Lehrstuhl für Mathematische Systemtheorie
Universität Stuttgart

Projektarbeit

im Bachelorstudiengang
Simulation Technology

Iterative Learning Control

Erstgutachter/in: Prof. Dr. Carsten W. Scherer
Zweitgutachter/in: ....

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 repeat it till he or she can play it well, and, possibly, without errors. A gymnastic tries to make a perfect move by exercising it douses and 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. This type of learned open-loop control strategy is the essence of Iterative Learning Control (ILC).

Nowadays, more and more routine tasks are carried out by industrial machines or robots. The machines can achieve better precision, can perform dangerous or dirty work. It is often a cheap and effective solution for big industrial companies. ILC can be here of great use if a task needs to be done frequently. From each iteration, we can learn something to achieve better performance.

For example, let us imagine that a robot arm needs to take the same path, again and again, with high accuracy. We can possibly find a controller, which might run the robot arm along this route. However, to achieve good precision, a simple controller might not be enough, and we need to apply more complicated techniques to improve the result.

The other possibility is to take the result we get from the initial controller, and try to adjust it for the next trial. Improving the signal by each program run, we can achieve a perfect result with less effort if the signals are adapted with a proper algorithm. In this work, we consider some of such algorithms and see, that under certain conditions we can achieve monotonic convergence to an input signal, which renders an optimal tracking.

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. We also aim to prove robustness property for these algorithms despite the parameter uncertainty. Besides, we want to apply the algorithms we provide on discrete- and continuous-time systems. Especially, we face the issues that occur due to system discretization, the amount of data and numerical instability.

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

$$u_{k+1} = f(e_k, e_{k+1}, u_k) \tag{1.1}$$

where f is some function independent on k [3].

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

$$\begin{array}{rcl} x(t+1) &=& Ax(t) + Bu(t) \\ y(t) &=& Cx(t) + Du(t), \\ x(0) &=& x_0, \end{array} \tag{1.2}$$

over a time horizon $t = 0, 1, 2, \ldots, N$. A, B, C, and D denote here the matrices of fitting dimensions. We are interested in the minimizing of the error between output $y(\cdot)$ and the tracking signal $r(\cdot)$

$$e(t) = r(t) - y(t),\ t = 0, 1, 2, \ldots, N. \tag{1.3}$$

An uncommon contemplation in this work is the final time horizon. Together with discreteness of the the system it invites us to rewrite the system (1.2) 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 we call supervectors.

Then the system (1.2) 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 times $t = 0, 1, \dots, N$ in one linear equation. This form already inspires some of ILC algorithms, for example Pseudo Inverse Model Algorithm

$$u_{k+1} = u_k + \beta GG^+ e_k,$$

where G^+ is the pseudo inverse of G and k denotes the trial of system run, $\beta \in (0, 2)$.

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. u_k denotes the supervector at trial k of algorithm running.

Five different algorithms are illustrated in this work. Two of them are derived on considering of inverse model, two are inspired by steepest descent, and one is based on feedback design. The algorithms in this thesis are taken from [3]. However, some of them were modified to make them more practically usable.

Besides, 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 if the given algorithm stays stable if the parameters of the system are uncertain.

Furthermore, we discuss the issues appearing by the application of the algorithms.

With the supervector description, the ILC algorithms need an lot of computational memory and lead to significant numerical errors. We need to modify the algorithms in the way that we can apply them to the real problems.

1.3 Structure

This work consists of 6 chapters. The thesis is composed as follows: in the first chapter we in detail formulate the problem. After that, 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, where further work is suggested.

Chapter 2

Basics

To understand the following Iterative Learning Control (ILC) algorithms, we need to make a departure to basic definitions we will use. We consider the system (1.2). 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, we write (A, B, C, D) and mean it to be the discrete-time control system (1.2).
We aim to find a "good" input signal $u(t)$, such that the reference $r(t)$ is tracked possibly well for all $t \geq 0$. With other words, it means, that the error norm $||e(\cdot)||$ must stay small in some $||\cdot||$, e.g. the L_2-norm. To find it we usually design a stabilizing controller – such that our system state goes not to infinity over time t. To design it, we must first clearly define what stability means. Since let us first formulate the common definitions and entertain ideas of a stabilizing controller design.

2.1 Definitions

The stability definition is usually given asymptotically. That means, we assume $N = \infty$. Only for definitions in this chapter, we make this assumption. We will see, that it quite make sense also for finite time horizon problems.

Definition 1 *A discrete time dynamical system*

$$x(t+1) = Ax(t) \tag{2.1}$$

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

$$\lim_{t\to\infty} x(t) = 0 \tag{2.2}$$

is satisfied. It is said to be stable, if

$$\limsup_{t\to\infty} ||x(t)|| < \infty \tag{2.3}$$

for all initial conditions $x(0) \in \mathbb{R}^n$. We call the matrix A (asymptotically) stable, if the dynamical system (2.1) *is (asymptotically) stable.*

In the same way we want to define (asymptotic) stabilizability and detectability.

Definition 2 *A 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.*

2.2 Stabilizing Controller

Following from its name the stabilizing controller is a one, which stabilizes the given system (A, B, C, D). There are a bunch of different design methods, as a linear quadratic regulator, H_∞ design or pole placement.
In our case we are more interested in better stabilizing properties, as in a good performance. The algorithms we will consider guarantee the convergence to a good input, but to apply this algorithms we also need some numerically stability.
We can find a stabilizing controller for (**??**) using feedback control. First let us consider the case $y(\cdot) = x(\cdot)$.
Then the system (**??**) becomes stable, if we choose $u(t) = -Fx(t)$, $t \geq 0$, with some matrix F, such that $(A - BF)$ is stable.
Clearly, if the system is (asymptotically) stabilizable, we can find such 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

$$V(u(\cdot)) = \sum_{t=0}^{\infty} \left((x(t))^T Q(x(t)) + u(t)^T R u(t) \right), \tag{2.4}$$

overall stabilizing piecewise constant input signals $u(\cdot) \in \mathbb{R}^m$, and with $x(\cdot)$ satisfying (**??**) (for $C = I$ and $D = 0$). $Q \succ 0$ and $R \succ 0$ are here weighing matrices, which allow us to decide how "important" the elements of the in- and output signals are. Using them, we can make one input signal impact the system more than another or drive one of the output signals faster.
Choosing the weighting matrices always depends on the system we work with and allows us to affect closed-loop behavior.
The optimal $u(\cdot)$ for minimizing of the cost function (2.4) is given via $u(t) = Fx(t)$ with

