SimTech Chair for Mathematical Systems Theory
University of Stuttgart

Project Thesis

in Bachelor Degree
Simulation Technology

Iterative Learning Control

by Tatiana Strelnikova

Examiner: Prof. Dr. Carsten W. Scherer
Examination Date: 20.06.2021

Lehrstuhl für Mathematische Systemtheorie
Universität Stuttgart

Projektarbeit

im Bachelorstudiengang
Simulation Technology

Iterative Learning Control

Prüfer: Prof. Dr. Carsten W. Scherer

vorgelegt von

Name
Straße
PLZ + Ort
E-mail

Studienfach
Fachsemester
Matrikelnummer
Abgabetermin: XX.XX.201X

Contents

List of Figures

List of Tables

Abkürzungsverzeichnis

O.B.d.A ...

Symbolverzeichnis

$G(s)$	Übertragungsfunktion eines LTI Systems
(A, B, C, D)	Realization des Systems im Zustandsraum

Chapter 1

Introduction

1.1 Motivation

It is well known that repetition is the mother of learning. A student learning a piano piece repeats it till he or she can play it well, and, possibly, without errors. A gymnastic tries to make a perfect move by exercising it dozens or hundreds of times. In both cases, the correct motion is learned and becomes ingrained into muscle memory. The converged muscle motion profile can bee seen as an open-loop control generated through repetition and learning [1]. This type of learned open-loop control strategy is the essence of Iterative Learning Control (ILC).

Nowadays, more and more tasks are carried out by industrial machines or robots. The machines can achieve better precision, and can perform dangerous or heavy work. It is often a cheap and effective solution for big industrial companies. ILC provides a great tool for performance improving, if a task needs to be done frequently.

For example, imagine a robot arm which needs to take the same path, again and again, with high accuracy. There possibly exists a controller, which might run the robot arm along this route. However, to achieve good precision, a simple controller might not be enough, and more complicated techniques must be applied to impact the result.

The other possibility is to design a simpler controller, and try to "learn" something from the resulting input and output signals. The input signal can than be adjusted to achieve better performance. By doing so run after run there is possible to achieve an almost perfect result with less effort, if the signal is adapted with a proper algorithm.

In this thesis some of such algorithms are presented for the discrete finite time dynamical systems. Their advantages and disadvantages are discussed, the robust stability property is considered. The examples, made using computer simulations, illustrate the practical use of the algorithms. For the application purposes there are also some difficulties discussed and remedies are proposed.

1.2 Problem Statement

There are many different ILC algorithms. One can find a good overview in [1]. The focus of this work is studying the so-called unit memory algorithms on the example of discrete-time dynamical systems over a finite time horizon.

The unit memory algorithms are the ones, which can "remember" the last pass. Mathematically speaking, if u_k is the input signal and e_k is the tracking error at trial k, then

$$u_{k+1} = f(e_k, e_{k+1}, u_k),$$

where f is some function independent on k [5].

The l-input m-output discrete time systems can be written as

$$\begin{array}{rcl} x(t+1) & = & Ax(t) + Bu(t) \\ y(t) & = & Cx(t) + Du(t), \\ x(0) & = & x_0, \end{array}$$

over a time horizon $t = 0, 1, 2, \ldots, N$. $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{l\times n}$, $C \in \mathbb{R}^{m\times n}$, and $D \in \mathbb{R}^{m\times l}$ denote real matrices. We are interested in the minimizing of the error between output $y(\cdot)$ and the tracking signal $r(\cdot)$

$$e(t) = r(t) - y(t),\ t = 0, 1, 2, \ldots, N.$$

An uncommon contemplation in this work is the final time horizon. Together with discreteness of the system it invites us to rewrite the system (**??**) in a more compact form. We "stock" the sequences

$$\{u(0), u(1), u(2), \ldots, u(N)\} \text{ and } \{y(0), y(1), y(2), \ldots, y(N)\}$$

as

$$u = \begin{pmatrix} u(0) \\ u(1) \\ \vdots \\ u(N) \end{pmatrix} \text{ and } y = \begin{pmatrix} y(0) \\ y(1) \\ \vdots \\ y(N) \end{pmatrix}.$$

The resulting vectors u and y are called supervectors.

Then the system (**??**) can be expressed as

$$y = Gu + d,$$

where

$$G = G(A, B, C, D) = \\ = \begin{pmatrix} D & 0 & \cdots & 0 & 0 & 0 \\ CB & D & \cdots & 0 & 0 & 0 \\ CAB & CB & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ CA^{N-2}B & CA^{N-3}B & \dots & CB & D & 0 \\ CA^{N-1}B & CA^{N-2}B & \dots & CAB & CB & D \end{pmatrix} \in \mathbb{R}^{m(N+1)\times l(N+1)}.$$

and

$$d = d(C, A, x_0) = \begin{pmatrix} Cx_0 \\ CAx_0 \\ CA^2x_0 \\ \vdots \\ CA^Nx_0 \end{pmatrix} \in \mathbb{R}^{m(N+1)}.$$

With other words, we put all the information about the system at the times $t = 0, 1, \dots, N$ in one linear equation. This form inspires some of ILC algorithms, for example the Inverse Model Algorithm (Chapter 3), which uses the inverse of the matrix G.

To make the notation less confusing, we denote with $u(t)$ the value of signal $u(\cdot)$ at time t, and with u the corresponding supervector form. With u_k is denoted the supervector in the k-th trial of algorithm run.

Five different algorithms are illustrated in this work. Two of them are derived based on inverse model consideration, two are inspired by steepest descent, and one is based on feedback design. The algorithms in this thesis are taken from [5]. However, some of them were modified in a more compact way.

Furthermore, we consider the robustness property of Unit Memory Algorithms despite the parametric uncertainty. Using Lyapunov techniques and Full-Block S-Procedure we derive a method, which allows us to check whether the given algorithm stays stable if the parameters of the system are uncertain.

Moreover, we discuss the issues appearing by the application of the algorithms. With the supervector description, the ILC algorithms need a lot of computational memory and lead to significant numerical errors. We need to modify the algorithms in a way, so that we can apply them to the real problems.

1.3 Structure

This work consists of 6 chapters. In the first chapter we formulate the problem. In Chapter 2, we recap the crucial basics for understanding the considered algorithms. In Chapter 3 we introduce the supervector notation and inspired by it, derive the ILC algorithms. In chapter 4 we succeed the requirements, under which the algorithms stay robust despite the parametric uncertainty. Finally, in chapter 5 we want to apply ILC algorithms to a real dynamical system. We face the issues the real problems yield, and suggest the remedies. The summary concludes this thesis, and further work is suggested.

Chapter 2

Basics

To understand the following Iterative Learning Control (ILC) algorithms, a few basic definitions are presented in this chapter. Again, the considered system is given via

$$\begin{array}{rcl} x(t+1) &=& Ax(t) + Bu(t) \\ y(t) &=& Cx(t) + Du(t), \\ x(0) &=& x_0. \end{array} \tag{2.1}$$

Here $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times l}$, $C \in \mathbb{R}^{m\times n}$, and $D \in \mathbb{R}^{m\times l}$ are real matrices, $r(t) \in \mathbb{R}^m$ is a reference signal at time t. For notation simplification, a short description for the discrete-time control system (2.1) is denoted with (A, B, C, D).
In the basic tracking issue, there is sought an input $u(\cdot)$, which renders the output $y(\cdot)$ of a system (2.1) to lie close to the tracking signal $r(\cdot)$. That means, that the error $e(\cdot) := y(\cdot) - r(\cdot)$ must stay small in some norm $||\cdot||$, e.g. the L_2-norm. To find it usually a stabilizing controller is designed – such that the system state $x(\cdot)$ goes not to infinity over time t. To design it, the stability needs a clear definition.
The stability definition is usually given asymptotically, i.e. $t \to \infty$. Though in this thesis there are the systems over a finite time horizon of interest, it still makes sense to take this "classical" definition of stability, as it is discussed later.

Definition 1 *A discrete time dynamical system*

$$x(t+1) = Ax(t) \tag{2.2}$$

for $t \geq 0$ is said to be asymptotically stable, if for all initial conditions $x(0) \in \mathbb{R}^n$

$$\lim_{t\to\infty} x(t) = 0 \tag{2.3}$$

is satisfied. It is said to be stable, if

$$\limsup_{t\to\infty} ||x(t)|| < \infty \tag{2.4}$$

for all initial conditions $x(0) \in \mathbb{R}^n$. The matrix A is (asymptotically) stable, if the dynamical system (2.2) is (asymptotically) stable.

Definition 2 *A discrete time system (A, B, C, D) or the pair (A, B) is (asymptotically) stabilizable, if there exists a matrix F, such that $A - BF$ is (asymptotically) stable. The system or the pair (A, C) is detectable, if (A^T, C^T) is assymptotically stabilizable.*

Another useful definition is the definiteness of a matrix. This is given as follows.

Definition 3 *A symmetric matrix $A \in \mathbb{R}^{n\times n}$ is negative definite if $x^T Ax < 0$ for all $x \in \mathbb{R}^n$, $x \neq 0$. It is negative semi-definite, if $x^T Ax \leq 0$ for all $x \in \mathbb{R}^n$. In the same way, the matrix A is positive definite, if $x^T Ax > 0$ for all $x \in \mathbb{R}^n$, $x \neq 0$, and positive semi-definite, if $x^T Ax \geq 0$ for all $x \in \mathbb{R}^n$.*
The symbols $\prec, \preceq, \succ, \succeq$ for two symmetric matrices $A, B \in \mathbb{R}^{n\times n}$ are defined as follows:

$$A \prec B \quad \text{if } A - B \text{ is negative definite,} \tag{2.5a}$$
$$A \preceq B \quad \text{if } A - B \text{ is negative semi-definite,} \tag{2.5b}$$
$$A \succ B \quad \text{if } A - B \text{ is positive definite,} \tag{2.5c}$$
$$A \succeq B \quad \text{if } A - B \text{ is positive semi-definite.} \tag{2.5d}$$

2.1 Supervector description

The supervectors are the most useful tool in this thesis. By "stocking" the signals together, it surrenders a compact system representation in linear form.

Definition 4 *Let $\{s(0), s(1), s(2), \ldots s(N)\} \subset \mathbb{R}^d$ be a finite sequence. Then the supervector corresponding to this sequence is denoted by s and is defined as*

$$s := \begin{pmatrix} s(0) \\ s(1) \\ s(2) \\ \vdots \\ s(N) \end{pmatrix}. \tag{2.6}$$

In this way are defined the supervectors $u \in \mathbb{R}^{l(N+1)}$ and e, y, $r \in \mathbb{R}^{m(N+1)}$. Then for system (2.1) the relation between u, y, and the error e can be written as

$$y = Gu + d, \tag{2.7}$$
$$e = r - y. \tag{2.8}$$

The *supermatrix* G represents the state-space model and is given via

$$G = G(A,B,C,D) = \\ = \begin{pmatrix} D & 0 & \cdots & 0 & 0 & 0 \\ CB & D & \cdots & 0 & 0 & 0 \\ CAB & CB & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ CA^{N-2}B & CA^{N-3}B & \dots & CB & D & 0 \\ CA^{N-1}B & CA^{N-2}B & \dots & CAB & CB & D \end{pmatrix} \in \mathbb{R}^{m(N+1)\times l(N+1)}. \tag{2.9}$$

The vector d depends on the initial condition x_0:

$$d = d(C,A,x_0) = \begin{pmatrix} Cx_0 \\ CAx_0 \\ CA^2x_0 \\ \vdots \\ CA^Nx_0 \end{pmatrix} \in \mathbb{R}^{m(N+1)}. \tag{2.10}$$

The stability requirement for A is highlighted here. As for the system (2.2) holds

$$x(t) = A^{t-1}x(0) \text{ for } t \geq 1, \tag{2.11}$$

the matrix A^t gets very large terms for the growing time t. In opposite, for a stable matrix A, A^t gets small terms for $t \gg 1$[1]. Intuitive, those terms can be neglected for all $p > p^\star$ for some $1 < p^\star < N$. This allows to reduce the amount of computer memory needed, if use the sparse matrices. In Chapter 5 an estimation based on neglecting of small terms is derived.

