Tuning Methods for Model Predictive Controllers

Daniel H. Olesen

Kongens Lyngby 2012
IMM-M.Sc.-2012-69

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-M.Sc.-2012-69

Summary (English)

Model Predictive Control (MPC) is an optimal control strategy, and can be considered as an extension of the Linear Quadratic Gaussian Controller. It has become a popular control strategy in industry, since it provides a systematic approach in handling constraints on outputs and actuators.

The aim of this thesis has been to investigate tuning methods for ARIMAX-based predictive controllers. This class of controllers have been chosen because of the ability to obtain off-set free tracking in the face of constant disturbances.

We have evaluated different performance measures for a closed loop control system to asses deterministic, stochastic and robust performance. The measures has been used to develop a tuning toolbox for SISO systems, which visualizes the performance of control designs. A study has been performed in expressing performance measures for MIMO systems as scalar quantities. The derived measures has been used to define an optimization problem, which synthesize tunings based on deterministic and stochastic objectives with ensured robustness.

We have succesfully applied the developed methods for tuning of a Gas-Oil Furnace, a Wood-Berry Distillation Column and a Cement Mill Circuit.

Summary (Danish)

Model prædiktiv regulering udspringer fra optimal reguleringsteori og kan betragtes som en udvidelse af en lineær kvadratisk regulator. I industrien har denne styringsmetode vundet indpas, da regulatoren har en systematisk håndtering af begrænsninger for aktuatorer og outputs.

Dette projekt beskæftiger sig med tuning af ARIMAX baserede prædiktive regulatorer. Denne klasse af regulatorer er anvendt, siden de kan anvendes til at opnå en styring uden stationær fejl fra konstante proces forstyrrelser.

Vi har evalueret forskellige mål til at bedømme deterministisk, stokastisk og robust ydelse af lukket sløjfe reguleringssystemer. Målene har dannet rammen for udvikling af en tuning toolbox for SISO systemer, som kan visualisere ydelsen af regulator designs. Der er undersøgt, hvorledes ydelsesmål for MIMO systemer kan repræsenteres som skalare størrelser. På baggrund af dette, formuleres et optimeringsproblem til at generere tunings for deterministiske og stokastiske objektiver med en garanteret robusthed.

De udviklede metoder har succesfuldt kunne anvendes til at tune en Gas-Olie ovn, en Wood-Berry distillations kolonne og en cement mølle proces.

Preface

This M. Sc. thesis was prepared at the department of Informatics and Mathematical Modelling at the Technical University of Denmark in the period January 30th to June 29th of 2012. The thesis has been conducted under the supervision of Associative Professor John B. Jørgensen.

I would like to thank my supervisor for excellent guiding and valuable feedback throughout the project period.

Lyngby, 29-June-2012

Daniel H. Olesen

Nomenclature

J	Integrated Absolute Error for Reference Tracking
J_d	Integrated Absolute Error for Disturbance Rejection
M_S	Maximum Sensitivity
ARIMAX	Auto Regressive Integrated Moving Avererage with eXogeneous input
ARMAX	Auto Regressive Moving Average with eXogenuous input
ARX	Auto Regressive with eXogenuous input
DMC	Dynamic Matrix Control
DoF	Degrees of Freedom
IAE	Integrated Absolute Error
KKT	Karush-Kuhn-Tucker
LQE	Linear Quadratic Estimator
LQG	Linear Quadratic Gaussian
LTI	Linear Time Invariant
MIMO	Multiple Inputs Multiple Outputs
MISO	Multiple Inputs Single Output
MPC	Model Predictive Control

MPHC	Model Predictive Heuristic Control
NLP	Non-Linear Program
PSO	Particle Swarm Optimization
QP	Quadratic Program
SISO	Single Input Single Output
SQP	Sequential Quadratic Programming
SVD	Singular Value Decomposition

Contents

Part I

Theory

Chapter 1

Introduction

Model Predictive Control (MPC) has evolved to become an industrial standard in advanced process control [QB03]. The control strategy can be considered to be an extension of the Linear Quadratic Gaussian (LQG) controllers developed in the 1960s by the work of Kalman *et al.* [QB03]. The LQG controller is a combination of a Linear Quadratic Estimator (Kalman filter) and a Linear Quadratic Regulator. Originally, LQG controllers was used in the aerospace industry, where it could be justified economically and physically to develop accurate models for the controllers. In the process industry LQG made less of an impact since it initially was not possible to handle constraints, ensure robustness and make use of unique performance criterions. In addition it was not clear how to identify sufficient accurate models from data.

The first recognized MPC algorithm was described by Richalet *et al.* in 1976 and was called Model Predictive Heuristic Control (MPHC) [RRTP76]. The formulation featured an on-line optimization of a quadratic performance index with a finite prediction horizon and implicit handling of input and output constraints. Independently of MPHC, engineers at Shell-Oil made their own MPC version termed Dynamic Matrix Control (DMC) [CR79]. Similar to MPHC, it featured a quadratic performance index with a finite prediction horizon. MPHC and DMC represents the first generation of MPC and both had a major impact on process control in industry [QB03]. The developement of MPC is acknowledged to have been industry-driven, and the number of implementations in industry have grown rapidly over the years. By the year 2003 more than 4500 industrial MPC applications had been implemented [QB03]. Aided by the increasing processing power of modern hardware and more efficient algorithms, MPC is no longer restricted to slow industrial processes, and can now be used to control faster systems.

1.1 Background of the Project

Despite the growing popularity of MPC, a systematic tuning practice has not evolved, and only few guidelines exist. The topic has not been short of research, as there are numerous academic publications on the subject [GS10]. A number of tuning rules have been proposed [HC94] [SC98] [TF03], but no one seems to have made a significant impact. Proposed methods have typically been limited to theoretical case-studies on specific systems and provides good advice for that particular class of systems, but only little or no advice in any other case. An interesting proposal has been to synthesize a MPC from a prototype linear controller, and allowing classical tuning tools to be used [CB10].

Our aim for this project has been to develop a methodology in relation to tuning of a MPC system. The main objectives can be summarized to be:

- Derive a closed loop description for an unconstrained MPC application.
- Investigate methods to asses closed loop performance in relation to the deterministic, stochastic and robust properties of the system.
- Develop a methodology for tuning of an ARIMAX-based MPC.

An important concept in this study has been to express an unconstrained MPC as a 2-DoF Linear Time Invariant (LTI) controller and derive a state space model for the controller. Garcia and Morari demonstrated how MPHC and DMC can be decomposed as a 2-DoF controller in relation to Internal Model Control [GM82]. They have further demonstrated how a transfer function description of the controller is obtained from the weigthed least squares problem, that forms an unconstrained MPC [GM85].

This study relies on a closed loop description of the controller and process for analysis of system performance. Similar approaches have been used [PSQ02] and [LY94]. Shah *et al.* proposed methods to asses the closed loop performance of a MPC using a closed loop model [PSQ02]. Lee *et al.* proposed to use a closed loop description for synthesis of a MPC by application of robust design techniques [LY94]. Another example of application of robust techniques have been to tune a MPC using H_∞ Loop Shaping [RM00].

The controller synthesis we propose do not incorporate robust methods directly. A simpler approach by evaluating maximum sensitivity as a robustness indicator has been used. We have required that the MPC algorithm should be able to provide off-set free tracking for constant disturbances. This has been obtained by using an algorithm proposed by Jørgensen *et al.* [JHR11]. The algorithm uses an Auto Regressive Integrated Moving Avererage with eXogeneous input (ARIMAX) based observer model. It further features a correct closed-loop expression for state space models in innovation form.

We have aimed to exploit the processing power of modern hardware by evaluating a large number of tuning configuations for the closed loop system using different performance indicators. For a Single Input Single Output (SISO) system, we have developed a Toolbox, which can be used to visualize the closed loop performance for a given evaluation range.

For Multiple Inputs Multiple Outputs (MIMO) systems it was not possible to provide a visual perspective to the tuning without reducing the degrees of freedom for the tuning variables. We use an optimization-based approach for tuning of MIMO systems. The application of optimization theory as a tuning tool is not a new approach in itself. It has been suggested to construct an objective function for rise time, overshoot, settling time and steady state error and use Paticle Swarm Optimization (PSO) to generate the tuning parameters [SKN+12]. Another approach is defining an optimization problem, which aims to minimize the residuals of the desired and actual responses of the control loop [GE11].

Our use of optimization shares some resemblance with the mentioned studies. The objectives has however been defined differently and we use a bound on maximum sensitivity to ensure robust performance. Stochastic properties has further been taken into account and can be used as an objective of the optimization. A strong incentive for the optimization approach is that it could form a basis for an auto tuning scheme.

The challenge in obtaining good tunings for a control systems should always remain a high priority, since even small improvements in throughput can be worth millions in yearly earnings for certain plants. Tuning is further important in relation to maintenance and plant life, since process actuators degrades over time and use.

1.2 Structure of the Thesis

A brief overview of the chapters forming the thesis is given below.

Model Predictive Control: We introduce a general MPC algorithm. A Kalman filter is stated for estimation of process states, and calculation of the optimal control signal is derived for an unconstrained MPC. We demonstrate how a state space formulation of a MPC can be derived.

ARIMAX based MPC: We introduce a SISO ARIMAX based observer model. It is demonstrated how the ARIMAX based model is converted to a state space model in innovation form. The model is further extended to MIMO systems.

Closed Loop Analysis: A closed-loop state space model is derived from the basis of previous chapters. We conduct a survey of closed-loop evaluation methods, which include sensitivity analysis and steady state covariance calculations using discrete Lyapunov equations.

Tuning of SISO Systems: We evaluate performance assesment methods for a control system using the closed-loop state space description for the ARIMAX-based MPC. Performance criterions are defined for deterministic, stochastic and robust performance. A SISO-tuning toolbox based on the criterions is developed and described.

Tuning of MIMO Systems: The performance assesment methods for SISO systems is extended to MIMO systems. An optimization based tuning procedure is proposed with deterministic and stochastic objectives with ensured robustness.

Gas-Oil Furnace: A SISO Gas-Oil Furnace has been used as a case study. We propose a tuning based on the developed methods. We further discuss the performance limitations for this system.

Wood-Berry Distillation Column: We apply and analyze optimization-based tunings for a Wood-Berry distillation column.

Cement Mill: This case study considers a industrial cement mill process. We propose a tuning for this system on basis of the optimization tuning procedure.

Chapter 2

Model Predictive Control

In this chapter, we present an algorithm for MPC. We demonstrate how a predictive Kalman filter can be used to estimate the states of the control object. The calculation of the optimal control signal trajectory is derived from the solution of a constrained optimization problem. Finally, we derive a state space model of the controller.

2.1 Introduction to MPC

It is assumed that, the process to be controlled can be described from the LTI state space model:

$$x_{k+1} = Ax_k + Bu_k + Gw_k + Ed_k \tag{2.1a}$$
$$y_k = C_y x_k + v_k \tag{2.1b}$$
$$z_k = C_z x_k \tag{2.1c}$$

x_k denotes the internal states of the system, u_k is the process inputs, w_k is process noise and d_k is a disturbance to the system. y_k is the measured outputs, and is influenced by sensornoise v_k. z_k is the outputs to be controlled.

The initial states is described from $x_0 \sim N(\hat{x}_{0|-1}, P_{0|-1})$, w_k and v_k are stochastic variables described by:

$$\begin{bmatrix} w_k \\ v_k \end{bmatrix} \sim N_{iid}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} R_{ww} & R_{wv} \\ R_{wv}^T & R_{vv} \end{bmatrix} \right) \tag{2.2}$$

The description of the process (2.1) assumes that the internal process states x_k are known. For most physical systems the states can not be measured and known exactly. Expected values of the states $\hat{x}_k$ and controlled outputs $\hat{z}_k$ has to be estimated from the measured outputs. We assume that disturbances to the process can not be measured. The notation of the MPC control law is taken from [JHR11]:

$$\min_{\{u_{k+j}\}_{j=0}^{N-1}} \phi = \frac{1}{2} \sum_{j=0}^{N-1} \left\| \hat{z}_{k+1+j|k} - r_{k+1+j|k} \right\|_{Q_z}^2 + \left\| \Delta u_{k+j} \right\|_S^2 \tag{2.3}$$

s.t.

$$\hat{x}_{k+1|k} = \hat{A}\hat{x}_{k|k} + \hat{B}u_{k|k} + \hat{G}\hat{w}_{k|k} \tag{2.4a}$$

$$\hat{x}_{k+1+j|k+j} = \hat{A}\hat{x}_{k+j|k+j-1} + \hat{B}u_{k+j|k} \quad (j \geq 1) \tag{2.4b}$$

$$\hat{z}_{k+j|k} = \hat{C}_z\hat{x}_{k+j|k} \tag{2.4c}$$

$$u_{min} \leq u_{k+j|k} \leq u_{max} \tag{2.5a}$$

$$\Delta u_{min} \leq \Delta u_{k+j|k} \leq \Delta u_{max} \tag{2.5b}$$

$$z_{min} \leq \hat{z}_{k+j|k} \leq z_{max} \tag{2.5c}$$

The optimal control signal trajectory is obtained by the solution of a constrained optimization problem (2.3). Two terms in the objective funtion ϕ are penalized; tracking errors and control signal movement. Each term is weighted by diagonal matrices Q_z and S. The constraints are given from a prediction model of the control object (2.4) and constraints imposed on actuators and outputs (2.5).

In (2.3) the most recent output is taken into account, while future outputs has to be predicted $N-1$ steps ahead using the internal prediction model $(\hat{A}, \hat{B}, \hat{C}_z, \hat{G})$. The prediction model is based on an estimate $\hat{x}_{k|k}$ of the process states, and an estimate of the process noise influencing the system $\hat{w}_{k|k}$. The predicted states

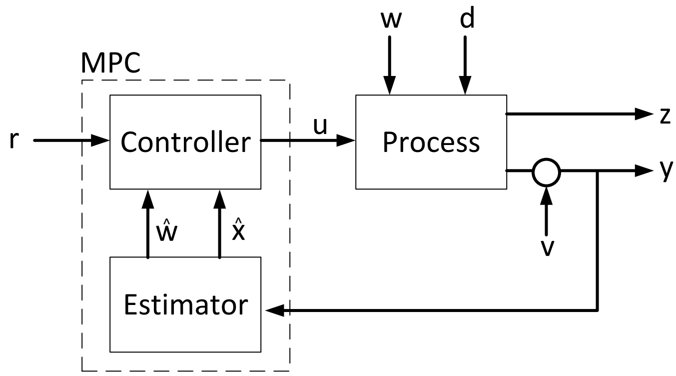


Figure 2.1: MPC control loop. The MPC consists of a controller and an estimator. The state estimates is obtained from the measured output of the process. The state estimates are used by the controller to predict future outputs and calculate an optimal control signal trajectory.

is calculated from the measured outputs y of the process, as shown in Figure 2.1. The procedure is repeated on each sample instant. MPC is also called Receding Horizon Control. This name describes that the prediction horizon is constant in length, but shifted in time and recalculated for each iteration.

Figure 2.2 shows the concept of MPC. The control signal calculated for $j = 0$ is applied to the process. If the predicted and future outputs is identical, the control signal trajectory $\{u_{k+j|k}\}_{j=0}^{N-1}$ is applied as the control signal $\{u_k\}_{k=0}^{N-1}$ at the respective occurences. This requires that the prediction horizon is chosen sufficiently long to emulate an infinite horizon controller. It can not be expected that predicted and future values is the same, due to noise, disturbances and modelling uncertainties.

2.2 State Estimation

It is common to use a Kalman filter as an estimator (observer) for the MPC. The Kalman filter is a Linear Quadratic Estimator (LQE), since the objective is to minimize the sum of squared errors between state values and estimated state values. The filter is regarded as an optimal observer if the estimation model matches the true system, the noise sources are white and the covariances of the

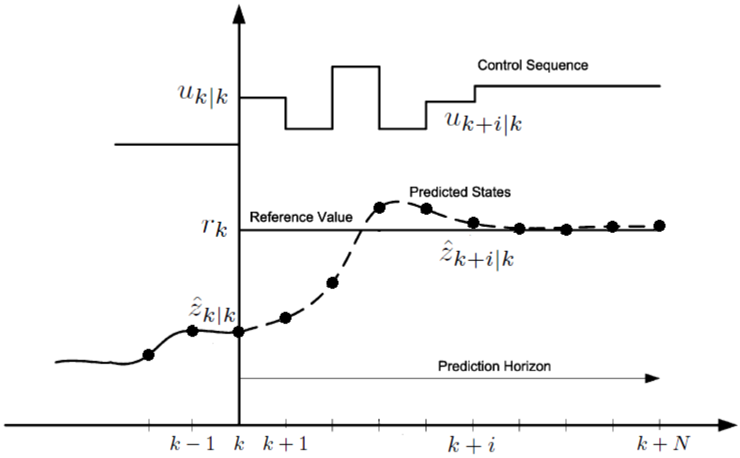


Figure 2.2: Concept of Predictive Control. The control signal sequence $\{u_{k+j|k}\}_{j=0}^{N-1}$ is calculated from the current output $\hat{z}_{k|k}$. The solid line for the controlled output indicates past outputs, and the dashed line indicates future values. *Modified Figure from [ARF11]*

noise sources are exactly known.

The Kalman filter is presented in a form where time and data update are separated. The notation is also known as the predictive Kalman filter, and is presented in a stationary version. The recursions are listed as:

Data update:

$$e_k = y_k - \hat{y}_{k|k-1} \tag{2.6a}$$
$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_{fx} e_k \tag{2.6b}$$
$$\hat{w}_{k|k} = K_{fw} e_k \tag{2.6c}$$

Time update:

$$\hat{x}_{k+1|k} = \hat{A}\hat{x}_{k|k} + \hat{B}u_k + \hat{G}\hat{w}_{k|k} \tag{2.6d}$$
$$\hat{y}_{k+1|k} = \hat{C}_y \hat{x}_{k+1|k} \tag{2.6e}$$
$$\hat{z}_{k+1|k} = \hat{C}_z \hat{x}_{k+1|k} \tag{2.6f}$$

K_{fx} determines how the measurements are weighted compared to predicted values. K_{fw} is zero if measurement noise and process noise are uncorrelated. K_{fw} is often neglected in the filter description, since it is common to assume that the noise sources are uncorrelated. For most physical models this is a reasonable assumption. However for systems in innovation form, there is perfect correlation between the process and measurement noise. This is the case for ARIMAX models, which will be discussed in a later section.

The estimation uncertainty (i.e. covariance) is calculated from the discrete algebraic Ricatti equation:

$$P = \hat{A}P\hat{A}^T + \hat{G}R_{ww}\hat{G}^T - (\hat{A}P\hat{C}_y^T + \hat{G}R_{wv})(\hat{C}_y P\hat{C}_y^T + R_{vv})^{-1}(\hat{A}P\hat{C}_y^T + \hat{G}R_{wv})^T \tag{2.7}$$

The solution of this equation is essential for calculating the Kalman gains:

$$K_{fx} = P\hat{C}_y^T(\hat{C}_y P\hat{C}_y^T + R_{vv})^{-1} \tag{2.8a}$$

$$K_{fw} = R_{wv}(\hat{C}_y P\hat{C}_y^T + R_{vv})^{-1} \tag{2.8b}$$

It should be apparent, that the quality of the state estimates is dependent on the accuracy of the estimation model ($\hat{A}$,$\hat{B}$,$\hat{C}_y$,$\hat{C}_z$,$\hat{G}$). A logical choice of the estimation model could be to select ($\hat{A}$,$\hat{B}$,$\hat{C}_y$,$\hat{C}_z$,$\hat{G}$) as the process model (A,B,C_y,C_z,G). For ARMIAX models, we will later use a modified process description for the estimator. The Kalman filter works optimally if the residuals of the estimation error (innovation sequence) is white.

2.3 Computation of Control Signal

The control input trajectory should be calculated such that (2.3) is minimized with respect to the imposed constraints. We assume, that only dynamic constraints are active (equality constraints), ie. no limitations on output and actuators (inequality constraints) exists. The assumptions are subsequently reffered to as unconstrained MPC. Furthermore, for simplicity we assume that $\hat{C} = \hat{C}_y = \hat{C}_z$, such that $\hat{z}_{k|k} = \hat{y}_{k|k}$. The MPC algorithm can be expressed as an estimation and regulation problem:

Estimator (Kalman filter):

$$e_k = y_k - \hat{y}_{k|k-1} \tag{2.9}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_{fx} e_k \tag{2.10}$$

$$\hat{w}_{k|k} = K_{fw} e_k \tag{2.11}$$

Regulator:

$$\min_{\{u_{k+j}\}_{j=0}^{N-1}} \phi = \frac{1}{2} \sum_{j=0}^{N-1} \left\| \hat{y}_{k+1+j|k} - r_{k+1+j|k} \right\|_{Q_z}^2 + \left\| \Delta u_{k+j} \right\|_S^2 \tag{2.12}$$

s.t.

$$\hat{x}_{k+1|k} = \hat{A}\hat{x}_{k|k} + \hat{B}u_{k|k} + \hat{G}\hat{w}_{k|k} \tag{2.13a}$$

$$\hat{x}_{k+1+j|k+j} = \hat{A}\hat{x}_{k+j|k+j-1} + \hat{B}u_{k+j|k} \quad (j \geq 1) \tag{2.13b}$$

$$\hat{y}_{k+1+j|k+j} = \hat{C}\hat{x}_{k+1+j|k+j} \tag{2.13c}$$

The regulation problem is a constrained quadratic optimization problem. The solution for this type of problem normally requires a set of conditions to be fullfilled, referred to as the Karush-Kuhn-Tucker (KKT) conditions. We use a method, which do not explictly use the KKT conditions in the solution due to state elimination. In Appendix B the equivalence between the conventional solution and the method to be used is shown.

We can express the MPC objective function as:

$$\phi = \frac{1}{2}\left\|Y_k - R_k\right\|_{\mathcal{Q}}^2 + \frac{1}{2}\left\|\Delta U_k\right\|_{\mathcal{S}}^2 \tag{2.14}$$

where

$$Y_k = \begin{bmatrix} \hat{y}_{k+1|k} \\ \hat{y}_{k+2|k} \\ \vdots \\ \hat{y}_{k+N|k} \end{bmatrix}, \ R_k = \begin{bmatrix} r_{k+1|k} \\ r_{k+2|k} \\ \vdots \\ r_{k+N|k} \end{bmatrix}, \ \Delta U_k = \begin{bmatrix} u_{k|k} - u_{k-1|k} \\ u_{k+1|k} - u_{k|k} \\ \vdots \\ u_{k+N-1|k} - u_{k+N-2|k} \end{bmatrix} \tag{2.15a}$$

$$\mathcal{Q} = \begin{bmatrix} Q_z & & & \\ & Q_z & & \\ & & \ddots & \\ & & & Q_z \end{bmatrix}, \ \mathcal{S} = \begin{bmatrix} S & & & \\ & S & & \\ & & \ddots & \\ & & & S \end{bmatrix} \tag{2.15b}$$

We can further represent Y_k as the sum of the forced and natural responses:

$$Y_k = \Gamma U_k + \Phi_x \hat{x}_{k|k} + \Phi_w \hat{w}_{k|k} \tag{2.16}$$

where Γ, Φ_x and Φ_w is given as:

$$\Gamma = \begin{bmatrix} H_1 & 0 & 0 & 0 & 0 \\ H_2 & H_1 & 0 & 0 & 0 \\ H_3 & H_2 & H_1 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ H_N & H_{N-1} & H_{N-2} & \cdots & H_1 \end{bmatrix}, \quad \Phi_x = \begin{bmatrix} \hat{C}\hat{A} \\ \hat{C}\hat{A}^2 \\ \hat{C}\hat{A}^3 \\ \vdots \\ \hat{C}\hat{A}^N \end{bmatrix} \tag{2.17a}$$

$$\Phi_w = \begin{bmatrix} \hat{C}\hat{G} \\ \hat{C}\hat{A}\hat{G} \\ \hat{C}\hat{A}^2\hat{G} \\ \vdots \\ \hat{C}\hat{A}^{N-1}\hat{G} \end{bmatrix}, \quad U_k = \begin{bmatrix} u_{k|k} \\ u_{k+1|k} \\ u_{k+2|k} \\ \vdots \\ u_{k+N-1|k} \end{bmatrix} \tag{2.17b}$$

and $H_i = \hat{C}\hat{A}^{i-1}\hat{B}, \ i = 1, 2, ..., N$

The regularization term ΔU_k can be expressed in terms of U_k by:

$$\Delta U_k = \Phi_u U_k - I_0 u_{k-1} \tag{2.18}$$

where Φ_u and I_0 is described as:

$$\Phi_u = \begin{bmatrix} I & 0 & 0 & 0 & 0 \\ -I & I & 0 & 0 & 0 \\ 0 & -I & I & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & -I & I \end{bmatrix}, \quad I_0 = \begin{bmatrix} I \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.19}$$

By substitution, the objective function can be expressed as:

$$\begin{aligned}\phi = &\ \frac{1}{2}\|Y_k - R_k\|_{\mathcal{Q}}^2 + \frac{1}{2}\|\Delta U_k\|_{\mathcal{S}}^2 \\ &= \frac{1}{2}\left\|\Gamma U_k + \Phi_x \hat{x}_{k|k} + \Phi_w \hat{w}_{k|k} - R_k\right\|_{\mathcal{Q}}^2 + \frac{1}{2}\|\Phi_u U_k - I_0 u_{k-1}\|_{\mathcal{S}}^2 \\ &= \frac{1}{2}(\Gamma U_k + \Phi_x \hat{x}_{k|k} + \Phi_w \hat{w}_{k|k} - R_k)^T \mathcal{Q}(\Gamma U_k + \Phi_x \hat{x}_{k|k} + \Phi_w \hat{w}_{k|k} - R_k) \\ &+ \frac{1}{2}(\Phi_u U_k - I_0 u_{k-1})^T \mathcal{S}(\Phi_u U_k - I_0 u_{k-1})\end{aligned} \tag{2.20}$$

From (2.20) we can express the optimization problem as an unconstrained Quadratic Program (QP):

$$\min_U \phi = \frac{1}{2}U_k^T H U_k + g_k^T U_k \tag{2.21}$$

with H and g_k given as:

$$H = \Gamma^T \mathcal{Q}\Gamma + \Phi_u^T \mathcal{S}\Phi_u \tag{2.22a}$$

$$g_k = -\Gamma^T \mathcal{Q}(R_k - \Phi_x \hat{x}_{k|k} + \Phi_w \hat{w}_{k|k}) + \Phi_u^T \mathcal{S} I_0 u_{k-1} \tag{2.22b}$$

Given that H is positive definite, the global minimum of (2.21) can be calculated as:

$$\begin{aligned}U_k &= -H^{-1} g_k \\ &= -H^{-1}(\Gamma^T \mathcal{Q} R_k - \Gamma^T \mathcal{Q}\Phi_x \hat{x}_{k|k} - \Gamma^T \mathcal{Q}\Phi_w \hat{w}_{k|k} + \Phi_u^T \mathcal{S} I_0 u_{k-1})\end{aligned} \tag{2.23}$$

The solution is obtained from unconstrained optimization by taking the derivative of the objective function and setting this equal to zero. The requirement for H ensures convexity, and is guaranteed if both Q_z and S is positive definite matrices.

From (2.23) we can see that the solution of U_k involves the terms: R_k, $\hat{x}_{k|k}$, $\hat{w}_{k|k}$ and u_{k-1}. The notation can be interpreted to an expression of the form:

$$U_k = \bar{L}_x \hat{x}_{k|k} + \bar{L}_w \hat{w}_{k|k} + \bar{L}_u u_{k-1} + \bar{L}_R R_k \tag{2.24}$$

If we assume that R_k is constant in the prediction horizon, the following notation can be used:

$$R_k = \begin{bmatrix} r_k & r_k & r_k & \cdots & r_k \end{bmatrix}^T \tag{2.25}$$

The assumption of a constant reference allows us to express u_k as:

$$u_k = L_x \hat{x}_{k|k} + L_w \hat{w}_{k|k} + L_u u_{k-1} + L_r r_k \tag{2.26}$$

with the gains:

$$L_x = I_0^T \bar{L}_x = -I_0^T H^{-1} \Gamma^T \mathcal{Q} \Phi_x \tag{2.27a}$$
$$L_w = I_0^T \bar{L}_w = -I_0^T H^{-1} \Gamma^T \mathcal{Q} \Phi_w \tag{2.27b}$$
$$L_r = I_0^T \bar{L}_R I_r = I_0^T H^{-1} \Gamma^T \mathcal{Q} I_r \tag{2.27c}$$
$$L_u = I_0^T \bar{L}_u = I_0^T H^{-1} \Phi_u^T \mathcal{S} I_0 \tag{2.27d}$$

where $I_r = \begin{bmatrix} I & I & I & \cdots & I \end{bmatrix}^T$.

The definition of I_r is required, since L_R is a matrix. It is required that the entire reference vector is supplied to calculate the control signal for $k = 0$. The assumption that the reference vector has repeated elements allows us to express R_k as $I_r r_k$.

It should be noticed, that the assumptions made to derive (2.26), have some restraining effects on the controller. The assumption for the reference (2.25) removes the possibility to announcing set-point changes in advance. MPC excels in this feature with the ability to make smooth transitions, because the controller can act on the system before the change is planned.

An important concept is that (2.26) is not intended as a proposal for implementing MPC controllers. The reason for the derived expressions has an analytical purpose, and can be used together with the Kalman filter to express an unconstrained MPC controller as a state space model.

2.4 Controller State Space Model

From the Kalman recursions (2.6), it is possible to express (2.26) in terms of $\hat{x}_{k|k-1}$ and y_k instead of $\hat{x}_{k|k}$ and $\hat{w}_{k|k}$:

$$u_k = (L_x - L_x K_{fx}\hat{C} - L_w K_{fw}\hat{C})\hat{x}_{k|k-1} + (L_x K_{fx} + L_w K_{fw})y_k + L_u u_{k-1} + L_r r_k \tag{2.28}$$

The state update in (2.6) can further be stated as a single recursion by substitution of (2.6b) and (2.6c) into (2.6d):

$$\hat{x}_{k+1|k} = (\hat{A} - \hat{A}K_{fx}\hat{C} - \hat{G}K_{fw}\hat{C})\hat{x}_{k|k-1} + \hat{B}u_k + (\hat{A}K_{fx} + \hat{G}K_{fw})y_k \tag{2.29}$$

From (2.29) and (2.28) we can derive a state space representation of the controller:

$$\begin{bmatrix} \hat{x}_{k+1|k} \\ u_k \end{bmatrix} = \begin{bmatrix} \hat{A} + \hat{B}L_x - \Theta\hat{C} & \hat{B}L_u \\ L_x - \theta\hat{C} & L_u \end{bmatrix} \begin{bmatrix} \hat{x}_{k|k-1} \\ u_{k-1} \end{bmatrix} + \begin{bmatrix} \Theta & \hat{B}L_r \\ \theta & L_r \end{bmatrix} \begin{bmatrix} y_k \\ r_k \end{bmatrix} \tag{2.30}$$

Θ and θ is used to denote common factors and are defined as:

$$\theta = L_x K_{fx} + L_w K_{fw} \tag{2.31a}$$

$$\Theta = \hat{A}K_{fx} + \hat{G}K_{fw} + \hat{B}\theta \tag{2.31b}$$

We can represent (2.30) in a more convenient manner as:

$$x_{k+1}^c = A_c x_k^c + B_{cy} y_k + B_{cr} r_k \tag{2.32a}$$

$$u_k = C_c x_k^c + D_{cy} y_k + D_{cr} r_k \tag{2.32b}$$

where the state space matrices are defined as:

$$A_c = \begin{bmatrix} \hat{A} + \hat{B}L_x - \Theta\hat{C} & \hat{B}L_u \\ L_x - \theta\hat{C} & L_u \end{bmatrix} \tag{2.33a}$$

$$B_{cy} = \begin{bmatrix} \Theta \\ \theta \end{bmatrix}, \qquad B_{cr} = \begin{bmatrix} \hat{B}L_r \\ L_r \end{bmatrix} \tag{2.33b}$$

$$C_c = \begin{bmatrix} L_x - \theta\hat{C} & L_u \end{bmatrix} \tag{2.33c}$$

$$D_{cy} = \theta, \qquad D_{cr} = L_r \tag{2.33d}$$

It should be noticed, that the derivation of (2.32) requires that the reference is constant over the entire prediction horizon, and actuator and output constraints are neglected. (2.32) should <u>not</u> be considered as a proposition for implementation. The state space model is solely derived for analytical purposes, as it represents the dynamic behaviour of a MPC under the given assumptions.

If the desired MPC application has range constraints on actuators and outputs (constrained MPC), the calculation of the control signal becomes more complicated. It is required to use iterative algorithms such as Active Set or Interior Point solvers for this type of problem. A constrained MPC has a nonlinear behaviour if range constraints are violated. The control signal trajectory is however identical for the unconstrained and constrained controller, provided that the control signal for the unconstrained controller do not violate the range constraints. It can then be argued that the derived state space model also can be used to describe the behaviour of a constrained MPC under typical working conditions.

2.5 Summary

We have derived the control law for an unconstrained MPC. We have further shown how the control signal u_k can be expressed as a state space system, provided the reference is constant over the prediction horizon, and no range constraints exists for control signals and outputs.

Chapter 3

ARIMAX based MPC

In this chapter, we derive a Kalman filter ($\hat{A}$,$\hat{B}$,$\hat{C}$,$\hat{G}$) for an ARIMAX-based system model, which ensures off-set free tracking in the face of unmeasured constant disturbances. The attention is initially brought to SISO systems, but later we derive a model for the general MIMO case. Finally, a state space model is derived for an ARIMAX-based MPC.

3.1 SISO ARIMAX Model

The estimation model ($\hat{A}$,$\hat{B}$,$\hat{C}$,$\hat{G}$) from Section 2.3 does generally not provide offset free control of the plant in the face of unmeasured constant disturbances. This property requires that a disturbance model is integrated. We consider how an ARIMAX based model can be used for that purpose.

We assume that the control object can be described using an Auto Regressive model with eXogenuous input (ARX). The ARX model is often produced from system identification and has the following structure:

$$A(q^{-1})y_k = B(q^{-1})u_k + d_k + \varepsilon_k \tag{3.1}$$

where $\frac{B(q^{-1})}{A(q^{-1})} = G_{zu}(q^{-1})$, i.e. the transfer function of the process from input to controlled output.

d_k is assumed to be an unknown constant disturbance ($d_k = d$), and ε_k is assumed to be a white noise source.

Since d_k is a constant, it can be cancelled out by multiplying with $(1 - q^{-1})$ in both sides of the equation. This operation is equivalent to perform numerical differentation.

The described method is used together with a noise filter, and can then be expressed as an ARIMAX model. As stated previously, this approach is inspired from [JHR11].

$$A(q^{-1})y_k = B(q^{-1})u_k + \frac{1 - \alpha q^{-1}}{1 - q^{-1}} e_k \tag{3.2}$$

Rearranging the equation gives:

$$A(q^{-1})y_k(1 - q^{-1}) = B(q^{-1})u_k(1 - q^{-1}) + (1 - \alpha q^{-1})e_k \tag{3.3}$$

which is an expression on Auto Regressive Moving Average with eXogeneous input (ARMAX) form, where the polynomials can be identified to be:

$$\bar{A}(q^{-1}) = (1 - q^{-1})A(q^{-1}) \tag{3.4a}$$
$$\bar{B}(q^{-1}) = (1 - q^{-1})B(q^{-1}) \tag{3.4b}$$
$$\bar{C}(q^{-1}) = (1 - \alpha q^{-1}) \tag{3.4c}$$

The ARMAX polynomials has the structure:

$$\bar{A}(q^{-1}) = 1 + \bar{a}_1 q^{-1} + \bar{a}_2 q^{-2} + \ldots + \bar{a}_n q^{-n} \tag{3.5a}$$
$$\bar{B}(q^{-1}) = \bar{b}_1 q^{-1} + \bar{b}_2 q^{-2} + \ldots + \bar{b}_n q^{-n} \tag{3.5b}$$
$$\bar{C}(q^{-1}) = 1 + \bar{c}_1 q^{-1} + \bar{c}_2 q^{-2} + \ldots + \bar{c}_n q^{-n} \tag{3.5c}$$

Equation (3.3) can also be expressed in Δ variables:

$$A(q^{-1})\Delta y_k = B(q^{-1})\Delta u_k + (1 - \alpha q^{-1})e_k \tag{3.6}$$

where $\Delta y_k = y_k - y_{k-1}$ and $\Delta u_k = u_k - u_{k-1}$.

From (3.6) it is clear that a constant disturbance is suppressed, since $\Delta d_k = d_k - d_{k-1} = 0$.

The effect of α can be interpreted as a measure of how active the integrator in the system is. It should be noted, that in the case of $\alpha = 1$, the integrator is cancelled out and the ARIMAX model simplifies to an ARX description. In the other extreme where $\alpha = 0$ there is no filtering of noise (residuals) and pure integration.

In some sense α can be considered a comparable measure to the integration time constant (τ_i) in PI- and PID-controllers. The parameter similarly influences the rejection rate for disturbances.

3.1.1 State Space Conversion

The ARMAX representation should be used as the observer model ($\hat{A}$,$\hat{B}$,$\hat{C}$,$\hat{G}$) for the controller, and needs to be transformed to a state space model. The ARMAX model can be realized as a state space model in innovation form:

$$x_{k+1} = Ax_k + Bu_k + K\epsilon_k \tag{3.7}$$

$$y_k = Cx_k + \epsilon_k \tag{3.8}$$

with (A,B,C,K) realized in observer canonical form:

$$A = \begin{bmatrix} -\bar{a}_1 & 1 & 0 & 0 & 0 \\ -\bar{a}_2 & 0 & 1 & 0 & 0 \\ \vdots & \vdots & & \ddots & 0 \\ -\bar{a}_{n-1} & 0 & 0 & \cdots & 1 \\ -\bar{a}_n & 0 & 0 & \cdots & 0 \end{bmatrix}, \quad B = \begin{bmatrix} \bar{b}_1 \\ \bar{b}_2 \\ \vdots \\ \bar{b}_{n-1} \\ \bar{b}_n \end{bmatrix}, \quad K = \begin{bmatrix} \bar{c}_1 - \bar{a}_1 \\ \bar{c}_2 - \bar{a}_2 \\ \vdots \\ \bar{c}_{n-1} - \bar{a}_{n-1} \\ \bar{c}_n - \bar{a}_n \end{bmatrix} \tag{3.9a}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix} \tag{3.9b}$$

As discussed previously, a Kalman filter should be used to estimate the state values, since they can not be measured in a physical sense.

The recursions of the Kalman filter becomes particularly simple due to the correlation between measurement and process noise. Since we have perfect correlation of measurement noise and process noise, we can conclude that the estimation uncertainty must be $P = 0$.

This property simplifies the Kalman recursions to be:

$$e_k = y_k - \hat{C}\hat{x}_{k|k-1} \tag{3.10a}$$

$$\hat{x}_{k+1|k} = \hat{A}\hat{x}_{k|k-1} + \hat{B}u_k + \hat{G}e_k \tag{3.10b}$$

$$\hat{y}_{k+1|k} = \hat{C}\hat{x}_{k+1|k} \tag{3.10c}$$

In relation to the Kalman filter in Section 2.2, $K_{fx} = 0$ since $P = 0$ and $K_{fw} = R_{wv}R_{vv}^{-1} = 1$.

$\hat{A}$, $\hat{B}$, $\hat{C}$ and $\hat{G}$ is given as A, B, C and K, respectively.

3.2 MIMO ARIMAX Model

We aim to derive an ARIMAX based model for MIMO systems. The first part of the derivation is to extend the ARIMAX based SISO model into a description, that allows multiple input and outputs. We make the transition gradually by examining Multiple Inputs Single Output (MISO) systems before turning the attention to MIMO systems.

