Master's Thesis 2014

Candidate: Huruy Kiflom Haile

Title: Evaluation and comparison of MPC algorithms applied to simulated processes

Telemark University College

Faculty of Technology
Kjølnes
3914 Porsgrunn
Norway

Telemark University College
Faculty of Technology
M.Sc. Programme

MASTER'S THESIS, COURSE CODE FMH606

Student: **Huruy Kiflom Haile**

Thesis title: **Evaluation and comparison of MPC algorithms applied to simulated processes**

Signature: . . . . . .

Number of pages: 94

Keywords: . . . . . .

. . . . . .

. . . . . .

Supervisor: Assoc. Prof. Finn Aakre Haugen sign.: . . . . . .

2nd Supervisor: <name> sign.: . . . . . .

Censor: <name> sign.: . . . . . .

External partner: <name> sign.: . . . . . .

Availability: <Open/Secret>

Archive approval (supervisor signature)**:** sign.: . . . . . **Date :** 04 June 2014 . .

. . . . . .

Abstract:

Model predictive controller is one of the most advanced control approaches which because of its good features like: ability to explicitly include constraints in its formulation, multivariable control, ability to look in to the future and act proactively has become popular in the industry. Although it was first introduced in the control of power production and in the petroleum industry, it has recently among others have been used in the automotive industry and medical applications like the artificial pancreas. It is thus worth making some research on the modern control system with good prospects for the future.

This thesis focuses on evaluating and comparing MPC algorithms (linear and nonlinear model predictive controller algorithms) applied to simulated processes. An air-heater model and a model for an anaerobic digestion reactor are considered as case studies. Linear and nonlinear model predictive controller algorithms are developed and used to control each of the models. Robustness, stability and computation time of each of the controller algorithms under disturbance, measurement noise and model errors are evaluated and compared.

Table of contents

Preface

This thesis is written in partial fulfillment for the award of Master of Science (M.Sc.) in Systems and Control Engineering at Telemark University College (TUC), Norway. The report deals with linear and nonlinear model predictive controller algorithms. Formulating optimization algorithms, evaluation and comparison of results from simulated processes, for both linear and nonlinear model predictive controllers are carried out.

In order to benefit from the report, the reader is expected to have some knowledge of optimization theory and model predictive controller. As MATLAB software is used for solving the optimization problems in the report, the reader is also expected to have good understanding of MATLAB.

The thesis has appendices which contain MATLAB codes.

I would like to take the opportunity to thank my supervisor Associate Professor Finn Aakre Haugen for his guidance, support and valuable suggestions.

Finally, I would also like to thank my family and specially my sister Tsega Kiflom for their continuous encouragement.

Porsgrunn, 31 May 2014

Huruy Kiflom Haile

Abbreviations

AD	Anaerobic Digestion
IAE	Integral of Absolut error
$LMPC$	Linear Model Predictive Controller
$MATLAB$	Matrix Laboratory
MPC	Model Predictive Controller
$NMPC$	Nonlinear Model Predictive Controller
QP	Quadratic programming
ODE	Ordinary Differential Equation
TUC	Telemark University College
UKF	Unscented Kalman Filter
VFA	Volatile Fatty Acids
BVS	Biodegradable Volatile Solids
VS	Volitile Solids

Nomenclature

A_f	Acidity constant [(g VFA/L)/ (g BVS/L)]
b	Retention time ratio [d/d]
B_0	Biodegradability constant[(g BVS/L)/ (g VS/L)]
e	Control error
F_{feed}	Influent or feed flow or load rate, assumed equal to effluent flow (constant volume) [L/d].
F_{meh}	Methane gas flow[L CH_4/d]
J	Objective function
k_1	Yield constant[g BVS/(g acidogens/L)]
k_2	Yield constant[g VFA/(g acidogens/L)]
k_3	Yield constant[g VFA/(g methanogens/L)]
k_5	Yield constant[L/g methanogens]
K	Heater gain in [˚C/V]
K_d	Specific death rate of acidogens[d^{-1}]
K_{dc}	Specific death rate of methanogens[d^{-1}]
K_s	Monod half-velocity constant for acidogens[g BVS/L]
K_{sc}	Monod half-velocity constant for methanogens[g VFA/L]
μ	Reaction(growth) rate of acidogens[d^{-1}]
μ_c	Reaction(growth) rate of methanogens[d^{-1}]
μ_m	Maximum reaction rate of acidogens[d^{-1}]
μ_{mc}	Maximum reaction rate of methanogens[d^{-1}]
P	Weighting matrix for control error
$\hat{P}_f$	state estimation error covariance
Q	Weighting matrix for input signal
$\hat{Q}_f$	Process noise covariance
R	Weighting matrix for rate for change of signal
$\hat{R}_f$	Measurement noise covariance

$S_{bvs_{in}}$	Concentration of BVS in influent [g BVS/L].
$S_{vfa_{in}}$	Concentration of volatile solids in influent [g VS/L].
S_{bvs}	Concentration of BVS in reactor [g BVS/L].
S_{vfa}	Concentration of VFA in reactor [g VFA/L].
$\mathrm{T_{amb}}$	Ambient temperature in[˚C]
$\mathrm{T_{const}}$	time-constant
$\mathrm{T_{heat}}$	Heater temperature[℃]
$\mathrm{T_{out}}$	Total air temperature at the tube outlet[˚C]
T_{reac}	Reactor temperature[℃]
T_s	Sampling time
τ	Time delay
u	Control signal[V]
v	Measurement noise
V	Effective reactor volume [L]
w	Process distubance
W_e	Cost(weight) factor for control error
W_u	Cost(weight) factor for input signal
$W_{\dot{u}}$	Cost(weight) factor for the rate of change of input signal
X_{acid}	Concentration of acidogens[g acidogens/L].
X_{meth}	Concentration of methanogens [g methanogens/L].
$\hat{X}$	Initial state estimate; initial aposteriori state estimate

Overview of Figures

Overview of Tables

1 Introduction

During the last three decades or so, the number of applications where control techniques based on dynamic optimization led to improved performance has significantly increased; for example, maximizing output or minimizing the energy use or raw material consumption Ferreau (2011). In order to make use of such control techniques, a mathematical model of the system to be controlled is required. The mathematical model helps for predicting future behavior of the system and for calculating optimal control actions. The optimal control actions can be calculated offline before the runtime of the process; however, some unknown disturbances require a feedback controller that repeatedly solves the optimal control problems during the runtime of the process. This repetitive optimal control refers to model predictive control (MPC).

There are important features that make MPC different from traditional control approaches. MPC gives the possibility to directly specify the control objective and the desired limitations on the process in an optimal control problem. Predictive information can also be directly included while formulating the MPC problem; this enables the MPC to react proactively to future changes. Moreover, MPC can handle processes with multiple inputs and outputs.

In many industries MPC is used to reduce energy and raw material consumption thereby saving natural resources. In the automotive industry, for example, it is not only used to save fuel and reduce emission, but is also expected to play a key role in new innovations like autonomous driving. In the health care sector, there are prospects of using MPC for optimal dose of insulin injections.

In order to enjoy the benefits of MPC, solving challenging optimal control problems in real-time is a must. Different algorithms are used to solve optimal control problems. Some of the algorithms are linear and nonlinear. Evaluation and comparison of these algorithms is the subject of this thesis.

1.1 Literature review

The MPC research area is vast and numerous reviews are available. The past present and future technology of MPC is widely discussed in M. Morari (1997). Tuning, constraints handling; both soft and hard are well discussed in Rossiter (2003) and Wang (2009). Maciejowski (2002) and Wang (2009) also give simplified ways of implementation in MATLAB and SIMULINK. Lie (2013) and Wang (2009) provide a detailed step by step MPC formulation.

As a number of references in the literature indicate using linear or nonlinear MPC has its own pros and cons. According a number of reviews that are mentioned in the following paragraphs the main contrast between linear and nonlinear MPC lies in computational speed which is the characteristic of LMPC against more realistic response which is the characteristic of NMPC.

It is easy to reduce the MPC problem for linear plants to simple quadratic or linear programs for which efficient software exists. By using a linear model and quadratic cost function, the problem becomes a highly structured convex problem and can be solved fastYuebin Yu (2013) . Thus, it is not surprising to see that majority of MPC applications are based on the linear dynamic model, mainly to take the computational advantages of linear MPC So-Ryeok Oh (2010), Bingfeng Gu (2008) .

However, Linear MPC can be inefficient for controlling nonlinear systems Andrey Alexandrovich Tyagunov (2004). If a realistic model is considered, the nonlinearities cannot be avoided and the capability of a linear model to approximate the nonlinear process diminishes and so does the quality of the linear MPC Chen (2009). Because for a linear MPC it can be difficult to directly handle nonlinear problems Yuebin Yu (2013) . In some cases, the influence of nonlinear dynamics effects is so important that the use of nonlinear model predictive control (NMPC) is unavoidable Chen (2009)

On the other hand application of MPC to nonlinear systems generally leads to nonlinear programming problems Andrey Alexandrovich Tyagunov (2004) which incur high computational cost Weiguo Xie (2011) . In fact, conventional approach with a global search solver and a detailed model simulator or direct nonlinear model predictive control incurs prohibitive computational cost Yuebin Yu (2013) . More details about optimization in nonlinear non-convex problems which can lead to local minima and the difficulties they pose to implementation in MPC can be found in Andrey Alexandrovich Tyagunov (2004) as well as in Weiguo Xie (2011).

1.2 Thesis objective

The aim of this thesis is to evaluate and compare the linear and nonlinear MPC algorithms as applied to simulated processes. A linear laboratory scale air-heater model Haugen (2012) and a nonlinear pilot reactor used for anaerobic digestion Haugen (2013) are selected for use with the linear and nonlinear MPC algorithms. The air-heater model is relatively simpler than the AD reactor model. Taking, the discussion on section 1.1 into consideration, the variation in complexity of these two models is expected to give good test on the applicapility of both LMPC and NMPC.

1.3 Thesis organization

The first part of the thesis namely chapters 2 to 5 discuss about the air heater model, formulation of LMPC, NMPC and Kalman filter along with the comparison of the results from LMPC and NMPC. Chapters 6 to 8 Discuss similar topics as applied to the AD reactor model.

Chapter 2: Linear MPC applied to the air-heater model

In chapter 2, the algorithm for the linear MPC is developed. The continuous time air heater model is discretized, a quadratic objective function with linear constraints is defined. The objective function is written in such a way that it is possible to use the built-in quadratic programming (QP) solver in MATLAB. Then, the algorithm is implemented in MATLAB and the MPC is tuned.

Chapter 3: Nonlinear MPC applied to the air-heater model

In chapter 3, the algorithm for the nonlinear MPC is developed. An objective function, similar to the one used in chapter 2 for the linear MPC is defined. However discretizing the continuous model was not necessary in this case. The 'fmincon' solver from MATLAB toolbox which that does not require discretization is used for the nonlinear optimization in this case. The algorithm is implemented in MATLAB and the MPC is tuned.

Chapter 4: Kalman filter

The temperature at the outlet of the air heater, which is the output from the process, is affected by an unknown ambient temperature. This unmeasured value of ambient temperature had to be estimated. Thus, in both the LMPC and NMPC algorithms, a Kalman filter is used. The Kalman filter is discussed in chapter 4.

Chapter 5: Comparing LMPC and NMPC results from the air-heater model

Based on some criteria like: tolerance to changes in process disturbance, measurement noise, gain, time constant, time delay, and computation time; the results from the LMPC and NMPC are compared. Effects of the various parameters are discussed in chapter 5.

Chapter 6: Linear MPC applied to the AD reactor model

The AD reactor model is a nonlinear one. In order to use a linear MPC, the model had to be linearized and discretized. Linearization and discretization, imposing equality, inequality and bounded constraints for the AD reactor model are explained in chapter 6.

Chapter 7: Nonlinear MPC applied to the AD reactor model

The nonlinear MPC applied to the AD reactor model which also uses fmincon solver from the MATLAB toolbox for solving the optimization problem is discussed in chapter 7.

Chapter 8: Comparing LMPC and NMPC results from the AD reactor model

Results obtained from simulation of the linear and nonlinear MPC algorithms used for controlling the AD reactor model are compared in chapter 8. Important points like computation time and inequality constraints are some of the points of discussion.

Chapter 9: Discussion and conclusion

The general discussion and conclusion of the thesis is given in chapter 9.

Chapter 10: Suggestions to future cork

Some suggestions for future work are stated in chapter **10.**

2 Linear MPC applied to the air-heater model

The behavior of industrial processes is nonlinear. However, in a certain limited range of operation, it can be approximated by a linear model. The linear MPC takes advantage of this approximation in order to control the nonlinear processes. Controlling a nonlinear process using linear MPC requires linear model of the process. The linear model could be found by linearizing the nonlinear model or using system identification techniques from input and output data of the process. In the sections that follow the linear MPC for the air-heater model will be designed and implemented in MATLAB.

2.1 The air-heater model

The air heater model as described by Haugen (2012) is shown in equations *(2-1)* and *(2-2)*.

$$\frac{dT_{heat}(t)}{dt} = \frac{-1}{T_{const}} T_{heat}(t) + \frac{K}{T_{const}} u(t-\tau) \qquad (2\text{-}1)$$

$$T_{out} = T_{heat} + T_{amb} \qquad (2\text{-}2)$$

Where: u is the control signal to the heater.

T_{out} is the total air temperature at the tube outlet in ˚C

T_{heat} is the additive contribution to the total temperature T_{out} due to the heater in ˚C

T_{const} is time-constant in seconds

K is heater gain in ˚C/V

τ is time-delay representing air transportation and sluggishness in the heater.

T_{amb} is the ambient temperature in ˚C

Thus, the air-heater model has one input, the control signal; one output, the total temperature at the tube outlet; three parameters: time constant, time delay, and heater gain; one state, the additive temperature contribution due to the heater; and one disturbance which is the ambient temperature. The ambient temperature is assumed to be unmeasured disturbance and treated as an augmented state so that it can be estimated using Kalman filter. A simplified model of the air-heater is shown in Figure 2.1.

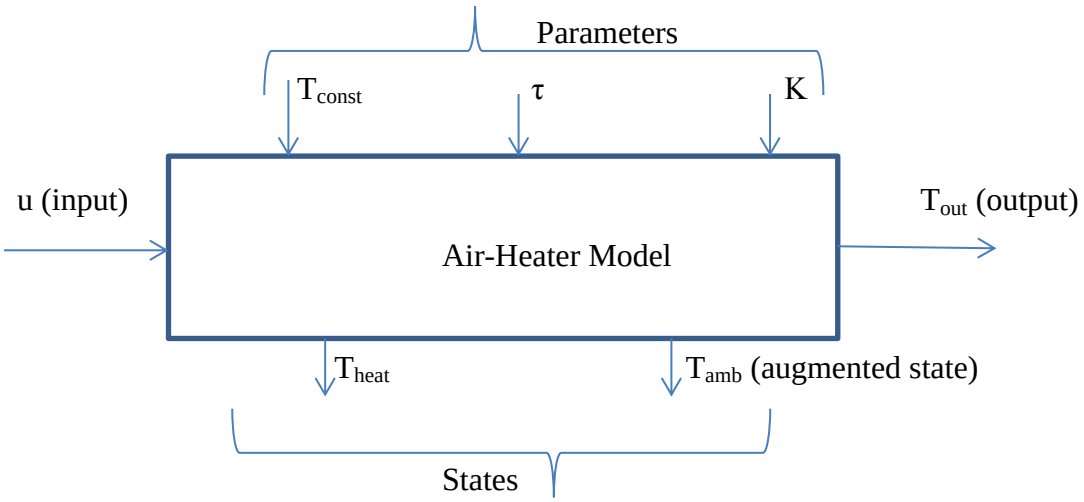


Figure 2.1 Air-Heater Model

2.2 Discretizing the model

As described in equations *(2-1)* and *(2-2)* the air heater model is linear model with time delay. The model is first discretized without the time delay; then, the time delay is incorporated in to the system using the same method used in Haugen (2014). The ambient temperature which is assumed to be unmeasured disturbance is treated as an augmented state and is estimated using Kalman filter. Assuming that the ambient temperature varies slowly its derivative is approximated to zero.

$$\frac{dT_{amb}}{dt} = 0 \qquad (2\text{-}3)$$

Thus, augmenting the ambient temperature in *(2-3)* to equation *(2-1)* as a state gives *(2-4)*.

$$\begin{bmatrix} \frac{dT_{heat}(t)}{dt} \\ \frac{dT_{amb}}{dt} \end{bmatrix} = \begin{bmatrix} {}^{1}/_{T_{const}} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} T_{heat} \\ T_{amb} \end{bmatrix} + \begin{bmatrix} \frac{K}{T_{const}} \\ 0 \end{bmatrix} u(t) \qquad (2\text{-}4)$$

Equation *(2-4)* can then be written as:

$$\dot{x} = A_c x + B_c u + w$$

$$\dot{y} = C_c x + v$$

Where: $x = \begin{bmatrix} T_{heat} \\ T_{amb} \end{bmatrix}$, w is process disturbance, v is measurement noise, and A_c, B_c, C_c are continuous time system matrices of compatible dimensions. The corresponding discrete time model can be expressed as:

$$x_{k+1} = Ax_k + Bu_k + w$$

$$y_k = Cx_k + v$$

Where:

$x_k \in \mathbb{R}^{n_x}$ is the state vector which represents the heater temperature in this case, $u_k \in \mathbb{R}^{n_u}$ is the control input vector which represents the sequence of actual control signals in volts, $y_k \in \mathbb{R}^{n_y}$ is the output vector which represents the total temperature at the tube outlet. A, B and C are discrete time system matrices of appropriate dimensions, and x_0 is the initial temperature of the heater. w and v represent process disturbance vector and measurement noise vector respectively. Both w and v are assumed to be constants or slowly varying.

The aim is to design an MPC algorithm which is not affected by the measurement noise or the process disturbance. In order to achieve that we use the deviation variables δx_k , δy_k and δu_k in the MPC algorithm Ruscio (2013).

Assuming that $x_k = x_k - x_{k-1}$, $\delta u_k = u_k - u_{k-1}$ and evaluating the deviations in state variables $x_{k+1} - x_k$ gives:

$$x_{k+1} - x_k = Ax_k + Bu_k + w - (Ax_{k-1} + Bu_{k-1} + w)$$

$$\underbrace{x_{k+1} - x_k}_{\delta x_{k+1}} = A(\underbrace{x_k - x_{k-1}}_{\delta x_k}) + B(\underbrace{u_k - u_{k-1}}_{\delta u_k})$$

Thus the deviation in state variables can be expressed as shown in *(2-5)*.

$$\delta x_{k+1} = A\delta x_k + B\delta u_k \qquad (2\text{-}5)$$

Similarly, evaluating the deviation in output variables $y_k - y_{k-1}$ gives:

$$y_k - y_{k-1} = Cx_k + v - (Cx_{k-1} + v)$$

$$\underbrace{y_k - y_{k-1}}_{\delta y_k} = C(\underbrace{x_k - x_{k-1}}_{\delta x_k})$$

This expression for δy_k is the same as the expression in equation *(2-6)*.

$$\delta y_k = C\delta x_k \qquad (2\text{-}6)$$

In addition to the discrete model shown in equations *(2-5)* and *(2-6)*, the control error and the rate of change of the input signal which also are part of the equality constraints are shown in equations *(2-7)* and *(2-8)*. The objective is to produce output y_k that follows the reference values r_k which means the control error e_k should be as close to zero as possible. The control error is weighted in the performance index. The rate of change of the control signal should neither be very slow nor abrupt one. In order to achieve a reasonable rate of the change of the control signal the rate of change of the control signal is also weighted in the performance index.

$$e_k = r_k - y_k \qquad (2\text{-}7)$$

$$\delta u_k = u_k - u_{k-1} \qquad (2\text{-}8)$$

2.3 Control objective

The cost function J that reflects the control objective is defined as in *(2-9)*

$$\mathrm{J} = \frac{1}{2}\sum_{i=1}^{N}[e_i^T P e_i + u_{i-1}^T Q u_{i-1} + \delta u_{i-1}^T R \delta u_{i-1}] \qquad (2\text{-}9)$$

Where the first term ($e_i^T P e_i$) is linked to the objective of minimizing the errors between the predicted output and the set-point, the second term ($u_{i-1}^T Q u_{i-1}$) reflects the consideration given to the amplitude of the input; the third term ($\delta u_{i-1}^T R \delta u_{i-1}$) corresponds to the rate of change of the input.

The error or the difference between the output and reference signal, ‘e’, the amplitude of the input signal, ‘u’, the rate of change of the input signal, ‘δu’ are to be optimized. P, Q and R are corresponding weighting matrices. The aim is to minimize the cost function J subject to the equality constraints specified in equations *(2-5)* to *(2-8)* while keeping the control signal in the range of 0 to 5 volts. The range of the input signal is determined by the physical limitations of the device, constraints that are governed by the physical limitations of the system are hard constraints. Hard constraints are constraints that cannot be violated. In this case it is not possible to get a supply below zero volts or above five volts, for instance. Thus, the controller should always satisfy this requirement.

2.4 Quadratic programming (QP)

In order to utilize the 'QP' solver from the MATLAB optimization toolbox, the optimization problem formulated in section 2.3 is rewritten as a quadratic programming problem. Quadratic programming (QP) problem involves minimizing or maximizing an objective function like the one shown in *(2-9)*, subject to bounds, linear equality and inequality constraints Mathworks (2014b). It can be described as the mathematical problem of finding a vector z that minimizes a quadratic function J as in *(2-10)*.

$$J = \frac{1}{2} z^T H z + c^T z \qquad (2\text{-}10)$$

Subject to:

$$A_e x \leq b_e \ (equality\ constraints)$$

$$A_i x \leq b_i \ (inequality\ constraints)$$

$$lb \leq x \leq ub (bounded\ constriants)$$

H is a constant matrix which is referred to as the Hessian matrix. It is assumed that H is symmetric, positive definite (i.e. H > 0)Ruscio (2001) . c is a vector which is independent of the unknown present and future inputs. c is defined by the model, a sequence of known past inputs and outputs.

Assuming that the vectors $u^T, \delta u^T, e^T, y^T$ and x^T are given by *(2-11)*.

$$\begin{cases} u^T = (u_1^T, \dots\dots, u_N^T), & u \in \mathbb{R}^{N.n_u \times 1} \\ \delta u^T = (\delta u_1^T, \dots\dots, \delta u_N^T), & \delta u \in \mathbb{R}^{N.n_u \times 1} \\ e^T = (e_1^T, \dots\dots, e_N^T), & e \in \mathbb{R}^{N.n_y \times 1} \\ y^T = (y_1^T, \dots\dots, y_N^T), & y \in \mathbb{R}^{N.n_y \times 1} \\ x^T = (x_1^T, \dots\dots, x_N^T), & x \in \mathbb{R}^{N.n_x \times 1} \end{cases} \qquad (2\text{-}11)$$

The cost function in *(2-9)* needs to be rewritten in a quadratic program form as in *(2-10)* as a function of the column vectors specified in (2-11). In order to do that, the unknown vectors are stacked in z as in *(2-12)*.

$$z^T = (u^T, \delta u^T, e^T, y^T, x^T), \quad z \in \mathbb{R}^{N.(2n_u + .2n_y + .n_x)} \qquad (2\text{-}12)$$

It is further assumed that $u_0, x_1\ and\ r_1 \dots\dots, r_N$ are known and then the quadratic program representation of the performance index interms of the unknown vectors according to Lie (2013) is calculated as in *(2-13)*.

$$J = \frac{1}{2} \underbrace{\begin{bmatrix} u \\ \delta u \\ e \\ y \\ x \end{bmatrix}^T}_{z^T} \underbrace{\begin{bmatrix} H_{11} & 0 & 0 & 0 & 0 \\ 0 & H_{22} & 0 & 0 & 0 \\ 0 & 0 & H_{33} & 0 & 0 \\ 0 & 0 & 0 & H_{44} & 0 \\ 0 & 0 & 0 & 0 & H_{55} \end{bmatrix}}_{H} \underbrace{\begin{bmatrix} u \\ \delta u \\ e \\ y \\ x \end{bmatrix}}_{z} + \underbrace{\begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \end{bmatrix}^T}_{C^T} \underbrace{\begin{bmatrix} u \\ \delta u \\ e \\ y \\ x \end{bmatrix}}_{z} \quad (2\text{-}13)$$

Where:

$$u^T H_{11} u = \begin{bmatrix} u_1 \\ \vdots \\ u_N \end{bmatrix}^T \begin{bmatrix} Q & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & Q \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_N \end{bmatrix} \qquad \Rightarrow H_{11} = I_N \otimes Q$$

$$\delta u^T H_{22} \delta u = \begin{bmatrix} \delta u_1 \\ \vdots \\ \delta u_N \end{bmatrix}^T \begin{bmatrix} R & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & R \end{bmatrix} \begin{bmatrix} \delta u_1 \\ \vdots \\ \delta u_N \end{bmatrix} \qquad \Rightarrow H_{22} = I_N \otimes R \qquad (2\text{-}14)$$

$$e^T H_{33} e = \begin{bmatrix} e_1 \\ \vdots \\ e_N \end{bmatrix}^T \begin{bmatrix} P & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & P \end{bmatrix} \begin{bmatrix} e_1 \\ \vdots \\ e_N \end{bmatrix} \qquad \Rightarrow H_{33} = I_N \otimes P$$

Here the variables x *and* y are not contributing to the performance index J. Therefore, the values of H_{44} and H_{55} are zeros.

$$\begin{cases} H_{44} = I_N \otimes 0_{n_y} \Rightarrow H_{44} = 0_{N.n_y} \\ H_{55} = I_N \otimes 0_{n_x} \Rightarrow H_{55} = 0_{N.n_x} \end{cases} \qquad (2\text{-}15)$$

There is no any linear term in the performance index. The absence of any linear term in z in the performance index implies that $c = 0_{n_z \times 1}$.

The symbol $\otimes$ represents the Kronecker product. [1]

2.5 Constraints

To complete the representation of the QP, it is required to formulate the constraints. The constraints can be bounded constraints (upper and/or lower limits), equality constraints and inequality constraints.