2.2 Separation Principle

There are several ways to construct a stabilizing controller for a system (A,B,C,D). For example the $\mathcal{H}_\infty$ design [2], pole placement [7] or the separation principle. The latest is presented here. Though it might render worse performance than the more advanced techniques, it offers a simple implementation. The tracking accuracy of the initial stabilizing controller (that is, implemented before using the ILC algorithms), is here not a deciding factor. The tracking accuracy will be improved later by using the ILC techniques.

First consider the case $y(\cdot) = x(\cdot)$. Then the system (2.1) becomes stable, if $u(t) = -Fx(t)$, $t \geq 0$, with some matrix F, such that $(A - BF)$ is stable. Clearly, if the

[1]The symbol $\gg$ means much greater

system is (asymptotically) stabilizable, there exists such a matrix F.

But what is a "good" choice for F? One option is the so-called Linear-Quadratic-Regulator (LQR), which minimizes the cost function

$$V(u(\cdot)) = \sum_{t=0}^{\infty} \left((x(t))^T Q(x(t)) + u(t)^T R u(t) \right), \tag{2.12}$$

overall stabilizing piecewise constant input signals $u(\cdot) \in \mathbb{R}^m$, and with $x(\cdot)$ satisfying (2.1) (for $C = I$ and $D = 0$). The matrices $Q \in \mathbb{R}^{n \times n}$ and $R \in \mathbb{R}^{m \times m}$ are positive definite weighing matrices, which allow to decide how "important" the elements of the in- and output signals are. By using them, one input signal can be made to impact the system more than another or drive one of the output signals faster. Choosing the weighting matrices always depends on the considered system and allows to affect the closed-loop behaviour.

The optimal $u(\cdot)$ for minimizing the cost function (2.12) is given via $u(t) = Fx(t)$ with

$$F = (R + B^T P B)^{-1} B^T P A \text{ [3].} \tag{2.13}$$

The matrix P is a unique symmetric positive definite matrix, the stabilizing solution of the discrete time algebraic Riccati equation

$$A^T A - A^T P B (R + B^T P B)^{-1} B^T P A + Q - P = 0 \text{ [9].} \tag{2.14}$$

Stabilizing solution means, that the matrix $(A - BF)$ resulting from (2.13) is stable, and hence this solution stabilizes the system.

For the case $y(\cdot) \neq x(\cdot)$, an additional observer can be implemented to approximate the signal $x(\cdot)$. Then the resulting controller is said to be based on *separation principle*. It requires the pair (A, C) to be detectable, i.e. there exists a matrix J, for which the matrix $A - JC$ is asymptotically stable.

Set

$$u(\cdot) = -F\hat{x}(\cdot), \text{ with } \hat{x}(\cdot) \text{ given via the dynamical system} \tag{2.15}$$

$$\begin{aligned} \hat{x}(t+1) &= A\hat{x}(t) + Bu(t) + J(y - \hat{y}(t)), \\ \hat{y}(t) &= C\hat{x}(t) + Du(t), \\ \hat{x}(0) &= x_0, \ t \geq 0. \end{aligned} \tag{2.16}$$

With this observer system, the estimation error can be calculated as

$$\tilde{x}(t+1) = x(t+1) - \hat{x}(t+1) = (A - JC)\tilde{x}(t), \ t \geq 0, \tag{2.17}$$

and as $(A - JC)$ is asymptotic stable, the error $\tilde{x}(t+1)$ asymptotically converges to zero.
For a constant reference signal $r(t) = r_0 \in \mathbb{R}^m$, $t \geq 0$, choose the input signal as

$$u(t) = -F\hat{x}(t) + Mr_0,\ t \geq 0, \tag{2.18}$$

with some matrix M of a fitting dimension. Then the dynamical system

$$\begin{pmatrix} x(t+1) \\ \tilde{x}(t+1) \end{pmatrix} = \begin{pmatrix} A - BF & BF \\ 0 & A - J \end{pmatrix} \begin{pmatrix} x(t) \\ \tilde{x}(t) \end{pmatrix} + \begin{pmatrix} BM \\ 0 \end{pmatrix} r(t),\ t \geq 0 \tag{2.19}$$

is asymptotically stable. With straightforward calculation, the steady state results in

$$\lim_{t\to\infty} \begin{pmatrix} x(t) \\ \tilde{x}(t) \end{pmatrix} = \begin{pmatrix} (I - A + BF)^{-1}BM \\ 0 \end{pmatrix} r_0. \tag{2.20}$$

Choose M as a right inverse of the matrix

$$D + (C - DF)(I - A + BF)^{-1}B, \tag{2.21}$$

then

$$\lim_{t\to\infty} y(t) = \begin{pmatrix} C - DF & DF \end{pmatrix} \begin{pmatrix} (I - A + BF)BM \\ 0 \end{pmatrix} r_0 + DMr_0 = r_0. \tag{2.22}$$

This controller design can also be used for the piecewise constant signals $r(\cdot)$, if the tracking signal does not change too fast.

Chapter 3

ILC Algorithms

Imagine, that some action needs to be performed multiple times. For example, a robot manipulator must put objects in a box with high accuracy, while the objects to put are always located in the same place. If an input sequence for issue solving already exists, it can be used for the next iteration and delivers the same precision. The other possibility is to "learn" from the previous iteration and try to enhance the exactness, while the tracking objective $r(\cdot)$ stays the same over all iterations.

We assume the system (**??**) to be stable. Moreover, for our purposes, for stabilizable systems we can assume it without loss of generality. Let $(\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D})$ be the stabilized closed loop system (with $r(\cdot)$ as input and $e(\cdot)$ as output). We define a new intput $\tilde{u}(\cdot) := u(\cdot) + v(\cdot)$. Then the system $(\tilde{A}, B, \tilde{C}, D)$ is stable, and with a new input signal $v(\cdot)$ is still has the room for improvement.

Example 5 *Let us consider the system (A, B, C, D) with*

$$A = \begin{pmatrix} 2 & 1 \\ 4 & 3 \end{pmatrix}, B = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, C = \begin{pmatrix} 0 & 1 \end{pmatrix}, D = 2. \tag{3.1}$$

This system is asymptotically stabilizable and detectable. We choose the weights $Q = 3$ and $R = 2$ and design an LQR controller with

$$F = \begin{pmatrix} 1.9674 & 1.3042 \end{pmatrix} \text{ and } J = \begin{pmatrix} 1.8705 & 4.7321 \end{pmatrix}. \tag{3.2}$$

We modify the system to

$$\begin{array}{rcl} x(t+1) & = & \begin{pmatrix} 0.0326 & -0.3042 \\ 0.0652 & 0.3916 \end{pmatrix} x(t) + \begin{pmatrix} 1 \\ 2 \end{pmatrix} v(t) \\ y(t) & = & \begin{pmatrix} -3.9348 & -1.6084 \end{pmatrix} x(t) + 2v(t), \qquad t = 0, 1, 2, \ldots, N \end{array} \tag{3.3}$$

with an addition input signal $v(\cdot)$.

3.1 Inverse Model Algorithms

Let us go back to the equation (2.7). The tracking problem here seems to be straightforward. If we know the reference signal r, we can find a perfect solution by choosing

$$u_\infty = G^{-1}r - d. \tag{3.4}$$

However, this choice does not look greatly credible. For example, if our measurements are disturbed by noise, we get the output signal

$$y + \varepsilon w = Gu_\infty + d + \varepsilon w = r + \varepsilon w \neq r, \tag{3.5}$$

for some non-zero vector $w \in \mathbb{R}^{m(N+1)}$ and a small $\varepsilon > 0$.
Also for uncertain models this method might not be applicable. If we could reformulate this idea in a more robust way, we may get a good and simple applicable algorithm, which we call the *Inverse Model Algorithm.*

Algorithm 6 *Let the matrix D in* (**??**) *have an inverse. Then the matrix $G = G(A, B, C, D)$ is non-singular as well, and the Inverse Model Algorithm (IA) is given via input update law*

$$\begin{aligned} u_{k+1} &= u_k + \beta G^{-1} e_k, \\ u_0 &\in \mathbb{R}^{l(N+1)}, \end{aligned} \tag{3.6}$$

The error evolution follows

$$\begin{aligned} e_{k+1} &= (1-\beta)e_k,\ k \geq 0, \\ e_0 &= r - Gu_0 - d. \end{aligned} \tag{3.7}$$

In particular, $(e_k)_{k\geq 0}$ converges to zero for $k \to \infty$ for any initial error $e_0 \in \mathbb{R}^{m(N+1)}$ if and only if

$$0 < \beta < 2.$$

Proof:
Firstly, the matrix G has lower triangular structure, with the matrix D on its diagonal. Hence, $\det(G) = \det(D)^{N+1}$, which is non-zero if and only if D has full rank. Taking the limit over relation (3.7) results in

$$\lim_{k\to\infty} e_{k+1} = \lim_{k\to\infty} (1-\beta)e_k = \lim_{k\to\infty} (1-\beta)^k e_0. \tag{$*1$}$$

Choosing $0 < \beta < 2$ yields the proof. ■

For β close to 0 or 2, we get slower convergence and better robustness. For $\beta = 1$ we get convergence in one iteration, but this choice might be non-robust [5], pp 149, 152-155.
We apply this algorithm on the system from Example 5 to see the algorithm performance.

Example 7 *In the system* (3.1) *the matrix* $D = 2$ *is invertible, and we can apply IA. We choose as* u_0 *the input sequence of LQR controller, and* $\beta = 0.1$. *Convergence to 0 with tolerance* 10^{-8} *is achieved after 74 iterations.The result is illustrated in Figure 3.1.*

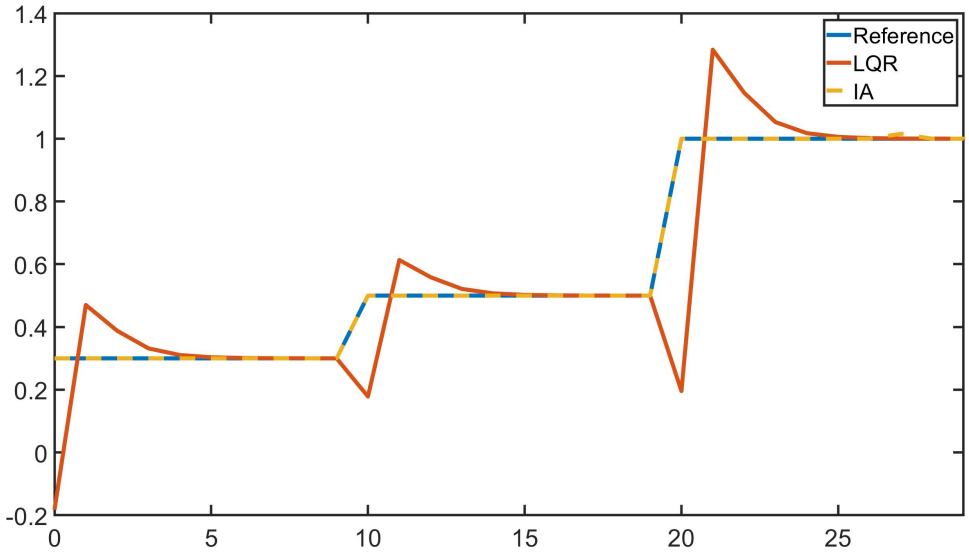


Figure 3.1: Tracking with LQR Controller (without ILC algorithm), and tracking with IA for the system (3.1) for $N = 30$

But if we try to increase the length of the time horizon N, for example to 50 steps, we can see a wrong system behavior (Figure 3.2).
The reason is the condition number of the matrix G, which equals $1.353\,10^{16}$. *We see: even for a triangular matrix with non-zero eigenvalues, the numerically disturbances can lead to an unexpected algorithm behaviour.*

The Inverse Model Algorithm sets up very strong requirements: the matrix G should be non-singular, and its condition number must be small enough that we can compute a numerically stable inverse. We can modify this algorithm by taking the Moore-Penrose inverse. This is the so-called pseudo inverse, which is uniquely defined and can be computed for all matrices [8].