$$F = (R + B^T P B)^{-1} B^T P A. \tag{2.5}$$

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

$$A^TA - A^TPB(R + B^TPB)^{-1}B^TPA + Q - P = 0. \tag{2.6}$$

Stabilizing solution means, that from (2.5) resulting matrix $(A - BF)$ is stable, and hence this solution stabilizes our system.
For the case $y(\cdot) \neq x(\cdot)$, we can additionally construct an observer to approximate the signal $x(\cdot)$. The controller we get is said to be based on *separation principle*. It requires the pair (A, C) to be detectable. So, we can find a matrix J, for which $A - JC$ is asymptotically stable.
We set

$$u(\cdot) = -F\hat{x}(\cdot), \text{ with } \hat{x}(\cdot) \text{ given via dynamical system} \tag{2.7}$$

$$\hat{x}(t+1) = A\hat{x}(t) + Bu(t) + J(y - \hat{y}(t)), \tag{2.8}$$
$$\hat{y}(t) = C\hat{x}(t) + Du(t), \tag{2.9}$$
$$\hat{x}(0) = x_0,\ t \geq 0. \tag{2.10}$$

With this observer system the estimation error can be calculated as

$$\tilde{x}(t+1) = x(t+1) - \hat{x}(t+1) = (A - JC)\tilde{x}(t),\ t \geq 0, \tag{2.11}$$

and hence $\hat{x}(\cdot)$ converges to $x(\cdot)$ at least asymptotically.
We elect

$$u(\cdot) = -F\hat{x}(\cdot) + Mr(\cdot), \tag{2.12}$$

where M is a right inverse of the matrix

$$D + (C - DF)(I - A + BF)^{-1}B. \tag{2.13}$$

This choice of input signal solves tracking issue for constant $r(\cdot)$. It still can be used for piecewise constant $r(\cdot)$, if the signal does not change too fast.

Chapter 3

ILC Algorithms

<Einfuerung>

Let us imagine that we need to process the same action multiple times. For example, a robot manipulator must put some objects in a box with high accuracy, while the objects to put are always located in the same place. If we already have some input sequence, which solves this issue, we can use the same data for the next iteration and get 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.

More formally, we consider the system (**??**) over a *finite time horizon* $t = 0, 1, 2, \ldots, N$, $N \in \mathbb{N}$. As we discussed in the previous chapter, we are interested in good tracking of the signal $r(\cdot)$. At first glance, the stability concept seems to be not necessary for $N < \infty$. Still, in this thesis, it is the requirement we presuppose on considered system. The reason is that also the discrete system over finite horizon can produce vast signals for large N, if the system is not stable. Especially, it means, that A has at least one eigenvalue larger that one, which implies very large norm of the matrices A^t for $t \gg 1$. This makes difficult the numerical calculations, since the solution of the difference equation (**??**) includes it.

Moreover, for our purposes, for stabilizable systems we can assume stability without loss of generality. If the system is stabilizable, we can find such a matrix F, such that $(A - BF)$ is stable. We define $u(t) := -Fx(t) + \tilde{u}(t)$, $t = 0, 1, 2, \ldots, N$ with a new input signal $\tilde{u}(t) \in \mathbb{R}^l$. Then the system (**??**) results in

$$\begin{aligned} x(t+1) &= (A - BF)x(t) + B\tilde{u}(t), \\ y(t) &= (C - DF)x(t) + D\tilde{u}(t),\ t = 0, 1, 2, \ldots, N. \end{aligned} \tag{3.1}$$

We can set

$$\begin{aligned} \tilde{A} &= (A - BF), \quad \tilde{B} = B \\ \tilde{C} &= (C - DF), \quad \tilde{B} = D. \end{aligned} \tag{3.2}$$

The shifted system $(\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D})$ is stable and we still can improve the input signal by considering $\tilde{u}(\cdot)$.

Example 3 *Let us consider the system (A, B, C, D) with*

$$A = \begin{pmatrix} 2 & 1 \\ 4 & 3 \end{pmatrix}, B = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, C = \begin{pmatrix} 0 & 1 \end{pmatrix}, D = 2. \tag{3.3}$$

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

$$F = \begin{pmatrix} 1.9674 & 1.3042 \end{pmatrix} \text{ and } J = \begin{pmatrix} 1.8705 & 4.7321 \end{pmatrix}. \tag{3.4}$$

To make the considered system stable we re-describe our system as

$$\begin{aligned} x(t+1) &= \begin{pmatrix} 0.0326 & -0.3042 \\ 0.0652 & 0.3916 \end{pmatrix} x(t) + \begin{pmatrix} 1 \\ 2 \end{pmatrix} \tilde{u}(t) \\ y(t) &= \begin{pmatrix} -3.9348 & -1.6084 \end{pmatrix} x(t) + 2\tilde{u}(t), \qquad t = 0, 1, 2, \ldots, N. \end{aligned} \tag{3.5}$$

We choose a piecewise constant reference $r(\cdot)$ and set $N = 30$.
With controller based on separation principle, we get the tracking illustrated in Figure 3.1.

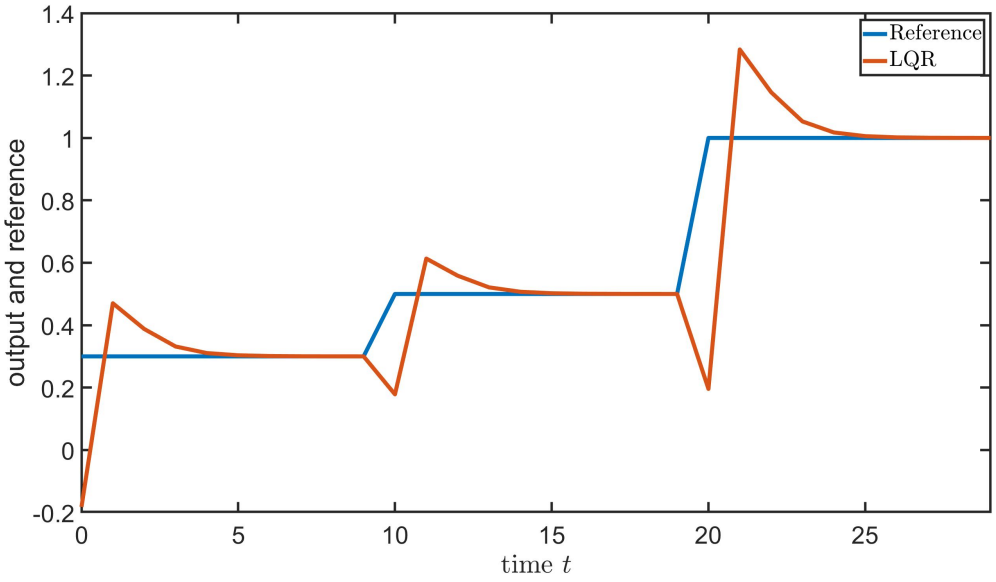

Figure 3.1: Tracking with LQR controller for system (3.5)