3.2.1 MISO ARIMAX Model

A MISO system with 2 inputs is considered, where the system can be described as:

$$Y(z) = G_{zu_1}(z)U_1(z) + G_{zu_2}(z)U_2(z) + G_{zw}W(z) + G_{zd}D(z) + V(z) \tag{3.11}$$

The numerator and denominator polynomials for the transfer functions are determined by:

$$G_{zu_i}(z) = \frac{B_i(z)}{A_i(z)}, \; G_{zd}(z) = \frac{B_d(z)}{A_d(z)}, \; G_{zw} = \frac{B_w(z)}{A_w(z)}$$

Initially, we only consider the deterministic properties of the system. The disturbance is unmeasured and not considered at first:

$$Y(z) = \frac{B_1(z)}{A_1(z)}U_1(z) + \frac{B_2(z)}{A_2(z)}U_2(z) \tag{3.12}$$

Both sides of the equation are multiplied with $A_1(z)A_2(z)$:

$$Y(z)A_1(z)A_2(z) = B_1(z)A_2(z)U_1(z) + B_2(z)A_1(z)U_2(z) \tag{3.13}$$

From this, we can derive an ARX expression by using the lag operator q^{-1} instead of z:

$$\bar{A}(q^{-1})y_k = \bar{B}_1(q^{-1})u_{1,k} + \bar{B}_2(q^{-1})u_{2,k} + d_k + \varepsilon_k \tag{3.14}$$

where $\bar{A}(q^{-1}) = A_1(q^{-1})A_2(q^{-1})$, $\bar{B}_1(q^{-1}) = B_1(q^{-1})A_2(q^{-1})$ and $\bar{B}_2(q^{-1}) = B_2(q^{-1})A_1(q^{-1})$.

The method can be generalized to a MISO control object with p inputs with the output description:

$$\begin{aligned} Y(z) = G_{zu_1}(z)U_1(z) + G_{zu_2}(z)U_2(z) + \ldots + G_{zu_p}U_p(z) \\ + G_{zw}W(z) + G_{zd}D(z) + V(z) \end{aligned} \tag{3.15}$$

The system can be described as the ARX model:

$$\bar{A}(q^{-1})y_k = \sum_{i=1}^{p} \bar{B}_i(q^{-1})u_{i,k} + d_k + \varepsilon_k \tag{3.16}$$

where the polynomials are calculated as:

$$\bar{A}(q^{-1}) = \prod_{i\in\{1,2,\ldots,p\}} A_i(q^{-1}) \tag{3.17a}$$

$$\bar{B}_i(q^{-1}) = B_i(q^{-1}) \prod_{j\in\{1,2,\ldots,p\}\setminus i} A_j(q^{-1}) \tag{3.17b}$$

In the same manner, as for the SISO case, an integrator is added together with a noise filter, and an ARIMAX formulation can be adopted:

$$\bar{A}(q^{-1})y_k = \sum_{i=1}^{p} \bar{B}_i(q^{-1})u_{i,k} + \frac{1-\alpha q^{-1}}{1-q^{-1}}\epsilon_k \tag{3.18}$$

The notation can be changed into an ARMAX description by multiplying both sides with $(1 - q^{-1})$:

$$\bar{A}(q^{-1})(1-q^{-1})y_k = \sum_{i=1}^{p} \bar{B}_i(q^{-1})(1-q^{-1})u_{i,k} + (1-\alpha q^{-1})\epsilon_k \tag{3.19}$$

The ARMAX polynomials are identified to be:

$$\begin{aligned} &\hat{A}(q^{-1}) = \bar{A}(q^{-1})(1-q^{-1}), \qquad \hat{B}_i(q^{-1}) = \bar{B}_i(q^{-1})(1-q^{-1}), \\ &\hat{C}(q^{-1}) = (1-\alpha q^{-1}) \end{aligned} \tag{3.20}$$

The polynomials has the following form:

$$\hat{A}(q^{-1}) = 1 + \hat{a}_1 q^{-1} + \hat{a}_2 q^{-2} + \ldots + \hat{a}_n q^{-n}, \tag{3.21a}$$

$$\hat{B}_i(q^{-1}) = \hat{b}_{i,1} q^{-1} + \hat{b}_{i,2} q^{-2} + \ldots + \hat{b}_{i,n} q^{-n}, \tag{3.21b}$$

$$\hat{C}(q^{-1}) = 1 + \hat{c}_1 q^{-1} + \hat{c}_2 q^{-2} + \ldots + \hat{c}_n q^{-n} \tag{3.21c}$$

3.2.1.1 State Space Conversion

Similar to the SISO case, (3.18) can be converted to a state space model in innovation form:

$$x_{k+1} = Ax_k + Bu_k + K\epsilon_k \tag{3.22}$$

$$y_k = Cx_k + \epsilon_k \tag{3.23}$$

where (A,B,K,C) can be realized in observer canonical form:

$$A = \begin{bmatrix} -\hat{a}_1 & 1 & 0 & 0 & 0 \\ -\hat{a}_2 & 0 & 1 & 0 & 0 \\ \vdots & \vdots & & \ddots & 0 \\ -\hat{a}_{n-1} & 0 & 0 & \cdots & 1 \\ -\hat{a}_n & 0 & 0 & \cdots & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} \hat{b}_{1,1} & \hat{b}_{2,1} & & \hat{b}_{p,1} \\ \hat{b}_{1,2} & \hat{b}_{2,2} & \cdots & \hat{b}_{p,2} \\ \vdots & \vdots & & \vdots \\ \hat{b}_{1,n-1} & \hat{b}_{2,n-1} & \cdots & \hat{b}_{p,n-1} \\ \hat{b}_{1,n} & \hat{b}_{2,n} & & \hat{b}_{p,n} \end{bmatrix},$$

$$K = \begin{bmatrix} \hat{c}_1 - \hat{a}_1 \\ \hat{c}_2 - \hat{a}_2 \\ \vdots \\ \hat{c}_{n-1} - \hat{a}_{n-1} \\ \hat{c}_n - \hat{a}_n \end{bmatrix}, \qquad C = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix} \tag{3.24}$$

3.2.2 MIMO systems

For a MIMO system with p inputs and m outputs, we assume that the following transfer function model describes the system:

$$\begin{bmatrix} Y_1(z) \\ Y_2(z) \\ \vdots \\ Y_m(z) \end{bmatrix} = \begin{bmatrix} G_{z_1u_1} & G_{z_1u_2} & \cdots & G_{z_1u_p} \\ G_{z_2u_1} & G_{z_2u_2} & \cdots & G_{z_2u_p} \\ \vdots & \vdots & & \vdots \\ G_{z_mu_1} & G_{z_mu_2} & \cdots & G_{z_mu_p} \end{bmatrix} \begin{bmatrix} U_1(z) \\ U_2(z) \\ \vdots \\ U_p(z) \end{bmatrix} + \begin{bmatrix} G_{z_1w} \\ G_{z_2w} \\ \vdots \\ G_{z_mw} \end{bmatrix} W(z) + \begin{bmatrix} G_{z_1d} \\ G_{z_2d} \\ \vdots \\ G_{z_md} \end{bmatrix} D(z) + \begin{bmatrix} V_1(z) \\ V_2(z) \\ \vdots \\ V_m(z) \end{bmatrix} \tag{3.25}$$

It is assumed, that the system has one process noise input $W(z)$ and one disturbance input $D(z)$. It is further assumed that every output is affected by measurement noise.

We propose, that each output is treated as a MISO subsystem, and a corresponding state space model should be calculated. A complete state space description can then be obtained by augmenting the MISO subsystems in the following manner:

$$A = \begin{bmatrix} A_1 & & & \\ & A_2 & & \\ & & \ddots & \\ & & & A_m \end{bmatrix}, \qquad B = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_m \end{bmatrix},$$
$$K = \begin{bmatrix} K_1 & & & \\ & K_2 & & \\ & & \ddots & \\ & & & K_m \end{bmatrix}, \qquad C = \begin{bmatrix} C_1 & & & \\ & C_2 & & \\ & & \ddots & \\ & & & C_m \end{bmatrix} \tag{3.26}$$

It should be noted, that a value of α should be determined for each output.

Alternatively, we could use another method of augmentation, which is applied directly at the ARMAX polynomials.

MIMO ARMAX polynomials

$$\hat{\mathbf{A}}(q^{-1})y_k = \hat{\mathbf{B}}(q^{-1})u_k + \hat{\mathbf{C}}(q^{-1})e_k \tag{3.27}$$

where the polynomials is determined from:

$$\hat{\mathbf{A}}(q^{-1}) = I_{(m\times m)} + \hat{A}_1 q^{-1} + \hat{A}_2 q^{-2} + \ldots + \hat{A}_n q^{-n} \tag{3.28a}$$
$$\hat{\mathbf{B}}(q^{-1}) = \hat{B}_1 q^{-1} + \hat{B}_2 q^{-2} + \ldots + \hat{B}_n q^{-n} \tag{3.28b}$$
$$\hat{\mathbf{C}}(q^{-1}) = I_{(m\times m)} + \hat{C}_1 q^{-1} + \hat{C}_2 q^{-2} + \ldots + \hat{C}_n q^{-n} \tag{3.28c}$$

The polynomial coefficients associated with the i'th sample delay is defined as:

$$\hat{A}_i = \begin{bmatrix} \hat{a}_{1,i} & & \\ & \hat{a}_{2,i} & \\ & & \hat{a}_{m,i} \end{bmatrix}, \qquad \hat{B}_i = \begin{bmatrix} \hat{b}_{11,i} & \hat{b}_{12,i} & \cdots & \hat{b}_{1p,i} \\ \hat{b}_{21,i} & \hat{b}_{22,i} & & \hat{b}_{2p,i} \\ \vdots & & \ddots & \vdots \\ \hat{b}_{m1,i} & & & \hat{b}_{mp,i} \end{bmatrix},$$
$$\hat{C}_i = \begin{bmatrix} \hat{c}_{1,i} & & \\ & \hat{c}_{2,i} & \\ & & \hat{c}_{m,i} \end{bmatrix} \tag{3.29}$$

From the definitions above, a canonical state space realization can similar to the SISO case, be stated as:

$$A = \begin{bmatrix} -\hat{A}_1 & I & 0 & 0 & 0 \\ -\hat{A}_2 & 0 & I & 0 & 0 \\ \vdots & \vdots & & \ddots & 0 \\ -\hat{A}_{n-1} & 0 & 0 & \cdots & I \\ -\hat{A}_n & 0 & 0 & \cdots & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} \hat{B}_1 \\ \hat{B}_2 \\ \vdots \\ \hat{B}_{n-1} \\ \hat{B}_n \end{bmatrix},$$

$$K = \begin{bmatrix} \hat{C}_1 - \hat{A}_1 \\ \hat{C}_2 - \hat{A}_2 \\ \vdots \\ \hat{C}_{n-1} - \hat{A}_{n-1} \\ \hat{C}_n - \hat{A}_n \end{bmatrix}, \qquad C = \begin{bmatrix} I & 0 & 0 & \cdots & 0 \end{bmatrix} \tag{3.30}$$

It should be mentioned, that the two ways of augmenting the MISO systems is equivalent. The latter method has the advantage of sharing structural resemblance with the SISO realization.

The Kalman recursions for the MIMO model is stated as:

$$e_k = y_k - \hat{C}\hat{x}_{k|k-1} \tag{3.31a}$$

$$\hat{x}_{k+1|k} = \hat{A}\hat{x}_{k|k-1} + \hat{B}u_k + \hat{G}e_k \tag{3.31b}$$

$$\hat{y}_{k+1|k} = \hat{C}\hat{x}_{k+1|k} \tag{3.31c}$$

The recursion are identical to the SISO system (3.10), but e_k, y_k and u_k are vectors instead of scalars. The matrices are given as: $\hat{A} = A$, $\hat{B} = B$, $\hat{C} = C$ and $\hat{G} = K$. The Kalman gains are given as: $K_{fx} = 0$ and $K_{fw} = I$.

It should be noted that the derived ARIMAX based model is not limited to systems of the form (3.25). There could possibly have been more disturbance and process noise inputs defined for the system, but this would not change the structure of the ARIMAX model.

3.3 Controller State Space Model

The controller state space model for an ARIMAX-based MPC is equivalently to (2.30) described as:

$$x^c_{k+1} = A_c x^c_k + B_{cy} y_k + B_{cr} r_k \tag{3.32a}$$
$$u_k = C_c x^c_k + D_{cy} y_k + D_{cr} r_k \tag{3.32b}$$

The matrices of the state space model becomes simpler than the general model (2.30). This is an effect of $K_{fx} = 0$. The matrices for the ARIMAX-based MPC is expressed as:

$$A_c = \begin{bmatrix} (\hat{A} - \hat{G}\hat{C}) + \hat{B}(L_x - L_w\hat{C}) & \hat{B}L_u \\ L_x - L_w\hat{C} & L_u \end{bmatrix} \tag{3.33a}$$
$$B_{cy} = \begin{bmatrix} \hat{G} + \hat{B}L_w \\ L_w \end{bmatrix}, \qquad B_{cr} = \begin{bmatrix} \hat{B}L_r \\ L_r \end{bmatrix} \tag{3.33b}$$
$$C_c^T = \begin{bmatrix} L_x - L_w\hat{C} \\ L_u \end{bmatrix} \tag{3.33c}$$
$$D_{cy} = L_w, \qquad D_{cr} = L_r \tag{3.33d}$$

3.4 Summary

The ARIMAX based MPC introduced in this chapter is to be used for the rest of this study. We have demonstrated how a Kalman filter for the ARIMAX based system is based on a state space model in innovation form. A controller state space model has been derived for the ARIMAX based MPC.

Chapter 4

Closed Loop Analysis

In this chapter, we derive a closed loop state space description of the system, i.e. the process and the controller. It is demonstrated how the closed loop description can be used to calculate output and control signal covariance for the system. We further examine how sensitivity functions can be used to asses closed loop performance.

4.1 Closed Loop State Space Model

The process and controller state space models should be recalled to be:

Process:

$$x_{k+1} = Ax_k + Bu_k + Gw_k + Ed_k \tag{4.1a}$$
$$y_k = Cx_k + v_k \tag{4.1b}$$
$$z_k = Cx_k \tag{4.1c}$$

Controller:

$$x^c_{k+1} = A_c x^c_k + B_{cy} y_k + B_{cr} r_k \tag{4.2a}$$
$$u_k = C_c x^c_k + D_{cy} y_k + D_{cr} r_k \tag{4.2b}$$

As the first step in the derivation of a closed loop description, we aim to express the process input u_k by the controller states x_k^c. This can be obtained by substituting (4.2b) into (4.1a):

$$x_{k+1} = Ax_k + B(C_c x_k^c + D_{cy} y_k + D_{cr} r_k) + Gw_k + Ed_k \tag{4.3}$$

Then (4.1b) is substituted into equation (4.3) to eliminate y_k:

$$x_{k+1} = Ax_k + B(C_c x_k^c + D_{cy}(Cx_k + v_k) + D_{cr} r_k) + Gw_k + Ed_k \tag{4.4}$$

By seperating the terms we obtain:

$$x_{k+1} = (A + BD_{cy}C)x_k + BC_c x_k^c + BD_{cy} v_k + BD_{cr} r_k + Gw_k + Ed_k \tag{4.5}$$

We now have a description of the process states with the controller states as an input.

A similar expression for the controller states with the process states as input should be derived. (4.1b) is substituted into (4.2a):

$$x_{k+1}^c = A_c x_k^c + B_{cy} C_y x_k + B_{cy} v_k + B_{cr} r_k \tag{4.6}$$

where the controller states can be expressed using the process states as input.

From (4.5) and (4.6) a closed loop description can be formed. We augment the states of the process x_k and controller states x_k^c and define the closed loop states:

$$x_k^{cl} = \begin{bmatrix} x_k \\ x_k^c \end{bmatrix} \tag{4.7}$$

The inputs to the closed loop system can be identified from (4.5) and we can define a system in the form:

$$x_{k+1}^{cl} = A_{cl} x_k^{cl} + B_{wcl} w_k + B_{vcl} v_k + B_{rcl} r_k + B_{dcl} d_k \tag{4.8a}$$
$$z_k = C_{zcl} x_k^{cl} \tag{4.8b}$$
$$y_k = C_{ycl} x_k^{cl} + v_k \tag{4.8c}$$
$$u_k = C_{ucl} x_k^{cl} + D_{cy} v_k + D_{cr} r_k \tag{4.8d}$$

The system matrices are identified from (4.1), (4.2), (4.5) and (4.6) to be:

$$A_{cl} = \begin{bmatrix} A + BD_{cy}C & BC_c \\ B_{cy}C & A_c \end{bmatrix} \tag{4.9a}$$
$$B_{wcl} = \begin{bmatrix} G \\ 0 \end{bmatrix}, \; B_{vcl} = \begin{bmatrix} BD_{cy} \\ B_{cy} \end{bmatrix}, \; B_{rcl} = \begin{bmatrix} BD_{cr} \\ B_{cr} \end{bmatrix}, \; B_{dcl} = \begin{bmatrix} E \\ 0 \end{bmatrix} \tag{4.9b}$$
$$C_{zcl} = \begin{bmatrix} C & 0 \end{bmatrix}, \; C_{ycl} = \begin{bmatrix} C & 0 \end{bmatrix}, \; C_{ucl} = \begin{bmatrix} D_{cy}C & C_c \end{bmatrix} \tag{4.9c}$$

4.2 Stochastic Analysis

The closed loop state space model can be used to calculate steady state covariances on the system output and control signal using the discrete Lyaupunov equation. We will derive the equation by initially defining a function for the state covariance:

$$R_{xx,k} = E\{(x_k^{cl} - \hat{x}_k^{cl})(x_k^{cl} - \hat{x}_k^{cl})^T\} \tag{4.10}$$

where $\hat{x}_k^{cl}$ is the mean state values.

We can express $x_{k+1}^{cl} - \hat{x}_{k+1}^{cl}$ as:

$$x_{k+1}^{cl} - \hat{x}_{k+1}^{cl} = A_{cl}(x_k^{cl} - \hat{x}_k^{cl}) + B_{wcl} w_k + B_{vcl} v_k \tag{4.11}$$

From (4.10) and (4.11) we can define an expression for $R_{xx,k+1}$:

$$\begin{aligned}R_{xx,k+1} =&E\{(A(x_k^{cl}-\hat{x}_k^{cl})+B_{wcl}w_k+B_{vcl}v_k)(A(x_k^{cl}-\hat{x}_k^{cl})+B_{wcl}w_k+B_{vcl}v_k)^T\}\\ =&E\{A(x_k^{cl}-\hat{x}_k^{cl})(x_k^{cl}-\hat{x}_k^{cl})^TA^T+A(x_k^{cl}-\hat{x}_k^{cl})w_k^TB_{wcl}^T\\ &+A(x_k^{cl}-\hat{x}_k^{cl})v_k^TB_{vcl}^T+B_{wcl}w_k(x_k^{cl}-\hat{x}_k^{cl})^TA^T+B_{wcl}w_kw_k^TB_{wcl}^T\\ &+B_{wcl}w_kv_k^TB_{vcl}^T+B_{vcl}v_k(x_k^{cl}-\hat{x}_k^{cl})^TA^T+B_{vcl}v_kw_k^TB_{wcl}^T\\ &+B_{vcl}v_kv_k^TB_{vcl}^T\}\end{aligned} \tag{4.12}$$

using the linearity property of the expectation operator, we can express $R_{xx,k+1}$ as:

$$\begin{aligned}R_{xx,k+1} =&A_{cl}R_{xx,k}A_{cl}^T+A_{cl}R_{xw,k}B_{wcl}^T+A_{cl}R_{xv,k}B_{vcl}^T\\ &+B_{wcl}R_{wx,k}A_{cl}^T+B_{wcl}R_{ww,k}B_{wcl}^T+B_{wcl}R_{wv,k}B_{vcl}^T\\ &+B_{vcl}R_{vx,k}A_{cl}^T+B_{vcl}R_{vw,k}B_{wcl}^T+B_{vcl}R_{vv,k}B_{vcl}^T\end{aligned} \tag{4.13}$$

We assume that w_k, v_k and x_k are uncorrelated:

$$R_{xx,k+1}=A_{cl}R_{xx,k}A_{cl}^T+B_{wcl}R_{ww,k}B_{wcl}^T+B_{vcl}R_{vv,k}B_{vcl}^T \tag{4.14}$$

In steady state, i.e. $k\rightarrow\infty$, we have provided that A_{cl} is stable, and the covariances of w_k and v_k are constant:

$$R_{xx}=A_{cl}R_{xx}A_{cl}^T+B_{wcl}R_{ww}B_{wcl}^T+B_{vcl}R_{vv}B_{vcl}^T \tag{4.15}$$

which is a discrete Lyapunov equation.

We have chosen to distinguish state covariance originating from process noise R_{xx}^w and state covariance originating from measurement noise R_{xx}^v. The separation is done for analytical reasons, since we find it useful to see the contribution on outputs and control signals from each noise source.

Output and control signal covariance from process noise:

$$R^w_{xx} = A_{cl} R^w_{xx} A^T_{cl} + B_{wcl} R_{ww} B^T_{wcl} \tag{4.16a}$$
$$R^w_{zz} = R_{yy} = C_{zcl} R^w_{xx} C^T_{zcl} \tag{4.16b}$$
$$R^w_{uu} = C_{ucl} R^w_{xx} C^T_{ucl} \tag{4.16c}$$
$$\tag{4.16d}$$

Output and control signal covariance from measurement noise:

$$R^v_{xx} = A_{cl} R^v_{xx} A^T_{cl} + B_{vcl} R_{vv} B^T_{vcl} \tag{4.17a}$$
$$R^v_{zz} = C_{zcl} R^v_{xx} C^T_{zcl} \tag{4.17b}$$
$$R^v_{yy} = C_{ycl} R^v_{xx} C^T_{ycl} + R_{vv} \tag{4.17c}$$
$$R^v_{uu} = C_{ucl} R^v_{xx} C^T_{ucl} + D_{cy} R_{vv} D^T_{cy} \tag{4.17d}$$

The total covariance can be calculated as:

$$R_{zz} = R^v_{zz} + R^w_{zz} \tag{4.18}$$
$$R_{yy} = R^v_{yy} + R^w_{yy} \tag{4.19}$$
$$R_{uu} = R^v_{uu} + R^w_{uu} \tag{4.20}$$

Alternatively, we could also calculate total output and control signal covariance on basis of (4.15).

4.3 Sensitivity Analysis

The use of sensitivity analysis to asses closed loop performance has been widely used as an aid in classical control theory. In this section, we derive sensitivity functions valid for an unconstrained MPC. Initially, we start by considering a simple 1-Degree of Freedom (DoF) control system, shown in Figure (4.1).

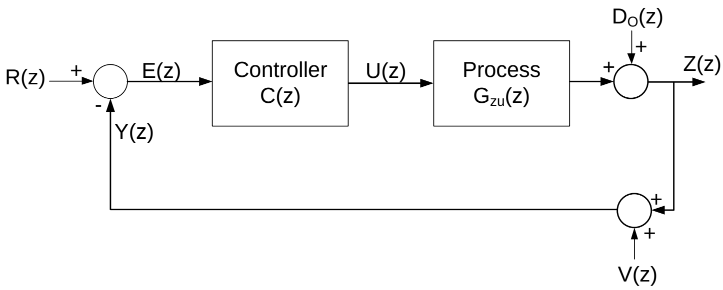


Figure 4.1: 1-DoF Feedback controller. The controller is given as $C(z)$ and the process is described as $G_{zu}(z)$. $D_O(z)$ is an output disturbance, $V(z)$ is measurement noise and $R(z)$ is the reference.

The loop gain for this system is defined as $L(z) = G(z)C(z)$, and the sensitivity and complementary sensitivity are defined as:

$$S(z) \triangleq \frac{E(z)}{R(z) - D_O(z)} = \frac{I}{I + L(z)} = \frac{Z(z)}{D_O(z)} \tag{4.21}$$

$$T(z) \triangleq I - S(z) = I - \frac{I}{I + L(z)} = \frac{L}{I + L(z)} = \frac{Z(z)}{R(z) - V(Z)} \tag{4.22}$$

From (4.21) and (4.22), the system output can be described in terms of $S(z)$ and $T(z)$:

$$Z(z) = T(z)R(z) + S(z)D_O(z) + T(z)V(z) \tag{4.23}$$

We can see from (4.23) that if $S(z)$ and $T(z)$ are known, the closed loop behaviour can be described from the sensitivity functions and the signals acting on the system. This property is useful in relation to tuning, since the perfomance tradeoffs immediately can be identified.

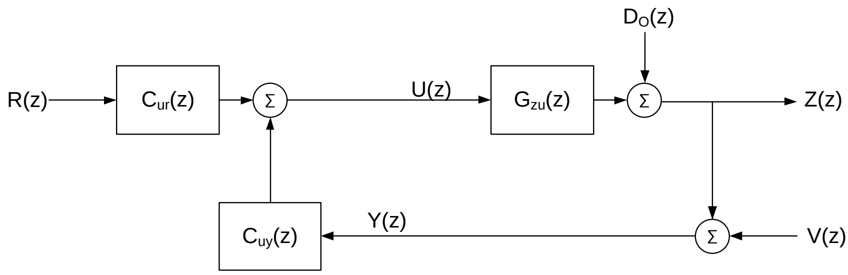


Figure 4.2: 2-DoF Feedback controller. The controller is represented by $C_{ur}(z)$ and $C_{uy}(z)$. $C_{ur}(z)$ is the transfer function from reference to control signal. $C_{uy}(z)$ is the transfer function from measured output to control signal.

4.3.1 Sensitivities for a MPC

The sensitivity functions presented in (4.21) and (4.22) are not valid for a MPC, since the control strategy has 2-DoF. A MPC has a unsymmetrical treatment of the reference signal $R(z)$ and the measured output $Y(z)$. A more general scheme, which can be used to represent an unconstrained MPC, is shown in Figure 4.2.

In the z-domain the MPC control law can be expressed as:

$$U(z) = C_{uy}(z)Y(z) + C_{ur}(z)R(z) \tag{4.24}$$

where $C_{uy}(z)$ and $C_{ur}(z)$ can be calculated from the controller state space model:

$$C_{uy}(z) = C_c(zI - A_c)^{-1}B_{cy} + D_{cy} \tag{4.25}$$

$$C_{ur}(z) = C_c(zI - A_c)^{-1}B_{cr} + D_{cr} \tag{4.26}$$

The sensitivities to the controlled output from reference, disturbance and measurement noise, can be derived from Figure 4.2:

$$\frac{Z(z)}{R(z)} = \frac{G_{zu}(z)C_{ur}(z)}{I - L(z)} \tag{4.27}$$

$$\frac{Z(z)}{V(z)} = \frac{L(z)}{I - L(z)} \tag{4.28}$$

$$\frac{Z(z)}{D_O(z)} = \frac{I}{I - L(z)} \tag{4.29}$$

where the loop gain of the system is defined to be: $L(z) = G_{zu}(z)C_{uy}(z)$.

In relation to the 1-DoF system, it is now required to calculate three sensitivity functions to describe the controlled outputs. If we instead consider the sensitivity to the measured outputs, we can derive functions similar to (4.21) and (4.22):

$$S(z) = \frac{Y(z)}{V(z)} = \frac{Y(z)}{D_O(z)} = \frac{Z(z)}{D_O(z)} = \frac{I}{I - G_{zu}(z)C_{uy}(z)} = \frac{I}{I - L(z)} \tag{4.30}$$

We define $T(z)$ as the sensitivity from reference to output:

$$T(z) = \frac{Y(z)}{R(z)} = \frac{Z(z)}{R(z)} = \frac{G_{zu}(z)C_{ur}(z)}{I - G_{zu}(z)C_{uy}(z)} = S(z)G_{zu}(z)C_{ur}(z) \tag{4.31}$$

At this point it should be noted, that the identity $S + T = I$ in general not can be expected to be valid for the sensitivity functions above. This can be demonstrated from:

$$S + T = \frac{I}{I - G_{zu}(z)C_{uy}(z)} + \frac{G_{zu}(z)C_{ur}(z)}{I - G_{zu}(z)C_{uy}(z)} = \frac{I + G_{zu}(z)C_{ur}(z)}{I - G_{zu}(z)C_{uy}(z)} \tag{4.32}$$

$S + T = I$ only holds true in the special case, where: $C_{ur}(z) = -C_{uy}(z)$.

From $S(z)$ and $T(z)$ the measured output can be expressed as:

$$Y(z) = T(z)R(z) + S(z)D_O(z) + S(z)V(z) \tag{4.33}$$

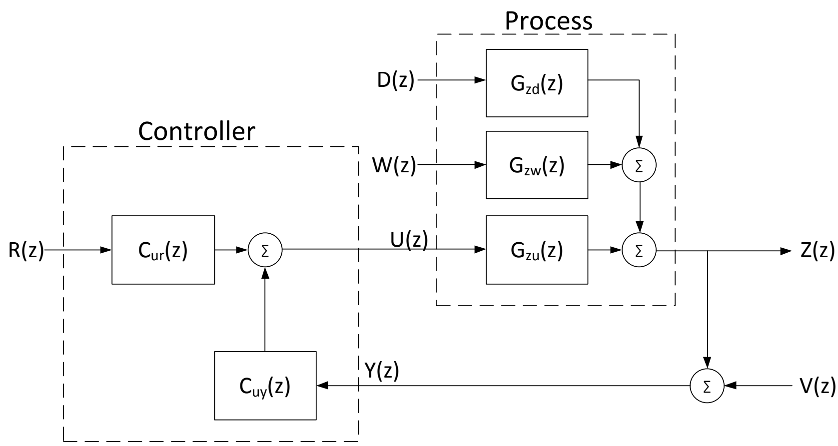


Figure 4.3: General 2-DoF control system. $W(z)$ is process noise and is propogated to the output from the transfer function $G_{zw}(z)$. $D(z)$ is a disturbance which enters the output through $G_{zd}(z)$.

In a similar fashion, the controlled output can be expressed as:

$$Z(z) = T(z)R(z) + S(z)D_O(z) + S(z)L(z)V(z) \tag{4.34}$$

Alternatively, to calculating $S(z)$ and $T(z)$ from (4.30) and (4.31), the sensitivities can be derived directly from the closed loop state space model:

$$S(z) = C_{ycl}(zI - A_{cl})^{-1}B_{vcl} + I \tag{4.35}$$

$$T(z) = C_{ycl}(zI - A_{cl})^{-1}B_{rcl} \tag{4.36}$$

It has so far only been considered how output disturbances and measurement noise affects the closed loop system. A more general system is shown in Figure 4.3.

The measured outputs for the system in Figure 4.3 can be described using $S(z)$ and $T(z)$ as:

$$Y(z) = S(z)G_{zw}(z)W(z) + S(z)V(z) + T(z)R(z) + S(z)G_{zd}(z)D(z) \tag{4.37}$$

4.4 Summary

We have derived a closed loop description from an augmented model of the plant and controller. The closed loop description allows the use of discrete Lyapunov equations to asses the stochastic properties of the system. Furthermore, we have conducted a sensitivity analysis of the closed loop system.

Part II

Application to Tuning

Chapter 5

Tuning of SISO systems

In this chapter, we consider how to asses closed loop performance in relation to tuning of an ARIMAX-based MPC for SISO systems. We define performance measures in relation to the deterministic properties, stochastic properties and robustness for a control system. Finally, we develop a toolbox, which can be used to analyze the performance of the closed loop MPC system for a given tuning evaluation range.

5.1 SISO MPC

The MPC algorithm introduced in Chapter 2 should be recalled to have two weight matrices, Q_z and S, which penalizes tracking error and control signal movement, respectively. For a SISO system the quantities are both scalars.

The terms in the objective function can be weighted relative to each other by defining: $\lambda = \frac{S}{Q_z}$. We can then express the optimization problem as:

$$\min_{\{u_{k+j}\}_{j=0}^{N-1}} \phi = \frac{1}{2} \sum_{j=0}^{N-1} \left\| \hat{y}_{k+1+j|k} - r_{k+1+j|k} \right\|_2^2 + \lambda \left\| \Delta u_{k+j} \right\|_2^2 \tag{5.1}$$

We consider λ and the ARIMAX coefficient α to be the tuning parameters to be determined. It is assumed, that the prediction horizon is chosen sufficiently long, such that an infinite horizon controller is emulated and no disrepancies exist between open loop and closed loop profiles. We further assume that the systems to be controlled has one disturbance input and one process noise input.

5.2 Performance Measures

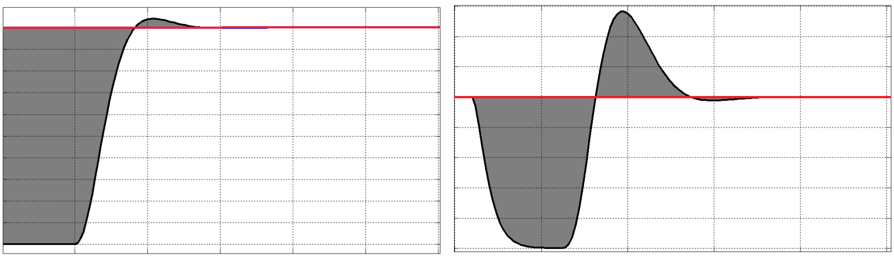
We categorize closed loop performance into three main categories: Deterministic measures, stochastic measures and robustness. The deterministic measures are associated with the tracking error in relation to changing set-point and disturbance rejection. Stochastic measures are related to the variance on the system output and control signal. Robustness is a qualitative indication of how the system reacts, when the model of the system is flawed or incomplete. Models are often obtained through system identification procedures rather than first principles. As a result the model estimate is often associated with a high degree of uncertainty, which must be accomodated by the controller. Robustness is further important, when actuators, due to wear and tear, degrades in performance.

5.2.1 Deterministic Performance Assesment

There exists several measures to determine transient behaviour for step responses. Classical measures are often divided into two main categories depending if the system is of 1. or 2. order (or higher). A 1. order response is categorized by the time constant, which is the value for which 63.2% of the steady state value is reached. A rule of thumb is that within 5 timeconstants 99.3% of the final value is reached.

If the system is of 2. or higher order, measures such as natural undamped frequency, damping factor, rise time, settling time, peak time etc., can be used to describe the response. For a specific system, the measures provide a high degree of information, but the interpretations can be complex.

We have chosen to use Integrated Absolute Error (IAE) to describe deterministic performance. The method is illustrated for a set-point change and rejection of a disturbance in Figure 5.1.



(a) IAE for reference change (b) IAE for disturbance rejection

Figure 5.1: Concept of IAE in relation to reference change and disturbance rejection. The red horizontal lines demonstrates the reference to the system, and the black curves shows the process output. The IAE values is calculated as the area of the grey markings.

The reference change and the disturbance occurs at $t = 0$. It should be noted, that the shown system is affected by dead time, which prevents the output to be influenced immediately from the changes.

Calculation of IAE for reference change and disturbance rejection can be done by step simulations of the closed loop state space system; $(A_{cl}, B_{rcl}, C_{zcl})$ and $(A_{cl}, B_{dcl}, C_{zcl})$. The IAE measure does not contain explicit information of the transient behaviour of the system. The accumulated absolute error of systems with different transient behaviours could be similar, i.e. control systems tuned to have either underdamped or overdamped characteristics could have identical IAE values. The IAE measure can however make for simple comparison of tunings, and is thereby a good relative performance indicator.

5.2.2 Stochastic Performance Assesment

The stochastic performance relates to the variance of the output and control signal. In Section 4.2 we demonstrated how to calculate steady state variances using the discrete Lyapunov equation and a closed loop state space model.

An alternative to evaluating the steady state variance is to examine the spectral densities of the output and control signal. The sensitivity function $S(z)$ can be used for evaluation of the spectral properties relating to measurement noise and process noise. In Appendix A we have shown how R_{yy}^{v} can be calculated from

$S(z)$.

We will only consider output and control signal variance for evaluation, since it translates to a simple scalar quantity. Variance can further be interpreted as the average noise power.

5.2.3 Robustness

In classical control design, robustness has been ensured by choosing sufficiently large gain and phase margins from open-loop Bode plots.

In modern control design robustness becomes more complex to ensure, since LQG/MPC designs approaches relies on the observer model. For aggressive tunings small deviations between the observer model and the process can cause instability or significantly degraded performance. It is possible to detune the system, such that deviations between process and model has more neglible effects. The price in doing so is degradation of performance, i.e. the system operates well beyond the theoretical limits.

Ensuring robustness has evolved to be an independent strategy, and dedicated robust controllers has been developed. The most popular is the range of H_∞ controllers, which is designed to optimize performance in face of process uncertainties.

We use Maximum sensitivity (M_S) to asses robustness for tuning configurations. The measure is further used as an optimization criterion in Robust Control Theory. From equation (4.30) it can be recalled that:

$$S(z) = \frac{Y(z)}{V(z)} = \frac{Y(z)}{D_O(z)} = \frac{Z(z)}{D_O(z)} \tag{5.2}$$

If process uncertainty is considered as an output disturbance to the nominal model, the sensitivity function can be used to determine the dynamic effect. M_S dictates the worst-case a model deviation causes at the output:

$$M_S = \underset{0 \leq \omega \leq \pi/T_s}{Max} |S(e^{j\omega T_s})| \tag{5.3}$$

In [SP05], the following relationsship between M_S and phase margin is given:

$$PM \geq 2\,\mathrm{arcsin}(\frac{1}{2M_s}) \geq \frac{1}{M_S} \tag{5.4}$$

The classical rule of thumb for ensuring $PM > 30°$ can be fullfilled by the requirement $M_S \leq 2$.

5.3 Toolbox

Based on the discussion about performance measures, a toolbox has been designed to evaluate the qualitative measures for any given SISO control object. The basic idea of the toolbox is to perform parameter sweeps for α and λ. The performance measures are then calculated for each (α,λ) combination. This strategy gives effectively an array of tuning combinations, where performance tradeoffs can be identified. The performance indicators for the toolbox are listed below.

Deterministic measures:

- IAE for reference step (J)
- IAE for disturbance rejection (J_d)

Stochastics measures:

- Variance on output
- Variance on control signal

Robust measures:

- Maximum sensitivity (M_S)

As previously discussed other measures could equally have been used for assesment of control performance, but the listed are chosen because of simplicity, quality and allround indications of the control performance.

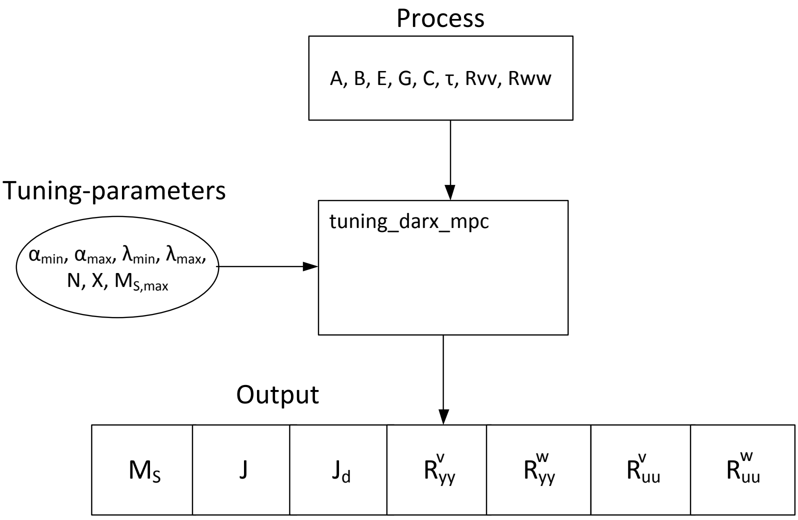


Figure 5.2: Data Flow diagram of SISO-Toolbox. A process model is supplied to the toolbox (tuning_darx_mpc). The tuning parameters specifies the evaluation range in which the performance measures are calculated.

The importance of each measure are often determined from the characteristics of the system to be controlled and the desired performance. M_S is however highlighted in this context, since the arguably most important property of a control system is stability and reliability. In the developed toolbox $M_{S,max}$ can be specified, such that tuning evaluations are only considered usable if $M_S < M_{S,max}$.

The toolbox requires a discrete state space model (A,B,E,G,C) of the control object. Dead times should not be represented in the state space model. Instead a dead time vector is specified:

$$\tau = [\ \tau_u \quad \tau_d \quad \tau_w\ ]$$

where τ_u is dead time from input to output, τ_d is dead time from disturbance to output and τ_w is deadtime from process noise to output.

The seperation is a numerical aid for the synthesis of the ARIMAX based controller model. R_{vv} and R_{ww} can be specified optionally, otherwise normalized values of 1 is used for evaluation.

In Figure 5.2 the parameter inputs and returned outputs are illustrated for the developed toolbox.

It is required, that the prediction horizon N of the controller is specified together with the evaluation ranges for α and λ. The toolbox is designed to evaluate X logarithmic spaced points in the range from λ_{min} and λ_{max} and X linearly spaced points from α_{min} and α_{max}.

Each output measure are presented in a matrix with the dimension $X \times X$, corresponding to an entry for all the $(\lambda,\ \alpha)$ combinations.

5.3.1 Toolbox Mechanisms

For each value of α the toolbox calculates an ARIMAX observer state space model. The input to the toolbox is a state space model and delays, so internally a transfer function is calculated as the basis for the ARIMAX model.

