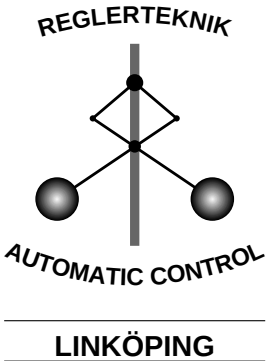
Linköping Studies in Science and Technology
Thesis No. 899

Robust Verification and Identification of Piecewise Affine Systems

—

Jacob Roll



Division of Automatic Control
Department of Electrical Engineering
Linköpings universitet, SE–581 83 Linköping, Sweden

Linköping 2001

Robust Verification and Identification of Piecewise Affine Systems

roll@isy.liu.se
http://www.control.isy.liu.se
Department of Electrical Engineering
Linköpings universitet
SE–581 83 Linköping
Sweden

ISBN 91-7373-101-3
ISSN 0280-7971
LiU-TEK-LIC-2001:36

Printed by UniTryck, Linköping, Sweden 2001

To the reader

Abstract

Piecewise affine systems constitute an important subclass of hybrid systems, and consist of several affine dynamic subsystems, between which switchings occur at different occasions. As for hybrid systems in general, there has been a growing interest for piecewise affine systems in recent years, and they occur in many application areas.

In many cases, safety is an important issue, and there is a need for tools that prove that certain states are never reached, or that some states are reached in finite time. The process of proving these kinds of statements is called *verification*. Many verification tools for hybrid systems have emerged in the last ten years. They all depend on a model of the system, which will in practice be an approximation of the real system. Therefore it would be desirable to learn how large the model errors can be, before the verification is not valid anymore. In this thesis, a verification method for piecewise affine systems is presented, where bounds on the allowed model errors are given along with the verification.

Apart from being necessary for verification, system models are needed for various control, stability analysis, and fault detection tools. Hence, *identification* of piecewise affine systems is an important area. Here, an overview of different approaches appearing in the literature is presented, and a new identification method based on mixed-integer programming is proposed. One notable property of the latter method is that the global optimum is guaranteed to be found within a finite number of steps. The complexity of the mixed-integer programming approach is discussed, and its relations to existing approaches are pointed out. The special case of identification of Wiener models is considered in detail, since that model structure makes it possible to reduce the computational complexity. A combination of the mixed-integer programming approach and a local minimisation approach is also investigated.

Acknowledgements

First of all I would like to thank my supervisor Professor Lennart Ljung, for guiding me through the various different phases of graduate studies. I would also like to thank Dr. Alberto Bemporad at the University of Siena, for a nice and fruitful collaboration, and for interesting and rewarding discussions. Fredrik Tjärnström also deserves special thanks, for persuading me to join the Automatic Control group and for many interesting mathematical discussions.

Ola Härkegård, Frida Gunnarsson, Martin Enqvist, and Johan Löfberg proofread the thesis, for which I am extremely grateful. Your comments and questions have undoubtedly improved the quality of the thesis considerably. Måns Östring and Mikael Norrlöf have also been of great help when it comes to LaTeX problems and similar issues. I would also like to thank Andreas Rietz at the Department of Mathematics, for good feedback on the problems in Section 2.4.

This work has been supported by the ECSEL graduate school in Linköping, which is gratefully acknowledged.

Finally, I would like to thank everyone in the Control and Communication group(s) (including the persons already mentioned), for creating a nice and friendly atmosphere. I really enjoy being a part of this group of people!

Linköping, 4 September, 2001

Jacob Roll

Contents

Acronyms

MILP	Mixed-integer linear program(ming), Section 2.3
MIQP	Mixed-integer quadratic program(ming), Section 2.3
LP	Linear program(ming), Section 2.3
QP	Quadratic program(ming), Section 2.3
PWA	Piecewise affine, Chapter 3
PWARX	Piecewise autoregressive exogenous, Section 3.1.4
HL CPWL	High level canonical piecewise linear, Section 3.1.6
HS	Hinging sigmoid, Section 5.1.2

1

Introduction

In the last decade, there has been a growing interest in *hybrid systems*. Hybrid systems are systems that have both continuous and discrete dynamics. A simple example could be a physical system with continuous dynamics, controlled by a discrete controller. The continuous dynamics of hybrid systems is typically associated with physical systems, possibly controlled by continuous controllers, while the discrete dynamics for instance may come from discrete controllers, inherent nonlinearities in the physical system, on/off switches, or external discrete events influencing the system.

Due to this hybrid structure, it can be hard to use, e.g., linearisation techniques for control design, analysis, etc., of such systems. Therefore, there has been a need for developing the theory of hybrid systems.

A special class of hybrid systems are the *piecewise affine (PWA) systems*. They consist of several affine subsystems, between which switchings occur at different occasions. Such systems occur in many applications, e.g., when there are physical limits (such as a tank that can get full or empty), bounds on the control signal, dead-zones, or when the system contains switches and thresholds. Piecewise affine systems can also be used to approximate nonlinear systems.

Example 1.1 (Bounded signal) *Consider the system in Figure 1.1, where a linear system is controlled by linear feedback, but where the control signal is*

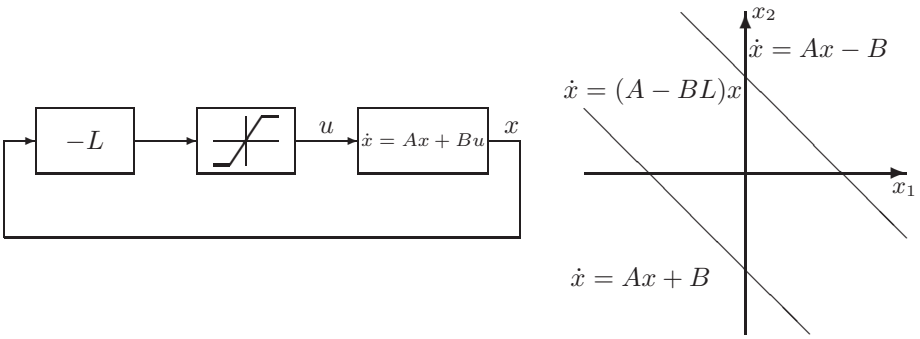


Figure 1.1: A linear system, controlled by linear feedback. Since the control signal is bounded, the system is a piecewise affine system.

bounded by $|u| \leq 1$. *As long as the control signal is kept within the bounds, the system is linear, but when* $|Lx| > 1$, *the dynamics is changed from* $\dot{x} = (A - BL)x$ *to* $\dot{x} = Ax \pm B$. *We get a piecewise affine system with three subsystems, as shown in the figure.*

Example 1.2 (Temperature control) *To control the temperature in a house, the heating system often uses thermostats, that switch the radiators on or off when the temperature reaches certain thresholds. This will give a piecewise affine behaviour, where one affine system is given by the radiators being turned off, and the other by the radiators being turned on.*

In many application areas, such as in chemical industry and aeronautics, safety is a very important issue. For example, given certain assumptions on the system, one would like to be able to ensure that the system never reaches certain specified bad (or dangerous) states. This type of problems belongs to the area of *verification*.

Many tools and methods for verification, as well as for control, stability analysis, etc., of hybrid systems have emerged in recent years. To be able to use these tools, however, a model of the system is needed. The area of *piecewise affine system identification* deals with the problem of finding a piecewise affine model of the system from experimental data.

Even if a system model is given, it is mostly just an approximation – good or bad – of the real system. Therefore, in this thesis, the verification problem is considered for piecewise affine systems with model uncertainties. A method is presented, where upper bounds on the uncertainties that can be tolerated are computed along with the verification process.

We will also consider the piecewise affine system identification problem, and give an overview of different approaches occurring in the literature. An approach

based on mixed-integer programming will also be presented. This approach guarantees that an optimal model (with respect to the particular criterion used and the experimental data available) is found, but this guarantee comes at a price of greater computational complexity.

1.1 Thesis Outline

The thesis consists of three parts:

- Background and introductory material can be found in Chapters 2 and 3. Chapter 2 contains some mathematical preliminaries needed in subsequent chapters, together with a very brief introduction to system identification. Chapter 3 introduces different classes of piecewise affine systems, and gives an overview of some of the previous research about different aspects of piecewise affine systems.

- Chapter 4 concerns robust verification, which is one of the main contributions of this thesis.

- In Chapters 5 and 6, the problem of identification of piecewise affine systems is studied. Chapter 5 gives a survey of existing approaches, while Chapter 6 considers the identification of piecewise affine systems using mixed-integer linear or quadratic programming, which is the second main contribution of the thesis.

1.2 Contributions

The main contributions of this thesis are:

- The robust verification methods in Chapter 4, together with the method of finding a maximally robust controller, outlined in Section 4.8.4.

- A survey over different piecewise affine system identification approaches, and a classification of the methods into different categories, found in Chapter 5.

- The techniques in Chapter 6 of using mixed-integer linear/quadratic programming (MILP/MIQP) to identify piecewise affine systems. A technique of reformulating products of continuous affine functions and discrete variables as linear inequalities, described in Section 6.1.2.

- Extension of an identification algorithm proposed by Hush and Horne [44], and showing its relationship to the MILP/MIQP formulations. This is found in Section 6.6.

Some of the material in Chapter 4 has previously been published in

J. Roll. Invariance of approximating automata for piecewise linear systems with uncertainties. In *Hybrid Systems: Computation and Control. Third International Workshop*, Lecture Notes in Computer Science 1790, pages 396–406. Springer-Verlag, 2000.

Other parts of Chapter 4 also appear in

J. Roll. Robust verification of piecewise affine systems. Submitted to 15th IFAC World Congress on Automatic Control, 2002.

Some of the material in Sections 6.1, 6.2, and 6.4 is joint work with Dr. Alberto Bemporad. This material is also published in

A. Bemporad, J. Roll, and L. Ljung. Identification of hybrid systems via mixed-integer programming. In *The 40th IEEE Conference on Decision and Control*, 2001. To appear.

2

Preliminaries

In this chapter, some general concepts and results needed in subsequent chapters are introduced. These include some material about convex sets and polyhedra, system identification, mixed-integer programming, and some combinatorial aspects in classification. References for further reading are given in each section.

2.1 Convex Sets and Polyhedra

For the piecewise affine system models used in this thesis, some concepts concerning convex sets and polyhedra will be needed. These are presented in this section.

A line passing through two different points $x_1, x_2 \in \mathbb{R}^n$, $x_1 \neq x_2$, can mathematically be described as the set

$$\{\lambda x_1 + (1-\lambda)x_2 \mid \lambda \in \mathbb{R}\}$$

The (closed) *line segment* between x_1 and x_2 is the part of the line that lies between the points (including x_1 and x_2). This corresponds to

$$\{\lambda x_1 + (1-\lambda)x_2 \mid 0 \leq \lambda \leq 1\}$$

If $x_1 = x_2$, the line segment consists of the single point x_1.

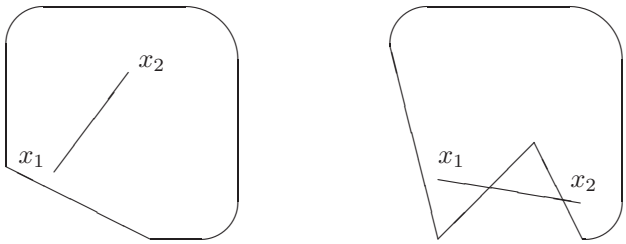


Figure 2.1: A convex set (left) and a nonconvex set (right).

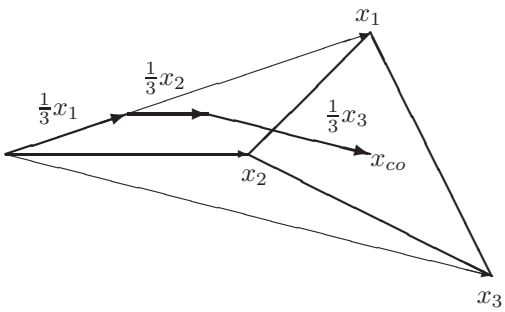


Figure 2.2: An example of a convex combination. Any point in the convex hull of $\{x_1, x_2, x_3\}$ can be written as a weighted mean (in the form (2.1)) of x_1, x_2 and x_3.

A set $P \subseteq \mathbb{R}^n$ is *convex* if the line segment between any two points in P lies entirely in P (see Figure 2.1). This condition can be written as

$$\forall x_1, x_2 \in P, \lambda \in [0,1] : \lambda x_1 + (1-\lambda)x_2 \in P$$

For any set S, the smallest convex set P that contains S (i.e., $S \subseteq P$) is called the *convex hull* of S.

We can extend the concept of line segment between two points to consider a weighted mean of a number of points $x_1, \ldots, x_r$:

$$x_{co} = \sum_{i=1}^{r} \lambda_i x_i, \quad \lambda_i \geq 0, \ \sum_{i=1}^{r} \lambda_i = 1 \tag{2.1}$$

This kind of weighted mean is called a *convex combination* of $x_1, \ldots, x_r$ (see Figure 2.2). It turns out that the convex hull of a set S is the set of all convex combinations of points in S (or alternatively, this could be used as the definition of convex hull).

An important example of convex sets are the *polyhedra*. Polyhedra are sets defined by linear inequalities, and can be written in the form

$$\{x \in \mathbb{R}^n \mid Cx \preccurlyeq d\} \tag{2.2}$$

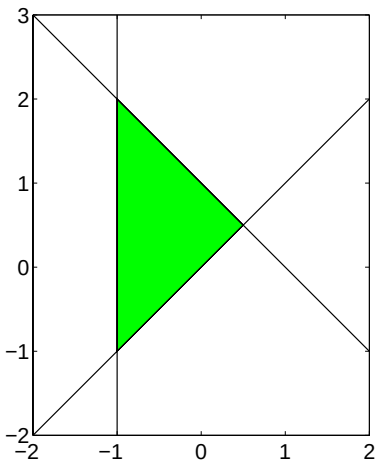


Figure 2.3: The polytope in Example 2.1.

where $C \in \mathbb{R}^{M \times n}$, $d \in \mathbb{R}^M$, and $\preccurlyeq$ denotes componentwise inequality. A bounded polyhedron is called a *polytope*.

Example 2.1 (Polytope) *The set $P \subseteq \mathbb{R}^2$ of points satisfying*

$$\begin{aligned} x_1 + x_2 &\leq 1 \\ x_1 - x_2 &\leq 0 \\ -x_1 &\leq 1 \end{aligned}$$

is a polyhedron (see Figure 2.3). Furthermore, since P is bounded, it is a polytope.

Polytopes can also be represented in an alternative way, namely as the set of convex combinations of the corners. This type of representation is called *direct representation*.

Example 2.2 (Direct representation) *The polytope in Example 2.1 can also be represented as*

$$\left\{ \lambda_1 \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix} + \lambda_2 \begin{pmatrix} -1 \\ 2 \end{pmatrix} + \lambda_3 \begin{pmatrix} -1 \\ -1 \end{pmatrix} \;\middle|\; \lambda_i \geq 0, \sum_{i=1}^{3} \lambda_i = 1 \right\}$$

The direct representation for unbounded polyhedra is a bit trickier. Apart from the corners, we must also include in the representation some vectors $x_{r+1}, \ldots, x_{r+h}$, which are parallel to the unbounded edges of the polyhedron:

$$\{\sum_{i=1}^{r+h} \lambda_i x_i \mid \lambda_i \geq 0,\ \sum_{i=1}^{r} \lambda_i = 1\} \tag{2.3}$$

Note here that $\lambda_{r+1}, \ldots, \lambda_{r+h}$ are not included in the set of λ_j that should sum up to one; they can be arbitrarily large.

Example 2.3 (Unbounded polyhedron) *If we would remove the constraint $-x_1 \leq 1$ from the polytope in Example 2.1, we would get an unbounded polyhedron defined by*

$$\begin{aligned} x_1 + x_2 &\leq 1 \\ x_1 - x_2 &\leq 0 \end{aligned}$$

This polyhedron can be represented as

$$\left\{ 1 \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix} + \lambda_2 \begin{pmatrix} -1 \\ -1 \end{pmatrix} + \lambda_3 \begin{pmatrix} -1 \\ 1 \end{pmatrix} \mid \lambda_i \geq 0 \right\}$$

Some special polyhedra (e.g., a hyperplane) do not have any corners. In that case, a "dummy" corner must be introduced somewhere in the polyhedron to allow for a direct representation.

Unfortunately, going from direct representation (2.3) to the form (2.2) is in general computationally quite complex.

In Chapter 4, we are going to use *open polyhedra*. By this will be meant a set, whose direct representation is

$$\{\sum_{i=1}^{r+h} \lambda_i x_i \mid \lambda_i > 0,\ \sum_{i=1}^{r} \lambda_i = 1\} \tag{2.4}$$

i.e., where all λ_i are strictly positive. Note that this does not necessarily mean that the set is open. However, relative to the smallest affine subspace containing the open polyhedron, it is an open set.

Example 2.4 (Open polyhedron) *The (open) line segment*

$$\left\{ \lambda_1 \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \lambda_2 \begin{pmatrix} 2 \\ 3 \end{pmatrix} \mid \lambda_i > 0, \lambda_1 + \lambda_2 = 1 \right\}$$

is an open polyhedron (in fact an open polytope), but not an open subset of $\mathbb{R}^2$. However, seen as a subset of the line through the points $\left(\begin{smallmatrix} 1 \\ 2 \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} 2 \\ 3 \end{smallmatrix}\right)$, it is open.