As one can see, the tracking in Example 3 is not perfect. However, we can extract a bunch of information from this controlled system. For example, we get the input and

output sequences

$$\{u(0),\, u(1),\, u(2)\, \ldots,\, u(N)\} \subset \mathbb{R}^l, \tag{3.6}$$

$$\{y(0),\, y(1),\, y(2),\, \ldots,\, y(N)\} \subset \mathbb{R}^m. \tag{3.7}$$

On the finite time horizon of interest the behavior of system is described by the finite input and output sequences. It seems to be an advantage.

If we put the signals we got from simulation together, we get a large, but finite dimensional vector. That means, the standard tools from linear algebra can be applied.

As a matter of fact, we can express our whole simulated system as a linear equation. We call such vectors *supervectors*, and the whole system is said to be *lifted*. This notation can describe the whole system dynamic in a compact form, and is a useful tool for ILC algorithms.

3.1 Supervector description

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*

$$s := \begin{pmatrix} s(0) \\ s(1) \\ s(2) \\ \vdots \\ s(N) \end{pmatrix}. \tag{3.8}$$

In this way we define the supervectors $u \in \mathbb{R}^{l(N+1)}$ and e, y, $r \in \mathbb{R}^{m(N+1)}$. Analogously, we determine supervectors $r \in \mathbb{R}^{m(N+1)}$ and $e \in \mathbb{R}^{m(N+1)}$.

For system (**??**) the relation between u and y, and the error e can be written in a linear matrix form

$$y = Gu + d, \tag{3.9}$$

$$e = r - y. \tag{3.10}$$

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

$$G = G(A,B,C,D) = \\ = \begin{pmatrix} D & 0 & \cdots & 0 & 0 & 0 \\ CB & D & \cdots & 0 & 0 & 0 \\ CAB & CB & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ CA^{N-2}B & CA^{N-3}B & \dots & CB & D & 0 \\ CA^{N-1}B & CA^{N-2}B & \dots & CAB & CB & D \end{pmatrix} \in \mathbb{R}^{m(N+1)\times l(N+1)}. \tag{3.11}$$

The vector d depends on the initial condition x_0:

$$d = d(C,A,x_0) = \begin{pmatrix} Cx_0 \\ CAx_0 \\ CA^2x_0 \\ \vdots \\ CA^Nx_0 \end{pmatrix} \in \mathbb{R}^{m(N+1)}. \tag{3.12}$$

The stability requirement for A is highlighted here. In opposite to vast values for unstable matrices, we get very small terms if $N \gg 1$. Intuitive, this terms can be neglected for all $p > p^\star$ for some $1 < p^\star < N$. This is usable since we can apply the sparse matrices. In the Chapter 5 we will see, that a good estimation can be derived.
For the system (3.9), the tracking problem seems to be straightforward. If we know the reference signal r, we can find a perfect solution by choosing

$$u_\infty = G^{-1}r - d. \tag{3.13}$$

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

$$y + \varepsilon w = Gu_\infty + d + \varepsilon w = r + \varepsilon w \neq r, \tag{3.14}$$

for some non-zero vector $w \in \mathbb{R}^{m(N+1)}$ and small $\varepsilon > 0$.
Also for uncertain models this method might be not 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.*

3.2 Inverse Model Algorithms

Algorithm 5 *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

$$\begin{aligned} u_{k+1} &= u_k + \beta G^{-1} e_k, \\ u_0 &\in \mathbb{R}^{l(N+1)}, \end{aligned} \tag{3.15}$$

with error evolution

$$\begin{aligned} e_{k+1} &= (1-\beta)e_k,\ k \geq 0, \\ e_0 &= r - Gu_0 - d. \end{aligned} \tag{3.16}$$

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 (3.16) 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 [3], pp 149, 152-155.
We apply this algorithm on the system from Example 3 to see the algorithm performance.

Example 6 *In the system* (3.5) *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 3.2*
But if we try to increase the length of the time horizon N, for example to 50steps, we can see a wrong system behavior (Figure 3.3).
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 matrix inversion can be numerically unstable.*

The Inverse Model Algorithm sets up very strong requirements: the matrix G should be non-singular, and its condition number must be small enough we can compute a numerically stable inverse. We can modify this algorithm by taking the Moore-Penrose

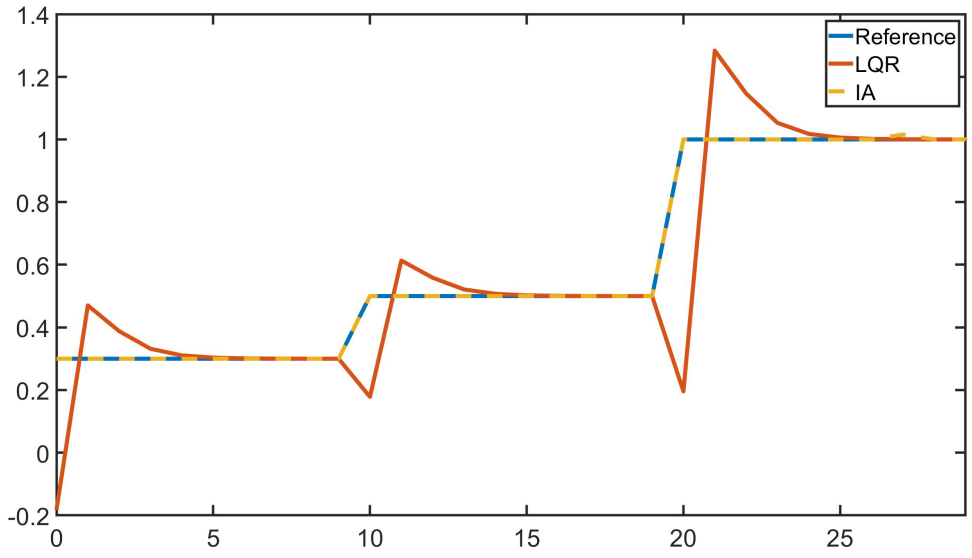

Figure 3.2: Tracking with LQR Controller (without ILC Algorithm), and tracking with IA for the system (3.5) for $N = 30$

.