---

[1] If A is an $m \times n$ matrix and B is a $p \times q$ matrix, then the Kronecker product A $\otimes$ B is the $mp \times nq$ block matrix:

$$A \otimes B = \begin{bmatrix} a_{11}b_{11} & \dots & a_{1n}b_{1q} \\ \vdots & \ddots & \vdots \\ a_{m1}b_{p1} & \dots & a_{mn}b_{pq} \end{bmatrix}$$

The equality constraints from equations *(2-5)* to *(2-8)* can be rewritten as shown in *(2-16)*.

$$A_e z = b_e \qquad (2\text{-}16)$$

Where:

$$A_e = \begin{pmatrix} A_{e,11} & A_{e,12} & A_{e,13} & A_{e,14} & A_{e,15} \\ A_{e,21} & A_{e,22} & A_{e,23} & A_{e,24} & A_{e,25} \\ A_{e,31} & A_{e,32} & A_{e,33} & A_{e,34} & A_{e,35} \\ A_{e,41} & A_{e,42} & A_{e,43} & A_{e,44} & A_{e,45} \end{pmatrix} \text{ and } b_e = \begin{pmatrix} b_{e,1} \\ b_{e,2} \\ b_{e,3} \\ b_{e,4} \end{pmatrix}$$

Each of the components of matrix A_e and vector b_e are described in equations *(2-17)* to *(2-24)*.

The measurement equation *(2-6)* can be expanded as shown in *(2-17)* where the unknowns are placed on the left hand side of the equality sign while the known values are kept on the right hand side of the equality sign.

$$\left.\begin{matrix} \delta y_1 = C\delta x_1 \\ \vdots \\ \delta y_N = C\delta x_N \end{matrix}\right\} \Rightarrow \quad \begin{matrix} \delta y_1 = C\delta x_1 \\ \delta y_2 - C\delta x_2 = 0 \\ \vdots \\ \delta y_N - C\delta x_N = 0 \end{matrix} \qquad (2\text{-}17)$$

$$\begin{cases} A_{e,11} = -I_N \otimes D \\ A_{e,12} = 0_{N.n_y \times N.n_y} \\ A_{e,13} = 0_{N.n_y \times N.n_y} \\ A_{e,14} = I_{N.ny} \\ A_{e,15} = -I_{N,-1} \otimes C \end{cases} \qquad (2\text{-}18)$$

$$b_{e,1} = \begin{bmatrix} C\delta x_1 & 0_1 & \cdots & 0_{(N-1).n_y} \end{bmatrix}^T$$

Similarly, from the state equation shown in (2-5) we get:

$$\left.\begin{matrix} \delta x_2 = A\delta x_1 + B_c\delta u_1 + B_d\delta w_1 \\ \delta x_3 = A\delta x_2 + B_c\delta u_2 + B_d\delta w_2 \\ \vdots \\ \delta x_N = A\delta x_{N-1} + B_c\delta u_{N-1} + B_d\delta w_{N-1} \end{matrix}\right\} \Rightarrow \qquad (2\text{-}19)$$

$$\Rightarrow \begin{cases} -B\delta u_1 + \delta x_2 = A\delta x_1 + B_d\delta w_1 \\ -B\delta u_2 - A\delta x_2 + \delta x_3 = B_d\delta w_2 \\ \vdots \\ -B\delta u_{N-1} - A\delta x_{N-1} + \delta x_N = B_d\delta w_{N-1} \end{cases}$$

$$\begin{cases} A_{e,21} = -I_N \otimes B \\ A_{e,22} = 0_{N.n_x \times N.n_u} \\ A_{e,23} = 0_{N.n_x \times N.n_y} \\ A_{e,24} = 0_{N.n_x \times N.n_y} \\ A_{e,25} = I_N \otimes I_{nx} - I_{N,-1} \otimes A = I_{N.nx} - I_{N,-1} \otimes A \end{cases} \quad (2\text{-}20)$$

$$b_{e,2} = [A\delta x_1 + B_d \delta w_1 \quad B_d \delta w_2 \quad \dots \quad B_d \delta w_{N-1}]^T$$

The equation in *(2-7)* representing the control error can be expressed as in (2-21) and *(2-22)*.

$$\left.\begin{matrix} e_1 = r_1 - y_1 \\ \vdots \\ e_N = r_N - y_N \end{matrix}\right\} \Rightarrow \quad \begin{matrix} e_1 + y_1 = r_1 \\ \vdots \\ e_N + y_N = r_N \end{matrix} \quad (2\text{-}21)$$

$$\begin{cases} A_{e,31} = 0_{N.n_y \times N.n_u} \\ A_{e,32} = 0_{N.n_y \times N.n_u} \\ A_{e,33} = I_{N.ny} \\ A_{e,34} = I_{N.ny} \\ A_{e,35} = 0_{N.n_y \times N.n_x} \end{cases} \quad (2\text{-}22)$$

$$b_{e,3} = [r_1 \quad r_2 \quad \dots \quad \dots \quad r_N]^T$$

Finally, equation *(2-8)* which represents the rate of change of the control signal can be rewritten as in *(2-23)*.

$$\left.\begin{matrix} \delta u_1 = u_1 - u_0 \\ \delta u_2 = u_2 - u_1 \\ \vdots \\ \delta u_N = u_N - u_{N-1} \end{matrix}\right\} \Rightarrow \quad \begin{matrix} -u_1 + \delta u_1 = -u_0 \\ u_1 - u_2 + \delta u_2 = 0 \\ \vdots \\ \vdots \\ u_{N-1} - u_N + \delta u_N = 0 \end{matrix} \quad (2\text{-}23)$$

$$\begin{matrix} \mathrm{A_{e,41}} = \mathrm{I_{N,-1}} \otimes \mathrm{I_{nu}} - \mathrm{I_N} \otimes \mathrm{I_{nu}} = \mathrm{I_{N,-1}} \otimes \mathrm{I_n} - \mathrm{I_{N.nu}} \\ \mathrm{A_{e,42}} = \mathrm{I_N} \otimes \mathrm{I_{nu}} = \mathrm{I_{N.nu}} \\ A_{e,43} = 0_{N.n_u \times N.n_y} \\ A_{e,44} = 0_{N.n_u \times N.n_y} \\ A_{e.45} = 0_{N.n_u \times N.n_x} \\ b_{e,4} = [-u_0 \quad 0 \quad \dots \quad \dots \quad 0]^T \end{matrix} \quad (2\text{-}24)$$

In this case there are no any inequality constraints. However, there is one bounded constraint; the input signal should be limited between 0 and 5 volts. All the other variables are assumed to vary from minus infinity to infinity as shown in *(2-25)*.

$$\begin{matrix} 0 \leq u \leq 5 \\ -\infty \leq \delta u \leq \infty \\ -\infty \leq e \leq \infty \\ -\infty \leq y \leq \infty \\ -\infty \leq x \leq \infty \end{matrix} \qquad (2\text{-}25)$$

2.6 Implementation in MATLAB

The idea described in sections 2.1 to 2.5 are implemented in MATLAB. See the code on Appendix 2. The continuous time matrices of the air heater model shown in *(2-1)* and *(2-2)* are used in the state space form in MATLAB. The values for the heater gain, time constant, and time delay are 3.5, 19 ˚C and 3 seconds respectively and are taken from Haugen (2012). The continuous time model is discretized using 'c2dm' function in MATLAB at a sampling time of one second. In this linear MPC the optimization algorithm used is QP and ambient temperature is estimated using unscented Kalman filter discussed in chapter 4.

2.7 LMPC Tuning

Tuning the LMPC was not easy. It was done by trial and error method. However, some sort of useful trend was observed while retuning the LMPC repeatedly. It is important to keep the ration of the weighting matrix for the rate of change of the input signal(R) to the weighting matrix for the control error small, and then adjust the weighting matrix for the amplitude of the control signal (Chen), between the values of R and P. The ration of R to P may vary from one problem to the other. In this case a ratio of 1:1000 gave good results. It is easier to start by setting Q to zero and keeping R/P very small. Then, the output can be tuned to track the set point by just varying Q between the values of P and R without changing the values of P and R.

An example is shown below to illustrate how the ration of R to P and changing Q affects the value of the output temperature in relation to the set point temperature. In Figure 2.2 Q is set to zero while the ratio of R to P is set to 1:1000. It can be seen that the output has steady state error and the output temperature is greater than the set point. Increasing the weighting matrix of the control signal is supposed to reduce the output temperature. This is tried in Figure 2.3 where Q is raised to 900 while R and P are kept at their original values (1 and 1000 respectively). Increasing Q has lowered the output below the set point which again has resulted in a steady state error again. However, this time the value of the output is lower than the reference value as shown in Figure 2.3. After some adjustments of Q a value of 777 gave a steady state error of zero, see Figure 2.4. In all of the cases the value of the ambient temperature was kept at 19 ˚C. This method of tuning was found useful as it has only two important steps to remember.

i. Keep the ration R to P small.

ii. Adjust Q between the values of R and P until you get satisfactory result

During this simulation, the prediction horizon is set to 10; the discretization time is set to 1 second. The weighting matrices for the amplitude of the input signal, the rate of change of the input signal and control error are set to 777, 1 and 1000 respectively. These values gave the best result and will be used in all tests of the air-heater model that follow.

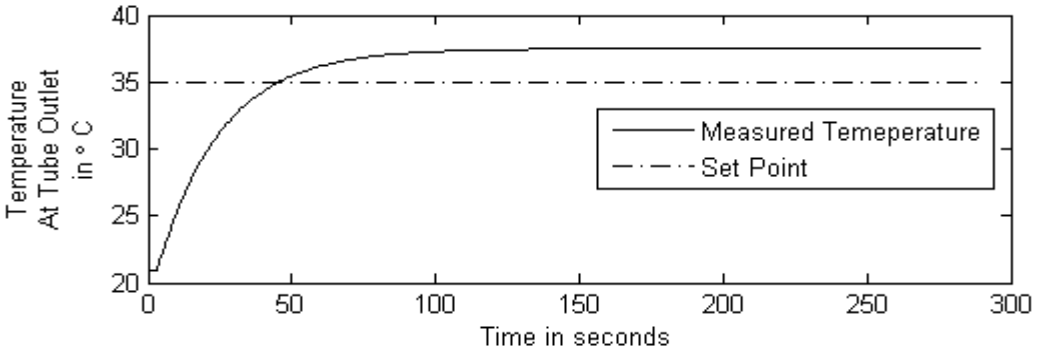


Figure 2.2 LMPC tuning, R:P =1:1000 , Q =0 , output > set point

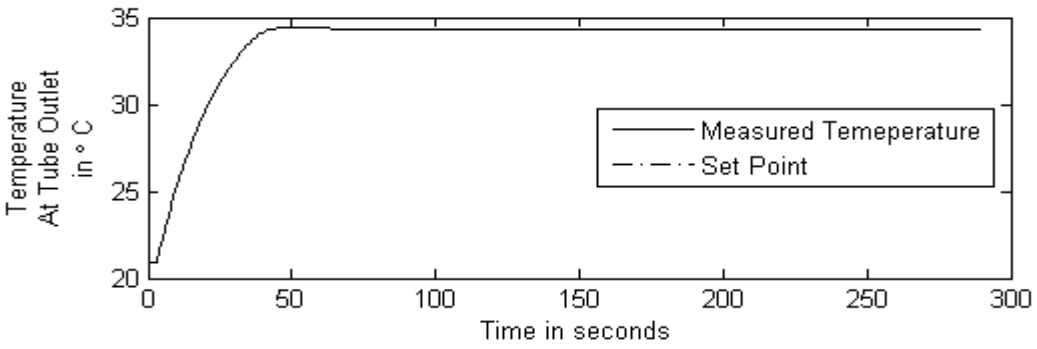


Figure 2.3 LMPC tuning, R:P =1:1000 , Q =900 , output < set point

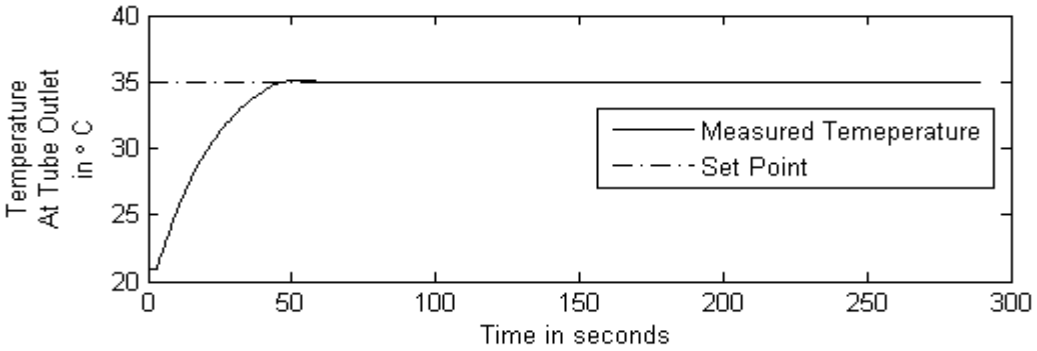


Figure 2.4 LMPC tuning, R:P =1:1000 , Q =777 , output = set point

3 Nonlinear MPC applied to the air-heater model

In the nonlinear MPC the built-in MATLAB solver 'fmincon' is used in order to solve the optimization. To use this solver, two functions need to be provided: the objective function and a function that calculates the constraints. Moreover, the initial state needs to be specified. When using the nonlinear solver, linearization and discretization are not required. The continuous time air-heater model described in equations *(2-1)* and *(2-2)* is also used in this chapter. Here, discretizing the model is not required.

3.1 Control objective

The following control objective is defined

$$J = \int_{t}^{t+\tau_h} (W_e e^2(t) + W_u u^2(t) + W_{\dot{u}} \dot{u}^2(t))dt \qquad (3\text{-}1)$$

Such that: $u_{min} \leq u(t) \leq u_{max}$

Where: e represents the control error

u represents the amplitude of the control signal

$\dot{u}$ represents the rate of change of the control signal

t is the present time

τ_h is the prediction horizon

$W_e, W_u \; and \; W_{\dot{u}}$ are weights for $e, u \; and \; \dot{u}$ respectively.

The forward (explicit) Euler method is used for solving the ODEs in the objective function. The forward Euler method is simple ODEs integrator Tallent (2012). The simulation that uses the forward Euler method only depends on the past values of state variables and state derivatives.

Let:

t_k be the time at k^{th} time-step

J_k be the computed solution at the k^{th} time-step

$T_s = t_k - t_{k-1}$ be the step size

Then, the forward Euler method can be summarized as:

$J_{k+1} = J_k + T_s(W_e e^2 + W_u u^2 + W_{\dot{u}} \dot{u}^2)_k$

The MATLAB code for the objective function is in Appendix 5.

3.2 Implementation in MATLAB

Implementing the nonlinear MPC in MATLAB is relatively easy. There is no need to write all the linear equations using the Kronecker product as in the linear MPC. Equations *(2-1)* and *(2-2)* are directly used with the 'fmincon' solver. The MATLAB code can be found in Appendix 3

3.3 NMPC Tuning

The NMPC is tuned in the same way as the LMPC. The ratio $W_{\dot{u}}$ to W_e (see section 3.1) is kept low (1:1000) and W_u is varied between the values of $W_{\dot{u}}$ and W_e. Results which are similar to the results of LMPC are obtained and are illustrated in Figure 3.1, Figure 3.2, and Figure 3.3. Although there is no big difference between the three figures as in the LMPC case, it can be observed that Figure 3.3 gave better result than the other two. It was observed that the NMPC gave quite good results to a wider range of values of W_u unlike the LMPC. The LMPC requires fine tuning of the weighting matrices in order to give good results. The ambient temperature is kept at 19˚C in the same way as in the LMPC tuning. In section 5.2, effect of changing ambient temperature on both NMPC and LMPC will be analyzed.

During this simulation, the prediction horizon is set to 10; the discretization time is set to 1 second. The weighting matrices for the amplitude of the input signal, the rate of change of the input signal and control error are set to 777, 1 and 1000 respectively. These values gave the best result and will be used in all air-heater model related tests that follow.

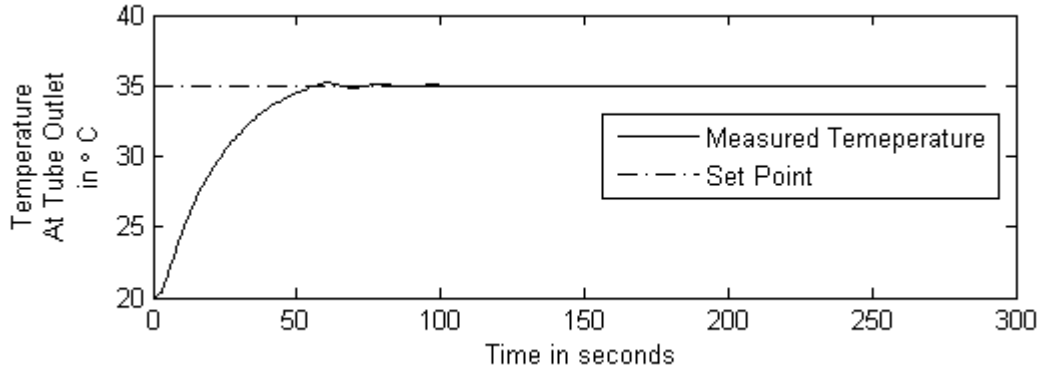


Figure 3.1 NMPC tuning, $W_{\dot{u}}$: W_e=1:1000 , Q =0

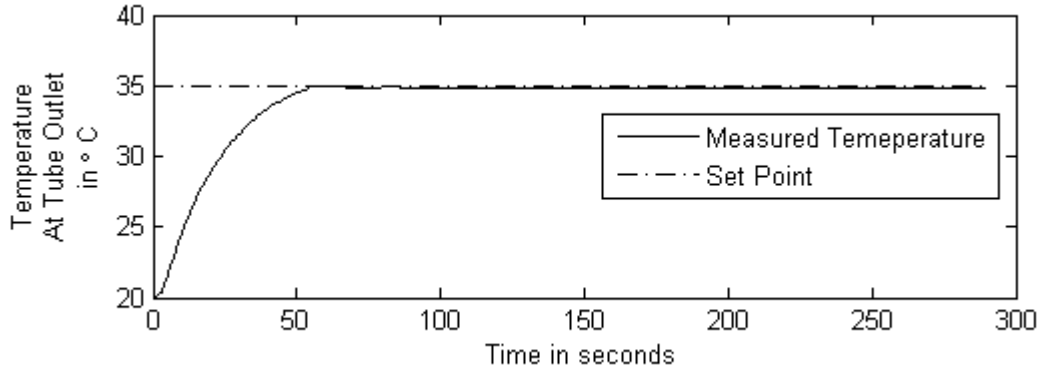


Figure 3.2 NMPC tuning, $W_{\dot{u}}$: W_e=1:1000 , Q =900

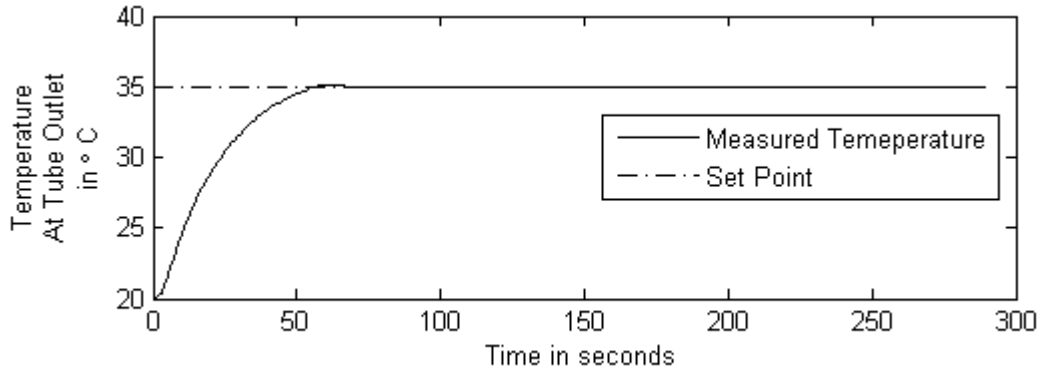


Figure 3.3 NMPC tuning, $W_{\dot{u}}$: W_e=1:1000 , Q =777t

3.4 Summary of MATLAB files used in the air-heater model

The following MATLAB functions are used in the Air-heater model:

1. Air_Heater_LMPC.m (Appendix 2)
   This function represents the LMPC for the Air-heater model and is invoked by the Air_Heater_LMPC_and_NMPC_plot.m
2. Air_Heater_NMPC.m (Appendix 3)
   This function represents the NMPC for the Air-heater model and is invoked by the Air_Heater_LMPC_and_NMPC_plot.m
3. Air_Heater_objective.m (Appendix 5)
   This function represents the objective function for the NMPC of the Air-heater model and is invoked by Air_Heater_NMPC.m
4. Air_Heater_constraint.m (Appendix 10)
   This function represents the constraint function for the NMPC of the Air-heater model and is invoked by Air_Heater_NMPC.m . This function has only empty matrices for the constraints it is just added so that the fmincon solver in Air_Heater_NMPC.m works properly.
5. Air_Heater_ukf.m (Appendix 4)
   This function represents the unscented Kalman filter for the NMPC of the Air-heater model and is invoked by Air_Heater_NMPC.m and Air_Heater_LMPC.m
6. Air_Heater_LMPC_and_NMPC_plot.m (Appendix 12)
   This script calls the Air_Heater_LMPC.m and the Air_Heater_NMPC.m and plots the results from the LMPC and NMPC functions.

4 Kalman filter

Kalman filter is used in order to estimate the states and disturbances in the air-heater model and the AD reactor model. The unscented Kalman filter (UKF) is used in this report. According to Simon (2006), the UKF does not need linearization and is less erroneous than the extended Kalman filter (EKF) which is the most widely used state estimation algorithm for nonlinear systems. As described in section 2.6 the parameters for the air heater model, namely: T_{const}, τ and K are 23 seconds, 3 seconds and 3.5 respectively. For the values of the parameters of the AD reactor model see Table 6-2.

4.1 Tuning the Kalman Filter

The Kalman filter is tuned using the parameters like the initial aposteriori state estimate, $\hat{X}$; the initial state estimation error covariance, $\hat{P}_f$; measurement noise covariance , $\hat{R}_f$ and process noise covariance, $\hat{Q}_f$. In both the air-heater model and the AD reactor model, the Kalman filter was tuned by trial and error.

In the air-heater model, the initial aposterior state estimate, $\hat{X}$ is set to $\begin{bmatrix} 1 \\ 19 \end{bmatrix}$; initial state estimation error covariance, $\hat{P}_f$ is $\begin{bmatrix} 100 & 0 \\ 0 & 36100 \end{bmatrix}$; measurement covariance, $\hat{R}_f$ is scalar as we have one measured value (T_{out}) and is 2.25 ; and process noise covariance, $\hat{Q}_f$ is $\begin{bmatrix} 25 & 0 \\ 0 & 9025 \end{bmatrix}$. The MATLAB code for the Kalman filter used in the air heater model is in Appendix 4.

In the AD reactor model, the same tuning parameters as in Finn Haugen (2014) are used. The initial aposterior state estimate $\hat{X}$, is set to $[4.14 \quad 0.8 \quad 1.8 \quad 0.39 \quad 30.2]^T$; initial state estimation error covariance, $\hat{P}_f$ is set to diagonal matrix as : $diag[(0.01\hat{X})^2]$; measurement covariance , $\hat{R}_f$ is normally a diagonal matrix. As we have only one measurement (F_{meth}), $\hat{R}_f$ is reduce to a scalar and is set to 1.44 (representative value from real time series); and process noise covariance, $\hat{Q}_f$ is set to $diag(k_Q\hat{X})^2$, where $k_Q = 0.0005 * diag([10\ 1\ 1\ 1\ 10])$ The MATLAB code for the Kalman filter used in the AD reactor model is in Appendix 11.

5 Comparing LMPC and NMPC results from the air-heater model

A number of criteria can be considered to compare the performance of a controller algorithm. Some of the criteria can be: robustness, stability and sensitivity. In this chapter, analysis of these criteria in connection with the air-heater shall be carried out. Some data and plots from the simulation results are also provided.

5.1 Introduction

In general, the term sensitivity refers to the effect that a change in one variable has on another variable Dale E. Seborg (2004). With this respect, the effect of some variables in the air-heater model, for example, effect of disturbance on output, effect of measurement noise on output or execution time, etc. shall be discussed.

If a control system designed using a model is to function properly, it should not be disproportionately sensitive to small changes in the process or to inaccuracies in the process model. A control system that satisfies this requirement is said to be robust or insensitive Dale E. Seborg (2004). This part will be analyzed by deliberately using different parameters in the process model and in the controller. For example, what happens when the heater gain, time constant or time delay of the process model or the real process are different from the values used in the controller? It is answered in the following sections.

Stability is also another important criteria to consider, the ratio of the amplitudes of subsequent peaks in the same direction (due to a step change of the disturbance or a step change of the set point in the control loop) is approximately ¼ Haugen (2010). Step changes are applied in both the ambient temperature (disturbance) and set point. The stability is quantified using the ¼ decay ratio.

All in all the bases for comparison of LMPC and NMPC used in the air-heater model can be summarized as:

i. Tolerance to changes in ambient temperature(section 5.2)
ii. Tolerance to measurement noise (section 5.3)
iii. Tolerance to model error (section 5.4)
    a. Sensitivity to changes in Heater gain
    b. Sensitivity to changes in time constant
    c. Sensitivity to changes in time delay
iv. Execution time(section 0)
v. Integral of absolute error, (IAE) (section 5.6)

5.2 Tolerance to changes in ambient temperature

The ambient temperature is changed from 18˚C to 35˚C while plotting the simulation results and recording the execution time. In most of the cases a bigger overshoot is observed in the NMPC than in the LMPC during the transient state. The ambient temperature was kept at 19˚C when tuning LMPC (section 2.7) and NMPC (section 3.3). Now, there is a slight difference in the output especially in the nonlinear one when the ambient temperature is changed to 29˚C, see Figure 5.1. From the simulations conducted, both the LMPC and NMPC have shown good tolerance to changing ambient temperature. However, a slightly bigger overshoot (red line for NMPC) compared to smother one (blue line for LMPC) is shown in Figure 5.1. Apart from the slightly bigger overshoot in the NMPC, both LMPC and NMPC outputs track the reference very well, while the ambient temperature changes.