An excellent introduction to convex sets can be found in [11].

2.2 System Identification

Chapters 5 and 6 are devoted to the *system identification problem* for piecewise affine systems. In general, the system identification problem is the problem of mathematically describing the relation between a collection of signals as well as possible, given experimental sets of data from the signals. Here, as in most settings, we are interested in the relation between the input signals, $u(t)$, and the output signals, $y(t)$. We assume that the experimental data consists of two time-series, $u(t)$ and $y(t)$, $t = 1, \ldots, N$. For notational convenience, we will use the notation

$$y_{t_1}^{t_2} \triangleq \{y(t_1), y(t_1+1), \ldots, y(t_2)\}$$
$$Z^t \triangleq \{u_1^t, y_1^t\}$$

The goal is to describe $y(t)$ as a function of previous outputs and inputs, together with some noise, $e(t)$, i.e.,

$$y(t) = f(Z^{t-1}, e_1^t) \tag{2.5}$$

The noise is often assumed to be white, and should be independent of the previous input and output. In this thesis, we will consider the case

$$y(t) = f(Z^{t-1}) + e(t) \tag{2.6}$$

where $E[e(t)] = 0$ and $e(t)$ is independent of Z^{t-1}.

For a good introduction to system identification, see, e.g., [63].

2.2.1 Prediction Error Methods

A family of identification methods containing many well-known and commonly used approaches are the *prediction error methods (PEM)*. These are so-called parametric methods, which means that we search for the best model within a family of models, parametrised by a set of parameters θ. In our context, this means that we only consider a parametrised family of functions $f(Z^{t-1}, \theta)$ with a certain structure. The system identification problem then reduces to finding the parameter values θ^* that fit the model to the experimental data as well as possible. In the prediction error methods, this is done by considering the predicted output

$$\hat{y}(t|\theta) = f(Z^{t-1}, \theta)$$

and the residual

$$\varepsilon(t, \theta) = y(t) - \hat{y}(t|\theta)$$

Here, $\hat{y}(t|\theta)$ is our guess of the value of $y(t)$, given the information Z^{t-1}. Since $e(t)$ is independent of Z^{t-1}, the best guess we can obtain is what we get by replacing $e(t)$ in (2.6) by its expected value, which is 0. The residual $\varepsilon(t, \theta)$ can be seen as a measure of how close the guess was.

Then the model is fitted to the data by minimising a *criterion function*

$$V(\theta, Z^N) = \frac{1}{N}\sum_{t=1}^{N} \ell(\varepsilon(t,\theta)) \tag{2.7}$$

that is,

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, V(\theta, Z^N)$$

There are many possible choices for the function $\ell(\varepsilon)$. A very common choice is

$$\ell(\varepsilon) = \varepsilon^2$$

which gives the *least squares* criterion.

Example 2.5 (ARX model) *An ARX model is a model with the following structure:*

$$\begin{aligned} y(t) &= -a_1 y(t-1) - \cdots - a_{n_a} y(t-n_a) + \\ &\quad + b_1 u(t-1) + \cdots + b_{n_b} u(t-n_b) + e(t) = \\ &= \varphi^T(t)\theta + e(t) \end{aligned}$$

where

$$\begin{aligned} \varphi(t) &= \begin{pmatrix} -y(t-1) & \dots & -y(t-n_a) & u(t-1) & \dots & u(t-n_b) \end{pmatrix}^T \\ \theta &= \begin{pmatrix} a_1 & \dots & a_{n_a} & b_1 & \dots & b_{n_b} \end{pmatrix}^T \end{aligned}$$

The predicted output is

$$\hat{y}(t|\theta) = \varphi^T(t)\theta$$

so the least squares criterion is given by

$$V(\theta, Z^N) = \frac{1}{N}\sum_{t=1}^{N} \varepsilon^2(t) = \frac{1}{N}\sum_{t=1}^{N} (y(t) - \varphi^T(t)\theta)^2$$

which is minimised by

$$\hat{\theta} = \left(\sum_{t=1}^{N} \varphi(t)\varphi^T(t)\right)^{-1} \sum_{t=1}^{N} \varphi(t) y(t)$$

2.2.2 Numerical minimisation

In the previous example, it was possible to minimise $V(\theta, Z^N)$ analytically. In general, however, $V(\theta, Z^N)$ can be a (possibly nonconvex) function which has to be minimised using numerical algorithms. Some of the most important numerical algorithms are the *Newton* and *quasi-Newton algorithms*. These are iterative algorithms, that update the estimate of $\operatorname{argmin}_\theta V(\theta)$ according to

$$\hat{\theta}^{(i+1)} = \hat{\theta}^{(i)} - \alpha^{(i)} \left(R(\hat{\theta}^{(i)}) \right)^{-1} V'(\hat{\theta}^{(i)}) \tag{2.8}$$

where $V'(\hat{\theta}^{(i)})$ is the gradient of $V(\theta)$ evaluated in $\hat{\theta}^{(i)}$. The role played by $R(\hat{\theta}^{(i)})$ will soon be explained. $\alpha^{(i)}$ is a scalar that determines the *step size* of the update, i.e., how much the estimate $\hat{\theta}^{(i)}$ should change in each iteration.

If $V(\theta)$ is a positive definite quadratic function of θ, $V(\theta) = 0.5(\theta - \theta_{\min})^T P(\theta - \theta_{\min})$, we could get right to the minimum in one step using $\alpha^{(0)} = 1$ and $R(\theta) = V''(\theta)$, since

$$\hat{\theta}^{(1)} = \hat{\theta}^{(0)} - \left(V''(\hat{\theta}^{(0)}) \right)^{-1} V'(\hat{\theta}^{(0)}) = \hat{\theta}^{(0)} - P^{-1} P(\hat{\theta}^{(0)} - \theta_{\min}) = \theta_{\min}$$

If $V(\theta)$ is not quadratic, we can use the Taylor expansion to approximate it with a quadratic function. However, since the approximation may be good only in a small region around $\hat{\theta}^{(i)}$, we might need to adjust the value of $\alpha^{(i)}$ to ensure that $V(\hat{\theta}^{(i)})$ will actually decrease for increasing i. Methods where $\alpha^{(i)} \neq 1$ are sometimes called *damped*.

The Newton algorithms use $R(\theta) = V''(\theta)$, as described above. However, it might be time-consuming to compute the Hessian $V''(\hat{\theta}^{(i)})$ in each step, compared to computing only the gradient. Moreover, the Hessian is not necessarily positive definite when being far away from local minima. An alternative is therefore to use quasi-Newton algorithms, such as *Gauss-Newton*, that instead form an estimate of $V''(\theta)$, and use that in (2.8). The estimates can be constructed in such a way that they are always positive semidefinite, which can be used to guarantee convergence to a stationary point. For more details about numerical minimisation algorithms, see [63].

Using numerical minimisation algorithms like the ones described above means that there is a risk of getting stuck in local minima. One way of getting around this problem is to try different initial values for the algorithm. In this way we might at least find several local minima, from which we can choose the one with the lowest value of $V(\theta, Z^N)$. In the presence of noise and model errors, it also might not be absolutely necessary to find the global minimum, since we do not know if it gives the correct description of the system anyway. Nevertheless, we would like to find a good model, that gives a low value for $V(\theta, Z^N)$, so the question arises: How do we find a good solution (or an initial value that leads to a good local minimum)? This question will be treated for piecewise affine systems in Chapters 5 and 6.

2.2.3 Regularization

In the Newton and quasi-Newton algorithms, the update equation (2.8) contains the inverse $(R(\hat{\theta}^{(i)}))^{-1}$. Sometimes $R(\theta)$ may be ill-conditioned or even singular. To make it better conditioned, a term λI can be added to $R(\theta)$. This is known as *regularization*. One can also add a term $\lambda\|\theta - \theta^{\#}\|^2$ to the criterion function $V(\theta, Z^N)$ in (2.7), which will also have the effect of adding $2\lambda I$ to $V''(\theta, Z^N)$. This can be interpreted as providing a guess $\theta^{\#}$ of the value of θ, and trying to keep θ close to the guess. The coefficient λ can be used to weigh the importance of staying close to $\theta^{\#}$ against minimising the prediction error.

2.3 MILP/MIQP

In Chapter 6, some different piecewise affine system identification problems are formulated as *Mixed-Integer Linear Programs (MILP)* and *Mixed-Integer Quadratic Programs (MIQP)*. These are two types of optimisation problems that involve both discrete and continuous variables. In general, an MILP problem has the following structure:

$$\begin{aligned} \min_{x,\delta} \quad & p^T \begin{pmatrix} x \\ \delta \end{pmatrix} \\ \text{subj. to} \quad & C \begin{pmatrix} x \\ \delta \end{pmatrix} \leq d \\ & x \in \mathbb{R}^n,\ \delta \in \{0,1\}^m \end{aligned} \tag{2.9}$$

The MIQP problem is almost identical, except that the objective function is quadratic:

$$\begin{aligned} \min_{x,\delta} \quad & \begin{pmatrix} x^T & \delta^T \end{pmatrix} Q \begin{pmatrix} x \\ \delta \end{pmatrix} + p^T \begin{pmatrix} x \\ \delta \end{pmatrix} \\ \text{subj. to} \quad & C \begin{pmatrix} x \\ \delta \end{pmatrix} \leq d \\ & x \in \mathbb{R}^n,\ \delta \in \{0,1\}^m \end{aligned} \tag{2.10}$$

We assume that Q is positive definite or semidefinite. If it is only positive semidefinite, one could use regularization techniques as in Section 2.2.3 to make it positive definite.

As we can see, the only difference compared to an ordinary LP or QP (Linear or Quadratic Programming) problem is that the δ variables are discrete. However, although LP and QP problems can be solved efficiently using standard algorithms, it has been shown that solving an MILP/MIQP problem is $\mathcal{NP}$-hard [35, 93]. This means that there is no known algorithm solving the MILP/MIQP problems with polynomial worst-case complexity, and that if such an algorithm is found, several other hard problems would also be solvable in polynomial time. It is quite easy to

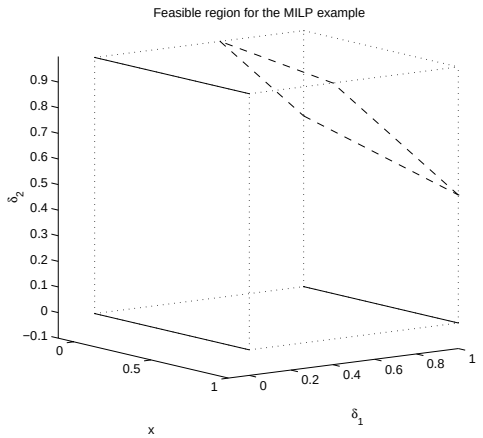


Figure 2.4: The feasible region for Example 2.6. The solid lines define the feasible region, the dotted lines show the bounds $0 \leq x \leq 1$, $0 \leq \delta \leq 1$, and the dashed lines show the hyperplane defined by $x + 5\delta_1 + 6\delta_2 = 9$.

find an algorithm, for which we in the worst case have to test all combinations of δ values, so the worst-case complexity is at most 2^m times the complexity of an LP/QP problem in n variables. However, there are algorithms that work better for many practical cases. Some solvers for the MILP/MIQP problems can be found in, e.g., [6, 25, 45, 84].

2.3.1 A Branch-and-Bound Algorithm

Let us take a closer look at (2.9) and (2.10). If we would replace the requirement $\delta \in \{0,1\}^m$ by $0 \leq \delta \leq 1$, the problems would become ordinary LP or QP problems. Similarly, if we fix some of the elements of δ to either 0 or 1 and relax the other elements, we would also get LP/QP problems. These combined relaxations and restrictions can be used to form a branch-and-bound algorithm, which will be illustrated by an example:

Example 2.6 (Branch-and-bound, depth first) *We would like to solve the problem*

$$\begin{aligned} \min_{x,\delta} \quad & J = x - 2\delta_1 - \delta_2 \\ \text{subj. to} \quad & x + 5\delta_1 + 6\delta_2 \leq 9 \\ & 0 \leq x \leq 1 \\ & \delta_1, \delta_2 \in \{0,1\} \end{aligned} \tag{2.11}$$

Figure 2.4 shows the feasible region for this problem. Let us start by solving the relaxed problem we get by replacing $\delta_1, \delta_2 \in \{0,1\}$ *by* $0 \leq \delta_1 \leq 1$, $0 \leq \delta_2 \leq 1$. *This*

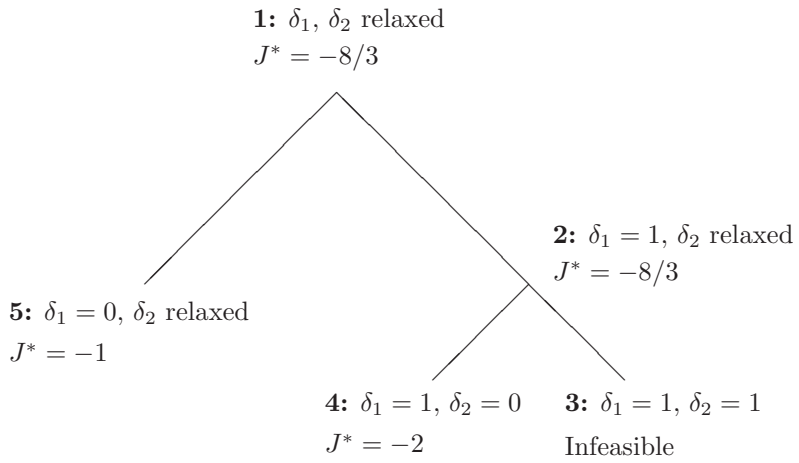


Figure 2.5: The search tree for Example 2.6.

is an LP problem with the solution

$$\begin{cases} J^* = -8/3 \\ x = 0 \\ \delta_1 = 1 \\ \delta_2 = 2/3 \end{cases}$$

In Figure 2.4, the feasible region for the relaxed problem is the three-dimensional region bounded by the cube and the plane $x + 5\delta_1 + 6\delta_2 = 9$. *Let us now fix one of the* δ *variables and solve the new, more restricted LP problem we get. From looking at the optimal solution to the relaxed problem, it seems natural to start by fixing* δ_1 *to* 1. *See Figure 2.5, where the relaxed problem corresponds to the root node, and the new problem corresponds to its right child. Note that, in general, since we restrict the domain over which we are minimising* J, *we can never get a lower value of* J *than the optimum we get for the relaxed case. In this special case, since the optimal value for* δ_1 *was* 1 *already in the relaxed case, we already know that* $J^* = -8/3$ *and* $\delta_2 = 2/3$ *is the optimal solution when* δ_1 *is set to* 1. *Hence we can continue directly by fixing* δ_2, *e.g., to* 1. *The problem has now become*

$$\begin{aligned} \min_{x,\delta} \quad & J = x - 2 - 1 \\ \text{subj. to} \quad & x + 5 + 6 \leq 9 \\ & 0 \leq x \leq 1 \end{aligned}$$

This problem is obviously infeasible, so instead we let $\delta_2 = 0$ and get

$$\begin{aligned} \min_{x,\delta} \quad & J = x - 2 \\ \text{subj. to} \quad & x + 5 \leq 9 \\ & 0 \leq x \leq 1 \end{aligned}$$

which has the optimal solution

$$\begin{cases} J^* = -2 \\ x = 0 \\ (\delta_1 = 1,\ \delta_2 = 0) \end{cases}$$

This optimum is our first feasible candidate so far for an optimum to the original problem. However, we have not yet excluded all possible combinations of values for δ. Therefore, we now let $\delta_1 = 0$, while δ_2 is relaxed. The problem to solve is

$$\begin{aligned} \min_{x,\delta} \quad & J = x - \delta_2 \\ \text{subj. to} \quad & x + 6\delta_2 \leq 9 \\ & 0 \leq x \leq 1 \\ & 0 \leq \delta_2 \leq 1 \end{aligned}$$

with the optimal solution

$$\begin{cases} J^* = -1 \\ x = 0 \\ \delta_2 = 1 \\ (\delta_1 = 0) \end{cases}$$

Apparently, this optimal point is worse than the feasible candidate point we got previously. Since we cannot get better values of J by restricting the domain further (i.e., fixing δ_2), we can conclude that our candidate point really is the optimum for the original problem, without having to test the remaining possibilities.

The procedure used in the previous example is summarised in the following algorithm, which is a version of a standard algorithm that can be found, e.g., in [98].