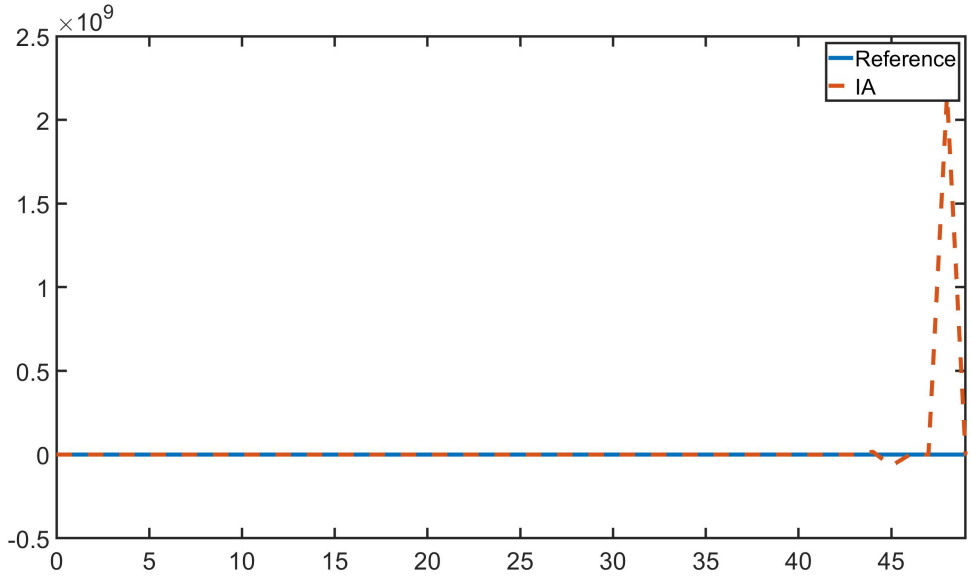

Figure 3.3: Tracking with IA for the system (3.5) for $N = 50$

inverse. This is the so-called pseudo inverse, which is uniquely defined and can be computed for all matrices [5].

Definition 7 *For a matrix $G \in \mathbb{R}^{m(N+1)\times l(N+1)}$ a psedo inverse is defined by matrix $G^+ \in \mathbb{R}^{l(N+1)\times m(N+1)}$, satisfying the criteria:*

(a) $GG^+G = G$

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

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

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

Algorithm 8 *The Pseudo Inverse Model Algorithm (PIA) is given via input update law*

$$\begin{aligned} u_{k+1} &= u_k + \beta G^+ e_k, \\ u_0 &\in \mathbb{R}^{l(N+1)}, \end{aligned} \tag{3.17}$$

with error evolution

$$\begin{aligned} e_{k+1} &= (1 - \beta G G^+) e_k, \ k \geq 0, \\ e_0 &= r - G u_0 - d. \end{aligned} \tag{3.18}$$

Monotonic convergence to

$$e_\infty = \lim_{k \to \infty} e_k = P_{\ker[G^+]} e_0, \tag{3.19}$$

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:

Recap the relation $(GG^+)^2 = GG^+GG^+ = GG^+$ the error evolution formula can be proven by mathematical induction:
for $k = 1$ it follows:

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

For $k \in \mathbb{N}$

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

To prove (3.19) recall that

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

$\oplus$ 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:

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

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

Example 9 *We calculate a solution for* (3.5) *with PIA, and illustrate it in Figure 3.4. 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.

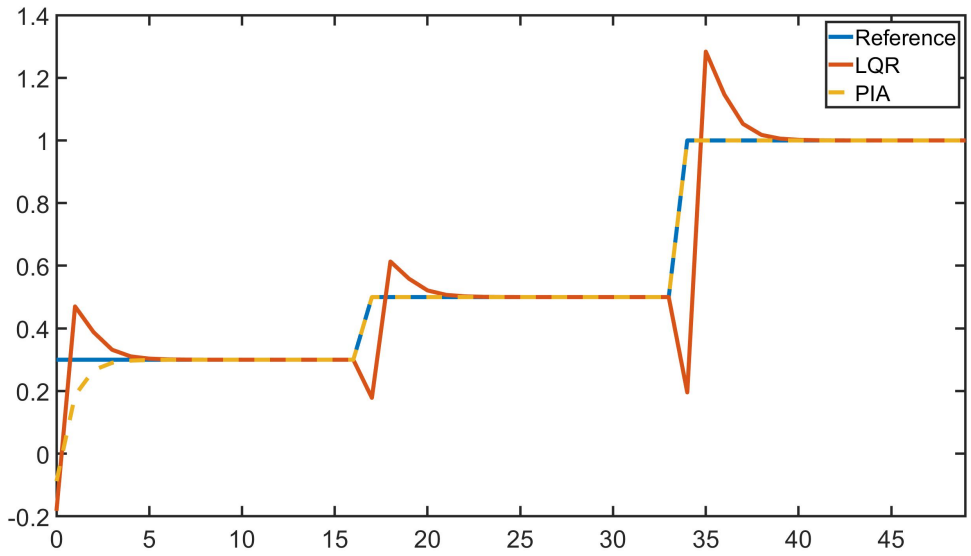

Figure 3.4: Reference signal tracking with LQR and PIA for the sysytem (3.5) for $N = 50$

We can replace the use of pseudo inverse by taking a left or right inverse, if it exists. However, it only makes sense if the calculating afford or numerical stability are better for left or right inverse calculation.
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 5 is a special case of PIA. However, numerically the use of inverse matrix might be more desirable since the calculation effort, if the condition number is not large.
(Pseudo) Inverse Algorithm belongs to the class of so-called Unit Memory Algorithms – these are the algorithms we assume to "remember" the last trial.