Figure 5.1 is a representative to a number of similar simulations that are conducted and gave similar results. From Figure 5.1 it can be seen that ambient temperature is increased from 29˚C to 31˚C at time t = 150 seconds and is dropped back to 29˚C at time t = 400 seconds. The figure shows that the LMPC is less oscillating and has slightly shorter settling time than then NMPC when the same perturbation is applied to both of them.

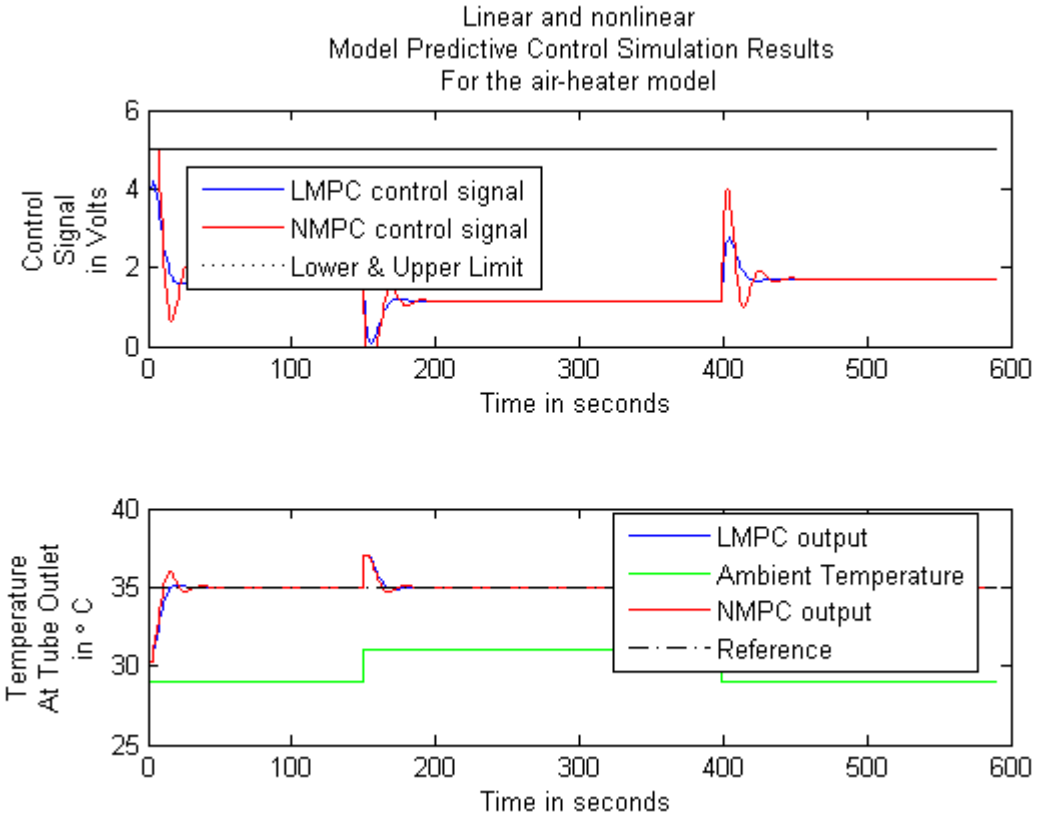
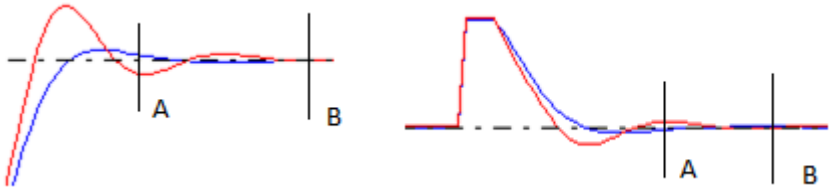


Figure 5.1 LMPC and NMPC simulation result when ambient temperature changes



The transient overshoot after about 10 seconds and the overshoot due to increased disturbance 150 seconds are magnified as shown above. If we see successive peaks that are pointing down at points A and B in the first sketch or successive peaks that are point up at points A and B in the second sketch, the decay ratio becomes zero in both cases. Which means the system is stable.

5.3 Tolerance to measurement noise

In the simulations shown in Figure 5.1, the measurement noise was set to zero. Here a random measurement noise with amplitude of 2°C is added to the output in both the LMPC and NMPC in order to test their sensitivity to measurement noise. The simulations conducted show that the LMPC is slightly less sensitive to measurement noise than the NMPC, although both have quite similar sensitivity to measurement noise. Figure 5.2 may help to visualize the slight difference in the results in the presence of the same amount of noise in both the LMPC and NMPC. The NMPC shows from 33.2°C to 37.07°C which means a maximum peak to peak variation of 3.81°C while the LMPC output varies from 33.45°C to 36.31°C which implies a maximum peak to peak variation of 2.86°C.

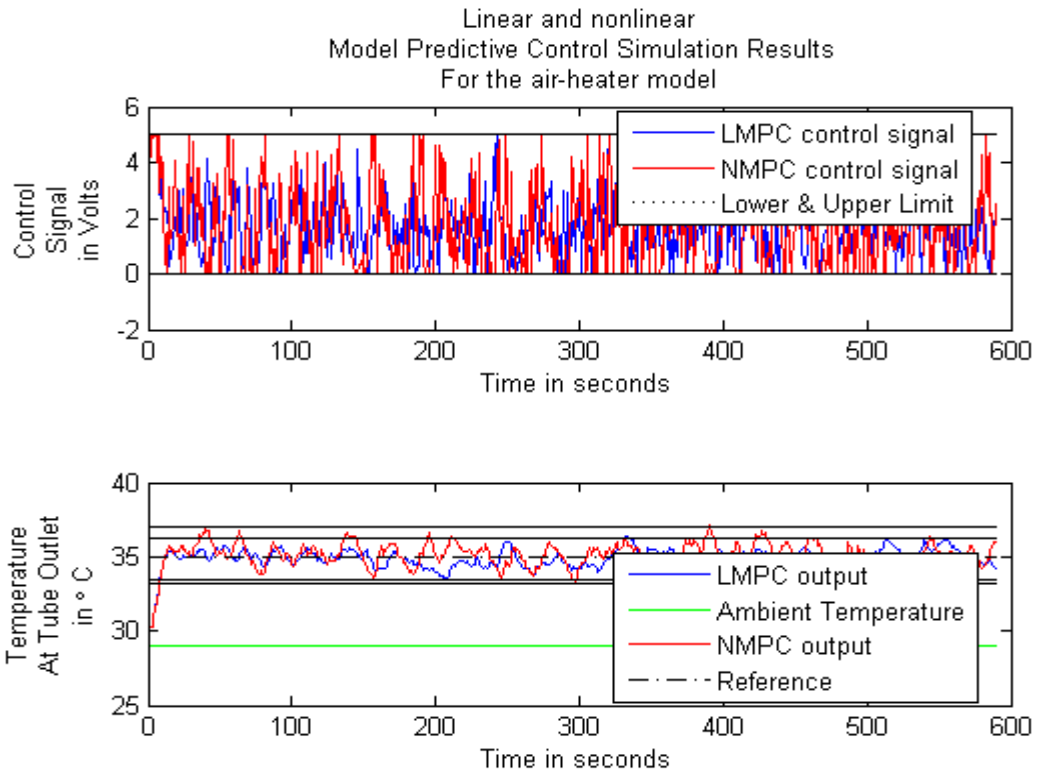


Figure 5.2 LMPC and NMPC simulation result when noise is added

5.4 Tolerance to model error

However accurately analyzed, the model can never be a perfect representation of the real process. There will always be model errors. Thus, it becomes important to evaluate the capability of the controller in situations where the actual process and the model parameters mismatch. In this section the robustness of both LMPC and NMPC against model errors is analyzed. The controller sensitivity to changes in model errors is analyzed by changing the process parameters: heater gain, time constant and time delay. In order to see the effect of the parameter under consideration, all the other parameters are kept at their nominal values while the one under test is subjected to change.

5.4.1 Tolerance to changes in heater gain

The heater gain was allowed to change from 75% to 200% of its nominal value while plotting the simulation results and recording the execution time at each change. The LMPC showed more tolerance to changes in the heater gain than the NMPC. The plot in Figure 5.3 shows results from the LMPC and NMPC when the heater gain is raised to 175% of its nominal value. The figure shows that the LMPC is more stable and less sensitive to changes in the heater gain than the NMPC in this case.

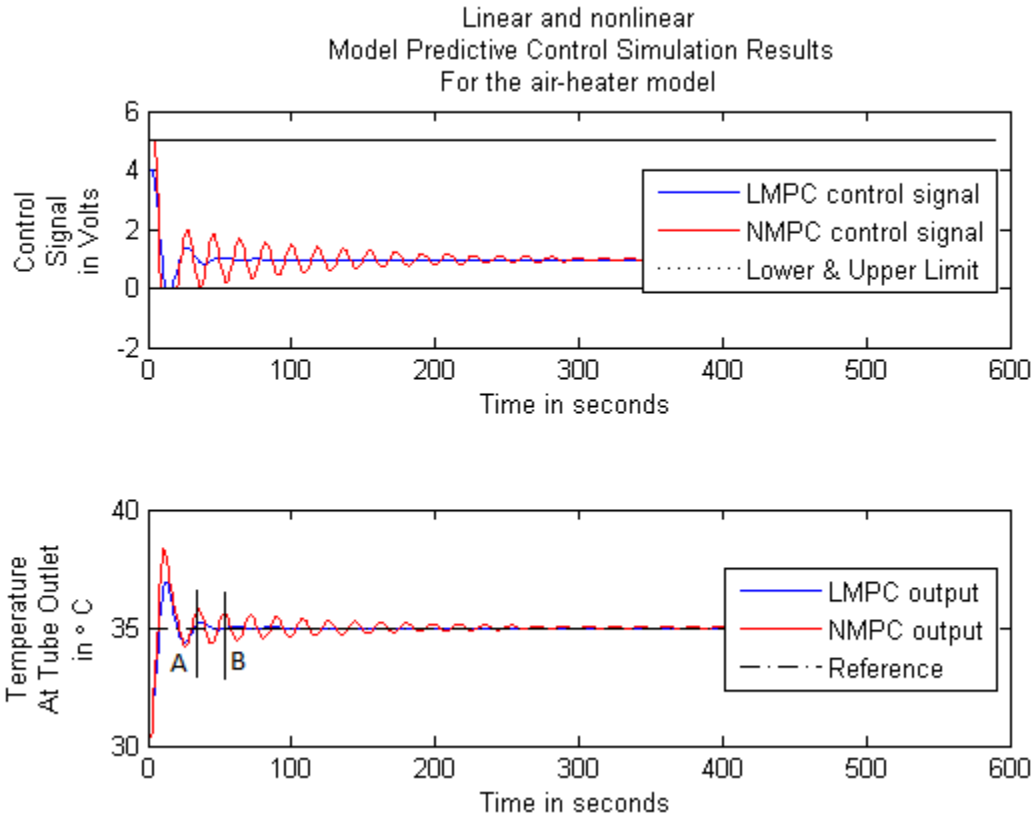


Figure 5.3 NMPC and LMPC simulation result when heater gain is 175% of its nominal value

If we consider peaks at points A and B in Figure 5.3 for example, the decay ratio can be summarized as in Table 5-1

Table 5-1 decay ration in NMPC and LMPC when heater gain is 175% of its nominal value

Controller	Decay Ratio	Remarks
LMPC	$\frac{peak\ atB}{peak\ at\ A} = \frac{0}{0.45} = 0$	Stable
NMPC	$\frac{peak\ atB}{peak\ at\ A} = \frac{0.64}{0.75} = 0.8533 > 0.25$	Unstable

5.4.2Tolerance to changes in time constant

Changing the time constant did not affect the controllers in general compared to the changes in the other parameters. The LMPC became unstable when the time constant reduced to and below 25% of its nominal value while the NMPC became unstable when the time constant is reduced to and below 50% of its nominal value. As the time constant increases the output response takes more time to come to a steady state value in both LMPC and NMPC. The simulation results shown in Figure 5.4 are taken when the time constant is increased to 150% of its nominal value.

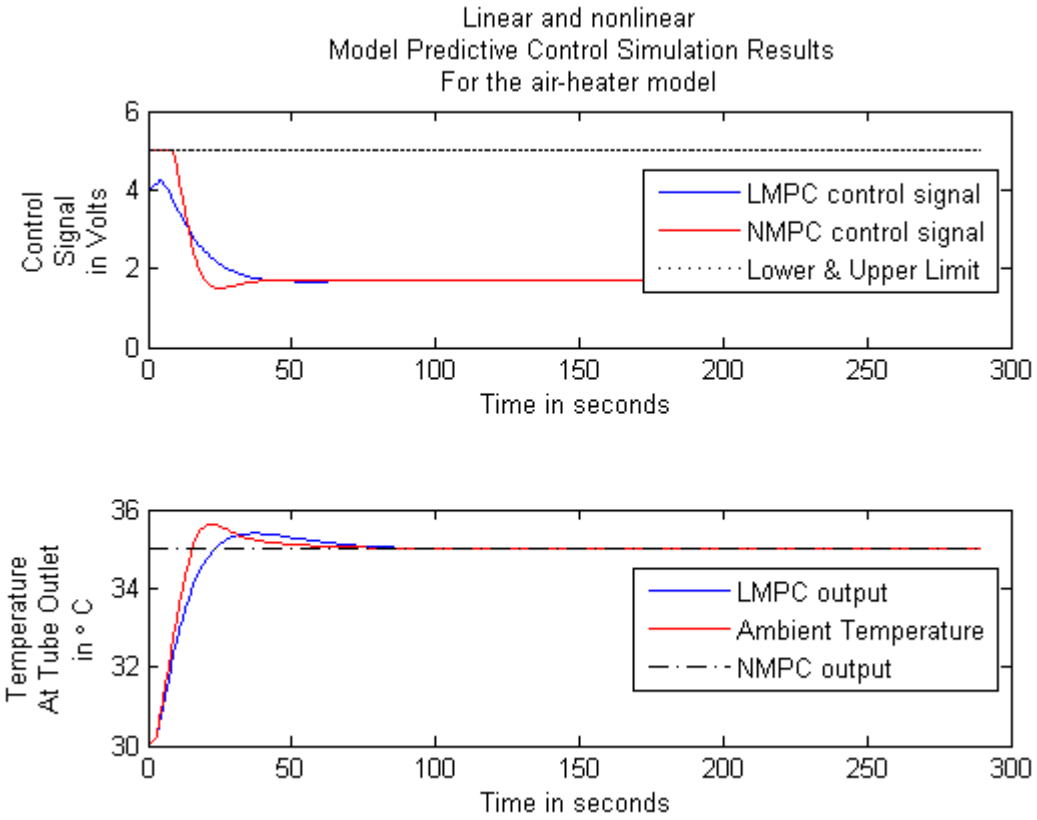


Figure 5.4 NMPC and LMPC simulation results when time constant is 150% of its nominal value

5.4.3Tolerance to changes in time delay

Both LMPC and NMPC have shown good tolerance for values of time delay from 0(no delay) to below 300% of its nominal value. When the time delay is increased to 300% of its nominal value, the output from the LMPC became oscillating as shown in Figure 5.5. However, the oscillations die out with time and eventually it reaches steady state. On the other hand, the NMPC became unstable when the time delay reached 300% of its nominal value. Although both controllers have shown good results to a wider range of time constant, the LMPC was found to be more robust against the changes in time delay.

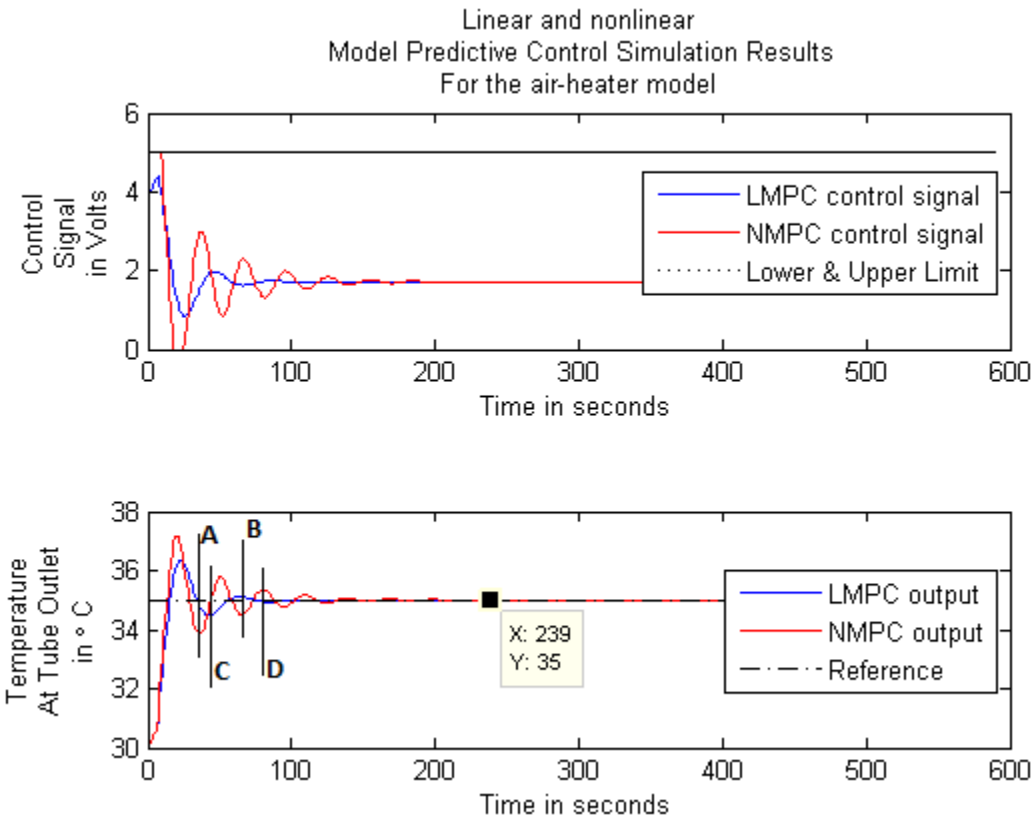


Figure 5.5 NMPC and LMPC simulation results when time delay is 300% of its nominal value

According to the decay ration comparison shown in Table 5-2, the NMPC becomes unstable when the time delay in the process model becomes double of that used in the controller. However, LMPC is stable when the value of time delay in the process model becomes twice of the time delay in the controller. Considering points A, B, C and D in Figure 5.5, the decay ration for the NMPC is 0.4 while the decay ratio of the LMPC is 0.

Table 5-2 decay ration in NMPC and LMPC when time delay is 300% of its nominal value

Controller	Decay Ratio	Remarks
LMPC	$\frac{peak\ at\ D}{peak\ at\ C} = \frac{0}{0.36} = 0 < 0.25$	Stable
NMPC	$\frac{peak\ at B}{peak\ at\ A} = \frac{0.44}{1.1} = 0.4 > 0.25$	Unstable

5.5 Execution time

The execution time was computed while each of the parameters mentioned in sections 5.2 to 5.4 , namely: ambient temperature, measurement noise, heater gain, time constant, and time delay were changing. In Table 5-3 to Table 5-7 the average, maximum and minimum execution times represent the time elapsed during a single iteration. The total execution time however, represents the duration from the start of the program to the finish.

Table 5-3 displays the execution time in both LMPC and NMPC with respect to changing ambient temperature. In the LMPC the average execution time is 7.7 seconds while in the NMPC the average execution time is 19.5. The execution time in NMPC is 2.6 times the execution time in LMPC. It means LMPC is about 3 times faster than NMPC.

Table 5-4 shows execution time in LMPC and NMPC with respect to increasing measurement noise. The execution time in LMPC did not show much variation while the measurement noise increases, and the average execution time is 7.5. In the NMPC however, the average execution time has increased to about three times its value without noise. As a result the average execution time in LMPC in the presence of measurement noise has become 8 times faster than the average execution time in NMPC.

Table 5-3 Execution time of LMPC and NMPC with respect to changing ambient temperature

Execution time in Linear MPC				Ambient Temp.	Execution time in Nonlinear MPC				$\frac{NMPC_{tot}}{LMPC_{tot}}$
Average Per cycle	Max Per cycle	Min Per cycle	Total		Average Per cycle	Max Per cycle	Min Per cycle	Total	
0.0148	0.6717	0.0089	8.4978	18	0.0676	1.4097	0.0157	22.9095	2.6959
0.0130	0.6637	0.0084	7.8504	19	0.0667	1.3973	0.0173	22.6783	2.8888
0.0111	0.6530	0.0061	7.5714	20	0.0635	1.3977	0.0155	22.1335	2.9233
0.0144	0.7362	0.0065	7.5391	21	0.0599	1.4748	0.0148	20.7334	2.7501
0.0114	0.6629	0.0061	7.7147	22	0.0589	1.3946	0.0154	20.3849	2.6423
0.0110	0.6578	0.0068	7.3350	23	0.0594	1.4662	0.0162	20.5319	2.7992
0.0108	0.6618	0.0065	7.3275	24	0.0588	1.6297	0.0160	20.3180	2.7728
0.0111	0.7611	0.0068	7.4485	25	0.0577	1.8666	0.0169	19.9875	2.6834
0.0111	0.7515	0.0059	7.5248	26	0.0547	1.7762	0.0153	19.1869	2.5498
0.0105	0.6200	0.0061	7.2969	27	0.0572	1.7776	0.0176	19.8705	2.7231
0.0107	0.5526	0.0071	7.3585	28	0.0575	2.1059	0.0167	19.7166	2.6794
0.0101	0.5498	0.0060	7.4476	29	0.0552	1.9517	0.0157	19.4131	2.6066
0.0121	0.6588	0.0062	8.0607	30	0.0558	1.5312	0.0165	19.5021	2.4194
0.0103	0.5973	0.0063	7.2536	31	0.0561	1.9780	0.0166	19.4896	2.6869
0.0102	0.5689	0.0062	7.2374	32	0.0530	1.9082	0.0164	18.6379	2.5752
0.0111	0.5829	0.0076	7.8120	33	0.0487	1.8994	0.0171	17.6589	2.2605
0.0103	0.5625	0.0069	7.1689	34	0.0475	1.5625	0.0179	17.4077	2.4282
0.0185	0.7005	0.133	9.6340	35	0.0248	1.2828	0.0168	10.3956	1.0791
0.0118	0.6452	0.0137	7.6710		0.0557	1.6561	0.0164	19.4976	2.5647

Table 5-4 Execution time of LMPC and NMPC with respect to changing measurement noise

Execution time for Linear MPC				Measurement Noise	Execution time for Nonlinear MPC			
Average Per cycle	Max Per cycle	Min Per cycle	Total		Average Per cycle	Max Per cycle	Min Per cycle	Total
0.0110	0.6612	0.0059	7.2683	0.0	0.0598	1.5042	0.0162	20.7273
0.0107	0.6667	0.0060	7.4045	0.5	0.1765	1.4797	0.0729	54.6043
0.0108	0.6595	0.0065	7.3735	1.0	0.1941	1.5290	0.0719	59.5685
0.0113	0.7696	0.0060	7.4956	1.5	0.2003	1.5251	0.0987	61.3508
0.0114	0.6503	0.0060	7.4964	2.0	0.1956	1.4752	0.0314	59.9552
0.0114	0.6662	0.0060	7.5793	2.5	0.1970	1.5119	0.0249	60.4228
0.0115	0.6483	0.0059	7.6433	3.0	0.2001	1.5253	0.412	61.2649
0.0119	0.6517	0.0067	7.5803	3.5	0.1952	1.5591	0.0279	59.8223
0.0123	0.6865	0.0070	7.8503	4.0	0.1988	1.5594	0.0230	61.0857

The execution time was also recorded to check if an error in the heater gain affects it. As can be seen from Table 5-5 the execution time in the LMPC does not vary so much. The execution time of the NMPC on the other hand increased with increasing heater gain. As the heater gain increased from 75% to 200% of its nominal value, the execution time in NMPC increased from 3 times to 6 times the execution time in LMPC.

Table 5-5 Execution time of LMPC and NMPC with respect to changing heater gain

Execution time for Linear MPC				Heater Gain	Execution time for Nonlinear MPC			
Average Per cycle	Max Per cycle	Min Per cycle	Total		Average Per cycle	Max Per cycle	Min Per cycle	Total
0.0146	0.7649	0.0082	8.8973	75%	0.0606	1.5367	0.0184	20.7352
0.0105	0.6628	0.0061	7.3814	100%	0.0610	1.6609	0.0158	21.0890
0.0108	0.7720	0.0064	7.7445	125%	0.0739	1.5417	0.0164	24.5876
0.0103	0.6642	0.0067	7.2447	150%	0.1129	1.4973	0.0183	35.9913
0.0104	0.6644	0.0066	7.2412	175%	0.1659	1.4989	0.0906	51.3128
0.0106	0.7002	0.0059	7.6758	200%	0.1810	1.5167	0.0177	55.6202

Table 5-6 shows execution time in LMPC and NMPC as the time constant changes from 25% to 400% of its nominal value. Total execution time in the LMPC is almost constant with an average value of 0.01 seconds. Total execution time in NMPC increases as the time constant deviated from its nominal value. When the heater reached twice its nominal value, execution time has become 2.7 times the value it had at the nominal value of heater gain. This can be seen from Table 5-5 where total execution time increased from 20.7 to 55.6 seconds.