Algorithm 2.1 (Branch-and-bound, depth first)

Given: *A problem like* (2.9) *or* (2.10).
Initialisation: *Set $LB = -\infty$ and $UB = \infty$. Relax the problem by allowing all elements in δ to take any value in the interval $[0, 1]$. Start building on a tree structure by creating a root node. (Each level in the tree structure will correspond to one more element in δ being fixed; cf. Figure 2.5.)*

1. *Solve the current LP/QP problem. If it is infeasible, go to 4. If it is feasible, denote the optimal point by* x^*, δ^*, *and the optimal value by* J^*. *If* $\delta^* \in \{0,1\}^m$, *go to 3. Otherwise, go to 2.*
2. *If* $J^* > UB$, *the optimal solution will not be in this branch of the tree, so go to 4. Else, fix one of the relaxed elements in* δ *to* 0 *or* 1, *according to some strategy. Add one child to the current node in the tree and move to the child node. If all of the currently fixed elements of* δ *have already previously been fixed to their other possible values, and if* $J^* > LB$, *set* $LB = J^*$. *Go to 1.*
3. *If* $J^* < UB$, *set* $UB = J^*$, *and let* $x_{min} = x^*$ *and* $\delta_{min} = \delta^*$. *Go to 4.*
4. *Go up one level in the tree, i.e., relax the* δ *element that was last fixed. If the other possible value of that element has not yet been tested, fix the element to that value. (In other words, if the element was fixed to* 0 *and has not yet been fixed to* 1, *set it to* 1.*) Otherwise, go up one level at a time in the tree until an untested value of a* δ *element is found. Then go to 1. If the root node is reached and no untested value is found, we are done. Then set* $LB = UB$.

When these iterations are done, the optimal value of the objective function equals $LB = UB$. *If* $LB = UB = \infty$, *the problem is infeasible. Otherwise, the optimal point is given by* x_{min} *and* δ_{min}.

There are some remarks to be made. Firstly, the algorithm does not specify how to choose the element to be fixed in step 2. There are several possible strategies to use for this. In Example 2.6, the element of δ that was closest to an integer value was chosen. The thought behind this is to quickly reach a good feasible point for the original problem. With its help, entire subtrees can be ruled out as being obviously bad (or at least worse than the best value found so far), which may save a large amount of computations, just as in the example.

During the iterations of Algorithm 2.1, the variables UB and LB give an upper and a lower bound on the optimal value. This could be useful, e.g., if the computation is stopped before the search for the optimal value is completed. Then the best suboptimal point so far might be usable, together with the lower bound, that shows how far from the optimum we can be at most.

There are variants of this algorithm. For example, one can also traverse the search tree in a best-first manner, where the nodes with the best optimal values J^* are traversed first, or in a breadth-first manner, or in a combination of the three. There are also several other algorithms besides the branch-and-bound algorithms for solving the MILP/MIQP problems. A good reference is [98].

2.4 Separating Points with Hyperplanes

When investigating the computational complexity of the algorithms in Chapter 6, a question that will arise is: In how many ways can we group N points in $\mathbb{R}^n$ by separating them with M hyperplanes? To be able to answer this question

unambiguously, we need to restrict ourselves to only consider a certain kind of point sets. The terminology here partly follows [24].

Definition 2.1
A set of points $X \in \mathbb{R}^n$ is in general position*, if for every subset of points $x_1, \ldots, x_k \in X$, where $k \leq n+1$, the vectors*

$$\begin{pmatrix} 1 \\ x_1 \end{pmatrix}, \ldots, \begin{pmatrix} 1 \\ x_k \end{pmatrix}$$

are linearly independent.

This definition is equivalent to specifying that no subset of k points should be contained in an affine subspace of a lower dimension than $\min\{k-1, n\}$. For example, if we consider points in $\mathbb{R}^2$, no points should coincide, and three points should never be on a line. Note that, non-strictly speaking, for an arbitrarily chosen set of points in $\mathbb{R}^n$, this condition is satisfied with probability one.

Let us begin by determining in how many ways a set of points can be split by one hyperplane. This result will follow as a corollary to the following theorem, which is also found in, e.g., [24].

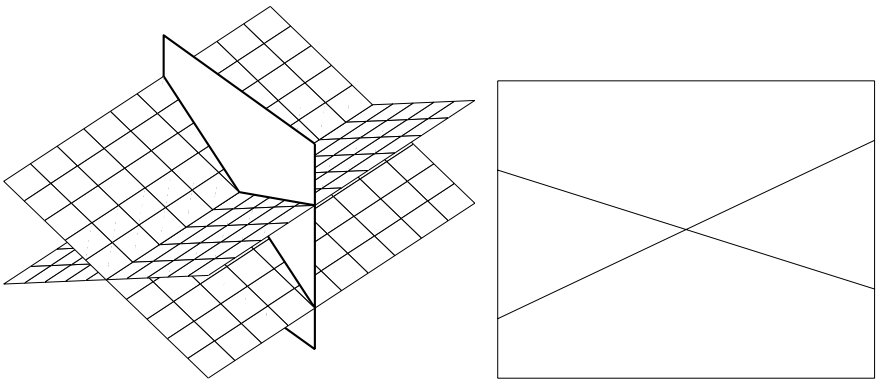
Theorem 2.1
Consider N hyperplanes in $\mathbb{R}^n$ going through the origin, such that the normal vectors of any subset consisting of n or fewer hyperplanes are linearly independent. Let $f(n, N)$ be the number of regions that $\mathbb{R}^n$ is divided into by the hyperplanes. Then $f(n, N)$ can be written as a sum of binomial terms

$$f(n, N) = 2 \sum_{k=0}^{n-1} \binom{N-1}{k} \tag{2.12}$$

for $n \geq 1$, $N \geq 1$ (we let $\binom{0}{0} = 1$ and $\binom{N}{k} = 0$ for $k > N$).

Proof We prove the theorem by induction. Clearly, $f(n, 1) = 2 = 2\sum_{k=0}^{n-1} \binom{0}{k}$ for $n \geq 1$, since one hyperplane divides the space into two half-spaces. We also have $f(1, N) = 2 = 2\binom{N-1}{0}$ for $N \geq 1$ (a one-dimensional line can only be divided into two half-lines, no matter how many points you put in the origin).

Now suppose that the theorem has been shown for all $\{(n, N) \mid n \leq n_0, N \leq N_0 - 1\}$, and suppose that we have placed $N_0 - 1$ hyperplanes in $\mathbb{R}^{n_0}$, and are about to place one more (see Figure 2.6(a)). The N_0th hyperplane is an $(n_0 - 1)$-dimensional subspace of $\mathbb{R}^{n_0}$, and will be cut by all the other $N_0 - 1$ hyperplanes. It will therefore be partitioned into $f(n_0 - 1, N_0 - 1)$ pieces (Figure 2.6(b)). But each of these pieces will be a border that divides one of the original regions into two new regions. Hence, the original $f(n_0, N_0 - 1)$ regions will increase in number



(a) Two planes crossing the origin, and a third to be placed.

(b) The new hyperplane will be partitioned into several pieces by the first planes. Each of these pieces will divide one of the old regions in Figure 2.6(a) into two new regions.

Figure 2.6: Illustration of Theorem 2.1.

by $f(n_0 - 1, N_0 - 1)$ when the N_0th hyperplane is placed in $\mathbb{R}^{n_0}$. This means that the new number of regions will be

$$f(n_0, N_0) = f(n_0, N_0 - 1) + f(n_0 - 1, N_0 - 1)$$

which means, according to the induction assumption,

$$\begin{aligned} f(n_0, N_0) &= 2 \sum_{k=0}^{n_0-1} \binom{N_0 - 2}{k} + 2 \sum_{k=0}^{n_0-2} \binom{N_0 - 2}{k} = \\ &= 2 + 2 \sum_{k=0}^{n_0-2} \left[\binom{N_0 - 2}{k+1} + \binom{N_0 - 2}{k} \right] = \\ &= 2 + 2 \sum_{k=0}^{n_0-2} \binom{N_0 - 1}{k+1} = \\ &= 2 \sum_{k=0}^{n_0-1} \binom{N_0 - 1}{k} \end{aligned}$$

and the theorem is proven. □

Corollary 2.1
A set of N points in general position in $\mathbb{R}^n$ can be partitioned in $f(n + 1, N)/2$ ways by a hyperplane.

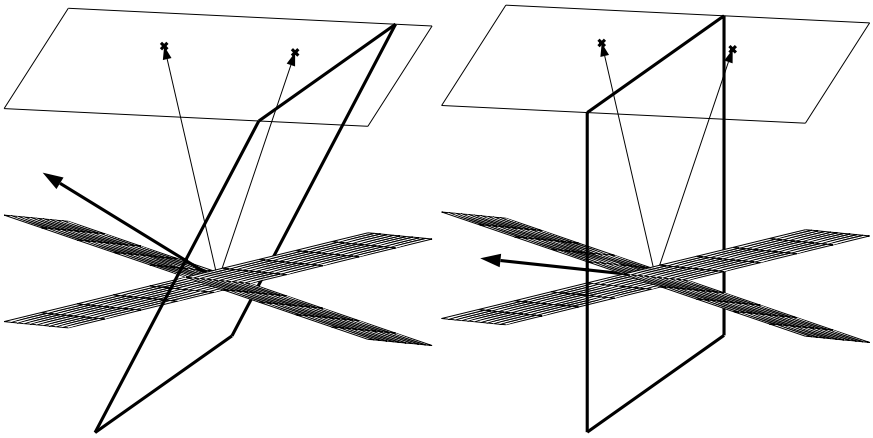


Figure 2.7: Illustration of the one-to-one correspondence in Corollary 2.1. The upper hyperplane is the affine subspace containing the given point set. Their associated vectors are normal vectors of hyperplanes dividing the space into different regions. The hyperplane drawn with thick lines partitions the point set into two subsets. We will get different partitions depending on in which region the normal vector of the thick-line hyperplane is.

Proof By considering the vectors

$$\begin{pmatrix} 1 \\ x_1 \end{pmatrix}, \ldots, \begin{pmatrix} 1 \\ x_N \end{pmatrix}$$

we can regard the points as being contained in an n-dimensional affine subspace of $\mathbb{R}^{n+1}$ (see Figure 2.7). The vectors are normal vectors to N hyperplanes, which by Theorem 2.1 divide $\mathbb{R}^{n+1}$ into $f(n+1, N)$ regions. Now consider an arbitrary hyperplane going through the origin of $\mathbb{R}^{n+1}$. Its normal vector can lie in one of two opposite regions (depending on how it is directed). There is a one-to-one correspondence between the set of opposite regions in which the normal vector of this hyperplane lies and how it* partitions the given points. To see this, note that each time the normal vector passes a boundary and enters a new region, the hyperplane also passes a point and gives rise to a new partition. This one-to-one correspondence proves the theorem. □

Now when we know in how many ways one hyperplane can divide a set of points, it is easy to extend the result to several hyperplanes. We make the restriction that no pair of hyperplanes should divide the point set in the same way; in that case, they could be replaced by one hyperplane. Since the trivial partitions obtained by letting all points lie on the same side of a hyperplane will be uninteresting in Chapter 6, we also exclude these cases. The remaining partitions are called *nontrivial*.

*Or rather its intersection with the affine subspace in which the given points are contained.

Corollary 2.2
A set of N points in general position in $\mathbb{R}^n$ can be partitioned in $\binom{f(n+1,N)/2-1}{M}$ nontrivial ways by M hyperplanes.

Proof From $f(n+1,N)/2-1$ nontrivial options of placing one hyperplane, we should choose M different ways to place the hyperplanes. □

2.5 Inverting a Scalar Piecewise Affine Function

In Chapter 6, we will also need to invert a scalar, continuous, piecewise affine function. The following lemma can be used for this purpose.

Lemma 2.1
Consider the function[†]

$$\begin{aligned} f(x) &= x - \alpha_0 + \sum_{i=1}^{M} \sigma_i \max\{\beta_i x - \alpha_i, 0\} \\ \beta_i &> 0, \quad \frac{\alpha_1}{\beta_1} < \frac{\alpha_2}{\beta_2} < \cdots < \frac{\alpha_M}{\beta_M} \end{aligned} \tag{2.13}$$

where $\sigma_i \in \{-1, 1\}$, and assume that f is strictly increasing, i.e.,

$$0 < \sum_{m=0}^{k} \sigma_m \beta_m < \infty \quad \forall k = 1, \ldots, M \tag{2.14}$$

Then the inverse of $f(x)$ is

$$\begin{aligned} f^{-1}(y) &= y + \alpha_0 - \sum_{k=1}^{M} \sigma_k \max\{\tilde{\beta_k} y - \tilde{\alpha_k}, 0\} \\ \tilde{\beta_k} &= \frac{\beta_k}{\sum_{m=0}^{k-1} \sigma_m \beta_m \sum_{m=0}^{k} \sigma_m \beta_m} \\ \tilde{\alpha_k} &= \frac{\sum_{m=0}^{k-1} \sigma_m (\beta_m \alpha_k - \beta_k \alpha_m)}{\sum_{m=0}^{k-1} \sigma_m \beta_m \sum_{m=0}^{k} \sigma_m \beta_m} \end{aligned} \tag{2.15}$$

Proof We prove the statement by induction over M. For $M = 0$, we have $f(x) = x - \alpha_0$, so $f^{-1}(y) = y + \alpha_0$. Now suppose that the statement holds for a certain $M-1 \geq 0$, and consider a function f with M breakpoints. For $x < \alpha_M/\beta_M$, f has only $M-1$ breakpoints, so in this region f^{-1} can be written as in (2.15), according to the induction assumption. For $x \geq \alpha_M/\beta_M$, we get

$$f(x) = \left(\sum_{m=0}^{M} \sigma_m \beta_m \right) x - \sum_{m=0}^{M} \sigma_m \alpha_m \tag{2.16}$$

[†]For the sake of a more compact notation, let $\sigma_0 = \beta_0 = 1$.

so

$$f^{-1}(y) = \frac{y + \sum_{m=0}^{M} \sigma_m \alpha_m}{\sum_{m=0}^{M} \sigma_m \beta_m} \tag{2.17}$$

Since $x = f^{-1}(y)$, we can rewrite the condition $x \geq \alpha_M/\beta_M$ as

$$\begin{aligned} & \frac{y + \sum_{m=0}^{M} \sigma_m \alpha_m}{\sum_{m=0}^{M} \sigma_m \beta_m} \geq \frac{\alpha_M}{\beta_M} \quad \Leftrightarrow \\ & \Leftrightarrow \quad \frac{1}{\sum_{m=0}^{M} \sigma_m \beta_m} \left(y - \frac{1}{\beta_M} \sum_{m=0}^{M-1} \sigma_m (\beta_m \alpha_M - \beta_M \alpha_m) \right) \geq 0 \\ & \Leftrightarrow \quad y \geq \frac{\tilde{\alpha}_M}{\tilde{\beta}_M} \end{aligned} \tag{2.18}$$

Now, for $y \geq \tilde{\alpha}_M/\tilde{\beta}_M$, by using simple but tedious calculations we can rewrite (2.17) as

$$\begin{aligned} f^{-1}(y) = & \frac{y + \sum_{m=0}^{M-1} \sigma_m \alpha_m}{\sum_{m=0}^{M-1} \sigma_m \beta_m} + \\ & + \frac{-\sigma_M \beta_M}{\sum_{m=0}^{M-1} \sigma_m \beta_m \sum_{m=0}^{M} \sigma_m \beta_m} \left(y - \frac{1}{\beta_M} \sum_{m=0}^{M-1} \sigma_m (\beta_m \alpha_M - \beta_M \alpha_m) \right) = \\ & = f_{M-1}^{-1}(y) - \sigma_M (\tilde{\beta}_M y - \tilde{\alpha}_M) \end{aligned} \tag{2.19}$$

where $f_{M-1}^{-1}(y)$ is the inverse of the given function without the Mth max function. Since this function equals $f(x)$ for $x < \alpha_M/\beta_M$, we get $f_{M-1}^{-1}(y) = f^{-1}(y)$ for $y < \tilde{\alpha}_M/\tilde{\beta}_M$. Furthermore, by using a max function and by the induction assumption we now can write

$$\begin{aligned} f^{-1}(y) &= f_{M-1}^{-1}(y) - \sigma_M \max\{\tilde{\beta}_M y - \tilde{\alpha}_M, 0\} = \\ &= y + \alpha_0 - \sum_{k=1}^{M-1} \sigma_k \max\{\tilde{\beta}_k y - \tilde{\alpha_k}, 0\} - \sigma_M \max\{\tilde{\beta}_M y - \tilde{\alpha}_M, 0\} = \\ &= y + \alpha_0 - \sum_{k=1}^{M} \sigma_k \max\{\tilde{\beta}_k y - \tilde{\alpha_k}, 0\} \end{aligned} \tag{2.20}$$

which is just what we wanted to prove. □

3

Piecewise Affine Systems

In Chapter 1, we saw some simple examples of different piecewise affine systems. This chapter will give a more detailed overview of piecewise affine systems, and of different topics related to them.

In the following section, some important classes of piecewise affine systems will be given. The remaining sections of the chapter give a short background about some important research areas, and set the stage for the rest of the thesis.

3.1 Different System Classes

In this section, some different classes and different descriptions of piecewise affine systems will be presented. We start by considering noiseless models, and treat models with noise in Section 3.1.7.

3.1.1 Continuous Time Models

A general continuous time piecewise affine system could be described by

$$\begin{aligned} \dot{x} &= A(v)x + B(v)u + b(v) \\ y &= C(v)x + D(v)u + d(v) \end{aligned} \tag{3.1}$$

where x is our state vector, u is the input to the system, and y is the system output. v is a *key vector*, describing in what *mode* (i.e., in what affine subsystem) the system is for the time being. v can be a function of x, t, u, or some other external input. We assume that v can only take a finite number of different values.