For each $(\alpha,\ \lambda)$ pair the algorithm synthesizes a closed loop state space description of the control system using the method described in section 3.3. Given the state space model, the following procedures are performed:

- Step response evaluation of reference tracking (IAE)
  - Simulate the closed loop state space model: A_{cl}, B_{rcl}, C_{ycl} with $r_t = 1$ for $t = 0 : 1 : N$
  - Calculate IAE of $r - y$
- Step response evaluation of disturbance rejection (IAE)
  - Simulate the closed loop state space model: A_{cl}, B_{dcl}, C_{ycl} with $d_t = 1$ and $r_t = 0$ for $t = 0 : 1 : N$
  - Calculate IAE of $r - y$
- Evaluation of M_S
  - Calculate sensitivity function: $S(z) = C_{ycl}(zI - A_{cl})^{-1}B_{vcl} + I$
  - Calculate $M_S = \underset{0\leq\omega\leq\pi}{Max}\ |S(e^{j\omega T_s})|$
- Calculate covariance on output and control signal from process noise
  - Calculate state covariance: $R^w_{xx} = A_{cl}R^w_{xx}A^T_{cl} + B_{wcl}R_{ww}B^T_{wcl}$
  - Calculate influence on output: $R^w_{yy} = C_{zcl}R^w_{xx}C^T_{zcl}$
  - Calculate influence on control signal: $R^w_{uu} = C_{ucl}R^w_{xx}C^T_{ucl}$

- Calculate covariance on output and control signal from measurement noise
  - Calculate state covariance: $R^v_{xx} = A_{cl}R^v_{xx}A^T_{cl} + B_{vcl}R_{vv}B^T_{vcl}$
  - Calculate influence on output: $R^v_{yy} = C_{ycl}R^v_{xx}C^T_{ycl} + R_{vv}$
  - Calculate influence on control signal: $R^v_{uu} = C_{ucl}R^v_{xx}C^T_{ucl} + D_{cy}R_{vv}D^T_{cy}$

It should be noted that unit steps are used for simulation of reference changes and disturbance rejection. This might not be realistical magnitudes for certain process applications. The toolbox is however designed to be used to asses the relative performance for different tuning combinations, in which case the signal scaling is not considered to be particular important.

5.3.2 MATLAB Script

This section is concerned with how the toolbox script is programmed. The toolbox consists of the two embedded loops shown in Algorithm 1. The primary loop sweeps through the values of α. For each α value the secondary loop sweeps the range of λ, and executes the computation algorithm.

Algorithm 1 tuning_darx_mpc

Require: $A, B, C, E, G, tau, T_s, \alpha_{min}, \alpha_{max}, \lambda_{min}, \lambda_{max}, X, N, M_{S,max}, R_{ww}, R_{vv}$

Create evaluation range for tuning parameters:

$\alpha_{range} = linspace(\alpha_{min}, \alpha_{max}, X)$

$\lambda_{range} = logspace(log_{10}(\lambda_{min}), log_{10}(\lambda_{max}), X)$

$i = 1, j = 1$

Sweep through every tuning-combinations in the evaluation range:

for $\alpha = \alpha_{range}$ **do**

for $\lambda = \lambda_{range}$ **do**

$$[M_S(i,j), J(i,j), J_d(i,j), R^v_{uu}(i,j), R^v_{yy}(i,j), R^w_{uu}(i,j), R^w_{yy}(i,j)] = tuning(A, B, C, E, G, T_s, \alpha, \lambda, N, R_{ww}, R_{vv})$$

$j = j + 1$

end for

$i = i + 1$

end for

In Algorithm 2 it is shown how the performance measures are computed. Initially, an ARIMAX based observer model is calculated from the approach in Chapter 3. From the ARIMAX model a state space model of the controller is calculated. The controller model is together with the process model then used to calculate a closed loop state space model.

The state space models are generated as minimum realizations. This is obtained from extracting the impulse response coefficients (Markov Parameters) from the system. The impulse response coefficients are organized in a Hankel matrix. A Singular Value Decomposition (SVD) is made to determine the necessary order to represent the system. From the decomposition, the state space matrices are then reconstructed in a minimal sense. The realization functions we have used in this project is taken from [Jø04]. The use of minimal realizations has particular importance if $\alpha = 1$ (pole-zero cancellation).

It should be noticed, that the controller gains L_x, L_w, L_r and L_u are not calculated directly by inversion of H as proposed in section 2.3. The reason is that H can be ill-conditioned [Mac02]. We use instead a Cholesky factorization of H and calculate the gains from the factorization.

Given the closed loop state space model, the performance measures can be calculated. In the evaluation of M_S a vector of frequencies is defined on a logarithmic scale in the range $10^{-4} \leq \omega \leq \pi/T_s$ with a 1000 points. The approach has proven robust in the cases considered in the project. It could be considered to construct a search algorithm, which can adapt the step size to find the maximum and be more computational effective.

Algorithm 2 tuning

Require: $A, B, C, E, G, tau, T_s, \alpha, \lambda, N, R_{ww}, R_{vv}$

Form observer model (ARIMAX):

$c = ceil(tau/T_S) - tau/T_s$

$G_{zu}(z) = \frac{B(z)}{A(z)} = ss2tf(A, B, C, 0) \cdot [\ c \quad (1-c) \cdot z^{-1}\ ] \cdot z^{-(tau/Ts)+1}$

$\bar{A} = (1 - q^{-1})A(q^{-1})$ // Calculate ARMAX polynomials

$\bar{B} = (1 - q^{-1})B(q^{-1})$

$\bar{C} = (1 - \alpha q^{-1})$

$[\hat{A}, \hat{B}, \hat{K}, \hat{C}] = armax2ss(\bar{A}, \bar{B}, \bar{C})$

Calculate closed loop state space model:

$[A_c, B_{cy}, B_{cr}, C_c, D_{cy}, D_{cr}] = ss\_MPC(\hat{A}, \hat{B}, \hat{C}, \hat{K}, K_{fx}, K_{fw}, 1, \lambda, N)$

$[A_{cl}, B_{wcl}, B_{vcl}, B_{rcl}, B_{dcl}, C_{zcl}, C_{ycl}, C_{ucl}] = ss\_closed\_loop$
$(A, B, C, C, E, G, A_c, B_{cy}, B_{cr}, C_c, D_{cy}, D_{cr})$

Maximum Sensitivity:

$[Sza, Szb] = ss2tf(A_{cl}, B_{vcl}, C_{ycl}, 1);\ Swz = frqrsp\_dtf(Sza, Szb, \omega, Ts)$

$M_S = max(abs(Swz))$

Evaluate step response in reference and disturbance input:

$T = 0 : 1 : N$ // the evaluation range is the prediction horizon

$[X1, X1u] = dstep\_rsp(A_{cl}, B_{rcl}, C_{zcl}, C_{ucl}, 0, D_{cr}, T)$

$[X2, X2u] = dstep\_rsp(A_{cl}, B_{dcl}, C_{zcl}, C_{ucl}, 0, 0, T)$

$J = sum(abs(ones(length(X1), 1) - X1));\ J_d = sum(abs(X2))$

Evaluate variance on output and control signal:

$R^w_{xx} = dlyap(A_{cl}, B_{wcl} R_{ww} B^T_{wcl});\ R^w_{yy} = C_{ycl} R^w_{xx} C^T_{ycl}$

$R^w_{uu} = C_{ucl} R^w_{xx} C^T_{ucl}$

$R^v_{xx} = dlyap(A_{cl}, B_{vcl} R_{vv} B^T_{vcl});\ R_{vy} = C_{ycl} R^v_{xx} C^T_{ycl} + R_{vv}$

$R^v_{uu} = C_{ucl} R^v_{xx} C^T_{ucl} + D_{cy} R_{vv} D^T_{cy}$

The actual programming differs from the pseudo code in a small but significant way. The loop evaluating different values of λ is embedded in Algorithm 2. This has been done since it otherwise would be required to recalculate the ARIMAX model for each value of λ.

The function $armax2ss$ is not a real function. This notation indicates the state space realization of the ARIMAX model. The realization is embedded in the algorithm as described in section 3.1.1.

The complete code of the toolbox can be found in Appendix E.

5.4 Summary

We have discussed different measures to asses closed loop performance of a SISO system. A number of key measures to asses the deterministic, stochastic and robust properties has been chosen. We have designed a toolbox based on the chosen measures, which can be used to assist the control engineer to tune the system for desired characteristics. The toolbox can exclude results, which do not satisfy requirements on robustness.

Chapter 6

Tuning of MIMO Systems

In this chapter, we consider performance assesment for MIMO systems. The measures from previous chapter are used, but will appear as either vectors or matrices. Different methods to represent matrix and vector information as scalar quantities are investigated. Finally, we propose a method to tune MIMO systems based on an optimization approach.

6.1 Performance Evaluation of MIMO Systems

The deterministic, stochastic and robust performance measures used for SISO systems is expanded to $(p \times m)$ MIMO systems. It is assumed, that the systems to be controlled only has one disturbance input. Initially, we consider how to express robustness for a MIMO system based on the sensitivity function $S(z)$.

6.1.1 Sensitivity

The sensitivity function for a $(p \times m)$ MIMO system is calculated as:

$$S(z) = C_{ycl}(zI - A_{cl})^{-1}B_{vcl} + I_{(m\times m)} \tag{6.1}$$

For a 2×2 system, $S(z)$ has the structure:

$$S(z) = \frac{Y(z)}{V(z)} = \begin{bmatrix} \frac{Y_1(z)}{V_1(z)} & \frac{Y_1(z)}{V_2(z)} \\ \frac{Y_2(z)}{V_1(z)} & \frac{Y_2(z)}{V_2(z)} \end{bmatrix} \tag{6.2}$$

Our primary interest in $S(z)$ is as a robustness parameter. For SISO systems, we used maximum sensitivity M_S to quantify robustness. This can be interpretated as the maximum magnitude for $S(z)$ in the frequency domain. It is proposed to use the maximum singular value of $S(z)$ as an indication for the worst case sensitivity of MIMO systems [DS81]. The proposed method reduces analytical complexity, as only one scalar quantity is considered. Mathematically this is denoted as the H_∞ norm of $S(z)$, and is defined as:

$$M_S = \|S(z)\|_\infty \triangleq \max_\omega \bar{\sigma}(S(e^{j\omega T_s})) \tag{6.3}$$

The singular values can be obtained by a SVD factorization. The factorization has proven to be a very useful tool in MIMO system analysis. The singular values can be thought of as the gains of the system, where the maximum gain, for any input direction, can be expressed as the maximum singular value $\bar{\sigma}$ [SP05]. The minimum gain can likewise be characterized from the minimum singular value $\underline{\sigma}$. It should be noticed that (6.3) is also valid for SISO systems and in that case equivalent to (5.3).

6.1.2 Covariance

For evaluation of the stochastic properties of a MIMO system, a covariance matrix with $p \times p$ elements should be analyzed for the control signal, and a matrix with $m \times m$ entries for the outputs. The covariance matrices can be calculated from the discrete Lyapunov equation as described in Section 4.2.

An output covariance matrix R_{yy} is considered for a system with m outputs:

$$R_{yy} = \begin{bmatrix} \sigma_{11}^2 & \sigma_{12}^2 & \cdots & \sigma_{1m}^2 \\ \sigma_{21}^2 & \sigma_{22}^2 & \cdots & \sigma_{2m}^2 \\ \vdots & \vdots & \ddots & \\ \sigma_{m1}^2 & \sigma_{2m}^2 & \cdots & \sigma_{mm}^2 \end{bmatrix} \tag{6.4}$$

We aim to express the information of the matrix as a scalar quantity for simple performance assesment.

From the study of optimal design of experiments, we consider three evaluation criterions:

- A-criterion (Average criterion)
- D-criterion (Determinant criterion)
- E-criterion (Eigenvalue criterion)

The A-criterion expresses the average variances for the outputs, and is for the $m \times m$ covariance matrix R_{yy} calculated as:

$$\Phi_A(R_{yy}) = \frac{1}{m} tr(R_{yy}) \tag{6.5}$$

where tr is the trace (sum of diagonals) in R_{yy}.

The D-criterion expresses the determinant of R_{yy}:

$$\Phi_D(R_{yy}) = Det(R_{yy}) \tag{6.6}$$

Finally, the E-criterion expresses the largest eigenvalue of the R_{yy}:

$$\Phi_E(R_{yy}) = \lambda_{max}(R_{yy})$$

The criterions can be interpreted geometrical as shown in Figure 6.1.

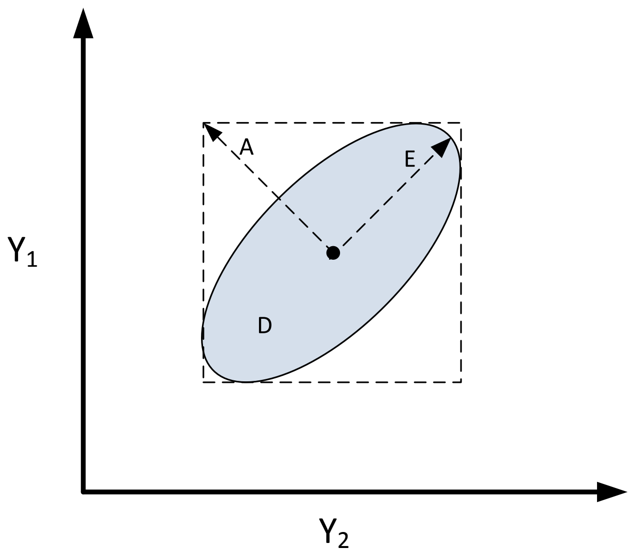
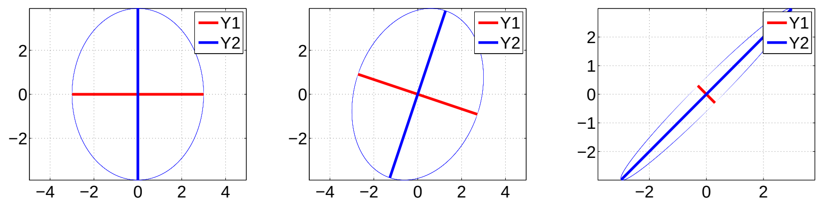


Figure 6.1: Geometrical Interpretation of A-, D- and E-criterions for a system with 2 outputs. The center point of the ellipsoid is the expected output values. The ellipsoid represent the joint 99% confidence interval.

Figure 6.1 illustrates a 2 output system in a steady state with the outputs Y_1 and Y_2. The ellipsoid represent the joint 99% confidence intervals of the outputs. The centerpoint of the ellipsoid represents the mean values of the outputs.

When the A-criterion is minimized, this corresponds to minimizing the enclosing box of the ellipsoid and minimizing the average variance for both outputs. Minimization of the E-criterion corresponds to shortening the major axis of the ellipsoid and reducing the highest variance of the system. The D-criterion can be interpreted as the area/volume of the confidence ellipsoid.

The direction of the ellipsoids major axis is dependent of the cross correlation of the outputs. If the outputs are entirely uncorrelated, the axis of the confidence ellipsoid is parallel to the cartesian axes. In Figure 6.2 confidence ellipsoids are drawed for three systems with different correlation properties.



(a) $\Phi_A = 1.34$, $\Phi_D = 1.69$, $\Phi_E = 1.69$ **(b)** $\Phi_A = 1.34$, $\Phi_D = 1.62$, $\Phi_E = 1.78$ **(c)** $\Phi_A = 1$, $\Phi_D = 0.04$, $\Phi_E = 1.98$

Figure 6.2: Confidence ellipsoids for three different systems.

The three cases in Figure 6.2 has the following covariances:

$$a) \quad \begin{bmatrix} 1 & 0 \\ 0 & 1.69 \end{bmatrix}, \; b) \quad \begin{bmatrix} 1 & 0.26 \\ 0.26 & 1.69 \end{bmatrix}, c) \quad \begin{bmatrix} 1 & 0.98 \\ 0.98 & 1 \end{bmatrix}$$

In Figure 6.2a the two outputs of the system are uncorrelated, and the major axis of the ellipsoid is parallel to the cartesian axes. In Figure 6.2b the variances of the outputs are the same as in Figure 6.2a but the outputs are correlated. It should be noted, that the smallest possible box encapsulating the two ellipsoids has identical dimensions (A-criterion). The length of the prinicipal axis and the areas of the ellipsoids are however different, giving different values for D- and E-criterions. In Figure 6.2c the outputs are nearly perfect correlated and the confidence ellipsoid has a narrow shape as a result. In the case of perfect correlation, the ellipsoid would converge to a line.

The confidence ellipsoids are calculated based on the χ^2 distribution and eigenvector analysis [Pou07].

It is not a trivial problem to choose the ideal criterion. A minimization of each of the criterions improves the overall stochastic properties and each criteria is concerned with minimization of the elliposoid in different senses.

6.1.3 Reference Tracking and Disturbance Rejection

In relation to evaluating IAE for reference tracking and disturbance rejection, it is required to evaluate a $m \times m$ matrix for reference tracking (J) and a vector with m elements for disturbance rejection (J_d).

For MIMO systems, a reference change on one output affects all the other outputs to some degree, depending on the controller and the structure of the system. We consider reference changes for each input seperately and calculates the effect in terms of IAE for each output. The IAE matrix is generated by simulating step responses on each reference input sequentially. Each row corresponds to a reference input and the columns represents the different outputs. The step responses are generated from the closed loop state space model: $(A_{cl}, B_{rcl}, C_{ycl})$.

The IAE vector for disturbance rejection is obtained from simulations of the closed loop state space model: $(A_{cl}, B_{dcl}, C_{ycl})$.

The disturbance rejection vector (J_d) and the reference response matrix (J) can be expressed as scalar quantities by using the euclidean or infinity norm. The latter can be interpretated as a worst case evaluation measure.

6.2 Tuning

In relation to tuning of a MPC for a $(p \times m)$ MIMO system, we should determine m diagonal entries in Q_z and p entries in S

$$\min_{\{u_{k+j}\}_{j=0}^{N-1}} \phi = \frac{1}{2} \sum_{j=0}^{N-1} \left\| \hat{y}_{k+1+j|k} - r_{k+1+j|k} \right\|_{Q_z}^2 + \left\| \Delta u_{k+j} \right\|_S^2 \tag{6.7}$$

For the ARIMAX observer model, we should furthermore determine m values of α.

Due to the number of inputs and outputs, it is not possible to visualize the tuning variables in the same manner as in the SISO case, unless the variables are bound together. In addition to the visualization problem, evaluating MIMO systems by making parameter sweeps of the tuning variables would be computationally far more expensive than for SISO systems.

We suggest a different strategy, by stating an optimization problem from a performance measure and define a bound for M_S to ensure robustness. The decision variables are the values of α and the diagonal entries in Q_z and S.

6.2.1 Tuning by Constrained Optimization

We propose that a controller is synthesized according to:

$$\min_{x} f(x) \tag{6.8a}$$

st.

$$M_S(x) - M_{S,max} \leq 0 \tag{6.8b}$$

where x is a vector of tuning parameters: $[\alpha_1, \alpha_2, ..., \alpha_m, q_1, q_2, ..., q_m, s_1, s_2, ..., s_p]$

$f(x)$ represent a performance measure, e.g. $\Phi_A(R_{yy})$ or $\|J\|_2 + \|J_d\|_2$.

The constrained optimization problem (6.8) is also known as a Non Linear Program (NLP). It should be recalled from section 2.3 and Appendix B, that the computation of the control signal for a MPC is similarly a constrained optimization problem. A MPC optimization problem is however a quadratic problem and is by proper selection of the weight matrices always a strictly-convex problem, i.e. only one global minimum exist. This feature is a great advantage for the algorithmic solvers as it guarantees convergence to the global minimum.

The optimization problem stated for tuning can not be expected to be convex. The proposed objectives relies on simulation results (J, J_d) and a solution of a discrete Lyapunov equation (R_{yy}) for which neither can be expected to be linear or quadratic.

6.3 Tuning Algorithm for MIMO Systems

For tuning of MIMO systems an algorithm has been developed based on the proposed optimization synthesis. Similar to the SISO Toolbox it is required a discrete state space model is supplied with externally defined dead times. Covariance of process and measurement noise should be supplied. Finally it is required that upper and lower boundaries is supplied for α, Q_z and S. The prediction horizon of the controller should ideally be specified such an infinite horizon controller is emulated for the entire evaluation range.

The embedded solvers in MATLAB has been used for solution of the optimzation problem. It is required to provide two functions to the solver; a function which returns the performance objective and a function which evaluate the constraints. It is futher required that a starting point is supplied (α_0, $Q_{z,0}$, S_0). The algorithm can use $\|J\|_2 + \|J_d\|_2$, $\|J_d\|_2$, $\|J_d\|_\infty$, $\Phi_E(R_{yy})$, $\Phi_A(R_{yy})$ and $\Phi_D(R_{yy})$ as optimization objectives.

In Algorithm 3 it is shown how fmincon is utilized to solve the optimization problem.

Algorithm 3 Tuning algorithm for MIMO systems

$sys = (A, B, C_y, C_z, E, G, tau, T_s, R_{ww}, R_{vv}, N)$ // System Properties

$x_0 = (\alpha_0, Q_{z,0}, S_0)$ // Starting point

$lb = (\alpha_{min}, Q_{z,min}, S_{min})$ // Bounds on tuning parameters
$ub = (\alpha_{max}, Q_{z,max}, S_{max})$

$x = fmincon(@(x)eval(sys, obj, x), x_0, lb, ub, @(x)eval\_MS(sys, x), options);$

From the options in Algorithm 3, the maximum number of allowed function evaluations can be defined. Furthermore, conditions for solver termination such as minimum step size for the decision variable (x) and function tolerance can be set. Finally, it can be specified if an active set, interior point or a S.Q.P. solver should be used.

The evaluation function for the optimization objective and for calulation of maximum sensitivity both require computation of the closed-loop state space model. The mechanisms to obtain the closed loop model are much the same as for the SISO toolbox, it should however be noted that the calculation of the ARIMAX observer model (Kalman filter) is more complex for MIMO systems. The ARIMAX model is calculated on basis of a state space model for each output as described in 3.2.1 and diagonally augmenting the models into a complete description. The augmented state space model is generated as described in Algorithm 4

Algorithm 4 MIMO ARIMAX based Controller Model

m=size(A,1), p=size(B,2)
for $m_x = 1 : m$ **do**
  for $p_x = 1 : p$ **do**
    $c = ceil(tau(m_x, p_x)/T_S) - (tau(mx, px)/T_S)$
    $cx = floor(tau(mx, px))$
    $G_{zu}(m_x, p_x) = ss2tf(A, B(:, p_x), C_y(m_x, :), 0) \cdot [\ c \quad (1-c) \cdot z^{-1}\ ]$
    $G_{zu}(m_x, p_x) = G_{zu}(m_x, p_x) \cdot z^{-cx+1}$
  end for
end for

$A\_arx = ones(m, 1)$
$B\_arx = ones(m, p)$

for $m_x = 1 : m$ **do**
  for $p_x = 1 : p$ **do**
    $A\_arx(m_x) = conv(A\_arx(m_x), G_{zu}(m_x, p_x).den)$
    for $p_{xx} = 1 : p$ **do**
      if $p_x = p_{xx}$ **then**
        $B\_arx(m_x, p_{xx}) = conv(B\_arx(m_x, p_{xx}), G_{zu}(m_x, p_{xx}).num)$
      else
        $B\_arx(m_x, p_{xx}) = conv(B\_arx(m_x, p_{xx}), G_{zu}(m_x, p_{xx}).den)$
      end if
    end for
    $B\_armax(m_x, p_x) = B\_arx(m_x, p_x)(1 - q^{-1})$
  end for
  $A\_armax(m_x) = A\_arx(m_x)(1 - q^{-1})$
  $C\_armax(m_x) = (1 - \alpha(m_x))$
  $[A(m_x), B(m_x), C(m_x), K(m_x)] =$
      $armax2ss(A\_armax(m_x), B\_armax(m_x, :), C\_armax(m_x))$
end for

$A_c = diag(A(:))$
$B_c = B(:)^T$
$K_c = diag(K(:))$
$C_c = diag(C(:))$

The transfer functions in Algorithm 4 are not directly derived from the state space model of the process using ss2tf(). In order to obtain a minimal realization, an impulse response is generated for each input-output pair. A minimal state space realization is then made for each SISO subsystem and from this converted to a transfer function description. In order to account for noninteger timedelays the algorithm has implemented a simple approximation which distributes the response between the surrounding sample instants. The other procedural steps for generation of the closed loop state space model is identical to the SISO case.

In terms of scaling, we use a magnitude of 1 for step response evaluation for each reference input and the disturbance input. This might not be realistical magnitudes, but it has the advantage that the optimization problem put equal emphasis on the tracking for all the outputs.

For a very brief insight to NLP solvers, an introduction to some key concepts is given in Appendix C.

6.4 Summary

We have discussed how scalar measures can be used to asses performance of a MIMO system in a similar fashion to SISO systems. Furthermore, we have proposed a tuning approach which is based on a constrained optimization problem with a bound on M_S to ensure robustness.

Part III

Case Studies

CHAPTER 7

Gas-Oil Furnace

In this chapter, we consider how the SISO toolbox can be used for tuning of a Gas-Oil furnace and examine performance limitations of the system. Furthermore, we apply the optimization tuning synthesis for the system with different objectives.

7.1 Process Description

We consider a Gas-Oil furnace, which is used to heat crude oil for refining. The process is described from the continuous transfer functions:

$$Y(s) = \frac{20}{(40s+1)(4s+1)}e^{-50s}U(s) - \frac{5}{(5s+1)^2}e^{-10s}(D(s)+W(s)) + V(s) \tag{7.1}$$

where $Y(s)$ is the temperature of the oil outlet $[^\circ C]$. $U(s)$ is the gas inflow to the heater $[l_s/min]$. $D(s)$ is the oil feed $[l/min]$. $W(s)$ is stochastic deviations in the oil feed and $V(s)$ is sensor noise. A process diagram of the gas-oil furnace is shown in Figure 7.1.

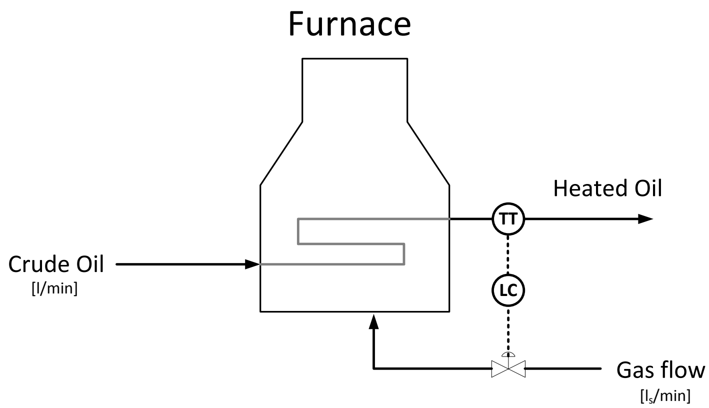


Figure 7.1: Gas-Oil Furnace. The crude oil is feeded into the furnace for heating. The Gas flow is used to control the temperature of the oil outflow.

The system has a dead time of 50 minutes from a change in the control input to the effect on the output. The disturbance and process noise input are modelled as a critically damped 2. order lowpass filter with a dead time of 10 minutes.

7.2 Application of SISO Toolbox

The continuos transfer function description is transformed into a discrete state space model in order to make use of the toolbox. We can identify the required dead time vector as:

$$\tau = [\ 50 \quad 10 \quad 10\ ] \tag{7.2}$$

We have selected the sampling period to be 2 minutes. The transfer function description (without dead times) are then transformed into a state space description of the form:

$$x_{k+1} = Ax_k + Bu_k + Ed_k + Gw_k$$

$$y_k = Cx_k + v_k, \qquad z_k = Cx_k$$

Initially, we apply the toolbox in the intervals $\alpha \in [0, 1]$ and $\lambda \in [10^2, 10^5]$. In both dimensions, 200 points are evaluated, which gives a total of 40000 evaluations. The prediction horizon of the controller is set to 150, which emulates an infinite horizon controller in the evaluation range. Normalized variances ($R_{vv} = 1$, $R_{ww} = 1$) has been used for the analysis. Initially, no bound is set on M_S. This allows us to investigate the characteristics of the system in the entire evaluation range. We present the performance measures generated from the toolbox as colour contour plots.

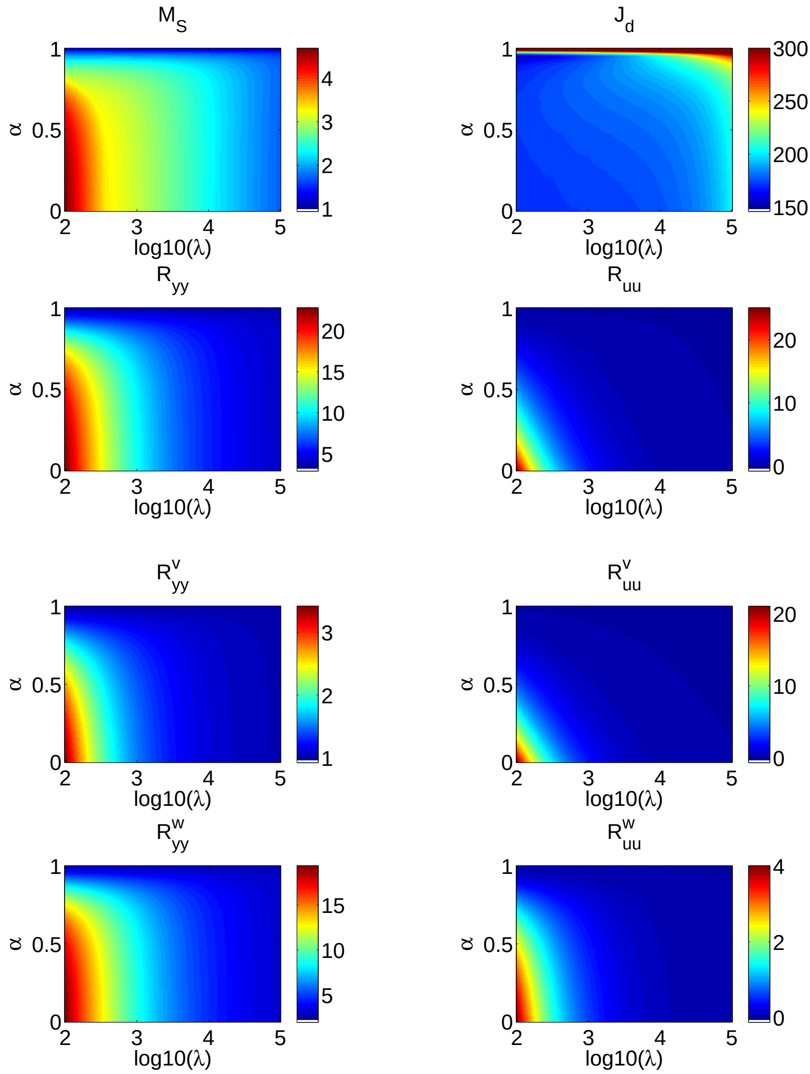


Figure 7.2: Colour contour plots of M_S, J_d, R_{yy}, R_{uu}, R^v_{yy}, R^v_{uu}, R^w_{yy} and R^w_{uu}.

From Figure 7.2, the stochastic minimum measures for the investigated interval can be identified to be: $R^w_{yy} = 2.472$, $R^v_{yy} = 1.00013$, $R^w_{uu} = 1.53 \cdot 10^{-5}$ and $R^v_{uu} = 1 \cdot 10^{-6}$. The minimum values are all located in the upper right corner, i.e. $\alpha = 1$ and $\lambda = 10^5$. The maximum values of the same parameters are located in the lower left corner, i.e. $\alpha = 0$ and $\lambda = 100$. The same tendancy applies for M_S.

J_d seem to be the most inconclusive measure as it has its minimum of 150.50 at $\alpha = 0.9497$ and $\lambda = 100$. The maximum 665.67 occurs at $\alpha = 1$ and $\lambda = 10^5$. In the contour plot the upper limit is set to 300 to provide a better dynamic visualization.

In Figure 7.3, J is presented seperately since the measure is solely dependent of λ. This is an effect caused by the 2 freedom degrees of the controller, since the transfer function from reference to control signal $C_{ur}(z)$ is decoupled from the feedback term $C_{uy}(z)$. The minimum value of J is 31.92 obtained at $\lambda = 100$, and the maximum value is 57.3 obtained at $\lambda = 10^5$.

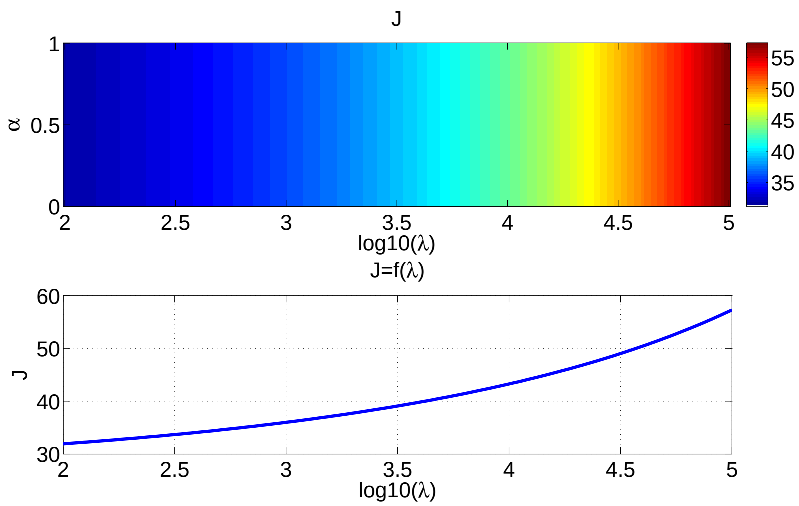


Figure 7.3: IAE for Reference Tracking (J).

The tuning problem is approached by requiring $M_S \leq 1.775$. In classical terms, this is equivalent to a phase margin of 33° approximately, which should provide sufficiently robustness for the system. We reevaluate the system using the Toolbox and interpretate the data using colour contour plots. It should be noted that the upper boundary of λ is changed to $1 \cdot 10^6$.

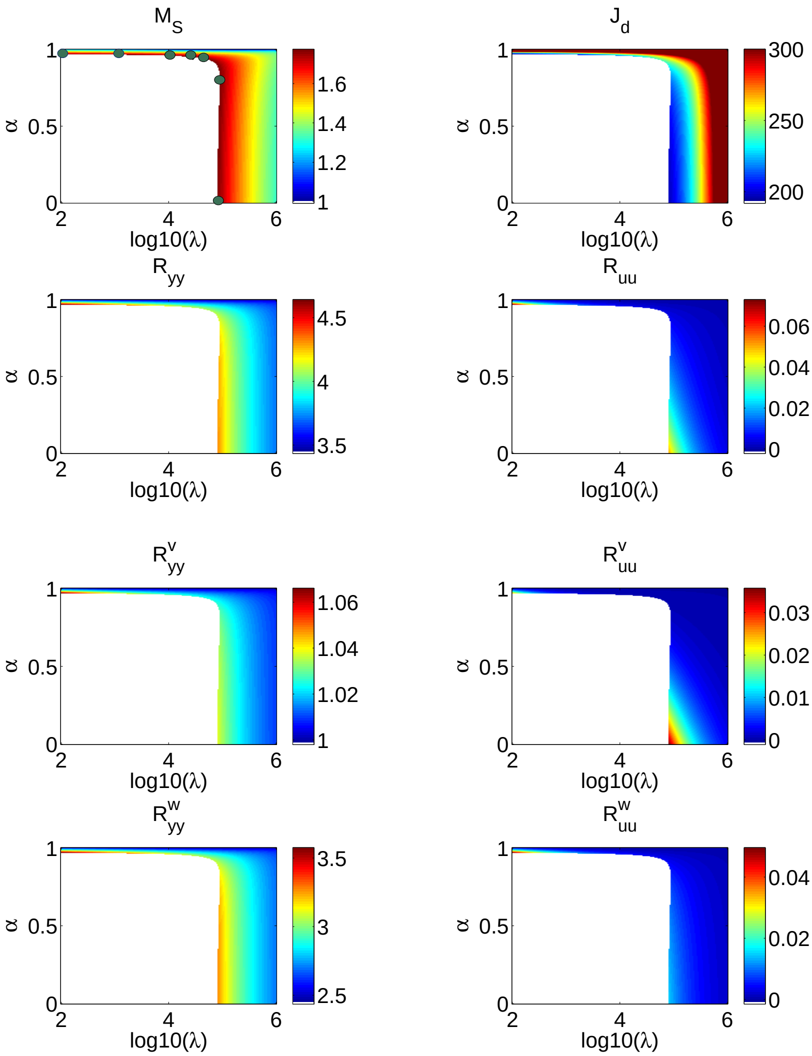


Figure 7.4: Colour contour plots of M_S, J_d, R_{yy}, R_{uu}, R^v_{yy}, R^v_{uu}, R^w_{yy} and R^w_{uu}. The tunings not satisfying $M_S \leq 1.775$ are shaded white. The grey circles in the plot for M_S shows evaluation points, satisfying $M_S = 1.775$.

In Figure 7.4 the performance measures are shown where tuning pairs not satisfying $M_S \leq 1.775$ are shaded white. This gives effectively an overview of which tunings that provides the required robustness. The trade offs between perfor-

α	λ	J	J_d	R_{yy}	R_{uu}	R_{yy}^w	R_{yy}^v	R_{uu}^w	R_{uu}^v
0.9698	100	**31.92**	**195.20**	4.64	0.073	3.577	1.066	0.050	0.023
0.9663	1000	36.00	213.47	4.39	0.020	3.344	1.042	0.0165	0.0033
0.9605	10000	43.26	232.26	4.14	0.006	3.114	1.028	0.0057	5.763e-4
0.9347	65173	53.84	245.93	**4.04**	**0.004**	**3.017**	**1.023**	**0.0035**	**2.735e-4**
0.9	89000	56.30	247.54	4.08	0.005	3.055	1.025	0.0041	3.876e-4
0.7	100000	57.38	222.86	4.19	0.010	3.163	1.031	0.0079	0.0021
0.0	79744	55.40	196.76	4.32	0.052	3.278	1.038	0.0152	0.0037

Table 7.1: Tuning pairs (α, λ) satisfying $M_S = 1.7750$

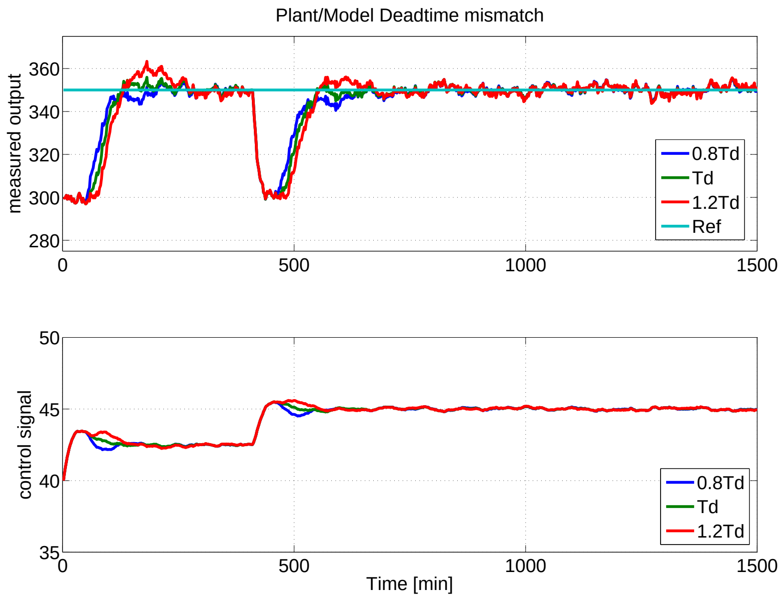
mance measures can be identified from the figure. We have evaluated tunings on the boundary of the shaded area $M_s = 1.775$. The tunings are summarized in Table 7.1.

Table 7.1 shows that the most desirable stochastic characteristics is obtained with $\alpha = 0.9347$ and $\lambda = 65173$. The best performance from a deterministic perspective is obtained with $\alpha = 0.9698$ and $\lambda = 100$. This tunings does however has the worst stochastic properties. Interestingly, the tuning with $\alpha = 0$ and $\lambda = 79744$ has a similar IAE value for disturbance rejection, but considerably better stochastic properties.