Definition 8 *For a matrix* $G \in \mathbb{R}^{m(N+1)\times l(N+1)}$ *a psedo inverse is defined by a matrix* $G^+ \in \mathbb{R}^{l(N+1)\times m(N+1)}$, *satisfying the criteria:*

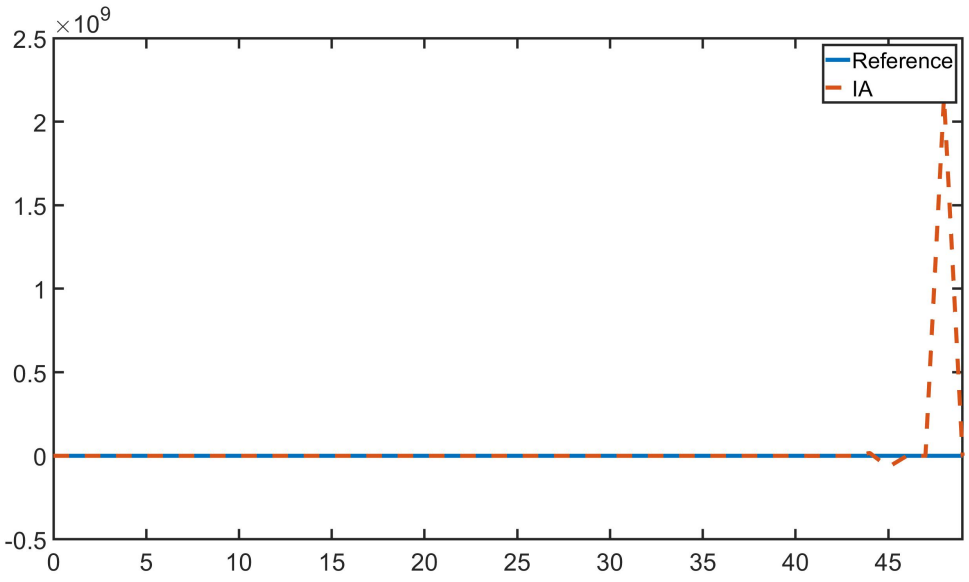


Figure 3.2: Tracking with IA for the system (3.1) for $N = 50$

(a) $GG^+G = G$,

(b) $G^+GG^+ = G^+$,

(c) $(GG^+)^T = GG^+$,

(d) $(G^+G)^T = G^+G$.

Soll man in der Liste oben die Kommas stellen? Und wenn nicht, dann ist es okay, dass der Satz keinen Punkt hat?

Algorithm 9 *The Pseudo Inverse Model Algorithm (PIA) is given via input update law*

$$\begin{aligned} u_{k+1} &= u_k + \beta G^+ e_k, \\ u_0 &\in \mathbb{R}^{l(N+1)}, \end{aligned} \tag{3.8}$$

with the error evolution

$$\begin{aligned} e_{k+1} &= (1 - \beta GG^+)e_k, \ k \geq 0, \\ e_0 &= r - Gu_0 - d. \end{aligned} \tag{3.9}$$

Monotonic convergence to

$$e_\infty = \lim_{k\to\infty} e_k = P_{\ker[G^+]} e_0, \tag{3.10}$$

is guaranteed if

$$0 < \beta < 2.$$

$P_{\ker[G^+]}$ denotes the positive orthogonal projection operator onto $\ker[G^+]$*. In particular, the zero convergence is attainable, if and only if the tracking signal r is feasible. That is, there exists an input $\hat{u}$, such that $r = G\hat{u}$.*

Proof:

Because of the pseudo inverse definition, it holds $(GG^+)^2 = GG^+GG^+ = GG^+$. Then the error evolution formula can be proven by mathematical induction over k.
For $k = 1$ it follows:

$$e_1 = (I - \beta GG^+)e_0 = (I + GG^+ - GG^+ - \beta GG^+)e_0 = (I + [(1-\beta) + 1]\, GG^+)e_0.$$

For $k \in \mathbb{N}$

$$\begin{aligned} e_{k+1} &= (I - \beta GG^+)e_k = (I - \beta GG^+)(I + \left[(1-\beta)^{k-1} - 1\right] GG^+)e_0 = \\ &= \left(I - \beta GG^+ + (1-\beta)^{k-1}GG^+ - GG^+ - \beta GG^+ \left[(1-\beta)^{k-1} - 1\right] GG^+\right) e_0 = \\ &= \left(I - (\beta - (1-\beta)^{k-1} + \beta \left[(1-\beta)^{k-1} - 1\right])GG^+\right) e_0 \\ &= \left(I - \left(\beta - (1-\beta)^{k-1} + 1 + \beta(1-\beta)^{k-1} - \beta\right) GG^+\right) e_0 \\ &= (I + \left[(1-\beta)^k - 1\right] GG^+)e_0 \end{aligned}$$

To prove (3.10) recall that

$$\mathbb{R}^{m(N+1)} = \mathrm{im}(G) \oplus \ker(G^+),$$

The symbol $\oplus$ denotes here the direct sum of two vector spaces. Hence, e_0 can be written as

$$e_0 = Gw_0 + v_0,$$

where $v_0 = (I - GG^+)e_0 \in \ker(G^+)$ is uniquely defined and $w_0 = G^+e_0 \in \mathbb{R}^{l(N+1)}$. It follows:

$$e_k = (1 + \left[(1-\beta)^{k-1} - 1\right] GG^+)e_0 = (1-\beta)^{k-1}Gw_0 + v_0.$$

For $\beta \in (0, 2)$ the error e_k converges to v_0 for $k \to \infty$. ■

Example 10 *We calculate a solution for* (3.1) *with PIA, and illustrate it in Figure 3.3. This time the tracking signal and system output fit well despite the large condition number. However, a deviation can be observed at the beginning, which causes the bad invertability of the matrix G.*

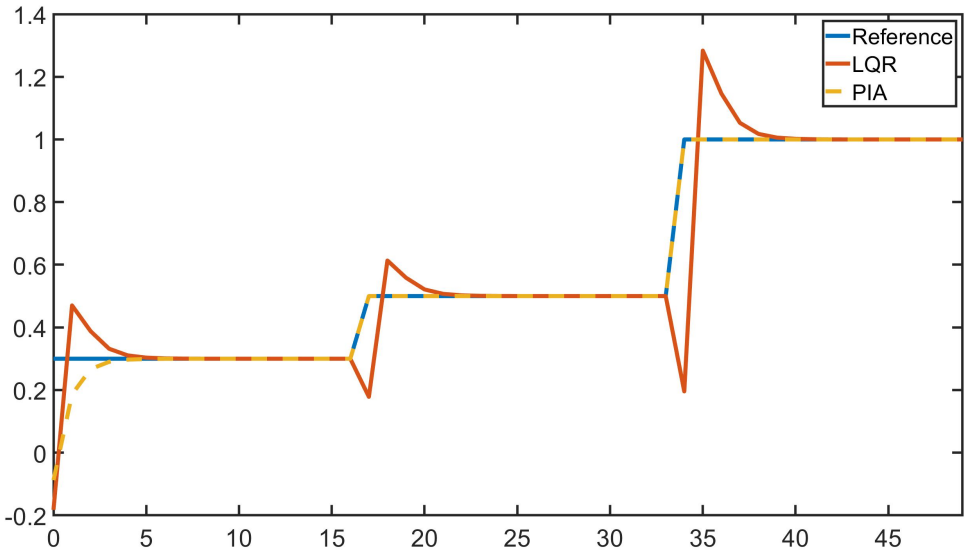


Figure 3.3: Reference signal tracking with LQR and PIA for the sysytem (3.1) for $N = 50$

Owens in his book [5] chooses the left and right inverse matrix instead of the pseudo inverse. The pseudo inverse has its advantage to equal the right/left inverse, if it exists. Moreover, for right/left inverse we need to restrict the rank requirements, as it exists not for every matrix. The algorithm in this formulation can be applied for any matrix G.

The Algorithm 6 is a special case of PIA. However, numerically the use of the inverse matrix might be more desirable as it requires less calculation effort, if the condition number is not large.

The (Pseudo) Inverse Algorithm belongs to the class of so-called Unit Memory Algorithms. These are the algorithms we assume to "remember" the last trial.

3.2 Unit Memory Algorithms

We are interested in the processes, which are executed repetitively. Let $k = 0, 1, 2, \ldots$ be the number of completed iterations. Then we get a system

$$\begin{aligned} y_k &= Gu_k + d, \\ e_k &= r - y_k, \end{aligned} \tag{3.11}$$

$u_0 \in \mathbb{R}^{l(N+1)}$, $k = 0, 1, 2, \ldots$.

To achieve a good tracking at trial k we need to ensure that the norm $||e_k||$ is small. The tracking behavior improves for consecutive iterations if the sequence $||e_k||$ is monotonically decreasing, i.e.,

$$||e_{k+1}|| < ||e_k|| \text{ for all } k \geq 0. \tag{3.12}$$

Note that in this case the sequence $||e_k||$ is guaranteed to converge. If the sequence even converges to zero we say that perfect tracking is achieved asymptotically.

We have already seen, that it is not always possible to achieve zero convergence. In Algorithm 9 the perfect tracking is possible, if the reference signal r is feasible – that is, there exists an input signal u_∞, such that $r = Gu_\infty + d$. The convergence properties can also depend on the choice u_0 [5].

We assume linear dependency of u_{k+1} upon u_k and e_k for $k \geq 0$, and define the following algorithm using feedback control.

Algorithm 11 *The Unit Memory Algorithm is given via the input update law*

$$\begin{aligned} u_{k+1} &= u_k + Ke_k, \\ u_0 &\in \mathbb{R}^{l(N+1)}. \end{aligned} \tag{3.13}$$

The error dynamic is then given via

$$e_{k+1} = (I - GK)e_k,\ k \geq 0, \tag{3.14}$$

$$e_0 = r - Gu_0 - d. \tag{3.15}$$

$K \in \mathbb{R}^{l(N+1)\times m(N+1)}$ *is called* ***learning matrix****.*

To ensure the stability of the algorithm, it is enough to consider the iteration process

$$e_{k+1} = (I - GK)e_k = (I - GK)^k e_0 := L^k e_0,\ k \geq 0. \tag{3.16}$$

This is a discrete time dynamical system over k. We can reformulate our goal as follows: find some matrix $K \in \mathbb{R}^{l(N+1)\times m(N+1)}$, which renders (3.16) stable.