3.3 Unit Memory Algorithm

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

$$\begin{aligned} y_k &= Gu_k + d, \\ e_k &= r - y_k, \end{aligned} \tag{3.20}$$

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

$$||e_{k+1}|| < ||e_k|| \text{ for all } k \geq 0. \tag{3.21}$$

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 8 the perfect tracking is possible, if the reference signal r is feasible – that is, there exists an input signal u_∞, such that $r = Gu_\infty + d$. In common the convergence properties can also depend on the choice u_0.
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 10 *The Unit Memory Algorithm is given via input update law*

$$\begin{aligned} u_{k+1} &= u_k + Ke_k, \\ u_0 &\in \mathbb{R}^{l(N+1)}. \end{aligned} \tag{3.22}$$

The error dynamic is then given via

$$e_{k+1} = (I - GK)e_k,\ k \geq 0, \tag{3.23}$$

$$e_0 = r - Gu_0 - d. \tag{3.24}$$

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

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

$$e_{k+1} = (I - GK)e_k = (I - GK)^k e_0 := L^k e_0,\ k \geq 0. \tag{3.25}$$

That it 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 (3.25) stable.
If we rewrite our system as

$$\begin{pmatrix} e_{k+1} \\ e_k \end{pmatrix} = \left(\begin{array}{c|c} I & -G \\ \hline I & 0 \end{array}\right) \begin{pmatrix} e_k \\ v_k \end{pmatrix}, \tag{3.26}$$

our problem reduces to finding a stabilizing controller K with

$$v_k = Ke_k. \tag{3.27}$$

We know how to work with such systems – for example, LQR controller provides a good solution. This is a reliable method, but since the matrix G can have large dimension, it also can be costly. Often we can achieve good results with much more simple methods. For example, we can reduce the number of parameters to be chosen.
In the Algorithms 5 and 8 we chose 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 6, the calculation of the inverse matrix is not the most reliable method. The pseudo inverse refers better results, but is also much more expensive.
An algorithm, which does not require the model inversion, might be here of use.

3.4 Gradient Algorithms

We want a new algorithm to render monotonic convergence. The direct calculation provides

$$||e_{k+1}|| < ||e_k|| \text{ for all } k > 0, \tag{3.28}$$

in some norm $||\cdot||$ in $\mathbb{R}^{m(N+1)}$.
We chose positive definite weighting matrices $Q(t)$ and $R(t)$, $t = 0, 1, 2, \ldots, N$, and define the scalar products

$$\langle y, z\rangle_Q = \sum_{t=0}^{N} y(t)^T Q(t) z(t),\ \langle u, v\rangle_R = \sum_{t=0}^{N} u(t)^T R(t) v(t), \tag{3.29}$$

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

We denote with $||\cdot||_Q$ and $||\cdot||_R$ the associated norms, and set

$$\begin{aligned} G^\star &= R^{-1}G^TQ \text{ for all } G \in \mathbb{R}^{m(N+1)\times l(N+1)}, \\ K^\star &= Q^{-1}K^TR \text{ for all } K \in \mathbb{R}^{l(N+1)\times m(N+1)}, \end{aligned}$$

where

$$Q := \operatorname{diag}(Q(0), Q(1), Q(2), \ldots, Q(N)) \text{ and } R := \operatorname{diag}(R(0), R(1), R(2), \ldots, R(N)). \tag{3.30}$$

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 (3.29).
Applying the weighted norms on (3.28) provides

$$\begin{aligned} ||e_{k+1}||_Q^2 = ||e_k - GKe_k||_Q^2 &= ||e_k||_Q - 2\langle e_k, GKe_k\rangle_Q + ||GKe_k||_Q^2 \\ &< ||e_k||_Q^2, \text{ if } ||GKe_k||_Q^2 < 2\langle e_k, GKe_k\rangle_Q. \end{aligned} \tag{3.31}$$

The last inequality is equivalent to

$$e_k^TK^TG^TQGKe_k < 2e_k^TQGKe_k. \tag{3.32}$$

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

$$\beta^2(GG^\star)^TQ(GG^\star) \prec \beta QGG^\star + \beta(QGG^\star)^T. \tag{3.33}$$

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

$$H := Q^{1/2}GG^\star Q^{-1/2}. \tag{3.34}$$

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

$$\begin{aligned} \det(I - \lambda H) &= \det\left(I - \lambda Q^{1/2}GG^\star Q^{-1/2}\right) \\ &= \det\left(Q^{1/2}\right)\det\left(I - \lambda GG^\star\right)\det\left(Q^{1/2}\right)^{-1} \\ &= \det\left(I - GG^\star\right) \text{ for any } \lambda \in \mathbb{R}. \end{aligned} \tag{3.35}$$

We rewrite (3.33) as

$$\beta^2H^TH \prec \beta(H + H^T). \tag{3.36}$$

This relation is fulfilled if

$$0 < \beta < \frac{2}{\lambda_{\max}(H)}. \tag{3.37}$$

This deliberation inspires the *Steepest Descent Algorithm.*

3.4.1 Steepest Descent Algorithm

Algorithm 11 *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*

$$\begin{aligned} u_{k+1} &= u_k + \beta G^\star e_k, \\ u_0 &\in \mathbb{R}^{l(N+1)}, \end{aligned} \tag{3.38}$$

and error evolution results in

$$e_{k+1} = (I - \beta G G^\star) e_k, \ k \geq 0. \tag{3.39}$$

Monotonic convergence to

$$e_\infty = \lim_{k \to \infty} e_k = P_{\ker[G^\star]} e_0, \tag{3.40}$$

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

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

For convergence to $e_{\ker}$ it is enough to show that $\lambda_{\max}(I - \beta G G^\star) < 1$: as $(I - \beta G G^\star)$ is a symmetric matrix, the convergence is guaranteed if and only if all the eigenvalues of $(I - \beta G G^\star)$ are less than 1 in its absolute value.
If $\lambda_{\max}(I - \beta G G^\star) = 0$, then $(I - \beta G G^\star) = 0$ 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

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

Since our choice of β, the inequality (3.33) 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:

$$|1 - \mu| < 2. \tag{$*5$}$$

It concludes $|\mu| < 1$ and proves the convergence. ■

An interesting observation is, that via this algorithms got input signal u_∞ is the unique solution of the (minimum norm) optimization problem

$$u_\infty = \arg \min_{u \in \mathbb{R}^{l(N+1)}} \{||u - u_0||_R^2 : \text{ subject to } r = Gu + d\}. \tag{3.41}$$

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 minimum energy solution u_∞ [3] 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 inputs and outputs 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 rapidly 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 at some time intervals, or if we need to reflect the physical units used.