The systems described by (3.1) include systems which are not always called piecewise affine systems. A common restriction (see for example [47]) is that v only depends on x. Also in this thesis, the general form (3.1) will not be considered, but only subclasses of (3.1); mainly the class of *switched systems* described next.

3.1.2 Switched Systems

The *switched systems*, as defined, e.g., in [31], constitute an important subclass of the systems described by (3.1). The restrictions compared to (3.1) are that $u = 0$ and that the modes (and hence v) are determined by a finite automaton, driven by discrete signals generated when the state vector x reaches different hyperplanes, here called *switching surfaces (or hyperplanes)*, and by discrete external inputs.

The systems can be structured according to Figure 3.1. The continuous affine subsystems are located in the plant, for which the dynamics is given by

$$\dot{x} = A(v)x + b(v)$$

The controller is purely discrete and is modelled as a finite automaton. Its output symbols $\tilde{v}$ are translated by the actuator into key vector signals v for the plant.

The generator contains information about a set of hyperplanes

$$\mathcal{H}_i = \{x \mid C_i x = d_i\}$$

and generates logical input symbols for the controller according to

$$\tilde{e}_{si} = \begin{cases} \texttt{true} & x \in \mathcal{H}_i \\ \texttt{false} & x \notin \mathcal{H}_i \end{cases}$$

Altogether, the system will behave like an affine system as long as there is no external input and the state does not reach one of the switching hyperplanes $\mathcal{H}_i$. When doing so, or receiving an external input, the value of v changes and the system switches to another mode, i.e., to another affine subsystem.

In Chapter 4, we will, for notational purposes, mostly consider a special case of switched systems. However, it should be noted that most of the techniques in Chapter 4 can be applied to the more general kind of switched systems described above. The systems considered will be in the form

$$\dot{x} = A(v)x + b(v) \ , \quad x \in X(v) \ , \quad v \in \{-1, 0, 1\}^M \tag{3.2}$$

where the state-space is partitioned into different regions $X(v)$ by M hyperplanes. The key vector v is a function of x, and has one element for each separating hyperplane. The elements of v are 1 or -1, depending on on which side of the

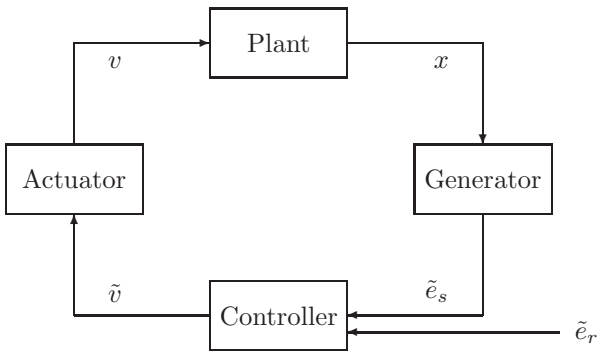


Figure 3.1: The structure of a switched system.

hyperplanes x is situated, or 0 if x lies in a hyperplane. In this way, v has a one-to-one relationship to $X(v)$. This also implies that the dynamics of a trajectory $x(t)$ just depends on x, not on t or on any external input. The different regions $X(v)$ will be polyhedra (see Section 2.1).

3.1.3 Discrete Time Models

The descriptions in Section 3.1.1 are continuous time models. It is also common to use discrete time models for the piecewise affine systems. A general form could be

$$\begin{aligned} x(t+1) &= A(v)x(t) + B(v)u(t) + b(v) \\ y(t) &= C(v)x(t) + D(v)u(t) + d(v) \end{aligned} \tag{3.3}$$

Other notable model structures for discrete time piecewise affine systems include *MLD (Mixed Logical Dynamical)* models [7], *MMPS (Max-Min-Plus-Scaling)* models, *LC (Linear Complementary)* models, and *ELC (Extended Linear Complementary)* models [39]. Essentially, these classes are the same as (3.3), where v is a piecewise constant function of x and u, and is constant over polyhedra in the combined state-space and input space (this is shown in [5] (for MLD), [39] (all classes)).

3.1.4 Models in Regression Form

In Chapters 5 and 6, we will use models in *regression form*, i.e.,

$$y(t) = \varphi^T(t)\theta(v) \tag{3.4}$$

where the key vector v only can take a finite number of values, and $\theta(v)$ is a function of v. The vector φ is our *regression vector*, which could for instance consist of old inputs and outputs, e.g.,

$$\varphi(t) = \begin{pmatrix} 1 & -y(t-1) & \dots & -y(t-n_a) & u(t-1) & \dots & u(t-n_b) \end{pmatrix}^T \tag{3.5}$$

Here we have included a constant 1 in the regression vector, to be able to express piecewise affine systems in the form (3.4). If we would not include the constant, (3.4) would be a piecewise linear system. Throughout this thesis, we will assume that the first element of $\varphi(t)$ equals 1. We will also assume that the key vector v is uniquely determined by the regression vector $\varphi(t)$, and that the regions of the different affine subsystems are polyhedral. When including additive white noise in the model (see Section 3.1.7), this kind of systems has been called *PWARX (PieceWise AutoRegressive eXogenous)* systems [8, 34]. These systems are a subclass of the systems described by (3.3), and can easily be transformed into that form by using the elements of $\varphi(t)$ (except the constant) as state variables.

However, there are also many other possible choices of φ. In general, $\varphi(t)$ could be any function of u and y, i.e.,

$$\varphi(t) = \begin{pmatrix} f_1(u,y) & \dots & f_n(u,y) \end{pmatrix}^T$$

In this case, a system in the form (3.4) could be called a *PWNARX (PieceWise Nonlinear AutoRegressive eXogenous)* system. This is of course not a piecewise affine system in general, but as we will see, most of the theory in Chapter 6 also applies to systems with this choice of $\varphi(t)$.

3.1.5 Chua's Canonical Representation and Hinging Hyperplane Models

A special class of piecewise linear functions are the functions that can be represented by the so-called *canonical representation* introduced by Chua and Kang [20, 54]:

$$y(t) = \alpha_0^T \varphi(t) + \sum_{i=1}^{M} c_i |\alpha_i^T \varphi(t)| \tag{3.6}$$

where $\varphi(t)$ is given from (3.5). As can be seen, the functions described by Chua's canonical representation are continuous in $\varphi(t)$. It turns out (see [20]) that every scalar, continuous, piecewise linear function of one variable can be uniquely expressed in this form with $\alpha_{i2} = 1$, i.e., as

$$y(t) = \alpha_{01} + \alpha_{02} x(t) + \sum_{i=1}^{M} c_i |\alpha_{i1} + x(t)|$$

(here we let $\varphi(t) = \begin{pmatrix} 1 & x(t) \end{pmatrix}^T$). However, not every multivariable, continuous, piecewise linear function can be expressed on the canonical representation (see Example 3.2). A necessary condition for this is that the regions of the different linear subsystems must be separated by hyperplanes. In [19], a necessary and sufficient condition for a function to be expressible in the canonical form is given. There are also other canonical representations covering larger classes of systems.

One of them is Güzeliş' model class [37], where two nested absolute values are allowed:

$$y(t) = \alpha_0^T \varphi(t) + \sum_{i=1}^{M} b_i |\alpha_i^T \varphi(t)| + \sum_{j=1}^{K} c_j \left| \gamma_j^T \varphi(t) + \sum_{i=1}^{M} d_{ij} |\alpha_i^T \varphi(t)| \right| \tag{3.7}$$

In [61], it is shown that every continuous piecewise linear function can be written using (possibly deeply) nested absolute values.

Hinging hyperplane functions were introduced by Breiman [14] as a model class for function approximation and classification. They are composed of a sum of *hinge functions*, which are two half-planes joined continuously together at the *hinge*. The hinging hyperplane functions can be written as

$$y(t) = \sum_{i=1}^{M} \pm \max\{{\beta_i^+}^T \varphi(t), {\beta_i^-}^T \varphi(t)\} \tag{3.8}$$

The $\pm$ signs play essentially the same role as c_i in (3.6), and are needed to allow for both convex and nonconvex functions. Since we can write

$$\begin{aligned}
&\max\{{\beta_i^+}^T \varphi(t),\, {\beta_i^-}^T \varphi(t)\} = \\
&\quad = \frac{1}{2}(\beta_i^+ + \beta_i^-)^T \varphi(t) + \\
&\qquad + \max\Big\{ \big(\beta_i^+ - \frac{1}{2}(\beta_i^+ + \beta_i^-)\big)^T \varphi(t),\, \big(\beta_i^- - \frac{1}{2}(\beta_i^+ + \beta_i^-)\big)^T \varphi(t) \Big\} = \\
&\quad = \frac{1}{2}(\beta_i^+ + \beta_i^-)^T \varphi(t) + \frac{1}{2} \max\Big\{ (\beta_i^+ - \beta_i^-)^T \varphi(t),\, -(\beta_i^+ - \beta_i^-)^T \varphi(t) \Big\} = \\
&\quad = \frac{1}{2}(\beta_i^+ + \beta_i^-)^T \varphi(t) + \frac{1}{2} \Big| (\beta_i^+ - \beta_i^-)^T \varphi(t) \Big|
\end{aligned}$$

it is easy to see that (3.6) and (3.8) define equivalent classes of functions. Another representation for the same class, which will be used here, and which can be transformed to Chua's or Breiman's form in a similar way, is

$$y(t) = \varphi^T(t)\theta_0 + \sum_{i=1}^{M^+} \max\{\varphi^T(t)\theta_i, 0\} - \sum_{i=M^+ +1}^{M} \max\{\varphi^T(t)\theta_i, 0\} \tag{3.9}$$

Here the piecewise affine function is written as a sum of an affine term and M (positive and negative) max functions. Instead of the $\pm$ signs of (3.8), the positive max functions have been collected in one sum, and the negative into another. However, we will use the shorthand notation

$$y(t) = \varphi^T(t)\theta_0 + \sum_{i=1}^{M} \pm \max\{\varphi^T(t)\theta_i, 0\} \tag{3.10}$$

thereby meaning exactly the same as in (3.9).

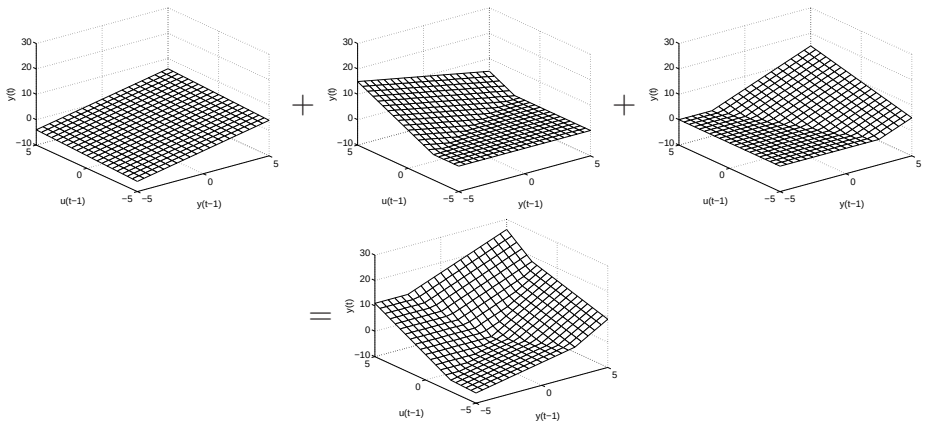


Figure 3.2: Example of a hinging hyperplane function. The upper leftmost function is the linear function in the first row of (3.11), and the middle and right diagrams correspond to the two max functions. The resulting function is shown in the lower diagram.

Example 3.1 (Hinging hyperplane model) *Consider Figure 3.2, where the following hinging hyperplane model is illustrated:*

$$\begin{aligned} y(t) = y(t-1) + 0.2u(t-1) + \\ + \max\{-y(t-1) + 2u(t-1), 0\} + \\ + \max\{2y(t-1) + u(t-1), 0\} \end{aligned} \tag{3.11}$$

If we express the function in Chua's canonical representation, it could take the following form:

$$\begin{aligned} y(t) = 1.5y(t-1) + 1.7u(t-1) + \\ + |-0.5y(t-1) + u(t-1)| + \\ + |y(t-1) + 0.5u(t-1)| \end{aligned}$$

In Breiman's form the same function can be written:

$$\begin{aligned} y(t) = \max\{2.2u(t-1), y(t-1) + 0.2u(t-1)\} + \\ + \max\{2y(t-1) + u(t-1), 0\} \end{aligned}$$

Example 3.2 *The following system (see Figure 3.3) cannot be expressed in the hinging hyperplane form:*

$$y(t) = \begin{cases} y(t-1) & \text{if } y(t-1) \geq 0,\ y(t-1) + u(t-1) \geq 0 \\ -y(t-1) & \text{if } y(t-1) < 0,\ -y(t-1) + u(t-1) \geq 0 \\ u(t-1) & \text{if } y(t-1) + u(t-1) < 0,\ -y(t-1) + u(t-1) < 0 \end{cases}$$

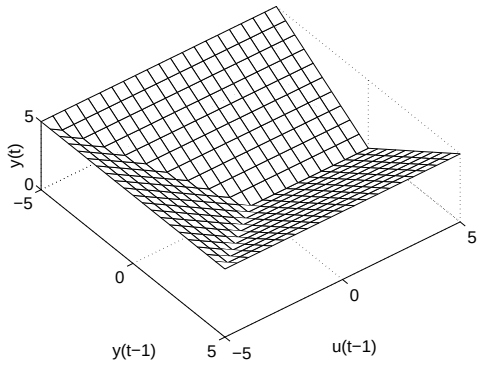


Figure 3.3: A piecewise affine system not expressible in the hinging hyperplane form.

This is due to the fact that the regions of the different subsystems are not separated by hyperplanes (but half-hyperplanes), so there is no way to place the hinges in the hinging hyperplane representation to get this partition of the state-space.

A property of the class described by (3.6), (3.8), and (3.9), which makes it suitable for function approximation, is that it has universal approximation properties [60]. Roughly speaking, this means that we can approximate any function arbitrarily well by using sufficiently many hinge functions (i.e., by letting $M \to \infty$).

We can note that the parameters of (3.9) are not uniquely determined, since for example

$$\varphi^T\theta_0 + \max\{\varphi^T\theta_1, 0\} = \varphi^T(\theta_0 + \theta_1) + \max\{-\varphi^T\theta_1, 0\}$$

This means that we can replace $\max\{\varphi^T\theta_i, 0\}$ by $\max\{-\varphi^T\theta_1, 0\}$, include the difference in the linear term, and still have the same system. Another reason for the ambiguity of (3.9) is that we can reorder the max functions. One way to at least partially avoid this ambiguity is to require that $0 \leq w^T\theta_1 \leq \cdots \leq w^T\theta_{M^+}$ and $0 \leq w^T\theta_{M^++1} \leq \cdots \leq w^T\theta_M$ for some constant (nonzero) vector w.

3.1.6 Representations with Fixed Regions

In function approximation applications, it is quite common to use piecewise affine models where the state-space is partitioned into a regular pattern of regions, with an affine function for each region. For example, in [52], a representation called *HL CPWL (High Level Canonical PieceWise Linear)* representation is used, which is based on a partition of the state space into simplices (polytopes with $n+1$ corners, where n is the dimension).

3.1.7 Noise

So far, we have not included any noise in our models. There are several ways to model the noise. A common way in linear modelling is to model the noise as independent, identically distributed stochastic variables with zero mean (white noise), filtered through a linear filter. In nonlinear modelling, there are several ways in which this could be extended. For example, the noise could be filtered through a nonlinear filter, it could enter the system in a multiplicative way (e.g., we could have products like $u(t)e(t)$, where u is the input and e is noise), etc. As mentioned in Section 2.2, in this thesis we are going to restrict the noise models to an additive term

$$y(t) = f(Z^{t-1}) + e(t)$$

where $E[e(t)] = 0$ and $e(t)$ is independent of Z^{t-1}. However, in Chapter 4, some terms could be interpreted as multiplicative noise, as we will see.

3.1.8 State Jumps

In this thesis, the state is assumed to move continuously in the continuous time models. However, there are situations when it is natural to allow state jumps, like in the following example.

Example 3.3 (Docking spacecraft) *Consider two spacecraft with unit mass, flying along a line in free space. The control signals are the forces acting on them. The system can be described by the following equations, where x_1 is the location and x_2 the velocity of the first spacecraft, and x_3 and x_4 the location and velocity of the second spacecraft, respectively:*

$$\dot{x} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} x + \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} u \tag{3.12}$$

However, if the two spacecraft dock with each other, the new dynamics becomes

$$\dot{x} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} x + \begin{pmatrix} 0 & 0 \\ 1/2 & 1/2 \\ 0 & 0 \\ 1/2 & 1/2 \end{pmatrix} u \tag{3.13}$$

Furthermore, we must ensure that $x_2 = x_4$ after the docking, i.e., that the velocities of the two spacecraft are equal. Denoting the values just before the docking by x_2^-, x_4^-, and the values just after the docking by x_2^+, x_4^+, preservation of the momentum gives $x_2^+ = x_4^+ = (x_2^- + x_4^-)/2$. This means that the state may jump when the mode switching occurs. (Note also that the dimension of the state-space can be reduced in the second mode, which would lead to state jumps between different state-spaces.)