If we rewrite our system as

$$\begin{pmatrix} e_{k+1} \\ e_k \end{pmatrix} = \left(\begin{array}{c|c} I & -G \\ \hline I & 0 \end{array}\right) \begin{pmatrix} e_k \\ v_k \end{pmatrix}, \tag{3.17}$$

our problem reduces to finding a stabilizing controller K with

$$v_k = Ke_k. \tag{3.18}$$

We know how to work with such systems – for example, the LQR controller provides a good solution. This is a reliable method, but since the matrix G can have a large dimension, it also can be costly. Often good results are possible to achieve with much more simple methods. For example, we can reduce the number of parameters to be chosen.

In the Algorithms 6 and 9 we chose

$$K = \beta K_0 \tag{3.19}$$

with a fixed matrix K_0 as the (pseudo) inverse of G, and left the parameter β to decide on. So, we reduced the number of degrees of freedom to 1, and still get satisfying results.

However, as we have seen in Example 7, the calculation of the inverse matrix is not the most reliable method. The pseudo inverse delivers better results, but is also much more expensive. An algorithm, which does not require the model inversion, might be of use here.

3.3 Gradient Algorithms

We want a new algorithm to render monotonic convergence. That means, the relation (3.12) must hold in some norm $||\cdot||$ in $\mathbb{R}^{m(N+1)}$.

We choose the norm by defining the weighting matrices $Q(t)$ and $R(t)$, $t = 0, 1, 2, \ldots, N$. Let the norms $||\cdot||_Q$, $||\cdot||_R$ to be defined associated with scalar products

$$\langle y, z\rangle_Q = \sum_{t=0}^{N} y(t)^T Q(t) z(t),\ \langle u, v\rangle_R = \sum_{t=0}^{N} u(t)^T R(t) v(t), \tag{3.20}$$

with $y(t), z(t) \in \mathbb{R}^m$, $u(t), v(t) \in \mathbb{R}^l$ for $t = 0, 1, 2, \ldots, N$. Furthermore, we define the matrix

$$G^\star = R^{-1} G^T Q \text{ for all } G \in \mathbb{R}^{m(N+1)\times l(N+1)},$$

where

$$Q := \operatorname{diag}(Q(0), Q(1), Q(2), \ldots, Q(N)) \text{ and } R := \operatorname{diag}(R(0), R(1), R(2), \ldots, R(N)). \tag{3.21}$$

Recall, that for $R = I$ and $Q = I$ the matrices $G^\star$ and $K^\star$ are just the conjugate transpose. The notation from above can be seen as conjugate transpose in the vector space with scalar products (3.20).

Applying the weighted norms on (3.12) provides

$$\begin{aligned} ||e_{k+1}||_Q^2 &= ||e_k - GKe_k||_Q^2 = ||e_k||_Q - 2\langle e_k, GKe_k\rangle_Q + ||GKe_k||_Q^2 \\ &< ||e_k||_Q^2, \text{ if } ||GKe_k||_Q^2 < 2\langle e_k, GKe_k\rangle_Q. \end{aligned} \tag{3.22}$$

The last inequality is equivalent to

$$e_k^T K^T G^T Q G K e_k < 2 e_k^T Q G K e_k. \tag{3.23}$$

If we set $K = \beta G^\star$ this is implicated by

$$\beta^2 (G G^\star)^T Q (G G^\star) \prec \beta Q G G^\star + \beta (Q G G^\star)^T. \tag{3.24}$$

Since the matrix Q is positive definite, we can find a positive definite matrix $Q^{1/2}$, such that $Q = Q^{1/2} Q^{1/2}$.
We write $Q^{-1/2}$ for $\left(Q^{1/2}\right)^{-1}$, and define a matrix

$$H := Q^{1/2} G G^\star Q^{-1/2}. \tag{3.25}$$

The matrix H has the same eigenvalues as $GG^\star$, as for its characteristic polynomial holds

$$\begin{aligned} \det(I - \lambda H) &= \det\left(I - \lambda Q^{1/2} G G^\star Q^{-1/2}\right) \\ &= \det\left(Q^{1/2}\right) \det\left(I - \lambda G G^\star\right) \det\left(Q^{1/2}\right)^{-1} \\ &= \det\left(I - G G^\star\right) \text{ for any } \lambda \in \mathbb{R}. \end{aligned} \tag{3.26}$$

We rewrite (3.24) as

$$\beta^2 H^T H \prec \beta (H + H^T). \tag{3.27}$$

This relation is fulfilled if

$$0 < \beta < \frac{2}{\lambda_{\max}(H)}. \tag{3.28}$$

This deliberation inspires the *Steepest Descent Algorithm.*

3.3.1 Steepest Descent Algorithm

Algorithm 12 *The Steepest Descent Algorithm is characterized by choosing $K = \beta G^\star$, where $\beta > 0$ is a real scalar gain. The input update law is given via*

$$\begin{aligned} u_{k+1} &= u_k + \beta G^\star e_k, \\ u_0 &\in \mathbb{R}^{l(N+1)}, \end{aligned} \tag{3.29}$$

and error evolution results in

$$e_{k+1} = (I - \beta G G^\star) e_k, \; k \geq 0. \tag{3.30}$$

Monotonic convergence to

$$e_\infty = \lim_{k\to\infty} e_k = P_{\ker[G^\star]} e_0, \tag{3.31}$$

is guaranteed if

$$0 < \beta < \frac{2}{\lambda_{\max}(H)}.$$

Convergence to zero is assured in the case that $e_0 \in \mathrm{im}[GG^\star]$ *holds.*

Proof:

To prove the statement remember that, as $GG^\star$ is a symmetric matrix, we can find unique vectors $e_{\mathrm{im}} \in \mathrm{im}(GG^\star)$, $e_{\ker} \in \ker(GG^\star)$, such that

$$e_0 = e_{\mathrm{im}} + e_{\ker}. \tag{$*1$}$$

Substituting e_0 in error evolution by this expression we get

$$e_{k+1} = (I - \beta GG^\star)e_k = (I - \beta GG^\star)^k e_{\mathrm{im}} + e_{\ker}. \tag{$*2$}$$

For convergence to $e_{\ker}$ it is enough to show that $\lambda_{\max}(I - \beta GG^\star) < 1$. As $(I - \beta GG^\star)$ is a symmetric matrix, the convergence is guaranteed if and only if all the eigenvalues of $(I - \beta GG^\star)$ are less than 1 in their absolute value.
If $\lambda_{\max}(I - \beta GG^\star) = 0$, then $(I - \beta GG^\star) = 0$ and convergence follows trivially.
Let $0 \neq \mu \in \mathbb{C}$ be an eigenvalue of the matrix $(I - \beta GG^\star)$. Then there exists some $0 \neq v \in \mathbb{R}^{l(N+1)}$, such that

$$(I - \beta GG^\star)v = \mu v \Rightarrow GG^\star v = \frac{1-\mu}{\beta} v. \tag{$*3$}$$

Because of our choice of β, the inequality (3.24) holds and implies

$$\beta^2 v^T (GG^\star)^T Q (GG^\star) v < 2\beta v^T (QGG^\star) v \Rightarrow (1-\mu)^2 ||v||_Q < 2(1-\mu)||v||_Q. \tag{$*4$}$$

This is the case if and only if the absolute value of $(1-\mu)$ is less than 2. This concludes $|\mu| < 1$ and proves the convergence. ■

An interesting observation is, that provided by this algorithm signal u_∞ is the unique solution of the (minimum norm) optimization problem

$$u_\infty = \arg \min_{u \in \mathbb{R}^{l(N+1)}} \{||u - u_0||_R^2 : \text{ subject to } r = Gu + d\}. \tag{3.32}$$

One consequence is that the choice of u_0 has more significance than the simple intuition.

A good choice will influence convergence rates beneficially. More generally, the limit u_∞ is the closest input to u_0 in the norm, and since choosing $u_0 = 0$ leads to the minimum energy solution u_∞ [5] p. 168.

Choice of the Weighting Matrices

The choice of the weighting matrices Q and R provides us additional degrees of freedom. Using them we can impact the behavior of the algorithm. For example, with the choice of the matrix Q we can weight some elements of input and output signals, depending on their importance in measuring accuracies and convergence rates, or accent the relative importance of different time intervals in measuring tracking accuracy and required convergence rates. For rapid convergence in the initial parts of the time interval, we can set $Q(t) = \varepsilon^{2t}\tilde{Q}$ with some time independent $\tilde{Q}$ and $\varepsilon \in (0, 1)$.
The choice of $R(\cdot)$ may arise out of the real need to converge to a minimum input energy solution. Again, using the ε-weighed $R(t) = \varepsilon^{2t}\tilde{R}$ with some fixed $\tilde{R}$, we can accent the input signals at some times $0 \leq t_1 \leq t \leq t_2 \leq N$. This can be used if we want to limit the control action in some time intervals, or if we need to reflect the physical units used.

Example 13 *In the Example 10 we could not achieve perfect tracking at the first trials. We can improve it by using the SDA, if we choose an adequate weight Q.*
With choice

$$Q = \operatorname{diag}\begin{pmatrix} 10^{-2} & 1 & 1 & 1 & 10^{-2} & \dots & 10^{-2} \end{pmatrix}. \tag{3.33}$$

we speed up the convergence at the second, third and fourth time step.
The result is illustrated in Figure 3.4.
By choosing the weight R we can regulate the control action. As we limit the possible input signals, it also leads to a rapid convergence. For example, for $R = I$ we need 8124 iterations, while with

$$R = \operatorname{diag}\begin{pmatrix} 1 & 1 & 1 & 1 & 0.1 & \dots & 0.1 \end{pmatrix} \tag{3.34}$$

only 2569 trials are required.

3.3.2 Suppression of eigenvalues

In the previous algorithms we assumed the learning matrix K to be constant for all iterations. We can presume another controller structure and set variable K_k for $k \geq 0$. The simplest modification way is to set an iteration varying the gain β for each step: $K_k = \beta_k K_0$, $k \geq 0$ for a fixed matrix K_0.

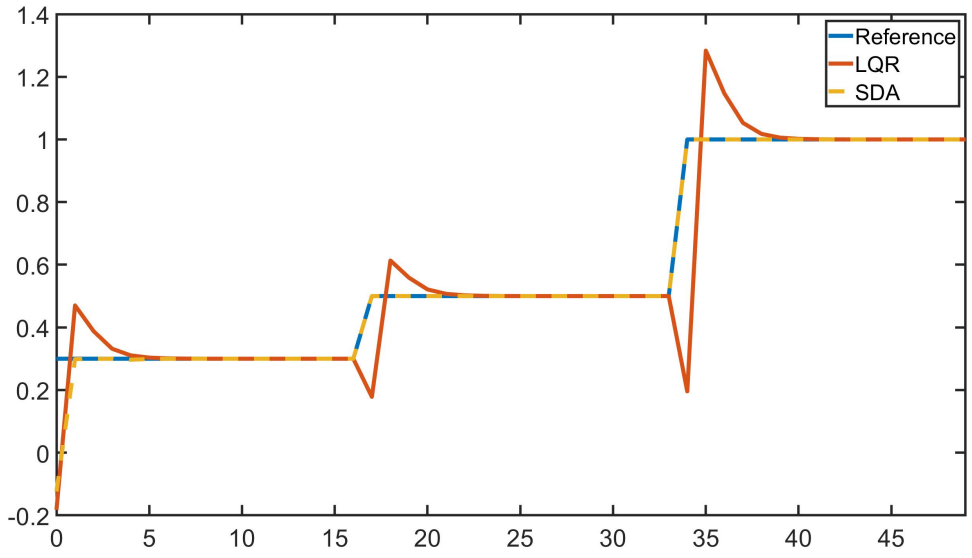


Figure 3.4: Reference signal tracking with LQR and SDA for the system (3.1)

Let us consider the eigenvalues of the matrix $GG^\star$ and choose the gains β_k, $k \geq 0$ according to them.