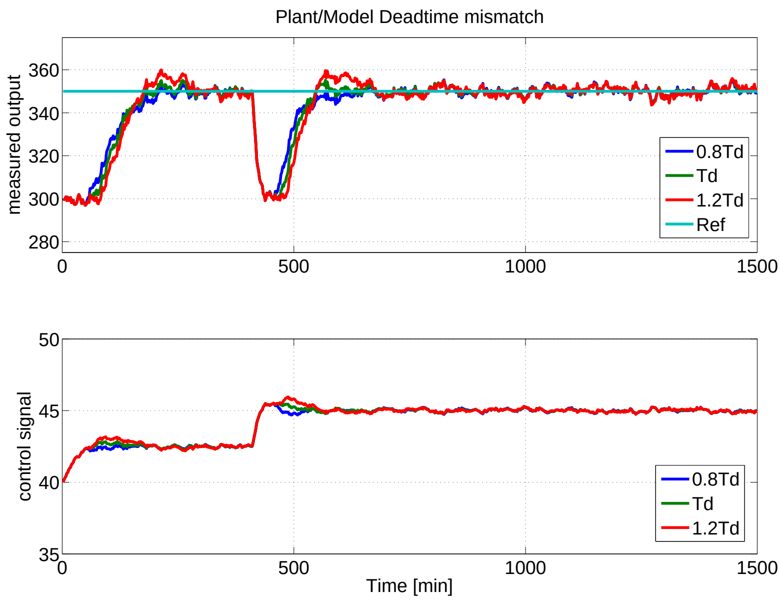
In [JHR11] it has been proposed to tune this system with $\lambda = 10^5$ and $\alpha = 0.7$, as a compromise between deterministic and stochastic properties. This choice also appears in Table 7.1. We would argue that a better trade-off between deterministic and stochastic perfomance is for $\alpha = 0.9605$ and $\lambda = 10^4$. This tuning has better stochastic properties and lower value of IAE for reference tracking than the proposed tuning from [JHR11]. The only parameter for which the tuning is inferior is from a higher value of J_d.

7.3 Simulations

A comparison of the tunings $\{\alpha = 0.9605,\ \lambda = 10^4\}$ and $\{\alpha = 0.7,\ \lambda = 10^5\}$ has been made in the face of process/model mismatch. We consider a situation, where the system has a oil feed of 100 l/min and the temperature of the outflow is 300°C. The reference is changed to 350°C at $t = 0$. A change in feed occurs at $t = 400$, where the oil inflow is changed to 110 l/min. The sensor noise variance and inflow variance is 1. Figure 7.5 and 7.6 shows simulations with deviations of ±20% in dead time and gain from the nominal system.

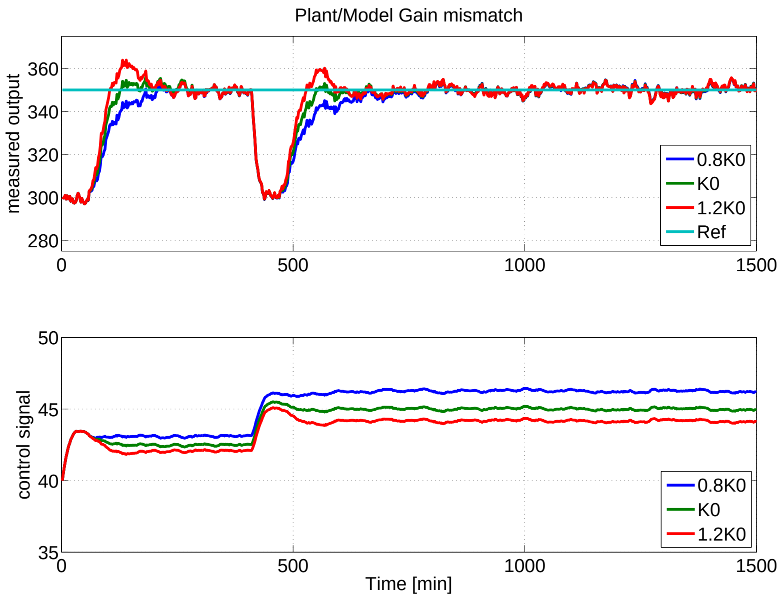


(a) $\lambda = 10000$, $\alpha = 0.9605$

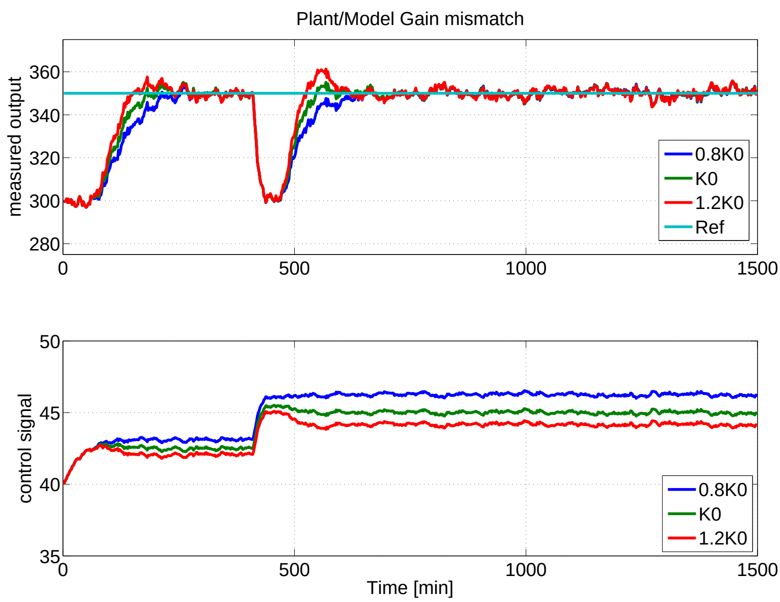


(b) $\lambda = 100000$, $\alpha = 0.7$

Figure 7.5: Comparison of tunings in relation to deadtime mismatch.



(a) $\lambda = 10000$, $\alpha = 0.9605$



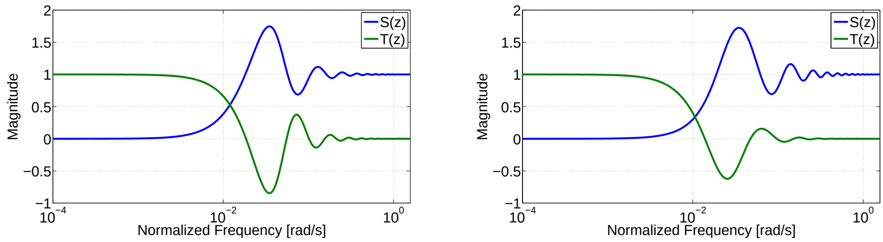
(b) $\lambda = 100000$, $\alpha = 0.7$

Figure 7.6: Comparison of tunings in relation to gain mismatch.

From Figure 7.5 and 7.6, it can be seen that both tunings offers similar robust performance in the face of mismatch. There are notable differences for setpoint tracking and especially control signal variance. The difference between the two tunings can be evaluated by observing the sensitivity and complementary sensitivity functions in Figure 7.7.

We can see that $S(z)$ is very similar for the two cases. This explains the similarity in the disturbance rejection response. For $T(z)$, we can see that the cut-off is at a higher frequency for the tuning configuration: $\alpha = 0.9605$ and $\lambda = 10^4$. The higher cut off explains the faster set-point tracking. It can also be observed that $T(z)$ for this tuning has a greater minimum peak, which similarly relates to a more agressive tuning.

The simulations so far, has concerned systems where $M_S = 1.775$. The Robustness properties of a system with $M_S = 2.5$ is demonstrated in Figure 7.8.



(a) $\lambda = 10000$, $\alpha = 0.9605$ **(b)** $\lambda = 100000$, $\alpha = 0.7$

Figure 7.7: Comparison of S(z) and T(z) for the considered tunings.

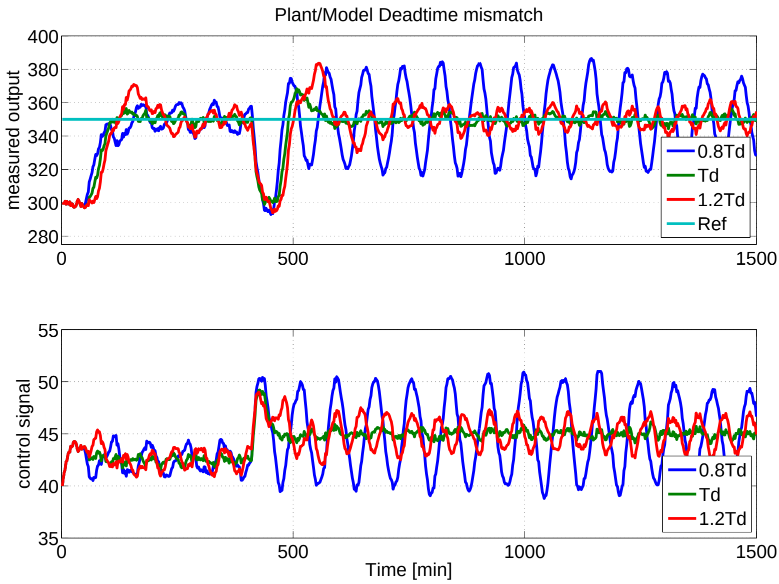


Figure 7.8: Deadtime mismatch with $M_S = 2.5$**.** $\{\lambda = 5 \cdot 10^3,\ \alpha = 0.7\}$

From Figure 7.8, we can see that an oscillatory response is obtained when there is dead time mismatch between the system and observer model. The phase margin of this tuning is approximately 12°. The small phase margin indicates that a small pertubation of the model can lead to instability.

7.4 Tuning by Optimization

We apply the optimization-based tuning procedure for the furnace. The procedure is intended for MIMO systems, but the SISO system allows us to compare the optimization result with the visualized performance parameters.

The optimization problem is stated as:

$$\min_{\lambda,\alpha} f(\lambda, \alpha) \tag{7.3a}$$

s.t.

$$M_S(\lambda, \alpha) \leq 1.775 \tag{7.3b}$$

We have used the boundaries $\alpha \in [0,1]$ and $\lambda \in [10^2, 10^6]$. 3 different starting points has been used for the algorithm:

$$\{\alpha_0, \lambda_0\} = \{0.5, 10^2\}, \{0.5, 10^4\}, \{0.5, 10^6\}$$

The furnace is a SISO system, which means that: $R_{yy} = \Phi_A(R_{yy}) = \Phi_D(R_{yy}) = \Phi_E(R_{yy})$. We have used $J_d(\alpha, \lambda)$ and $R_{yy}(\alpha, \lambda)$ as optimization objectives. The result of the optimization algorithm is shown in Table 7.2.

Objective	α	λ	$f(\alpha, \lambda)$
J_d	0.0000	79744	196.7630
R_{yy}	1.0000	1000000	3.4692

Table 7.2: Tuning by optimization

For optimization of J_d we can note, that the global minimum is not found. The global minimum was from Table 7.1, located at $\{\lambda = 100, \alpha = 0.9698\}$ and has the value $J_d = 195.20$. The difference between the global minimum and the result of the optimization is however marginal. The minimization of R_{yy} can be validated from Figure 7.4 to produce the global minimum within the evaluation boundaries. From the figure, we can also note that the global minimum of J_d is placed very isolated, which partly explains why the algorithm did not produce this solution.

The results in Table 7.2 has been produced by using the starting points on an active set, interior point and S.Q.P. solver. In all cases the solvers has converged to the same minima. In Appendix D the details of the optimization is stated.

Conclusively, it should be noted, that the optimization procedure primarily is intended for MIMO systems. For SISO systems, the SISO Toolbox provides a visualization of the performance measures, which makes it straightforward to choose the ideal compromise between the measures. This is not obtainable for MIMO systems.

7.5 Limitation of Performance

The industrial furnace has the peculiar relation, that the controller is unable to reduce output variance with increasing the effort on the control signal. This behaviour is originating from the dead time of the system. In this section, we intend to visualize how the dead time from input to output infuence the performance of the system. We analyze the case when no dead time is present from input to output.

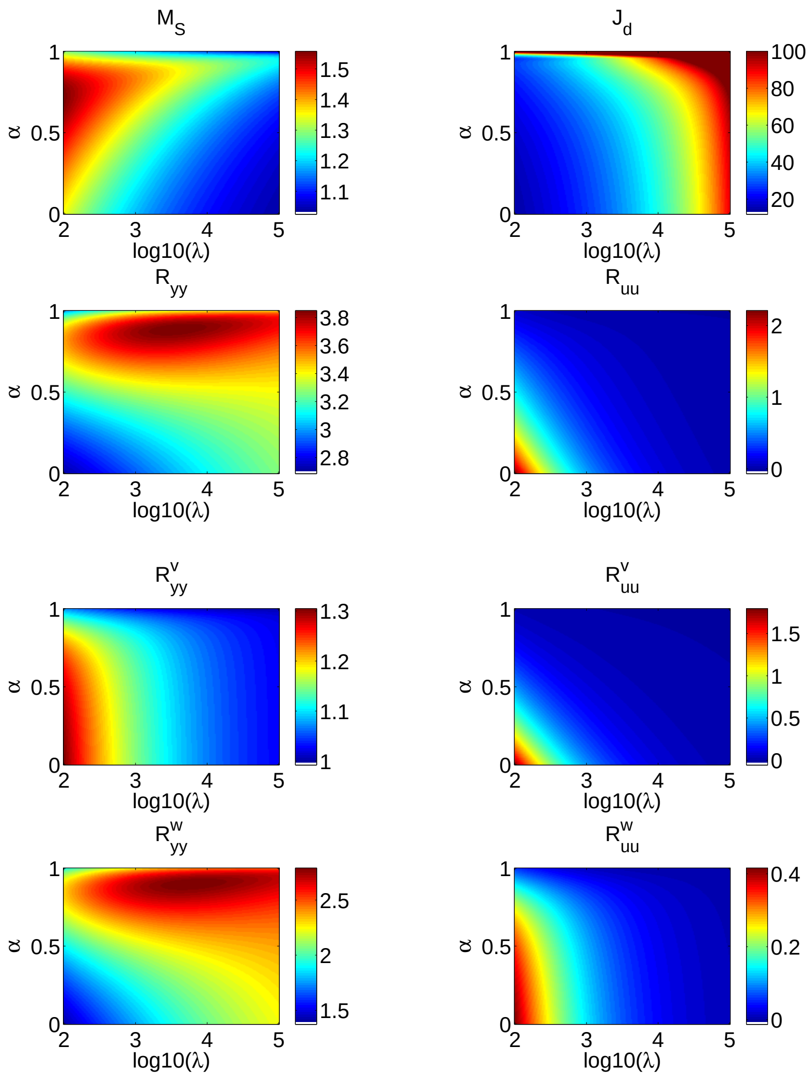


Figure 7.9: Visual interpretation of performance measures

From Figure 7.9, we can see the lack of dead time has a clear impact on M_s, where the highest value is around 1.6. The highest value of the real system is close to 5. The countour shapes of M_s, J_d and R_{yy} are notably differrent from before, and the magnitudes are generally lower. The contour shape of R_{uu} has the highest similarity with the original system, but has a large difference in magnitude. It should be noticed, that the relation between R_{yy} and R_{uu} has changed, such the highest variance of the input gives the lowest output variance.

The structure of R_{yy} primarily originates from R_{yy}^w. The structures of R_{yy}^v and R_{uu}^v is similar to the real system, but the magnitude levels differ.

It should be clear from Figure 7.9, that a system without dead time can be controlled much tighter without compromising robustness. Conclusively, we can state that the performance of any control system is always limited by the properties of the control object.

7.6 Summary

We have demonstrated how the SISO Toolbox can be used to asses control performance for a Gas-Oil furnace. A tuning has been proposed which provide a good tradeoff between deterministic and stochastic tuning objectives. We have visualized how M_S can provide a measure of robustness. It has been examined how dead time affects control performance. Finally, it was shown how the optimization based tuning procedure can be applied for the system.

Chapter 8

Wood-Berry Distillation Column

In this chapter, we investigate the optimization based tuning procedure on a Wood and Berry distillation column. Different objectives is posed as optimization problems and the results are analyzed. Finally, we propose a tuning for the system based on the results from optimization.

8.1 Process Description

We consider a Wood-Berry distillation column, which is expressed by the continuos transfer function description:

$$\begin{bmatrix} Y_1(s) \\ Y_2(s) \end{bmatrix} = \begin{bmatrix} \frac{12.8}{16.7s+1}e^{-s} & \frac{-18.9}{21.0s+1}e^{-3s} \\ \frac{6.6}{10.9s+1}e^{-7s} & \frac{-19.4}{14.4s+1}e^{-3s} \end{bmatrix} \begin{bmatrix} U_1(s) \\ U_2(s) \end{bmatrix} + \begin{bmatrix} \frac{3.8}{14.9s+1}e^{-8.1s} \\ \frac{4.9}{13.2s+1}e^{-3.4s} \end{bmatrix} (D(s) + W(s)) + V(s) \quad (8.1)$$

The process seperates methanol and water, where Y_1 is distillate methanol [mol%], Y_2 is bottom methanol [mol%], U_1 is reflux flow rate $[lb/min]$, U_2 is steam flow rate $[lb/min]$ and D is unmeasured feed flow rate $[lb/min]$.

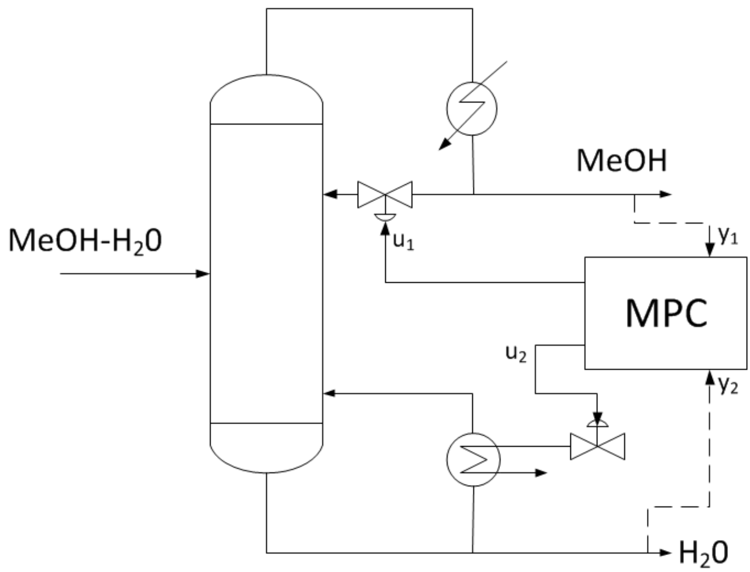


Figure 8.1: Process Diagram of a Wood-Berry Distillation Column. The Methanol-Water mixture is feeded to the distillation column. The top product is the distillated methanol and the bottom product is water. The MPC controls the reflux rate u_1 and steam flow rate u_2 to meet target concentration levels.

The time constants of the input-output matrix varies from 10.9 to 21 minutes, and dead times varies from 1 to 5 minutes. The mixture feed input has time constants of 14.9 and 13.2 minutes, and dead times of 8.1 and 3.4 minutes for the respective outputs. In Figure 8.1 a process diagram of the distillation column is shown.

The system is discretized with $T_s = 1$. The tuning algorithm requires, that a deadtime matrix is identified:

$$\tau = \begin{bmatrix} \tau_u & \tau_d & \tau_w \end{bmatrix}$$

where $\tau_u = \begin{bmatrix} 1 & 3 \\ 7 & 3 \end{bmatrix}$, $\tau_d = \begin{bmatrix} 8.1 \\ 3.4 \end{bmatrix}$ and $\tau_w = \begin{bmatrix} 8.1 \\ 3.4 \end{bmatrix}$.

The model (8.1) without deadtime is converted to a discrete state space model:

$$x_{k+1} = Ax_k + Bu_k + Ed_k + Gw_k$$

$$y_k = Cx_k + v_k, \qquad z_k = Cx_k$$

Measurement and process noise covariance is assumed to be:

$$R_{vv} = 0.01 \cdot I, \quad R_{ww} = 0.01$$

8.2 Tuning by Optimization

For this system we should determine the ARIMAX coefficients $\{\alpha_1,\ \alpha_2\}$, and determine the tuning weights Q_z and S.

There are six parameters to be determined by optimization: α_1, α_2, q_1, q_2, s_1 and s_2. We further use the bound $M_S \leq 1.775$ to ensure robustness.

The optimization problem is stated as:

$$\underset{x}{min}\ f(x)$$

s.t.

$$M_s(x) - 1.775 \leq 0$$

where $f(x)$ is the objective function and $x = \{\alpha_1, \alpha_2, q_1, q_2, s_1, s_2\}$

Bounds are placed on the tuning variables such:

$$\{\alpha_1, \alpha_2\} \in [0, 1], \quad \{q_1, q_2, s_1, s_2\} \in [1e1, 1e6]$$

The prediction horizon is set to $N = 400$ for the controller. Three different starting points has been used for the optimization. The optimization results is calculated using an active set, interior point and S.Q.P. solver. The solver details and individual results are listed in Appendix D.2.

In Table 8.1, the tuning objectives and results of the optimizations are listed.

$f(x)$	α_1, α_2	q_1, q_2	s_1, s_2	J		J_d	R_{yy}	
$\|J_d\|_\infty$	0.9653	23.54	$6.00 \cdot 10^5$	32.60	4.61	8.04	0.0117	0.0018
	0.9269	99.49	$9.02 \cdot 10^4$	19.52	13.69	18.82	0.0018	0.0177
$\|J_d\|_2$	0.9676	10.00	$7.86 \cdot 10^5$	51.89	5.00	4.91	0.0115	0.0018
	0.9202	73.49	$7.60 \cdot 10^4$	36.93	14.48	19.05	0.0018	0.0177
$\|J\|_2 + \|J_d\|_2$	0.9632	87.34	$4.87 \cdot 10^4$	9.78	2.85	7.22	0.0134	0.0019
	0.9331	57.84	$6.88 \cdot 10^4$	1.78	12.08	20.27	0.0019	0.0178
Φ_A	1.0000	10.52	$3.46 \cdot 10^5$	41.98	0.60	291.77	0.0111	0.0010
	1.0000	598.14	$9.48 \cdot 10^5$	27.21	11.66	457.71	0.0010	0.0158
Φ_D	1.0000	10.63	$2.99 \cdot 10^5$	36.98	0.46	166.63	0.0111	0.0011
	1.0000	573.46	$2.99 \cdot 10^5$	22.11	9.69	351.62	0.0011	0.0162
Φ_E	1.0000	10.00	$1.00 \cdot 10^6$	57.48	0.72	279.57	0.0112	0.0011
	1.0000	623.30	$8.14 \cdot 10^5$	41.02	11.97	463.99	0.0011	0.0157

Table 8.1: Tuning by optimization $M_s \leq 1.775$

The three solvers have in general produced consistent results for each starting point, i.e. converged to the same minimum. It should be noted, that the values generated of Q_z and S are not unique. The solution of the MPC optimization problem is not changed if the same scaling is applied on both matrices.

It can be concluded from Table 8.1 that the objectives Φ_A, Φ_D and Φ_E results in $\{\alpha_1, \alpha_2\} = 1$, i.e. both integrators are disabled. The stochastic objectives have neglible differences for output covariance, but significant differences in terms of J and J_d. The inactive integrators makes neither of them realistic tuning proposals, as off-set free control can not be obtained.

From Table 8.1 we can see that the objective $\|J\|_2 + \|J_d\|_2$ produces an attractive tuning in terms of J and J_d. The tuning although has the highest covariance for the outputs. The objectives miniminizing J_d produces the best obtainable disturbance rejections, but has poor reference tracking properties.

We suggest, that the best trade-off is obtained with $\|J\|_2 + \|J_d\|_2$, since this tuning has the best tracking properties. The cost in covariance is considered marginal compared to the other objectives.

In order to demonstrate the effect of M_S, a tuning is synthesized by minimizing $\|J\|_2 + \|J_d\|_2$ for $M_S \leq 3.5$. The tuning properties can be seen in Table 8.2.

$f(x)$	α_1, α_2	q_1, q_2	s_1, s_2	J		J_d	R_{yy}	
$\|J\|_2 + \|J_d\|_2$	0.9582	62.34	93.00	3.25	0.53	1.70	0.0357	0.0132
	0.9216	27.37	131.77	0.33	5.37	5.64	0.0132	0.0539

Table 8.2: Tuning by optimization $M_s \leq 3.5$

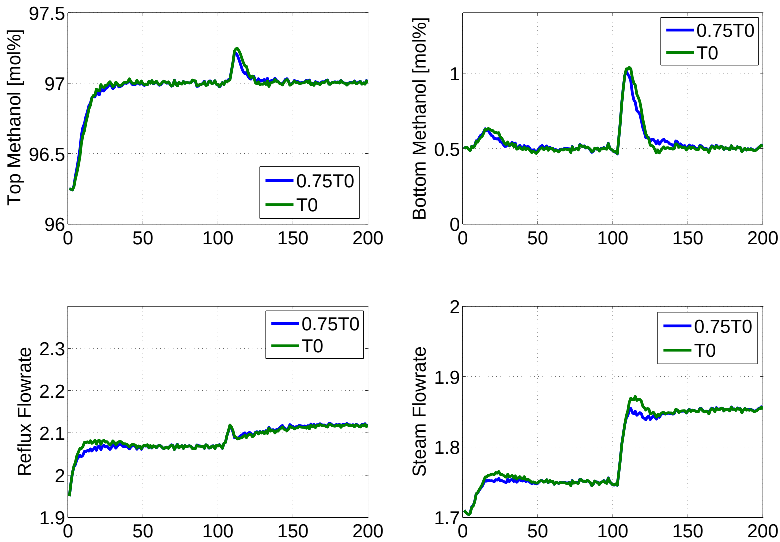


Figure 8.2: Simulation of Distillation Column. A reference change on the distillate methanol occurs at $t = 0$, and a disturbance enters the system at $t = 100$. $M_S = 1.775$.

From Table 8.2 it can be seen that better values of IAE for reference tracking and disturbance rejection is obtained, although at the expense of covariance and robustness.

8.3 Simulations

We have adopted a simulation profile from [ZWMG95]. The simulation profile features a reference step for y_1 from 96.25 $mol\%$ to 97 $mol\%$, while the setpoint for y_2 is constant on 0.5 $mol\%$. A constant feed of 2.34 lb/min is assumed for the plant with a step disturbance of 0.34 lb/min occurring at $t = 100$. It is assumed that the process noise variance is 0.01. The measurement noise is assumed to be $0.01 \cdot I$. In the simulations, we consider the nominal case and a process/model mismatch, where it assumed the the time constant is 75% of the nominal values. A simulation of the tuning minimizing $\|J\|_2 + \|J_d\|_2$ with $M_S \leq 1.775$ can be seen in Figure 8.2.

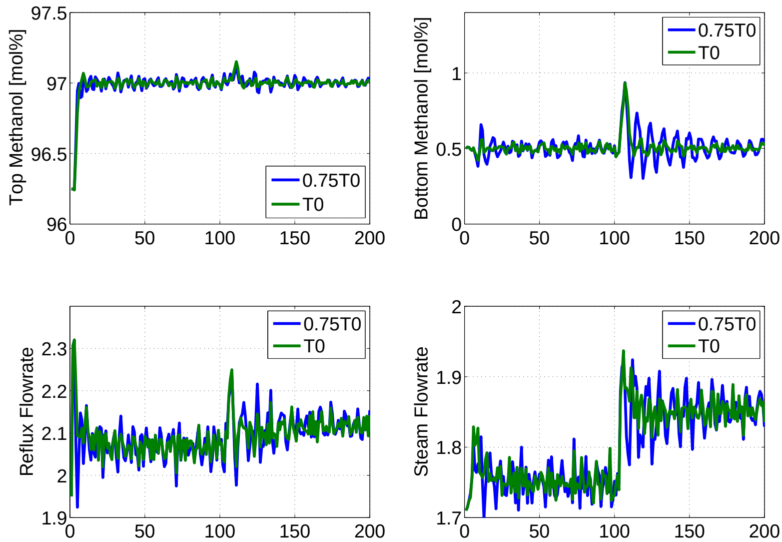


Figure 8.3: Simulation of Distillation Column. A reference change on the distillate methanol occurs at $t = 0$, and a disturbance enters the system at $t = 100$. $M_S = 3.5$.

The simulation shows, that the top methanol tracks the reference within 20 minutes. The reference change has a slight impact on the bottom methanol. The disturbance is effectively rejected for both outputs. The impact of the disturbance can be seen to greater on y_2 than y_1. This can be explained by the dead time from u_1 to y_1 is the lowest and thus allowing the controller to compensate the disturbance fastest for y_1. The simulation further shows, that the system is robust to modelling errors. In [ZWMG95] a simulation is performed for an LQR controller developed for the distillation column. In comparison the tuning we propose offers superior performance for reference tracking and disturbance rejection for both ouputs. Further, our tuning has a better suppresion of the transient on bottom methanol caused by the setpoint change on top methanol.

We have performed an identical simulation on the tuning satisfying $M_S \leq 3.5$ for comparison. The simulation is shown in Figure 8.3.

For the nominal situation it should be noted, that faster tracking rates are obtained. The tuning does have significant higher variance on each output and particularly on the control signals. In the mismatch case, it can be seen that the Bottom methanol gets an oscillatory response when the disturbance occurs. This would clearly be unacceptable in a real world situation.

8.4 Summary

We have shown how to synthesize tunings from solutions of optimization problems. The best result has been obtained by the tuning minimizing $\|J\|_2+\|J_d\|_2$ and the bound $M_S \leq 1.775$. The tuning is robust to modelling errors and have good tracking properties.

Chapter 9

Cement Mill

In this chapter, we apply the optimization based tuning procedure for a industrial cement mill circuit.

9.1 Process Description

The cement mill is inspired from [PRCJ10] and is for a nominal case described from the continuous transfer function description:

$$\begin{bmatrix} Y_1(s) \\ Y_2(s) \end{bmatrix} = \begin{bmatrix} \frac{0.62}{(45s+1)(8s+1)}e^{-5s} & \frac{0.29(8s+1)}{(2s+1)(38s+1)}e^{-1.5s} \\ \frac{-15}{60s+1}e^{-5s} & \frac{5}{(14s+1)(s+1)}e^{-0.1s} \end{bmatrix} \begin{bmatrix} U_1(s) \\ U_2(s) \end{bmatrix} + \begin{bmatrix} \frac{-1}{(32s+1)(21s+1)}e^{-3s} \\ \frac{60}{(30s+1)(20s+1)} \end{bmatrix} (D(s) + W(s)) + V(s) \quad (9.1)$$

where Y_1 is elevator load $[kW]$, Y_2 is cement fineness $[cm^2/g]$, U_1 is feed flow rate $[TPH]$, U_2 is seperator speed [%] and D is the clinker hardness $[HGI]$.

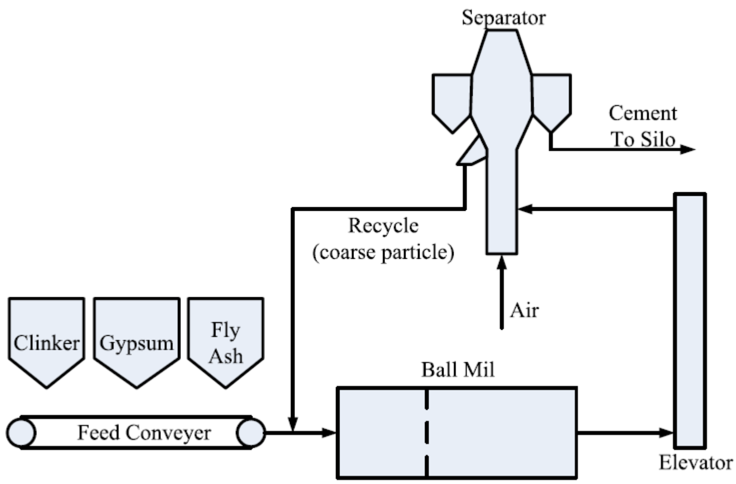


Figure 9.1: Diagram of Cement-Mill Circuit. Clinker, gypsum and fly ash is feeded into a ball mill. The outlet of the ball mill is transported into a seperator, which extracts the cement. Coarse particles are sent back into the ball mill for further grinding. *Figure from [PRCJ10]*

A process diagram of the cement mill is shown in Figure 9.1.

The cement mill system is discretized with $T_s = 2$.

The transfer function model without dead times is converted to a discrete state space model:

$$x_{k+1} = Ax_k + Bu_k + Ed_k + Gw_k$$

$$y_k = Cx_k + v_k, \qquad z_k = Cx_k$$

Dead times are organized in a matrix τ, as required for the optimization procedure. We assume the following covariances for measurement and process noise:

$$R_{vv} = \begin{bmatrix} 0.1 & 0 \\ 0 & 100 \end{bmatrix}, \quad R_{ww} = 1$$

9.2 Tuning by Optimization

The prediction horizon of the controller has been selected to $N = 400$. The objectives of the optimizations are $\|J\|_2+\|J_d\|_2$ and $\Phi_A(R_{yy})$. We use the stochastic objective to obtain a minimum for the stochastic performance of the system. For both optimization objectives we use the bound $M_S \leq 1.775$ to ensure robustness. The result of the optimizations has been produced using three different starting points and three optimization algorithms. The details of each execution is listed in Appendix D.3. The obtained results are shown in Table 9.1.

$f(x)$	α_1,α_2	q_1,q_2	s_1,s_2	J		J_d	R_{yy}	
$\|J\|_2+\|J_d\|_2$	0.9849	14.05	$1.19 \cdot 10^4$	81.63	0.85	42.54	0.3959	0.7701
	0.0000	91.58	$1.38 \cdot 10^4$	2.35	7.15	22.70	0.7701	154.16
Φ_A	1.0000	85.86	$9.68 \cdot 10^5$	104.27	13.72	1278.50	0.3353	−0.0325
	1.0000	31.37	$9.29 \cdot 10^3$	5.02	9.18	3020.08	−0.0325	115.79

Table 9.1: Tuning by optimization $M_s \leq 1.775$

From Table 9.1, the stochastic objective (Φ_A) results in both integrators to being turned off. The inactive integrators makes the configuration unable to suppres disturbances and ensure off-set free control. The deterministic objective results in full integration for y_2. The output covariance for this system is considerably worse than the stochastic objective, and is unacceptable for the application.

We can choose different strategies for obtaining off-set free tracking and improve the stochastic properties. A trial-and error approach could be used on the stochastic tuning by gradually turning on both integrators. This method does however require, that M_S is evaluated for each iteration to ensure the demands on robustness is met.

Another approach is to use optimization of the deterministic objective with a lower bound on M_S. This would arguably result in a less agressive and more robust controller. In Table 9.2, a tuning is produced for a deterministic objective with the bound: $M_S \leq 1.3$

$f(x)$	α_1,α_2	q_1,q_2	s_1,s_2	J		J_d	R_{yy}	
$\|J\|_2+\|J_d\|_2$	0.9923	382.38	$9.91 \cdot 10^5$	97.78	2.52	155.49	0.1666	0.2042
	0.8523	705.83	$4.87 \cdot 10^5$	2.13	10.68	139.75	0.2042	121.26

Table 9.2: Tuning by optimization $M_s \leq 1.3$

The tuning presented in Table 9.2, shows that the lower bound on M_S gives a better stochastic performance. The covariance matrix are closer to the stochastic tuning in Table 9.1. The disturbance rejection measure, J_d, has increased since a less agressive controller has been produced. The difference in reference tracking properties is however only slightly increased.

9.3 Simulations

We use a simulation profile, where the process is in a steady state. The initial elevator load is 26 kW and the cement finess is 3100 cm^2/g. The reference for the elevator load is at $t = 0$ increased to 30 kW. A change in clinker hardness is introduced at $t = 500$ with a relative change of $+5$ HGI.

The two deterministic tunings are simulated in a nominal and model mismatch case, where the dead times of the system has increased by 50%.

In Figure 9.2 the tuning minimizing $\|J\|_2 + \|J_d\|_2$ with $M_S \leq 1.775$ is shown.

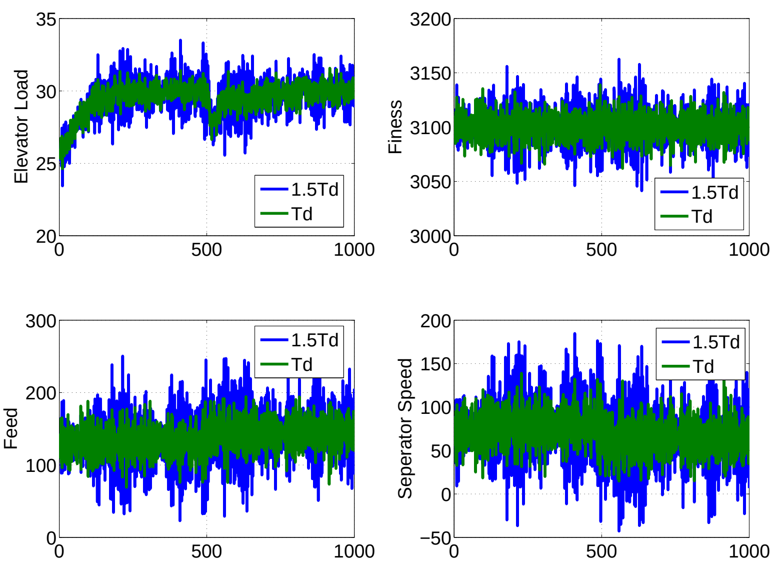


Figure 9.2: Simulation of tuning minimizing $\|J\|_2 + \|J_d\|_2$ ($M_S \leq 1.775$). The mismatch case results in increased variance for both outputs and control signals.

The simulation in Figure 9.2 shows, that the tuning provide a very good disturbance rejection. The rejection is most efficient on y_2 and can be explained from the low dead time and small dominant time constant from u_2 to y_2. The stochastic properties for this tuning is however not acceptable. The actuators on the system is stressed excessively from the high variance of the control signals. The controller remains stable for the mismatch, but the variance on outputs and control signals are significantly increased. In Figure 9.3 a simulation is shown for the tuning with the bound $M_S \leq 1.3$.

From Figure 9.3, the lower bound on M_S can be seen to have affected the disturbance rejection rates, but the stochastic properties are more desirable for this tuning. For the clinker hardness change at $t = 500$, the controller reacts more smootly by slowly changing the feed and seperator speed to obtain desired targets. The control signals has significantly reduced variance, making this tuning more friendly for systems actuators.

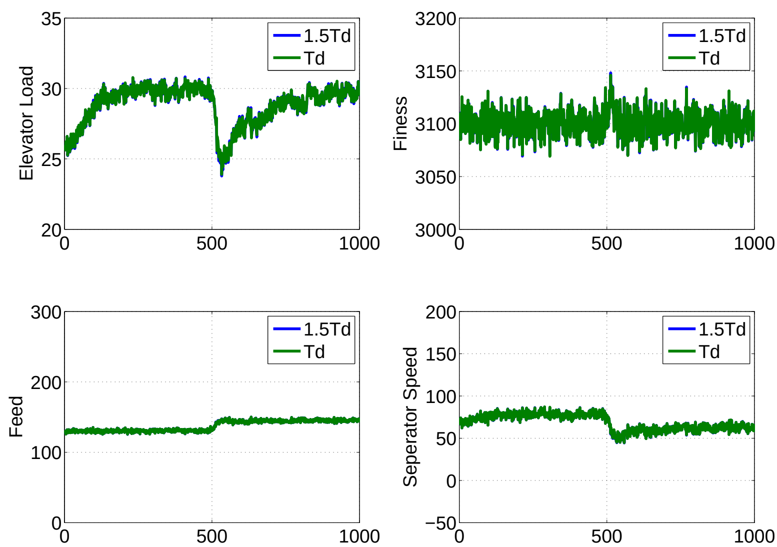


Figure 9.3: Simulation of tuning minimizing $\|J\|_2 + \|J_d\|_2$ ($M_S \leq 1.3$). The nominal and mismatch case are indistinguishable for this tuning.

9.4 Summary

We have demonstrated how the the optimization-procedure can be applied for a cement mill. The process has been more affected by sensor and process noise than the distillation column. As a result, the controller is synthesized with a lower bound on M_S to obtain better stochastic properties and less stress on actuators.

Part IV

Conclusion

Chapter 10

Conclusion

We have accomplished the following objectives:

- Derived a closed loop state space model for a process and an unconstrained ARIMAX-based MPC.
- Investigated methods to asses closed loop performance for SISO and MIMO control systems.
- Developed a toolbox for SISO systems, which allows the designer to visualize performance measures and identify the best trade-offs.
- Developed an optimization based procedure for tuning of MIMO systems.