3.2 Control and Analysis

Since piecewise affine systems, like other hybrid systems, are highly nonlinear, they might be difficult to analyse. Several approximating methods for analysis and control design have therefore been developed for different classes of hybrid systems.

An excellent overview of different approaches to modelling and control using multiple models (e.g., piecewise affine systems) is given in [72]. Some approaches for optimal control of hybrid systems can be found in [7, 13, 64, 74]. Other sub-optimal approaches have also been proposed, such as [69, 77]. In [56], supervisory control based on a discrete approximation of the system is studied. Controllability problems have been considered, e.g., in [5, 89].

In recent years, some stability analysis results using Lyapunov theory have appeared [12, 27, 47, 48, 59, 76, 83].

3.3 Verification

For control design, typical requirements might include that the system should never reach some specific states, that the system should reach a certain region in the state-space, and/or that there should be invariant regions (once you get there, you will never leave the region). After the design process, one would like to make sure that the specified requirements are satisfied. This process is known as *verification*.

Solving a verification problem exactly is in general possible only for some restricted classes of hybrid systems [1, 3]. Many verification methods for the problem of avoiding bad states found in the literature are based on computing a conservative approximation of the system [1, 2, 9, 21, 22, 23, 31, 57, 92]. This means that either the model of the system is replaced by a (computationally) simpler model, which is an *outer approximation* of the original system, or that the trajectories from a given initial set of states are replaced by an outer approximation. An outer approximation is an approximation that guarantees that if a trajectory is allowed by the original system, it is also allowed in the simplified model. In this way it can be guaranteed that if a certain bad state is never reached in the simple model, it is never reached in the original system either. A good overview over numerous different approaches is given in [32].

For the problem of assuring that a certain region is reached, an *inner approximation* can be used analogously to what is described above [31]. If a transition is guaranteed to occur in an inner approximation, it is also guaranteed to occur in the original system. However, other techniques might be needed to compute the inner approximation.

3.3.1 Robust Verification

The verification method presented in Chapter 4 is partly built upon a method presented in [31]. For this method, we will consider systems in the form (3.2). To verify the desired properties, the behaviour of the vector field $\dot{x}(t)$ at the borders of the regions $X(v)$ is analysed. Specifically, questions such as "At a given face of

the polyhedron $X(v)$, is there a point, x_0, such that $\dot{x}_0$ is pointing out of $X(v)$, or are all trajectories at this face going into $X(v)$?" are answered (this kind of computations has also been used by others, e.g., by [47]). The information obtained is used to determine which transitions between different regions are possible, which transitions are guaranteed to occur non-deterministically (i.e., one transition out of a set of transitions from a given polyhedron is guaranteed to occur) and which are not. Then finite automata are constructed, showing the guaranteed or possible transitions. The finite automata give an approximation of the system, and can be used for different kinds of verification. For example, we can guarantee that certain states in the original system are not reachable from some other initial states, by proving that there is no sequence of possible transitions in the finite automata, taking the system state from the region of the initial states to the region of the final states.

Example 3.4 (Verification) *Consider the simple system*

$$\dot{x} = \begin{cases} -2x & x < -2 \\ -x + 1 & -2 \leq x \leq 2 \\ x - 3 & x > 2 \end{cases}$$

Suppose that we would like to make sure that if $x(0) \geq -2$, $x(t)$ will never get below -2. This can easily be verified by considering the trajectories at $x = -2$. Here, $\dot{x} = -x + 1 = -(-2) + 1 = 3$, which means that if x gets close to -2 (from above), it will increase, and hence never pass the border $x = -2$.

By making calculations like above, we can construct a finite automaton showing what transitions between the three regions are possible (see the left automaton in Figure 3.4). This automaton is an outer approximation of the original system. Apart from the property shown above, we can see, e.g., that once we get into the region $|x| < 2$ we never leave it.

We can also construct an automaton which is an inner approximation of the system (the right automaton in Figure 3.4). With the help of this automaton we can guarantee that we will end up in the region $|x| < 2$, if starting with $x \leq 2$.

Like all other methods mentioned above, the method in [31] assumes that a model of the system is given. It would be desirable to be able to perform the verification, and simultaneously compute how sensitive the verification proof (e.g., the approximating automata) is to changes in the underlying systems, both in the dynamics and in the switching surfaces. This is what is called *robust verification* in this thesis. Such information could be used to get a measure of how robust the verification process is to model errors, or as an aid in a control design process, if we would like to adjust the system dynamics without losing the verified property. Sometimes we would only be interested in that some crucial transitions should not change, whereas in other cases we might want the entire approximating automata to remain invariant.

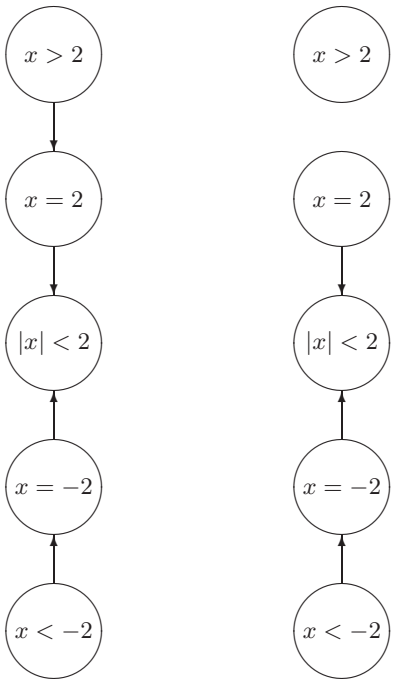


Figure 3.4: Automata showing possible (left figure) and guaranteed (right figure) transitions for the system in Example 3.4.

Since the approximating method considers the behaviour of $\dot{x}(t)$ at the borders of the regions $X(v)$, we must determine how this behaviour changes with varying $A(v)$ and $b(v)$, and with translations of the surfaces that bound $X(v)$. How this can be done is the topic of Chapter 4.

3.4 Identification

Identification of hybrid systems (e.g., piecewise affine systems) is an area that is related to many other research fields within nonlinear system identification. For example, one can find several different methods and approaches which are applicable, or at least related to the piecewise affine system identification problem. Some examples of approaches that result in piecewise affine systems are neural networks with piecewise affine perceptrons [4, 36, 58], Chua's canonical representation and hinging hyperplanes [14, 19, 20, 51, 53, 54, 79], self-exciting threshold autoregressive (SETAR) models for time-series analysis [66, 67], and some function approximation approaches [50, 52]. In [86], which is a good overview of different nonlinear identification techniques, the relations between several different approaches are explored.

In Chapters 5 and 6, we will mostly consider models in regression form like

in (3.4). It turns out that once the partitioning of the state-space is known, the piecewise affine system identification problem reduces to a problem, comparable to a linear system identification problem in terms of complexity. However, finding the best partition may be a very complex problem. Hence, there are two fundamental approaches: Either an a priori partition can be used, or the partitioning can be estimated along with the different subsystems. The latter can be done simultaneously or iteratively. The first approach gives a simple estimation process, but the number of regions needed to give enough flexibility in the model structure may be very large. In the second approach, the number of subsystems can be kept low, but the estimation will be more complex. This dilemma will be treated more in detail in Chapters 5 and 6.

3.4.1 Approximating Nonlinear Systems

As previously mentioned, many classes of piecewise affine systems have universal approximation properties, which make them suitable for approximating arbitrary nonlinear functions. Therefore, an efficient identification method for piecewise affine systems would also be of interest for identification of nonlinear functions. In this case, the exact shape of the regions is of less direct importance, since the true system does not consist of affine subsystems. Instead, being able to approximate the nonlinear function well, using few parameters and a representation that is easy to handle, becomes the main question.

4

Robust Verification

As mentioned in the previous chapter, the problem of verification is to prove that a set of bad states is always avoided, or alternatively that a set of target states is actually reached. These types of questions often arise in applications in connection with safety issues. Verification of discrete systems is a well-established research field, with a broad range of applications. With the growing interest of hybrid systems in the last decade, verification of hybrid systems has also found a number of applications in control. One application area where verification methods for hybrid systems have been applied is the chemical industry, where safety is an important issue [32]. Another application example is the verification of some properties of the landing gear system of a Swedish aircraft [31, 73]. Verification has also been used for special cases of a collision avoidance system for air-traffic around airports [62].

In this chapter, the main question will be the one of proving that some states are never reached. As most system models contain uncertainties in one way or the other, an important question is how large uncertainties can be tolerated before the verified properties can no longer be guaranteed to hold. The system may also be disturbed by noise, which must be taken into account in the verification. These two problems will be addressed.

We will consider systems, where the state-space is partitioned into several polyhedral regions, with one affine subsystem for each region. The mathematical definition of the system class is done in Section 4.1. As in Example 3.4, we will study the behaviour of the trajectories at the boundaries of the regions. From this infor-

mation, a finite automaton which is an outer approximation of the original system can be constructed, analogously to what was done in Example 3.4. The concept of inner and outer approximations was explained in Section 3.3. We will not consider how the discrete verification problem is performed, given the approximating automaton. This is described in more detail in, e.g., [31]. Instead we will assume that we are given a set of transitions, for which we should prove if they will possibly occur or not. In other words, we expect to be given a guarantee of the kind: "If these transitions never occur, then the set of bad states will never be reached". In the example in Section 4.8, we will see that it can sometimes be quite easy to find such sets of transitions by direct inspection. The problems solved in this chapter are formulated in Section 4.2 and Section 4.3.

The methods can also be extended to find an inner approximation of the system behaviour, and hence to show that a set of states will really be reached in finite time. How this can be done is studied in Section 4.7.1. In Section 4.7.2, we will show that the methods work for switched systems as well.

It would also be nice to be able to combine the verification with other objectives. Section 4.8 shows an example of how this could be done.

4.1 Some Notation and Assumptions

The systems considered in this chapter are in the form (3.2), that is,

$$\dot{x} = A(v)x + b(v), \quad x \in X(v) \ , \quad v \in \{-1, 0, 1\}^M \tag{4.1}$$

where $A(v) \in \mathbb{R}^{n\times n}$, $b(v) \in \mathbb{R}^n$ are given for all $v \in \{-1, 1\}^M$ such that $X(v)$ is nonempty (this will be explained in more detail below). The vector v is a key vector, which is a piecewise constant function of x and can be seen as a label for each region $X(v)$. The regions $X(v) \subset \mathbb{R}^n$ are polyhedra, separated from each other by M hyperplanes. These are defined by

$$\{x \in \mathbb{R}^n \mid C_i x = d_i\}, \quad i = 1, \ldots, M \tag{4.2}$$

where $C_i \in \mathbb{R}^{1\times n}$ and $d_i \in \mathbb{R}$ are given. To allow for a compact representation, we let C_i form the rows in an $M \times n$ matrix C, and collect d_i into the vector d.

Now v is defined according to the following rule:

$$v_i = \begin{cases} -1 & \text{if } C_i x < d_i \\ 0 & \text{if } C_i x = d_i \\ 1 & \text{if } C_i x > d_i \end{cases} \tag{4.3}$$

In this way there is a one-to-one relationship between v and $X(v)$. Since $v_i = 0$ implies that $X(v) \subseteq \{x \in \mathbb{R}^n \mid C_i x = d_i\}$, we notice that a nonempty $X(v)$ will be contained in an $(n-1)$-dimensional affine subspace of $\mathbb{R}^n$ if and only if there is at least one zero entry in v. Nonempty polyhedra $X(v)$ corresponding to vectors v with no zero entries will be called *full-dimensional* polyhedra. The corners of

a full-dimensional polyhedron will be regions $X(v^c)$, where v^c contains (at least) n zeros. Here we will only treat the case when v^j contains exactly n zeros; other cases can be regarded as degenerate special cases of this, with several corners in the same point.

As mentioned, we assume that $A(v)$ and $b(v)$ are given for all full-dimensional polyhedra $X(v)$. For the faces, corners etc., we assume that the dynamics is given as a convex combination (whose weights might be unknown) of the dynamics of all adjacent full-dimensional polyhedra. The following example clarifies this assumption.

Example 4.1 *Consider two adjacent full-dimensional polyhedra $X(v^1), X(v^2) \in \mathbb{R}^2$, the face $X(v^f)$ between them, and the corners $X(v^{c_1})$, $X(v^{c_2})$ at the ends of the face. Figure 4.1 shows three different ways in which the trajectories could be directed, and the corresponding outer approximating automata.*

In Figure 4.1(a), the trajectories from $X(v^1)$ are pointing towards $X(v^f)$, while the trajectories in $X(v^2)$ are going away from $X(v^f)$. Since the dynamics of $X(v^f)$ is a convex combination of the two, this implies that the trajectories will go through $X(v^f)$ in the direction from $X(v^1)$ to $X(v^2)$. If the transition from $X(v^1)$ to $X(v^f)$ takes place at the right corner $X(v^{c_2})$, we might get a transition to the corner, since the trajectories at $X(v^f)$ always have a component pointing to the right.

In Figure 4.1(b), the trajectories from both regions $X(v^1)$ and $X(v^2)$ are all going inwards towards $X(v^f)$. This means that we will have a chattering *along the line. According to our assumption, the state could go either to $X(v^1)$ or $X(v^2)$ from $X(v^f)$, but as soon as it leaves $X(v^f)$, it will be immediately pushed back. In the automaton we approximate this behaviour by stating that there cannot be any transitions at all from $X(v^f)$ to $X(v^1)$ or $X(v^2)$. However, since all trajectories are pointing to the right, we know that the state will eventually end up in $X(v^{c_2})$.*

Figure 4.1(c) is quite similar to Figure 4.1(b) in the sense that we will get a chattering behaviour along $X(v^f)$, but here we do not know in which corner (if any) we will end up, $X(v^{c_1})$ or $X(v^{c_2})$.

4.2 Problem Formulation for Known Systems

Now consider one of the full-dimensional polyhedra $X(v)$, and how the trajectories $x(t)$ behave inside it. Let $X(v')$ be one of the faces of the polyhedron (which means that $v'_i = 0$ for some i, and $v'_j = v_j$, $j \neq i$). Basically, there are three different options for the qualitative behaviour of the trajectories near $X(v')$ (see Figure 4.2):

1. They are all exiting $X(v)$ (some may be parallel to $X(v')$).
2. They are all entering $X(v)$ (some may be parallel to $X(v')$).
3. Some trajectories are entering and some are exiting $X(v)$.

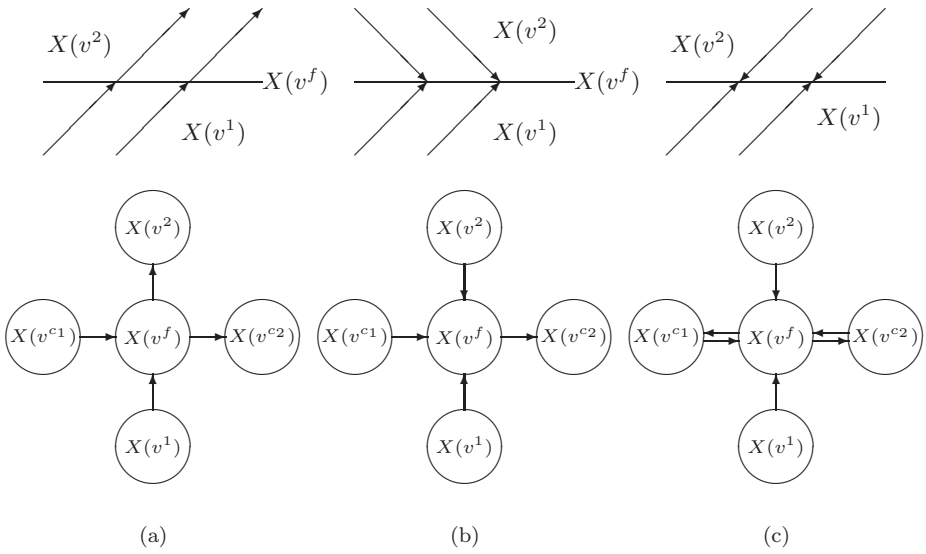


Figure 4.1: Three different cases for the dynamics at a boundary $X(v^f)$, and the corresponding outer approximating automata. $X(v^{c1})$ is the left corner of $X(v^f)$, and $X(v^{c2})$ is the right corner.

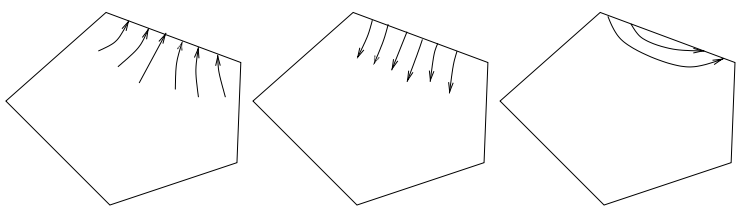


Figure 4.2: Three options for the behaviour of the trajectories in the vicinity of a polyhedron face.

Since the system is affine inside the polyhedron, the trajectories are smooth, and therefore these three cases are the only possible options. The special case of all trajectories being parallel to $X(v')$ is included in both the first cases, but this is of little practical importance.