Theorem 14 *Assume, that the matrix $GG^\star$ has $q+1$ non-zero ordered eigenvalues $\lambda_0, \lambda_1, \ldots, \lambda_q$. Set $\beta_k = \frac{1}{\lambda_k}$ for $k = 0, 1, \ldots, q$ and consider the update law*

$$\begin{aligned} u_{k+1} &= u_k + \beta_k G^\star e_k, \ k \geq 0 \\ u_0 &\in \mathbb{R}^{l(N+1)}. \end{aligned} \tag{3.35}$$

with error evolution

$$e_{k+1} = (I - \beta_{k+1} GG^\star) e_k = \prod_{l=0}^{k+1} (I - \beta_l GG^\star) e_0, \ k \geq 0. \tag{3.36}$$

Then the error sequence $(e_k)_{k\geq 0}$ converges in a finite number of iterations.

Proof:
Set $\tilde{N} := m(N+1) - 1$. With the spectral theorem, there exists a basis of the eigenvectors $\{v_0, v_1, \ldots, v_{\tilde{N}}\}$ with corresponding eigenvalues $\lambda_0, \lambda_1, \ldots, \lambda_q, 0, \ldots 0$. Then we can write e_0 as

$$e_0 = \sum_{p=0}^{\tilde{N}} \gamma_p v_p \tag{$*1$}$$

with uniquely defined $\gamma_p \in \mathbb{R}$, $p = 0, 1, \ldots, \tilde{N}$.

For $k \in \mathbb{N}$ it follows

$$e_k = \left(\prod_{j=0}^{k} (I - \beta_j G G^\star) \right) e_0 = \sum_{p=0}^{\tilde{N}} \gamma_p \left(\prod_{j=0}^{k} (I - \beta_j \lambda_p I) \right) v_p. \tag{$*2$}$$

For the first iteration the error becomes

$$e_1 = \sum_{p=0}^{\tilde{N}} \gamma_p (I - \beta_1 \lambda_p I) v_p = \sum_{p=1}^{\tilde{N}} \gamma_p (I - \beta_1 \lambda_p I) v_p. \tag{$*3$}$$

The component v_0 is eliminated from the error. Assuming, that for $k \geq 1$ the first k components were eliminated, we get

$$e_{k+1} = \sum_{p=0}^{\tilde{N}} \gamma_p \left(\prod_{j=0}^{k} (I - \beta_j \lambda_p I) \right) v_p = \sum_{p=k+1}^{\tilde{N}} \gamma_p \left(\prod_{j=0}^{k} (I - \beta_j \lambda_p I) \right), \tag{$*4$}$$

and hence the first $k+1$ components $v_0, v_1, \ldots, v_k$ are eliminated. By induction, the iteration process terminates after, at most, $m(N+1) - q - 1$ iterations, as all non-zero eigenvalues have been covered and hence all corresponding eigenvectors are eliminated. ■

Although this algorithm is conceptually interesting, it is not suitable for real-world problems. One can see, that the small non-zero eigenvalues of $GG^\star$ will lead to very large values of β_k and hence we get extremely large transient variations in the error norm.

Also computing the eigenvalues can be numerically difficult or deliver not precise results. Moreover, to achieve the monotonic convergence we need to consider only the eigenvalues $\lambda > \frac{1}{2}\lambda_{\max}(H)$. If our model is inaccurate, and some eigenvalues are uncertain, it has a direct impact on the algorithm result.

That all makes this algorithm not solid. Still, we can use the idea to apply the eigenvalues of $GG^\star$ – but not directly. We can try to "pick" the compatible eigenvalues.

Algorithm 15 *Choose a finite number $N_p + 1$ of points $p_0, p_1, \ldots$ spread over the half-open interval $(\frac{1}{2}\lambda_{\max}(H), \lambda_{\max}(H)]$. The Gradient Algorithm with Suppression of Eigenvalues (SoE Algorithm) is defined via choosing of the iteration-depending control law $K_k = \beta_k G^\star e_k$, where*

$$\begin{aligned} \beta_k &= \frac{1}{p_k} \text{ for } k = 0, 1, \ldots, N_p, \\ \beta_k &= \beta \text{ for } k > N_p, \end{aligned} \tag{3.37}$$

with $\beta \in (0, \frac{2}{\sigma^2_{\max}})$.

Then the input update law becomes

$$u_{k+1} = u_k + \beta_k G^\star e_k,\ k \geq 0, \tag{3.38}$$

$$u_0 \in \mathbb{R}^{l(N+1)}, \tag{3.39}$$

and error evolution results in

$$\begin{aligned} e_k &= \left[\prod_{l=0}^{k}(I - \beta_l G G^\star)\right] e_0,\ k = 0, 1, \ldots, N_p, \\ e_k &= (I - \beta G G^\star)^{k-N_p} e_{N_p},\ k > N_p. \end{aligned} \tag{3.40}$$

This algorithm has the same convergence properties as Algorithm 12, but potentially better convergence rates due to the eigenvalue suppression. Intuitively, the approach will increase the convergence speed in the first $N_p + 1$ iterations, if N_p is large enough for good approximation of the interval $(\frac{1}{2}\lambda_{\max}(H), \lambda_{\max}(H)]$.

Example 16 *We apply the SoE Algorithm on the system* (3.1) *with weights* $Q = I$, $R = I$. *The error evolution we get is illustrated in Figure 3.5.*
The new algorithm needs only 121 iterations, while for the not adjusted SDA 730 trials are necessary. Indeed, for 31 of 51 eigenvalues λ *of* $GG^\star$ *holds*

$$\lambda > \frac{1}{2}\lambda_{\max}(H) = 461.5\,. \tag{3.41}$$

We can also see the influence of the weighting matrices on the convergence speed. Since we do not need the better performance at the first time steps, we also need much less iterations.

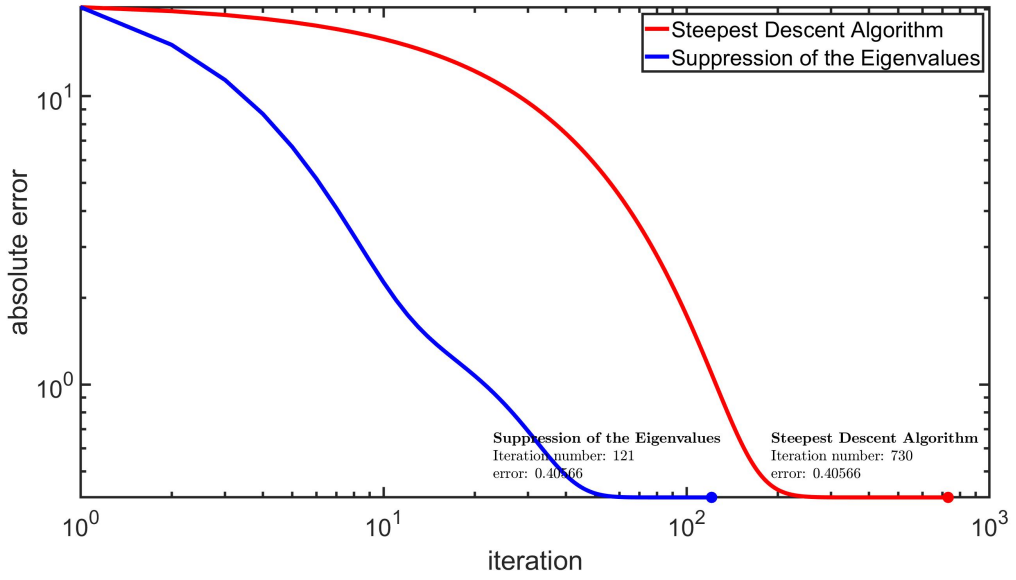


Figure 3.5: Error evolution for SDA and SoE for system (3.1)

3.4 Feedback Design

With the previous algorithms we can achieve a better tracking for repetitively executed processes. But they do not accent the special features of the model. For example, we can not manage the non-minimum-phase zeros, which can impact the convergence rate [5]. If we consider the plant structure and construct a feedback controller, and then convert it into a steepest descent-like algorithm, we possibly can achieve better performance and robustness properties.
Let us denote with $G(z)$ the transfer matrix in variable z of the system (A, B, C, D), and with G the supervector matrix.
More precisely, we consider a forward path compensator $K_c(z)$ in a unity negative feedback system for (**??**). The design criteria for $K_c(z)$ include the closed-loop stability and the ability to track, albeit approximately, the reference signal r. With this compensator, depending on the model, we can remedy the plant properties such as oscillation, loop interaction, or the effects of non-minimum-phase zeros.
We denote the complementary sensitivity of the resulting closed loop with $T(z)$ and its sensitivity with $S(z)$:

$$T(z) = (I + G(z)K_c(z))^{-1}G(z)K_c(z) \text{ and } S(z) = I - T(z) = (I + G(z)K_c(z))^{-1}, \tag{3.42}$$

and define the controller

$$K_0(z) = K_c(z)S(z). \tag{3.43}$$

Then the compensated Steepest Descent Algorithm is given as follows.

Algorithm 17 *Write the transfer functions in supervector description as K_0, K_c, T and S. The compensated Steepest Descent Algorithm is characterized by choosing $K = \beta K_0$, $\beta \in \mathbb{R}$. The iterative law is given via*

$$\begin{aligned} u_{k+1} &= u_k + \beta K_0 T^* e_k, \\ e_{k+1} &= (I - \beta T T^*)e_k,\ k \geq 0, \\ u_0 &\in \mathbb{R}^{l(N+1)}. \end{aligned} \tag{3.44}$$

Monotonic convergence to

$$e_\infty = \lim_{k\to\infty} e_k = P_{\ker[T^*]}e_0, \tag{3.45}$$

is guaranteed if

$$0 < \beta < \frac{2}{\lambda_{\max}(TT^*)}.$$

T^* *stays here for conjugate transpose of the matrix* T.

Proof:
The proof is identical to that of Algorithm 12 if we consider the system

$$\tilde{y} = T\tilde{u} + \tilde{d}. \qquad (*1)$$

■

The effect of the operator TT^* can be seen by regarding the closed-loop relation

$$y = Tr. \qquad (3.46)$$

We get a perfect tracking, if $T = I$, which is not achievable by feedback control. Still, we can assume a good tracking if $||T|| \approx 1$.
Therefore if K_c provides excellent feedback control of G, then rapid convergence could be attained with a choice of the gain $\beta \approx 1$ [5].

Chapter 4

Robustness of the ILC Algorithms

If our model is not precisely known, a chosen ILC algorithm might not provide the desired result. Therefore it is useful to consider how robust the algorithms are. In this thesis, we are going to consider the parametric uncertainty. That means, we do not know the true value of a real parameter (or the parameters) but know some range in which this parameter must lie.