Example 12 *In the Example 9 we could not achieve perfect tracking at first trials. We can improve it by using the SDA, if we choose adequate weight Q.*

With choice

$$Q = \operatorname{diag}\begin{pmatrix} 10^{-2} & 1 & 1 & 1 & 10^{-2} & \dots & 10^{-2} \end{pmatrix}. \tag{3.42}$$

we speed up the convergence at the second, third and fourth time step.
The result is illustrated in Figure 3.5.

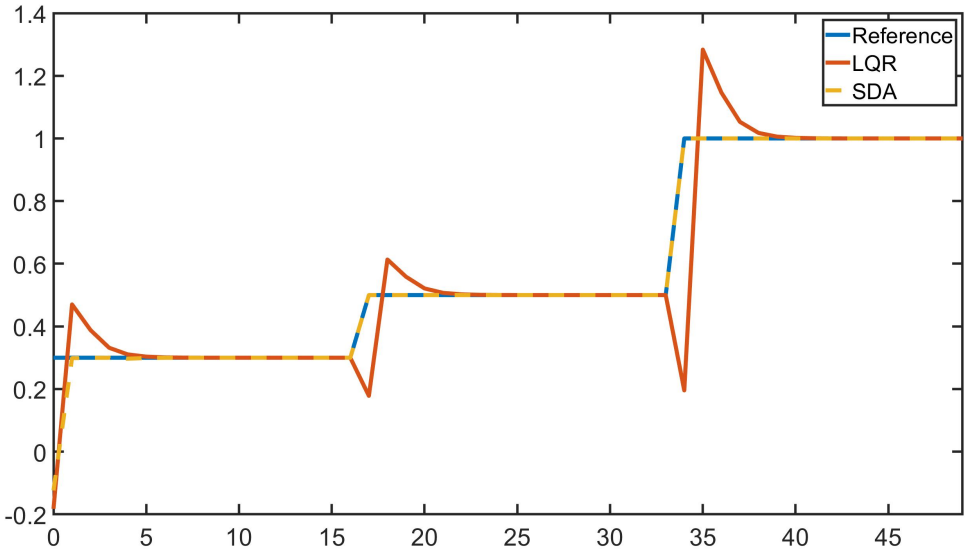

Figure 3.5: Reference signal tracking with LQR and SDA for the system (3.5)

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

$$R = \operatorname{diag}\begin{pmatrix} 1 & 1 & 1 & 1 & 0.1 & \dots & 0.1 \end{pmatrix} \tag{3.43}$$

only 2569 trials are required.

3.4.2 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 gain β for each step: $K_k = \beta_k K_0$, $k \geq 0$ for a fixed matrix K_0.
Let us consider the eigenvalues of the matrix $GG^\star$ and choose the gains β_k, $k \geq 0$ according to them.

Theorem 13 *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*

$$\begin{aligned} u_{k+1} &= u_k + \beta_k G^\star e_k, \\ e_{k+1} &= (I - \beta_{k+1} G G^\star) e_k = \prod_{l=0}^{k+1} (I - \beta_l G G^\star) e_0, \ k \geq 0, \\ u_0 &\in \mathbb{R}^{l(N+1)}. \end{aligned} \tag{3.44}$$

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 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 error norm.

Also computing the eigenvalues can be numerically questionable. 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 directly 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 14 *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*

$$\begin{aligned} \beta_k &= \frac{1}{p_k} \text{ for } k = 0, 1, \ldots, N_p, \\ \beta_k &= \beta \text{ for } k > N_p, \end{aligned} \tag{3.45}$$

with $\beta \in (0, \frac{2}{\sigma^2_{\max}})$.
Then the input update law becomes

$$u_{k+1} = u_k + \beta_k G^\star e_k,\ k \geq 0, \tag{3.46}$$

$$u_0 \in \mathbb{R}^{l(N+1)}, \tag{3.47}$$

and error evolution results in

$$\begin{aligned} e_k &= \left[\prod_{l=0}^{k}(I - \beta_l GG^\star)\right] e_0,\ k = 0, 1, \ldots, N_p, \\ e_k &= (I - \beta GG^\star)^{k-N_p} e_{N_p},\ k > N_p, \end{aligned} \tag{3.48}$$

This algorithm has the same convergence properties as Algorithm 11, 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 15 *We apply the SoE Algorithm on the system* (3.5) *with weights $Q = I$, $R = I$. The error evolution we get is illustrated in Figure 3.6.*
The new algorithms needs only 121 iterations, while the not adjusted SDA 730 trials are necessary. Indeed, for 31 of 51 eigenvalues λ of $GG^\star$ holds

$$\lambda > \frac{1}{2}\lambda_{\max}(H) = 461.5\,. \tag{3.49}$$

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.

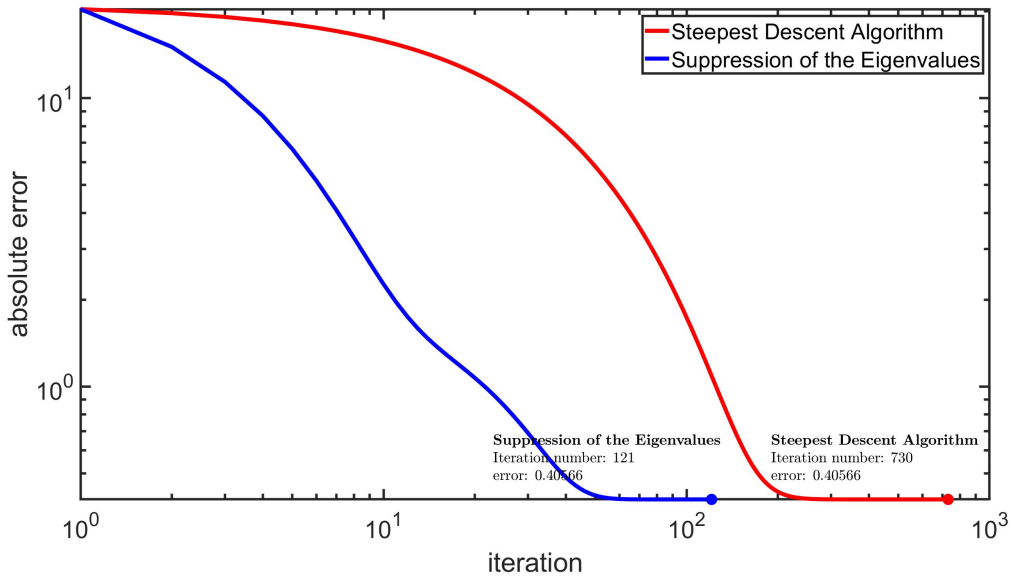

Figure 3.6: Error evolution for SDA and SoE for system (3.5)