The three different cases will lead to different approximating automata (see Figure 4.3). What we need to find out is therefore which of the cases we get, given $A(v)$, $b(v)$. From (4.3), it is not hard to see that C_i is a normal vector of the polyhedron face $X(v')$. Using this together with (4.1), we can rewrite the three different cases as

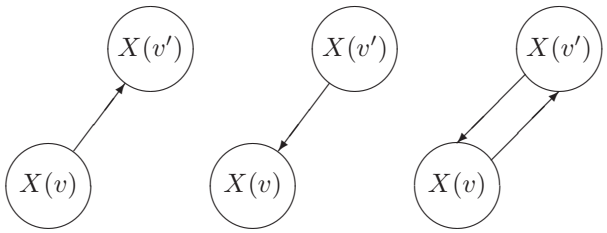


Figure 4.3: Parts of the automata corresponding to the three cases in Figure 4.2.

1. $C_i(A(v)x + b(v)) \geq 0$ for all $x \in X(v')^*$.
2. $C_i(A(v)x + b(v)) \leq 0$ for all $x \in X(v')$.
3. $C_i(A(v)x^1 + b(v)) > 0$ and $C_i(A(v)x^2 + b(v)) < 0$ for some $x^1, x^2 \in X(v')$.

These conditions can all be easily checked for given $A(v)$, $b(v)$, C and d, for example by finding the maximum and minimum values on $X(v')$ of $C_i(A(v)x+b(v))$, yielding two LP problems.

4.3 The Problem of Robust Verification

In Section 4.2, we assumed that the dynamics of the system was completely known. In practice, however, there will almost always be model errors, and the systems might be disturbed by noise. To that end, let us introduce some uncertainty in our system model:

$$\dot{x} = (A(v) - \Delta(v))x + b(v) - \delta(v), \quad x \in X(v) \ , \quad v \in \{-1, 0, 1\}^M \tag{4.4}$$

where v is defined by

$$v_i = \begin{cases} -1 & \text{if } C_i x < d_i + \gamma_i \\ 0 & \text{if } C_i x = d_i + \gamma_i \\ 1 & \text{if } C_i x > d_i + \gamma_i \end{cases} \tag{4.5}$$

Depending on the applications, the matrices $\Delta(v) \in \mathbb{R}^{n\times n}$, $\delta(v) \in \mathbb{R}^n$ and $\gamma \in \mathbb{R}^M$ can be viewed either as uncertainties in the model, or as matrices of our choice (the latter case can occur, e.g., in the control design process). We can see that the different matrices affect the system dynamics in different ways: $\Delta(v)$ and $\delta(v)$ affect the dynamics of the different subsystems, while γ affects the partitioning of the state-space.

*Note that this condition corresponds to the case "All trajectories are exiting $X(v)$" only if C_i is pointing out of $X(v)$, that is, if $v_i = -1$. Otherwise conditions 1 and 2 are switched. To avoid this, one could replace condition 1 by "$v_i C_i(A(v)x + b(v)) \leq 0$ for all $x \in X(v')$", and similarly for condition 2. However, this is mainly a matter of taste.

We will only allow values of γ that do not affect the topology of the state-space partition compared to the case $\gamma = 0$. In other words, the regions $X(v)$ will always have the same number of faces and the same neighbours as they would have if the separating hyperplanes were defined by $Cx = d$. This also means that we will not allow regions to disappear or new regions to be created when changing γ. An equivalent way of stating this assumption is to say that for each corner $X(v^c)$, v^c should remain constant, and not be changed because a γ value causes the corner to cross another hyperplane.

Example 4.2 (Preserving topology) *Consider the polyhedral partition of the state-space in Example 2.1. The partition can be described using the matrices*

$$C = \begin{pmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 0 \end{pmatrix}, \quad d = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}$$

The open polytope corresponding to P in Example 2.1 can now be written as $X(v)$, where $v = \begin{pmatrix} -1 & -1 & 1 \end{pmatrix}^T$ (P is the closure of this open polytope).

Now assume that we do not know where the real location of the hyperplane corresponding to the third row of C and d (so far modelled by $x_1 = -1$) is. We can include this uncertainty in the model by writing $x_1 = -1 + \gamma_3$. Now, if $\gamma_3 > 1.5$, we can see that the hyperplane is moved to the right of the corner where the other two hyperplanes meet. In this way, the polytope P disappears, and a new polytope is created. To avoid these kinds of phenomena, we can require that $\gamma_3 < 1.5$.

Assuming (4.4), the three cases from the previous section now become

1. $C_i[(A(v) - \Delta(v))x + b(v) - \delta(v)] \geq 0$ for all $x \in X(v')$.
2. $C_i[(A(v) - \Delta(v))x + b(v) - \delta(v)] \leq 0$ for all $x \in X(v')$.
3. $C_i[(A(v) - \Delta(v))x^1 + b(v) - \delta(v)] > 0$ and $C_i[(A(v) - \Delta(v))x^2 + b(v) - \delta(v)] < 0$ for some $x^1, x^2 \in X(v')$.

An interesting question is: How much could $\Delta(v)$, $\delta(v)$ and γ change, without changing the qualitative behaviour at each face of the polyhedron, i.e., without one face switching from, say, case 1 to case 3? In other words, for what values of $\Delta(v)$, $\delta(v)$ and γ do the different cases occur? This is the main question of this chapter.

Depending on how many of the parameters are varied simultaneously, this question may be more or less difficult to answer. If all of them are varied, the problem is a nonconvex quadratic problem, and will just be mentioned shortly. In the following section the question is answered for some different combinations of parameter variations, and an example is given in Section 4.8. We will mostly consider the two first cases (all trajectories exiting/entering the polytope), since the solution sets of these problems will turn out to be the easiest to describe. The solution sets for the third case can be obtained, either as the complement of the union of the other solution sets, or in some problems as a union of hyperplanes (see [80]).

4.4 Solutions to the Robust Verification Problems

Let us now consider the problem corresponding to case 1 of the previous sections. Rearranging the terms, we can write it as

$$C_i(A(v)x + b(v)) \geq C_i(\Delta(v)x + \delta(v)) \quad \text{for all } x \in X(v') \tag{4.6}$$

This form has a natural interpretation: On the left hand side we have the nominal flow through the face of the polyhedron, and the right hand side is the part of the flow that is affected by the variable matrices $\Delta(v)$ and $\delta(v)$. What (4.6) tells us is that the variable flow (the one caused by $\Delta(v)$ and $\delta(v)$) must be made small enough, or that $X(v')$ must lie in a region where the variable flow is small enough; otherwise we will get a total flow in the other direction from what was specified.

In the following subsections we solve this problem for some special cases, when at least one of $\Delta(v)$, $\delta(v)$ and γ is zero. Problem 2 can be solved completely analogously. Problem 3 is considered in a special case in Section 4.4.1.

4.4.1 $\Delta(v) = 0$, $\gamma = 0$, $\delta(v)$ is Varied

The simplest problem arises when we only allow $\delta(v)$ in (4.4) to vary, and let $\Delta(v)$ and γ equal 0. Equation (4.6) then takes the form

$$C_i(A(v)x + b(v)) \geq C_i\delta(v) \quad \text{for all } x \in X(v') \tag{4.7}$$

This problem can be solved immediately by solving the LP problem

$$\begin{aligned} \min_x \quad & C_i(A(v)x + b(v)) \\ \text{subj. to} \quad & x \in \overline{X(v')} \end{aligned} \tag{4.8}$$

where $\overline{X(v')}$ is the closure of $X(v')$. Denoting the solution of the LP problem by x_*, the solution set to (4.7) is

$$S_\delta = \{\delta \in \mathbb{R}^n \mid C_i\delta \leq C_i(A(v)x_* + b(v))\} \tag{4.9}$$

Problem 2 is solved by maximising instead of minimising $C_i(A(v)x+b(v))$ to get the limit x^*, and the solution set consists of all δ satisfying $C_i\delta \geq C_i(A(v)x^*+b(v))$. The values of δ for which $C_i\delta$ lies in between x_* and x^* are solutions to problem 3. Thus, in this case all solution sets will be convex.

4.4.2 $\gamma = 0$, $\Delta(v)$ and $\delta(v)$ are Varied

When letting both $\Delta(v)$ and $\delta(v)$ vary, we need to find a direct representation (see Section 2.1) of $X(v')$ to get a solution to (4.6). For the case when $X(v')$ is bounded, the direct representation will be on the following form:

$$X(v') = \{\sum_{j=1}^{r} \lambda_j x^j \mid \lambda_j \in \mathbb{R},\ \lambda_j > 0,\ \sum_{j=1}^{r} \lambda_j = 1\} \tag{4.10}$$

Here $x^j \in \mathbb{R}^n$, $j = 1, \ldots r$ are the corners of $X(v')$.

According to Equation (2.4), when $X(v')$ is unbounded, we must include in the direct representation some vectors, $x^{r+1}, \ldots, x^{r+h}$, which are parallel to the unbounded edges of $X(v')$. A direct representation of $X(v')$ would then be:

$$X(v') = \{\sum_{j=1}^{r+h} \lambda_j x^j \mid \lambda_j \in \mathbb{R},\ \lambda_j > 0,\ \sum_{j=1}^{r} \lambda_j = 1\} \tag{4.11}$$

Note that, as in (2.4), $\lambda_{r+1}, \ldots, \lambda_{r+h}$ are not included in the set of λ_j that should sum up to one; they can be arbitrarily large. In some cases there are no corners, e.g., when $X(v')$ consists of an entire hyperplane. As mentioned in Section 2.1, in such cases, a "dummy" corner must be introduced somewhere in $X(v')$.

For notational simplicity, let us drop the argument v of $A(v)$ etc. for a while. Now the following theorem gives the solutions to our problem.

Theorem 4.1
Consider the system given by (4.4) *and* (4.5). *Assume that* $\gamma = 0$. *Then the set of solutions to the problem* (4.6) *is given by*

$$\begin{aligned} S_{\Delta\delta} = \{(\Delta, \delta) \quad \mid \quad & C_i(Ax^j + b) \geq C_i(\Delta x^j + \delta), \quad j = 1, \ldots, r; \\ & C_i A x^{r+j} \geq C_i \Delta x^{r+j}, \quad j = 1, \ldots, h\} \end{aligned} \tag{4.12}$$

Proof To show that the inequality (4.6) is satisfied for an arbitrary point $x \in X(v')$, we use the direct representation (4.11):

$$\begin{aligned} C_i(Ax + b) &= C_i(A\sum_{j=1}^{r+h} \lambda_j x^j + b) = \\ &= \sum_{j=1}^{r} \lambda_j C_i(Ax^j + b) + \sum_{j=r+1}^{r+h} \lambda_j C_i A x^j \geq \\ &\geq \sum_{j=1}^{r} \lambda_j C_i(\Delta x^j + \delta) + \sum_{j=r+1}^{r+h} \lambda_j C_i \Delta x^j = \\ &= C_i(\Delta \sum_{j=1}^{r+h} \lambda_j x^j + \delta) = \\ &= C_i(\Delta x + \delta) \end{aligned}$$

To see that the first r conditions of (4.12) are necessary, we just need to notice that they are precisely (4.6) for the corners of $X(v')$. If a corner would not satisfy (4.6), then for continuity reasons, there would be points in $X(v')$ not satisfying (4.6) either. To show the necessity of the last h conditions, fix Δ and δ, suppose

that $C_i A x^{r+j} < C_i \Delta x^{r+j}$ for some j, and study $x = x^1 + \lambda x^{r+j} \in X(v')$, where $\lambda > \frac{C_i(Ax^1+b-\Delta x^1-\delta)}{C_i(\Delta x^{r+j}-Ax^{r+j})} > 0$. We then have

$$\begin{aligned} C_i(Ax+b) &= C_i(Ax^1+b) + \lambda C_i A x^{r+j} = \\ &= C_i(Ax^1+b) + \lambda C_i(Ax^{r+j} - \Delta x^{r+j}) + \lambda C_i \Delta x^{r+j} < \\ &< C_i(Ax^1+b) - C_i(Ax^1+b-\Delta x^1-\delta) + \lambda C_i \Delta x^{r+j} = \\ &= C_i(\Delta x^1 + \delta) + \lambda C_i \Delta x^{r+j} = \\ &= C_i(\Delta x + \delta) \end{aligned}$$

where the direction of the inequality follows since λ is multiplied by a negative number. This means that (Δ, δ) is outside the solution set, and the necessity is shown. □

When $X(v')$ is a polytope, Theorem 4.1 can interpreted as follows: The inequality (4.6) holds for all $x \in X(v')$ if and only if it holds for the corners of $X(v')$. This property follows directly from the inequality being linear in x.

Note that the solution set $S_{\Delta\delta}$ is a polyhedron in the space $\mathbb{R}^{n\times n} \times \mathbb{R}^n$, and therefore convex.

4.4.3 Multiple Requirements

So far, we have only been looking at one single polyhedron face. In most cases, the requirements may stipulate that several transitions of an approximating automaton should remain invariant. This case is easily handled by partitioning the problem into subproblems of the form treated above, and then taking the intersection of the solution sets as the solution set for the entire problem. How many transitions we need to consider will depend on the system and what we want to verify. For example, if all we are interested in is keeping the state on one side of a hyperplane, we only need to consider transitions through this hyperplane. It should be noted that considering fewer transitions will lead to a larger – and therefore less conservative – solution set, and will also require less computations.

4.4.4 $\Delta = 0$, δ and γ are Varied

The problem gets more complicated as soon as γ is not fixed anymore. We immediately notice from (4.4) and (4.5) that the regions $X(v)$ will no longer be fixed, but vary with γ. We also have to consider several regions $X(v)$ simultaneously, since moving a hyperplane will affect all regions adjacent to it. As mentioned in Section 4.3, we will only allow values of γ that keep the topology of the state-space regions invariant. If we do not make this requirement, new regions – for which we do not know the system dynamics – can be created by moving the hyperplanes defined by $Cx = d + \gamma$.

With this requirement, and with $\Delta(v) = 0$ for all v, the problem is still convex, as will be shown in the following. What we need to do first is to express the

corners as functions of γ. Having done that, we can use these functions to express the topology preservation requirements and the properties to be verified.

First, remember that each corner x^j of the full-dimensional polyhedron $X(v)$ is itself a region $X(v^j)$, where v^j contains n zeros. The zeros of v^j correspond to the equations $C_i x^j = d_i + \gamma_i$ that x^j satisfies. To be able to pick out the corresponding rows of C_i, d_i, and γ_i, we need to introduce some new matrices.

Let $D_{[v^j]} = \mathrm{diag}(v^j)$. From $D_{[v^j]}$ we then construct $Q_{[v^j]} \in \mathbb{R}^{(M-n)\times M}$ by deleting all rows containing only zeros. Similarly, we define $P_{[v^j]} \in \mathbb{R}^{n\times M}$ by deleting all rows in $I - D^2_{[v^j]}$ containing only zeros.

Now $P_{[v^j]}$ has the following property: When multiplying another matrix from the left by $P_{[v^j]}$, it picks out the rows corresponding to the zero entries of v^j. $Q_{[v_j]}$, on the other hand, picks out the rows not picked out by $P_{[v^j]}$, and furthermore multiplies the rows corresponding to the -1 entries of v^j by -1.

With this notation, we can pick out the equalities in (4.5) by writing

$$P_{[v^j]} C x^j = P_{[v^j]}(d + \gamma)$$

Since x^j is uniquely determined by the equalities in (4.5), $P_{[v^j]}C$ will always be invertible. Hence, we can write

$$x^j = (P_{[v^j]}C)^{-1} P_{[v^j]}(d + \gamma). \tag{4.13}$$

Example 4.3 *Consider once again the partition in Examples 2.1 and 4.2. The corner* $x^j = \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}$ *corresponds to the key vector* $v^j = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T$. *Hence,*

$$D_{[v^j]} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad Q_{[v^j]} = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}, \quad P_{[v^j]} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

Assume now that the positions of the hyperplanes are not certain, but that they could be translated by γ. *By* (4.13), *we can write the corner* x^j *as*

$$\begin{aligned} x^j &= (P_{[v^j]}C)^{-1} P_{[v^j]}(d + \gamma) = \\ &= \left[\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 0 \end{pmatrix} \right]^{-1} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \left(\begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} + \gamma \right) = \\ &= \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}^{-1} \left(\begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix} \right) = \\ &= (1 + \gamma_1) \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix} + \gamma_2 \begin{pmatrix} 1/2 \\ -1/2 \end{pmatrix} \end{aligned}$$

Note that if $\gamma = 0$, *we retain the original point.*

The demand that the topology should be preserved is the same as saying that it should always be possible to express each corner x^j as a region $X(v^j)$, where v^j must be constant. In other words, the equalities and inequalities (4.5) that define v^j should remain invariant. Of course, the equalities are satisfied (x^j is constructed from them in (4.13)). Thus, we get the following set of inequalities for each corner $X(v^j)$ and all $i = 1, \ldots, M$:

$$\begin{aligned} C_i(P_{[v^j]}C)^{-1}P_{[v^j]}(d+\gamma) &< d_i + \gamma_i \quad \text{if} \quad v_i^j = -1 \\ C_i(P_{[v^j]}C)^{-1}P_{[v^j]}(d+\gamma) &> d_i + \gamma_i \quad \text{if} \quad v_i^j = 1, \end{aligned} \tag{4.14}$$

or more compactly

$$Q_{[v^j]}\left(C(P_{[v^j]}C)^{-1}P_{[v^j]} - I\right)(d+\gamma) \succ 0 \tag{4.15}$$

where $\succ$ denotes componentwise inequality.