We have investigated methods to asses closed-loop performance of ARIMAX-based predictive controllers. The most important concept has been to derive a closed loop state space model. The model has allowed us to use discrete Lyapunov equations to asses the steady state covariance on outputs and control signals. Furthermore, we have derived sensitivity functions based on the state space model, which has been used as a measure of robustness. Assesment of reference tracking and disturbance rejection properties, has been obtained by making simulations of the closed loop model.

The performance measures, has been used to develop a toolbox for tuning of SISO systems. The toolbox sweeps through the tuning parameters and evaluate the performance measures for each setting. The toolbox provides a visualization of control performance, and is intended to assist the control engineer in tuning systems for desired objectives. The applicability of the toolbox has succesfully been demonstrated for a Gas-Oil furnace.

For MIMO systems we proposed, that a constrained optimization tuning procedure could be used. We have succesfully tested the procedure on a Wood-Berry distillation column and a cement mill. For both systems it was demonstrated, that a optimality-based approach is a sensible tuning strategy. The best results has been obtained using $\|J\|_2+\|J_d\|_2$ as the optimization objective.

We have performed nine executions of the optimization algorithm for each objective, i.e. by using three starting points and an active set, interior point and S.Q.P. solver. This has been done to validate if the same solutions was found for different solvers and starting points. In general, the consistency of the solvers have been reasonable good. We were able to validate if a global minimum was isolated by application of the optimization procedure on the Gas-Oil furnace. It was shown that the global minimum for R_{yy} was isolated, but not for J_d. It has not been possible to make similar evaluations for the MIMO systems. Regardless of global optimality has been achieved, we conclude that the tuning approach have produced good tuning proposals for both MIMO systems.

Chapter 11

Future Research

The optimization based tuning procedure must be considered to be in an early developement phase. Despite encouraging results, we have not been able to determine, which optimization algorithm has the best performance. Furthermore, it remains unknown how the procedure will perform on systems with a higher number of inputs and outputs.

Another future research direction could be to investigate if PSO would be a better strategy for the optimization problem. PSO is a metaheuristic method, which can search large spaces of candidate solutions and do not require gradients. The method is normally not associated with constrained optimization, but it has been applied succesfully on inequality constrained optimization problems [PV02].

Part V

Appendix

Appendix A

Relation between $S(z)$ and R^v_{yy}

In this section an explanation of the relation between $S(z)$ and R^v_{yy} is given.

The steady state output variance caused by the measurement noise can be calculated using the discrete Lyapunov Equation

$$R^v_{xx} = A_{cl} R^v_{xx} A^T_{cl} + B_{vcl} R_{vv} B^T_{vcl}$$

$$R^v_{yy} = C_{ycl} R^v_{xx} C^T_{ycl} + \sigma^2_v$$

The Sensitivity function $S(z)$ is defined as:

$$S(z) = \frac{Y(z)}{V(z)} = C_{ycl}(zI - A_{cl})^{-1} B_{vcl} + I$$

$V(z)$ is the Z-transform of $v(k) \sim N_{iid}(0, R_{vv})$. The autocorrelation function of $v(k)$ is $r_{vv}(k) = R_{vv}\delta(k)$, where $\delta(k)$ is the kronecker delta function. From

Wiener–Khinchins theorem, we can describe the measurement noise spectral density:

$$S_{vv}(i\omega) = \sum_{k=-\infty}^{\infty} r_{vv}(k)e^{-ik\omega} = R_{vv}e^{-i0} = R_{vv}$$

The spectral density of the output can be calculated as:

$$S_{yy}(i\omega) = S(e^{i\omega T_s})R_{vv}S(e^{-i\omega T_s})^T$$

The autocorrelation function of the output r_{yy} can be found as the inverse fourier transform of S_{yy}

$$r_{yy}(k) = \frac{1}{2\pi}\int_{2\pi} S_{yy}(\omega)e^{-ik\omega}d\omega$$

The output variance can be extracted as:

$$R^v_{yy} = r_{yy}(0) = \frac{1}{2\pi}\int_{2\pi} S_{yy}(\omega)d\omega$$

We have shown how the output variance can be calculated from the closed loop state space model using a discrete Lyapunov equation. Equivalently, we have shown how the output variance can be derived from the spectral density of $S(z)$.

Appendix B

State Elimination Method for Unconstrained MPC

In this section it is shown how an unconstrained MPC (equality constrained QP) is calculated. It is further shown how the solution can be calculated as an factorization of a KKT system and how a state-elimination technique can be applied.

$$min\ \phi = \frac{1}{2}\sum_{k=0}^{N-1} \parallel z_{k+1} - r_{k+1} \parallel_{Q_Z}^2 + \parallel \Delta u_k \parallel_S^2 \tag{B.1}$$

s.t.

$$x_{k+1} = Ax_k + Bu_k \tag{B.2a}$$

$$z_k = C_z x_k \tag{B.2b}$$

Where N is the prediction horizon and Q_z is a weight penalizing deviations from the set point. The second term in the objective function is a regularization term, where S is a penalty on movement of the control signal.The objective function have strong similarities with the standard Linear Quadratic control problem. If the reference briefly is regarded as zero, the term originating from index k, can be described as:

$$\begin{aligned} x_k^T C_z^T Q_z C_z x_k + \Delta u_k^T S \Delta u_k &= x_k^T C_z^T Q_z C_z x_k + (u_k - u_{k-1})^T S (u_k - u_{k-1}) \\ &= x_k^T C_z^T Q_z C_z x_k + u_k^T S u_k - u_k^T S u_{k-1} - u_{k-1}^T S u_k + u_{k-1}^T S u_{k-1} \end{aligned}$$

The stage cost for this problem can be defined as:

$$l_n(x_k, u_{k-1}, u_k) = \begin{bmatrix} x_k \\ u_{k-1} \\ u_k \end{bmatrix}^T \begin{bmatrix} Q & 0 & 0 \\ 0 & S & -S \\ 0 & -S & S \end{bmatrix} \begin{bmatrix} x_k \\ u_{k-1} \\ u_k \end{bmatrix} \tag{B.3}$$

Where $Q = C_z^T Q_z C_z$ The definition of the stage cost allows the possibility to state the MPC objective function as:

$$min\ \phi = \frac{1}{2} \sum_{0}^{N-1} l_k(x_k, u_{k-1}, u_k) + \frac{1}{2} x_N^T P_N x_N \tag{B.4}$$

s.t.

$$x_{k+1} = A x_k + B u_k \tag{B.5}$$

The LQ problem can be formulated as a QP

$$min\ \phi = \frac{1}{2} x^T H x + g^T x \tag{B.6}$$

Where

$$x = \begin{bmatrix} u_0 \\ x_1 \\ u_1 \\ x_2 \\ u_2 \\ \vdots \\ x_{N-2} \\ u_{N-2} \\ x_{N-1} \\ u_{N-1} \\ x_N \end{bmatrix}, \; H = \begin{bmatrix} 2S & & -S & & & & & & & & \\ & Q & & & & & & & & & \\ -S & & 2S & & -S & & & & & & \\ & & & Q & & & & & & & \\ & & -S & & 2S & & \ddots & & & & \\ & & & & & \ddots & & -S & & & \\ & & & & \ddots & & Q & & & & \\ & & & & & -S & & 2S & & -S & \\ & & & & & & & & Q & & \\ & & & & & & & -S & & S & \\ & & & & & & & & & & P_N \end{bmatrix},$$

$$g = \begin{bmatrix} -Su_{-1} \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The proof of equivalence is left as an excercise for the reader. It should however be noticed that the terms $x_0^T Q x_0$, $u_{-1}^T S u_{-1} 4$ and $-u_0^T S u_{-1}$ is left out in the QP. This is justified, since x_0 and u_{-1} is initial conditions and not decision variables. In both cases the constraints can be expressed as:

$$c(x) = A^T x - b = \begin{bmatrix} B_0 & -I & 0 & 0 & 0 & 0 \\ 0 & A_1 & B_1 & -I & 0 & 0 \\ 0 & 0 & 0 & A_2 & B_2 & -I \end{bmatrix} \begin{bmatrix} u_0 \\ x_1 \\ u_1 \\ x_2 \\ u_2 \\ x_3 \end{bmatrix} - \begin{bmatrix} -A_0 x_0 \\ 0 \\ 0 \end{bmatrix} = 0$$

The quadratic program and the constraints can thus be solved by a solution of the associated KKT system of linear equations:

$$\begin{bmatrix} H & -A \\ -A^T & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = - \begin{bmatrix} g \\ b \end{bmatrix} \tag{B.7}$$

Often a more compact notation and solution can be adopted, in which the states of the system only contains the control signals from $u_0, ..., u_{N-1}$. The notation uses the concept of Markov Parameters and builds on describing the output of a system as the sum of forced and natural response:

$$Z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_N \end{bmatrix} = \begin{bmatrix} C_zA \\ C_zA^2 \\ C_zA^3 \\ \vdots \\ C_zA^N \end{bmatrix} x_0 + \begin{bmatrix} H_1 & 0 & 0 & 0 & 0 \\ H_2 & H_1 & 0 & 0 & 0 \\ H_3 & H_2 & H_1 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ H_N & H_{N-1} & H_{N-2} & \cdots & H_1 \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{bmatrix}$$
$$= \Phi x_0 + \Gamma_U$$

where H is the Markov parameters (impulse response coefficients) defined as:

$$\begin{aligned} H_i &= 0 && for\ i = 0 \\ H_i &= C_zA^{i-1}B && for\ i > 0 \end{aligned}$$

An equivalent matrix formulation of without the regularization term is:

$$\phi = \frac{1}{2} \parallel Z - R \parallel_{Qz}^2$$

where R is a vector of reference values $R = [\ r_1 \quad r_2 \quad \cdots \quad r_{N-1} \quad r_N\ ]^T$

This notation corresponds to the QP

$$\underset{U}{min}\ \phi = \frac{1}{2}U^THU + g^TU$$

in which,

$$\begin{aligned} H &= \Gamma^TQ_z\Gamma, \\ g &= -\Gamma^TQ_zb = -\Gamma^TQ_z(R - \Phi x_0) \end{aligned}$$

The regularization term can be expressed as:

$$\phi_{\Delta u} = \frac{1}{2}\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}^T \begin{bmatrix} 2S & -S & & & \\ -S & 2S & -S & & \\ & -S & 2S & -S & \\ & & -S & 2S & -S \\ & & & -S & S \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} + \left(- \begin{bmatrix} S \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} u_{-1} \right)^T \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} + \frac{1}{2} u_{-1} S u_{-1} = U^T H_S U + (M_{u-1} u_{-1})^T U + \frac{1}{2} u_{-1} S u_{-1}$$

and included in the program by: $H = \Gamma^T Q_z \Gamma + H_s, \qquad g = -\Gamma^T Q_z b = -\Gamma^T Q_z (R - \Phi x_0) + M_{u-1} u_{-1}$ It should be noted the equality constraints imposed by the system dynamics is implicit described in H and g and thereby not needed to be stated explicit as previously.

Appendix C

Non-Linear Optimization

A fundamental element in constrained optimization theory is the the lagrange function (C.1)

$$L(x, \lambda) = f(x) - \pi^T c(x) \tag{C.1}$$

Where λ is known as a lagrange multiplier. A lagrange multiplier exist for each constraint.

It is required that the following conditions holds true for x^* to be a local minimizer:

$$\nabla_x L(x^*, \pi^*) = \nabla f(x^*) - \nabla c(x^*)\pi^* = 0 \tag{C.2a}$$

$$c(x^*) \leq 0 \tag{C.2b}$$

$$\pi^* \leq 0 \tag{C.2c}$$

$$\pi^* c(x^*) \leq 0 \tag{C.2d}$$

The conditions in (C.2) is known as the necessary 1. order KKT conditions. If the objective funtion is convex the conditions is also sufficient. The KKT conditions examines the first derivatives of the objective function and constraints at x^*. If any arbitrary small step h is taken from x^* in a feasible direction, the first order approximation $f(x^*)+\nabla f(x^*)^T h$ of the ojective function $f(x^*+h)$ will either increase or remain constant. In the latter case second order derivatives provides additionally information. It can be checked whether x^* is a strict local minimum from the sufficient condition:

$$h^T \nabla^2_{xx} L(x^*, \pi^*) h > 0 \tag{C.3}$$

in which h denotes all feasible directions from x^*.

Algorithms which solves NLP problems of the form (6.8) are typically based on the 1. and 2. order conditions. A common type of solver is the Sequential Quadratic Programming (SQP) solver, which at the k-th iterate solves a quadratic subproblem C.4.

$$\min_p \frac{1}{2} p^T \nabla^2_{xx} L(x_k, \pi_k) p + \nabla f(x_k)^T p \tag{C.4a}$$

s.t.

$$\nabla c(x)^T p + c(x) \leq 0 \tag{C.4b}$$

$$x_{k+1} = x_k + p \tag{C.4c}$$

The algorithm continues until one or more stopping criteria are met.

Appendix D

Test of Convergence for NLP solvers

D.1 Oil-Gas Furnace

Tuning/system settings:

$\alpha \in [0, 1]$, $\lambda \in [10^2, 10^6]$, $M_{s,max} = 1.775$

$N = 150$

$T_s = 2$

Solver options:

$tolFun = 10^{-8}$, $TolX = 10^{-10}$ and $MaxFun = 1000$

We use 3 different starting points for the solver:

$x_0(1) = [\ 0.5 \quad 2\ ]$
$x_0(2) = [\ 0.5 \quad 4\ ]$
$x_0(3) = [\ 0.5 \quad 6\ ]$

D.1.1 Optimization of J_d

$f(x) = J_d(\alpha, log(\lambda))$

Algorithm	iter.	f. eval.	x_0	x		$f(x)$
I.P.	20	75	$x_0(1)$	0.0000	4.9017	196.7630
	11	47	$x_0(2)$	0.0000	4.9017	196.7631
	22	82	$x_0(3)$	0.0000	4.9017	196.7630
A.S.	5	20	$x_0(1)$	0.0000	4.9017	196.7630
	5	18	$x_0(2)$	0.0000	4.9017	196.7630
	4	15	$x_0(3)$	0.0000	4.9017	196.7630
S.Q.P.	5	19	$x_0(1)$	0.0000	4.9017	196.7630
	6	21	$x_0(2)$	0.0000	4.9017	196.7630
	6	25	$x_0(3)$	0.0000	4.9017	196.7630

Table D.1: Test of algorithms for J_d and $M_S \leq 1.775$

D.1.2 Optimization of R_{yy}

$f(x) = R_{yy}(\alpha, log(\lambda))$

Algorithm	iter.	f. eval.	x_0	x		$f(x)$
I.P.	20	80	$x_0(1)$	1.0000	6.0000	3.4692
	20	74	$x_0(2)$	1.0000	6.0000	3.4692
	18	66	$x_0(3)$	1.0000	6.0000	3.4692
A.S.	3	12	$x_0(1)$	1.0000	6.0000	3.4692
	1	6	$x_0(2)$	1.0000	6.0000	3.4692
	2	9	$x_0(3)$	1.0000	6.0000	3.4692
S.Q.P.	5	18	$x_0(1)$	1.0000	6.0000	3.4692
	2	9	$x_0(2)$	1.0000	6.0000	3.4692
	3	12	$x_0(3)$	1.0000	6.0000	3.4692

Table D.2: Test of algorithms for R_{yy} and $M_S \leq$ 1.775

D.2 Wood-Berry Distillation Column

Tuning/system settings:

$(\alpha_1, \alpha_2) \in [0, 1]$, $(q_1, q_2) \in [10^1, 10^6]$, $(s_1, s_2) \in [10^1, 10^6]$, $M_{s,max} = 1.775$

$N = 400$

$T_s = 1$

Solver options:

$tolFun = 10^{-8}$, $TolX = 10^{-10}$ and $MaxFun = 1000$

The tuning parameters are organized as

$x = [\ \alpha_1 \quad \alpha_2 \quad log(q_1) \quad log(q_2) \quad log(s_1) \quad log(s_2)\ ]$

We use 3 different starting points for the solver:

$$
\begin{aligned}
x_0(1) &= [\ 0.99 \quad 0.99 \quad 1.01 \quad 1.01 \quad 1.01 \quad 1.01\ ] \\
x_0(2) &= [\ 0.99 \quad 0.99 \quad 5.99 \quad 5.99 \quad 5.99 \quad 5.99\ ] \\
x_0(3) &= [\ 0.01 \quad 0.01 \quad 3.00 \quad 3.00 \quad 3.00 \quad 3.00\ ]
\end{aligned}
$$

D.2.1 Optimization of $\|J_d\|_\infty$

Algorithm	iter.	f. eval.	x_0	x						$f(x)$
I.P.	62	513	$x_0(1)$	0.9653	0.9272	1.3145	1.9349	5.7030	4.8915	18.8312
	83	725	$x_0(2)$	0.9653	0.9272	1.3094	1.9297	5.6979	4.8863	18.8312
	38	394	$x_0(3)$	0.4341	0.8789	1.3169	1.7648	5.4455	5.1340	23.0262
A.S.	86	685	$x_0(1)$	0.9653	0.9269	1.2524	1.8796	5.6594	4.8360	18.8271
	88	733	$x_0(2)$	0.9653	0.9269	1.5777	2.2039	5.9849	5.1614	18.8271
	67	533	$x_0(3)$	0.9653	0.9269	1.3718	1.9978	5.7782	4.9552	18.8271
S.Q.P.	83	784	$x_0(1)$	0.9653	0.9269	1.5880	2.2143	5.9959	5.1718	18.8227
	81	710	$x_0(2)$	0.9663	0.7364	1.0000	2.2492	6.0000	5.9331	18.7942
	35	452	$x_0(3)$	0.9653	0.9269	1.2292	1.8553	5.6360	4.8127	18.8271

Table D.3: Test of algorithms for $\|J_d\|_\infty$ and $M_S \leq 1.775$

D.2.2 Optimization of $\|J_d\|_2$

Algorithm	iter.	f. eval.	x_0	x						$f(x)$
I.P.	101	861	$x_0(1)$	0.9676	0.9203	1.0049	1.8734	5.9000	4.8863	19.6661
	76	647	$x_0(2)$	0.9675	0.9205	1.0592	1.9281	5.9418	4.9404	19.671
	59	508	$x_0(3)$	0.9642	0.6772	1.0000	2.1552	5.5740	6.0000	19.671
A.S.	86	682	$x_0(1)$	0.9676	0.9202	1.0839	1.9519	5.9793	4.9648	19.6661
	57	468	$x_0(2)$	0.9676	0.9202	1.0001	1.8680	5.8954	4.8809	19.6661
	67	533	$x_0(3)$	0.9642	0.6772	1.0000	2.1552	5.5740	6.0000	19.6328
S.Q.P.	70	825	$x_0(1)$	0.9676	0.9202	1.0101	1.8783	5.9053	4.8912	19.6661
	32	257	$x_0(2)$	0.9660	0.7533	1.0000	2.3398	6.0000	6.0000	19.6722
	47	507	$x_0(3)$	0.9676	0.9202	1.0131	1.8812	5.9084	4.8940	19.6661

Table D.4: Test of algorithms for $\|J_d\|_2$ and $M_S \leq 1.775$

D.2.3 Optimization of $\|J\|_2$+$\|J_d\|_2$

Algorithm	iter.	f. eval.	x_0	x						$f(x)$
I.P.	40	349	$x_0(1)$	0.9632	0.9330	2.1345	1.9555	4.8817	5.0324	35.08
	39	368	$x_0(2)$	0.9632	0.9331	2.1115	1.9324	4.8579	5.0082	35.06
	65	558	$x_0(3)$	0.9632	0.9331	2.1404	1.9615	4.8868	5.0374	35.06
A.S.	41	356	$x_0(1)$	0.9632	0.9331	1.9412	1.7622	4.6874	4.8378	35.06
	67	539	$x_0(2)$	0.9632	0.9331	1.2134	1.0343	3.9595	4.1100	35.06
	67	565	$x_0(3)$	0.9632	0.9331	2.6010	2.4220	5.3472	5.4976	35.06
S.Q.P.	52	732	$x_0(1)$	0.9632	0.9331	1.2755	1.0964	4.0216	4.1721	35.06
	39	457	$x_0(2)$	0.9632	0.9331	1.7055	1.5264	4.4516	4.6021	35.06
	48	625	$x_0(3)$	0.9632	0.9331	1.6754	1.4964	4.4216	4.5720	35.06

Table D.5: Test of algorithms for $\|J\|_2$+$\|J_d\|_2$ and $M_S \leq 1.775$

D.2.4 Optimization of Φ_E

Algorithm	iter.	f. eval.	x_0	x						$f(x)$
I.P.	123	969	$x_0(1)$	1.0000	1.0000	1.0000	2.7947	5.9106	6.0000	0.015968
	117	935	$x_0(2)$	1.0000	1.0000	1.0000	2.7947	5.9104	6.0000	0.015968
	84	638	$x_0(3)$	1.0000	1.0000	1.0030	2.7917	5.9020	5.9969	0.015968
A.S.	27	200	$x_0(1)$	1.0000	1.0000	1.0000	2.7947	5.9107	6.0000	0.015968
	20	149	$x_0(2)$	1.0000	1.0000	1.0000	2.7946	5.9099	6.0000	0.015968
	53	380	$x_0(3)$	1.0000	1.0000	1.0000	2.7947	5.9099	6.0000	0.015968
S.Q.P.	30	324	$x_0(1)$	1.0000	1.0000	1.0000	2.7947	5.9106	6.0000	0.015968
	27	379	$x_0(2)$	1.0000	0.5829	1.0000	1.2456	4.9955	6.0000	0.016362
	41	373	$x_0(3)$	1.0000	1.0000	1.0000	2.7947	5.9106	6.0000	0.015968

Table D.6: Test of algorithms for $\Phi_E(R_{yy})$ and $M_S \leq 1.775$

D.2.5 Optimization of Φ_A

Algorithm	iter.	f. eval.	x_0	x						$f(x)$
I.P.	114	930	$x_0(1)$	1.0000	1.0000	1.0191	2.7786	5.5418	5.9795	0.026852
	65	472	$x_0(2)$	1.0000	1.0000	1.0219	2.7768	5.5387	5.9770	0.026852
	104	773	$x_0(3)$	1.0000	1.0000	1.0220	2.7768	5.5387	5.9771	0.026852
A.S.	17	127	$x_0(1)$	1.0000	1.0000	1.0000	2.7596	5.5227	5.9605	0.026852
	23	180	$x_0(2)$	0.9961	0.5798	1.0000	1.2633	4.7672	6.0000	0.027530
	15	113	$x_0(3)$	0.9961	0.5797	1.0000	1.2634	4.7671	6.0000	0.027530
S.Q.P.	30	266	$x_0(1)$	1.0000	1.0000	1.0000	2.7596	5.5228	5.9605	0.026852
	31	305	$x_0(2)$	1.0000	1.0000	1.0001	2.7597	5.5228	5.9606	0.026852
	47	495	$x_0(3)$	0.9961	0.5798	1.0000	1.2633	4.7671	6.0000	0.027530

Table D.7: Test of algorithms for $\Phi_A(R_{yy})$ and $M_S \leq 1.775$

D.2.6 Optimization of Φ_D

Algorithm	iter.	f. eval.	x_0	x						$f(x)$
I.P.	83	648	$x_0(1)$	1.0000	1.0000	1.0279	2.7597	5.4781	5.9685	0.017373
	98	788	$x_0(2)$	1.0000	1.0000	1.0288	2.7606	5.4790	5.9694	0.017373
	96	1000	$x_0(3)$	0.9943	0.5928	1.0002	1.3007	4.7639	5.9998 [1]	0.018275
A.S.	34	272	$x_0(1)$	1.0000	1.0000	1.0266	2.7585	5.4767	5.4767	0.017373
	23	180	$x_0(2)$	1.0000	1.0000	1.0010	2.7328	5.4512	5.9416	0.017373
	22	172	$x_0(3)$	1.0000	1.0000	1.0592	2.7912	5.5096	6.0000	0.017373
S.Q.P.	27	313	$x_0(1)$	0.9911	0.6734	1.0000	1.5990	6.0000	6.0000	0.018694
	25	328	$x_0(2)$	0.9911	0.6734	1.0000	1.5990	6.0000	6.0000	0.018694
	51	541	$x_0(3)$	1.0000	1.0000	1.0263	2.7582	5.4766	5.9670	0.017373

[1] Solver stopped prematurely

Table D.8: Test of algorithms for $\Phi_D(R_{yy})$ and $M_S \leq 1.775$

D.2.7 Optimization of $\|J\|_2+\|J_d\|_2$ ($M_S \leq 3.5$)

Algorithm	iter.	f. eval.	x_0	x						$f(x)$
I.P.	43	431	$x_0(1)$	0.9582	0.9216	3.4745	3.1050	3.6462	3.7926	11.37
	41	394	$x_0(2)$	0.9582	0.9217	3.4160	3.0563	3.5959	3.7454	11.37
	69	619	$x_0(3)$	0.9582	0.9217	3.4606	3.1015	3.6410	3.7908	11.37
A.S.	35	319	$x_0(1)$	0.9582	0.9216	1.9537	1.5958	2.1272	2.2784	11.35
	41	419	$x_0(2)$	0.9582	0.9216	5.4599	5.1023	5.6332	5.7847	11.35
	47	356	$x_0(3)$	0.7644	0.0000	2.3688	2.5583	3.9693	5.0557	15.72
S.Q.P.	68	797	$x_0(1)$	0.8114	0.0000	1.1806	1.4009	2.7811	3.8790	15.72
	80	814	$x_0(2)$	0.9582	0.9216	1.7948	1.4372	1.9685	2.1198	11.35
	60	864	$x_0(3)$	0.9582	0.9216	2.9613	2.6045	3.1350	3.2869	11.35

Table D.9: Test of algorithms for $\|J\|_2+\|J_d\|_2$ and $M_S \leq 1.775$

D.3 Cement-Mill

Tuning/system settings:

$\alpha \in [0,1]$, $\lambda \in [10^2, 10^5]$, $M_{s,max} = 1.775$

$nbs = 400$

$T_s = 2$

Solver options:

$tolFun = 10^{-8}$, $TolX = 10^{-10}$ and $MaxFun = 1200$

The tuning parameters are organized as

$x = [\ \alpha_1 \quad \alpha_2 \quad log(q_1) \quad log(q_2) \quad log(s_1) \quad log(s_2)\ ]$

We use 3 different starting points for the solver:

$$\begin{aligned} x_0(1) &= [\ 0.99 \quad 0.99 \quad 1.01 \quad 1.01 \quad 1.01 \quad 1.01\ ] \\ x_0(2) &= [\ 0.99 \quad 0.99 \quad 5.99 \quad 5.99 \quad 5.99 \quad 5.99\ ] \\ x_0(3) &= [\ 0.01 \quad 0.01 \quad 3.00 \quad 3.00 \quad 3.00 \quad 3.00\ ] \end{aligned}$$

D.3.1 Optimization of $\|J\|_2 + \|J_d\|_2$

Algorithm	iter.	f. eval.	x_0	x						$f(x)$
I.P.	58	542	$x_0(1)$	0.9849	0.0000	1.1477	1.9618	4.0743	4.1404	129.93
	49	434	$x_0(2)$	0.9849	0.0000	2.9367	3.7509	5.8633	5.9295	129.93
	122	1195	$x_0(3)$	0.9872	1.0000	1.0008	1.7436	6.0000	1.5795 [1]	167.45
A.S.	93	767	$x_0(1)$	0.9856	0.6076	1.4122	2.1193	4.2269	4.3199	134.69
	84	675	$x_0(2)$	0.9856	0.6233	1.3017	2.0189	4.1091	4.2041	134.69
	51	408	$x_0(3)$	0.9856	0.6279	1.0798	1.8004	3.8851	3.9806	134.69
S.Q.P.	47	644	$x_0(1)$	0.9856	0.6201	1.4288	2.1443	4.2378	4.3323	134.69
	49	535	$x_0(2)$	0.9856	0.6175	1.5089	2.2223	4.3191	4.4133	134.69
	51	551	$x_0(3)$	0.9856	0.6271	3.0990	3.8189	5.9046	6.0000	134.69

[1] solver stopped prematurely (MaxFun exceeded)

Table D.10: Test of algorithms for $\|J\|_2 + \|J_d\|_2$ and $M_S \leq 1.775$

D.3.2 Optimization of Φ_A

Algorithm	iter.	f. eval.	x_0	x						$f(x)$
I.P.	44	408	$x_0(1)$	1.0000	1.0000	1.8743	1.4364	5.9187	3.9080	116.13
	77	707	$x_0(2)$	1.0000	1.0000	1.8739	1.4356	5.9221	3.9071	116.13
	131	1200	$x_0(3)$	1.0000	1.0000	1.7969	1.3618	5.8368	3.8334 [1]	116.13
A.S.	28	210	$x_0(1)$	1.0000	1.0000	1.9338	1.4965	5.9858	3.9681	116.13
	23	174	$x_0(2)$	1.0000	1.0000	1.8870	1.4471	6.0000	3.9188	116.13
	29	231	$x_0(3)$	1.0000	1.0000	1.7363	1.2980	5.7845	3.7696	116.13
S.Q.P.	18	208	$x_0(1)$	1.0000	0.9948	1.6606	2.8623	6.0000	5.4258	116.79
	22	201	$x_0(2)$	1.0000	0.7390	2.4596	1.0000	6.0000	4.5182	117.45
	32	364	$x_0(3)$	1.0000	1.0000	1.9079	1.4697	5.9563	3.9412	116.13

[1] solver stopped prematurely (MaxFun exceeded)

Table D.11: Test of algorithms for $\Phi_A(R_{yy})$ and $M_S \leq 1.775$

D.3.3 Optimization of $\|J\|_2+\|J_d\|_2$ ($M_S \leq 1.3$)

Algorithm	iter.	f. eval.	x_0	x						$f(x)$
I.P.	70	621	$x_0(1)$	0.9923	0.8504	1.1057	1.3721	4.5192	4.2116	306.07
	51	549	$x_0(2)$	0.9931	0.0547	1.7875	1.8948	5.4504	5.1563	315.18
	124	1196	$x_0(3)$	0.9923	0.8503	2.5825	2.8487	5.9959	5.6890 [1]	306.07
A.S.	61	1206	$x_0(1)$	0.8926	0.8786	6.0266	5.5699	5.0479	1.4301 [2]	33.24
	94	808	$x_0(2)$	0.9923	0.8504	2.5750	2.8415	5.9885	5.6809	306.91
	43	351	$x_0(3)$	0.9602	0.5590	1.0000	1.9387	6.0000	6.0000	726.73
S.Q.P.	32	410	$x_0(1)$	0.9923	0.8505	1.3119	1.5784	4.7253	4.4175	306.90
	44	560	$x_0(2)$	0.9923	0.8501	1.0046	1.2702	4.4184	4.1107	306.90
	57	589	$x_0(3)$	0.9923	0.8505	1.0020	1.2685	4.4154	4.1078	306.90

[1] solver stopped prematurely (MaxFun exceeded)
[2] solver stopper prematurely and unfeasible solution

Table D.12: Test of algorithms for $\|J\|_2+\|J_d\|_2$ and $M_S \leq 1.3$

Appendix E

MATLAB code

The MATLAB code in this section is organized in the following manner:

- Simulation Files
  - Industrial Furnace
  - Wood-Berry Distillation column
  - Cement-Mill
  - Common files (Simulation functions)
- Toolbox
  - SISO Toolbox
    - Initiation of Toolbox: Industrial Furnace
    - SISO Toolbox algorithm
    - SISO Optimization Procedure
  - MIMO Optimization Toolbox
    - Initiation of Toolbox: Wood-Berry Distillation Column
    - Initiation of Toolbox: Cement-Mill
    - MIMO Optimization algorithm
  - Common files

E.1 Simulation Files

E.1.1 Industrial Furnace

```
%---------------------------------------------------------------------%
% Simulation of Industrial Furnace
%
% Daniel Olesen s100094, DTU
%---------------------------------------------------------------------%

clear all;
%close all;
clc;

%---------------------------------------------------------------------%
% Nominal system / Basis for controller model
%---------------------------------------------------------------------%

addpath MPC_tuning\Realization
addpath MPC_tuning\MPC_dir

Ts=2;

G_num = 20;
G_den = [160 44 1];
G_tau = 50;

H_num = -5;
H_den = [25 10 1];
H_tau = 10;

nbs_range=[100];

% Sample interval

% Initialize cell vector required for ss computation function

num=cell(1,2); den=cell(1,2); tau=zeros(1,2);
num{1}=G_num; num{2}=H_num;
den{1}=G_den; den{2}=H_den;
tau(1)=G_tau; tau(2)=H_tau;

% Conditions for computation function

Nmax=100; tol=1e-8;

[Ad,Bd,Cd,Dd,sH]=mimoctf2dss(num,den,tau,Ts,Nmax,tol);

%---------------------------------------------------------------------%
% Converting to Armax representation
%---------------------------------------------------------------------%

```

```
z=tf('z');

Gzu=c2d(tf(G_num,G_den),Ts)*z^(-G_tau/Ts);
Gzd=c2d(tf(H_num,H_den),Ts)*z^(-H_tau/Ts);
Gzw=Gzd;

alpha_range=[0.7]

for alpha=alpha_range

A_arx = cell2mat(Gzu.den);
B_arx = cell2mat(Gzu.num);
B_arx = B_arx(2:end);

C_arx=1;
F_arx=[1 -1];                   % [1 -q^-]
G_arx=[1 -alpha];               % [1 -alpha*q^-1]

A_arx_m = [A_arx 0] - [0 A_arx]; % F_arx*A_arx = [1 -q^-1]*A_arx
B_arx_m = [B_arx 0] - [0 B_arx]; % F_arx*B_arx = [1 -q^-1]*B_arx
C_arx_m=G_arx;

%--------------------------------------------------------------------------%
% State space model on innovation form (based on ARMAX representation)
%--------------------------------------------------------------------------%

A_inn=zeros(length(B_arx_m));
A_inn(1:length(A_arx_m)-1)=-A_arx_m(2:end)';
A_inn(1:end-1,2:end)=eye(length(B_arx_m)-1);

B_inn=B_arx_m';

K_inn=([C_arx_m(2:end) zeros(1,length(B_arx_m)-length(C_arx_m)+1)]')...
    +A_inn(:,1);

C_inn=[1 zeros(1,length(A_inn)-1)];

%--------------------------------------------------------------------------%
% Actual System (Differs from nominal if plant/model mismatch)
%--------------------------------------------------------------------------%

delay_range = [0.8 1 1.2];
y_test=zeros(2000,length(delay_range));
u_test=zeros(2000,length(delay_range));
j=1;

    for delay=delay_range

    G_num1 = 20;
    G_den1 = [160 44 1];
    G_tau1 = 50*delay;

    H_num1 = -5;
    H_den1 = [25 10 1];
    H_tau1 = 10;
```

```matlab

% Initialize cell vector required for ss computation function

num1=cell(1,2); den1=cell(1,2); tau1=zeros(1,2);
num1{1}=G_num1; num1{2}=H_num1;
den1{1}=G_den1; den1{2}=H_den1;
tau1(1)=G_tau1; tau1(2)=H_tau1;

% Conditions for computation function

Nmax=100; tol=1e-8;

[Ad1,Bd1,Cd1,Dd1,sH]=mimoctf2dss(num1,den1,tau1,Ts,Nmax,tol);

%--------------------------------------------------------------------------%
% Closed Loop Simulation
%--------------------------------------------------------------------------%

A=A_inn; B=B_inn; Cz=C_inn; G=K_inn;

nx = size(A,1);          % number of states in controller model
nu = size(B,2);          % number of inputs
nz = size(Cz,1);         % number of outputs

% Initialization

N=2000;

% Initializing constraints

umin = [-99999]';   umax = [99999]';
dumin = [-99999]';  dumax = [99999]';
zmin = [-99999]';   zmax = [99999]';


    for nbs=nbs_range     % prediction horizon

    % weight on deviation on the trajectory
    % qz is penalizing norm(z_k - r_k)
    % S is included as a weight in the regularization term

    qz = ones(1,length(Cz(:,1)));

        S_range=[5e3];

        for S_pen=S_range
        S  = S_pen*ones(1,length(B(1,:)));

        Rvec=zeros(nbs,N);
        Ref=[50*ones(1,150) 50*ones(1,N-150)];
        Rvec(:,:)=repmat(Ref,nbs,1);
        %Rvec(:,150-nbs+1:150)=fliplr(tril(Rvec(:,151:150+nbs)));

        x=zeros(length(Ad1),N+1);    u=zeros(nu,N);
        y=zeros(nz,N+1);             d=zeros(1,N);
```

```
                                        d(1,200:N)=10*ones(1,length(200:N));

        % Process noise

        sig_w=1;%0.5;
        sig_v=1;%0.2;

        randn('state',200);
        w=sig_w*randn(1,N);
        randn('state',500);
        v=sig_v*randn(1,N);

        test=zeros(nbs,N);

        yh_pr=zeros(nz,N);xh_pr=zeros(nx,N);e=zeros(nz,N);

        [H,Gamma,Phi,Phi_w,Mx0,Mum1,MR,Mw,Lambda] = ...
            MPCdesignMatrix_inn(A,B,G,Cz,qz,S,nbs);

        MPC_matrix=cell(9);
        constr=cell(3,2);

        MPC_matrix{1} = H;        MPC_matrix{2} = Gamma;
        MPC_matrix{3} = Phi;      MPC_matrix{4} = Mx0;
        MPC_matrix{5} = Mum1;     MPC_matrix{6} = MR;
        MPC_matrix{7} = Mw;       MPC_matrix{8} = Lambda;
        MPC_matrix{9} = Phi_w;

        constr{1,1} = umin;         constr{1,2} = umax;
        constr{2,1} = dumin;        constr{2,2} = dumax;
        constr{3,1} = zmin;         constr{3,2} = zmax;

            for i=2:N

                y(:,i)=Cd1*x(:,i)+v(:,i); % Actual System

            um1=u(:,i-1);

            [yh,xh,err,U0,uNew] = MPC_closed_loop(A,B,G,Cz,...
                xh_pr(:,i-1),um1,y(:,i),Rvec(:,i),qz,S,nbs,nu,...
                MPC_matrix,constr);

            test(:,i)=(Gamma*U0);

            xh_pr(:,i)=xh; yh_pr(:,i)=yh; e(:,i)=err; u(:,i)=uNew;

                x(:,i+1)=Ad1*x(:,i)+Bd1(:,1)*u(:,i)+Bd1(:,2)*(w(:,i)...
                 + d(:,i)); % Actual system update

            end


        end
```

```

        end

y_test(:,j)=y(1:N)'+300;
u_test(:,j)=u(1:N)'+40;
j=j+1;


    end



end

p_title=sprintf('Alpha = %g, S = %g',alpha,S_range);
figure('name',p_title),
    subplot(211);plot(Ts:Ts:750*Ts,[y_test(1:750,:)...
        350*ones(1,750)'] ,'LineWidth',3); legend('0.8Td','Td','1.2Td','Ref');
    grid;
    axis([0 1500 275 375])
    set(gca,'FontSize',20);
    title('Plant/Model Deadtime mismatch','FontSize',20);
    ylabel('measured output','FontSize',20);
    subplot(212);plot(Ts:Ts:750*Ts,u_test(1:750,:),'LineWidth',3);
    legend('0.8Td','Td','1.2Td');
    set(gca,'FontSize',20);
    ylabel('control signal','FontSize',20);
    grid;
    axis([0 1500 35 50])
    xlabel('Time [min]','FontSize',20);
```