3.5 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 [3]. 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 $S(z)$:

$$T(z) = (I + G(z)K_c(z))^{-1}G(z)K_c(z) \text{ and } S(z) = I - T(z) = (I + G(z)K_c(z))^{-1}, \tag{3.50}$$

and define the controller

$$K_0(z) = K_c(z)S(z). \tag{3.51}$$

Then the compensated Steepest Descent Algorithm is given as follows.

Algorithm 16 *Write the transfer functions in supervector description as K_0, K_c, T and S. The compensated Steepest Descent Algorithm is is characterized by choosing $K = \beta K_0$, $\beta \in \mathbb{R}$. The iterative law is given via*

$$\begin{aligned} u_{k+1} &= u_k + \beta K_0 T^* e_k, \\ e_{k+1} &= (I - \beta T T^*) e_k, \ k \geq 0, \\ u_0 &\in \mathbb{R}^{l(N+1)}. \end{aligned} \tag{3.52}$$

Monotonic convergence to

$$e_\infty = \lim_{k \to \infty} e_k = P_{\ker[T^*]} e_0, \tag{3.53}$$

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 this in Algorithm 11 if consider the system

$$\tilde{y} = T\tilde{u} + \tilde{d}. \tag{$*1$}$$

■

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

$$y = Tr. \tag{3.54}$$

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 gain $\beta \approx 1$ [3].

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 should 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 (3.25) stays stable for all possible value of parameters.

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

<Einfuehrung>

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

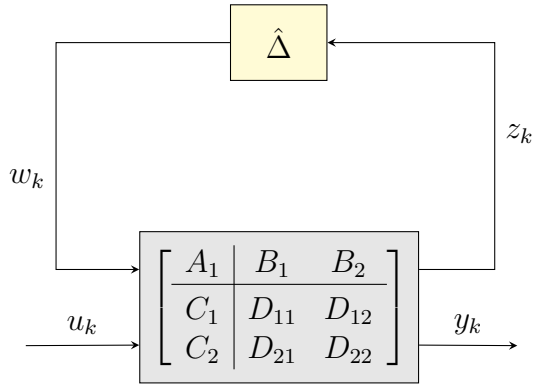

Figure 4.1: Uncertain system as feedback interconnection of LFR and the uncertainty $\hat{\Delta}$

of 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 find as star product (or linear fractional transformation, 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 [4].

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 ensuring of the stability for the system (4.19).
The most obviously 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 17 *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 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 the uncertain systems. Much more elegant results we can reach by considering stability using Lyapunov techniques.

4.2 Full-Block S-Procedure

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

Theorem 18 *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 claim.
For not 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 19 *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, 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 to

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

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 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 (19) 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 the example 17. 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

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

With the choice

$$X = \mathrm{diag}(2, .5) \tag{4.36}$$

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

$$\lambda_1 = -2.35,\ \lambda_2 = -1.21 \text{ and } \lambda_3 = -.488. \tag{4.38}$$

Hence, according to the Theorem 19, the system is stable for all $\alpha \in [0, 2]$.

Example 20 *In the 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* (3.3)*, 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 19 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 they might not be good for real issues. 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 the more "real" plant and see the problems that occur.

5.1 The case $D = 0$

Example 21 *We consider a dynamical system, which describes a flexible satellite. As known from [4] the continuous-time transfer function has the form*

$$G(s) = \frac{.036(s + 25.28)}{s^2(s^2 + .0396s + 1)}. \tag{5.1}$$

Since we are working with discrete time systems, we discretize it for sample time $T_s = 10^{-3}$ *and get the state space description*

$$\left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right) = \left(\begin{array}{cccc|c} 1.000 & 0.005 & 0.000 & 0.000 & 1.047 \cdot 10^{-13} \\ 0 & 1.000 & 0.001 & 0.000 & 8.333 \cdot 10^{-11} \\ 0 & 0 & 1.000 & 0.001 & 2.5 \cdot 10^{-7} \\ 0 & 0 & -0.001 & 1.000 & 5 \cdot 10^{-4} \\ \hline 0.362 & 0.072 & 0 & 0 & 0 \end{array}\right). \tag{5.2}$$

In this example, 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.
Let us consider the output signal at time $1 \leq t \leq N$:

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

Here the "new" D-term is represented by the matrix CB, if it is not zero. With other words, the input $u(t)$ 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*

$$CB = 6.038 \cdot 10^{-12} \neq 0. \tag{5.5}$$

The small value of the term does not disturb us, as the whole system consists of the matrices with little element values.
We calculate the solution with LQR. Result is illustrated in Figure 5.1.