We can normalize such uncertainties without loss of generality, as any uncertainty $\delta \in [a, b] \subset \mathbb{R}$ can be expressed as

$$\delta = \delta_0 + w\hat{\delta} \tag{4.1}$$

with $\hat{\delta} \in [-1, 1]$, nominal value $\delta_0 \in \mathbb{R}$ and weight $w \in \mathbb{R}$.

In order to guarantee the convergence of the algorithms despite such parametric uncertainties it is required that (3.16) stays stable for all possible parameter values.

Let us consider how the uncertainty looks like in the lifted system and deliberate over the method to solve the robustness issue.

4.1 Uncertain Lifted System

Let us consider a system (**??**) with uncertainty $\hat{\Delta}$ from the set

$$\hat{\mathbf{\Delta}} = \left\{ \begin{pmatrix} \delta_1 I_{p_1} & & \\ & \ddots & \\ & & \delta_\tau I_{p_\tau} \end{pmatrix} \;\middle|\; \delta_l \in [-1,\ 1], p_l \in \mathbb{N} \text{ for } l = 1, 2, \ldots, \tau. \right\}. \tag{4.2}$$

$\tau \in \mathbb{N}$ is the number of the uncertain parameter.

Each uncertain system $\left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]\left(\hat{\Delta}\right)$ we can rewrite as a feedback interconnection of

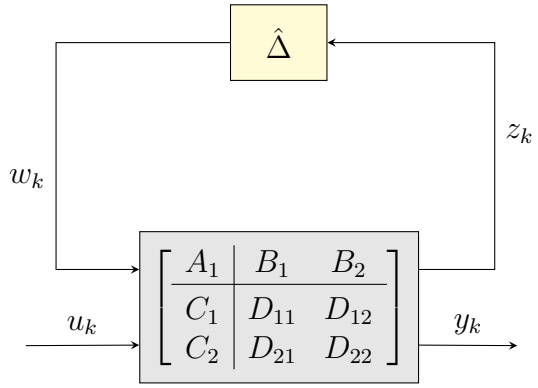


Figure 4.1: Uncertain system as feedback interconnection of LFR and the uncertainty $\hat{\Delta}$

the system $\left[\begin{array}{c|cc} A_1 & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right]$ and $\hat{\Delta}$. Then the original system can be found as star product (also called linear fractional transformation or LFT)

$$\left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] = \hat{\Delta} \star \left[\begin{array}{c|cc} A_1 & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right] \tag{4.3}$$

We illustrate it in the Figure 4.1. We speak here about the "pulling out" the uncertainty, or the *linear fractional representation* (LFR).

With LFR we rewrite the system (**??**) as

$$\begin{aligned} x(t+1) &= A_1 x(t) + B_1 u(t) + B_2 w(t), \\ y(t) &= C_1 x(t) + D_{11} u(t) + D_{12} w(t), \\ z(t) &= C_2 x(t) + D_{21} u(t) + D_{22} w(t), \\ w(t) &= \hat{\Delta} z(t), \\ x(0) &\in \mathbb{R}^n, t = 0, 1, 2, \ldots, N, \end{aligned} \tag{4.4}$$

with error $e(\cdot) = r(\cdot) - y(\cdot)$.

The corresponding lifted system has the form

$$y = G_1 u + G_2 w + d_1, \tag{4.5}$$

$$z = G_3 u + G_4 w + d_2, \tag{4.6}$$

$$e = r - y, \tag{4.7}$$

$$w = \Delta z, \tag{4.8}$$

with the matrices

$$G_1 = \begin{pmatrix} D_{11} & 0 & \cdots & 0 & 0 & 0 \\ C_1B_1 & D_{11} & \cdots & 0 & 0 & 0 \\ C_1A_1B_1 & C_1B_1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ C_1A_1^{N-2}B_1 & C_1A_1^{N-3}B_1 & \dots & C_1B_1 & D_{11} & 0 \\ C_1A_1^{N-1}B_1 & C_1A_1^{N-2}B_1 & \dots & C_1A_1B_1 & C_1B_1 & D_{11} \end{pmatrix}, \tag{4.9}$$

$$G_2 = \begin{pmatrix} D_{12} & 0 & \cdots & 0 & 0 & 0 \\ C_1B_2 & D_{12} & \cdots & 0 & 0 & 0 \\ C_1A_1B_2 & C_1B_2 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ C_1A_1^{N-2}B_2 & C_1A_1^{N-3}B_2 & \dots & C_1B_2 & D_{12} & 0 \\ C_1A_1^{N-1}B_2 & C_1A_1^{N-2}B_2 & \dots & C_1A_1B_2 & C_1B_2 & D_{12} \end{pmatrix}, \tag{4.10}$$

$$G_3 = \begin{pmatrix} D_{21} & 0 & \cdots & 0 & 0 & 0 \\ C_2B_1 & D_{21} & \cdots & 0 & 0 & 0 \\ C_2A_1B_1 & C_2B_1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ C_2A_1^{N-2}B_1 & C_2A_1^{N-3}B_1 & \dots & C_2B_1 & D_{21} & 0 \\ C_2A_1^{N-1}B_1 & C_2A_1^{N-2}B_1 & \dots & C_2A_1B_1 & C_2B_1 & D_{21} \end{pmatrix}, \tag{4.11}$$

$$G_4 = \begin{pmatrix} D_{22} & 0 & \cdots & 0 & 0 & 0 \\ C_2B_2 & D_{22} & \cdots & 0 & 0 & 0 \\ C_2A_1B_2 & C_2B_2 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ C_2A_1^{N-2}B_2 & C_2A_1^{N-3}B_2 & \dots & C_2B_2 & D_{22} & 0 \\ C_2A_1^{N-1}B_2 & C_2A_1^{N-2}B_2 & \dots & C_2A_1B_2 & C_2B_2 & D_{22} \end{pmatrix}, \tag{4.12}$$

and vectors

$$d_1 = \begin{pmatrix} C_1 x_0 \\ C_1 A_1 x_0 \\ C_1 A_1^2 x_0 \\ \vdots \\ C_1 A_1^N x_0 \end{pmatrix}, \tag{4.13}$$

$$d_2 = \begin{pmatrix} C_2 x_0 \\ C_2 A_1 x_0 \\ C_2 A_1^2 x_0 \\ \vdots \\ C_2 A_1^N x_0 \end{pmatrix}. \tag{4.14}$$

Δ is here a block diagonal matrix

$$\Delta = \mathrm{diag}(\underbrace{\hat{\Delta}, \hat{\Delta}, \ldots, \hat{\Delta}}_{(N+1) \text{ times}}) \tag{4.15}$$

and we define

$$\mathbf{\Delta} = \{\mathrm{diag}(\hat{\Delta}, \hat{\Delta}, \ldots, \hat{\Delta}) |\ \hat{\Delta} \in \hat{\mathbf{\Delta}}\}. \tag{4.16}$$

Using LFT, we get

$$y = G(\Delta)u + d(\Delta) := \left[G_1 + G_2\Delta(I - G_4\Delta)^{-1}G_3\right] u + \left[G_2\Delta(I - G_4\Delta)^{-1}d_2 + d_1\right], \tag{4.17}$$

while the inverse of the matrix $(I - G_4\Delta)$ is assumed to exist. It is always the case if the norm of the matrix G_4 is (strictly) bounded by 1 [6].

With ILC algorithms we get the input update rule

$$u_{k+1} = u_k + Ke_k, \tag{4.18}$$

and hence

$$e_{k+1} = L(\Delta)e_k = (I - G(\Delta)K)e_k. \tag{4.19}$$

If we pull out the uncertainty from $L(\Delta)$, we get

$$\left(\begin{array}{c|c} e_{k+1} \\ \hline z_k \end{array}\right) = \left(\begin{array}{c|c} \mathcal{A} & \mathcal{B} \\ \hline \mathcal{C} & \mathcal{D} \end{array}\right) \left(\begin{array}{c|c} e_k \\ \hline w_k \end{array}\right) := \left(\begin{array}{c|c} I - G_1 & -G_2 \\ \hline G_3 K & G_4 \end{array}\right) \left(\begin{array}{c|c} e_k \\ \hline w_k \end{array}\right), \tag{4.20}$$
$$w_k = \Delta z_k,$$

with the uncertain system defined through the uncertain matrix

$$L(\Delta) = \Delta \star \left(\begin{array}{c|c} \mathcal{A} & \mathcal{B} \\ \hline \mathcal{C} & \mathcal{D} \end{array}\right) = \mathcal{A} + \mathcal{B}\Delta(I - \mathcal{D}\Delta)^{-1}\mathcal{C}. \tag{4.21}$$

We are interested in stability ensuring for the system (4.19).
The most intuitive way is to ensure $||L(\Delta)|| < 1$ for all uncertainties $\Delta \in \mathbf{\Delta}$. However, this criteria might be too restrictive. We illustrate it on the following example.

Example 18 *Let us consider the system*

$$e_{k+1} = L(\alpha)e_k = \begin{pmatrix} .1 & \alpha \\ 0 & .1 \end{pmatrix} e_k \tag{4.22}$$

with an uncertain parameter $\alpha \in \mathbb{R}$. Then the norm $||L(\alpha)||$ is proportional to $|\alpha|$.
For example, with the spectral norm, we can ensure stability only for $|\alpha| < 1$.
On the other hand, one can directly see, that the system is stable for all $\alpha \in \mathbb{R}$, since the eigenvalues of the matrix equal .1.

This example motivates to devise another method for ensuring the stability of such uncertain systems. We can reach much more elegant results by considering stability using Lyapunov techniques.

4.2 Robustness Proof with LMI

First, let us consider the uncertainty Δ without taking into account its structure. Using the Lyapunov Linear Matrix Inequality (LMI) we formulate the following Theorem.

Theorem 19 *Suppose that there exists some positive definite matrix X satisfying*

$$\begin{pmatrix} I \\ L(\Delta) \end{pmatrix}^T \begin{pmatrix} -X & 0 \\ 0 & X \end{pmatrix} \begin{pmatrix} I \\ L(\Delta) \end{pmatrix} = L(\Delta)XL(\Delta) - X \preceq 0 \text{ for all } \Delta \in \mathbf{\Delta}. \tag{4.23}$$

Then the discrete dynamical system (4.19) is stable for all $\Delta \in \mathbf{\Delta}$ and for all $e_0 \in \mathbb{R}^{m(N+1)}$. It is asymptotically stable if the inequality is strict.

Proof:
First, let us prove the Theorem for the strict LMI. Then, we can find some $\gamma \in (0, 1)$,

such that

$$\begin{pmatrix} I \\ L(\Delta) \end{pmatrix} \begin{pmatrix} -\gamma X & 0 \\ 0 & X \end{pmatrix} \begin{pmatrix} I \\ L(\Delta) \end{pmatrix} = L(\Delta)^T X L(\Delta) - \gamma X \prec 0 \text{ for all } \Delta \in \mathbf{\Delta}. \qquad (*1)$$

Moreover, since $X \succ 0$, there exist some $\alpha, \beta > 0$, such that

$$\alpha I \prec X \prec \beta I. \qquad (*2)$$

Set $\eta_k := e_k^T X e_k$. Then we have

$$\eta_{k+1} = e_{k+1}^T X e_{k+1} = e_k^T L(\Delta)^T X L(\Delta) e_k \leq \gamma e_k X e_k = \gamma \eta_k \text{ for all } k \geq 0. \qquad (*3)$$

From $(*2)$ we get

$$\alpha ||e_k||^2 \leq \eta_k \leq \beta ||e_k||^2 \text{ for all } k \geq 0, \qquad (*4)$$

and with $(*3)$ it follows

$$||e_k||^2 \leq \frac{1}{\alpha}\eta_k \leq \frac{\gamma^k}{\alpha}\eta(0) \leq \frac{\beta}{\alpha}\gamma^k ||e_0||^2. \qquad (*5)$$

Since $\gamma \in (0,\ 1)$, $||e_k||^2 \to 0$ for $k \to \infty$, and taking the square root yields the proof. For not the strict case, we set in $(*3)$ and $(*5)$ $\gamma = 1$, and replace in $(*1)$ "$\prec$" with "$\preceq$". Then $||e_k||^2$ (and hence $||e_k||$) is bounded over all $k \geq 0$ and hence the system (4.19) is stable.