E.1.2 Wood-Berry Distillation Column

```
clear all;
%close all;
clc;

Ts=1;

Tf=200;
nbs=400;


xs=[0.9632 0.9331 log10(87.34) log10(57.84) log10(4.87e4) log10(6.88e4)];
%xs=[0.9582 0.9216 1.7948 1.4372 1.9685 2.1198];
alpha=[xs(1) xs(2)];
qz=diag([10^(xs(3)) 10^(xs(4))]);
lambda1=10^xs(5); lambda2=10^xs(6);
S=diag([lambda1 lambda2]);

%R = [20;20]
R=[0.75;0];
```

```
%R=[1;1]

addpath MPC_tuning
addpath MPC_tuning/MPC_dir
addpath MPC_tuning/Realization

%---------------- Continuos Time transfer functions (Gyu) ----------------%

num1 = 12.8;        %Y1_U1
den1 = [16.7 1];
tau1  = 1;

num2 = -18.9;       %Y1_U2
den2 = [21 1];
tau2 = 3;

num3 = 6.6;         %Y2_U1
den3 = [10.9 1];
tau3  = 7;

num4 = -19.4;       %Y2_U2
den4 = [14.4 1];
tau4 = 3;

% Continuous time disturbance transfers (Gyd)

dnum1 = 3.8;        %Y1_D1
dden1 = [14.9 1];
dtau1 = 8.1;

dnum2 = 4.9;
dden2 = [13.2 1];
dtau2 = 3.4;

% Initialize cell vector required for ss computation function

num=cell(2,4); den=cell(2,4); tau=zeros(2,4);
num{1,1}=num1; num{1,2}=num2; num{1,3}=dnum1; num{1,4}=dnum1;
num{2,1}=num3; num{2,2}=num4; num{2,3}=dnum2; num{2,4}=dnum2;
den{1,1}=den1; den{1,2}=den2; den{1,3}=dden1; den{1,4}=dden1;
den{2,1}=den3; den{2,2}=den4; den{2,3}=dden2; den{2,4}=dden2;

% Conditions for computation function

Nmax=100; tol=1e-8;

[Ad,Bd,Cd,Dd,sH]=mimoctf2dss(num,den,tau,Ts,Nmax,tol);

tau(1,1)=tau1; tau(1,2)=tau2; tau(1,3)=dtau1; tau(1,4)=dtau1;
tau(2,1)=tau3; tau(2,2)=tau4; tau(2,3)=dtau2; tau(2,4)=dtau2;

%---------------------------- ARIMAX Model ----------------------------%

m=2;
p=2;
```

```
z=tf('z');

Gzu=cell(m,p); A_arx=cell(m,1); B_arx=cell(m,p);
A_armax=cell(m,1); B_armax=cell(m,p); C_armax=cell(m,1);
A_arx1=[];

Ac=cell(m,m); Bc=cell(m,p); Cc=cell(m); Kc=cell(m,m);

for mx=1:m % Generate discrete transferfunction description
    for px=1:p

    [h,th]=sisodss2dimpulse(Ad,Bd(:,px),Cd(mx,:),0,0,100,Ts);

    if ceil(tau(mx,px)/Ts)==floor(tau(mx,px)/Ts)
        hxx=zeros(1,1,length(h)); hxx(:,:,1:end)=h;
    else
        c=ceil(tau(mx,px)/Ts)-(tau(mx,px)/Ts);
        hxx=zeros(1,1,length(h));
        hxx(:,:,1:end)=[c*h(1:end-1); 0]+[0; (1-c)*h(1:end-1)];
    end
    [a11,b11,c11,d11]=sisodimpulse2dss(hxx,1e-8);
    [a1,b1]=ss2tf(a11,b11,c11,d11,Ts);
    Gzu{mx,px}=tf(a1,b1,Ts)*z^(-floor(tau(mx,px)/Ts)+1);
    end
end

for mx=1:m % Compute ARMAX polynomials
    for px=1:p
        if px==1
            a_arx = conv(Gzu{mx,px}.den{1,1},Gzu{mx,px+1}.den{1,1});
        elseif px >= 3
            a_arx = conv(a_arx,Gzu{mx,px}.den{1,1});
        end
        if p > 2
            for pxx=1:p-1
                if pxx==~px
                    if pxx+1==px
                        t = conv(Gzu{mx,pxx}.den{1,1},Gzu{mx,pxx+2}.den{1,1});
                    else
                        t = conv(Gzu{mx,pxx}.den{1,1},Gzu{mx,pxx+1}.den{1,1});
                    end
                    if isempty(A_arx1)&&p>2
                        A_arx1=t;
                    else
                        A_arx1=conv(A_arx1,t);
                    end
                end
            end
        else
            if px==1
               A_arx1 = Gzu{mx,2}.den{1,1};
            else
               A_arx1 = Gzu{mx,1}.den{1,1};
            end
```

```
        end


        B_arx{mx,px} = conv(Gzu{mx,px}.num{1,1}, A_arx1);
        A_arx1=[]; % clear variable before next iteration
    end
    A_arx{mx}=a_arx;
    a_arx=[];

    A_armax{mx}=[A_arx{mx} 0] - [0 A_arx{mx}];   % [1 - q^-1] A_arx
    nn=0;

    for i=1:p
    B_armax{mx,i} = [B_arx{mx,i} 0] - [0 B_arx{mx,i}]; % [1 - q^-1] B_arx
        if length(B_armax{mx,i})>nn
            nn=length(B_armax{mx,i});
        end
    end
    C_armax{mx}=[1 -alpha(mx)]; % [1 -alpha*q^-1]

%------------------------- State Space Model -----------------------------%
A1=zeros(nn);
A1(1:length(A_armax{mx,1})-1)=-A_armax{mx,1}(2:end)';
A1(1:end-1,2:end)=eye(nn-1);
K1=([C_armax{mx,1}(2:end) zeros(1,nn-length(C_armax{mx,1})+1)]')...
    +A1(:,1);
C1=[1 zeros(1,length(A1)-1)];

Ac{mx,mx}=A1;
Cc{mx,mx}=C1;
Kc{mx,mx}=K1;

    for i=1:p
        Bc{mx,i}=B_armax{mx,i}';
    end

end
%---------------------- Augmented State Space Model ----------------------%
for i=1:m % rows
    for j=1:m % columns
            if not(i==j)
            Ac{i,j}=zeros(length(Ac{i,i}),length(Ac{j,j}));
            Ac{j,i}=zeros(length(Ac{j,j}),length(Ac{i,i}));
            Cc{i,j}=zeros(1,length(Cc{j,j}));
            Cc{j,i}=zeros(1,length(Cc{i,i}));
            Kc{i,j}=zeros(length(Kc{i,i}),1);
            Kc{j,i}=zeros(length(Kc{j,j}),1);
            end
    end
end
Ac = cell2mat(Ac);
Bc = cell2mat(Bc);
Kc = cell2mat(Kc);
Cc = cell2mat(Cc);
```

```
%------------------------------ minimum realization-------------------------------------%

tol=1e-8;

Nmax=100;

H = mimodss2dimpulse(Ac,[Bc Kc],Cc,zeros(2,4),Nmax);
[Ad1,Bd1,Cd1,Dd1,sH1]=mimodimpulse2dss(H,tol);

A=Ad1; B=Bd1(:,1:2); K=Bd1(:,3:end); C=Cd1;
%------------------------------ Model - Plant Mismatch ------------------------------%

[Ad,Bd,Cd,Dd,sH]=mimoctf2dss(num,den,tau,Ts,Nmax,tol);


deviation = [0.75 1];
cnt=0;

for dev=deviation

cnt=cnt+1;

num1 = 12.8;       %Y1_U1
den1 = [16.7*dev 1];
tau1  = 1;

num2 = -18.9;      %Y1_U2
den2 = [21*dev 1];
tau2 = 3;

num3 = 6.6;        %Y2_U1
den3 = [10.9*dev 1];
tau3  = 7;

num4 = -19.4;      %Y2_U2
den4 = [14.4*dev 1];
tau4 = 3;

% Continuous time disturbance transfers (Gyd)

dnum1 = 3.8;       %Y1_D1
dden1 = [14.9 1];
dtau1 = 8.1;

dnum2 = 4.9;
dden2 = [13.2 1];
dtau2 = 3.4;

% Initialize cell vector required for ss computation function

num=cell(2,4); den=cell(2,4); tau=zeros(2,4);
num{1,1}=num1; num{1,2}=num2; num{1,3}=dnum1; num{1,4}=dnum1;
num{2,1}=num3; num{2,2}=num4; num{2,3}=dnum2; num{2,4}=dnum2;
den{1,1}=den1; den{1,2}=den2; den{1,3}=dden1; den{1,4}=dden1;
den{2,1}=den3; den{2,2}=den4; den{2,3}=dden2; den{2,4}=dden2;
```

```
tau(1,1)=tau1; tau(1,2)=tau2; tau(1,3)=dtau1; tau(1,4)=dtau1;
tau(2,1)=tau3; tau(2,2)=tau4; tau(2,3)=dtau2; tau(2,4)=dtau2;

% Conditions for computation function

Nmax=100; tol=1e-8;

[Ad1,Bd1,Cd1,Dd1,sH]=mimoctf2dss(num,den,tau,Ts,Nmax,tol);

%----------------------------------------------------------------------%
% Closed Loop Simulation
%----------------------------------------------------------------------%


    nx = size(A,1);          % number of states in controller model
    nu = size(B,2);          % number of inputs
    nz = size(C,1);         % number of outputs

    % Initialization

    N=Tf/Ts;


        % weight on deviation on the trajectory
        % qz is penalizing norm(z_k - r_k)
        % S is included as a weight in the regularization term

        %qz = eye(2);%
        %qz = [1 0; 0 2];
        %qz = sqrt(qz);
        %qz=ones(1,length(Cz(:,1)));


            Rvec=zeros(nbs*nz,N);
            Ref=[R(1)*ones(1,N);
                 R(2)*ones(1,N)];

            Rvec(:,:)=repmat(Ref,nbs,1);

            x=zeros(length(Ad1),N+1);    u=zeros(nu,N+1);
            y=zeros(nz,N+1);             d=0*ones(1,N);
                                         d(:,100:N)=0.34+d(:,100:N);
                                         diff_u=zeros(2,N);

            % Process and measurement noise

            sig_w=0.01;
            sig_v=0.01;%sqrt(0.0005);

            randn('state',200);
            w=sqrt(sig_w)*randn(1,N);
            randn('state',500);
            v=sqrt(sig_v)*randn(nz,N);
            cov(v')
            cov(w')
```

```
            yh_pr=zeros(nz,N);xh_pr=zeros(nx,N);e=zeros(nz,N);

            [H,Gamma,Phi,Phi_w,Mx0,Mum1,MR,Mw,Lambda] = ...
                MPCdesignMatrix_inn(A,B,K,C,qz,S,nbs);

            MPC_matrix=cell(9);
            constr=cell(3,2);

            MPC_matrix{1} = H;      MPC_matrix{2} = Gamma;
            MPC_matrix{3} = Phi;    MPC_matrix{4} = Mx0;
            MPC_matrix{5} = Mum1;   MPC_matrix{6} = MR;
            MPC_matrix{7} = Mw;     MPC_matrix{8} = Lambda;
            MPC_matrix{9} = Phi_w;


                for i=2:N

                    y(:,i)=Cd1*x(:,i)+v(:,i); % Actual System

                um1=u(:,i-1);

                [yh,xh,err,uNew,diff_u(:,i-1)] = MPC_closed_loop_unconstrained(A,B,K,C,...
                    xh_pr(:,i-1),um1,y(:,i),Rvec(:,i),qz,S,nbs,nu,...
                    MPC_matrix);

                xh_pr(:,i)=xh; yh_pr(:,i)=yh; e(:,i)=err; u(:,i)=uNew;

                x(:,i+1)=Ad1*x(:,i)+Bd1(:,1:nu)*u(:,i)+Bd1(:,3)*(w(:,i)...
                     + d(:,i)); % Actual system update
                end

T=0:Ts:Tf;
figure,subplot(211),plot(T,y'); legend('y1','y2');
subplot(212),plot(T,u'); legend('u1','u2');

if cnt==1
y_test(1:2,:)=y+repmat([96.25;0.5],1,N+1);
u_test(1:2,:)=u+repmat([1.95;1.71],1,N+1);
end

if cnt==2
y_test(3:4,:)=y+repmat([96.25;0.5],1,N+1);
u_test(3:4,:)=u+repmat([1.95;1.71],1,N+1);
end

end

p_title=sprintf('Alpha = %g, S = %g',alpha,S);
figure('name',p_title),
    title('Plant/Model Time Constant Mismatch','FontSize',20);
    subplot(221);plot([y_test(1,1:end-1)' y_test(3,1:end-1)'],'LineWidth',3);
    legend('0.75T0','T0');
    grid;
    set(gca,'FontSize',20);
    ylabel('Top Methanol [mol%]','FontSize',20);
```

```
    subplot(222);plot([y_test(2,1:end-1)' y_test(4,1:end-1)'],'LineWidth',3);
    legend('0.75T0','T0');
    grid;
    set(gca,'FontSize',20);
    ylabel('Bottom Methanol [mol%]','FontSize',20);
    subplot(223);plot([u_test(1,1:end-1)' u_test(3,1:end-1)'],'LineWidth',3);
    legend('0.75T0','T0');
    grid;
    set(gca,'FontSize',20);
    ylabel('Reflux Flowrate','FontSize',20);
    subplot(224);plot([u_test(2,1:end-1)' u_test(4,1:end-1)'],'LineWidth',3);
    legend('0.75T0','T0');
    grid;
    set(gca,'FontSize',20);
    ylabel('Steam Flowrate','FontSize',20);

    cov(y_test(3:4,1:end-1)')
    cov(y_test(1:2,1:end-1)')
    disp('control signal variance');
    cov(u_test(3:4,1:end-1)')
    cov(u_test(1:2,1:end-1)')
```

E.1.3 Cement-Mill

```
clear all;
close all;
clc;

Ts=2;

Tf=2000;
nbs=400;

%xs=[0.9849 0.0000 1.1477 1.9618 4.0743 4.1404] % det MS=1.775
xs=[0.9923 0.8503 2.5825 2.8487 5.9959 5.6890]; % det MS=1.3


alpha=[xs(1) xs(2)];
q1=10^xs(3); q2=10^xs(4);
qz=diag([q1 q2]);
s1=10^xs(5); s2=10^xs(6);
S=diag([s1 s2]);

R = [4;0]
%R=[20;20];
%R=[0;0];

addpath MPC_tuning
addpath MPC_tuning/MPC_dir
addpath MPC_tuning/Realization

%---------------- Continuos Time transfer functions (Gyu) ----------------%
```

```
num1 = 0.62;                    % B1
den1 = conv([45 1],[8 1]);      % A1
tau1  = 5;

num2 = 0.29*[8 1];              % B2
den2 = conv([2 1],[38 1]);      % A2
tau2 = 1.5;

num3 = -15;                     % B3
den3 = [60 1];                  % A3
tau3  = 5;

num4 = 5;                       % B4
den4 = conv([14 1],[1 1]);      % A4
tau4 = 0.1;

% Continuous time disturbance transfers (Gyd)

dnum1 = -1;      %Y1_D1
dden1 = conv([32 1],[21 1]);
dtau1 = 3;

dnum2 = 60;
dden2 = conv([30 1],[20 1]);
dtau2 = 3.4;

% Initialize cell vector required for ss computation function

num=cell(2,4); den=cell(2,4); tau=zeros(2,4);
num{1,1}=num1; num{1,2}=num2; num{1,3}=dnum1; num{1,4}=dnum1;
num{2,1}=num3; num{2,2}=num4; num{2,3}=dnum2; num{2,4}=dnum2;
den{1,1}=den1; den{1,2}=den2; den{1,3}=dden1; den{1,4}=dden1;
den{2,1}=den3; den{2,2}=den4; den{2,3}=dden2; den{2,4}=dden2;

% Conditions for computation function

Nmax=100; tol=1e-8;

[Ad,Bd,Cd,Dd,sH]=mimoctf2dss(num,den,tau,Ts,Nmax,tol);

tau(1,1)=tau1; tau(1,2)=tau2; tau(1,3)=dtau1; tau(1,4)=dtau1;
tau(2,1)=tau3; tau(2,2)=tau4; tau(2,3)=dtau2; tau(2,4)=dtau2;


%---------------------------- ARIMAX Model -----------------------------------%

m=2;
p=2;

z=tf('z');

Gzu=cell(m,p); A_arx=cell(m,1); B_arx=cell(m,p);
A_armax=cell(m,1); B_armax=cell(m,p); C_armax=cell(m,1);
A_arx1=[];
```

```

Ac=cell(m,m); Bc=cell(m,p); Cc=cell(m); Kc=cell(m,m);

for mx=1:m % Generate discrete transferfunction description
    for px=1:p

    [h,th]=sisodss2dimpulse(Ad,Bd(:,px),Cd(mx,:),0,0,100,Ts);

    if ceil(tau(mx,px)/Ts)==floor(tau(mx,px)/Ts)
        hxx=zeros(1,1,length(h)); hxx(:,:,1:end)=h;
    else
        c=ceil(tau(mx,px)/Ts)-(tau(mx,px)/Ts);
        hxx=zeros(1,1,length(h));
        hxx(:,:,1:end)=[c*h(1:end-1); 0]+[0; (1-c)*h(1:end-1)];
    end
    [a11,b11,c11,d11]=sisodimpulse2dss(hxx,1e-8);
    [a1,b1]=ss2tf(a11,b11,c11,d11);
    Gzu{mx,px}=tf(a1,b1,Ts)*z^(-floor(tau(mx,px)/Ts)+1);
    end
end

for mx=1:m % Compute ARMAX polynomials
    for px=1:p
        if px==1
            a_arx = conv(Gzu{mx,px}.den{1,1},Gzu{mx,px+1}.den{1,1});
        elseif px >= 3
            a_arx = conv(a_arx,Gzu{mx,px}.den{1,1});
        end
        if p > 2
            for pxx=1:p-1
                if pxx==~px
                    if pxx+1==px
                        t = conv(Gzu{mx,pxx}.den{1,1},Gzu{mx,pxx+2}.den{1,1});
                    else
                        t = conv(Gzu{mx,pxx}.den{1,1},Gzu{mx,pxx+1}.den{1,1});
                    end
                    if isempty(A_arx1)&&p>2
                        A_arx1=t;
                    else
                        A_arx1=conv(A_arx1,t);
                    end
                end
            end
        else
            if px==1
                A_arx1 = Gzu{mx,2}.den{1,1};
            else
                A_arx1 = Gzu{mx,1}.den{1,1};
            end
        end


        B_arx{mx,px} = conv(Gzu{mx,px}.num{1,1}, A_arx1);
        A_arx1=[]; % clear variable before next iteration
    end
```

```
    A_arx{mx}=a_arx;
    a_arx=[];

    A_armax{mx}=[A_arx{mx} 0] - [0 A_arx{mx}];   % [1 - q^-1] A_arx
    nn=0;

    for i=1:p
    B_armax{mx,i} = [B_arx{mx,i} 0] - [0 B_arx{mx,i}]; % [1 - q^-1] B_arx
        if length(B_armax{mx,i})>nn
            nn=length(B_armax{mx,i});
        end
    end
    C_armax{mx}=[1 -alpha(mx)]; % [1 -alpha*q^-1]

%------------------------------ State Space Model ------------------------------%
A1=zeros(nn);
A1(1:length(A_armax{mx,1})-1)=-A_armax{mx,1}(2:end)';
A1(1:end-1,2:end)=eye(nn-1);
K1=([C_armax{mx,1}(2:end) zeros(1,nn-length(C_armax{mx,1})+1)]')...
    +A1(:,1);
C1=[1 zeros(1,length(A1)-1)];

Ac{mx,mx}=A1;
Cc{mx,mx}=C1;
Kc{mx,mx}=K1;

    for i=1:p
        Bc{mx,i}=B_armax{mx,i}';
    end

end
%------------------------ Augmented State Space Model ------------------------%
for i=1:m % rows
    for j=1:m % columns
            if not(i==j)
            Ac{i,j}=zeros(length(Ac{i,i}),length(Ac{j,j}));
            Ac{j,i}=zeros(length(Ac{j,j}),length(Ac{i,i}));
            Cc{i,j}=zeros(1,length(Cc{j,j}));
            Cc{j,i}=zeros(1,length(Cc{i,i}));
            Kc{i,j}=zeros(length(Kc{i,i}),1);
            Kc{j,i}=zeros(length(Kc{j,j}),1);
            end
    end
end
Ac = cell2mat(Ac);
Bc = cell2mat(Bc);
Kc = cell2mat(Kc);
Cc = cell2mat(Cc);

%------------------------------ minimum realization------------------------------%

tol=1e-8;

Nmax=100;

```

```
H = mimodss2dimpulse(Ac,[Bc Kc],Cc,zeros(2,4),Nmax);
[Ad1,Bd1,Cd1,Dd1,sH1]=mimodimpulse2dss(H,tol);

A=Ad1; B=Bd1(:,1:2); K=Bd1(:,3:end); C=Cd1;


%————————————————————— Model — Plant Mismatch —————————————————————%

deviation = [1.5 1];
cnt=0;

for dev=deviation

cnt=cnt+1;

num1 = 0.62                          % B1
den1 = conv([45 1],[8 1]);           % A1
tau1  = 5*dev;

num2 = 0.29*[8 1];                   % B2
den2 = conv([2 1],[38 1]);           % A2
tau2 = 1.5*dev;

num3 = -15;                          % B3
den3 = [60 1];                       % A3
tau3  = 5*dev;

num4 = 5;                            % B4
den4 = conv([14 1],[1 1]);           % A4
tau4 = 0.1*dev;

% Continuous time disturbance transfers (Gyd)

dnum1 = -1;       %Y1_D1
dden1 = conv([32 1],[21 1]);
dtau1 = 3;

dnum2 = 60;
dden2 = conv([30 1],[20 1]);
dtau2 = 3.4;

% Initialize cell vector required for ss computation function

num=cell(2,4); den=cell(2,4); tau=zeros(2,4);
num{1,1}=num1; num{1,2}=num2; num{1,3}=dnum1; num{1,4}=dnum1;
num{2,1}=num3; num{2,2}=num4; num{2,3}=dnum2; num{2,4}=dnum2;
den{1,1}=den1; den{1,2}=den2; den{1,3}=dden1; den{1,4}=dden1;
den{2,1}=den3; den{2,2}=den4; den{2,3}=dden2; den{2,4}=dden2;
tau(1,1)=tau1; tau(1,2)=tau2; tau(1,3)=dtau1; tau(1,4)=dtau1;
tau(2,1)=tau3; tau(2,2)=tau4; tau(2,3)=dtau2; tau(2,4)=dtau2;

% Conditions for computation function

Nmax=100; tol=1e-8;
```

```
[Ad1,Bd1,Cd1,Dd1,sH]=mimoctf2dss(num,den,tau,Ts,Nmax,tol);

figure,dstep(Ad1,Bd1(:,1:2),Cd1,Dd1(:,1:2));

%-------------------------------------------------------------------------%
% Closed Loop Simulation
%-------------------------------------------------------------------------%


    nx = size(A,1);         % number of states in controller model
    nu = size(B,2);         % number of inputs
    nz = size(C,1);        % number of outputs

    % Initialization

    N=Tf/Ts;


        % weight on deviation on the trajectory
        % qz is penalizing norm(z_k - r_k)
        % S is included as a weight in the regularization term

        %qz = eye(2);%
        %qz=ones(1,length(Cz(:,1)));


            Rvec=zeros(nbs*nz,N);
            Ref=[R(1)*ones(1,N);
                 R(2)*ones(1,N)];

            Rvec(:,:)=repmat(Ref,nbs,1);

            x=zeros(length(Ad1),N+1);    u=zeros(nu,N+1);
            y=zeros(nz,N+1);             d=zeros(1,N);
                                         d(:,500:N)=5*ones(1,length(500:N));
                                         diff_u=zeros(2,N);

            % Process and measurement noise

            sig_w=1;%0.5;
            sig_v=diag([0.1 100]);

            randn('state',200);
            w=sqrt(sig_w)*randn(1,N);
            randn('state',500);
            v=sqrt(sig_v)*randn(nz,N);

            yh_pr=zeros(nz,N);
            xh_pr=zeros(nx,N);
            e=zeros(nz,N);

            [H,Gamma,Phi,Phi_w,Mx0,Mum1,MR,Mw,Lambda] = ...
                MPCdesignMatrix_inn(A,B,K,C,qz,S,nbs);

            MPC_matrix=cell(9);
```

```
            constr=cell(3,2);

            MPC_matrix{1} = H;       MPC_matrix{2} = Gamma;
            MPC_matrix{3} = Phi;     MPC_matrix{4} = Mx0;
            MPC_matrix{5} = Mum1;    MPC_matrix{6} = MR;
            MPC_matrix{7} = Mw;      MPC_matrix{8} = Lambda;
            MPC_matrix{9} = Phi_w;

                for i=2:N

                    y(:,i)=Cd1*x(:,i)+v(:,i); % Actual System

                um1=u(:,i-1);

                [yh,xh,err,uNew,diff_u(:,i-1)] = MPC_closed_loop_unconstrained(A,B,K,C,..
                    xh_pr(:,i-1),um1,y(:,i),Rvec(:,i),qz,S,nbs,nu,...
                    MPC_matrix);

                xh_pr(:,i)=xh; yh_pr(:,i)=yh; e(:,i)=err; u(:,i)=uNew;

%                 x(:,i+1)=Ad1*x(:,i)+Bd1(:,1:nu)*u(:,i)+Bd1(:,nu+1:end)*(w(:,i)...
%                      + d(:,i)); % Actual system update
                  x(:,i+1)=Ad1*x(:,i)+Bd1(:,1:nu)*u(:,i)+Bd1(:,3)*(w(:,i)...
                       + d(:,i)); % Actual system update
                end

T=0:Ts:Tf;
figure,subplot(211),plot(T,y'); legend('y1','y2');
subplot(212),plot(T,u'); legend('u1','u2');

if cnt==1
y_test(1:2,:)=y+repmat([26;3100],1,N+1);
u_test(1:2,:)=u+repmat([128;70],1,N+1);
end

if cnt==2
y_test(3:4,:)=y+repmat([26;3100],1,N+1);
u_test(3:4,:)=u+repmat([128;70],1,N+1);
end

end

p_title=sprintf('Alpha = %g, S = %g',alpha,S);
figure('name',p_title),
    title('Plant/Model deadtime mismatch (MS=3.5)','FontSize',20);
    subplot(221);plot([y_test(1,1:end-1)' y_test(3,1:end-1)'],'LineWidth',3);
    legend('1.5Td','Td');
    grid;
    set(gca,'FontSize',20);
    ylabel('Elevator Load','FontSize',20);
    subplot(222);plot([y_test(2,1:end-1)' y_test(4,1:end-1)'],'LineWidth',3);
    legend('1.5Td','Td');
    grid;
    set(gca,'FontSize',20);
    ylabel('Finess','FontSize',20);
```

```
    subplot(223);plot([u_test(1,1:end-1)' u_test(3,1:end-1)'],'LineWidth',3);
    legend('1.5Td','Td');
    grid;
    set(gca,'FontSize',20);
    ylabel('Feed','FontSize',20);
    subplot(224);plot([u_test(2,1:end-1)' u_test(4,1:end-1)'],'LineWidth',3);
    legend('1.5Td','Td');
    grid;
    set(gca,'FontSize',20);
    ylabel('Seperator Speed','FontSize',20);

%     cov(y_test(3:4,1:end-1)')
%     cov(y_test(1:2,1:end-1)')
%     disp('control signal variance');
%     cov(u_test(3:4,1:end-1)')
%     cov(u_test(1:2,1:end-1)')
```

E.1.4 Common files

```
function [H,Gamma,Phi,Phi_G,Mx0,Mum1,MR,Mw,Lambda] = ...
            MPCdesignMatrix_inn(A,B,G,Cz,qz,S,nbs)
%   [H,Gamma,Mx0,Mum1,MR,Mw,Lambda]=MPCdesignMatrix(A,B,G,Cz,qz,S,nbs)
%
% Description:
%   Designing the matrices for a quadratic problem, based on the impulse
%   response coefficients.
%
% Input:
%               :
%
% Output:
%               :
%
% Author        :   Soeren Nymann Thomsen & Daniel Haugaard Olesen
% Created       :   03.11.2011
% Revised       :   15.12.2011
% ------------------------------------------------------------------------

%% Designing the impulse resonse matrices
% Based on Lection 7 slide 10
[Phi,Phi_G Gamma] = IMPRdesign(A,B,Cz,G,nbs,'cell');

%% Weight matrices
% Based on Lection 7 slide 10
[QZ]       = QZdesign(qz,nbs,Cz);     % Weighting the different proces parameter
[HS,Mum1] = HSdesign(S ,nbs, B);      % Weighting the different control signals

%% Designing QP matrices
% This contitute the data for the MPC stated as a QP
% g = Mx0*x0 + MR*R + Mu_1 *u_1 + MD*D
% if isvector(R)
%     nr=length(R);
```

```
%      Rmpc = repmat(R,1,nbs)
%      Rvec = Rmpc(:);
%      warning('SNT:MPCsntQP:TransformToMatrix','R has been tranformed into a matrix')
% end

H    = Gamma' * QZ * Gamma + HS;
Mx0  = Gamma' * QZ * Phi;
MR   = -Gamma'* QZ;
Mw   = Gamma' * QZ * Phi_G;

%Lambda = eye(nbs)+diag(ones(nbs-1,1),-1);
Lambda = eye(nbs)-diag(ones(nbs-1,1),-1);
nu      = size(B,2);
Lambda = kron(Lambda,eye(nu));


function [Phi, Phi_G, Gamma]=IMPRdesign(A,B,Cz,G,nbs,algorithm)
%
% x_{k+1} = A * x_k + B * u_k + E * d_k
% z_k      = C_z * x_k    % Proces output
% y_k      = C_y * x_k    % Messurement
%
% nbs :      Prediction horizon
% E   :      Disturbance propegation matrice
%            Input may be empty
% algorithm:Options 'mat' or 'cell'. 'mat' return the matrices, 'cell'
%            return a cell structure.
%

%test if C is a row vector

if nargin<6
    algorithm = 'cell';
end

% Initializing the size the matrices
nx = length(A);
nu = size(B,2);
nz = size(Cz,1);

switch algorithm
    case 'mat'
        % See slide page 19 - Design MPC (offline)
        % Based on hints from John
        Gamma   = zeros(nbs*nz,nbs*nu);
        Phi     = zeros(nbs*nz,nx);
        Phi_G   = zeros(nbs*nz,nx);
        T    = Cz;
        T_G  = Cz;
        ii1 = 1;
        ii2 = nz;
        for ii=1:nbs
            Gamma( ii1:ii2,1:nu) = T*B;   % T equals C*A^(k-1)
            T = T*A;
            Phi(ii1:ii2,1:nx)  = T;       % T equals C*A^(k)
```

```
      Phi_G(ii1:ii2,1:nx) = T_G*G; % equals C*A^(k-1)*G;
      ii1 = ii1+nz;
      ii2 = ii2+nz;
      T_G = T_G*A;
    end
    % ERROR in indexing
    for jj=1:nbs
      ii1=jj*nz+1;   % starting row
      jj1=jj*nu+1;   % starting column
      jj2=(jj+1)*nu; % ending column
      Gamma(ii1:end,jj1:jj2) =Gamma( 1:end-ii1+1,1:nu);
    end
    for jj=1:nbs
      ii1=jj*nz+1;   % starting row
    end

  case 'cell'
    % See slide page 19 - Design MPC (offline)

    Phi = cell(nbs, 1);
    Phi_G = cell(nbs, 1);
    Hiu = cell(nbs, nbs);
    Hid = cell(nbs, nbs);
    u_temp = zeros(nz,nu);
    for ii=1:nbs
      for jj=1:nbs
        Hiu{ii,jj}=u_temp;
      end
    end
    for ii=1:nbs
      Phi{ii,1} = Cz * A^ii;
      Phi_G{ii,1} = Cz * A^(ii-1)*G;
      Hiu{ii,1} = Cz * A^(ii-1) * B;
    end
    for jj=2:nbs
      for ii=jj:nbs
        Hiu{ii,jj}=Hiu{ii-jj+1,1};
      end
    end
    Phi    = cell2mat(Phi);
    Phi_G  = cell2mat(Phi_G);
    Gamma  = cell2mat(Hiu);

end

function [QZ] = QZdesign(qz,nbs,Cz)
% Weighting the diffenrent process parameters
%
% x_{k+1} = A * x_k + B * u_k + E * d_k
% z_k     = C_z * x_k   % Proces output
% y_k     = C_y * x_k   % Messurement
%
%
%  qz  :   May be a - Weighting matrice eg. qz=[8 0;0 4]
%                 - Weighting vector eg. qz=[8,4]
```

```
%   nbs :   Prediction horizon
%   Cz  :   Proces output (not required)

%%

if nargin>2
    nz = size(Cz,1);
    if length(qz)~=nz
        if isscalar(qz)
            warning('SNT:MPC_QZDesign:Process_output_mismatch',...
                ['"qz" does not fit the number of process variables ',...
                '(',num2str(nz),'). \n',...
                'A diagonal matrice of size ', num2str(nz), 'x', num2str(nz)',...
                ' has been constructed.'])
            qz=eye(nz)*qz;      % Generates a matrix if qz is a scalar and nz>1
        else
            [x,y]=size(qz);
            error('SNT:MPC_QZDesign:Process_output_mismatch',...
                ['"qz" has the size ',num2str(x),'x',num2str(y),...
                ' and does not fit the number of process variables ',...
                '(',num2str(nz),').'])
        end
    end
end

if isvector(qz)
    qz=diag(qz,0);      % Generates a matrix if qz is a vector (or a "scalar")
end

QZ=kron(eye(nbs),qz);


function [HS,Mum1] = HSdesign(hs,nbs,B)
% weighting the different control signals
% Note ref.: Lecture 7 slide 10
%
%
% x_{k+1} = A * x_k + B * u_k + E * d_k
% z_k     = C_z * x_k   % Proces output
% y_k     = C_y * x_k   % Messurement
%
%
%   hs  :   May be a - Weighting matrice eg. hs=[8 0;0 4]
%                    - Weighting vector eg. hs=[8,4]
%   nbs :   Prediction horizon
%   Cz    :   Control matrice (

%%
if nargin>2
    nb = size(B,2);
    if length(hs)~=nb
        if isscalar(hs)
            warning('SNT:MPC_HSDesign:Control_signal_mismatch',...
                ['"hs" does not fit the number of control signals ',...
                '(',num2str(nb),'). \n',...
```

```
                'A diagonal matrice of size ', num2str(nb), 'x', num2str(nb)',...
                ' has been constructed.'])
            hs=eye(nb)*hs;      % Generates a matrix if hs is a scalar and nb>1
        else
            [x,y]=size(hs);
            error('SNT:MPC_HSDesign:Control_signal_mismatch',...
                ['"hs" has the size ',num2str(x),'x',num2str(y),...
                ' and does not fit the number of control signals ',...
                '(',num2str(nb),').'])
        end
    end
end

if isvector(hs)
    hs=diag(hs,0);      % Generates a matrix if hs is a vector (or a "scalar")
end

% Designing the diagonal of HS
nbs1=nbs-1;
HS=diag(ones(nbs1,1)*2,0);
HS(nbs,nbs)=1;

% Designing the subdiagonal of HS
HS=HS-diag(ones(nbs1,1),-1)-diag(ones(nbs1,1),1);
HS=kron(HS,hs);

% Designing the Mu_{-1}
Mum1=zeros(nbs,1);
Mum1(1,1)=1;
Mum1=-kron(Mum1,hs);
```

```
function [yh_pr,xh_pr,e,u,diff_u] = ...
MPC_closed_loop_unconstrained(A,B,K,Cz,x0,um1,y,Rvec,qz,S,nbs,nu,MPC_matrix)

H = MPC_matrix{1};      Gamma = MPC_matrix{2};
Phi = MPC_matrix{3};    Mx0 = MPC_matrix{4};
Mum1 = MPC_matrix{5};   MR = MPC_matrix{6};
Mw = MPC_matrix{7};     Lambda = MPC_matrix{8};
Phi_w = MPC_matrix{9};

    yh_pr=Cz*x0;

    e=y-yh_pr;

%  g = Mx0*x0 + MR*Rvec + Mum1*um1 + Mw*e;

   [R,p]=chol(H);

    if (p>0)
        error('H not positive def');
    end

%   U0=-(R\(R'\g));

```

```
    Lx0=-(R\(R'\Mx0));
    LR=-(R\(R'\MR));
    Lum1=-(R\(R'\Mum1));
    Lw=-(R\(R'\Mw));

    U0=Lx0*x0 + LR*Rvec + Lum1*um1 + Lw*e;

I0=[eye(nu); zeros(nu*(nbs-1),nu)];

     uNew=I0'*Lx0*x0 + I0'*LR*Rvec + I0'*Lum1*um1 + I0'*Lw*e;

%    uNew=U0(1:nu);

Kx0=I0'*Lx0;
KR=I0'*LR;
Kr=KR*repmat(eye(2),nbs,1);
Kum1=I0'*Lum1;
Kw=I0'*Lw;

Ref=Rvec(1:2);

u = Kx0*x0 + Kr*Ref + Kum1*um1 + Kw*e;


diff_u=uNew-u;
%    u=uNew;

xh_pr=A*x0+B*u+K*e; % time update
```

E.2 Toolbox

E.2.1 SISO Toolbox

E.2.1.1 Initiation of SISO-Toolbox: Industrial Furnace

```
%------------------------------------------------------------------%
% using MPC_Tuning_Toolbox to evaluate tuning of industrial furnace
%
% Daniel Olesen s100094, DTU
%------------------------------------------------------------------%

clear all;
close all;
clc;

addpath MPC_tuning
addpath MPC_tuning/Realization

%------------------ TF model description of Furnace ---------------%
```

```

Ts=2;             % Sampling Time

G_num = 20;
G_den = [160 44 1];
G_tau = 50;

H_num = -5;
H_den = [25 10 1];
H_tau = 10;

%------------------------------ SS conversion ------------------------------%

num=cell(1,3); den=cell(1,3); tau=zeros(1,3);
num{1}=G_num; num{2}=H_num; num{3}=H_num;
den{1}=G_den; den{2}=H_den; den{3}=H_den;

Nmax=100; tol=1e-8; % Conditions for computation function

[Ad1,Bd1,Cd1,Dd1,sH1]=mimoctf2dss(num,den,tau,Ts,Nmax,tol);
tau(1)=G_tau; tau(2)=H_tau; tau(3)=H_tau;

A=Ad1; B=Bd1(:,1);
E=Bd1(:,2); G=Bd1(:,3);
Cy=Cd1; Cz=Cd1;

%--------------------------- Toolbox settings ------------------------------%

lmin=1e2;         % Minimum value of Lambda
lmax=1e6;         % Maximum value of Lambda
alpha_min=0;      % Minimum value of alpha
alpha_max=1;      % Maximum value of alpha

N=200;             % The number of parameterevaluations between lmin and lmax
                  % and alpha_min and alpha_max

nbs=150;          % Prediction Horizon

MS_max=1.775;


[MS_array MT_array J_array Jd_array Ruv_array Ryv_array Ruw_array Ryw_array]...
    =tuning_darx_mpc(A,B,Cy,Cz,E,G,tau,Ts,alpha_min, alpha_max,...
    lmin, lmax, N, nbs, MS_max);

Ruu_array=Ruv_array+Ruw_array;  % Total control signal variance
Ryy_array=Ryv_array+Ryw_array;  % Total output signal variance
```

E.2.1.2 Algorithm: SISO Toolbox

```
%--------------------------------------------------------------------------%
% Performance evaluation based on MS, J, Jd, Ruu and Ryy (SISO systems)
```

```
%
% Parameter sweep in alpha, lambda
%
% Daniel Olesen, DTU
%
% 2012-03-17 (v4.00)
%--------------------------------------------------------------------------%
%
% function [MS, MT, J, Jd, Rvu; Rvy, Rwu, Rwy] =
% tuning_darx_MPC(num, den, tau, Ts, alpha_min, alpha_max, lmin, lmax,...
% N, nbs, MS_max,Rww,Rvv)
%
%
%   alpha_min, alpha_max, lmin, lmax denotes the boundaries of
%   investigation
%
%   alpha_min => 0, alpha_max =< 1
%
%   Rvv is the measurement noise covariance
%   Rww is the process noise covariance
%   if no values are specified, Rvv=Rww=1
%
%   Evaluations which yield MS > MSMAX is cancelled out

function [MS_array MT_array J_array Jd_array Rvu_array Rvy_array,...
          Rwu_array Rwy_array]=tuning_darx_mpc(A,B,Cy,Cz,E,G,tau,Ts,...
          alpha_min, alpha_max, lmin, lmax, N, nbs,MS_max,Rww,Rvv,Neval)

if nargin < 16
    Rvv=1; Rww=1;
end

if nargin < 18
    Neval = nbs;
end


alpha_range=linspace(alpha_min,alpha_max,N);

MS_array=zeros(N,N);
MT_array=zeros(N,N);
J_array=zeros(N,N);
Jd_array=zeros(N,N);
Rvu_array=zeros(N,N);
Rvy_array=zeros(N,N);
Rwu_array=zeros(N,N);
Rwy_array=zeros(N,N);

cnt=1;
for alpha = alpha_range
    [MS MT J Jd Rvu Rvy Rwu Rwy] = tuning_lambda_2(A,B,Cy,Cz,E,G,tau...
        ,Ts, alpha,lmin,lmax,N,nbs,MS_max,Rww,Rvv,Neval);
    MS_array(cnt,:)=MS;
    MT_array(cnt,:)=MT;
    J_array(cnt,:)=J;
```

```
    Jd_array(cnt,:)=Jd;
    Rvu_array(cnt,:)=Rvu;
    Rvy_array(cnt,:)=Rvy;
    Rwu_array(cnt,:)=Rwu;
    Rwy_array(cnt,:)=Rwy;

    cnt=cnt+1
end

if nargin == 5
Rvu_array=Rvu_array+Rwu_array;
Rvy_array=Rvy_array+Rwy_array;
clear Rwy_array; clear Rwu_array;
end
```

```
%--------------------------------------------------------------------------%
% Performance evaluation based on MS, J, Jd, Ruu and Ryy (SISO systems)
%
% Parameter sweep in lambda
%
% Daniel Olesen, DTU
%
% 2012-05-18 (v5.00)
%--------------------------------------------------------------------------%
function [MS MT J Jd Rvu Rvy Rwu Rwy] = tuning_lambda_2(A,B,Cy,Cz,E,G,tau...
    ,Ts,alpha,lmin,lmax,N,nbs,MS_max,Rww,Rvv,Neval)

if nargin < 15
    Rvv=1; Rww=1;
end

if nargin < 17
    Neval = nbs;
end

addpath MPC_tuning\MPC_dir
addpath MPC_tuning\Realization

lmin=log10(lmin);
lmax=log10(lmax);

%--------------------- Delta-ARX Controller Model -------------------------%

z=tf('z');
[a,b]=ss2tf(A,B,Cy,0);
Gzu=minreal(tf(a,b,Ts))*z^(-ceil(tau(1)/Ts));

A_arx1 = cell2mat(Gzu.den);
B_arx1 = cell2mat(Gzu.num); B_arx1 = B_arx1(2:end);

A_arx_m1 = [A_arx1 0] - [0 A_arx1]; % [1 - q^-1] A_arx
B_arx_m1 = [B_arx1 0] - [0 B_arx1]; % [1 - q^-1] B_arx
C_arx_m1 = [1 -alpha];              % [1 -alpha*q^-1]