Table 5-6 Execution time of LMPC and NMPC with respect to changing time constant

Execution time for Linear MPC				Time Constant	Execution time for Nonlinear MPC			
Average Per cycle	Max Per cycle	Min Per cycle	Total		Average Per cycle	Max Per cycle	Min Per cycle	Total
0.0109	0.6757	0.0064	7.2220	25%	0.1904	1.5374	0.0253	58.6525
0.0103	0.6807	0.0066	7.1607	50%	0.1791	1.6078	0.0753	55.1015
0.0105	0.6689	0.0064	7.1812	75%	0.0775	1.6013	0.0184	25.6501
0.0106	0.7020	0.0065	7.2883	100%	0.0602	1.5237	0.0157	20.5457
0.0104	0.6632	0.0059	7.1972	125%	0.0611	1.5045	0.0159	20.8813
0.0109	0.6737	0.0064	7.6424	150%	0.0676	1.4603	0.0162	22.7328
0.0111	0.6566	0.0068	7.3319	175%	0.0711	1.5117	0.0161	23.8164
0.0113	0.7472	0.0068	7.8192	200%	0.0786	1.9545	0.0168	25.9299
0.0115	0.6817	0.0068	7.4491	250%	0.0907	1.6701	0.0165	29.5000
0.0116	0.6562	0.0064	7.4853	300%	0.0999	1.5591	0.0184	32.1957
0.0118	0.6602	0.0060	7.5299	350%	0.1121	1.5144	0.0265	35.6527
0.0119	0.7074	0.0061	7.6373	400%	0.1215	1.5196	0.0340	38.3689

Similarly to the effects of the other parameters on execution time, LMPC did not show any change in execution time with increasing time delay; and NMPC showed an increase in execution time with increasing time delay as can be seen from Table 5-7

Table 5-7 Execution time of LMPC and NMPC with respect to changing time delay

Execution time for Linear MPC				Time Delay	Execution time for Nonlinear MPC			
Average Per cycle	Max Per cycle	Min Per cycle	Total		Average Per cycle	Max Per cycle	Min Per cycle	Total
0.0109	0.6506	0.0058	7.7250	0%	0.0443	1.5921	0.0165	16.3103
0.0108	0.6522	0.0058	7.8042	100%	0.0590	1.5035	0.0165	20.2590
0.0107	0.6580	0.0058	7.7979	200%	0.01185	1.4986	0.0254	37.4363
0.0110	0.6474	0.0058	7.6566	300%	0.1628	1.4468	0.0820	50.2641
0.0106	0.6549	0.0058	7.8101	400%	0.1604	1.4475	0.0852	49.6402
0.0116	0.6526	0.0071	7.4627	500%	0.1586	1.4585	0.0842	49.1297

As can be seen from the LMPC and NMPC plots on Figure 5.6, it is the first iteration which takes much time. The average computation time in the first iteration was calculated from the collected data and compared with the average computation time per cycle. The time elapsed during the first iteration was found out to be approximately 56 to 63 times greater than the average elapsed time per iteration in the LMPC, and 10 to 35 times greater than the average elapsed time in the NMPC.

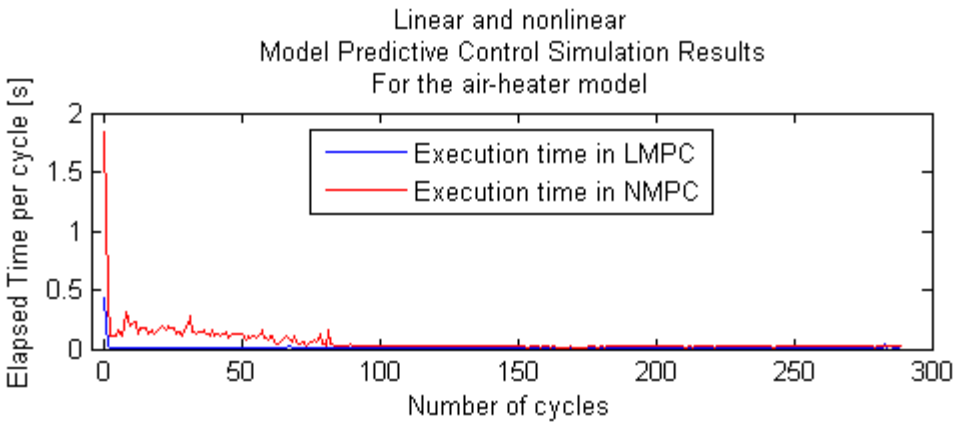


Figure 5.6 Elapsed time per cycle Vs number of cycles for LMPC and NMPC

Another important factor to look at is the number of iteration that the optimizations solver makes before it converges to a solution. Figure 5.7 shows the maximum number of iteration in the LMPC and NMPC. It can be clearly seen that the NMPC makes more iteration before it converges than the LMPC. The average number of iterations in the NMPC is 7 whereas the average number of iteration in the LMPC is1.4.

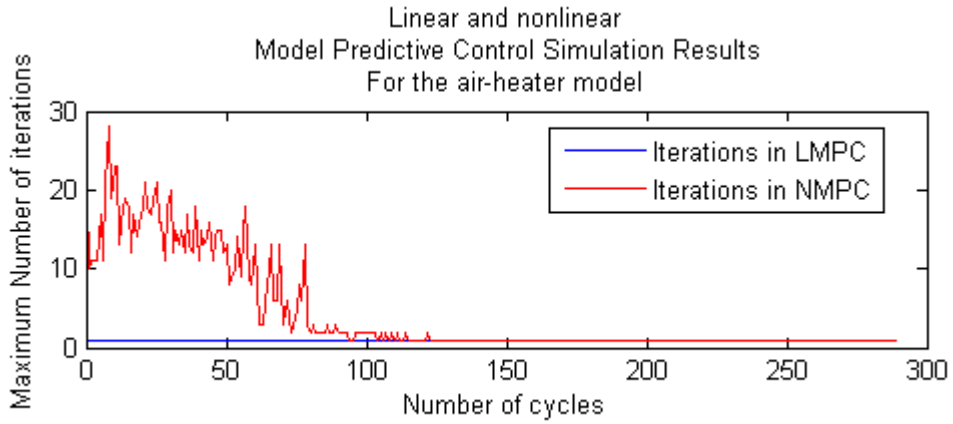


Figure 5.7 Maximum number of iteration vs number of cycles in LMPC and NMPC

There is an important thing to remember when simulating the NMPC – updating the initial sequence of optimal control signals. If the initial guessed vector of optimal control signals is not updated, the simulation works fine. However it takes about 3 times longer time than when the initial vector of control signals is updated. In Figure 5.8, the initial value of the optimal control sequence is not updated and the solver takes a random guess every time. Thus, it has to make much iteration to converge throughout the simulation loop. If the initial guess supplied to the objective function is updated by the optimal sequence from the previous iteration as in Figure 5.7, the number of iteration that the solver makes before it converges to the solution reduces dramatically and the controller becomes faster.

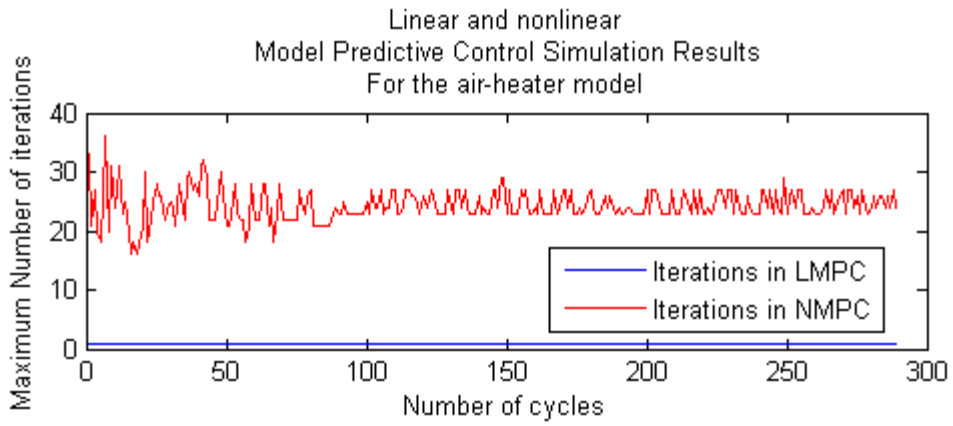


Figure 5.8 Maximum number of iteration vs number of cycles in NMPC and LMPC when u is not updated

Some additional information is also obtained from the iterative display. Iterative display is a table of statistics describing the calculations in each iteration of a solver Mathworks (2014a).The LMPC solver in this case is using QP , active set algorithm and it does not have the option for making iterative display. So, the iterative display is made only for the NMPC. Just for comparison the iterative display from the first and the last iterations are shown in Table 5-8 andTable 5-9. In the tables: Iter represents the iteration number, F_count represents the number of times the objective function is evaluated, and f(x) shows the value of the objective function etc. Table 5-8 shows that the solver took 27 iteration to converge while Table 5-9 shows that the solver actually converged in one iteration. Similarly, the objective function is evaluated 340 times in the first iteration and only 34 times in last iteration.

Table 5-8 Iterative display during the first iteration

Iter	F-count	f(x)	Max constraint	Line search steplength	Directional derivative	First-order optimality	Procedure
0	11	1.12433e+06	-1				
1	22	805885	0	1	-4.78e+04	2.96e+04	
2	33	795608	0	1	-7.35e+03	6.26e+03	
3	46	792619	0	0.25	-3.19e+03	7.81e+03	
4	59	790528	0	0.25	-2.76e+03	7.94e+03	
5	71	789751	0	0.5	-1.89e+03	1.88e+03	
6	84	789277	0	0.25	-1.48e+03	1.62e+03	
7	96	789020	0	0.5	-1.75e+03	1.81e+03	
8	107	787933	0	1	-1.91e+03	813	
9	124	787928	0	0.0156	-129	567	
10	136	787881	0	0.5	-599	401	
11	150	787876	0	0.125	-137	371	
12	163	787847	0	0.25	-410	418	
13	174	787817	0	1	-321	73.4	
14	185	787817	4.337e-19	1	-15.3	7.07	
15	196	788059	0	1	-2.96	828	
16	207	787817	0	1	-870	12.8	
17	220	787817	0	0.25	-11.9	5	
18	234	787817	0	0.125	-2.19	2	
19	248	787817	0	0.125	-1.56	2.81	
20	259	787817	0	1	-3.98	3.15	
21	270	787817	0	1	-3.07	2.25	
22	283	787817	0	0.25	-1.16	1.38	
23	295	787817	0	0.5	-2	0.967	
24	306	787817	0	1	-1.06	0.406	
25	318	787817	0	0.5	-0.235	0.133	Hessian modified
26	329	787817	1.355e-20	1	-0.192	0.0716	Hessian modified
27	340	787817	2.118e-22	1	-0.0815	0.018	Hessian modified

Local minimum possible. Constraints satisfied.

Table 5-9 Iterative display during the last iteration; u_guess is updated

Iter	F-count	f(x)	Max constraint	Line search steplength	Directional derivative	First-order optimality	Procedure
0	11	19040.2	-8.61e-07				
1	34	19040.2	-8.608e-07	0.000244	-0.0663	0.021	

Local minimum possible. Constraints satisfied.

Table 5-10 is basically the same like Table 5-9 it shows the results during the last iteration. It takes more number of iterations and function counts only because the initial guess of control signal is not updated by the optimal sequence.

Table 5-10 Iterative display during the last iteration; u_guess is not updated

Iter	F-count	f(x)	Max constraint	Line search steplength	Directional derivative	First-order optimality	Procedure
0	11	37576.7	-1				
1	23	37355.9	-0.5	0.5	-7.7e+03	2.52e+04	
2	34	25910.7	0	1	-7.44e+03	1.07e+04	
3	47	22247.2	0	0.25	-5.01e+03	1.75e+04	
4	60	20000.3	0	0.25	-3.12e+03	5.52e+03	
5	74	19849.2	0	0.125	-703	945	
6	86	19488.4	-0.082	0.5	-1.48e+03	1.69e+03	
7	100	19379.7	-0.07175	0.125	-592	805	
8	115	19283.1	-0.06726	0.0625	-662	2.11e+03	
9	128	19233.8	-0.05045	0.25	-487	504	
10	143	19194.6	-0.04729	0.0625	-407	624	
11	155	19152.9	-0.02365	0.5	-496	455	
12	168	19094.2	-0.01774	0.25	-502	782	
13	182	19066.7	-0.01552	0.125	-304	382	
14	193	19046.6	0	1	-285	135	
15	204	19040.6	0	1	-144	31.5	
16	224	19040.6	0	0.00195	-2.42	25.9	
17	235	19040.3	0	1	-31.7	8.06	
18	255	19040.3	-0.0001784	0.00195	-1.75	9.36	
19	266	19040.3	4.879e-19	1	-11.3	0.759	
20	277	19040.3	0	1	-1.56	0.064	
21	289	19040.3	-1.025e-05	0.5	-0.0771	0.0591	
22	301	19040.3	-1.006e-05	0.5	-0.0697	0.0742	Hessian modified
23	312	19040.3	0	1	-0.0613	0.0401	Hessian modified

Local minimum possible. Constraints satisfied.

5.6 Integral of absolute value of the error (IAE)

The integral of absolute error which is a measure of set point tracking Haugen (2014) is given by :

$$IAE_s = \int_0^\infty |e(t)|dt \qquad (5\text{-}1)$$

The IAE_d measures the disturbance compensation and is given by:

$$IAE_d = \int_0^\infty |e(t)|dt \qquad (5\text{-}2)$$

In Figure 5.9 a step change is applied at 300 seconds. The IAE_s is calculated from zero to 300 seconds while the IAE_d is calculated from 300 to 600 seconds and the results are given in Table 5-11.

Table 5-11 IAE results for LMPC and NMPC

	LMPC	NMPC	LMPC/NMPC
IAE_s	0.9117	1.6191	0.5631
IAE_d	9.4459	9.9231	0.9519

From Table 5-11it can be seen that the IAE_s of the LMPC is 56% of the IAE_s of the NMPC. This shows that the set point tracking of LMPC is much better than that of NMPC in this case. The IAE_d of LMPC is 95% of the IAE_d of NMPC. Both have comparable disturbance compensation.

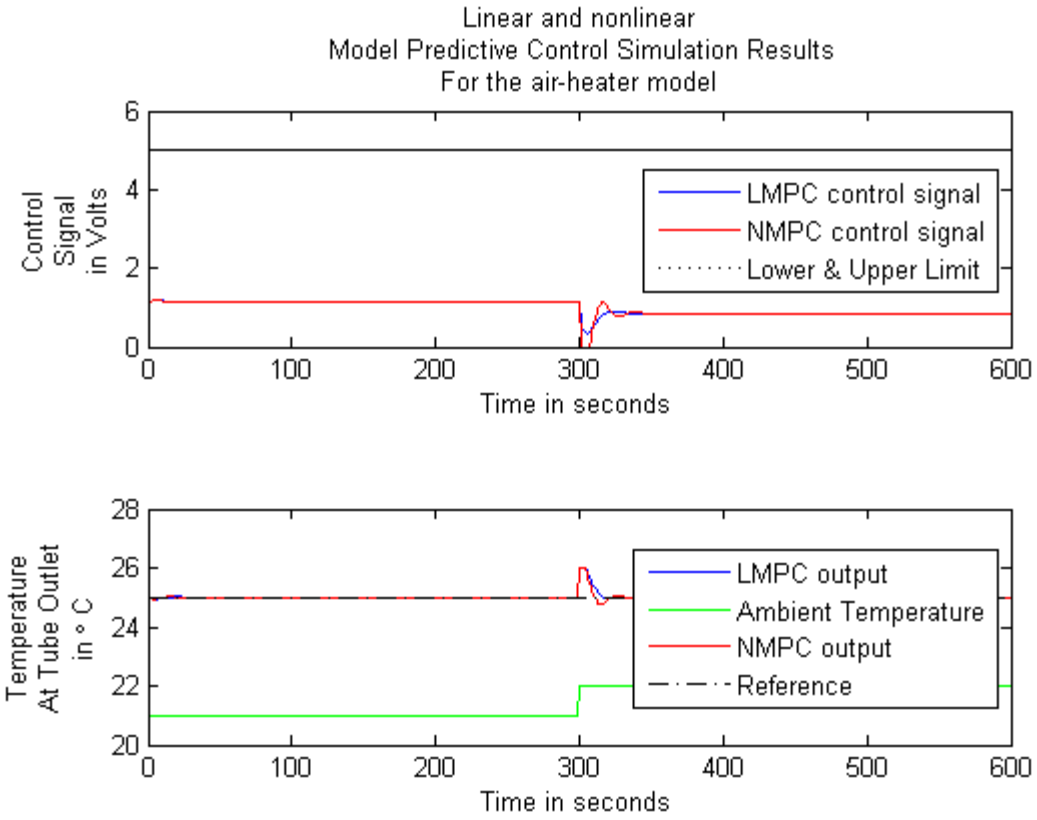


Figure 5.9 Disturbance rejection and set point tracking in LMPC and NMPC

6 Linear MPC applied to the AD reactor model

In this chapter linear MPC is designed for a pilot reactor used for anaerobic digestion (AD) of dairy manure and compared against a nonlinear MPC results already designed by Finn Haugen (2013). Many of the techniques used to develop the LMPC for the air-heater model in chapters 2 to 4 are also applicable for the AD reactor model. So, some topics are referenced to the relevant chapters whenever applicable.

6.1 Modified Hill's Model

The anaerobic digestion reactor model is taken from Finn Haugen (2013) . In the article the model is mentioned as modified Hill's model. The equations that constitute the model are presented below:

The portion of the raw waste that can serve as substrate is defined as:

$$S_{bvs_{in}} = B_0 S_{vs_{in}} \tag{6-1}$$

The portion of the biodegradable material which is initially in the acid form is defined as:

$$S_{vfa_{in}} = A_f S_{bvs_{in}} \tag{6-2}$$

Mass balance of biodegradable volatile solids:

$$\dot{S}_{bvs} = (S_{bvs_{in}} - S_{bvs}) \frac{F_{feed}}{V} - \mu k_1 X_{acid} \tag{6-3}$$

Mass balance of total volatile fatty acids (VFA):

$$\dot{S}_{vfa} = (S_{vfa_{in}} - S_{vfa}) \frac{F_{feed}}{V} + \mu k_2 X_{acid} - \mu_c k_3 X_{meth} \tag{6-4}$$

Mass balance of acidogens:

$$\dot{X}_{acid} = (V - K_d - \frac{\frac{F_{feed}}{b}}{V}) X_{acid} \tag{6-5}$$

Mass balance of methanogens:

$$\dot{X}_{meth} = (\mu_c - K_{dc} - \frac{\frac{F_{feed}}{b}}{V}) X_{meth} \tag{6-6}$$

Methane gas flow rate (gas production)

$$F_{meth} = V\mu_c\, k_5 X_{meth} \tag{6-7}$$

Where the reaction rates with Monod kinetics, are:

$$\mu = \mu_m \frac{S_{bvs}}{K_s + S_{bvs}} \tag{6-8}$$

$$\mu_c = \mu_{mc} \frac{S_{vfa}}{K_{sc} + S_{vfa}} \tag{6-9}$$

The maximum reaction rates μ_m and μ_{mc} are functions of the reactor temperature and are given as:

$$\mu_m(T_{reac}) = \mu_{mc}(T_{reac}) = 0.013T_{reac} - 0.129 \tag{6-10}$$

$$(20^\circ C < T_{reac} < 60^\circ C)$$

The simplified representation of the model is given in Figure 6.1.

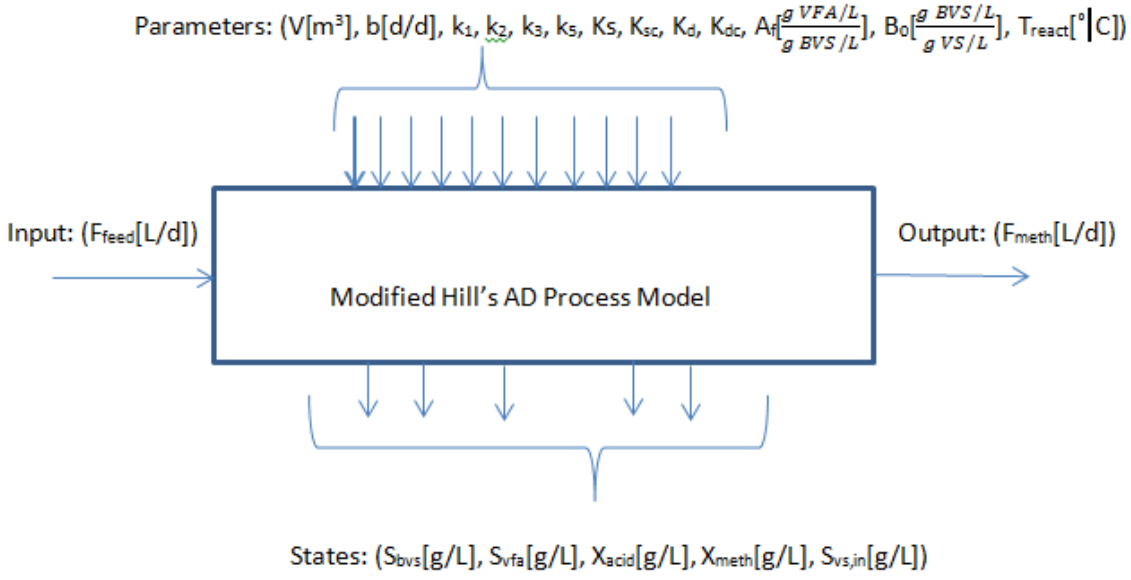


Figure 6.1 AD reactor model

F_{feed} is the control variable and F_{meth} is the controlled variable. S_{vsin} is augmented as a state and its rate of change is assumed to be zero. Reactor temperature is considered as a parameter.

6.2 Linearizing and discretizing the AD reactor model

The linear MPC algorithm requires a linear model. Thus, the nonlinear AD reactor model (shown in equations *(6-1)* to *(6-10)*) needs to be linearized. The linearization is briefly described here based on Lie (2005).

Assuming a nonlinear model given by *(6-11)*:

$$\frac{dx}{dt} = f(x,u) \qquad (6\text{-}11)$$

Where x represents the states and u represents inputs and is further assumed that an operating point (x_s, u_s) is chosen and Taylor series expansion of $f(x,u)$ is valid at (x_s, u_s) .

Then,

$$f(x,u) \cong f(x_s,u_s) + \left[\frac{\partial f(x,u)}{\partial x}\right]_{\substack{x=x_s \\ u=u_s}} (x - x_s) + \left[\frac{\partial f(x,u)}{\partial u}\right]_{\substack{x=x_s \\ u=u_s}} (u - u_s)$$

For simplicity define A as in *(6-12)*:

$$A = \left[\frac{\partial f(x,u)}{\partial x}\right]_{\substack{x=x_s \\ u=u_s}} \qquad (6\text{-}12)$$

And B as in *(6-13)*:

$$B = \left[\frac{\partial f(x,u)}{\partial u}\right]_{\substack{x=x_s \\ u=u_s}} \qquad (6\text{-}13)$$

Thus:

$$\frac{dx}{dt} = f(x_s,u_s) + \ A(x - x_s) + B(u - u_s)$$

Assuming constant x_s leads to $\frac{dx_s}{dt} = 0$, such that

$$\frac{dx}{dt} = \frac{dx}{dt} - \frac{dx_s}{dt} = f(x_s,u_s) + \ A(x - x_s) + B(u - u_s)$$

$$\frac{d(x - x_s)}{dt} = f(x_s,u_s) + \ A(x - x_s) + B(u - u_s)$$

Defining: $\delta x \equiv x - x_s$ and $\delta u \equiv u - u_s$ we get:

$$\frac{d\delta x}{dt} = A\delta x + B\delta u + f(x_s,u_s)$$

Choosing the operating point(x_s, u_s) such that

$$\frac{dx}{dt} = f(x_s, u_s) = 0$$

Gives an equation that defines the deviation between $x\ and\ x_s$ as shown in *(6-14)*

$$\frac{d\delta x}{dt} = A\delta x + B\delta u \qquad (6\text{-}14)$$

Similarly we can linearize the equation from input variable x to output variable y as:

$$y = g(x)$$

$$y = g(x) \cong g(x_s) + \frac{\partial g}{\partial x}|_{x_s} + (x - x_s)$$

Defining $y_s = g(x_s)$ and

$$C = \frac{\partial g}{\partial x}|_{x_s} \qquad (6\text{-}15)$$

and $\delta y \equiv y - y_s$ we get the equation that defines the deviation between $y\ and\ y_s$ as in *(6-16)*

$$\delta y = C\delta x \qquad (6\text{-}16)$$

In the AD reactor model the input $u = F_{feed}$, the output $y = F_{meth}$ and the vector of states $x = \left[S_{bvs}, S_{vfa}, X_{acid}, X_{meth}, S_{vs,in}\right]^T$. Following the same method as in equations *(6-11)* to *(6-16)*, the AD reactor model is linearized around a steady state operating point values shown in Table 6-1.

