-increments
    omega = np.linspace(0, 1 - delta, p // 2)
    B_star = np.zeros((2, 2, num_points))
    for i in range(num_points):
        # Finding starting points 'x_star' for the trajectories
        x_star = np.random.choice(omega, size=2)
        # Obtaining strictly lower triangular matrix 'B' of 1's
        B = np.tril(np.ones((2, 2)), -1)
        # Obtaining matrix 'J' of 1's for the copying of x_star
        J = np.ones((2, 2))
        # Obtaining matrix 'D_star', factor moves with +-delta?
        D_star = np.diag(np.random.choice([-1, 1], size=2))
        # Obtaining random permutation matrix 'P_star' of 0/1's
        P_star = np.random.permutation(np.eye(2, 2))
        # Finally obtaining randomized sampling matrix 'B_star'
        B_star[:, :, i] = (((J[0, :] * x_star) + (delta / 2.) * ((2 * B - J) @ D_star + J)) @ P_star)
    # Obtaining the random values of both 'X1' & 'X2'
    return B_star[:, 0, :], B_star[:, 1, :]


num_points = 50  # use few num_points
x1_rand, x2_rand = rnd_sampl(num_points)
x1_latn, x2_latn = lhs_sampl(num_points)
x1_morr, x2_morr = mor_sampl(num_points)

rcParams["axes.grid"] = True
rcParams["text.usetex"] = True
rcParams["grid.linewidth"] = 2.
rcParams["axes.titlesize"] = 27.
rcParams["axes.labelsize"] = 27.
rcParams["xtick.labelsize"] = 23.
rcParams["ytick.labelsize"] = 23.
rcParams["legend.fontsize"] = 23.

rcParams["figure.constrained_layout.use"] = True
rcParams["figure.constrained_layout.hspace"] = .0200
rcParams["figure.constrained_layout.wspace"] = .0200
rcParams["figure.constrained_layout.h_pad"] = .04167
rcParams["figure.constrained_layout.w_pad"] = .04167
cl = (rcParams["axes.prop_cycle"].by_key()["color"])

fig, ax = plt.subplots(nrows=1, ncols=3, sharex="all", figsize=(30, 10))
ax[0].scatter(x1_rand, x2_rand, marker="o", alpha=.95, s=150., c=cl[0])
ax[1].scatter(x1_latn, x2_latn, marker="o", alpha=.95, s=150., c=cl[0])
ax[2].scatter(x1_morr, x2_morr, marker="o", alpha=.95, s=150., c=cl[0])

ax[0].set_title(r"$          \textrm{Random sampling}$")
ax[1].set_title(r"$\textrm{Latin Hypercube sampling}$")
ax[2].set_title(r"$          \textrm{Morris sampling}$")

ax[0].set_ylabel(r"$X_{2}$")
ax[0].set_xlabel(r"$X_{1}$")
ax[1].set_ylabel(r"$X_{2}$")
ax[1].set_xlabel(r"$X_{1}$")
ax[2].set_ylabel(r"$X_{2}$")
ax[2].set_xlabel(r"$X_{1}$")

ax[0].set_ylim([-.05, 1.05])
ax[0].set_xlim([-.05, 1.05])
ax[1].set_ylim([-.05, 1.05])
ax[1].set_xlim([-.05, 1.05])
ax[2].set_ylim([-.05, 1.05])
ax[2].set_xlim([-.05, 1.05])
fig.show()
```

Listing 6.6: xplots.py - The plot-file for visualizing the sampling methods

NTNU
Norwegian University of
Science and Technology