■

In this Theorem, we still have the uncertainty in our LMI problem. To get rid of it, we fix the matrix $X \succ 0$ and try to find a multiplier $P = P^T$, which fulfills

$$\begin{pmatrix} I \\ \Delta \end{pmatrix}^T P \begin{pmatrix} I \\ \Delta \end{pmatrix} \succeq 0, \qquad (4.24)$$

and, additionally,

$$\begin{pmatrix} I & 0 \\ \mathcal{A} & \mathcal{B} \end{pmatrix}^T \begin{pmatrix} -X & 0 \\ 0 & X \end{pmatrix} \begin{pmatrix} I & 0 \\ \mathcal{A} & \mathcal{B} \end{pmatrix} + \begin{pmatrix} \mathcal{C} & \mathcal{D} \\ 0 & I \end{pmatrix}^T P \begin{pmatrix} \mathcal{C} & \mathcal{D} \\ 0 & I \end{pmatrix} \preceq 0. \qquad (4.25)$$

If we can find such a matrix P, the stability of the system (4.19) can be proven.
We formulate it as a Theorem.

Theorem 20 *Let $\mathbb{P}$ be a subset of the symmetric matrices, and $X \succ 0$ be fixed. Then the system* (4.19) *is stable if there exists a matrix $P = P^T \in \mathbb{P}$, such that* (4.24) *and*

(4.25) *are fulfilled. If the inequality* (4.25) *is strict, the system becomes asymptotic stable.*

Proof:
Let $P = P^T \in \mathbb{P}$ be some matrix, which fulfills the relations (4.24) and (4.25).
We multiply (4.25) with

$$H = \begin{pmatrix} I \\ \Delta(I - \mathcal{D}\Delta)^{-1}C \end{pmatrix} \tag{$*1$}$$

right and its transpose left. Then the first term results in

$$H^T \begin{pmatrix} I & 0 \\ \mathcal{A} & \mathcal{B} \end{pmatrix}^T \begin{pmatrix} -X & 0 \\ 0 & X \end{pmatrix} \begin{pmatrix} I & 0 \\ \mathcal{A} & \mathcal{B} \end{pmatrix} H = L(\Delta)^T X L(\Delta) - X. \tag{$*2$}$$

Hence, if we can prove that the second term multiplied with H right and its transpose left is positive semi-definite, this matrix P will ensure the monotonic convergence of the algorithms. Multiplying H yields

$$H^T \begin{pmatrix} \mathcal{C} & \mathcal{D} \\ 0 & I \end{pmatrix}^T P \begin{pmatrix} \mathcal{C} & \mathcal{D} \\ 0 & I \end{pmatrix} H = (I - \Delta\mathcal{D})^{-T} \underbrace{\begin{pmatrix} I \\ \Delta \end{pmatrix}^T P \begin{pmatrix} I \\ \Delta \end{pmatrix}}_{\succeq 0} (I - \Delta\mathcal{D}) \succeq 0,$$

what proofs the first statement. If the inequality (4.25) is strict, the LMI (4.23) is also strict and we achieve the asymptotic stability. ■

So, the robustness issue can be reduced to a certain LMI over a multiplier set $\mathbb{P}$.

The choosing of the set $\mathbb{P}$ is here the cornerstone. It must be large enough, to be able to find such multiplier P, if the system is stable, but also adequate small, such that the searching action is not too costly.

Let us begin with a single uncertain parameter δ. Then the uncertainty $\Delta = \delta I$ with $\delta \in [-1,\ 1]$. A possible choice for a multiplier set is

$$\mathbb{P} = \{ \begin{pmatrix} S & R \\ R^T & -S \end{pmatrix} \mid S \succeq 0,\ R + R^T = 0 \}. \tag{4.26}$$

There is

$$\begin{pmatrix} I \\ \delta I \end{pmatrix}^T \begin{pmatrix} S & R \\ R^T & -S \end{pmatrix} \begin{pmatrix} I \\ \delta I \end{pmatrix} = \underbrace{(1 - \delta^2)}_{\geq 0} S + \delta \underbrace{(R + R^T)}_{=0} \succeq 0, \tag{4.27}$$

and hence the requirements on the set $\mathbb{P}$ for the Theorem (20) are fulfilled.

For $\tau \in \mathbb{N}$ uncertain real parameters the uncertainty has the form

$$\Delta = \operatorname{diag}(\underbrace{\hat{\Delta}_\tau, \hat{\Delta}_\tau, \ldots, \hat{\Delta}_\tau}_{(N+1) \text{ times}}) \tag{4.28}$$

with

$$\hat{\Delta}_\tau = \begin{pmatrix} \delta_1 I_{p_1} & & & \\ & \delta_2 I_{p_2} & & \\ & & \ddots & \\ & & & \delta_\tau I_{p_\tau} \end{pmatrix}. \tag{4.29}$$

We choose a permutation matrix $\mathcal{P}$, such that

$$\Delta_{\text{perm}} = \mathcal{P}\Delta = \begin{pmatrix} \delta_1 I_{(p_1 \cdot N)} & & & \\ & \delta_2 I_{(p_2 \cdot N)} & & \\ & & \ddots & \\ & & & \delta_\tau I_{(p_\tau \cdot N)} \end{pmatrix}. \tag{4.30}$$

If we choose the multiplier set

$$\mathbb{P}_\tau = \left\{ \begin{pmatrix} I_{\tau(N+1)} \\ \mathcal{P} \end{pmatrix}^T \begin{pmatrix} S & R \\ R^T & -S \end{pmatrix} \begin{pmatrix} I_{\tau(N+1)} \\ \mathcal{P} \end{pmatrix} \middle| \begin{array}{c} S = \operatorname{diag}(S_1, S_2, \ldots, S_\tau) \succeq 0, \\ R = \operatorname{diag}(R_1, R_2, \ldots, R_\tau), R + R^T = 0 \end{array} \right\}, \tag{4.31}$$

then for $P_\tau \in \mathbb{P}_\tau$, the following relation is satisfied

$$\begin{pmatrix} I \\ \Delta \end{pmatrix}^T P_\tau \begin{pmatrix} I \\ \Delta \end{pmatrix} = \begin{pmatrix} I \\ \Delta_{\text{perm}} \end{pmatrix}^T \begin{pmatrix} S & R \\ R^T & -S \end{pmatrix} \begin{pmatrix} I \\ \Delta_{\text{perm}} \end{pmatrix} \tag{4.32}$$

$$= \begin{pmatrix} (1-\delta_1^2)S_1 & & & \\ & (1-\delta_2^2)S_2 & & \\ & & \ddots & \\ & & & (1-\delta_\tau^2)S_\tau \end{pmatrix}. \tag{4.33}$$

Go back to Example 18. Let $\alpha \in [0, 2]$ be an uncertainty with the nomimal value $\alpha_0 = 1$. With linear fractional transformation the system

$$e_{k+1} = \begin{pmatrix} .1 & \alpha \\ 0 & .1 \end{pmatrix} e_k \tag{4.34}$$

results in

$$\begin{pmatrix} e_{k+1} \\ z_k \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0.1 & 1 \\ 0 & 0 & 0.1 \end{pmatrix} \begin{pmatrix} e_k \\ w \end{pmatrix}. \tag{4.35}$$

With the choice

$$X = \mathrm{diag}(2, 0.5) \tag{4.36}$$

we can compute with MATLAB the solution

$$P = \mathrm{diag}(4.61, -4.61), \tag{4.37}$$

which renders the eigenvalues of the left side term in the relation (4.25) to

$$\lambda_1 = -2.35,\ \lambda_2 = -1.21 \text{ and } \lambda_3 = -0.488. \tag{4.38}$$

Hence, according to the Theorem 20, the system is stable for all $\alpha \in [0, 2]$.

Example 21 *In Chapter 3 we said, that the choice of β might impact the robustness properties of the Inverse Model Algorithms. Let us illustrate it on the system* (3.1)*, but with uncertain parameter $a \in [\underline{a},\ \overline{a}]$ and nominal value $a_0 = 4$:*

$$A = \begin{pmatrix} 2 & 1 \\ a & 3 \end{pmatrix} \tag{4.39}$$

For $N = 23$ and $\beta = .1$ our system still stays robust for $\overline{a} = -\underline{a} = 2$. If we set $\beta = .4$, we can not ensure the robustness using Theorem 20 even for $\overline{a} = -\underline{a} = .1$.
However, the ability to find the adequate matrices X and P depends on the time horizon N. For $N = 20$, we get a much more robust system, for which the stability can be proven for $\overline{a} = -\underline{a} = 5$.

Chapter 5

Application

The listed algorithms have their theoretical benefits, but we did not test them on real problems. The numeric obstacles require the trade-offs between robustness, stability, and calculation time. The computer memory has limited capacity; systems discretization adds the plant errors. We considered the examples do not have a physical reference system. Now, we try to apply the algorithms on a more "real" plant and see the problems that occur.

We consider a continuous-time 3-input 3-output system from $COMPl_{e^i}b$ [4], given as

$$\dot{x}(\tau) = \begin{pmatrix} 0 & 0 & 1.132 & 0 & -1 \\ 0 & -0.0538 & -0.1712 & 0 & 0.0705 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0.0485 & 0 & -0.8556 & -1.013 \\ 0 & -0.2909 & 0 & 1.0532 & -0.6859 \end{pmatrix} x(\tau) + \begin{pmatrix} 0 & 0 & 0 \\ -0.12 & 1 & 0 \\ 0 & 0 & 0 \\ 4.419 & 0 & -1.6650 \\ 1.575 & 0 & -0.0732 \end{pmatrix} u(\tau),$$
$$y(\tau) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} x(\tau), \tag{5.1}$$

where τ is time in seconds, $0 \leq \tau \leq 30$.

Since we are working with discrete time systems, we discretize it with Matlab command

`c2d` for sample time `Ts`$= 10^{-3}$s and get the state space description

$$\left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right) = \left(\begin{array}{cccccccc} 1 & 0 & 0.0011 & 0 & -0.001 & 0 & 0 & 0 \\ 0 & 0.9999 & -0.0002 & 0 & 0.0001 & -0.0001 & 0.001 & 0 \\ 0 & 0 & 1 & 0.001 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.9991 & 0 & 0.0044 & 0 & -0.0017 \\ 0 & -0.0003 & 0 & 0.0011 & 0.9993 & 0.0016 & 0 & -0.0001 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{array}\right). \tag{5.2}$$

This discrete system is assumed to make N steps, each with time increment of `Ts`.