```

```
%--------------- Conversion to State Space (innovation form ------------------%

A1=zeros(length(B_arx_m1));
A1(1:length(A_arx_m1)-1)=-A_arx_m1(2:end)';
A1(1:end-1,2:end)=eye(length(B_arx_m1)-1);
B1=B_arx_m1';
K1=([C_arx_m1(2:end) zeros(1,length(B_arx_m1)-length(C_arx_m1)+1)]')...
    +A1(:,1);
C1=[1 zeros(1,length(A1)-1)];

%--------------------------- minimum realization---------------------------------%

tol=1e-8;

Nmax=100;

H = mimodss2dimpulse(A1,[B1 K1],C1,zeros(1,2),Nmax);
[Ad,Bd,Cd,Dd,sH]=mimodimpulse2dss(H,tol);

A1a=Ad; B1a=Bd(:,1); K1a=Bd(:,2); C1a=Cd;

%--------------------------- add delays to ss-model ------------------------------%

m=size(Cy,1); p=size([B E G],2);
H = mimodss2dimpulse(A,[B E G],Cy,zeros(m,p),Nmax);
H1=zeros(m,p,size(H,3)+max(tau)/Ts);
for mx=1:m
    for px=1:p
    H1(mx,px,1:(size(H,3)+(tau(mx,px)/Ts)))=...
    cat(3,zeros(1,1,(tau(mx,px)/Ts)),H(mx,px,1:end));
    end
end

[Ad2,Bd2,Cd2,Dd2,sH2]=mimodimpulse2dss(H1,tol);
A=Ad2;
B=Bd2(:,1);
E=Bd2(:,2);
G=Bd2(:,3);
Cy=Cd2;
Cz=Cy;


%------------------------------------------ MPC ----------------------------------%
lambda_range=logspace(lmin,lmax,N);

MS=-ones(1,N);
MT=-ones(1,N);
J =-ones(1,N);
Jd =-ones(1,N);
Rwy = -ones(1,N); % Process noise to output
Rwu = -ones(1,N); % Process noise to control signal
Rvy = -ones(1,N); % Measurement noise to output
Rvu = -ones(1,N); % Measurement nosie to control signal
Ru = -ones(1,N);
Ry = -ones(1,N);
```

```

T = 0:Ts:Neval*Ts;
w=logspace(-4, log10(pi/Ts),1000); % Decide number of points (default 1000)
X1 = zeros(length(T),1);
X1u = zeros(length(T),1);
X2 = zeros(length(T),1);
X2u = zeros(length(T),1);
Swz1 = zeros(length(w),1);

i=N;

for lambda=fliplr(lambda_range)


    [Ac1 Bcy1 Bcr1 Cc1 Dcy1 Dcr1] = ss_MPC(A1a,B1a,C1a,K1a,zeros(length(A1a),1),...
    1,1,lambda,nbs);


[Acl1 Bwcl1 Bvcl1 Brcl1 Bdcl1 Czcl1 Cycl1 Cucl1] = ss_closed_loop(A,...
    B,Cy,Cz,E,G,Ac1,Bcy1,Bcr1,Cc1,Dcy1,Dcr1);

[X1,X1u] = dstep_rsp(Acl1,Brcl1,Czcl1,Cucl1,0,Dcr1,T);
[X2,X2u] = dstep_rsp(Acl1,Bdcl1,Czcl1,Cucl1,0,0,T);


[Sz1a,Sz1b]=ss2tf(Acl1,Bvcl1,Cycl1,1);
Swz1=frqrsp_dtf(Sz1a,Sz1b,w,Ts);
MS(i)=max(abs(Swz1));
[Tz1a,Tz1b]=ss2tf(Acl1,Brcl1,Cycl1,0);
Twz1=frqrsp_dtf(Tz1a,Tz1b,w,Ts);

MT(i)=min(real(Twz1));

if MS(i) > MS_max
    MS(i)=-1;
    break

end

J(i)=sum(abs(ones(size(X1))-X1));
Jd(i)=sum(abs(X2));

% Processnoise propogation

Rxx=dlyap(Acl1,Bwcl1*Rww*Bwcl1');
Ryy=Cycl1*Rxx*Cycl1';

Rwy(i)=Ryy;

Ryy=Cucl1*Rxx*Cucl1';

Rwu(i)=Ryy;

% measurement noise propogation

```

```
Rxx=dlyap(Acl1,Bvcl1*Rvv*Bvcl1');
Ryy=Cycl1*Rxx*Cycl1'+Rvv;

Rvy(i)=Ryy;

Ryy=Cucl1*Rxx*Cucl1'+Dcy1*Rvv*Dcy1';

Rvu(i)=Ryy;

Ru(i)=Rvu(i)+Rwu(i);
Ry(i)=Rvy(i)+Rwy(i);

i=i-1;
end

if nargout == 6
    clear Rvu; clear Rvy; clear Rwy; clear Rwu;
    Rvu=Ru;
    Rvy=Ry;
end
```

E.2.1.3 Data analysis/Visual Presentation

```
%-----------------------------------------------------------------------%
% Evaluation of minimum performance measures for a constant value of MS
%
% Daniel Olesen s100094, DTU
%-----------------------------------------------------------------------%

clc;

Ryy_array=Ryv_array+Ryw_array;
Ruu_array=Ruv_array+Ruw_array;

MS_max=1.775;

alpha_range=linspace(alpha_min,alpha_max,N);

idx=find((MS_array>0.995*MS_max)&(MS_array<1.005*MS_max));

[min_Ryw i]=min(Ryw_array(idx));

col=floor(idx(i)/N);
if ~((idx(i)/N)==floor(idx(i)/N))
    col = col + 1;
end
row=idx(i)-(col-1)*N;

alpha=alpha_range(row);
lambda=x(col);

xx = sprintf('Minimum Ryw = %g, alpha = %g, lambda = %g',...
```

```
    Ryw_array(row,col),alpha,lambda);
disp(xx);

disp(sprintf('Ryv = %g, Ruv = %g, Ruw = %g, Ryy = %g, Ruu = %g, J = %g,Jd = %g',...
Ryv_array(row,col),Ruv_array(row,col), Ruw_array(row,col), Ryy_array(row,col),...
Ruu_array(row,col), J_array(row,col), Jd_array(row,col)));

disp('-');

[min_Ryv i]=min(Ryv_array(idx));

col=floor(idx(i)/N);
if ~((idx(i)/N)==floor(idx(i)/N))
    col = col + 1;
end
row=idx(i)-(col-1)*N;

alpha=alpha_range(row);
lambda=x(col);

xx = sprintf('Minimum Ryv = %g, alpha = %g, lambda = %g',...
    Ryv_array(row,col),alpha,lambda);
disp(xx);

disp(sprintf('Ryw = %g, Ruv = %g, Ruw = %g, Ryy = %g, Ruu = %g, J = %g,Jd = %g',...
Ryw_array(row,col),Ruv_array(row,col), Ruw_array(row,col), Ryy_array(row,col),...
Ruu_array(row,col), J_array(row,col), Jd_array(row,col)));

disp('-');

[min_Ruw i]=min(Ruw_array(idx));

col=floor(idx(i)/N);
if ~((idx(i)/N)==floor(idx(i)/N))
    col = col + 1;
end
row=idx(i)-(col-1)*N;

alpha=alpha_range(row);
lambda=x(col);

xx = sprintf('Minimum Ruw = %g, alpha = %g, lambda = %g',...
    Ruw_array(row,col),alpha,lambda);
disp(xx);

disp(sprintf('Ryw = %g, Ruv = %g, Ryv = %g, Ryy = %g, Ruu = %g, J = %g,Jd = %g',...
Ryw_array(row,col),Ruv_array(row,col), Ryv_array(row,col), Ryy_array(row,col),...
Ruu_array(row,col), J_array(row,col), Jd_array(row,col)));

disp('-');


[min_Ruw i]=min(Ruv_array(idx));

col=floor(idx(i)/N);
```

```
if ~((idx(i)/N)==floor(idx(i)/N))
    col = col + 1;
end
row=idx(i)-(col-1)*N;

alpha=alpha_range(row);
lambda=x(col);

xx = sprintf('Minimum Ruv = %g, alpha = %g, lambda = %g',...
    Ruv_array(row,col),alpha,lambda);
disp(xx);

disp(sprintf('Ryw = %g, Ryv = %g, Ruw = %g, Ryy = %g, Ruu = %g, J = %g,Jd = %g',...
Ryw_array(row,col),Ryv_array(row,col), Ruw_array(row,col), Ryy_array(row,col),...
Ruu_array(row,col), J_array(row,col), Jd_array(row,col)));

disp('-');

[min_Jd i]=min(Jd_array(idx));

col=floor(idx(i)/N);
if ~((idx(i)/N)==floor(idx(i)/N))
    col = col + 1;
end
row=idx(i)-(col-1)*N;

alpha=alpha_range(row);
lambda=x(col);

xx = sprintf('Minimum Jd = %g, alpha = %g, lambda = %g',...
    Jd_array(row,col),alpha,lambda);
disp(xx);

disp(sprintf('Ryw = %g, Ryv = %g, Ruv = %g, Ruw = %g, Ryy = %g, Ruu = %g, J = %g',...
Ryw_array(row,col),Ryv_array(row,col),Ruv_array(row,col), Ruw_array(row,col),...
Ryy_array(row,col), Ruu_array(row,col), J_array(row,col)));

disp('-');

[min_J i]=min(J_array(idx));

col=floor(idx(i)/N);
if ~((idx(i)/N)==floor(idx(i)/N))
    col = col + 1;
end
row=idx(i)-(col-1)*N;

alpha=alpha_range(row);
lambda=x(col);

xx = sprintf('Minimum J = %g, alpha = %g, lambda = %g',...
    J_array(row,col),alpha,lambda);
disp(xx);

disp(sprintf('Ryw = %g, Ryv = %g, Ruv = %g, Ruw = %g, Ryy = %g, Ruu = %g, Jd = %g',...
```

```
Ryw_array(row,col),Ryv_array(row,col),Ruv_array(row,col), Ruw_array(row,col),...
Ryy_array(row,col), Ruu_array(row,col), Jd_array(row,col)));

disp('-');

[min_Ryy i]=min(Ryy_array(idx));

col=floor(idx(i)/N);
if ~((idx(i)/N)==floor(idx(i)/N))
    col = col + 1;
end

row=idx(i)-(col-1)*N;

alpha=alpha_range(row);
lambda=x(col);

xx = sprintf('Minimum Ryy = %g, alpha = %g, lambda = %g',...
    Ryy_array(row,col),alpha,lambda);
disp(xx);

disp(sprintf('Ryw = %g, Ryv = %g, Ruv = %g, Ruw = %g, Ruu = %g, Jd = %g, J =%g',...
Ryw_array(row,col),Ryv_array(row,col),Ruv_array(row,col), Ruw_array(row,col),...
Ruu_array(row,col), Jd_array(row,col),J_array(row,col)));

disp('-');


[min_Ruu i]=min(Ruu_array(idx));

col=floor(idx(i)/N);
if ~((idx(i)/N)==floor(idx(i)/N))
    col = col + 1;
end

row=idx(i)-(col-1)*N;

alpha=alpha_range(row);
lambda=x(col);

xx = sprintf('Minimum Ruu = %g, alpha = %g, lambda = %g',...
    Ruu_array(row,col),alpha,lambda);
disp(xx);

disp(sprintf('Ryw = %g, Ryv = %g, Ruv = %g, Ruw = %g, Ryy = %g, Jd = %g, J =%g',...
Ryw_array(row,col),Ryv_array(row,col),Ruv_array(row,col), Ruw_array(row,col),...
Ryy_array(row,col), Jd_array(row,col),J_array(row,col)));

disp('-');
```

```
%--------------------------------------------------------------------------%
% Create colour contour plots from SISO-Toolbox
%
% Daniel Olesen s100094, DTU
```

```
%------------------------------------------------------------------------%


load('MyColormaps','mycmap');

Ruu_array=Ruv_array+Ruw_array;
Ryy_array=Ryv_array+Ryw_array;

idx=find(MS_array>0);                   % Exclude result not satisfying
                                        % MS < MS_max


x=logspace(log10(lmin),log10(lmax),N);
y=linspace(alpha_min,alpha_max,N);
idx=find(MS_array>0);
max_MS=max(MS_array(idx));
min_MS=min(MS_array(idx));
min_MS=min_MS-2*((max_MS-min_MS)/64);
figure, subplot(221),imagesc(log10(x),y,MS_array,[min_MS max_MS]);
axis xy;
ylabel('\alpha','FontSize',20),xlabel('log10(\lambda)','FontSize',20);

title('MS','FontSize',20);
set(gca,'FontSize',20);
set(gcf,'Colormap',mycmap);
colorbar;

max_Jd=max(Jd_array(idx));
min_Jd=min(Jd_array(idx));
min_Jd=min_Jd-2*((max_Jd-min_Jd)/64);

subplot(222), imagesc(log10(x),y,Jd_array,[min_Jd max_Jd]);
axis xy;
ylabel('\alpha','FontSize',20),xlabel('log10(\lambda)','FontSize',20);
set(gca,'FontSize',20);
title('Jd','FontSize',20);
set(gcf,'Colormap',mycmap);
colorbar;

max_Ryy=max(Ryy_array(idx));
min_Ryy=min(Ryy_array(idx));
min_Ryy=min_Ryy-2*((max_Ryy-min_Ryy)/64);


subplot(223), imagesc(log10(x),y,Ryy_array,[min_Ryy max_Ryy]);
axis xy;
ylabel('\alpha','FontSize',20),xlabel('log10(\lambda)','FontSize',20);
title('Ryy','FontSize',20);
set(gca,'FontSize',20);
set(gcf,'Colormap',mycmap);
colorbar;

max_Ruu=max(Ruu_array(idx));
min_Ruu=min(Ruu_array(idx));
min_Ruu=min_Ruu-2*((max_Ruu-min_Ruu)/64);
```

```
subplot(224), imagesc(log10(x),y,Ruu_array,[min_Ruu max_Ruu]);
axis xy;
ylabel('\alpha','FontSize',20),xlabel('log10(\lambda)','FontSize',20);
title('Ruu','FontSize',20);
set(gca,'FontSize',20);
set(gcf,'Colormap',mycmap);
colorbar;

max_Ryv=max(Ryv_array(idx));
max_Ryv=5;
min_Ryv=min(Ryv_array(idx));
min_Ryv=min_Ryv-2*((max_Ryv-min_Ryv)/64);

figure,subplot(221), imagesc(log10(x),y,Ryv_array,[min_Ryv max_Ryv]);
axis xy;
ylabel('\alpha','FontSize',20),xlabel('log10(\lambda)','FontSize',20);
title('Ryv','FontSize',20);
set(gca,'FontSize',20);
set(gcf,'Colormap',mycmap);
colorbar;


max_Ryw=max(Ryw_array(idx));
min_Ryw=min(Ryw_array(idx));
min_Ryw=min_Ryw-2*((max_Ryw-min_Ryw)/64);

subplot(223), imagesc(log10(x),y,Ryw_array,[min_Ryw max_Ryw]);
axis xy;
ylabel('\alpha','FontSize',20),xlabel('log10(\lambda)','FontSize',20);
title('Ryw','FontSize',20);
set(gca,'FontSize',20);
set(gcf,'Colormap',mycmap);
colorbar;



max_Ruv=max(Ruv_array(idx));
min_Ruv=min(Ruv_array(idx));
min_Ruv=min_Ruv-2*((max_Ruv-min_Ruv)/64);

subplot(222), imagesc(log10(x),y,Ruv_array,[min_Ruv max_Ruv]);
axis xy;
ylabel('\alpha','FontSize',20),xlabel('log10(\lambda)','FontSize',20);
title('Ruv','FontSize',20);
set(gca,'FontSize',20);
set(gcf,'Colormap',mycmap);
colorbar;


max_Ruw=max(Ruw_array(idx));
min_Ruw=min(Ruw_array(idx));
min_Ruw=min_Ruw-2*((max_Ruw-min_Ruw)/64);

subplot(224), imagesc(log10(x),y,Ruw_array,[min_Ruw max_Ruw]);
```

```
axis xy;
ylabel('\alpha','FontSize',20),xlabel('log10(\lambda)','FontSize',20);
title('Ruw','FontSize',20);
set(gca,'FontSize',20);
set(gcf,'Colormap',mycmap);
colorbar;
```

E.2.1.4 Initiation of SISO optimization procedure

```
clear all;
close all;
clc;

addpath MPC_tuning
addpath MPC_tuning\Realization


%-------------------- TF model description of Furnace -------------------------%

Ts=2;           % Sampling Time

G_num = 20;
G_den = [160 44 1];
G_tau = 50;

H_num = -5;
H_den = [25 10 1];
H_tau = 10;

%------------------------------ SS conversion -------------------------------%

num=cell(1,3); den=cell(1,3); tau=zeros(1,3);
num{1}=G_num; num{2}=H_num; num{3}=H_num;
den{1}=G_den; den{2}=H_den; den{3}=H_den;

Nmax=100; tol=1e-8; % Conditions for computation function

[Ad1,Bd1,Cd1,Dd1,sH1]=mimoctf2dss(num,den,tau,Ts,Nmax,tol);
tau(1)=G_tau; tau(2)=H_tau; tau(3)=H_tau;

A=Ad1; B=Bd1(:,1);
E=Bd1(:,2); G=Bd1(:,3);
Cy=Cd1; Cz=Cd1;

Rvv=1;
Rww=1;

x0=[1 2];
x1=[1 6];
x2=[0 2];
x3=[0 6]

```

```
for xt=[x0' x1' x2' x3']

[MS J Jd Rvu Rvy Rwu Rwy x]=optim_tuning_siso(A,B,Cy,Cz,E,G,tau,Ts,0, 1,...
    1e2, 1e6,150,1.775,Rww,Rvv,'interior-point',6,xt')

end

for xt=[x0' x1' x2' x3']

[MS J Jd Rvu Rvy Rwu Rwy x]=optim_tuning_siso(A,B,Cy,Cz,E,G,tau,Ts,0, 1,...
    1e2, 1e6,150,1.775,Rww,Rvv,'active-set',6,xt')

end


for xt=[x0' x1' x2' x3']

[MS J Jd Rvu Rvy Rwu Rwy x]=optim_tuning_siso(A,B,Cy,Cz,E,G,tau,Ts,0, 1,...
     1e2, 1e6,150,1.775,Rww,Rvv,'sqp',6,xt')

end
```

E.2.1.5 Algorithm: SISO Optimization procedure

```
%---------------------------------------------------------------------------%
% Constrained Optimization of tuning (MIMO)
%
% Daniel Olesen, DTU
%
% 2012-06-20 (v3.00)
%---------------------------------------------------------------------------%

function [MS J Jd Rvu Rvy Rwu Rwy x]...
    =optim_tuning_siso(A,B,Cy,Cz,E,G,tau,Ts,alpha_min,alpha_max,...
    lmin, lmax, nbs, MS_max, Rww, Rvv, solver, obj, x0)

lb=[alpha_min log10(lmin)];
ub=[alpha_max log10(lmax)];

disp(solver);
options = optimset('Algorithm',solver,'tolFun',1e-8,'TolX',1e-10,...
    'display','iter','MaxFunEvals',600);

x = fmincon(@(x)eval(A,B,Cy,Cz,E,G,tau,Ts,Rww,Rvv,x,nbs,obj),...
    x0,[],[],[],[],lb,ub,@(x)eval_MS(A,B,Cy,Cz,E,G,tau,Ts,x,nbs,MS_max),options);

[MS J Jd Rvu Rvy Rwu Rwy]=evaluation(A,B,Cy,Cz,E,G,tau,Ts,Rww,Rvv,x,nbs);

function [zz] = eval(A,B,Cy,Cz,E,G,tau,Ts,Rww,Rvv,x,nbs,obj)

alpha=x(1);
```

```
lambda=diag(10.^x(2));
%-------------------------- Delta-ARX Controller Model ------------------------------%
z=tf('z');
[a,b]=ss2tf(A,B,Cy,0);
Gzu=minreal(tf(a,b,Ts))*z^(-ceil(tau(1)/Ts));
A_arx1 = cell2mat(Gzu.den);
B_arx1 = cell2mat(Gzu.num); B_arx1 = B_arx1(2:end);
A_arx_m1 = [A_arx1 0] - [0 A_arx1]; % [1 - q^-1] A_arx
B_arx_m1 = [B_arx1 0] - [0 B_arx1]; % [1 - q^-1] B_arx
C_arx_m1 = [1 -alpha];              % [1 -alpha*q^-1]
%---------------- Conversion to State Space (innovation form --------------------%
A1=zeros(length(B_arx_m1));
A1(1:length(A_arx_m1)-1)=-A_arx_m1(2:end)';
A1(1:end-1,2:end)=eye(length(B_arx_m1)-1);
B1=B_arx_m1';
K1=([C_arx_m1(2:end) zeros(1,length(B_arx_m1)-length(C_arx_m1)+1)]')...
    +A1(:,1);
C1=[1 zeros(1,length(A1)-1)];
%--------------------------- minimum realization-----------------------------------%
tol=1e-8;
Nmax=100;
H = mimodss2dimpulse(A1,[B1 K1],C1,zeros(1,2),Nmax);
[Ad,Bd,Cd,Dd,sH]=mimodimpulse2dss(H,tol);
A1a=Ad; B1a=Bd(:,1); K1a=Bd(:,2); C1a=Cd;
%--------------------------- add delays to ss-model -------------------------------%
m=size(Cy,1); p=size([B E G],2);
H = mimodss2dimpulse(A,[B E G],Cy,zeros(m,p),Nmax);
H1=zeros(m,p,size(H,3)+max(tau)/Ts);
for mx=1:m
    for px=1:p
    H1(mx,px,1:(size(H,3)+(tau(mx,px)/Ts)))=cat(3,zeros(1,1,...
    (tau(mx,px)/Ts)),H(mx,px,1:end));
    end
end
[Ad2,Bd2,Cd2,Dd2,sH2]=mimodimpulse2dss(H1,tol);
A=Ad2;
B=Bd2(:,1);
E=Bd2(:,2);
G=Bd2(:,3);
Cy=Cd2;
Cz=Cy;
```

```
[Ac1 Bcy1 Bcr1 Cc1 Dcy1 Dcr1] = ss_MPC(A1a,B1a,C1a,K1a,zeros(length(A1a),...
        size(C1a,1)),1,eye(size(C1a,1)),lambda,nbs);

[Acl1 Bwcl1 Bvcl1 Brcl1 Bdcl1 Czcl1 Cycl1 Cucl1] = ss_closed_loop(A,...
        B,Cy,Cy,E,G,Ac1,Bcy1,Bcr1,Cc1,Dcy1,Dcr1);


if obj==1

T = 0:1:nbs;
[X2,X2u] = dstep_rsp(Acl1,Bdcl1,Czcl1,Cucl1,zeros(1,1),zeros(1,1),T);
zz=sum(abs(X2));
end

% Processnoise propogation

Rxx=dlyap(Acl1,Bwcl1*Rww*Bwcl1');
Ryy=Cycl1*Rxx*Cycl1';

Ryw=Ryy;

% measurement noise propogation

Rxx=dlyap(Acl1,Bvcl1*Rvv*Bvcl1');
Ryy=Cycl1*Rxx*Cycl1'+Rvv;

Ryv=Ryy;

Ryy=Ryw+Ryv;

if obj==2
zz=max(eig(Ryy));
end
if obj==3
zz=det(Ryy);
end
if obj==4
zz=trace(Ryy);
end
if obj==5
T = 0:1:nbs;
[X2,X2u] = dstep_rsp(Acl1,Bdcl1,Czcl1,Cucl1,zeros(1,1),zeros(1,1),T);
zz=sum(abs(X2));
zz=zz*Ryy;
end
if obj==6
T = 0:1:nbs;
[X1,X1u] = dstep_rsp(Acl1,Brcl1,Czcl1,Cucl1,0,Dcr1,T);
[X2,X2u] = dstep_rsp(Acl1,Bdcl1,Czcl1,Cucl1,zeros(1,1),zeros(1,1),T);
zz=sum(abs(X2))+sum(abs(ones(size(X1,1),size(X1,2))-X1),2);
end

function [in,eq] = eval_MS(A,B,Cy,Cz,E,G,tau,Ts,x,nbs,MS_max)
```

```
lambda=diag(10.^x(2));
alpha=x(1);

%------------------------- Delta-ARX Controller Model -----------------------------%

z=tf('z');
[a,b]=ss2tf(A,B,Cy,0);
Gzu=minreal(tf(a,b,Ts))*z^(-ceil(tau(1)/Ts));

A_arx1 = cell2mat(Gzu.den);
B_arx1 = cell2mat(Gzu.num); B_arx1 = B_arx1(2:end);

A_arx_m1 = [A_arx1 0] - [0 A_arx1]; % [1 - q^-1] A_arx
B_arx_m1 = [B_arx1 0] - [0 B_arx1]; % [1 - q^-1] B_arx
C_arx_m1 = [1 -alpha];              % [1 -alpha*q^-1]

%---------------- Conversion to State Space (innovation form ---------------------%

A1=zeros(length(B_arx_m1));
A1(1:length(A_arx_m1)-1)=-A_arx_m1(2:end)';
A1(1:end-1,2:end)=eye(length(B_arx_m1)-1);
B1=B_arx_m1';
K1=([C_arx_m1(2:end) zeros(1,length(B_arx_m1)-length(C_arx_m1)+1)]')...
    +A1(:,1);
C1=[1 zeros(1,length(A1)-1)];

%----------------------------- minimum realization-----------------------------------%

tol=1e-8;

Nmax=100;

H = mimodss2dimpulse(A1,[B1 K1],C1,zeros(1,2),Nmax);
[Ad,Bd,Cd,Dd,sH]=mimodimpulse2dss(H,tol);

A1a=Ad; B1a=Bd(:,1); K1a=Bd(:,2); C1a=Cd;

%----------------------------- add delays to ss-model ------------------------------%

m=size(Cy,1); p=size([B E G],2);
H = mimodss2dimpulse(A,[B E G],Cy,zeros(m,p),Nmax);
H1=zeros(m,p,size(H,3)+max(tau)/Ts);
for mx=1:m
    for px=1:p
    H1(mx,px,1:(size(H,3)+(tau(mx,px)/Ts)))=cat(3,zeros(1,1,...
    (tau(mx,px)/Ts)),H(mx,px,1:end));
    end
end

[Ad2,Bd2,Cd2,Dd2,sH2]=mimodimpulse2dss(H1,tol);
A=Ad2;
B=Bd2(:,1);
E=Bd2(:,2);
G=Bd2(:,3);
Cy=Cd2;
```

```matlab
Cz=Cy;


[Ac1 Bcy1 Bcr1 Cc1 Dcy1 Dcr1] = ss_MPC(A1a,B1a,C1a,K1a,zeros(length(A1a),...
        size(C1a,1)),1,eye(size(C1a,1)),lambda,nbs);

[Acl1 Bwcl1 Bvcl1 Brcl1 Bdcl1 Czcl1 Cycl1 Cucl1] = ss_closed_loop(A,...
        B,Cy,Cy,E,G,Ac1,Bcy1,Bcr1,Cc1,Dcy1,Dcr1);

w=0:0.001:(pi/Ts);

    [mag,phase]=bode(ss(Acl1,Bvcl1,Cycl1,1,Ts),w);

    [val i]= max(mag);

    if i==length(w)
        w1=linspace(w(i-1),w(i),100);
    end
    if i==1
        w1=linspace(w(i),w(i+1),100);
    end
    if i > 1 && i < length(w)
        w1=linspace(w(i-1),w(i+1),100);
    end

    [mag,phase]=bode(ss(Acl1,Bvcl1,Cycl1,1,Ts),w1);

    MS=max(mag);

    in=MS-MS_max;
    eq=[];

function [MS J Jd Rvu Rvy Rwu Rwy]=...
    evaluation(A,B,Cy,Cz,E,G,tau,Ts,Rww,Rvv,x,nbs)

lambda=diag(10.^x(2));
alpha=x(1);

%----------------------- Delta-ARX Controller Model --------------------------%

z=tf('z');
[a,b]=ss2tf(A,B,Cy,0);
Gzu=minreal(tf(a,b,Ts))*z^(-ceil(tau(1)/Ts));

A_arx1 = cell2mat(Gzu.den);
B_arx1 = cell2mat(Gzu.num); B_arx1 = B_arx1(2:end);

A_arx_m1 = [A_arx1 0] - [0 A_arx1]; % [1 - q^-1] A_arx
B_arx_m1 = [B_arx1 0] - [0 B_arx1]; % [1 - q^-1] B_arx
C_arx_m1 = [1 -alpha];              % [1 -alpha*q^-1]

%--------------- Conversion to State Space (innovation form -----------------%

A1=zeros(length(B_arx_m1));
A1(1:length(A_arx_m1)-1)=-A_arx_m1(2:end)';
```

```
A1(1:end-1,2:end)=eye(length(B_arx_m1)-1);
B1=B_arx_m1';
K1=([C_arx_m1(2:end) zeros(1,length(B_arx_m1)-length(C_arx_m1)+1)]')...
    +A1(:,1);
C1=[1 zeros(1,length(A1)-1)];

%------------------------- minimum realization-----------------------------%

tol=1e-8;

Nmax=100;

H = mimodss2dimpulse(A1,[B1 K1],C1,zeros(1,2),Nmax);
[Ad,Bd,Cd,Dd,sH]=mimodimpulse2dss(H,tol);

A1a=Ad; B1a=Bd(:,1); K1a=Bd(:,2); C1a=Cd;

%------------------------- add delays to ss-model --------------------------%

m=size(Cy,1); p=size([B E G],2);
H = mimodss2dimpulse(A,[B E G],Cy,zeros(m,p),Nmax);
H1=zeros(m,p,size(H,3)+max(tau)/Ts);
for mx=1:m
    for px=1:p
    H1(mx,px,1:(size(H,3)+(tau(mx,px)/Ts)))=cat(3,zeros(1,1,...
    (tau(mx,px)/Ts)),H(mx,px,1:end));
    end
end

[Ad2,Bd2,Cd2,Dd2,sH2]=mimodimpulse2dss(H1,tol);
A=Ad2;
B=Bd2(:,1);
E=Bd2(:,2);
G=Bd2(:,3);
Cy=Cd2;
Cz=Cy;


[Ac1 Bcy1 Bcr1 Cc1 Dcy1 Dcr1] = ss_MPC(A1a,B1a,C1a,K1a,zeros(length(A1a),...
        size(C1a,1)),1,eye(size(C1a,1)),lambda,nbs);

[Acl1 Bwcl1 Bvcl1 Brcl1 Bdcl1 Czcl1 Cycl1 Cucl1] = ss_closed_loop(A,...
        B,Cy,Cy,E,G,Ac1,Bcy1,Bcr1,Cc1,Dcy1,Dcr1);


T=1:nbs;

ny=size(Cycl1,1);

[X1,X1u] = dstep_rsp(Acl1,Brcl1,Czcl1,Cucl1,0,Dcr1,T);
[X2,X2u] = dstep_rsp(Acl1,Bdcl1,Czcl1,Cucl1,zeros(1,1),zeros(1,1),T);

w=logspace(-4, log10(pi/Ts),1000); % Decide number of points (default 1000)

[Sz1a,Sz1b]=ss2tf(Acl1,Bvcl1,Cycl1,1);
```

```
Swz1=frqrsp_dtf(Sz1a,Sz1b,w,Ts);
MS=max(abs(Swz1));

J=sum(abs(ones(size(X1,1),size(X1,2))-X1),2);
Jd=sum(abs(X2));

% Processnoise propogation

Rxx1=dlyap(Acl1,Bwcl1*Rww*Bwcl1');
Ryy=Cycl1*Rxx1*Cycl1';

Rwy=Ryy;
Rwu=Cucl1*Rxx1*Cucl1';

% measurement noise propogation

Rxx2=dlyap(Acl1,Bvcl1*Rvv*Bvcl1');
Ryy=Cycl1*Rxx2*Cycl1'+Rvv;

Rvy=Ryy;

Rvu=Cucl1*Rxx2*Cucl1'+Dcy1*Rvv*Dcy1';
```

E.2.2 MIMO Optimization Toolbox

E.2.2.1 Initiation of MIMO-Toolbox: Wood-Barry

```
clear all;
close all;
clc;

addpath MPC_tuning
addpath MPC_tuning\Realization

% Test file for optim_tuning

num1 = 12.8;       %Y1_U1
den1 = [16.7 1];
tau1  = 1;

num2 = -18.9;      %Y1_U2
den2 = [21 1];
tau2 = 3;

num3 = 6.6;        %Y2_U1
den3 = [10.9 1];
tau3  = 7;

num4 = -19.4;      %Y2_U2
den4 = [14.4 1];
tau4 = 3;
```

```
% Continuous time disturbance transfers (Gyd)

dnum1 = 3.8;       %Y1_D1
dden1 = [14.9 1];
dtau1 = 8.1;

dnum2 = 4.9;
dden2 = [13.2 1];
dtau2 = 3.4;

% Initialize cell vector required for ss computation function

num=cell(2,4); den=cell(2,4); tau=zeros(2,4);
num{1,1}=num1; num{1,2}=num2; num{1,3}=dnum1; num{1,4}=dnum1;
num{2,1}=num3; num{2,2}=num4; num{2,3}=dnum2; num{2,4}=dnum2;
den{1,1}=den1; den{1,2}=den2; den{1,3}=dden1; den{1,4}=dden1;
den{2,1}=den3; den{2,2}=den4; den{2,3}=dden2; den{2,4}=dden2;

% Conditions for computation function

Ts=1;

Nmax=100; tol=1e-8;

[Ad,Bd,Cd,Dd,sH]=mimoctf2dss(num,den,tau,Ts,Nmax,tol);

tau(1,1)=tau1; tau(1,2)=tau2; tau(1,3)=dtau1; tau(1,4)=dtau1;
tau(2,1)=tau3; tau(2,2)=tau4; tau(2,3)=dtau2; tau(2,4)=dtau2;

A=Ad; B=Bd(:,1:2); E=Bd(:,3); G=Bd(:,4);

Cy=Cd; Cz=Cd;

Rvv=eye(2);
Rww=1;

x0=[0.99 0.99 1.01 1.01 1.01 1.01];
x1=[0.99 0.99 5.99 5.99 5.99 5.99];  % feasible starting point (MS=1.1144)
x2=[0.01 0.01 3 3 3 3];

  for xt=[x0' x1' x2']

 [MS J Jd Rvu Rvy Rwu Rwy x]=optim_tuning5(A,B,Cy,Cz,E,G,tau,Ts,0, 1,...
     1e1, 1e6,1e1,1e6,400,3.5,Rww,Rvv,'interior-point',7,xt')

  end

  for xt=[x0' x1' x2']

  [MS J Jd Rvu Rvy Rwu Rwy x]=optim_tuning5(A,B,Cy,Cz,E,G,tau,Ts,0, 1,...
      1e1, 1e6,1e1,1e6,400,3.5,Rww,Rvv,'active-set',7,xt')

  end
```

```
  for xt=[x0' x1' x2']

  [MS J Jd Rvu Rvy Rwu Rwy x]=optim_tuning5(A,B,Cy,Cz,E,G,tau,Ts,0, 1,...
      1e1, 1e6,1e1,1e6,400,3.5,Rww,Rvv,'sqp',7,xt')

  end
%nbs=400;
%[MS J Jd Rvu Rvy Rwu Rwy]=evaluation2(A,B,Cy,Cz,E,G,tau,Ts,Rww,Rvv,x2,nbs)
```

E.2.2.2 Initiation of MIMO-Toolbox: Cement Mill

```
clear all;
close all;
%clc;

addpath MPC_tuning
addpath MPC_tuning\Realization

% Test file for optim_tuning

num1 = 0.62;                    % Y1
den1 = conv([45 1],[8 1]);      %
tau1  = 5;

num2 = 0.29*[8 1];              %
den2 = conv([2 1],[38 1]);      %
tau2 = 1.5;

num3 = -15;                     % Y2
den3 = [60 1];                  % A3
tau3  = 5;

num4 = 5;                       % B4
den4 = conv([14 1],[1 1]);      % A4
tau4 = 0.1;

% Continuous time disturbance transfers (Gyd)

dnum1 = -1;       %Y1_D1
dden1 = conv([32 1],[21 1]);
dtau1 = 3;

dnum2 = 60;
dden2 = conv([30 1],[20 1]);
dtau2 = 3.4;

% Initialize cell vector required for ss computation function

num=cell(2,4); den=cell(2,4); tau=zeros(2,4);
num{1,1}=num1; num{1,2}=num2; num{1,3}=dnum1; num{1,4}=dnum1;
num{2,1}=num3; num{2,2}=num4; num{2,3}=dnum2; num{2,4}=dnum2;
```

```
den{1,1}=den1; den{1,2}=den2; den{1,3}=dden1; den{1,4}=dden1;
den{2,1}=den3; den{2,2}=den4; den{2,3}=dden2; den{2,4}=dden2;

% Conditions for computation function

Ts=2;

Nmax=100; tol=1e-8;

[Ad,Bd,Cd,Dd,sH]=mimoctf2dss(num,den,tau,Ts,Nmax,tol);

tau(1,1)=tau1; tau(1,2)=tau2; tau(1,3)=dtau1; tau(1,4)=dtau1;
tau(2,1)=tau3; tau(2,2)=tau4; tau(2,3)=dtau2; tau(2,4)=dtau2;

A=Ad; B=Bd(:,1:2); E=Bd(:,3); G=Bd(:,4);

Cy=Cd; Cz=Cd;

Rvv=diag([0.1 100]);
Rww=1;

x0=[0.99 0.99 1.01 1.01 1.01 1.01];
x1=[0.99 0.99 5.99 5.99 5.99 5.99];  % feasible starting point (MS=1.1144)
x2=[0.01 0.01 3 3 3 3];

  for xt=[x0' x1' x2']

  [MS J Jd Rvu Rvy Rwu Rwy x]=optim_tuning5(A,B,Cy,Cz,E,G,tau,Ts,0, 1,...
     1e1, 1e6,1e1,1e6,400,1.3,Rww,Rvv,'interior-point',7,xt')

  end

  for xt=[x0' x1' x2']

  [MS J Jd Rvu Rvy Rwu Rwy x]=optim_tuning5(A,B,Cy,Cz,E,G,tau,Ts,0, 1,...
      1e1, 1e6,1e1,1e6,400,1.3,Rww,Rvv,'active-set',7,xt')

  end


  for xt=[x0' x1' x2']

   [MS J Jd Rvu Rvy Rwu Rwy x]=optim_tuning5(A,B,Cy,Cz,E,G,tau,Ts,0, 1,...
      1e1, 1e6,1e1,1e6,400,1.3,Rww,Rvv,'sqp',7,xt')

  end


%x=[0.9923 0.8503 2.5825 2.8487 5.9959 5.6890]; % det MS=1.3

%nbs=400;
%[MS J Jd Rvu Rvy Rwu Rwy]=evaluation2(A,B,Cy,Cz,E,G,tau,Ts,Rww,Rvv,x,nbs)
%Ryy=Rwy+Rvy
```