Example 4.4 *For the point x^j in 4.3, the inequality* (4.15) *becomes*

$$\begin{aligned} 0 < &\begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \cdot \\ &\cdot \left[\begin{pmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 0 \end{pmatrix} \left[\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 0 \end{pmatrix} \right]^{-1} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} - \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \right] \cdot \\ &\cdot \left(\begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} + \gamma \right) = \\ = &\frac{3}{2} + \frac{1}{2}\gamma_1 + \frac{1}{2}\gamma_2 - \gamma_3 \end{aligned}$$

Here we can note that, if $\gamma_1 = \gamma_2 = 0$, we get back the requirement that was imposed on γ_3 in Example 4.2.

What we need to do now is to take care of the requirements on the flow through the surfaces. This can be done completely analogously to Section 4.4.2, but with $\Delta = 0$. However, we need to plug in the expression for the corners into the inequalities of (4.12), to get, e.g.,

$$\begin{aligned} &C_i(A(P_{[v^j]}C)^{-1}P_{[v^j]}(d+\gamma) + b) \geq C_i\delta, \quad j = 1, \ldots, r; \\ &C_i A x^{r+j} \geq 0, \quad j = 1, \ldots, h \end{aligned} \tag{4.16}$$

for problem 1. Since the surfaces are only translated, the directions of the unbounded edges do not change, so x^{r+j} are not affected by γ. Inequalities like these, together with (4.15), give the final solution set. As can be seen, all inequalities are linear in δ and γ, and the resulting solution set is therefore a polyhedron.

4.4.5 Δ, δ and γ are Varied

The final case, when all parameters are allowed to vary, is quite similar to the one when only Δ is fixed. Like in Section 4.4.4 we get the inequalities (4.15). We also get the requirements on the flow through the surfaces of a region by plugging in the expression (4.13) for x^j into (4.12):

$$\begin{aligned} &C_i \left((A-\Delta)(P_{[v^j]}C)^{-1}P_{[v^j]}(d+\gamma)+b-\delta\right) \geq 0, \quad j=1,\ldots,r; \\ &C_i A x^{r+j} \geq C_i \Delta x^{r+j}, \quad j=1,\ldots,h \end{aligned} \tag{4.17}$$

However, here the inequalities become quadratic, and the solution set is nonconvex. This makes it harder to efficiently represent and work with the solution set, and therefore this case will not be further discussed.

4.5 Interpretations

Perhaps the most obvious interpretation is to view $\Delta(v)$, $\delta(v)$ and γ in (4.4) and (4.5) as uncertainties due to model errors and/or noise. The methods in Section 4.4 then provide bounds for the uncertainties for the requirements of the approximating automata to hold. For natural reasons, the bounds may be very asymmetric, indicating that the system is more sensitive to certain types of model errors than to others.

The problem formulation is quite general in that no structure of $\Delta(v)$, $\delta(v)$ or γ is assumed. It should be stressed that these uncertainties by no means need to be constant over time; we have showed that as long as they are kept within the computed bounds, the verification will hold, no matter how the uncertainties vary. The only structure that is assumed is the topology of the different polyhedral regions $X(v)$. If the uncertainty has some further structure, we can parametrise $\Delta(v)$ and $\delta(v)$ accordingly, thereby reducing the dimensionality and simplifying the problem. For example, if only the upper left element of $A(v)$ is uncertain, we can write $\Delta(v)$ as

$$\Delta(v) = \begin{pmatrix} \Delta_1 & 0 \\ 0 & 0 \end{pmatrix}$$

Another example is to set $\Delta(v) = 0$ as in Section 4.4.1, which can be interpreted as a model with additive noise:

$$\dot{x} = A(v)x + b(v) - \delta(v), \quad x \in X(v)\ , \quad i=1,\ldots,N \tag{4.18}$$

A nonzero $\Delta(v)$ can also be interpreted as multiplicative noise.

Another type of parametrisation is used in the example in Section 4.8. In this parametrisation, some of the elements of $\Delta(v)$ and $\delta(v)$ are common to several polyhedra.

An alternative interpretation is to consider $\Delta(v)$, $\delta(v)$ and/or γ as parameters of our choice, to be used for control design. One natural parametrisation would

then be $\gamma = 0$, $\delta(v) = 0$, $\Delta(v) = B(v)L(v)$, where $B(v)$ are fixed vectors that depend on the system, while we can choose $L(v)$ freely. In this way we get (piecewise) linear state feedback control, and the problem becomes that of finding the linear state feedback vectors $L(v)$ that make our system fulfil the requirements on the approximating automata. Another parametrisation would be the one in Section 4.4.4, where we can regard γ as a vector that lets us place the switching surfaces of a controller in an optimal manner.

4.6 Computational Complexity

In this section, some aspects on the computational complexity will be discussed. However, no rigorous analysis is made.

The computational complexity depends on what parameters we let vary. As we could see in Section 4.4.1, when only δ is varied we just need to solve one or two LP problems for each requirement. This can be done efficiently (see, e.g., [28]).

When only γ is fixed, the complexity gets a bit worse. From Section 4.4.2 we see, that once we know a direct representation of $X(v')$, it is trivial to divide $\mathbb{R}^{n\times n} \times \mathbb{R}^n$ into the three solution sets corresponding to problem 1, 2 and 3. Conversely, if we want the solutions to be written as intersections of half-spaces (as in (4.12)), we need to know the direct representation of $X(v')$. Therefore the computational complexity for this problem is essentially identical to that of finding the direct representation. Unfortunately, the number of vectors needed in such a representation grows very quickly with the size of the problem. An upper bound for the number of corners in a polyhedron can be calculated in the following way: In a corner, n linearly independent faces meet (where n is still the dimension of the state-space). Since the polyhedron has m faces, the number of corners cannot be larger than $\binom{m}{n}$.

However, if we restrict ourselves to the case where the polyhedra are formed by the state-space being divided by hyperplanes (as we have done in this chapter), it is fairly easy (but still quite time-consuming) to calculate the direct representation of all the polyhedra once and for all. The total number of corners is then bounded above by $\binom{M}{n}$, where M is the number of separating hyperplanes.

Finally, letting δ and γ vary leads to a similar set of inequalities as in the previous case. However, we cannot find the corners directly, since they are functions of γ. Instead we can store the inverses $(P_{[v^j]}C)^{-1}$ for each corner. Thus, we will need n times as large storage, and about the same amount of computations as when δ and Δ vary.

4.7 Extensions

The considered methods can be extended in different ways. We will consider two kinds of extensions: Using inner instead of outer approximations (that can be used to prove that a certain region of the state-space really is reached in finite time), and extending the model class to switched systems.

4.7.1 Inner Approximations

So far, we have only dealt with the behaviour of the trajectories at the border of each region. As we have seen, using the analysis of this behaviour we can construct outer approximating finite automata, with the help of which we may be able to guarantee that some states are never reached. Another kind of verification, that one might be interested in, is to prove that a certain region really is reached. This can be harder, since it is not enough to consider only what happens at the borders of a region. The reason for this is that the state, being in a region $X(v)$, might never get to the border of $X(v)$ if the system has a stable equilibrium or a limit cycle inside $X(v)$. Another difficulty with this kind of verification is that even if we know that the state trajectory leaves the region $X(v)$, we might not know through which face it will exit, since this information may be lost when approximating the system with a discrete automaton. We only know that one out of a set of transitions will occur. Following the terminology of [31], we say that the transitions are *guaranteed to occur nondeterministically*.

In this thesis, we will not go into detail with how the analysis of the resulting nondeterministic approximating automata is performed. For more details about this, see [31]. Instead, a sufficient condition for the trajectories to be guaranteed to exit $X(v)$ will be given. This condition also comes from [31]:

Proposition 4.1
A sufficient condition for the state trajectory to exit a polyhedron, $X(v)$, is that the normal vector, C_i, of one of its faces, $X(v')$, satisfies

$$v_i C_i \dot{x} < 0, \quad \forall x \in \overline{X(v)}. \tag{4.19}$$

Proof See [31]. □

Intuitively, the condition in Proposition 4.1 means that the velocity vector is always pointing towards the hyperplane $C_i x = d_i$ (or $C_i x = d_i + \gamma_i$ for a system with uncertainties). Now, given the face $X(v')$, we can include this case in our framework with the following theorem:

Theorem 4.2
Consider the system given by (4.4) *and* (4.5). *Let a direct representation of $X(v)$ be given by*

$$X(v) = \{\sum_{j=1}^{s+k} \lambda_j x^j \;\big|\; \lambda_j \in \mathbb{R},\ \lambda_j > 0,\ \sum_{j=1}^{s} \lambda_j = 1\} \tag{4.20}$$

Assume that $\gamma = 0$. Then, for a given $X(v')$, the set of solutions to (4.19) *is given by*

$$\begin{aligned} S_{\Delta\delta} = \{(\Delta, \delta) \quad \big| \quad & v_i C_i (A x^j + b) < v_i C_i (\Delta x^j + \delta), \quad j = 1, \ldots, s; \\ & v_i C_i A x^{s+j} < v_i C_i \Delta x^{s+j}, \quad j = 1, \ldots, k\} \end{aligned} \tag{4.21}$$

If we assume instead that $\Delta = 0$, the set of solutions is given by

$$S_{\delta\gamma} = \{(\delta,\gamma) \quad \Big| \quad \begin{array}{l} C_i(A(P_{[v^j]}C)^{-1}P_{[v^j]}(d+\gamma)+b) \geq C_i\delta, \quad j=1,\ldots,s; \\ C_iAx^{s+j} \geq 0, \quad j=1,\ldots,k \\ Q_{[v^c]}\left(C(P_{[v^c]}C)^{-1}P_{[v^c]}-I\right)(d+\gamma) \succ 0, \quad \forall \text{ corners } v^c\} \end{array} \tag{4.22}$$

Proof Analogous to the proof of Theorem 4.1 and the results in Section 4.4.4. □

Note that since in Theorem 4.2 a face $X(v')$ must be given, the condition is somewhat more restrictive than in Proposition 4.1, where it is sufficient that any of the faces satisfies (4.19).

4.7.2 Switched Systems

The methods described in this chapter can also be extended to switched systems, as defined in Section 3.1.2. Looking back at the equations, we can see that what is required is that we have a polyhedral partition of the state-space, and affine systems within each region. We should also be able to approximate the system by inner and outer approximating finite automata.

In the case of switched systems, there may be several possible systems in each region, depending on the input v to the plant. This means that the approximating automata will also need to have several states for each region, namely one state for each possible affine subsystem in that region. Switchings between these states can only occur if a border of a polyhedral region is reached, or if an external input is received. If there is no external input, essentially nothing is changed in our analysis methods above. However, if the system can receive an external input signal at any time, which causes switchings between a set of different affine subsystems, the situation is a bit different. The properties to be verified then need to be checked for all the subsystems reachable by only giving external input signals.

In the following section, an example of a switched system will be considered.

4.8 An Example: A Chemical Reactor

To demonstrate the properties of this kind of problems and solutions, we can look at a simple example. In [30], a (fictional) chemical reactor is modelled, and a control strategy is proposed, after which some properties are verified. Here we assume that some of the parameter values are uncertain, and try to determine how large errors can be tolerated before the verification is not valid any more.

4.8.1 System Model

A picture of the chemical reactor is shown in Figure 4.4. It consists of a tank containing a mixture of two fluids. When a certain temperature is reached, an

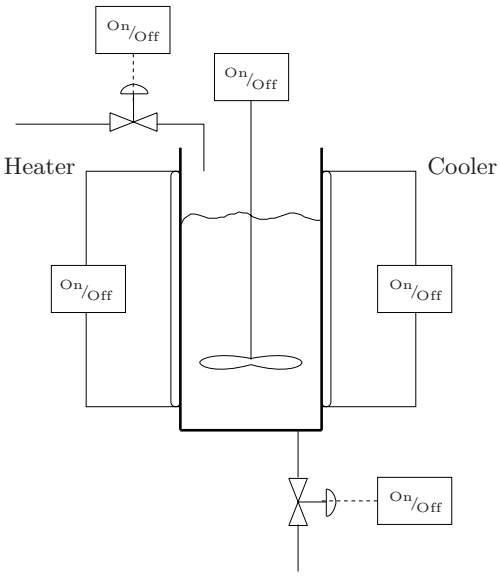


Figure 4.4: A schematic picture of the chemical reactor.

exothermal reaction between the two fluids starts, giving the desired product. The temperature can be controlled by a heater and a cooler. There is also a blender helping to mix the fluid. The mixture is provided through an inflow valve. There is also a draining valve. The valves can be either open or closed.

The system model derived in [30] has two continuous state variables: the fluid level x_1 and the temperature x_2. Furthermore, there are six control signals, each one taking a value in $\{0, 1\}$. They are described in Table 4.1. It could be worth mentioning that u_r is an artificial, uncontrollable signal that indicates whether or not the reaction is in progress.

Table 4.1: Inputs to the chemical reactor

Signal	Interpretation
u_b	blender signal
u_i	inflow valve signal
u_d	draining valve signal
u_h	heater signal
u_c	cooler signal
u_r	reaction signal

The plant dynamics is described by

$$\dot{x} = A(u)x + b(u) \tag{4.23}$$

where

$$A(u) = \begin{pmatrix} -a_{\mathrm{h}} u_{\mathrm{d}} & 0 \\ 0 & -(a_{\mathrm{T}_1}(1-u_{\mathrm{b}}) + a_{\mathrm{T}_2} u_{\mathrm{b}}) \end{pmatrix} \tag{4.24}$$

$$b(u) = \begin{pmatrix} b_{\mathrm{h}} u_{\mathrm{i}} \\ b_{\mathrm{heat}} u_{\mathrm{h}} + b_{\mathrm{cool}} u_{\mathrm{c}} + b_{\mathrm{reac}} u_{\mathrm{r}} \end{pmatrix} \tag{4.25}$$

To begin with, we will assume here that the coefficients in $A(u)$ and $b(u)$ are uncertain, and that they are given by

$$\begin{pmatrix} a_{\mathrm{h}} \\ a_{\mathrm{T}_1} \\ a_{\mathrm{T}_2} \\ b_{\mathrm{h}} \\ b_{\mathrm{heat}} \\ b_{\mathrm{cool}} \\ b_{\mathrm{reac}} \end{pmatrix} = \begin{pmatrix} 1.23 \cdot 10^{-3} \\ 0.15 \cdot 10^{-3} \\ 0.22 \cdot 10^{-3} \\ 9.838 \\ 29.43 \cdot 10^{-3} \\ -44.15 \cdot 10^{-3} \\ 44.15 \cdot 10^{-3} \end{pmatrix} - \begin{pmatrix} \delta_{\mathrm{ah}} \\ \delta_{\mathrm{T}_1} \\ \delta_{\mathrm{T}_2} \\ \delta_{\mathrm{bh}} \\ \delta_{\mathrm{heat}} \\ \delta_{\mathrm{cool}} \\ \delta_{\mathrm{reac}} \end{pmatrix} \tag{4.26}$$

where the numerical values are the nominal parameter values used in [30].

The controller is designed such that the control signals are switched on or off when the state reaches certain hyperplanes. The rules are listed below (the last rule is given by the physical properties of the system):

1. $u_{\mathrm{b}} = 0$ when $x_1 < 3$, $u_{\mathrm{b}} = 1$ otherwise.
2. u_{i} is set to 0 when $25x_1 + x_2 \geq 300$, and is set to 1 when $25x_1 + x_2 \leq 250$.
3. $u_{\mathrm{d}} = 0$ when $x_2 < 50$, $u_{\mathrm{d}} = 1$ otherwise.
4. $u_{\mathrm{h}} = 1$ when $x_2 < 50$, $u_{\mathrm{h}} = 0$ otherwise.
5. u_{c} is set to 0 when $x_2 \leq 110$, and is set to 1 when $x_2 \geq 130$.
6. $u_{\mathrm{r}} = 0$ when $x_2 < 50$, $u_{\mathrm{r}} = 1$ otherwise.

Note that the system contains hysteresis in u_{i} and u_{c}. This is handled by considering each polyhedron where the hysteresis occurs as two polyhedra with two different subsystems.

In Section 4.8.4, we will consider changing the thresholds for different control actions.

The switching hyperplanes and an example trajectory are shown in Figure 4.5. For further details concerning the system model, see [30].

4.8.2 What to Verify

There are certain requirements on the controller, which are verified in [30]. These are:

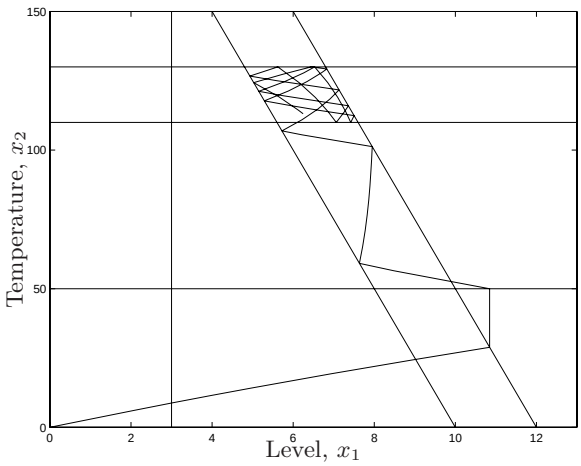


Figure 4.5: The switching hyperplanes and an example trajectory.