5.1 The case $D = 0$

In the system (5.2) the matrix D is zero, what makes our algorithms not applicable. Moreover, the D-part's neglecting is typical: the system output must mostly image the system state and not the known input we send.

But let us consider the output signal at discrete time increment $t \geq 1$:

$$y(t) = Cx(t) = CAx(t-1) + CBu(t-1). \tag{5.3}$$

Here the "new" D-term is represented by the matrix CB, if it is not zero. With other words, the input $u(\cdot)$ begins to impact the output after the first time step. We say, that the *relative degree* of the system equals 1. We define it mathematically precise.

Definition 22 *The relative degree of the system (A, B, C, D) is 0 if $D \neq 0$. For $D = 0$ it is the smallest integer k^* for which $CA^{k^*-1}B \neq 0$.*

We put the first k^* terms away and consider our dynamical system from time step k^*:

$$\begin{aligned} x(t+1) &= Ax(t) + Bu(t), \\ y(t) &= CA^{k^*+1}x(t) + CA^{k^*}Bu(t), \text{ for } t = k^*, k^*+1, \dots N. \end{aligned} \tag{5.4}$$

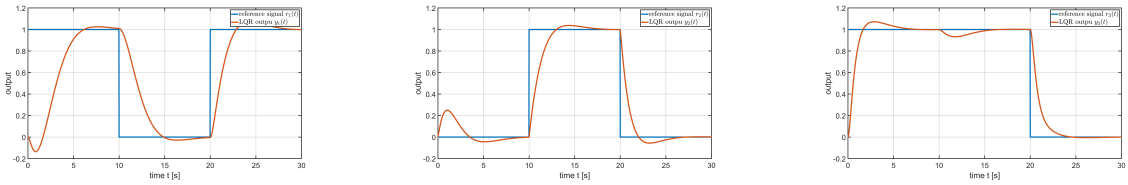
The new initial condition equals $x(k^*)$, and the start input condition is $u(k^*)$.

We can not approve the input signal at the first time steps, but still can improve it for the rest of them.

Example 23 *For our system* (5.2) *the relative degree is 1, as*

$$CB = \begin{pmatrix} -7.8726e-07 & 4.8474e-11 & 3.656e-08 \\ -0.0001 & 0.0009 & -2.5527e-09 \\ 2.2086e-06 & 8.0937-12 & -8.3225e-07 \end{pmatrix} \tag{5.5}$$

The small values of the matrix elements result from the small terms of the matrix B, and in this case does not disturb us.
We calculate the solution with LQR. Result for each of the three dimentions is illustrated in Figure 5.1.



(a) First signal (b) Second signal (c) Third signal

Figure 5.1: LQR tracking for the system (5.2)

It looks like something we can improve. However, for $Ts = 10^{-3}s$ we get $N = 10^3$ time steps even for one simulation second. The resulting matrix G can not easily be handled and needs vast masses of computer memory.

5.2 Long time horizon

While in originally discrete systems the number of time steps can be not large (as each time step represents one iteration, which does not necessarily depends upon time), the discretezized continuous systems mostly have a large time horizon.
In Example 7 the Inverse Model Algorithm is not applicable even for $N = 50$. It means, we need a trade off between the time horizon and robustness. Already for SISO system the matrix G will have $(N+1)^2$ elements.
The more obvious way to reduce the wasted memory is to use the sparse matrices. The matrix G has a lower diagonal structure, what invites to save only the non-zero elements. Even a more space saving method is to save only the last "line" of the block matrices

$$G_{\text{line}} = \begin{pmatrix} CA^{N-1}B & CA^{N-2}B & \dots & CBD \end{pmatrix}. \tag{5.6}$$

For SDA the matrix $G^\star$ can be perfectly calculated from this simplified form. However, the eigenvalues of the matrix H can not as easily be calculated. However, we can estimate the norm of the matrix $GG^\star$, and choose β from a possibly smaller region.
The solution for PIA is a bit more complicated.

One possibility to reduce the wasted memory is to split our system in the smaller ones. Let us denote the time horizon length with $N_{\max}$, and choose an N with $1 < N \leq N_{\max}$ We can apply our algorithms on the systems with the same state space description, but different initial values. For simplicity, let us assume, that we can divide our time horizon $0, 1, 2, \ldots, N_{\max}$ in p equal parts with time horizon of length N[1].

Then after we N iterations, we get for the next N iterations the initial condition

$$x_{0_1} = A^N x(0) + \sum_{\tau=0}^{N} A^\tau B u(N-\tau). \tag{5.7}$$

For the next pass we adjust the next initial value as

$$x_{0_2} = A^{2N} x(0) + \sum_{\tau=0}^{2N} A^\tau B u(t-\tau) = A^N x(N+1) + \sum_{\tau=0}^{N} A^\tau B u(2N-\tau). \tag{5.8}$$

Doing so fare $1 \leq p^* \leq p$ algorithm passes, we get

$$\begin{aligned} x_{0_{p^*}} =& A^N x((N+1)(p^*-1)) + \sum_{\tau=0}^{N} A^\tau B u(p^* N - \tau) \\ =& A^N x_{0_{p^*-1}} + \sum_{\tau=0}^{N} A^\tau B u(p^* N - \tau). \end{aligned} \tag{5.9}$$

That means, we have no need to calculate even more than N powers of A.

However, if the matrix is asymptotically stable, it might have an advantage to neglect some terms CA^pB for $1 < p < N$ large enough. It allows us to skip the little elements of the matrix G, which might lead to bad condition number.

Another benefit of small terms neglecting is the use of sparse matrices. Our matrix G has a lower triangular structure, and even without any neglecting we can decrease the memory usage by saving only the elements of G, which are not zero. With "reduced" matrix G we can maintain more space and hence choose a larger N to divide our system in.

The following theorem illustrates it and supposes a method for choosing of this p^*.

Theorem 24 *We consider the "reduced" lifted system*

$$\tilde{y} = \tilde{G}u + d, \tag{5.10}$$

[1] In a more general case, where $N_{\max}$ can not be devided by p without reminder, We can apply the algorithm one more time on the rest elements, or add them to the last applying iteration of the algorithm.

$$\tilde{G} = \begin{pmatrix} D & & & & & & & & \\ CB & D & & & & & & & \\ CAB & CB & D & & & & & & \\ \vdots & \vdots & \vdots & \ddots & & & & & \\ CA^{p^\star-1}B & CA^{p^\star-2}B & CA^{p^\star-3}B & \dots & D & & & & \\ 0 & CA^{p^\star-1}B & CA^{p^\star-2}B & \dots & CB & D & & & \\ 0 & 0 & CA^{p^\star-1}B & \dots & CAB & CB & D & & \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \\ 0 & 0 & 0 & \dots & CA^{p^\star-1} & CA^{p^\star-2}B & CA^{p^\star-3}B & \dots & D \end{pmatrix}. \tag{5.11}$$

If $||A|| < 1$*, the error* $||y(t) - \tilde{y}(t)||$*,* $p^* < t$*, can be estimated undepended on* t *as*

$$||y(t) - \tilde{y}(t)|| < 2||C||\,||B||\,||A||^{p^*}\,||(I-A)^{-1}||\,||u_{\max}||, \tag{5.12}$$

with

$$||u_{\max}|| = \max_{\tau=0}^{t-p^*} ||u(\tau)||. \tag{5.13}$$

$||\cdot||$ *denotes here any vector norm and the corresponding induced matrix norm.*

Proof:
As $||A|| < 1$, all the eigenvalues of A are less than 1 in its absolute value and hence the inverse of $(I - A)$ exists. Simple calculation provides

$$\begin{aligned} ||y(t) - \tilde{y}(t)|| &= ||C \sum_{\tau=p^*}^{t} A^\tau B u(\tau)|| \le ||C||\,||\sum_{\tau=p^*}^{t} A^\tau||\,||B||\,||u_{\max}|| \\ &= ||C||\,||B||\,||(A^{p^*} - A^{t+1})(I-A)^{-1}||\,||u_{\max}|| \\ &\le 2||C||\,||B||\,||A||^{p^*}\,||(I-A)^{-1}||\,||u_{\max}||. \end{aligned} \tag{$*1$}$$

■

This theorem says, that it does not matter how large is our N: we still can estimate our output with the same p^*. Thou this estimation requires the systems to be asymptotically stable and the norm of the matrix A should be less than one. In addition, this estimation is not very precise, and hence we can not apply this theorem to any system.

Example 25 *Four our system* (3.1) *we have for* $||u_{\max}|| \le 1.5$ $||y(t) - \tilde{y}(t)|| < 3.4033\,10^{-11}$ *for* $N = 40$*. Hence, we can apply the algorithms for very long time horizon without getting a large error.*

Example 26 *For example* (**??**) *we can not easily apply Theorem 24: for* $p^* = 10\,000$ *the estimation refers*

$$||y(t) - \tilde{y}(t)|| \leq 0.8879 \text{ for } p^* < t < N, \tag{5.14}$$

if $||u_{\max}|| \leq 1$.
One can also try to adapt the controller, such that the closed loop matrix A *has the eigenvalues of smaller absolute value. However, in our case it does not seem to be a better solution:* H_∞ *design does not provide a better applicable solution, and pole placement yields the matrix* F *with very large values.*
But we still can divide our system in the smaller parts and apply the formula (5.9).
Choose $N = 100$, *and simulate the system dynamics for 30 seconds with time step of* 10^{-3}.
But before we illustrate the result, let us consider one more issue.

Hier können auch Schlußfolgerungen präsentiert und ein Ausblick gegeben werden.

Bibliography

[1] D. A. Bristow, M. Tharayil, and A. G. Alleyne. "A survey of iterative learning control". In: *IEEE Control Systems Magazine* 26.3 (2006), pp. 96–114. DOI: 10.1109/MCS.2006.1636313.

[2] Eli Gershon. "Linear Discrete-Time Systems - H_∞ Dynamic Output-Feedback Control with Preview". In: *Informatics in Control, Automation and Robotics*. Ed. by Oleg Gusikhin and Kurosh Madani. Cham: Springer International Publishing, 2020, pp. 464–481.

[3] Snezhana Kostova et al. "Infinite horizon LQR problem of linear discrete time positive systems". In: *Comptes Rendus De L Academie Bulgare Des Sciences* 66.8 (2013), pp. 1167–1174.

[4] Friedemann Leibfritz. *COMPl$_{e^i}$b: COnstraint Matrix-optimization Problem library*. Version 1.1. URL: http://www.complib.de/.

[5] David H. Owens. *Iterative Learning Control*. Springer, 2016.

[6] Carsten W. Scherer. *Lecture Notes Robust Control*.

[7] Carsten W. Scherer. *Linear Control Theory*.

[8] Carsten W. Scherer. *Lineare Algebra and Analytische Geometrie*.

[9] "The discrete algebraic Riccati equation and linear matrix inequality". In: *Linear Algebra and its Applications* 274.1 (1998), pp. 317–365.

.1 Beweise

.1.1 Beweis 1

$$1 + 1 = 2 \tag{15}$$

Eidesstattliche Erklärung

Ich erkläre, dass ich meine Bachelor-Arbeit [*Titel der Arbeit*] selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe und dass ich alle Stellen, die ich wörtlich oder sinngemäß aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe. Die Arbeit hat bisher in gleicher oder ähnlicher Form oder auszugsweise noch keiner Prüfungsbehörde vorgelegen.
Stuttgart, den XX.XX.2013

(Name des Kandidaten)

Appendix A

Zusammenfassung und Ausblick