Table 6-1 steady state operating point taken from Finn Haugen (2014)

F_{feed} = 35.3L/d
T_{reac} = 35 ℃
S_{bvs} = 4.14 g/L
S_{vfa} = 0.8 g/L
X_{acid} = 1.80g/L
X_{meth} = 0.39g/L
S_{vsin} = 30.2 g/L
F_{meth} = 174.2 L CH4/d

Defining the ODEs mentioned in equations *(6-3)* and *(6-6)* as f_1, f_2, f_3, f_4 and augmenting $f_5 = \dot{S}_{vs_{in}}$ as zero rate of change of $S_{vs_{in}}$ assuming constant or slowly varying $S_{vs_{in}}$:

$$f_1 = \dot{S}_{bvs} = (S_{bvs_{in}} - S_{bvs})\frac{F_{feed}}{V} - \mu k_1 X_{acid}$$

$$f_2 = \dot{S}_{vfa} = (S_{vfa_{in}} - S_{vfa})\frac{F_{feed}}{V} + \mu k_2 X_{acid} - \mu_c k_3 X_{meth}$$

$$f_3 = \dot{X}_{acid} = (V - K_d - \frac{\frac{F_{feed}}{b}}{V})X_{acid}$$

$$f_4 = \dot{X}_{meth} = (\mu_c - K_{dc} - \frac{\frac{F_{feed}}{b}}{V})X_{meth}$$

$$f_5 = \dot{S}_{vs_{in}} = 0$$

Then, from equation *(6-12)* A is given by:

$$A = \left[\frac{\partial f(x,u)}{\partial x}\right]_{\substack{x=x_s \\ u=u_s}} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \frac{\partial f_1}{\partial x_4} & \frac{\partial f_1}{\partial x_5} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \frac{\partial f_2}{\partial x_4} & \frac{\partial f_2}{\partial x_5} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} & \frac{\partial f_3}{\partial x_4} & \frac{\partial f_3}{\partial x_5} \\ \frac{\partial f_4}{\partial x_1} & \frac{\partial f_4}{\partial x_2} & \frac{\partial f_4}{\partial x_3} & \frac{\partial f_4}{\partial x_4} & \frac{\partial f_4}{\partial x_5} \\ \frac{\partial f_5}{\partial x_1} & \frac{\partial f_5}{\partial x_2} & \frac{\partial f_5}{\partial x_3} & \frac{\partial f_5}{\partial x_4} & \frac{\partial f_5}{\partial x_5} \end{pmatrix}$$

$$A = \begin{pmatrix} -\frac{F_{feed}}{V} - \frac{\mu_m k_1 K_s X_{acid}}{(K_s + S_{bvs})^2} & 0 & -\frac{k_1 \mu_m S_{bvs}}{(K_s + S_{bvs})} & 0 & \frac{B_0 F_{feed}}{V} \\ \frac{\mu_m k_2 K_s X_{acid}}{(K_s + S_{bvs})^2} & -\frac{F_{feed}}{V} - \frac{\mu_{mc} k_3 K_{sc} X_{meth}}{(K_{sc} + S_{vfa})^2} & \frac{k_2 \mu_m S_{bvs}}{(K_s + S_{bvs})} & -\frac{k_3 \mu_{mc} S_{vfa}}{(K_{sc} + S_{vfa})} & \frac{A_f B_0 F_{feed}}{V} \\ \frac{\mu_m K_s X_{acid}}{(K_s + S_{bvs})^2} & 0 & -\frac{F_{feed}}{bV} + \frac{\mu_m S_{bvs}}{(K_s + S_{bvs})} - K_d & 0 & 0 \\ 0 & \frac{\mu_{mc} K_{sc} X_{meth}}{(K_{sc} + S_{vfa})^2} & 0 & -\frac{F_{feed}}{bV} + \frac{\mu_{mc} S_{vfa}}{(K_s + S_{vfa})} - K_{dc} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Similarly following equation *(6-13)* B is given by:

$$B = \left[\frac{\partial f(x,u)}{\partial u}\right]_{\substack{x=x_s \\ u=u_s}} = \begin{pmatrix} \frac{\partial f_1}{\partial u} \\ \frac{\partial f_2}{\partial u} \\ \frac{\partial f_3}{\partial u} \\ \frac{\partial f_4}{\partial u} \\ \frac{\partial f_5}{\partial u} \end{pmatrix} = \begin{pmatrix} \frac{B_0 S_{vs_{in}} - S_{bvs}}{V} \\ \frac{A_f B_0 S_{vs_{in}} - S_{vfa}}{V} \\ -\frac{X_{acid}}{bV} \\ -\frac{X_{meth}}{bV} \\ 0 \end{pmatrix}$$

And from equaiton *(6-15)* C is given by:

$$C = \frac{\partial g}{\partial x}|_{x_s} = \begin{pmatrix} \frac{\partial F_{meth}}{\partial x_1} & \frac{\partial F_{meth}}{\partial x_2} & \frac{\partial F_{meth}}{\partial x_3} & \frac{\partial F_{meth}}{\partial x_4} & \frac{\partial F_{meth}}{\partial x_5} \end{pmatrix}$$

$$C = \frac{\partial g}{\partial x}|_{x_s} = \begin{pmatrix} 0 & \frac{\mu_{mc} V k_5 K_{sc} X_{meth}}{(K_{sc} + S_{vfa})^2} & 0 & \frac{V k_5 \mu_{mc} S_{vfa}}{(K_{sc} + S_{vfa})} & 0 \end{pmatrix}$$

The values of the parameters in the AD reactor model are shown in Table 6-2.

Table 6-2 Parameters in the AD reactor model Finn Haugen (2013)

Parameter	Value	Unit	Comment
A_f	0.69	(g VFA/L)/ (g BVS/L)	Acidity constant
b	2.90	d/d	Retention time ratio
B_0	0.25	(g BVS/L)/ (g VS/L)	Biodegradability constant
k_1	3.89	g BVS/(g acidogens/L)	Yield constant
k_2	1.76	g VFA/(g acidogens/L)	Yield constant
k_3	31.7	g VFA/(g methanogens/L)	Yield constant
k_5	26.3	L/g methanogens	Yield constant
K_d	0.02	d^{-1}	Specific death rate of acidogens
K_{dc}	0.02	d^{-1}	Specific death rate of methanogens
K_s	15.5	g BVS/L	Monod half-velocity constant for acidogens
K_{sc}	3	g VFA/L	Monod half-velocity constant for methanogens
V	250	L	Effective reactor volume

Now that the model is linearized, the steady state operating point and the parameters are known, the model can be discretized and implemented in MATLAB. The AD reactor model is MATLAB software and The MATLAB code for linearization and discretization is shown in Appendix 6

6.3 Constraints

Once the AD reactor model is linearized and discretized it has the same form as the linear air heater model shown in equations *(2-5)* and *(2-6)*. The only difference is that the model has become more complicated with more number of states and parameters. Thus, the control objective (section 2.3), quadratic programming (section 2.4), equality constraints (section 2.5) described for the LMPC of the air-heater model are also applicable for the AD reactor model. They are not thus repeated here. In this AD reactor model an inequality constraint on one of the states is added. In Finn Haugen (2014) it is assumed that $S_{vfa} \leq 0.8$ defines safe operation of the reactor. The same assumption is made here and an inequality constraint is added in the LMPC in order to keep the S_{vfa} at safe level. In order to do that, the inequality matrix A_i and vector b_i need to be supplied to the quadratic programming (see section 2.4). Similar to the equality constraints, four equations that correspond to the states, output, error and input are constructed and are stacked in a matrix as shown below.

$$A_i = \begin{pmatrix} A_{i,xu} & A_{i,xx} & A_{i,xe} & A_{i,xy} & A_{i,xdu} \\ A_{i,yu} & A_{i,yx} & A_{i,ye} & A_{i,yy} & A_{i,ydu} \\ A_{i,eu} & A_{i,ex} & A_{i,ee} & A_{i,ey} & A_{i,edu} \\ A_{i,uu} & A_{i,ux} & A_{i,ue} & A_{i,uy} & A_{i,udu} \end{pmatrix} \text{ and } b_i = \begin{pmatrix} b_{i,x} \\ b_{i,y} \\ b_{i,e} \\ b_{i,u} \end{pmatrix}$$

All the sub matrices of matrix A_i are zeroes of appropriate dimensions except $A_{i,xx}$ which is given by:

$$A_{i,xx} = [0_{N.nx \times N(nx-(nx-1))} \; 1_{N.nx \times N(nx-(nx-1))} \; 0_{N.nx \times N(nx-(nx-3))}]$$

Therefore A_i is given by:

$$A_i = \begin{pmatrix} 0_{N.nx \times N.nu} & A_{i,xx} & 0_{N.nx \times N.ny} & 0_{N.nx \times N.ny} & 0_{N.nu \times N.nu} \\ 0_{N.ny \times N.nu} & 0_{N.ny \times N.nx} & 0_{N.ny \times N.ny} & 0_{N.ny \times N.ny} & 0_{N.nu \times N.nu} \\ 0_{N.nu \times N.nu} & 0_{N.nu \times N.nx} & 0_{N.nu \times N.ny} & 0_{N.nu \times N.ny} & 0_{N.nu \times N.nu} \\ 0_{N.nu \times N.nu} & 0_{N.nu \times N.nx} & 0_{N.nu \times N.ny} & 0_{N.nu \times N.ny} & 0_{N.nu \times N.nu} \end{pmatrix}$$

In the equality matrix A_i, the first to the fourth raw correspond to the state, output, error and input equations while the first to the fifth columns correspond to the five variables, namely: input, state, output, error, and error increment respectively. The input, error and error increment are weighted in the performance index (the same performance index as in section 2.3 is used) while the state and output are not.

The b_i vector is given by:

$$b_i = \begin{pmatrix} b_{i,x} \\ b_{i,y} \\ b_{i,e} \\ b_{i,u} \end{pmatrix} = \begin{pmatrix} 0.8_{N.nx\times 1} \\ 0_{N.ny\times 1} \\ 0_{N.ny\times 1} \\ 0_{N.nu\times 1} \end{pmatrix}$$

The input signal is constrained between 0 volts and 40 volts.

6.4 Implementation in MATLAB

The control objective, quadratic programming and equality constraints which are formulated in the same way as in section 2.5 along with the formulations added in chapter 6 are implemented in MATLAB. The parameters of the AD reactor model are set to the values shown in Table 6-2. The MATLAB code for the LMPC as applied to the AD reactor model is shown in Appendix 7.

7 Nonlinear MPC applied to the AD reactor model

Similar to the nonlinear MPC in the air-heater model, fmincon solver is also used here. The objective function which is used for the AD reactor model is shown in equation (7-1). The AD reactor model shown in equations (6-1) to (6-10) is used where F_{feeed} is the input and F_{meth} is an output. In addition to the objective function a constraint function is needed by the solver. An unscented Kalman filter is also used to estimate the states and S_{vsin} which is augmented as state. The code for objective function is shown in Appendix 9. The initial values and the parameters are set to the values shown in Table 6-2. When using the nonlinear solver, linearization and discretization are not required.

7.1 Control objective

The following control objective is defined for the AD reactor model

$$J = \int_{t}^{t+\tau_h} (W_e e^2(t) + W_u u^2(t) + W_{\dot{u}} \dot{u}^2(t))dt \qquad (7\text{-}1)$$

Such that: $u_{min} \leq u(t) \leq u_{max}$

Where: e represents the control error

u represents the amplitude of the control signal

$\dot{u}$ represents the rate of change of the control signal

t is the present time

τ_h is the prediction horizon

$W_e, W_u \; and \; W_{\dot{u}}$ are weights for $e, u \; and \; \dot{u}$ respectively.

The forward (explicit) Euler method is used for solving the ODEs in the objective function. The forward Euler method is simple ODEs integrator Tallent (2012). The simulation that uses the forward Euler method only depends on the past values of state variables and state derivatives.

Let:

t_k be the time at k^{th} time-step

J_k be the computed solution at the k^{th} time-step

$T_s = t_k - t_{k-1}$ be the step size

Then, the forward Euler method can be summarized as:

$J_{k+1} = J_k + T_s(W_e e^2 + W_u u^2 + W_{\dot{u}} \dot{u}^2)_k$

7.2 Summary of the MATLAB functions used in the AD reactor model

The implementation in MATLAB for the AD reactor model is similar to the air-heater model and the following MATLAB functions are used in the AD reactor model:

7. AD_reactor_Linearize_and_Discretize.m (Appendix 6)
   This function linearizes and discretizes the nonlinear AD reactor model and is used by the AD_reactor_LMPC.m
8. AD_reactor_LMPC.m (Appendix 7)
   This function represents the LMPC for the AD reactor model and is invoked by the AD_reactor_LMPC_and_NMPC_plot.m
9. AD_reactor_NMPC.m (Appendix 8)
   This function represents the NMPC for the AD reactor model and is invoked by the AD_reactor_LMPC_and_NMPC_plot.m
10. AD_reactor_objective.m (Appendix 9)
    This function represents the objective function for the NMPC of the AD reactor model and is invoked by AD_reactor_NMPC.m
11. AD_reactor_constraint.m (Appendix 10)
    This function represents the constraint function for the NMPC of the AD reactor model and is invoked by AD_reactor_NMPC.m . This function has only empty matrices for the constraints it is just added so that the fmincon slover in AD_reactor_NMPC.m works properly.
12. AD_reactor_ukf.m (Appendix 11)
    This function represents the unscented Kalman filter for the NMPC of the AD reactor model and is invoked by AD_reactor_NMPC.m
13. AD_reactor_LMPC_and_NMPC_plot.m (Appendix 13)
    This script calls the AD_reactor_LMPC.m and the AD_reactor_NMPC.m and plots the results from the LMPC and NMPC functions.

8 Comparing LMPC and NMPC results from the AD reactor model

In this chapter a brief comparison of the results from the AD reactor LMPC and NMPC is done. Topics like: inequality constraints, IAE and computation time are discussed in the sections that follow.

8.1 Handling inequality constraints

The S_{vfa} is assumed to be limited to a maximum value of 0.8 for safe reactor operation. In the nonlinear model this is fulfilled by controlling S_{vfa} indirectly though other variables. Such control action will be effective if the model is good and if the reference is not set so high that S_{vfa} does not pass its maximum limit. However if the model is erroneous or somehow reference is set very high, then, S_{vfa} can pass its maximum limit as shown in Figure 8.1.

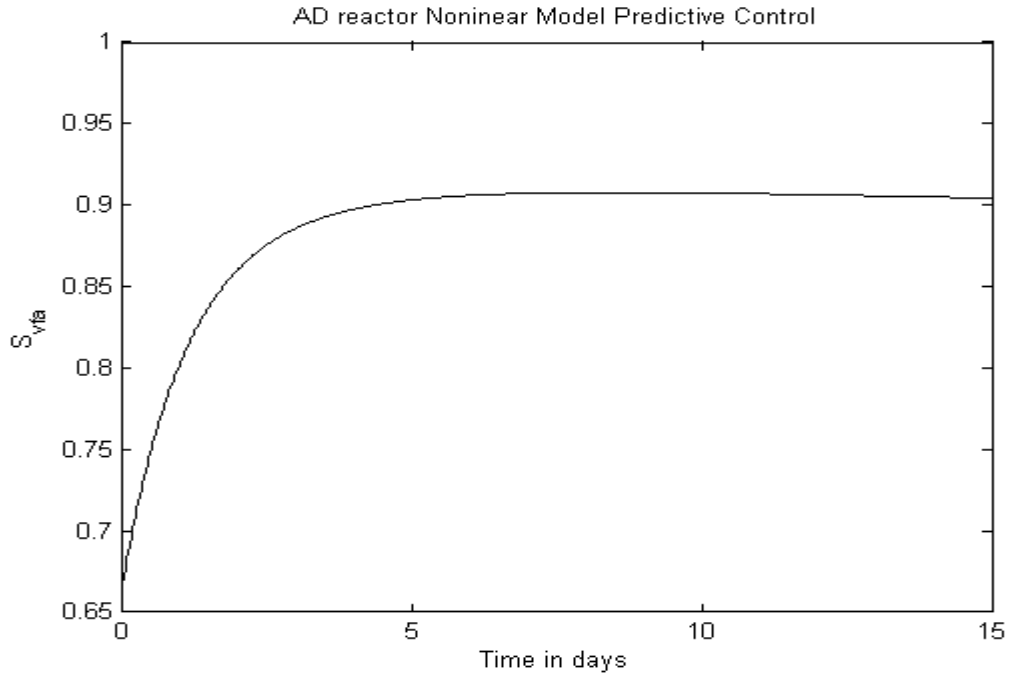


Figure 8.1 one sample of S_{vfa} simulation in AD reactor NMPC

Thus it would be advantageous if the concentration of VFA can be explicitly constrained to its maximum level. In fact in Finn Haugen (2014) it is mentioned that it will be safer to define S_{vfa} maximum limit explicitly instead of relying on the model alone(as it was done in the NMPC) to predict a possible failure. In the LMPC the constraint is directly applied to S_{vfa} (section 6.3) and it can be limited to a save level as shown in Figure 8.2 regardless of the values of the other variables in the model.

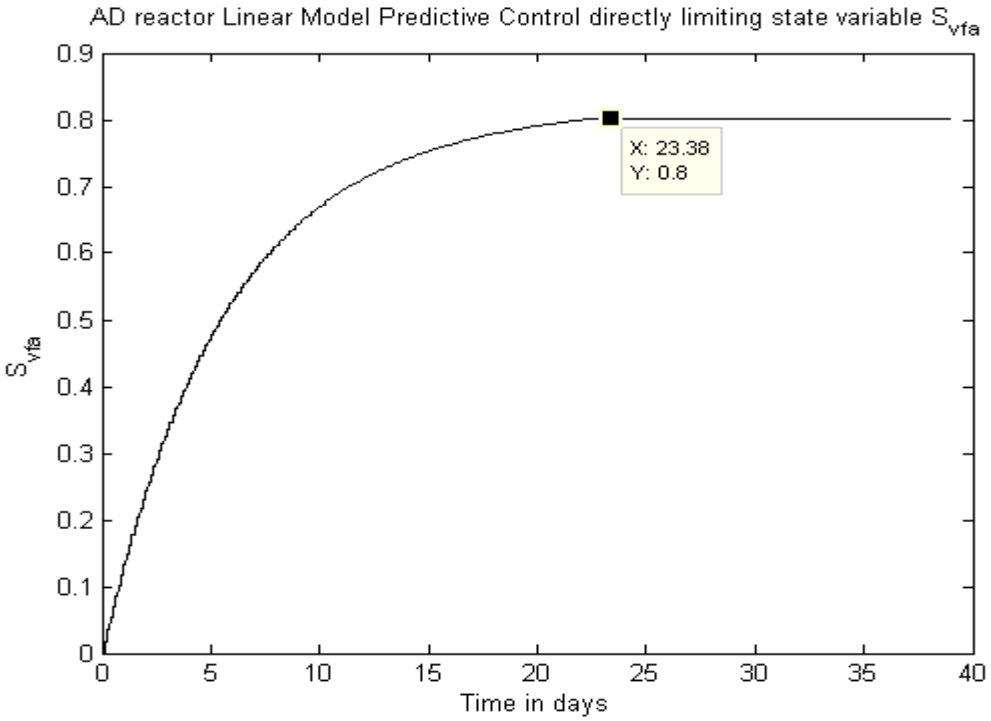


Figure 8.2 one sample of S_{vfa} simulation in AD reactor LMPC

8.2 AD reactor LMPC and NMPC simulation results

The simulation results for the AD reactor LMPC and NMPC are shown in Figure 8.3. Both LMPC and NMPC show comparable set point tracking. However, it can be noticed that the LMPC uses more input to give the same output as the NMPC. It is not clear why the LMPC uses bigger amount of input to produce the same output as the NMPC produces.

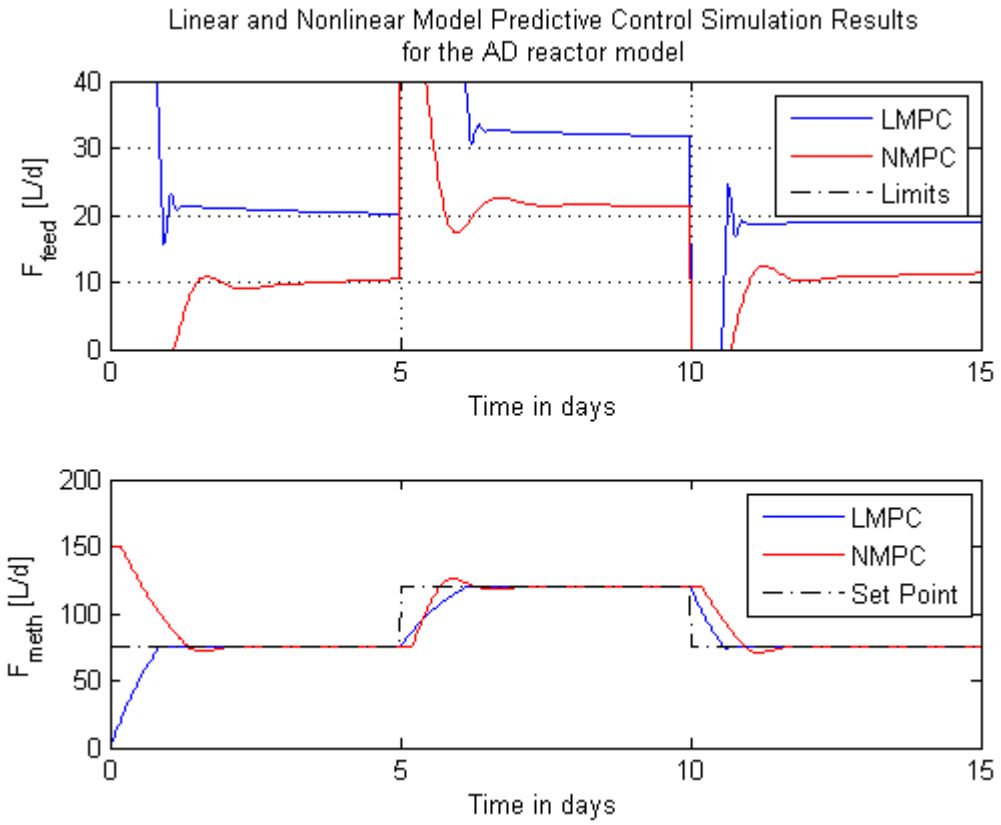


Figure 8.3 AD reactor LMPC and NMPC simulation result

8.3 Computation time and IAE

Other important factors to look at are the computation time and the IAE. As far as the execution time is concerned the LMPC has a clear advantage over the NMPC. On the other hand the NMPC is better on IAE basis, see Table 8-1. The IAE for the simulation result shown in Figure 8.3 are calculated from 0 to 7 days and the results are shown in Table 8-1.

Table 8-1 Comparing computation time and IAE for the AD reactor LMPC and NMPC

	LMPC	NMPC	NMPC/LMPC
Total execution time	8.322879 seconds	414.849299 seconds	49.8444
IAE(0 to 7 for Figure 8.1)	48.9200	26.6288	1.8371

The results in Table 8-1 show that the LMPC is about 50 times faster than the NMPC but the IAE value of the LMPC is approximately twice of that of NMPC for the simulations under consideration; i.e. Figure 8.3.

9 Discussion and Conclusion

In this thesis linear and nonlinear MPC algorithms are developed and evaluated as applied to an air-heater model and AD reactor model. In the linear case the model is linearized, discretized and formulated in to a quadratic programming problem in such a way that it is possible to use the QP solver in MATLAB. In the nonlinear case, the fmincon solver is used which directly optimizes the model without linearization. An unscented Kalman filter is used to estimate the states and disturbances.

The LMPC and NMPC are compared based on their ability to tolerate process disturbance, measurement noise and model error. Moreover, their computation time ease of tuning and implementation are considered.

In the case of the air-heater model, both LMPC and NMPC are simulated while the ambient temperature changes from 18 ˚C to 35 ˚C. Some overshoot was observed in the NMPC with changing ambient temperature while the simulation results from LMPC did not show significant overshoot. Generally speaking both MPC algorithms have shown good tolerance to process disturbance.

Another factor used for comparison of LMPC and NMPC in the air heater model was measurement noise. Both MPCs have shown quite similar responses under the same effect of measurement noise.

For the air heater model, the effect of model error is more visible in the nonlinear MPC applied to the air-heater model. The NMPC became unstable at 175% of nominal gain, and at 300% of nominal value of the time delay while the LMPC remained stable in both cases.

Execution time was about 3 times slower in the NMPC and gets slower and slower with increase of model error. In the LMPC the execution time was not affected by model error. It remained constant while the model error increases.

Two important qualities were observed in the case of LMPC for the AD reactor model. Firstly, the computation time was about 50 times shorter than in the NMPC. Secondly, it was easier to add inequality constraint to the state variable S_{vfa} to keep it at safe level. This helps to directly impose the required maximum limit on S_{vfa} instead of relying on the model to keep VFA concentration at safe level. It is good that an explicit inequality constraint is added in the LMPC as it is not clear to what extent the model is able to predict a possible failure of the real reactor due to high concentration of VFA (Finn Haugen, 2014) . The LMPC, however uses more input to produce the same output as the NMPC.

The NMPC applied to the AD reactor, on the other hand has its own good qualities. Its IAE is half of that of the LMPC and has shown stable response over a wider operation range than the LMPC.

In both the air-heater model and the AD reactor model, it is easier to implement the NMPC than the LMPC. It does not require linearization and discretization. When tuning the NMPC gives good results over a wide range of the tuning parameters as a result it is easier to tune it.

Therefore, when dealing with relatively simple models that the linearized approximation of a nonlinear process can be acceptable, and when execution time is a priority, the LMPC appears to be the better choice. On the other hand, when speed does not matter much, and the model is believed to be less erroneous, but relatively complex or highly nonlinear where it will be difficult to accurately approximate it by a linear model, the NMPC can be better alternative as it can retain the nonlinearities of the process and is easier to implement and tune it.

10 Suggestions for future work

1. Investigating why the LMPC uses more F_{feed} to produce the same amount of F_{meth} as the NMPC. Trying another way of formulating the LMPC could probably be a solution to this, for example treating T_{reac} and $S_{vs,in}$ as input disturbances may help improve the LMPC.
2. As the results from the simulations of the AD reactor model show, it is vital having a constraint on the VFA consecration. Introducing a constraint on the state variable S_{vfa} in the case of the NMPC, is thus recommended.
3. It is recommended to research on reducing the excessively large computation time in the NMPC. Using paraller search with multis start option from the MATLAB toolbox could probably reduce the computation time in the NMPC.

Appendices

Overview of appendices

Appendix 1 Task Description

Appendix 2 MATLAB code for linear model predictive control

Appendix 3 MATLAB code for nonlinear model predictive control

Appendix 4 MATLAB code for Kalman filter

Appendix 5 MATLAB code for the objective function

Appendix 6 MATLAB code for linearization and discretization of AD reactor model

Appendix 7 MATLAB file for the LMPC applied to the AD reactor model

Appendix 8 MATLAB file for the NMPC applied to the AD reactor model

Appendix 9 MATLAB file for the objective function which is invoked by the fmincon in NMPC applied to the AD reactor

Appendix 10 Constraint function added to complete the NMPC applied to both air-heater model and AD reactor model. This function is invoked by the fmincon solver used in the NMPC.

Appendix 11 MATLAB code for the Kalman filter used for the AD reactor model

Appendix 12 MATLAB script for plotting LMPC and NMPC simulation results from the air-heater model

Appendix 13 MATLAB script for plotting LMPC and NMPC simulation results from the AD reactor model

Appendix 1

Task Description

Telemark University College
Faculty of Technology

FMH606 Master's Thesis

Title: *Evaluation and comparison of MPC algorithms applied to simulated processes*

TUC supervisor: Associate Professor Finn Haugen

External partner: -

Task description:

The main aim of the thesis is to evaluate and compare a number of model-based predictive control (MPC) algorithms as applied to a number of representative simulated processes.