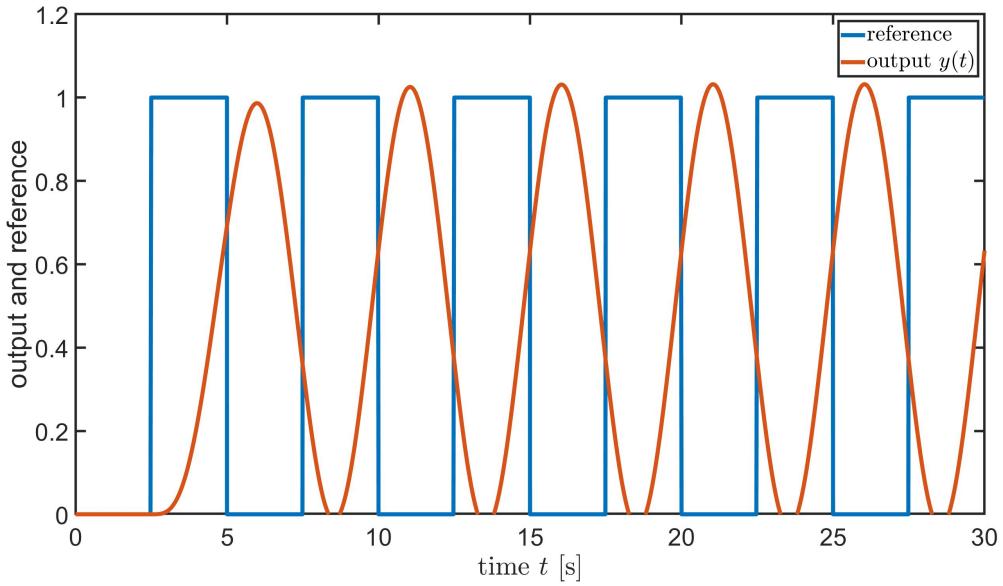

Figure 5.1: LQR Solution for the system (5.2)

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 matrix* G *has more than* 10^9 *elements and can not easily be handled.*

5.2 Long time horizon

In Example 6 the Inverse Model Algorithm is not applicable even for $N = 50$. It means, we need a trade off between the time horizon and robustness. However, for real problems the sample times are mostly small, and hence even for one second of the simulation we get huge number for time horizon. Already for SISO system the matrix G will have $(N+1)^2$ elements. Our algorithms become unusable for casual applications since they need vast masses of memory.

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, and the matrix $D = 0$.

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

$$x_{0_1} = A^N x(0) + \sum_{\tau=0}^{N} A^\tau B u(t-\tau). \tag{5.6}$$

For the next pass we adjust the next initial value as

$$x_{0_2} = A^{2N} x(0) + \sum_{\tau=0}^{2N} A^\tau B u(t-\tau) = A^N x(N+1) + \sum_{\tau=0}^{N} A^\tau B u(2N-\tau). \tag{5.7}$$

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

$$x_{0_{p^*}} = A^N x((N+1)(p^*-1)) + \sum_{\tau=0}^{N} A^\tau B u(p^* N - \tau). \tag{5.8}$$

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 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 "reduces" 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*

$$\tilde{y} = \tilde{G} + d. \tag{5.9}$$

The matrix

$$\tilde{G} = \begin{pmatrix} D & & & & & & & & \\ CB & D & & & & & & & \\ CAB & CB & D & & & & & & \\ \vdots & \vdots & \vdots & \ddots & & & & & \\ CA^{p^\star-1}B & CA^{p^\star-2}B & CA^{p^\star-3}B & \dots & D & & & & \\ 0 & CA^{p^\star-1}B & CA^{p^\star-2}B & \dots & CB & D & & & \\ 0 & 0 & CA^{p^\star-1}B & \dots & CAB & CB & D & & \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \\ 0 & 0 & 0 & \dots & CA^{p^\star-1} & CA^{p^\star-2}B & CA^{p^\star-3}B & \dots & D \end{pmatrix}, \tag{5.10}$$

with asymptotic stable A.
Then the error $||y(t) - \tilde{y}(t)||$, $p^* < t$, can be estimated undepended on t as

$$||y(t) - \tilde{y}(t)|| < 2||C||\,||B||\,||A^{p^*}||\,||(I-A)^{-1}||\,||u_{\max}||, \tag{5.11}$$

with

$$||u_{\max}|| = \max_{\tau=0}^{t-p^*} ||u(\tau)||. \tag{5.12}$$

Proof:
Since A is asymptotically stable, 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 is not very precise, and hence is not a good tool for any system.

Example 25 *Four our system* (3.5) *we have for* $||u_{\max}|| \le 1.5$ $||y(t) - \tilde{y}(t)|| < 5.13\,10^{-11}$.
TODO: Beispiel

Example 26 *For example* (5.2) *we can not easily apply Theorem 24: for* $p^* = 10\,000$

the estimation refers

$$||y(t) - \tilde{y}(t)|| \leq 0.8879 \text{ for } p^* < t < N, \tag{5.13}$$

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 (5.8).
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.

5.3 Online and Offline learning

In the real problems we often can not have all the information immediately. If we apply the input sequences on the real system, we might have only the results at current time step and the previous ones.
This is the difference between online and offline learning. In online mode we only have the information from previous iterations. We can not see in the feature and know what results we will get. In opposite, with offline learning we first calculate the new input sequence with an algorithm, and after that send the calculated result into the system and obtain the output.
In the case of offline learning no more modification are required. We can discretize the system if it is required, split it in the smaller intervals and apply the algorithms. Especially for simulation this might be a good solution, as we can improve the whole sequence at once.
The online learning requires a bit more effort. Here we apply a chosen ILC algorithm at each time step. This might need more iterations as offline learning, but still the number might rise not too large: since w already applied the algorithm on te previous steps, the new approach should probably need less trials.
For online learning, the following pseudo code can be used:

```
x0new = x0
for i = 1 to N
        x0new = A x0new + B u(i)
        unew = ILC(S, x0, i)
        u(1:i) = unew
end for
```

```
for i = N+1 to Nmax
        x0 := A x0new + B u(i)
        unew = ILC(S, x0, N)
        u(i-N:i) = unew
end for
```

$ILC(S, x0, k)$ represents here an ILC algorithm over a system S, with initial condition x_0 and with k – given time horizon.

Example 27 *We continue Example 26, and apply the online learning on it. The result for IA is illustrrated in the figure TODO.*

Chapter 6

Questions

For SDA:

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} G_1 & 0 \\ G_3 & G_1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} + \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}. \tag{6.1}$$

$$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix}_{k+1} = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}_k + \begin{pmatrix} G_1^* & G_3^* \\ 0 & G_1^* \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}_k. \tag{6.2}$$

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] Z. Kowalczuk. "Discrete approximation of continuous-time systems: a survey". In: *IEE Proceedings G - Circuits, Devices and Systems* 140.4 (1993), pp. 264–278. DOI: 10.1049/ip-g-2.1993.0045.

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

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

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

.1 Beweise

.1.1 Beweis 1

$$1 + 1 = 2 \tag{3}$$

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