E.2.2.3 Algorithm: MIMO Optimization Toolbox

```
%------------------------------------------------------------------------%
% Constrained Optimization of tuning (MIMO)
%
% Daniel Olesen, DTU
%
% 2012-06-20 (v3.00)
%------------------------------------------------------------------------%

function [MS J Jd Rvu Rvy Rwu Rwy x]...
    =optim_tuning5(A,B,Cy,Cz,E,G,tau,Ts,alpha_min,alpha_max,...
    Qmin,Qmax,Smin, Smax, nbs, MS_max, Rww, Rvv, solver, obj, x0)

lb=[alpha_min alpha_min log10(Qmin) log10(Qmin) log10(Smin) log10(Smin)];
ub=[alpha_max alpha_max log10(Qmax) log10(Qmax) log10(Smax) log10(Smax)];

disp(solver);
options = optimset('Algorithm',solver,'tolFun',1e-8,'TolX',1e-10,...
    'display','iter','MaxFunEvals',1200);

x = fmincon(@(x)eval(A,B,Cy,Cz,E,G,tau,Ts,Rww,Rvv,x,nbs,obj),...
    x0,[],[],[],[],lb,ub,@(x)eval_MS(A,B,Cy,Cz,E,G,tau,Ts,x,nbs,MS_max),options);


[MS J Jd Rvu Rvy Rwu Rwy]=evaluation(A,B,Cy,Cz,E,G,tau,Ts,Rww,Rvv,x,nbs);

function [zz] = eval(A,B,Cy,Cz,E,G,tau,Ts,Rww,Rvv,x,nbs,obj)

m=size(Cy,1);
p=size(B,2);

Q=diag(10.^x(m+1:2*m));
S=diag(10.^x(2*m+1:end));
alpha=x(1:m);

%------------------------------ ARIMAX Model ------------------------------%

z=tf('z');

Gzu=cell(m,p); A_arx=cell(m,1); B_arx=cell(m,p);
A_armax=cell(m,1); B_armax=cell(m,p); C_armax=cell(m,1);
A_arx1=[];

Ac=cell(m,m); Bc=cell(m,p); Cc=cell(m); Kc=cell(m,m);

for mx=1:m % Generate discrete transferfunction description
    for px=1:p

    [h,th]=sisodss2dimpulse(A,B(:,px),Cy(mx,:),0,0,100,Ts);

    if ceil(tau(mx,px)/Ts)==floor(tau(mx,px)/Ts)
        hxx=zeros(1,1,length(h)); hxx(:,:,1:end)=h;
```

```
    else
        c=ceil(tau(mx,px)/Ts)-(tau(mx,px)/Ts);
        hxx=zeros(1,1,length(h));
        hxx(:,:,1:end)=[c*h(1:end-1); 0]+[0; (1-c)*h(1:end-1)];
    end
    [a11,b11,c11,d11]=sisodimpulse2dss(hxx,1e-8);
    %[a1,b1]=ss2tf(a11,b11,c11,d11,Ts);
    [a1,b1]=ss2tf(a11,b11,c11,d11);
    Gzu{mx,px}=tf(a1,b1,Ts)*z^(-floor(tau(mx,px)/Ts)+1);
    end
end

for mx=1:m % Compute ARMAX polynomials
    for px=1:p
        if px==1
            a_arx = conv(Gzu{mx,px}.den{1,1},Gzu{mx,px+1}.den{1,1});
        elseif px >= 3
            a_arx = conv(a_arx,Gzu{mx,px}.den{1,1});
        end
        if p > 2
            for pxx=1:p-1
                if pxx==~px
                    if pxx+1==px
                        t = conv(Gzu{mx,pxx}.den{1,1},Gzu{mx,pxx+2}.den{1,1});
                    else
                        t = conv(Gzu{mx,pxx}.den{1,1},Gzu{mx,pxx+1}.den{1,1});
                    end
                    if isempty(A_arx1)&&p>2
                        A_arx1=t;
                    else
                        A_arx1=conv(A_arx1,t);
                    end
                end
            end
        else
            if px==1
               A_arx1 = Gzu{mx,2}.den{1,1};
            else
               A_arx1 = Gzu{mx,1}.den{1,1};
            end
        end


        B_arx{mx,px} = conv(Gzu{mx,px}.num{1,1}, A_arx1);
        A_arx1=[]; % clear variable before next iteration
    end
    A_arx{mx}=a_arx;
    a_arx=[];

    A_armax{mx}=[A_arx{mx} 0] - [0 A_arx{mx}];   % [1 - q^-1] A_arx
    nn=0;

    for i=1:p
    B_armax{mx,i} = [B_arx{mx,i} 0] - [0 B_arx{mx,i}]; % [1 - q^-1] B_arx
        if length(B_armax{mx,i})>nn
```

```matlab
            nn=length(B_armax{mx,i});
        end
    end
    C_armax{mx}=[1 -alpha(mx)]; % [1 -alpha*q^-1]

%-------------------------------- State Space Model -------------------------------------%
A1=zeros(nn);
A1(1:length(A_armax{mx,1})-1)=-A_armax{mx,1}(2:end)';
A1(1:end-1,2:end)=eye(nn-1);
K1=([C_armax{mx,1}(2:end) zeros(1,nn-length(C_armax{mx,1})+1)]')...
    +A1(:,1);
C1=[1 zeros(1,length(A1)-1)];

Ac{mx,mx}=A1;
Cc{mx,mx}=C1;
Kc{mx,mx}=K1;

    for i=1:p
        Bc{mx,i}=B_armax{mx,i}';
    end

end
%------------------------------ Augmented State Space Model ------------------------------%
for i=1:m % rows
    for j=1:m % columns
            if not(i==j)
            Ac{i,j}=zeros(length(Ac{i,i}),length(Ac{j,j}));
            Ac{j,i}=zeros(length(Ac{j,j}),length(Ac{i,i}));
            Cc{i,j}=zeros(1,length(Cc{j,j}));
            Cc{j,i}=zeros(1,length(Cc{i,i}));
            Kc{i,j}=zeros(length(Kc{i,i}),1);
            Kc{j,i}=zeros(length(Kc{j,j}),1);
            end
    end
end
Ac = cell2mat(Ac);
Bc = cell2mat(Bc);
Kc = cell2mat(Kc);
Cc = cell2mat(Cc);

%------------------------------- minimum realization------------------------------------%

tol=1e-8;

Nmax=100;

H = mimodss2dimpulse(Ac,[Bc Kc],Cc,zeros(2,4),Nmax);

[Ad1,Bd1,Cd1,Dd1,sH1]=mimodimpulse2dss(H,tol);

Ac=Ad1; Bc=Bd1(:,1:2); Kc=Bd1(:,3:end); Cc=Cd1;
%------------------------------- add delays to ss-model --------------------------------%

m=size(Cy,1); p=size([B E G],2);
```

```
H = mimodss2dimpulse(A,[B E G],Cy,zeros(m,p),100);

H1=zeros(m,p,size(H,3)+ceil(max(max(tau))/Ts));

for mx=1:m
    for px=1:p

        if ceil(tau(mx,px)/Ts)==floor(tau(mx,px)/Ts)

        H1(mx,px,1:(size(H,3) + ceil((tau(mx,px)/Ts))))=...
        cat(3,zeros(1,1,ceil((tau(mx,px)/Ts))),H(mx,px,1:end));

        else

        xxx = ceil(tau(mx,px)/Ts);
        xxy = floor(tau(mx,px)/Ts);
        xxz = xxx-(tau(mx,px)/Ts);
        aa=cat(3,H(mx,px,1:end),zeros(1,1,1));
        ab=cat(3,zeros(1,1,1),H(mx,px,1:end));
        ac=xxz*aa+(1-xxz)*ab;
        H1(mx,px,1:(size(H,3)+xxx))=cat(3,zeros(1,1,xxy),ac);

        end
    end
end

[Ad2,Bd2,Cd2,Dd2,sH2]=mimodimpulse2dss(H1(:,:,1:100),1e-12);

Ad=Ad2;
Bd=Bd2(:,1:2); Ed=Bd2(:,3); Gd=Bd2(:,4);
Cyd=Cd2;
Czd=Cyd;

[Ac1 Bcy1 Bcr1 Cc1 Dcy1 Dcr1] = ss_MPC(Ac,Bc,Cc,Kc,zeros(length(Ac),...
        size(Cc,1)),1,Q,S,nbs);

[Acl1 Bwcl1 Bvcl1 Brcl1 Bdcl1 Czcl1 Cycl1 Cucl1] = ss_closed_loop(Ad,...
        Bd,Cyd,Cyd,Ed,Gd,Ac1,Bcy1,Bcr1,Cc1,Dcy1,Dcr1);

ny=size(Cycl1,1);

[sv,w]=sigma(ss(Acl1,Bvcl1,Cycl1,eye(ny),Ts));

MS=max(max(sv));

if obj==1

T = 0:1:nbs;
[X2,X2u] = dstep_rsp(Acl1,Bdcl1,Czcl1,Cucl1,zeros(2,2),zeros(2,2),T);
zz=max(sum(abs(X2),2));
end

% Processnoise propogation

Rxx=dlyap(Acl1,Bwcl1*Rww*Bwcl1');
```

```
Ryy=Cycl1*Rxx*Cycl1';

Ryw=Ryy;

% measurement noise propogation

Rxx=dlyap(Acl1,Bvcl1*Rvv*Bvcl1');
Ryy=Cycl1*Rxx*Cycl1'+Rvv;

Ryv=Ryy;

Ryy=Ryw+Ryv;

if obj==2
zz=max(eig(Ryy));
end
if obj==3
zz=det(Ryy);
end
if obj==4
zz=trace(Ryy);
end
if obj==5
T = 0:1:nbs;
[X2,X2u] = dstep_rsp(Acl1,Bdcl1,Czcl1,Cucl1,zeros(2,2),zeros(2,2),T);
zz=sum(abs(X2),2);
zz=diag(zz)*Ryy;
zz=trace(zz);
end
if obj==6
T = 0:1:nbs;
[X2,X2u] = dstep_rsp(Acl1,Bdcl1,Czcl1,Cucl1,zeros(2,2),zeros(2,2),T);
zz=norm(sum(abs(X2),2));
end
if obj==7
T = 0:1:nbs;
[X1a,X1u] = dstep_rsp(Acl1,Brcl1(:,1),Czcl1,Cucl1,0,Dcr1,T);
[X1b,X1u] = dstep_rsp(Acl1,Brcl1(:,2),Czcl1,Cucl1,0,Dcr1,T);
J(1,1)=sum(abs(ones(size(X1a(1,:),1),size(X1a(1,:),2))-X1a(1,:)),2);
J(1,2)=sum(abs(zeros(size(X1a(2,:),1),size(X1a(2,:),2))-X1a(2,:)),2);
J(2,1)=sum(abs(zeros(size(X1b(1,:),1),size(X1b(2,:),2))-X1b(1,:)),2);
J(2,2)=sum(abs(ones(size(X1b(2,:),1),size(X1b(2,:),2))-X1b(2,:)),2);
[X2,X2u] = dstep_rsp(Acl1,Bdcl1,Czcl1,Cucl1,zeros(2,2),zeros(2,2),T);
zz=norm(J)+norm(sum(abs(X2),2));
end

function [in,eq] = eval_MS(A,B,Cy,Cz,E,G,tau,Ts,x,nbs,MS_max)
m=size(Cy,1);
p=size(B,2);

Q=diag(10.^x(m+1:2*m));
S=diag(10.^x(2*m+1:end));
alpha=x(1:m);

%---------------------------- ARIMAX Model ----------------------------%
```

```
z=tf('z');

Gzu=cell(m,p); A_arx=cell(m,1); B_arx=cell(m,p);
A_armax=cell(m,1); B_armax=cell(m,p); C_armax=cell(m,1);
A_arx1=[];

Ac=cell(m,m); Bc=cell(m,p); Cc=cell(m); Kc=cell(m,m);

for mx=1:m % Generate discrete transferfunction description
    for px=1:p

    [h,th]=sisodss2dimpulse(A,B(:,px),Cy(mx,:),0,0,100,Ts);

    if ceil(tau(mx,px)/Ts)==floor(tau(mx,px)/Ts)
        hxx=zeros(1,1,length(h)); hxx(:,:,1:end)=h;
    else
        c=ceil(tau(mx,px)/Ts)-(tau(mx,px)/Ts);
        hxx=zeros(1,1,length(h));
        hxx(:,:,1:end)=[c*h(1:end-1); 0]+[0; (1-c)*h(1:end-1)];
    end
    [a11,b11,c11,d11]=sisodimpulse2dss(hxx,1e-8);
    %[a1,b1]=ss2tf(a11,b11,c11,d11,Ts);
    [a1,b1]=ss2tf(a11,b11,c11,d11);
    Gzu{mx,px}=tf(a1,b1,Ts)*z^(-floor(tau(mx,px)/Ts)+1);
    end
end

for mx=1:m % Compute ARMAX polynomials
    for px=1:p
        if px==1
            a_arx = conv(Gzu{mx,px}.den{1,1},Gzu{mx,px+1}.den{1,1});
        elseif px >= 3
            a_arx = conv(a_arx,Gzu{mx,px}.den{1,1});
        end
        if p > 2
            for pxx=1:p-1
                if pxx==~px
                    if pxx+1==px
                        t = conv(Gzu{mx,pxx}.den{1,1},Gzu{mx,pxx+2}.den{1,1});
                    else
                        t = conv(Gzu{mx,pxx}.den{1,1},Gzu{mx,pxx+1}.den{1,1});
                    end
                    if isempty(A_arx1)&&p>2
                        A_arx1=t;
                    else
                        A_arx1=conv(A_arx1,t);
                    end
                end
            end
        else
            if px==1
               A_arx1 = Gzu{mx,2}.den{1,1};
            else
               A_arx1 = Gzu{mx,1}.den{1,1};
```

```
            end
        end


        B_arx{mx,px} = conv(Gzu{mx,px}.num{1,1}, A_arx1);
        A_arx1=[]; % clear variable before next iteration
    end
    A_arx{mx}=a_arx;
    a_arx=[];

    A_armax{mx}=[A_arx{mx} 0] - [0 A_arx{mx}];   % [1 - q^-1] A_arx
    nn=0;

    for i=1:p
    B_armax{mx,i} = [B_arx{mx,i} 0] - [0 B_arx{mx,i}]; % [1 - q^-1] B_arx
        if length(B_armax{mx,i})>nn
            nn=length(B_armax{mx,i});
        end
    end
    C_armax{mx}=[1 -alpha(mx)]; % [1 -alpha*q^-1]

%-------------------------- State Space Model ------------------------------%
A1=zeros(nn);
A1(1:length(A_armax{mx,1})-1)=-A_armax{mx,1}(2:end)';
A1(1:end-1,2:end)=eye(nn-1);
K1=([C_armax{mx,1}(2:end) zeros(1,nn-length(C_armax{mx,1})+1)]')...
    +A1(:,1);
C1=[1 zeros(1,length(A1)-1)];

Ac{mx,mx}=A1;
Cc{mx,mx}=C1;
Kc{mx,mx}=K1;

    for i=1:p
        Bc{mx,i}=B_armax{mx,i}';
    end

end
%---------------------- Augmented State Space Model ----------------------%
for i=1:m % rows
    for j=1:m % columns
            if not(i==j)
            Ac{i,j}=zeros(length(Ac{i,i}),length(Ac{j,j}));
            Ac{j,i}=zeros(length(Ac{j,j}),length(Ac{i,i}));
            Cc{i,j}=zeros(1,length(Cc{j,j}));
            Cc{j,i}=zeros(1,length(Cc{i,i}));
            Kc{i,j}=zeros(length(Kc{i,i}),1);
            Kc{j,i}=zeros(length(Kc{j,j}),1);
            end
    end
end
Ac = cell2mat(Ac);
Bc = cell2mat(Bc);
Kc = cell2mat(Kc);
Cc = cell2mat(Cc);
```

```
%——————————————————— minimum realization———————————————————————%

tol=1e−8;

Nmax=100;

H = mimodss2dimpulse(Ac,[Bc Kc],Cc,zeros(2,4),Nmax);

[Ad1,Bd1,Cd1,Dd1,sH1]=mimodimpulse2dss(H,tol);

Ac=Ad1; Bc=Bd1(:,1:2); Kc=Bd1(:,3:end); Cc=Cd1;
%——————————————————— add delays to ss−model ————————————————————%

m=size(Cy,1); p=size([B E G],2);

H = mimodss2dimpulse(A,[B E G],Cy,zeros(m,p),100);

H1=zeros(m,p,size(H,3)+ceil(max(max(tau))/Ts));

for mx=1:m
    for px=1:p

        if ceil(tau(mx,px)/Ts)==floor(tau(mx,px)/Ts)

        H1(mx,px,1:(size(H,3) + ceil((tau(mx,px)/Ts))))=...
        cat(3,zeros(1,1,ceil((tau(mx,px)/Ts))),H(mx,px,1:end));

        else

        xxx = ceil(tau(mx,px)/Ts);
        xxy = floor(tau(mx,px)/Ts);
        xxz = xxx−(tau(mx,px)/Ts);
        aa=cat(3,H(mx,px,1:end),zeros(1,1,1));
        ab=cat(3,zeros(1,1,1),H(mx,px,1:end));
        ac=xxz*aa+(1−xxz)*ab;
        H1(mx,px,1:(size(H,3)+xxx))=cat(3,zeros(1,1,xxy),ac);

        end
    end
end

[Ad2,Bd2,Cd2,Dd2,sH2]=mimodimpulse2dss(H1(:,:,1:100),1e−12);

Ad=Ad2;
Bd=Bd2(:,1:2); Ed=Bd2(:,3); Gd=Bd2(:,4);
Cyd=Cd2;
Czd=Cyd;

[Ac1 Bcy1 Bcr1 Cc1 Dcy1 Dcr1] = ss_MPC(Ac,Bc,Cc,Kc,zeros(length(Ac),...
        size(Cc,1)),1,Q,S,nbs);

[Acl1 Bwcl1 Bvcl1 Brcl1 Bdcl1 Czcl1 Cycl1 Cucl1] = ss_closed_loop(Ad,...
        Bd,Cyd,Cyd,Ed,Gd,Ac1,Bcy1,Bcr1,Cc1,Dcy1,Dcr1);
```

```
ny=size(Cycl1,1);

    w=0:0.001:(pi/Ts);

    [sv,w]=sigma(ss(Acl1,Bvcl1,Cycl1,eye(ny),Ts),w);

    [val i]=max(max(sv));

    if i==length(w)
        w1=linspace(w(i−1),w(i),100);
    end
    if i==1
        w1=linspace(w(i),w(i+1),100);
    end
    if i > 1 && i < length(w)
        w1=linspace(w(i−1),w(i+1),100);
    end

    [sv,w]=sigma(ss(Acl1,Bvcl1,Cycl1,eye(ny),Ts),w1);

    MS=max(max(sv));

    in=MS−MS_max;
    eq=[];

function [MS J Jd Rvu Rvy Rwu Rwy]=...
    evaluation(A,B,Cy,Cz,E,G,tau,Ts,Rww,Rvv,x,nbs)

m=size(Cy,1);
p=size(B,2);

Q=diag(10.^x(m+1:2*m));
S=diag(10.^x(2*m+1:end));
alpha=x(1:m);
%−−−−−−−−−−−−−−−−−−−−−−−−−−−− ARIMAX Model −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−%

z=tf('z');

Gzu=cell(m,p); A_arx=cell(m,1); B_arx=cell(m,p);
A_armax=cell(m,1); B_armax=cell(m,p); C_armax=cell(m,1);
A_arx1=[];

Ac=cell(m,m); Bc=cell(m,p); Cc=cell(m); Kc=cell(m,m);

for mx=1:m % Generate discrete transferfunction description
    for px=1:p

    [h,th]=sisodss2dimpulse(A,B(:,px),Cy(mx,:),0,0,100,Ts);

    if ceil(tau(mx,px)/Ts)==floor(tau(mx,px)/Ts)
        hxx=zeros(1,1,length(h)); hxx(:,:,1:end)=h;
    else
        c=ceil(tau(mx,px)/Ts)−(tau(mx,px)/Ts);
        hxx=zeros(1,1,length(h));
        hxx(:,:,1:end)=[c*h(1:end−1); 0]+[0; (1−c)*h(1:end−1)];
```

```
    end
    [a11,b11,c11,d11]=sisodimpulse2dss(hxx,1e-8);
    %[a1,b1]=ss2tf(a11,b11,c11,d11,Ts);
    [a1,b1]=ss2tf(a11,b11,c11,d11);
    Gzu{mx,px}=tf(a1,b1,Ts)*z^(-floor(tau(mx,px)/Ts)+1);
    end
end

for mx=1:m % Compute ARMAX polynomials
    for px=1:p
        if px==1
            a_arx = conv(Gzu{mx,px}.den{1,1},Gzu{mx,px+1}.den{1,1});
        elseif px >= 3
            a_arx = conv(a_arx,Gzu{mx,px}.den{1,1});
        end
        if p > 2
            for pxx=1:p-1
                if pxx==~px
                    if pxx+1==px
                        t = conv(Gzu{mx,pxx}.den{1,1},Gzu{mx,pxx+2}.den{1,1});
                    else
                        t = conv(Gzu{mx,pxx}.den{1,1},Gzu{mx,pxx+1}.den{1,1});
                    end
                    if isempty(A_arx1)&&p>2
                        A_arx1=t;
                    else
                        A_arx1=conv(A_arx1,t);
                    end
                end
            end
        else
            if px==1
               A_arx1 = Gzu{mx,2}.den{1,1};
            else
               A_arx1 = Gzu{mx,1}.den{1,1};
            end
        end


        B_arx{mx,px} = conv(Gzu{mx,px}.num{1,1}, A_arx1);
        A_arx1=[]; % clear variable before next iteration
    end
    A_arx{mx}=a_arx;
    a_arx=[];

    A_armax{mx}=[A_arx{mx} 0] - [0 A_arx{mx}];   % [1 - q^-1] A_arx
    nn=0;

    for i=1:p
    B_armax{mx,i} = [B_arx{mx,i} 0] - [0 B_arx{mx,i}]; % [1 - q^-1] B_arx
        if length(B_armax{mx,i})>nn
            nn=length(B_armax{mx,i});
        end
    end
    C_armax{mx}=[1 -alpha(mx)]; % [1 -alpha*q^-1]
```

```

%------------------------------ State Space Model ------------------------------%
A1=zeros(nn);
A1(1:length(A_armax{mx,1})-1)=-A_armax{mx,1}(2:end)';
A1(1:end-1,2:end)=eye(nn-1);
K1=([C_armax{mx,1}(2:end) zeros(1,nn-length(C_armax{mx,1})+1)]')...
    +A1(:,1);
C1=[1 zeros(1,length(A1)-1)];

Ac{mx,mx}=A1;
Cc{mx,mx}=C1;
Kc{mx,mx}=K1;

    for i=1:p
        Bc{mx,i}=B_armax{mx,i}';
    end

end
%------------------------------ Augmented State Space Model ------------------------------%
for i=1:m % rows
    for j=1:m % columns
            if not(i==j)
            Ac{i,j}=zeros(length(Ac{i,i}),length(Ac{j,j}));
            Ac{j,i}=zeros(length(Ac{j,j}),length(Ac{i,i}));
            Cc{i,j}=zeros(1,length(Cc{j,j}));
            Cc{j,i}=zeros(1,length(Cc{i,i}));
            Kc{i,j}=zeros(length(Kc{i,i}),1);
            Kc{j,i}=zeros(length(Kc{j,j}),1);
            end
    end
end
Ac = cell2mat(Ac);
Bc = cell2mat(Bc);
Kc = cell2mat(Kc);
Cc = cell2mat(Cc);

%------------------------------ minimum realization------------------------------%

tol=1e-8;

Nmax=100;

H = mimodss2dimpulse(Ac,[Bc Kc],Cc,zeros(2,4),Nmax);

[Ad1,Bd1,Cd1,Dd1,sH1]=mimodimpulse2dss(H,tol);

Ac=Ad1; Bc=Bd1(:,1:2); Kc=Bd1(:,3:end); Cc=Cd1;

figure,dstep(Ac,Bc,Cc,zeros(2,2));

%------------------------------ add delays to ss-model ------------------------------%

m=size(Cy,1); p=size([B E G],2);

H = mimodss2dimpulse(A,[B E G],Cy,zeros(m,p),100);
```

```
H1=zeros(m,p,size(H,3)+ceil(max(max(tau))/Ts));

for mx=1:m
    for px=1:p

        if ceil(tau(mx,px)/Ts)==floor(tau(mx,px)/Ts)

        H1(mx,px,1:(size(H,3) + ceil((tau(mx,px)/Ts))))=...
        cat(3,zeros(1,1,ceil((tau(mx,px)/Ts))),H(mx,px,1:end));

        else

        xxx = ceil(tau(mx,px)/Ts);
        xxy = floor(tau(mx,px)/Ts);
        xxz = xxx-(tau(mx,px)/Ts);
        aa=cat(3,H(mx,px,1:end),zeros(1,1,1));
        ab=cat(3,zeros(1,1,1),H(mx,px,1:end));
        ac=xxz*aa+(1-xxz)*ab;
        H1(mx,px,1:(size(H,3)+xxx))=cat(3,zeros(1,1,xxy),ac);

        end
    end
end

[Ad2,Bd2,Cd2,Dd2,sH2]=mimodimpulse2dss(H1(:,:,1:100),1e-12);

figure,dstep(Ad2,Bd2,Cd2,Dd2);

Ad=Ad2;
Bd=Bd2(:,1:2); Ed=Bd2(:,3); Gd=Bd2(:,4);
Cyd=Cd2;
Czd=Cyd;

[Ac1 Bcy1 Bcr1 Cc1 Dcy1 Dcr1] = ss_MPC(Ac,Bc,Cc,Kc,zeros(length(Ac),...
        size(Cc,1)),1,Q,S,nbs);

[Acl1 Bwcl1 Bvcl1 Brcl1 Bdcl1 Czcl1 Cycl1 Cucl1] = ss_closed_loop(Ad,...
        Bd,Cyd,Cyd,Ed,Gd,Ac1,Bcy1,Bcr1,Cc1,Dcy1,Dcr1);

T=1:nbs;

ny=size(Cycl1,1);

[X1,X1u] = dstep_rsp(Acl1,Brcl1,Czcl1,Cucl1,0,Dcr1,T);
[X2,X2u] = dstep_rsp(Acl1,Bdcl1,Czcl1,Cucl1,zeros(2,2),zeros(2,2),T);

figure,plot(X2');

w=0:0.001:(pi/Ts);

    [sv,w]=sigma(ss(Acl1,Bvcl1,Cycl1,eye(ny),Ts),w);

    [val i]=max(max(sv));

```

```
    if i==length(w)
        w1=linspace(w(i-1),w(i),100);
    end
    if i==1
        w1=linspace(w(i),w(i+1),100);
    end
    if i > 1 && i < length(w)
        w1=linspace(w(i-1),w(i+1),100);
    end

    [sv,w]=sigma(ss(Acl1,Bvcl1,Cycl1,eye(ny),Ts),w1);

    MS=max(max(sv));

[X1a,X1u] = dstep_rsp(Acl1,Brcl1(:,1),Czcl1,Cucl1,0,Dcr1,T);
[X1b,X1u] = dstep_rsp(Acl1,Brcl1(:,2),Czcl1,Cucl1,0,Dcr1,T);
J(1,1)=sum(abs(ones(size(X1a(1,:),1),size(X1a(1,:),2))-X1a(1,:)),2);
J(1,2)=sum(abs(zeros(size(X1a(2,:),1),size(X1a(2,:),2))-X1a(2,:)),2);
J(2,1)=sum(abs(zeros(size(X1b(1,:),1),size(X1b(2,:),2))-X1b(1,:)),2);
J(2,2)=sum(abs(ones(size(X1b(2,:),1),size(X1b(2,:),2))-X1b(2,:)),2);

Jd=sum(abs(X2),2);

% Processnoise propogation

Rxx1=dlyap(Acl1,Bwcl1*Rww*Bwcl1');
Ryy=Cycl1*Rxx1*Cycl1';

Rwy=Ryy;
Rwu=Cucl1*Rxx1*Cucl1';

% measurement noise propogation

Rxx2=dlyap(Acl1,Bvcl1*Rvv*Bvcl1');
Ryy=Cycl1*Rxx2*Cycl1'+Rvv;

Rvy=Ryy;

Rvu=Cucl1*Rxx2*Cucl1'+Dcy1*Rvv*Dcy1';
```

E.2.3 Common files

```
%------------------------------------------------------------------------%
% Calculation of step response of disrete state-space model
%
% Daniel Olesen, DTU
%
% 2012-05-13 (v2.00)
%------------------------------------------------------------------------%
function [Y,U] = dstep_rsp(Ad,Bd,Cz,Cu,Dz,Du,T)

nr=size(Bd,2);
```

```
nu=size(Cu,1);
ny=size(Cz,1);
nx=size(Ad,1);

Y=zeros(ny,length(T));
U=zeros(nu,length(T));
x = zeros(nx,length(T));

k=1;

for i=T
    Y(:,k)=Cz*x(:,k)+Dz*ones(nu,1);
    U(:,k)=Cu*x(:,k)+Du*ones(nu,1);
    k=k+1;
    x(:,k)=Ad*x(:,k-1)+Bd*ones(nr,1);
end
```

```
%--------------------------------------------------------------------------%
% Calculation of Frequency response of disrete transferfunctiosn
%
% Daniel Olesen, DTU
%
% 2012-02-19 (v1.00)
%--------------------------------------------------------------------------%

function [val] = frqrsp_dtf(a,b,w,Ts)

a_iw=zeros(length(a),length(w));
b_iw=zeros(length(b),length(w));

for j=1:max(length(a),length(b));

    tmp=exp(i*w*Ts).^(j-1);
    if j <= length(a)
    a_iw(j,:)= tmp;
    end
    if j <= length(b)
    b_iw(j,:)=tmp;
    end

end

val=(fliplr(a)*a_iw)./(fliplr(b)*b_iw);

% a_iw=zeros(length(a),length(w));
%
% for j=1:length(a)
%     a_iw(j,:)=exp(i*w*Ts).^(j-1);
% end
%
% b_iw=zeros(length(b),length(w));
%
% for j=1:length(b)
%     b_iw(j,:)=exp(i*w*Ts).^(j-1);
```

```
% end
%
% val=(fliplr(a)*a_iw)./(fliplr(b)*b_iw);
```

```
function [H,Mx0,Mum1,MR,Mw] = ...
            MPC_matrix(A,B,G,Cz,qz,S,nbs)


% Initializing the size the matrices
nx = length(A);
nu = size(B,2);
nz = size(Cz,1);

        Gamma  = zeros(nbs*nz,nbs*nu); % note systems must be siso
        Phi    = zeros(nbs*nz,nx);
        Phi_G  = zeros(nbs*nz,1);
        T   = Cz;
        T_G = Cz;
        ii1 = 1;
        ii2 = nz;

        for ii=1:nbs
            Gamma( ii1:ii2,1:nu) = T*B;  % T equals C*A^(k-1)
            T = T*A;
            Phi(ii1:ii2,1:nx) = T;       % T equals C*A^(k)
            Phi_G(ii1:ii2,1:length(Cz*G)) = T_G*G; % equals C*A^(k-1)*G;
            ii1 = ii1+nz;
            ii2 = ii2+nz;
            T_G = T_G*A;
        end

        row_idx=nz+1;

        for column=nu+1:nu:size(Gamma,2)
            Gamma(row_idx:end,column:column+nu-1)=...
                Gamma(1:size(Gamma,1)-row_idx+1,1:nu);
            row_idx=row_idx+nz;
        end



QZ=kron(eye(nbs),qz);


if isvector(S)
    hs=diag(S,0);       % Generates a matrix if S is a vector (or a "scalar")
end

% Designing the diagonal of HS
nbs1=nbs-1;
HS=diag(ones(nbs1,1)*2,0);
HS(nbs,nbs)=1;

% Designing the subdiagonal of HS
```

```
HS=HS-diag(ones(nbs1,1),-1)-diag(ones(nbs1,1),1);
HS=kron(HS,S);

% Designing the Mu_{-1}
Mum1=zeros(nbs,1);
Mum1(1,1)=1;
Mum1=-kron(Mum1,S);

Gamma_QZ = Gamma' * QZ;

% H   = Gamma' * QZ * Gamma + HS;
% Mx0 = Gamma' * QZ * Phi;
% MR  = -Gamma'* QZ;
% Mw  = Gamma' * QZ * Phi_G;

H   = Gamma_QZ * Gamma + HS;
Mx0 = Gamma_QZ * Phi;
MR  = -Gamma_QZ;
Mw  = Gamma_QZ * Phi_G;
```

```
%-------------------------------------------------------------------------%
% Calculation of closed loop state space model (SISO system)
%
% Daniel Olesen, DTU
%
% 2012-02-19 (v1.00)
%-------------------------------------------------------------------------%

function [Acl Bwcl Bvcl Brcl Bdcl Czcl Cycl Cucl] = ss_closed_loop(A,B,Cy,...
          Cz,E,G,Ac,Bcy,Bcr,Cc,Dcy,Dcr)

Acl = [A+B*Dcy*Cy B*Cc;
       Bcy*Cy Ac];

Bwcl = [G;
       zeros(length(Ac),size(G,2))];

Bvcl = [B*Dcy;
        Bcy];

Brcl = [B*Dcr;
        Bcr];

Bdcl = [E;
        zeros(size(Ac,1),size(E,2))];


Czcl = [Cz zeros(size(Cz,1),size(Ac,1))];

Cycl = [Cy zeros(size(Cy,1),size(Ac,1))];

Cucl = [Dcy*Cy Cc];
```

```matlab
%-------------------------------------------------------------------------%
% Calculation of MPC controller state space model (SISO)
%
% Daniel Olesen, DTU
%
% 2012-02-19 (v1.00)
%-------------------------------------------------------------------------%

function [Ac Bcy Bcr Cc Dcy Dcr] = ss_MPC(A,B,Cz,G,Kfx,Kfw,qz,S,nbs)
[H,Mx0,Mum1,MR,Mw] = ...
                MPC_matrix(A,B,G,Cz,qz,S,nbs);
% [H,Gamma,Phi,Phi_G,Mx0,Mum1,MR,Mw,Lambda] = ...
%               MPCdesignMatrix_inn(A,B,G,Cz,qz,S,nbs);

[R,p]=chol(H);

if (p>0)
    error('H not positive def');
end

nz = size(Cz,1);
nu = size(B,2);

I0=[eye(nu); zeros(nu*(nbs-1),nu)];

Lx0=    -(R\(R'\Mx0));
LR=     -(R\(R'\MR));
Lum1=   -(R\(R'\Mum1));
Lw=     -(R\(R'\Mw));

Kx0=I0'*Lx0;
KR=I0'*LR;
Kr=KR*repmat(eye(nz),nbs,1);
Kum1=I0'*Lum1;
Kw=I0'*Lw;

Ac=[A-(A*Kfx*Cz+G*Kfw*Cz)+B*(Kx0-(Kx0*Kfx*Cz+Kw*Kfw*Cz)) B*Kum1;
    Kx0-((Kx0*Kfx+Kw*Kfw)*Cz) Kum1];

Bcy=[A*Kfx+G*Kfw+B*(Kx0*Kfx+Kw*Kfw); Kx0*Kfx+Kw*Kfw];
Dcy=(Kx0*Kfx)+(Kw*Kfw);
Bcr=[B*Kr; Kr];
Dcr=Kr;

Cc = [Kx0-((Kx0*Kfx+Kw*Kfw)*Cz) Kum1];
```

Bibliography

[ARF11] T. Amraee, A. M. Ranjbar, and R. Feuillet. Adaptive undervoltage load shedding scheme using model predictive control. *Electric Power Systems Research*, 81:1507–1513, 2011.

[CB10] S. D. Cairano and A. Bemporad. Model predictive control tuning by controller matching. *IEEE Transactions on Automatic Control*, 55:185–190, 2010.

[CR79] C. R. Cutler and B. L. Ramaker. Dynamic matrix control - a computer control algorithm. In *AICHE national meeting*, Houston, TX, April 1979.

[DS81] J. C. Doyle and G. Stein. Multivariable feedback design: Concepts for a classical/modern synthesis. *IEEE Transactions on Automatic Control*, AC26(1):4–16, 1981.

[GE11] S. Gaurang and S. Engell. Tuning MPC for desired closed-loop performance for MIMO systems. In *Proceedings of the American Control Conference*, pages 4404–4409, 2011.

[GM82] C. E. Garcia and M. Morari. Internal model control 1. a unifying review and some new results. *Ind. Eng. Chem. Process Design and Developement*, 63:308–323, 1982.

[GM85] C. E. Garcia and M. Morari. Internal model control 3. multivariable control law computation and tuning guidelines. *Ind. Eng. Chem. Process Design and Developement*, 24:484–494, 1985.

[GS10] J. L. Garriga and M. Soroush. Model predictive control tuning methods: A review. *Ind. Eng. Chem. Process Design and Developement*, 49:3505–3515, 2010.

[HC94] R. F. Hinde and D. J. Cooper. A pattern-based approach to excitation diagnostics for adaptive process control. *Chem. Eng. Sci.*, 49:1403–1415, 1994.

[Jø04] J. B Jørgensen. *Linear Model Predictive Control Toolbox - User's Guide*. 2-control ApS (www.2-control.com), 2004.

[JHR11] J. B. Jørgensen, J. K. Huusom, and J. B. Rawlings. Finite horizon MPC for systems in innovation form. In *50th IEEE Conference on Decision and Control and European Control Conference*, pages 1896–1903, 2011.

[LY94] J. H. Lee and Z. H. Yu. Tuning of model predictive controllers for robust perfomance. *Computers in Chemical Engineering*, 18:15–37, 1994.

[Mac02] J. M Maciejowski. *Predictive Control with Constraints*. Prentice Hall, 2002.

[Pou07] N. K Poulsen. *Stokastisk Adaptiv Regulering*. Published by DTU-IMM, 2007.

[PRCJ10] G. Prasath, B. Recke, M. Chidambaram, and J.B. Jørgensen. Application of soft constrained MPC to a cement mill circuit. In *Proceedings of the 9th International Symposium on Dynamics and Control of Process Systems (DYCOPS 2010)*, pages 288–293, 2010.

[PSQ02] R. S. Patwardhan, S. L. Shah, and K. S. Qi. Assessing the performance of model predictive controllers. *The Canadian Journal of Chemical Engineering*, 80:954–966, 2002.

[PV02] K. E. Parsopoulos and M. N. Vrahatis. Particle swarm optimization method for constrained optimization problems. *Intelligent Technologies - Theory and Application*, 76:214–220, 2002.

[QB03] S. Joe Qin and Thomas A. Badgwell. A survey of industrial model predictive control technology. *Control Engineering Practice*, 11:733–764, 2003.

[RM00] C. Rowe and J. Maciejowski. Tuning mpc using H_∞ loop shaping. In *Proceedings of the American Control Conference*, volume 2, pages 1332–1336, 2000.

[RRTP76] J. Richalet, A. Rault, J. L. Testud, and J. Papon. Algorithmic control of industrial processes. In *Proceedings of the 4th IFAC symposium on identification and system parameter estimation.*, pages 1119–1167, 1976.

[SC98] R. Shridhar and D. J. Cooper. A tuning strategy for unconstrained multivariable model predictive control. *Ind. Eng. Chem.*, 37:4003–4016, 1998.

[SKN+12] R. Susuki, F. Kawai, C. Nakazawa, T. Matsui, and E. Aiyoshi. Parameter optimization of model predictive control using PSO. *Electrical Engineering In Japan*, 178:40–49, 2012.

[SP05] S. Skogestad and I. Postlethwaithe. *Multivariable Feedback Control - Analysis and Design.* Wiley, 2005.

[TF03] J. O. Trierweiler and L. A. Farina. RPN tuning strategy for model predictive control. *J. Process Control*, 13:591–598, 2003.

[ZWMG95] C. Zhou, J. R. Whiteley, E. A. Misawa, and K. A. M. Gasem. Application of enhanced LQG/LTR for distillation control. *IEEE CONTROL SYSTEMS MAGAZINE*, 15:56–63, 1995.