Task background:

It is of interest to compare MPC algorithms with respect to implementation and use.

Student category:

SCE student.

Practical arrangements:

The thesis will be accomplished at TUC.

Signatures:

Student (date and signature): 13.02.'14

Supervisor (date and signature): Finn Haugen Feb 4, 2014

Thesis code: SIV-20-14

Address: Kjølnes ring 56, NO-3918 Porsgrunn, Norway. **Phone:** 35 57 50 00. **Fax:** 35 55 75 47.

Appendix 2

MATLAB code for linear model predictive control used for the air-heater model

```
function [Input_signal,Output_Temperature,
T_amb_array,Reference,tElapsed,it,ul,uu,e_mpc]=MPC_KF42(T_amb_init,std_T_ou
t_noise, ...
    K,K_real,T_const,T_const_real,Timedelay_real_process,
Timedelay_mpc,Ts,N,Np,Nsim,t,Set_Point)
%% Linear Model Predictive Controller for the Air Heater Model
%%-------------------------------------------------------------------------
% tic
% % clear all
% % close all
% clc
% -------------------------------------------------------------------------
%% Parameters
% K = 3.5; % heater gain
% T_const = 23; % time constant
p = [K, T_const];
% -------------------------------------------------------------------------
%% Parameters
% K_real = 3.5;
% T_const_real = 23;
p_real = [K_real, T_const_real];
% Ts=1;

% -------------------------------------------------------------------------
% %% Continous Time System Matrices
% Ac = [-K/T_const 0; 0 0] ;
% Bc = [K/T_const; 0];
% Cc = [1 1];
% Dc = 0;
%% Continous Time System Matrices
Ac = [-K/T_const, 0;0 0]  ;
Bc = [K/T_const;0] ;
Cc = [1 1];
Dc = 0;
% -------------------------------------------------------------------------
%% Discretizing the Continous time model
[A, B, C, D]=c2dm(Ac,Bc,Cc,Dc,Ts,'zoh');
% [Ad,Bd,Cd,Dd] = pade_approximation()
% A=blkdiag(Ad,1)
% B=[Bd;0]
% C=[Cd 0]
% D=Dd
% [Ad,Bd,Cd,Dd] = c2dm(A,B,C,D,Ts,'zoh')
%% Dimension of system matrices
nx = size(A,1);
nu = size(B,2);
ny = size(C,1);
%-------------------------------------------------------------------------
% N=10;
% Np=N/Ts;
% t_start=0;
% t_stop=600;
% Nsim=(t_stop-t_start)/Ts;
% t=[t_start:Ts:t_stop-Ts]';
%-------------------------------------------------------------------------
%Initial states:
T_heat_init = 4;
% T_amb_init = 21;
```

```
T_out_init = 25;
u0=1;
% std_T_out_noise=0;

%% Weighting Matrice
We = 10000;       % error weighting matrix
Wu = 7754;      % input weigting matrix
Wdu = 1;      % change of input weighting matrix
% ------------------------------------------------------------------------
% %Initial guessed optimal control sequence:
% u_guess=0*zeros(Np,1)+u0;
%-------------------------------------------------------------------------
%% State-space model implementing time-delay
% Timedelay_real_process=3;
nd=ceil(Timedelay_real_process/Ts);
Ad=diag([ones(nd-1,1)],-1);
Bd=[nd>=1;zeros(nd-1,1)];
Cd=[zeros(1,nd-1),nd>=1];
Dd=[nd==0];
x_delay_k=zeros(length(Ad),1)+u0;

%-------------------------------------------------------------------------
%% Kalman Filter Tuning
x_init=[T_heat_init, T_amb_init]';
x_apost_k_minus_1=x_init;
k_P=10;
P_init=diag((x_init*k_P).*(x_init*k_P));
P_apost_k_minus_1=P_init;
k_Q=5*diag([1 1]);
Q_cont=diag(x_init.*x_init)*k_Q^2;
Q=Q_cont;
T_out_noise_ukf=1.5;
R_cont=T_out_noise_ukf^2;
R=R_cont;%Likn (8.15) i Simon.
%-------------------------------------------------------------------------
%% Lower and upper bounds
ul = 0;
uu = 5;
dul = -inf;
duu = inf;
el= -inf;
eu = inf;
xl = -inf;
xu = inf;
yl=-inf;
yu=inf;
uLB = ones(N*nu,1)*ul;
uUB = ones(N*nu,1)*uu;
duLB = ones(N*nu,1)*dul;
duUB = ones(N*nu,1)*duu;
eLB = ones(N*ny,1)*el;
eUB = ones(N*ny,1)*eu;
yLB = -inf(N*ny,1);
yUB = inf(N*ny,1);
xLB = ones(N*nx,1)*xl;
xUB = ones(N*nx,1)*xu;

zl = [uLB; duLB; eLB; yLB; xLB];
zu = [uUB; duUB; eUB; yUB; xUB];
% ------------------------------------------------------------------------
%% Hessian matrix
H11 = kron(eye(N),Wu);
```

```
H22 = kron(eye(N),Wdu);
H33 = kron(eye(N),We);
H44 = zeros(N*ny);
H55 = zeros(N*nx);

H = blkdiag(H11, H22, H33, H44, H55);

c = zeros(1,(2*nu+2*ny+nx)*N);


%% Memory allocation
Output_Temperature=zeros(ny);
Input_signal=zeros(nu);
Reference=zeros(ny);
Control_signal=zeros(nu);
tElapsed=zeros(ny);
T_out=zeros(Nsim-Np,1)+T_out_init;
T_heat=zeros(Nsim-Np,1)+T_heat_init;
T_out_est=zeros(Nsim-Np,1)+T_out_init;
T_amb=zeros(Nsim-Np,1)+T_amb_init;
u=zeros(nu);
% ------------------------------------------------------------------------
%% For-loop for calculating optimal control sequence applied to simulated
process:
for k=1:Nsim-Np
    %% Reference
    %     r = 25;
    %     if k<=100 || k>=200
    T_out_sp=ones(length(t),1)*Set_Point;
    %     else
    %         T_out_sp=ones(length(t),1)*Set_Point+10;
    %
    %     end

    if k<=300
        T_amb(k)=T_amb_init;
    else
        T_amb(k)=T_amb_init+1;

    end


    %% Equality Constraints
    % measurement equation
    Ae1u = -kron(eye(N),D);
    Ae1du = zeros(N*ny,N*nu);
    Ae1e = zeros(N*ny,N*ny);
    Ae1dy = eye(N*ny);
    Ae1dx  = -kron(diag(ones(N-abs(-1),1),-1),C);

    % state equation
    Ae2u = -kron(eye(N),B);
    Ae2du = zeros(N*nx,N*nu);
    Ae2e = zeros(N*nx,N*ny);
    Ae2dy = zeros(N*nx,N*ny);
    Ae2dx = eye(N*nx)-kron(diag(ones(N-abs(-1),1),-1),A);

    % error equation
    Ae3u = zeros(N*ny,N*nu);
    Ae3du = zeros(N*ny,N*nu);
    Ae3e = eye(N*ny);
```

```
    Ae3dy = eye(N*ny);
    Ae3dx = zeros(N*ny,N*nx);

    % input equation
    Ae4u = kron(diag(ones(N-abs(-1),1),-1),eye(nu)) - eye(N*nu);
    Ae4du = eye(N*nu);
    Ae4e = zeros(N*nu,N*ny);
    Ae4dy = zeros(N*nu,N*ny);
    Ae4dx = zeros(N*nu,N*nx);
    %matrix Ae
    Ae = [Ae1u Ae1du Ae1e Ae1dy Ae1dx;
        Ae2u Ae2du Ae2e Ae2dy Ae2dx;
        Ae3u Ae3du Ae3e Ae3dy Ae3dx;
        Ae4u Ae4du Ae4e Ae4dy Ae4dx];

    % vector be
    be1 = [C*x_apost_k_minus_1; zeros(ny*(N-1),1)];
    be2 = [A*x_apost_k_minus_1 ; zeros((nx*(N-1)),1)];
    be3 = ones(ny*N,1)*T_out_sp(k);
    be4 = [-u0; zeros(nu*(N-1),1)];

    be = [be1; be2; be3; be4];

    %% Use Quadratic Programming
    tStart = tic;
    options = optimset('Display', 'off','LargeScale', 'on','MaxIter',10000,
'Algorithm', 'active-set');
    [z,fval,exitflag, output,lambda] =
quadprog(H,c,[],[],Ae,be,zl,zu,[],options);
    tElapsed(k) = toc(tStart);

    u(k)=z(1);
    %     u0=z(1);
    du(k)=z(11);

    %% Time Delay
    in_delay_real_process_k=u(k);
    x_delay_real_process_k_plus_1=Ad*x_delay_k+Bd*in_delay_real_process_k;
    out_delay_real_process_k=Cd*x_delay_k+Dd*in_delay_real_process_k;
    Control_signal(k)=out_delay_real_process_k;
    x_delay_k=x_delay_real_process_k_plus_1;

    %% Upadate state equation
    dT_heat_dt = -(1/T_const_real)*T_heat(k) +
(K_real/T_const_real)*Control_signal(k);
    T_heat(k+1) = T_heat(k) + Ts*dT_heat_dt;
    T_out(k)=T_heat(k) + T_amb(k);

    %-------------------------------------------------------------------------
----
    %% Store values for plotting
    % output
    it(k)=output.iterations;
    %    fc(k)=output.funcCount;
    %e_mpc(k)=T_out_sp(k)-T_out(k);
    Input_signal(k)=u(k);
    Reference(:,k)=T_out_sp(k);
    e_mpc(k) = Reference(k)-T_out(k);
    T_amb_array(:,k) =T_amb(k);
    T_heat_array(:,k)=T_heat(k);
    Output_Temperature(k)=T_out(k);
```

```
    %-------------------------------------------------------------------------
----
    %% Call Kalman Filter

    y_k=T_out(k)+std_T_out_noise*randn;
    control_signal_k=u(k);

    [P_apost_k,x_apost_k,y_pred_k,K_k]=...

Air_Heater_ukf(y_k,P_apost_k_minus_1,x_apost_k_minus_1,Ts,control_signal_k,
p,Q,R);
    T_out_est(k)=y_pred_k;
    % K_k

    %Time shift:
    x_apost_k_minus_1=x_apost_k;
    P_apost_k_minus_1=P_apost_k;

    T_heat(k)=x_apost_k(1);
    if k<=100 && k> 200
        T_amb(k)=x_apost_k(2);
    end
end
t_iae_s_init=0;%d
t_iae_s_final=300;%10;%d
index_iae_s=find(t>t_iae_s_init&t<t_iae_s_final);
iae_s_LMPC=Ts*sum(abs(e_mpc(index_iae_s)))

%
t_iae_d_init=300;%10;%d
t_iae_d_final=600;%17;%d
index_iae_d=find(t>t_iae_d_init&t<t_iae_d_final);
iae_d_LMPC=Ts*sum(abs(e_mpc(index_iae_d)))

%    tElapsed;
%    tAverage = sum(tElapsed)/(Nsim-Np)
%    tMax=max(tElapsed)
%    tMin=min(tElapsed)

%% Plotting:
t_plot=t(1:Nsim-Np);
t_plot_start=t_plot(1);
t_plot_stop=t_plot(end);
i_plot=[1:length(t_plot)];
% -------------------------------------------------------------------------
% close all
% figure(1)
% subplot(211)
% plot(t_plot,Input_signal(i_plot),'k-');%,'LineWidth',1.5);
% ylabel({'Control';'Signal';'in Volts'})
% xlabel('Time in seconds')
% hold on
% plot(t_plot,ul,'k:');
% hold on
% plot(t_plot,uu,'k:');
% % grid
% % axis([0 300 -1 6])
% title('Linear Model Predictive Control Simulation Results')
% legend('Control Signal','Lower and upper limits','Location','best')
% subplot(212)
```

```
% plot(t_plot,Output_Temperature(i_plot),'k-');%,'LineWidth',1);
% hold on
% ylabel({'Temperature';'At Tube Outlet';'in \circ C'})
% xlabel('Time in seconds')
% plot(t_plot,Reference(i_plot),'k-.');
%
% hold on
% plot(t_plot,T_amb_array(i_plot),'r-');
% legend('Measured Temeperature','Set Point','Location','best')
% % hold on
% % plot(t_plot,T_heat_array(i_plot),'c-');
% % legend('Measured Temeperature','Set Point','Location','best')
% grid minor
% axis([0 300 20 36])

% figure(2)
% plot(t_plot, tElapsed(i_plot),'k-')
% title({'Linear Model Predictive Control Simulation Results';' Execution
time Vs Number of cycles'})
% xlabel('Number of cycles')
% ylabel('Elapsed Time per cycle [s]')
%  axis([-5 300 0 1.2])
% % grid
% % figure(3)
% % plot(t_plot, fc(i_plot),'k*')
% % title({'Linear Model Predictive Control Simulation Results';'Maximum
Function Count Vs Number of cycles'})
% % xlabel('Number of cycles')
% % ylabel('Maximum Function Count')
% %  axis([-5 300 0 3])
% figure(3)
% plot(t_plot, it(i_plot),'k-')
% title({'Linear Model Predictive Control Simulation Results';'Maximum
Number of iterations Vs Number of cycles'})
% xlabel('Number of cycles')
% ylabel('Maximum Number of iterations')
%  axis([-5 300 0 8])
% toc
end
```

Appendix 3

MATLAB code for nonlinear model predictive control used for the air-heater model

```
function [u_mpc,T_out_mpc_array,T_amb_array,T_out_sp,tElapsed,fc,it,e_mpc]
=Air_Heater_NMPC(T_amb_init,std_T_out_noise,...

K,K_real,T_const,T_const_real,Timedelay_real_process,Timedelay_mpc,Ts,~,Np,
Nsim,t,Set_Point)
%% Nonlinear Model Predictive Controller for the Air Heater Model
%--------------------------------------------------------------------------
% tic
% clear all
% close all
% clc
%--------------------------------------------------------------------------
% K = 3.5;
```

```
% T_const = 23;
p = [K, T_const];
%--------------------------------------------------------------------------
% K_real = 3.5;
% T_const_real = 23;
% p_real = [K_real, T_const_real];

%--------------------------------------------------------------------------
% Ts=1; %Time-step
% t_pred_horizon=10;
% Np=t_pred_horizon/Ts;
% t_start=0;
% t_stop=600;
% Nsim=(t_stop-t_start)/Ts;
% t=[t_start:Ts:t_stop-Ts]';
%--------------------------------------------------------------------------
%Init states:
T_heat_init = 4;
% T_amb_init = 21;
T_out_init = 25;
Cotrol_signal_const = 1;
% std_T_out_noise=0;
%
%--------------------------------------------------------------------------
%% Weights
we=10000;
wu=7754;
wdu=1;
wN=0;
wde=0;
costs=[we wu wdu wN wde];
%--------------------------------------------------------------------------

%Initial guessed optimal control sequence:
u_const=Cotrol_signal_const;
u_guess=0*zeros(Np,1)+u_const;
%--------------------------------------------------------------------------
%State-space model implementing time-delay in "real" process for MPC:
% Timedelay_real_process=3;
nd_real_process=ceil(Timedelay_real_process/Ts);
Ad_real_process=diag([ones(nd_real_process-1,1)],-1);
Bd_real_process=[nd_real_process>=1;zeros(nd_real_process-1,1)];
Cd_real_process=[zeros(1,nd_real_process-1),nd_real_process>=1];
Dd_real_process=[nd_real_process==0];
x_delay_real_process_k=zeros(length(Ad_real_process),1)+u_const;

% %State-space model implementing time-delay in MPC:
% Timedelay_mpc=3;
nd_mpc=ceil(Timedelay_mpc/Ts);
Ad_mpc=diag([ones(nd_mpc-1,1)],-1);
Bd_mpc=[nd_mpc>=1;zeros(nd_mpc-1,1)];
Cd_mpc=[zeros(1,nd_mpc-1),nd_mpc>=1];
Dd_mpc=[nd_mpc==0];
x_delay_mpc_k=zeros(length(Ad_mpc),1)+u_const;
%--------------------------------------------------------------------------
%% Kalman filter tuning
x_init=[T_heat_init, T_amb_init]';
x_apost_k_minus_1=x_init;
k_P=10;
P_init=diag((x_init*k_P).*(x_init*k_P));
P_apost_k_minus_1=P_init;
k_Q=5*diag([1 1]);
```

```
Q_cont=diag(x_init.*x_init)*k_Q^2;
Q=Q_cont;
T_out_noise_ukf=1.5;
R_cont=T_out_noise_ukf^2;
R=R_cont;%Likn (8.15) i Simon.
%--------------------------------------------------------------------------
%% Memory allocation
u_mpc=zeros(Nsim-Np,1)+T_out_init;
T_out_mpc_array=zeros(Nsim-Np,1)+T_out_init;
T_out_sp_array=zeros(Nsim-Np,1)+T_out_init;
tElapsed=zeros(Nsim-Np,1)+T_out_init;
T_out=zeros(Nsim-Np,1)+T_out_init;
T_heat=zeros(Nsim-Np,1)+T_heat_init;
T_out_est=zeros(Nsim-Np,1)+T_out_init;
T_amb=zeros(Nsim-Np,1)+T_amb_init;
Control_signal=zeros(Nsim-Np,1)+u_const;
u=zeros(Nsim-Np,1)+u_const;
%--------------------------------------------------------------------------
%Matrices defining linear constraints for use in fmincon:
A=[];
B=[];
Aeq=[];
Beq=[];
%--------------------------------------------------------------------------
%Lower and upper limits of optim variable for use in fmincon:
lb=u_guess*0;
ub=u_guess*0+5;
% ul=0;
% uu=5;
%--------------------------------------------------------------------------
%% For-loop for calculating optimal control sequence applied to simulated
process:
for k=1:Nsim-Np
    %     %     t(k)
    %     if k<=100 || k>=200
    T_out_sp=ones(length(t),1)*Set_Point;
    %     else
    %         T_out_sp=ones(length(t),1)*Set_Point+10;
    %     end

    if k<=300
        T_amb(k)=T_amb_init;
    else
        T_amb(k)=T_amb_init+1;

    end
    %Updating future setpoint profile as time elapses:
    T_out_sp_to_optim=T_out_sp(k:k+Np);

    %Time-shift of state:
    x_mpc_init=x_apost_k_minus_1;


    tStart = tic;
    %Using fmincon for Calculating optimal future control sequence:
    optim_options=optimset('Algorithm','active-
set','LargeScale','on','MaxIter',10000,'MaxFunEvals',1000*length(u_guess),'
Display','off');
    [u_opt,fval,exitflag,output,lambda,grad,hessian] =...
        fmincon(@(u)
Air_Heater_objective(Ad_mpc,Bd_mpc,Cd_mpc,Dd_mpc,u,T_out_sp_to_optim,p,cost
```

```
s,x_mpc_init,Np,Ts),u_guess,A,B,Aeq,Beq,lb,ub,@Air_Heater_constraint,optim_
options);
    tElapsed(k) = toc(tStart);
    %     output(k)=output;
    %     toc
    %     k
    u_guess=u_opt; %Using optimal control sequence as guessed optim
solution in next iteration.
    u(k)=u_opt(1); %Applied controller output set as first sample of
optimal control sequence.


    %Applying optimal control signal to simulat "real" process:

    in_delay_real_process_k=u(k);

x_delay_real_process_k_plus_1=Ad_real_process*x_delay_real_process_k+Bd_rea
l_process*in_delay_real_process_k;

out_delay_real_process_k=Cd_real_process*x_delay_real_process_k+Dd_real_pro
cess*in_delay_real_process_k;
    Control_signal(k)=out_delay_real_process_k;
    x_delay_real_process_k=x_delay_real_process_k_plus_1;

    dT_heat_dt = -(1/T_const_real)*T_heat(k) +
(K_real/T_const_real)*Control_signal(k);

    T_heat(k+1) = T_heat(k) + Ts*dT_heat_dt;

    T_out(k)=T_heat(k) + T_amb(k);
    T_amb_array(k) =T_amb(k);
    e_mpc(k)=T_out_sp(k)-T_out(k);
    %     e_mpc(k)=T_out_sp(k)-T_out(k);
    %-------------------------------------------------------------------------
----------------------------------------
    %% Call Kalman Filter

    y_k=T_out(k)+std_T_out_noise*randn;
    Control_signal_k=Control_signal(k);

    [P_apost_k,x_apost_k,y_pred_k,K_k]=...

Air_Heater_ukf(y_k,P_apost_k_minus_1,x_apost_k_minus_1,Ts,Control_signal_k,
p,Q,R);
    T_out_est(k)=y_pred_k;
    %     K_k
    %Time shift:
    x_apost_k_minus_1=x_apost_k;
    P_apost_k_minus_1=P_apost_k;

    T_heat(k)=x_apost_k(1);
    T_amb(k)=x_apost_k(2);
    %-------------------------------------------------------------------------
    %% Store values for plotting
    u_mpc(k)=u(k);
    T_out_mpc_array(k)=T_out(k);
    T_out_sp_array(k)=T_out_sp(k);
    %     Output(k)=output(k);
    it(k)=output.iterations;
    fc(k)=output.funcCount;
```

```
    % Display Results

    % fprintf('Iterations:         \t%4.0f \n',Output.iterations);
    % fprintf('Function Count:       \t%4.0f\n',Output.funcCount);
    % fprintf('Output.lssteplength:\t%4.0f\n\n',Output.lssteplength);

end
t_iae_s_init=0;%d
t_iae_s_final=300;%10;%d
index_iae_s=find(t>t_iae_s_init&t<t_iae_s_final);
iae_s_NMPC=Ts*sum(abs(e_mpc(index_iae_s)))

%
t_iae_d_init=300;%10;%d
t_iae_d_final=600;%17;%d
index_iae_d=find(t>t_iae_d_init&t<t_iae_d_final);
iae_d_NMPC=Ts*sum(abs(e_mpc(index_iae_d)))

%    tAverage = sum(tElapsed)/(Nsim-Np)
%    tMax=max(tElapsed)
%    tMin=min(tElapsed)

%    Output.iterations;
%    fc=Output.funcCount
%    Output.lssteplength;
%    Output.stepsize;
%    Output.firstorderopt;
%    Output.constrviolation;
%    Output.message;
%

%    T_out
% %Plotting:
% t_plot=t(1:Nsim-Np);
% t_plot_start=t_plot(1);
% t_plot_stop=t_plot(end);
% i_plot=[1:length(t_plot)];
% close all
% figure(1)
% subplot(211)
% plot(t_plot,u_mpc(i_plot),'k-');
% ylabel({'Control';'Signal';'in Volts'})
% xlabel('Time in seconds')
% hold on
% plot(t_plot,ul,'k-.');
% hold on
% plot(t_plot,uu,'k-.');
% axis([0 300 -1 6])
% % grid
%
% % axis([0 60 -1 6])
% title('Nonlinear Model Predictive Control Simulation Results')
% legend('Applied Control Signal','Lower & Upper
Limit','Location','SouthEast')
% subplot(212)
% plot(t_plot,T_out_mpc_array(i_plot),'k-');
% hold on
% ylabel({'Temperature';'At Tube Outlet';'in \circ C'})
% xlabel('Time in seconds')
% plot(t_plot,T_out_sp_array(i_plot),'k-.');
% hold on
```

```
% plot(t_plot,T_amb_array(i_plot),'r-');
% legend('Measured Temeperature','Set Point','Location','best')
% grid minor

% axis([0 300 16 36])
% figure(2)
% plot(t_plot, tElapsed(i_plot),'k-')
% title({'Nonlinear Model Predictive Control Simulation Results';'
Execution time Vs Number of cycles'})
% xlabel('Number of cycles')
% ylabel('Elapsed Time per cycle [s]')
% %  axis([-5 300 0 3])
% % % grid
% figure(3)
% plot(t_plot, fc(i_plot),'k-')
% title({'Nonlinear Model Predictive Control Simulation Results';'Maximum
Function Count Vs Number of cycles'})
% xlabel('Number of cycles')
% ylabel('Maximum Function Count')
% %  axis([-5 300 0 355])
% figure(4)
% plot(t_plot, it(i_plot),'k-')
% title({'Nonlinear Model Predictive Control Simulation Results';'Maximum
Number of iterations Vs Number of cycles'})
% xlabel('Number of cycles')
% ylabel('Maximum Number of iterations')
% %  axis([-5 300 0 35])
% toc
end
```

Appendix 4

MATLAB code for Kalman filter used for the air-heater model

```
%% Unscented Kalman filter
function [P_apost_k,x_apost_k,y_pred_k,K_k]=...

Air_Heater_ukf(y_k,P_apost_k_minus_1,x_apost_k_minus_1,Ts,Control_signal_k,
p,Q,R)
% parameters
K=p(1);
T_const=p(2);

%-------Time updates:
%Calculation of sigma points
n=length(x_apost_k_minus_1);
m=length(y_k);
M1=real(sqrtm(n*P_apost_k_minus_1));
x_tilde_matrix_1=zeros(n,2*n);
for i=1:n
    x_tilde_matrix_1(:,i)=M1(i,:)';
    x_tilde_matrix_1(:,i+n)=-M1(i,:)';
end
x_sigma_i_k_minus_1_matrix=zeros(n,2*n);
for i=1:(2*n)

x_sigma_i_k_minus_1_matrix(:,i)=x_apost_k_minus_1+x_tilde_matrix_1(:,i);
```

```
end

%Transformation of sigma points using dynamic model
x_sigma_i_k_matrix=zeros(n,2*n);
for i=1:(2*n)
    x_sigma_i_k_minus_1=x_sigma_i_k_minus_1_matrix(:,i);
    T_heat_sigma_i_k_minus_1=x_sigma_i_k_minus_1(1);
    T_amb_sigma_i_k_minus_1=x_sigma_i_k_minus_1(2);
    T_heat_sigma_i_k =T_heat_sigma_i_k_minus_1 +...
        Ts*((-1/T_const)*T_heat_sigma_i_k_minus_1 +
(K/T_const)*Control_signal_k);
    T_amb_sigma_i_k=T_amb_sigma_i_k_minus_1+Ts*0;
    x_sigma_i_k_matrix(:,i)=[T_heat_sigma_i_k, T_amb_sigma_i_k ]';
end

%Calculation of apriori state estimate at time k
%x_apri_k=(1/(2*n))*(sum(x_sigma_k_vec'))';
x_apri_k=zeros(n,1);
for i=1:(2*n)
    x_apri_k=x_apri_k+(1/(2*n))*x_sigma_i_k_matrix(:,i);
end

%Calculation of apriori covariance at time k
P_apri_k=zeros(n,n);
for i=1:(2*n)
    P_apri_k=P_apri_k+(1/(2*n))*(x_sigma_i_k_matrix(:,i)-x_apri_k)*...
        (x_sigma_i_k_matrix(:,i)-x_apri_k)';
end
P_apri_k=P_apri_k+Q;

%Measurement-based updates:
%Calculation of sigma points
M2=real(sqrtm(n*P_apri_k));
x_tilde_matrix_2=zeros(n,2*n);
for i=1:n
    x_tilde_matrix_2(:,i)=M2(i,:)';
    x_tilde_matrix_2(:,i+n)=-M2(i,:)';
end
x_sigma_meas_update_i_k_matrix=zeros(n,2*n);
for i=1:(2*n)
    x_sigma_meas_update_i_k_matrix(:,i)=x_apri_k+x_tilde_matrix_2(:,i);
end

%Transformation of sigma points using the measurement equation
y_sigma_i_k_matrix=zeros(m,2*n);
for i=1:(2*n)
    x_sigma_meas_update_i_k=x_sigma_meas_update_i_k_matrix(:,i);
    T_heat_sigma_meas_update_i_k=x_sigma_meas_update_i_k(1);
    T_amb_sigma_meas_update_i_k=x_sigma_meas_update_i_k(2);
    T_out_sigma_i_k=T_heat_sigma_meas_update_i_k +
T_amb_sigma_meas_update_i_k;
    y_sigma_i_k_matrix(:,i)=[T_out_sigma_i_k]';
end

%Calculation of predicted measurement at time k
y_pred_k=zeros(m,1);
for i=1:(2*n)
    y_pred_k=y_pred_k+(1/(2*n))*y_sigma_i_k_matrix(:,i);
end

%Calculation of covariance of y at time k
```

```
P_y_k=zeros(m,m);
for i=1:(2*n)
    P_y_k=P_y_k+(1/(2*n))*(y_sigma_i_k_matrix(:,i)-y_pred_k)*...
        (y_sigma_i_k_matrix(:,i)-y_pred_k)';
end
P_y_k=P_y_k+R;
%Calculation of cross-covariance of x apri and y pred at time k
P_xy_k=zeros(n,m);
for i=1:(2*n)
    P_xy_k=P_xy_k+(1/(2*n))*(x_sigma_i_k_matrix(:,i)-x_apri_k)*...
        (y_sigma_i_k_matrix(:,i)-y_pred_k)';
end
%Calculation of Kalman filter gain and aposteriori estimates
K_k=P_xy_k/P_y_k;
x_apost_k=x_apri_k+K_k*(y_k-y_pred_k);
P_apost_k=P_apri_k-K_k*P_y_k*K_k';
end
```

Appendix 5

MATLAB code for the objective function of the air-heater model

```
function f =
Air_Heater_objective(Ad_mpc,Bd_mpc,Cd_mpc,Dd_mpc,u,T_out_sp,p,costs,x_mpc_i
nit,Np,Ts)

K=p(1);
T_const=p(2);

%Weights:
c_e=costs(1);
c_u=costs(2);
c_du=costs(3);
c_final=costs(4);
c_de=costs(5);

%Preallocation and initialization:
T_heat=zeros(1,Np)+x_mpc_init(1);
T_amb=zeros(1,Np)+x_mpc_init(2);
x_delay_mpc_k=zeros(length(Ad_mpc),1)+u(1);
Control_signal=zeros(length(Ad_mpc),1)+u(1);
T_out=zeros(1,Np);
e=zeros(1,Np);
J1=zeros(1,Np);
u_km1=u(1);
e_km1=e(1);
for k=1:Np
in_delay_mpc_k=u(k);
x_delay_mpc_k_plus_1=Ad_mpc*x_delay_mpc_k+Bd_mpc*in_delay_mpc_k;
out_delay_mpc_k=Cd_mpc*x_delay_mpc_k+Dd_mpc*in_delay_mpc_k;
Control_signal(k)=out_delay_mpc_k;
dT_heat_dt = -(1/T_const)*T_heat(k) + (K/T_const)*Control_signal(k);

T_heat(k+1) = T_heat(k) + Ts*dT_heat_dt;
T_out(k)=T_heat(k) + T_amb(k);

e(k)=T_out_sp(k)-T_out(k);
du_dt_k=(u(k)-u_km1)/Ts;
```

```
de_dt_k=(e(k)-e_km1)/Ts;
J1(k+1)=J1(k)+Ts*(c_e*e(k)*e(k)+c_u*u(k)*u(k)+c_du*du_dt_k*du_dt_k+c_de*de_
dt_k*de_dt_k);
u_km1=u(k);
e_km1=e(k);
x_delay_mpc_k=x_delay_mpc_k_plus_1;
end
J=J1(end)+c_final*e(end)*e(end);
f = J;
end
```

Appendix 6

MATLAB code for linearization and discretization of AD reactor model

```
function [A,B,C,D]=AD_reactor_Linearize_and_Discritize(x,Ffeed,Ts)
%% parameters
Af = 0.69;
b = 2.90;
B0 = 0.25;
k1 = 3.89;
k2 = 1.76;
k3 = 31.7;
k5 = 26.3;
Kd = 0.02;
Kdc = 0.02;
Ks = 15.5;
Ksc = 3;
V = 250;
Treac = 35;
um = 0.013*Treac-0.129;
umc = um;
%%
% x =[5.2155;1.0094;1.3128;0.3635;30.2];
% Ffeed=45;
% Ts=0.025;
%%
%Naming states
Sbvs = x(1);
Svfa = x(2);
Xacid = x(3);
Xmeth = x(4);
Svsin = x(5);
% A[(Sbvs*Xacid*k1*um/(Ks+Sbvs))-((Xacid*k1*um)/(Ks+Sbvs))-(Ffeed/V), 0, -
(Sbvs*k1*um)/(Ks+Sbvs),0,B0*Ffeed/V;
% linearizing the model
Ac =[((Sbvs)*(Xacid)*(k1)*(um))/((Ks) + (Sbvs))^2 -
((Xacid)*(k1)*(um))/((Ks) + (Sbvs)) - (Ffeed)/(V), 0, -
((Sbvs)*(k1)*(um))/((Ks) + (Sbvs)),0,((B0)*(Ffeed))/(V);
    ((Xacid)*(k2)*(um))/((Ks) + (Sbvs)) - ((Sbvs)*(Xacid)*(k2)*(um))/((Ks)
+ (Sbvs))^2, ((Svfa)*(Xmeth)*(k3)*(umc))/((Ksc) + (Svfa))^2 -
((Xmeth)*(k3)*(umc))/((Ksc) + (Svfa)) -
(Ffeed)/(V),((Sbvs)*(k2)*(um))/((Ks) + (Sbvs)), -((Svfa)*(k3)*(umc))/((Ksc)
+ (Svfa)), ((Af)*(B0)*(Ffeed))/(V);
    (Xacid)*((um)/((Ks) + (Sbvs)) - ((Sbvs)*(um))/((Ks) + (Sbvs))^2), 0,
((Sbvs)*(um))/((Ks) + (Sbvs)) - (Kd) - (Ffeed)/((V)*(b)),0, 0;
```

```
    0, (Xmeth)*((umc)/((Ksc) + (Svfa)) - ((Svfa)*(umc))/((Ksc) +
(Svfa))^2),0, ((Svfa)*(umc))/((Ksc) + (Svfa)) - (Kdc) - (Ffeed)/((V)*(b)),
0;
    0,  0,  0,  0, 0];


Bc = [ -((Sbvs) - (B0)*(Svsin))/(V); -((Svfa) - (Af)*(B0)*(Svsin))/(V);-
(Xacid)/((V)*(b)); -(Xmeth)/((V)*(b)); 0];






Cc =[ 0, (V*Xmeth*k5*umc)/(Ksc + Svfa) - (Svfa*V*Xmeth*k5*umc)/(Ksc +
Svfa)^2, 0, (Svfa*V*k5*umc)/(Ksc + Svfa), 0];
Dc=0;
% Discritizing the model
[A, B, C, D]=c2dm(Ac,Bc,Cc,Dc,Ts,'zoh');


end
```

Appendix 7

MATLAB file for the LMPC applied to the AD reactor model

```
function [ U,ul,uu,Y,R ] = AD_reactor_LMPC(  )
tic
clear all
clc
%-----------------------------------------------------

% x0=[5.81, 0.8672, 1.32, 0.39, 30.2]';
% x0 =[4.14;0.8;1.8;0.39;30.2];
% x0 =[4.14;0.8;1.8;150;30.2];
% x0=0.73*[3.6438, 0.7160, 1.9225, 0.3874, 30.2]';

% x0=[5.81 0.8672 1.32 0.39 30.8]
x0=[3.416868358475480   0.661329359704896   2.029485253109554
0.387435040175433,30.2]';%x_end with Fmeth = 150
% x =[5.2155;1.0094;1.3128;0.3635;30.2];
% Ffeed=35.3;
Ffeed= 28.190831337510776;%F_feed_end with Fmeth = 150;

Ts=0.025;

%linearizing and discritizing the model
[A,B,C,D]=AD_reactor_Linearize_and_Discritize(x0,Ffeed,Ts);
% Svsin = 30.2;
% Treact = 35;
% xs =[5.2155;1.0094;1.3128;0.3635];
% us=[Ffeed, Svsin, Treact];
% [A,B1,C,D]=linearizing_using_syms(x0,us,Ts)
% B=B1(:,1)
%%
```

```
% x0 = [1,1,1,1,1]';
x0 = [0,0,0,0,0]';
% x0 =[4.14;0.8;1.8;0.39;30.2];
%
x0=0.55*[3.416868358475480,0.661329359704896,2.029485253109554,0.3874350401
75433,0]';
nu = size(B,2);
nx = length(x0);
ny = size(C,1);
% Initial u
u0 = 20;
% n=16;
N=1;                          % Prediction horizon
Np=N/Ts;                      % Number of steps
t_start=0;
t_stop=16;
Nsim=(t_stop-t_start)/Ts;
t=[t_start:Ts:t_stop-Ts]';
%
P = 1e5;      % error weighting matrix
Q= 1;       % input weighting matrix
R=0;     % input increment weighting matrix

%-----------------------------------------------

F_feed_const= 28.190831337510776;
u_const=F_feed_const;
Timedelay_real_process=0.0;
nd_real_process=ceil(Timedelay_real_process/Ts);
Ad_real_process=diag([ones(nd_real_process-1,1)],-1);
Bd_real_process=[nd_real_process>=1;zeros(nd_real_process-1,1)];
Cd_real_process=[zeros(1,nd_real_process-1),nd_real_process>=1];
Dd_real_process=[nd_real_process==0];
x_delay_real_process_k=zeros(length(Ad_real_process),1)+u_const;
%
H1 = diag(Q*ones(1,N));
H2 = zeros(N*nx,N*nx);
H3 = diag(P*ones(1,N));
H4 = zeros(N,N);
H5 = diag(R*ones(1,N));
H = blkdiag(H1,H2,H3,H4,H5);
%
f = zeros(1,(2*nu+2*ny+nx)*N);
%
Ae11 = -kron(eye(N,N),B);
Ae12 = kron(eye(N,N),eye(nx,nx)) - kron(diag(ones(N-abs(-1),1),-1),A);
Ae22 = -kron(eye(N,N),C);
Ae24 = eye(N*ny,N*ny);
Ae33 = eye(N*ny,N*ny);
Ae34 = eye(N*ny,N*ny);
%

Aixu=zeros(N*nx,N*nu);
Aixx=[zeros(N*nx,N*1) ones(N*nx,N*1) zeros(N*nx,N*3)];
% Aixx=[zeros(N*nx,N*(nx-(nx-1))) ones(N*nx,N*(nx-(nx-1)))
zeros(N*nx,N*(nx-(nx-2)))];
Aixe=zeros(N*nx,N*ny);
Aixy=zeros(N*nx,N*ny);
Aixdu=zeros(N*nx,N*nu);

Aiyu = zeros(N*ny,N*nu);
Aiyx = zeros(N*ny,N*nx);
```

```
Aiye = zeros(N*ny,N*ny);
Aiyy = zeros(N*ny,N*ny);
Aiydu = zeros(N*ny,N*nu);

Aieu = zeros(N*ny,N*nu);
Aiex = zeros(N*ny,N*nx);
Aiee = zeros(N*ny,N*ny);
Aiey = zeros(N*ny,N*ny);
Aiedu = zeros(N*ny,N*nu);

Aiuu = zeros(N*nu,N*nu);
Aiux = zeros(N*nu,N*nx);
Aiue = zeros(N*nu,N*ny);
Aiuy = zeros(N*nu,N*ny);
Aiudu = zeros(N*nu,N*nu);

Ai=[Aixu Aixx Aixe Aixy Aixdu
    Aiyu Aiyx Aiye Aiyy Aiydu
    Aieu Aiex Aiee Aiey Aiedu
    Aiuu Aiux Aiue Aiuy Aiudu];

Ae = [Ae11,Ae12,zeros(nx*N,ny*N),zeros(nx*N,ny*N),zeros(N*nx,N*nu)
    zeros(ny*N,nu*N),Ae22,zeros(ny*N,ny*N),Ae24,zeros(N*ny,N*nu)
    zeros(ny*N,nu*N),zeros(ny*N,nx*N),Ae33,Ae34,zeros(N*ny,N*nu)
    kron(diag(ones(N-abs(-1),1),-1),eye(nu)) -
eye(N*nu),zeros(N*nu,N*nx),zeros(N*nu,N*ny),zeros(N*nu,N*ny),eye(N*nu)];
% Lower and upper bounds
ul = 0;
uu = 40;
dul = -inf;
duu = inf;
el= -inf;
eu = inf;
xl = -inf;
xu = inf;
yl=-inf;
yu=inf;
uLB = ones(N*nu,1)*ul;
uUB = ones(N*nu,1)*uu;
duLB = ones(N*nu,1)*dul;
duUB = ones(N*nu,1)*duu;
eLB = ones(N*ny,1)*el;
eUB = ones(N*ny,1)*eu;
yLB = -inf(N*ny,1);
yUB = inf(N*ny,1);
xLB = ones(N*nx,1)*xl;
xUB = ones(N*nx,1)*xu;

zl = [uLB; xLB; eLB; yLB;duLB];
zu = [uUB; xUB; eUB; yUB;duUB];

Y=zeros(ny);
X=zeros(nx);
R=zeros(N,1);
U=zeros(nu);

for k = 1:Nsim-Np
    %     k
    %      r0 =170;
    if k<=200 ||k>400
        r0=75;
```

```
    else
        r0=120;
    end

    % Present state and measurements
    x = x0;
    y = C*x0;

    rN = r0*ones(N,1);
    be = [A*x0;zeros(nx*(N-1),1);zeros(ny*N,1);rN;-u0; zeros(nu*(N-1),1)];
    bix=ones(N*nx,1)*0.8;
    %     bix=ones(N*1,1)*0.8;

    biy=zeros(N*ny,1);
    bie=zeros(N*ny,1);
    biu=zeros(N*nu,1);
    bi=[bix; biy; bie; biu];

    %   bi=[zeros(N*1,1);bix;zeros(N*3,1); biy; bie; biu];
    % Calculate present u
    options = optimset('Display', 'off','LargeScale', 'on','MaxIter',10000,
'Algorithm', 'active-set');
    U0 = quadprog(H,f,Ai,bi,Ae,be,zl,zu,[],options);
    u0 = U0(1:nu);
    u = u0;

    %Time delay:
    in_delay_real_process_k=u;

x_delay_real_process_k_plus_1=Ad_real_process*x_delay_real_process_k+Bd_rea
l_process*in_delay_real_process_k;

out_delay_real_process_k=Cd_real_process*x_delay_real_process_k+Dd_real_pro
cess*in_delay_real_process_k;
    ud=out_delay_real_process_k;
    x_delay_real_process_k=x_delay_real_process_k_plus_1;

    % Update state
    x0 = A*x + B*ud;
    % error
    e_mpc(k)=r0-y;

    % store values
    R(:,k)=rN;
    Y(:,k)=y;
    U(:,k)=u;
    X(:,k)=x(2);
end
t_iae_s_init=0;%d
t_iae_s_final=7;%10;%d
index_iae_s=find(t>t_iae_s_init&t<t_iae_s_final);
iae_s_mpc=Ts*sum(abs(e_mpc(index_iae_s)))

t_iae_d_init=7;%10;%d
t_iae_d_final=14;%17;%d
index_iae_d=find(t>t_iae_d_init&t<t_iae_d_final);
iae_d_mpc=Ts*sum(abs(e_mpc(index_iae_d)))

close all
t_plot=t(1:Nsim-Np);
```

```
t_plot_start=t_plot(1);
t_plot_stop=t_plot(end);
i_plot=[1:length(t_plot)];
figure(1)
subplot(211)
plot(t_plot,U(i_plot),'k-');%,'LineWidth',1.5);
ylabel({'F_{feed} [L/d]'})
xlabel('Time in days')
hold on
plot(t_plot,ul,'k:');
hold on
plot(t_plot,uu,'k:');
% % grid
% axis([0 7 -1 41])
title('Linear Model Predictive Control Simulation Results for the AD
reactor model')
legend('Control Signal','Lower and upper limits','Location','best')
subplot(212)
plot(t_plot,Y(i_plot),'k-');
hold on
ylabel({'F_{meth} [L/d]'})
xlabel('Time in days')
plot(t_plot,R(i_plot),'k-.');
legend('output','Set Point','Location','best')
figure(2)
plot(t_plot,X(i_plot),'k-');
ylabel({'S_{vfa}'})
xlabel('Time in days')
title('AD reactor Linear Model Predictive Control directly limiting state
variable S_{vfa}')



% index=1:Nsim-Np;
% t_plot=index*0.6/24;
% subplot(211)
% plot(t_plot,U,'k-');
% % ylabel('hg[m]')
% ylabel({'Gate';'Opening';'in meters'})
% hold on
% plot(t_plot,ul,'k-.');
% hold on
% plot(t_plot,uu,'k-.');
% title('Model Predictive Control Simulation Results')
%
% subplot(212)
% plot(t_plot,Y,'k-');
% hold on
% ylabel('h1[m]')
% ylabel({'Level at';'Merkebek';'in meters'})


% plot(time,R,'g-');
% hold on
% plot(time,YL,'r-');
% hold on
% plot(time,YU,'r-');

% subplot(513)
% plot(time,V_in,'*-');
% ylabel({'Inflow';'in cubic';'meters'})
```

```
%
% % ylabel('$\dot{Vin[m^3/s]}$')
%
% subplot(514)
% plot(time,V_t,'*-');
% ylabel({'Flow';'Through';'Turbine';'in cubic';'meters'})
%
% subplot(515)
% plot(time,V_g,'*-');
% ylabel({'Flow';'Through';'Gate in';'cubic';'meters'})
% xlabel('Time in Days')
% % % ------------------------------------------------------------------------
----
% close all
% figure(1)
% subplot(211)
% plot(,U,'k-');%,'LineWidth',1.5);
% ylabel({'Control';'Signal';'in Volts'})
% xlabel('Time in seconds')
% hold on
% plot(time,ul,'k:');
% hold on
% plot(time,uu,'k:');
% grid
% % axis([0 7 -1 41])
% title('Linear Model Predictive Control Simulation Results')
% legend('Control Signal','Lower and upper limits','Location','best')
% subplot(212)
% plot(time,Y,'k-');
% hold on
% ylabel({'output'})
% xlabel('Time in seconds')
% plot(time,R,'k-.');
% legend('output','Set Point','Location','best')
% figure(2)
% plot(time,X,'k-');
% ylabel({'Svfa'})
% xlabel('Time in seconds')
toc
```

Appendix 8

MATLAB script for the NMPC applied to the AD reactor model

```
function [ u_mpc,ul,uu,F_meth,r ] = AD_reactor_NMPC(  )

%----------------------------------
tic
clear all
close all
%----------------------------------
% parameters
b = 2.9;
K_s = 15.5;
K_sc = 3;
K_d = 0.02;
K_dc = 0.02;
V = 250;
k1 = 3.9;
```

```
k2 = 1.76;
k3 = 31.7;
k5 = 26.3;
Af=0.69;
B0=0.25;
p=[b,K_s,K_sc,K_d,K_dc,V,k1,k2,k3,k5,Af,B0];
%-------------------------------------------------------------------------
------
%parameters representing "real" process
b_real = 2.9;
K_s_real = 15.5;
K_sc_real = 3;
K_d_real = 0.02;
K_dc_real = 0.02;
V_real = 250;
k1_real = 3.9;
k2_real = 1.76;
k3_real = 31.7;
k5_real = 26.3;
Af_real=0.69;
B0_real=0.25;

p_real=[b_real,K_s_real,K_sc_real,K_d_real,K_dc_real,...
    V_real,k1_real,k2_real,k3_real,k5_real,Af_real,B0_real];
T_reac=35;
std_F_meth_noise=1.2*0;
%-----------------------------------------------
Ts=0.025;
t_pred_horizon=1;
Np=t_pred_horizon/Ts;
t_start=0;
t_stop=16;
Nsim=(t_stop-t_start)/Ts;
t=[t_start:Ts:t_stop-Ts]';
%---------------------------------------
%Initial states:
x_final=[3.416868358475480   0.661329359704896   2.029485253109554
0.387435040175433];%x_end with Fmeth = 150
% x_final = [1,1,1,1,1]';
S_bvs_init= x_final(1);
S_vfa_init= x_final(2);
X_acid_init = x_final(3);
X_meth_init = x_final(4);
F_meth_init = 150;
S_vs_in_init=30.2;
F_feed_const= 28.190831337510776;%F_feed_end with Fmeth = 150;
%Initial guessed optimal control sequence:
u_const=F_feed_const;
u_guess=0*zeros(Np,1)+u_const;
%-----------------------------------------------------
%State-space model implementing time-delay in "real" process:
Timedelay_real_process=0.0;
nd_real_process=ceil(Timedelay_real_process/Ts);
Ad_real_process=diag([ones(nd_real_process-1,1)],-1);
Bd_real_process=[nd_real_process>=1;zeros(nd_real_process-1,1)];
Cd_real_process=[zeros(1,nd_real_process-1),nd_real_process>=1];
Dd_real_process=[nd_real_process==0];
x_delay_real_process_k=zeros(length(Ad_real_process),1)+u_const;

%State-space model implementing time-delay in MPC:
Timedelay_mpc=0.2;
nd_mpc=ceil(Timedelay_mpc/Ts);
```

```
Ad_mpc=diag([ones(nd_mpc-1,1)],-1);
Bd_mpc=[nd_mpc>=1;zeros(nd_mpc-1,1)];
Cd_mpc=[zeros(1,nd_mpc-1),nd_mpc>=1];
Dd_mpc=[nd_mpc==0];
x_delay_mpc_k=zeros(length(Ad_mpc),1)+u_const;
%-------------------------------------------------------

F_meth_sp=ones(length(t),1)*F_meth_init;

S_vs_in=ones(length(t),1)*30.2;

%-------------------------------------------------

%Tuning of UKF:
x_init=[S_bvs_init,S_vfa_init,X_acid_init,X_meth_init,S_vs_in_init]';
x_apost_k_minus_1=x_init;

k_P=0.01;
P_init=diag((x_init*k_P).*(x_init*k_P));
P_apost_k_minus_1=P_init;

% k_Q=0.0005*diag([1 1 1 1 20]);
k_Q=0.0005*diag([10 1 1 1 10]);
Q_cont=diag(x_init.*x_init)*k_Q^2;
Q=Q_cont;
% Q_UKF_augm_vs_in=Q_UKF_augm_vs_in_cont*Ts;
std_F_meth_noise_ukf=1.2;
R_cont=std_F_meth_noise_ukf^2;
R=R_cont
%-----------------------------------------
%Preallocation of arrays:
S_bvs=zeros(Nsim-Np,1)+S_bvs_init;
S_vfa=zeros(Nsim-Np,1)+S_vfa_init;
X_acid=zeros(Nsim-Np,1)+X_acid_init;
X_meth=zeros(Nsim-Np,1)+X_meth_init;
F_meth=zeros(Nsim-Np,1)+F_meth_init;
F_meth_est=zeros(Nsim-Np,1)+F_meth_init;
%-----------------------------------------
%Matrices defining linear constraints for use in fmincon:
A=[];
B=[];
Aeq=[];
Beq=[];
%-----------------------------------------
%Lower and upper limits of optim variable for use in fmincon:
ul=0;
uu=40;
lb=u_guess*0;
ub=u_guess*0+40;
%-----------------------------------------
%MPC costs:
c_e=1;
c_u=0;
c_du=0.01;
c_final=0;
costs=[c_e c_u c_du c_final];
%-----------------------------------------

%For-loop for calculating optimal control sequence applied to simulated
process:
for k=1:Nsim-Np
```

```
    t(k)
    %   r0 =250;
    if k<=200 ||k>400
        F_meth_sp=ones(length(t),1)*(F_meth_init-F_meth_init/2);
        r(k)=F_meth_init-F_meth_init/2;
        %       r(k)=250;
    else
        F_meth_sp=ones(length(t),1)*120;
        r(k)=120;
    end

    %Updating future setpoint profile as time elapses:
    F_meth_sp_to_optim=F_meth_sp(k:k+Np);

    %Time-shift of state:
    x_mpc_init=x_apost_k_minus_1;


    %Using fmincon for Calculating optimal future control sequence:
    optim_options=optimset('Algorithm','active-
set','LargeScale','on','MaxIter',10000,'MaxFunEvals',1000*length(u_guess),'
Display','off');
    [u_opt,fval,exitflag,output,lambda,grad,hessian] =...

fmincon(@(u)AD_reactor_objective(Ad_mpc,Bd_mpc,Cd_mpc,Dd_mpc,u,T_reac,F_met
h_sp_to_optim,p,costs,x_mpc_init,Np,Ts),u_guess,A,B,Aeq,Beq,lb,ub,@AD_react
or_constraint,optim_options);

    u_guess=u_opt; %Using optimal control sequence as guessed optim
solution in next iteration.
    u(k)=u_opt(1); %Applied controller output set as first sample of
optimal control sequence.

    %Applying optimal control signal to simulat "real" process:
    mu_m=0.013*T_reac-0.129;
    mu_mc=mu_m;
    mu=mu_m/(K_s_real/S_bvs(k)+1);
    mu_c=mu_mc/(K_sc_real/S_vfa(k)+1);

    in_delay_real_process_k=u(k);

x_delay_real_process_k_plus_1=Ad_real_process*x_delay_real_process_k+Bd_rea
l_process*in_delay_real_process_k;

out_delay_real_process_k=Cd_real_process*x_delay_real_process_k+Dd_real_pro
cess*in_delay_real_process_k;
    F_feed(k)=out_delay_real_process_k;

    dS_bvs_dt=(B0_real*S_vs_in(k)-S_bvs(k))*F_feed(k)/V_real-
mu*k1_real*X_acid(k);
    dS_vfa_dt=(Af_real*B0_real*S_vs_in(k)-
S_vfa(k))*F_feed(k)/V_real+mu*k2_real*X_acid(k)-mu_c*k3_real*X_meth(k);
    dX_acid_dt=(mu-K_d_real-(F_feed(k)/b_real)/V_real)*X_acid(k);
    dX_meth_dt=(mu_c-K_dc_real-(F_feed(k)/b_real)/V_real)*X_meth(k)

    S_bvs(k+1)=S_bvs(k)+Ts*dS_bvs_dt;
    S_vfa(k+1)=S_vfa(k)+Ts*dS_vfa_dt;
    X_acid(k+1)=X_acid(k)+Ts*dX_acid_dt;
    X_meth(k+1)=X_meth(k)+Ts*dX_meth_dt;

    x_delay_real_process_k=x_delay_real_process_k_plus_1;
```

```
    % store values for plotting
    u_mpc(k)=u(k);
    F_meth(k)=V*k5_real*mu_c*X_meth(k);
    e_mpc(k)=F_meth_sp(k)-F_meth(k);

    %-------------------------------------------------------
    %Kalman Filter:

    y_k=F_meth(k)+std_F_meth_noise*randn;
    F_feed_k=F_feed(k);

    [P_apost_k,x_apost_k,y_pred_k,K_k]=...
AD_reactor_ukf(y_k,P_apost_k_minus_1,x_apost_k_minus_1,Ts,F_feed_k,p,T_reac
,Q,R);

    F_meth_est(k)=y_pred_k;
    %Time shift:
    x_apost_k_minus_1=x_apost_k;
    P_apost_k_minus_1=P_apost_k;
    K_k

    % S_bvs_est(k)=x_apost_k(1);
    % S_vfa_est(k)=x_apost_k(2);
    % X_acid_est(k)=x_apost_k(3);
    % X_meth_est(k)=x_apost_k(4);
    % S_vs_in_est(k)=x_apost_k(5);
    %-------------------------------------------------------
    % F_meth_mpc_array(k)=F_meth(k);
end
% %IAE:
t_iae_s_init=1;%d
t_iae_s_final=7;%10;%d
index_iae_s=find(t>t_iae_s_init&t<t_iae_s_final);
iae_s_mpc=Ts*sum(abs(e_mpc(index_iae_s)))

t_iae_d_init=7;%10;%d
t_iae_d_final=14;%17;%d
index_iae_d=find(t>t_iae_d_init&t<t_iae_d_final);
iae_d_mpc=Ts*sum(abs(e_mpc(index_iae_d)))
% %% Plotting:
% t_plot=t(1:Nsim-Np);
% t_plot_start=t_plot(1);
% t_plot_stop=t_plot(end);
% i_plot=[1:length(t_plot)];
% % ------------------------------------------------------------------------
--
% close all
% figure(1)
% subplot(211)
% plot(t_plot,u_mpc(i_plot),'k-');%,'LineWidth',1.5);
% ylabel({'Control';'Signal';'in Volts'})
% xlabel('Time in seconds')
% hold on
% plot(t_plot,ul,'k:');
% hold on
% plot(t_plot,uu,'k:');
% grid
% % axis([0 300 -1 6])
% title('Linear Model Predictive Control Simulation Results')
% legend('Control Signal','Lower and upper limits','Location','best')
```

```
% subplot(212)
% plot(t_plot,F_meth(i_plot),'k-');%,'LineWidth',1);
% hold on
% ylabel({'Temperature';'At Tube Outlet';'in \circ C'})
% xlabel('Time in seconds')
% plot(t_plot,r(i_plot),'k-.');
% % hold on
% % plot(t_plot,T_amb_array(i_plot),'k*');
% legend('Measured Temeperature','Set Point','Location','best')
% %-------------------------------------------------------
% figure(2)
% plot(t_plot,S_vfa(i_plot),'k-');
% ylabel({'S_{vfa}'})
% xlabel('Time in days')
% title('AD reactor Noninear Model Predictive Control')
% toc
end
```

Appendix 9

MATLAB file for the objective function which is invoked by the fmincon in NMPC applied to the AD reactor

```
function f =
AD_reactor_objective(Ad_mpc,Bd_mpc,Cd_mpc,Dd_mpc,u,T_reac,F_meth_sp,p,costs
,x_mpc_init,Np,Ts)
% p=[b,K_s,K_sc,K_d,K_dc,V,k1,k2,k3,k5,Af,B0];
b=p(1);
K_s=p(2);
K_sc=p(3);
K_d=p(4);
K_dc=p(5);
V=p(6);
k1=p(7);
k2=p(8);
k3=p(9);
k5=p(10);
Af=p(11);
B0=p(12);

%Weights:
c_e=costs(1);
c_u=costs(2);
c_du=costs(3);
c_final=costs(4);

%Preallocation and initialization:
S_bvs=zeros(1,Np)+x_mpc_init(1);
S_vfa=zeros(1,Np)+x_mpc_init(2);
X_acid=zeros(1,Np)+x_mpc_init(3);
X_meth=zeros(1,Np)+x_mpc_init(4);
S_vs_in=zeros(1,Np)+x_mpc_init(5);
x_delay_mpc_k=zeros(length(Ad_mpc),1)+u(1);

F_meth=zeros(1,Np);
F_feed=zeros(1,Np);
e=zeros(1,Np);
J1=zeros(1,Np);
```

```
u_km1=u(1);

for k=1:Np
    mu_m=0.013*T_reac-0.129;
    mu_mc=mu_m;
    mu=mu_m/(K_s/S_bvs(k)+1);
    mu_c=mu_mc/(K_sc/S_vfa(k)+1);

    in_delay_mpc_k=u(k);
    x_delay_mpc_k_plus_1=Ad_mpc*x_delay_mpc_k+Bd_mpc*in_delay_mpc_k;
    out_delay_mpc_k=Cd_mpc*x_delay_mpc_k+Dd_mpc*in_delay_mpc_k;
    F_feed(k)=out_delay_mpc_k;

    dS_bvs_dt=(B0*S_vs_in(k)-S_bvs(k))*F_feed(k)/V-mu*k1*X_acid(k);
    dS_vfa_dt=(Af*B0*S_vs_in(k)-S_vfa(k))*F_feed(k)/V+mu*k2*X_acid(k)-
mu_c*k3*X_meth(k);
    dX_acid_dt=(mu-K_d-(F_feed(k)/b)/V)*X_acid(k);
    dX_meth_dt=(mu_c-K_dc-(F_feed(k)/b)/V)*X_meth(k);

    S_bvs(k+1)=S_bvs(k)+Ts*dS_bvs_dt;
    S_vfa(k+1)=S_vfa(k)+Ts*dS_vfa_dt;
    X_acid(k+1)=X_acid(k)+Ts*dX_acid_dt;
    X_meth(k+1)=X_meth(k)+Ts*dX_meth_dt;

    F_meth(k)=V*k5*mu_c*X_meth(k);

    e(k)=F_meth_sp(k)-F_meth(k);
    du_dt_k=(u(k)-u_km1)/Ts;
    J1(k+1)=J1(k)+Ts*(c_e*e(k)*e(k)+c_u*u(k)*u(k)+c_du*du_dt_k*du_dt_k);
    u_km1=u(k);

    x_delay_mpc_k=x_delay_mpc_k_plus_1;
end
J=J1(end)+c_final*e(end)*e(end);
f = J;
end
```

Appendix 10

Constraint function added to complete the NMPC applied to both air-heater model and AD reactor model. This function is invoked by the fmincon solver used in the NMPC.

```
function [cineq,ceq]=AD_reactor_constraint(F_feed)
cineq =  [];
ceq = [];
end
```

Appendix 11

MATLAB code for the Kalman filter used for the AD reactor model

```
function [P_apost_k,x_apost_k,y_pred_k,K_k]=...
```

```
AD_reactor_ukf(y_k,P_apost_k_minus_1,x_apost_k_minus_1,Ts,F_feed_k,p,T_reac
,Q,R)
% p=[b,K_s,K_sc,K_d,K_dc,V,k1,k2,k3,k5,Af,B0];
b=p(1);
K_s=p(2);
K_sc=p(3);
K_d=p(4);
K_dc=p(5);
V=p(6);
k1=p(7);
k2=p(8);
k3=p(9);
k5=p(10);
Af=p(11);
B0=p(12);
%Calculation of sigma points
n=length(x_apost_k_minus_1);
m=length(y_k);
M1=real(sqrtm(n*P_apost_k_minus_1));
x_tilde_matrix_1=zeros(n,2*n);
for i=1:n
    x_tilde_matrix_1(:,i)=M1(i,:)';
    x_tilde_matrix_1(:,i+n)=-M1(i,:)';
end
x_sigma_i_k_minus_1_matrix=zeros(n,2*n);
for i=1:(2*n)

x_sigma_i_k_minus_1_matrix(:,i)=x_apost_k_minus_1+x_tilde_matrix_1(:,i);
end
%Transformation of sigma points using dynamic model
x_sigma_i_k_matrix=zeros(n,2*n);
for i=1:(2*n)
    x_sigma_i_k_minus_1=x_sigma_i_k_minus_1_matrix(:,i);

    S_bvs_sigma_i_k_minus_1=x_sigma_i_k_minus_1(1);
    S_vfa_sigma_i_k_minus_1=x_sigma_i_k_minus_1(2);
    X_acid_sigma_i_k_minus_1=x_sigma_i_k_minus_1(3);
    X_meth_sigma_i_k_minus_1=x_sigma_i_k_minus_1(4);
    S_vs_in_sigma_i_k_minus_1=x_sigma_i_k_minus_1(5);

    mu_m=0.013*T_reac-0.129;
    mu_mc=mu_m;
    mu=mu_m/(K_s/S_bvs_sigma_i_k_minus_1+1);
    mu_c=mu_mc/(K_sc/S_vfa_sigma_i_k_minus_1+1);

    VS_in_k_minus_1=B0*S_vs_in_sigma_i_k_minus_1;
    VFA_in_k_minus_1=Af*VS_in_k_minus_1;

    S_bvs_sigma_i_k=S_bvs_sigma_i_k_minus_1+...
        Ts*((VS_in_k_minus_1-S_bvs_sigma_i_k_minus_1)*F_feed_k/V-...
        mu*k1*X_acid_sigma_i_k_minus_1);
    S_vfa_sigma_i_k=S_vfa_sigma_i_k_minus_1+...
        Ts*((VFA_in_k_minus_1-S_vfa_sigma_i_k_minus_1)*F_feed_k/V+...
        mu*k2*X_acid_sigma_i_k_minus_1-mu_c*k3*X_meth_sigma_i_k_minus_1);
    X_acid_sigma_i_k=X_acid_sigma_i_k_minus_1+...
        Ts*((mu-K_d-(F_feed_k/b)/V)*X_acid_sigma_i_k_minus_1);
    X_meth_sigma_i_k=X_meth_sigma_i_k_minus_1+...
        Ts*((mu_c-K_dc-(F_feed_k/b)/V)*X_meth_sigma_i_k_minus_1);
    S_vs_in_sigma_i_k=S_vs_in_sigma_i_k_minus_1+Ts*0;

    x_sigma_i_k_matrix(:,i)=...
```

```
        [S_bvs_sigma_i_k,S_vfa_sigma_i_k,X_acid_sigma_i_k,X_meth_sigma_i_k
S_vs_in_sigma_i_k]';
end
%Calculation of apriori state estimate at time k
x_apri_k=zeros(n,1);
for i=1:(2*n)
    x_apri_k=x_apri_k+(1/(2*n))*x_sigma_i_k_matrix(:,i);
end
%Calculation of apriori covariance at time k
P_apri_k=zeros(n,n);
for i=1:(2*n)
    P_apri_k=P_apri_k+(1/(2*n))*(x_sigma_i_k_matrix(:,i)-x_apri_k)*...
        (x_sigma_i_k_matrix(:,i)-x_apri_k)';
end
P_apri_k=P_apri_k+Q;
%Calculation of sigma points:
M2=real(sqrtm(n*P_apri_k));
x_tilde_matrix_2=zeros(n,2*n);
for i=1:n
    x_tilde_matrix_2(:,i)=M2(i,:)';
    x_tilde_matrix_2(:,i+n)=-M2(i,:)';
end
x_sigma_meas_update_i_k_matrix=zeros(n,2*n);
for i=1:(2*n)
    x_sigma_meas_update_i_k_matrix(:,i)=x_apri_k+x_tilde_matrix_2(:,i);
end
%Transformation of sigma points using the measurement equation
y_sigma_i_k_matrix=zeros(m,2*n);
for i=1:(2*n)
    x_sigma_meas_update_i_k=x_sigma_meas_update_i_k_matrix(:,i);

    S_bvs_sigma_meas_update_i_k=x_sigma_meas_update_i_k(1);
    S_vfa_sigma_meas_update_i_k=x_sigma_meas_update_i_k(2);
    X_acid_sigma_meas_update_i_k=x_sigma_meas_update_i_k(3);
    X_meth_sigma_meas_update_i_k=x_sigma_meas_update_i_k(4);
    S_vs_in_sigma_meas_update_i_k=x_sigma_meas_update_i_k(5);

    mu_m=0.013*T_reac-0.129;
    mu_mc=mu_m;
    mu=mu_m/(K_s/S_bvs_sigma_meas_update_i_k+1);
    mu_c=mu_mc/(K_sc/S_vfa_sigma_meas_update_i_k+1)

    F_meth_sigma_i_k=V*k5*mu_c*X_meth_sigma_meas_update_i_k;

    y_sigma_i_k_matrix(:,i)=[F_meth_sigma_i_k]';
end
%Calculation of predicted measurement at time
y_pred_k=zeros(m,1);
for i=1:(2*n)
    y_pred_k=y_pred_k+(1/(2*n))*y_sigma_i_k_matrix(:,i);
end
%Calculation of covariance of y at time
P_y_k=zeros(m,m);
for i=1:(2*n)
    P_y_k=P_y_k+(1/(2*n))*(y_sigma_i_k_matrix(:,i)-y_pred_k)*...
        (y_sigma_i_k_matrix(:,i)-y_pred_k)';
end
P_y_k=P_y_k+R;
%Calculation of cross-covariance of x apri and y pred at time k
P_xy_k=zeros(n,m);
for i=1:(2*n)
    P_xy_k=P_xy_k+(1/(2*n))*(x_sigma_i_k_matrix(:,i)-x_apri_k)*...
```

```
        (y_sigma_i_k_matrix(:,i)-y_pred_k)';
end
%Calculation of Kalman filter gain and aposteriori estimates
K_k=P_xy_k/P_y_k;
x_apost_k=x_apri_k+K_k*(y_k-y_pred_k);
P_apost_k=P_apri_k-K_k*P_y_k*K_k';
end
```

Appendix 12

MATLAB script for plotting LMPC and NMPC simulation results from the air-heater model

```
clear all
close all
T_amb_init = 21;
std_T_out_noise=0;
K = 3.5;
T_const = 23;
%--------------------------------------------------------------------------
K_real = 3.5;
T_const_real = 23;
Timedelay_real_process=3;
Timedelay_mpc=3;

Ts=1;
t_pred_horizon=10;
Np=t_pred_horizon/Ts;
t_start=0;
t_stop=610;
Nsim=(t_stop-t_start)/Ts;
t=(t_start:Ts:t_stop-Ts)';
Set_Point=25;

[u_NMPC,T_out_NMPC,T_amb_array,T_out_sp,tElapsed_NMPC,fc_NMPC,it_NMPC,e_NMP
C] =Air_Heater_NMPC(T_amb_init,std_T_out_noise,...

K,K_real,T_const,T_const_real,Timedelay_real_process,Timedelay_mpc,Ts,t_pre
d_horizon,Np,Nsim,t,Set_Point);

[u_LMPC,Output_Temperature_LMPC,T_amb_LMPC,Reference,tElapsed_LMPC,it_LMPC,
ul,uu,e_LMPC]=Air_Heater_LMPC(T_amb_init,std_T_out_noise, ...

K,K_real,T_const,T_const_real,Timedelay_real_process,Timedelay_mpc,Ts,t_pre
d_horizon,Np,Nsim,t,Set_Point);
%--------------------------------------------------------------------------

%% Plotting:
t_plot=t(1:Nsim-Np);
t_plot_start=t_plot(1);
t_plot_stop=t_plot(end);
i_plot=1:length(t_plot);
% --------------------------------------------------------------------------
close all
figure(1)
subplot(211)
plot(t_plot,u_LMPC(i_plot),'b-');
```

```
hold on
plot(t_plot,u_NMPC(i_plot),'r-');
hold on
plot(t_plot,ul,'k:');
hold on
plot(t_plot,uu,'k:');
ylabel({'Control';'Signal';'in Volts'})
xlabel('Time in seconds')
legend('LMPC control signal','NMPC control signal','Lower & Upper
Limit','Location','East')
title({'Linear and nonlinear';' Model Predictive Control Simulation
Results';'For the air-heater model'})

subplot(212)
plot(t_plot,Output_Temperature_LMPC(i_plot),'b-');
hold on
plot(t_plot,T_amb_LMPC(i_plot),'g-');
hold on
plot(t_plot,T_out_NMPC(i_plot),'r-');
hold on
plot(t_plot,Reference(i_plot),'k-.');
% hold on
% plot(t_plot,36.31,'k:');
% hold on
% plot(t_plot,33.45,'k:');
% hold on
% plot(t_plot,33.2,'k:');
% hold on
% plot(t_plot,37.01,'k:');

ylabel({'Temperature';'At Tube Outlet';'in \circ C'})
xlabel('Time in seconds')
legend('LMPC output','Ambient Temperature','NMPC
output','Reference','Location','East')
% legend('LMPC output','NMPC output','Reference','Ambient
Temperature','Location','East')

% grid minor

figure(2)
subplot(211)
plot(t_plot, tElapsed_LMPC(i_plot),'b-')
hold on
plot(t_plot, tElapsed_NMPC(i_plot),'r-')

title({'Linear and nonlinear';' Model Predictive Control Simulation
Results';'For the air-heater model'})
legend('Execution time in LMPC','Execution time in NMPC','Location','best')
xlabel('Number of cycles')
ylabel('Elapsed Time per cycle [s]')
axis([-5 300 0 2])
subplot(212)
plot(t_plot,Reference(i_plot),'g-');
legend('Reference','Location','best')
ylabel({'Reference';'Temperature';'in \circ C'})
xlabel('Time in seconds')
% plot(t_plot,T_amb_LMPC(i_plot),'g-');
% legend('Ambient Temperature','Location','best')
% ylabel({'Ambient';'Temperature';'in \circ C'})
% xlabel('Time in seconds')
```

```
figure(3)
subplot(211)
plot(t_plot, it_LMPC(i_plot),'b-')
hold on
plot(t_plot, it_NMPC(i_plot),'r-')
%  axis([-5 300 0 40])
% hold on
% plot(t_plot,T_amb_LMPC(i_plot),'g-');
title({'Linear and nonlinear';' Model Predictive Control Simulation
Results';'For the air-heater model'})
legend('Iterations in LMPC','Iterations in NMPC','Location','best')
xlabel('Number of cycles')
ylabel('Maximum Number of iterations')
subplot(212)
% plot(t_plot,Reference(i_plot),'g-');
% legend('Reference','Location','best')
% ylabel({'Reference';'Temperature';'in \circ C'})
% xlabel('Time in seconds')

% plot(t_plot, it_NMPC(i_plot),'r-')
%  axis([-5 300 0 40])
% legend('Iterations in NMPC','Location','best')
% xlabel('Number of cycles')
% ylabel('Maximum Number of iterations')

plot(t_plot,T_amb_LMPC(i_plot),'g-');
legend('Ambient Temperature','Location','best')
ylabel({'Ambient';'Temperature';'in \circ C'})
xlabel('Time in seconds')
%  axis([-5 300 0 40])

% figure(4)
% % plot(t_plot, fc_LMPC(i_plot),'b-')
% % hold on
% plot(t_plot, it_NMPC(i_plot),'r-')
%
% title({'Nonlinear Model Predictive Control';' Simulation Results for the
air heater model';'Maximum Function Count Vs Number of cycles'})
% xlabel('Number of cycles')
% ylabel('Maximum Function Count')
figure(4)
plot(t_plot, e_LMPC(i_plot),'b-')
hold on
plot(t_plot, e_NMPC(i_plot),'r-')

title({'Linear and nonlinear';' Model Predictive Control Simulation
Results';'For the air-heater model'})
legend('error in LMPC','error in NMPC','Location','best')
xlabel('Time in seconds')
ylabel('Error ')
```

Appendix 13

MATLAB script for plotting LMPC and NMPC simulation results from the AD reactor model

```
% function [  ] = AD_reactor_LMPC_and_NMPC(  )

[ u_LMPC,ul,uu,F_meth_LMPC,Reference ] = AD_reactor_LMPC(  );
[ u_NMPC,ul,uu,F_meth_NMPC,r ] = AD_reactor_NMPC(  );

%% Plotting:
Ts=0.025;
t_pred_horizon=1;
Np=t_pred_horizon/Ts;
t_start=0;
t_stop=16;
Nsim=(t_stop-t_start)/Ts;
t=[t_start:Ts:t_stop-Ts]';
t_plot=t(1:Nsim-Np);
t_plot_start=t_plot(1);
t_plot_stop=t_plot(end);
i_plot=[1:length(t_plot)];
% ------------------------------------------------------------------------
close all
figure(1)
subplot(211)
plot(t_plot,u_LMPC(i_plot),'b-');
hold on
plot(t_plot,u_NMPC(i_plot),'r-');
ylabel({'F_{feed} [L/d]'})
xlabel('Time in days')
hold on
plot(t_plot,ul,'k-.');
hold on
plot(t_plot,uu,'k-.');
grid
% axis([0 300 -1 6])
title({'Linear and Nonlinear Model Predictive Control Simulation
Results';'for the AD reactor model'})
legend('LMPC','NMPC','Limits','Location','NorthEast')
subplot(212)
plot(t_plot,F_meth_LMPC(i_plot),'b-');
hold on
plot(t_plot,F_meth_NMPC(i_plot),'r-');
hold on
plot(t_plot,Reference(i_plot),'k-.');
ylabel({'F_{meth} [L/d]'})
xlabel('Time in days')
legend('LMPC','NMPC','Set Point','Location','NorthEast')
% end
```

References

Andrey Alexandrovich Tyagunov gtK, Rusland. (2004) High-Performance Model Predictive Control for Process Industry.

Bingfeng Gu YPG. (2008) Control of nonlinear processes by using linear model predictive control algorithms. *ScienceDirect*.

Chen CTaY. (2009) Linear and Nonlinear Model Predictive Control Using A General Purpose Optimal Control Problem Solver RIOTS 95. *IEEE Xplore.*

Dale E. Seborg TFE, Duncan A. Mellichamp. (2004) *Process Dynamics and Control*.

Ferreau HJ. (2011) Model Predictive Control Algorithms for Applications with Millisecond Timescales. Arenberg Doctoral School of Science, Engineering & Technology, Faculty of Engineering, Department of Electrical Engineering.

Finn Haugen RB, Bernt Lie. (2013) Adapting Dynamic Mathematical Model to a Pilot Anaerobic Digestion Reactor. *MIC* 34.

Finn Haugen RB, Bernt Lie. (2014) State Estimation and Model-based Control of a Pilot Anaerobic Digestion Reactor

Haugen F. (2010) Ziegler-Nichols' Closed-Loop Method.

Haugen F. (2012) *Reguleringsteknikk.*

Haugen F. (2014) Optimal Design Operation and Control of Anaerobic Digestion Reactor. Telemark University College.

Lie B. (2005) Modeling of Dynamic Systems.

Lie B. (2013) Predictive Control with Implementation(Lecture Note).

M. Morari JHL. (1997) *Model predictive control: Past, present and future.*

Maciejowski JM. (2002) *Predictive Control With Constraints*.

Mathworks. (2014a) *Iterative Desplay*. Available at: http://www.mathworks.com/help/optim/ug/iterative-display.html.

Mathworks. (2014b) *Quadratic Programming*. Available at: http://www.mathworks.com/discovery/quadratic-programming.html.

Rossiter JA. (2003) *Model Based Predictive Control: A practical approach*, USA: CRC Press.

Ruscio DD. (2001) Model Predictive Control and Optimization(Lecture Notes).

Ruscio DD. (2013) Optimal Model Based Control(Lecture Notes).

Simon D. (2006) *Optimal State Estimation*: Wiley.

So-Ryeok Oh J. (2010) Path followingofunderactuatedmarinesurfacevesselsusingline-of-sight based modelpredictivecontrol. *Ocean Engineering*.

Tallent É. (2012) *The forward (explicit) Euler method*. Available at: http://quantcorner.wordpress.com/2012/12/31/forward-explicit-euler-method/.

Wang L. (2009) *Model Predictive Control System Design and Implementation Using MATLAB*: Springer.

Weiguo Xie IB, Constantinos Theodoropoulos. (2011) Off-line model reduction for on-line linear MPC of nonlinear large-scale distributed systems. *Computers and Chemical Engineering*.

Yuebin Yu VL, Daihong Yu (2013) Multi-structural fast nonlinear model-based predictive control of a hydronic heating system. *Building and Environment*.