1. The temperature should stay between 0 and 150.
2. The tank must not be empty, and it must not overflow. The maximum level is 13.
3. There should be an operating region with moderate temperature and fluid level which should be invariant. In [30], this region is chosen to be
$$\{x \mid 250 \leq 25x_1 + x_2 \leq 300,\ 110 \leq x_2 \leq 130\} \tag{4.27}$$
4. The operating region should always be reached from the initial states in finite time. For simplicity, and to avoid introducing additional conservatism, we will not consider this requirement in this thesis.

The requirements can be translated to mathematical formulas:

1. (a) $\dot{x}_2 \geq 0$ when $0 \leq x_1 \leq 13$, $x_2 = 0$.
   (b) $\dot{x}_2 \leq 0$ when $0 \leq x_1 \leq 13$, $x_2 = 150$.
2. (a) $\dot{x}_1 \geq 0$ when $x_1 = 0$, $\ 0 \leq x_2 \leq 150$.
   (b) $\dot{x}_1 \leq 0$ when $x_1 = 13$, $0 \leq x_2 \leq 150$.
3. (a) $\dot{x}_2 \geq 0$ when $250 \leq 25x_1 + x_2 \leq 300$, $x_2 = 110$, and $u_c = 0$.
   (b) $\dot{x}_2 \leq 0$ when $250 \leq 25x_1 + x_2 \leq 300$, $x_2 = 130$, and $u_c = 1$.
   (c) $\begin{pmatrix}25 & 1\end{pmatrix} \dot{x} \geq 0$ when $25x_1 + x_2 = 250$, $110 \leq x_2 \leq 130$, and $u_i = 1$.
   (d) $\begin{pmatrix}25 & 1\end{pmatrix} \dot{x} \leq 0$ when $25x_1 + x_2 = 300$, $110 \leq x_2 \leq 130$, and $u_i = 0$.

We also have to know what affine subsystems $\dot{x}$ will satisfy in the different cases. We get those by considering the control rules.

4.8.3 Deriving Bounds for Parameter Uncertainties

Since we assume that the parameter values are uncertain, the question is how large the errors can get before the requirements are violated. Following the procedure in Section 4.4.2, we first need to find the direct representation to all polytope sides involved in the mathematical formulations of the requirements, which is basically the same as finding all the corners. Then, by using (4.12) we can get an exact answer to our question: The errors have to lie in a polyhedron described by

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 150 & 0 & 0 & 0 & -1 & -1 \\ 0 & 0 & 150 & 0 & 0 & -1 & -1 \\ 13 & 0 & 0 & 0 & 0 & 0 & 0 \\ -140 & 0 & -110 & 25 & 0 & 0 & 1 \\ -120 & 0 & -130 & 25 & 0 & 0 & 1 \\ 0 & 0 & -110 & 0 & 0 & 0 & 1 \\ -140 & 0 & -110 & 25 & 0 & 1 & 1 \\ -120 & 0 & -130 & 25 & 0 & 1 & 1 \\ 0 & 0 & 130 & 0 & 0 & -1 & -1 \\ 190 & 0 & 110 & 0 & 0 & 0 & -1 \\ 170 & 0 & 130 & 0 & 0 & 0 & -1 \\ 190 & 0 & 110 & 0 & 0 & -1 & -1 \\ 170 & 0 & 130 & 0 & 0 & -1 & -1 \end{pmatrix} \begin{pmatrix} \delta_{\mathrm{ah}} \\ \delta_{\mathrm{T_1}} \\ \delta_{\mathrm{T_2}} \\ \delta_{\mathrm{bh}} \\ \delta_{\mathrm{heat}} \\ \delta_{\mathrm{cool}} \\ \delta_{\mathrm{reac}} \end{pmatrix} \preccurlyeq \begin{pmatrix} 9.8380 \\ 0.0294 \\ 0.0225 \\ 0.0330 \\ 0.0160 \\ 245.7977 \\ 245.8179 \\ 0.0200 \\ 245.7536 \\ 245.7738 \\ 0.0286 \\ 0.2137 \\ 0.1935 \\ 0.2579 \\ 0.2377 \end{pmatrix} \tag{4.28}$$

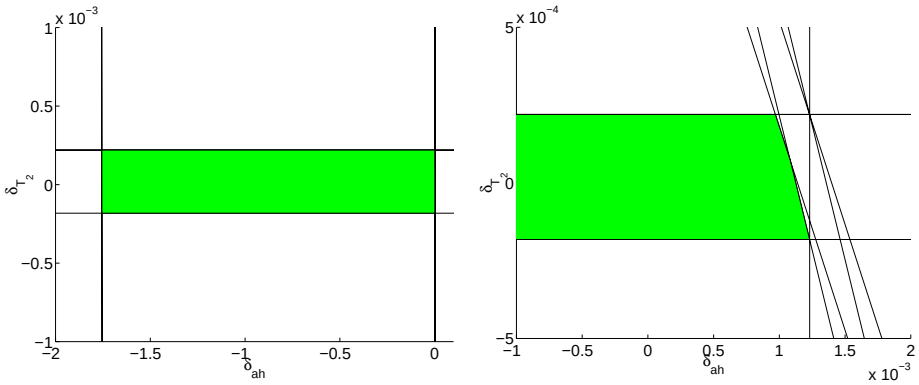
The expression above has been simplified, in that redundant inequalities have been removed. We notice immediately that the polyhedron contains the origin, which means that the nominal system satisfies the requirements. To get a more intuitive feeling for the bounds, one can also consider a subset of the errors and set the other errors to zero. For example, suppose that only a_{h} and $a_{\mathrm{T_2}}$ are uncertain. In order not to violate the requirements, their deviations from the nominal values have to be contained in the polyhedron shown in Figure 4.6. As we can see, the basic effect of the requirements (4.28) in this case is that $\delta_{\mathrm{T_2}}$ has to lie in the interval $[-0.18 \cdot 10^{-3}, 0.22 \cdot 10^{-3}]$, while δ_{h} approximately can vary between -1.76 and $1 \cdot 10^{-3}$.

4.8.4 Adjusting Control Rules

Let us now turn to the question of finding bounds, inside which we can move the thresholds of the controller rules without affecting the properties to verify. Here we assume that $A(u)$ is known exactly, but that the parameters of $b(u)$ are still unknown. The hyperplanes we are going to move are

$$\begin{aligned} x_1 &= 3 + \gamma_1 \\ 25x_1 + x_2 &= 250 + \gamma_2 \\ 25x_1 + x_2 &= 300 + \gamma_3 \\ x_2 &= 110 + \gamma_4 \\ x_2 &= 130 + \gamma_5 \end{aligned}$$

These are the thresholds for the blender, the inflow valve, and the cooler signals. The other hyperplanes are assumed to be fixed (by physical properties of the plant). We would like to know how much we can change the values of $\gamma_1, \ldots, \gamma_5$.



(a) The entire allowed region.

(b) Close-up near the origin.

Figure 4.6: The allowed region for δ_{h} and δ_{T_2}, assuming that the other errors are equal to zero. Figure 4.6(b) shows a close-up of the region near the origin.

First we should make sure that we preserve the topology. Applying (4.15) to all corners yields the condition (after removing several redundant inequalities)

$$\begin{pmatrix}
-1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 1 \\
25 & -1 & 0 & 1 & 0 \\
25 & -1 & 0 & 0 & 1 \\
25 & -1 & 0 & 0 & 0 \\
25 & 0 & -1 & 1 & 0 \\
25 & 0 & -1 & 0 & 1 \\
25 & 0 & -1 & 0 & 0 \\
0 & 1 & -1 & 0 & 0 \\
0 & -1 & 0 & 1 & 0 \\
0 & 1 & 0 & -1 & 0 \\
0 & -1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & -1 \\
0 & 0 & -1 & 1 & 0 \\
0 & 0 & 1 & -1 & 0 \\
0 & 0 & -1 & 0 & 1 \\
0 & 0 & 1 & 0 & -1 \\
0 & 0 & 0 & 1 & -1
\end{pmatrix} \gamma \prec \begin{pmatrix}
3 \\ 10 \\ 100 \\ 75 \\ 150 \\ 25 \\ 60 \\ 40 \\ 80 \\ 20 \\ 65 \\ 45 \\ 25 \\ 115 \\ 95 \\ 75 \\ 50 \\ 140 \\ 185 \\ 120 \\ 205 \\ 190 \\ 135 \\ 170 \\ 155 \\ 20
\end{pmatrix} \tag{4.29}$$

The first ten inequalities make sure that the moving hyperplanes do not pass any of the corners of the fixed hyperplanes (compare with Figure 4.5). The remaining inequalities state the relations between the moving hyperplanes (for example, the last inequality tells us that the cooler should be turned off at a lower temperature than when it is turned on).

In addition to the topological requirement, the system also has to satisfy the properties to verify. This is achieved if δ_{bh}, δ_{heat}, δ_{cool}, δ_{reac}, and γ satisfy the following set of inequalities, obtained from (4.16):

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 25 & 0 & 0 & 1 & 0 & 0.0012 & 0 & -0.001 & 0 \\ 25 & 0 & 0 & 1 & 0 & 0.0012 & 0 & 0 & -0.001 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0.0002 & 0 \\ 25 & 0 & 1 & 1 & 0 & 0.0012 & 0 & -0.001 & 0 \\ 25 & 0 & 1 & 1 & 0 & 0.0012 & 0 & 0 & -0.001 \\ 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & -0.0002 \\ 0 & 0 & 0 & -1 & 0 & 0 & -0.0012 & 0.001 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & -0.0012 & 0 & 0.001 \\ 0 & 0 & -1 & -1 & 0 & 0 & -0.0012 & 0.001 & 0 \\ 0 & 0 & -1 & -1 & 0 & 0 & -0.0012 & 0 & 0.001 \end{pmatrix} \begin{pmatrix} \delta_{\text{bh}} \\ \delta_{\text{heat}} \\ \delta_{\text{cool}} \\ \delta_{\text{reac}} \\ \gamma_1 \\ \vdots \\ \gamma_5 \end{pmatrix} \preccurlyeq \begin{pmatrix} 9.8380 \\ 0.0294 \\ 0.0225 \\ 245.7977 \\ 245.8179 \\ 0.0200 \\ 245.7536 \\ 245.7738 \\ 0.0286 \\ 0.2138 \\ 0.1935 \\ 0.2579 \\ 0.2377 \end{pmatrix} \tag{4.30}$$

Together with (4.29), (4.30) now describes the region, in which the values of δ_{bh}, δ_{heat}, δ_{cool}, δ_{reac}, and γ are allowed to vary. We can also note that setting $\delta_{\text{ah}} = \delta_{\text{T}_1} = \delta_{\text{T}_2} = 0$ in (4.28) and $\gamma = 0$ in (4.30) both give the same restrictions for the remaining uncertainties δ_{bh}, δ_{heat}, δ_{cool}, and δ_{reac}.

From (4.30), one can go further and, e.g., find the values of γ that allow as large uncertainties as possible in the b parameters. This gives in a sense the maximally robust controller satisfying the verification, and is calculated similarly to the computation of the *Chebychev centre* of a polyhedron (the centre of the largest sphere inscribed in a polyhedron; see [11]). Here we would like to find the largest disc, parallel to the γ coordinate axes and with its centre somewhere along the subspace $\delta = 0$, inscribed in the polyhedron defined by (4.29) and (4.30). Let us represent the disc as

$$\mathcal{D} = \{ \begin{pmatrix} 0 \\ \gamma \end{pmatrix} + \begin{pmatrix} \delta \\ 0 \end{pmatrix} \mid \|\delta\| \leq R \} \tag{4.31}$$

where the vector $\left(\begin{smallmatrix} 0 \\ \gamma \end{smallmatrix}\right)$ is the centre of the disc and R is the radius. We would like to maximise R subject to the constraint that $\mathcal{D}$ satisfies (4.29) and (4.30). Let us stack (4.29) and (4.30) on top of each other, and denote the resulting matrices according to

$$F \begin{pmatrix} \delta \\ \gamma \end{pmatrix} \preccurlyeq g \tag{4.32}$$

For each row i in (4.32), the constraint that $\mathcal{D}$ should satisfy $F_i \left(\begin{smallmatrix} \delta \\ \gamma \end{smallmatrix}\right) \leq g_i$ can be expressed

$$F_i \begin{pmatrix} 0 \\ \gamma \end{pmatrix} + \sup_{\|\delta\| \leq R} F_i \begin{pmatrix} \delta \\ 0 \end{pmatrix} \leq g_i$$

But since

$$\sup_{\|\delta\| \leq R} F_i \begin{pmatrix} \delta \\ 0 \end{pmatrix} = \sup_{\|\delta\| \leq R} F_i \begin{pmatrix} I \\ 0 \end{pmatrix} \delta = R \left\| F_i \begin{pmatrix} I \\ 0 \end{pmatrix} \right\|$$

(where I is an identity matrix with the same number of rows as δ), we get

$$F_i \begin{pmatrix} 0 \\ \gamma \end{pmatrix} + R \left\| F_i \begin{pmatrix} I \\ 0 \end{pmatrix} \right\| \leq g_i$$

This is a linear inequality in γ and R, and the desired value of γ can be computed from the LP

$$\begin{aligned} \max_{\gamma, R} \quad & R \\ \text{subj. to} \quad & F_i \begin{pmatrix} 0 \\ \gamma \end{pmatrix} + R \left\| F_i \begin{pmatrix} I \\ 0 \end{pmatrix} \right\| \leq g_i, \quad i = 1, \ldots, M_F \end{aligned} \tag{4.33}$$

where M_F is the number of rows of F.

4.9 Conclusions

We have suggested an approach to investigate how sensitive approximating automata for piecewise affine systems might be to changes in the underlying subsystems, to noise, and to translations of the switching surfaces. Section 4.4 provided the sets of system matrices that satisfy certain demands on the behaviour of the system. As pointed out, these can either be seen as giving a measure of how robust the approximating automata are to uncertainties in the system, or as giving limits for how much the system can be changed, e.g., in a control design process, without altering the overall behaviour described by the approximating automata. The methods can also be extended to switched systems, which was mentioned in Section 4.7.2.

It would be natural to combine these demands with other objectives. One example of this was given in Section 4.8.4. To give another example, when using the state feedback parametrisation described in Section 4.5, one would probably want to find $L(v)$ that are optimal in a certain respect. Since the solution sets of the two first problems in Section 4.4.2 and Section 4.4.4 are convex, we can form all sorts of convex optimisation problems, which can be solved very efficiently once we know the direct representations of the polyhedra (see for example [11]).

5

Identification of Piecewise Affine Systems

To be able to control and analyse a system, one needs a model of the system that shows the relation between input and output signals, and the sensitivity to disturbances. Depending on what one would like to achieve, the model might be more or less accurate. In many cases, a linear model around the working point will be sufficient. However, when the nonlinearities are not negligible within the operating region, a piecewise affine model might be an attractive alternative. This can be the case both when the real system actually is piecewise affine, and for a general nonlinear system. Obtaining such a model from experimental data can be done in several different ways.

In this chapter, some existing approaches are investigated and compared. Some of the approaches are quite general nonlinear identification methods, for which special cases lead to piecewise affine models, while other approaches are more specialised on piecewise affine systems.

The system models considered in this and the next chapter will mostly be in regression form as described in Section 3.1.4, i.e.,

$$y(t) = \varphi^T(t)\theta(v) + e(t) \tag{5.1}$$

where $\theta(v)$ is a function of the key vector v, which in turn is a piecewise constant function $v = v(\varphi)$ of the regression vector φ, taking only a finite number of values.

The regression vector could consist of old inputs and outputs, e.g.,

$$\varphi(t) = \begin{pmatrix} 1 & -y(t-1) & \dots & -y(t-n_a) & u(t-1) & \dots & u(t-n_b) \end{pmatrix}^T$$

The modes are consequently determined by v. If we let n_v be the number of different modes of the system, and let $\theta = \begin{pmatrix} \theta(v_1) & \dots & \theta(v_{n_v}) \end{pmatrix}^T$, the criterion function to be minimised (cf. Section 2.2.1) for this kind of models can be written as

$$V(\theta, Z^N) = \frac{1}{N} \sum_{t=1}^{N} \sum_{j=1}^{n_v} \ell \left(y(t) - \varphi^T(t)\theta(v_j) \right) \chi_{v_j}(\varphi(t)) \tag{5.2}$$

where

$$\chi_{v_j}(\varphi) = \begin{cases} 1 & v(\varphi) = v_j \\ 0 & \text{otherwise} \end{cases}$$

The two most common choices for the function ℓ in (5.2), and the ones that will be considered here, are the squared Euclidean norm (2-norm)

$$\ell_2(\varepsilon) = \varepsilon^2$$

and the 1-norm

$$\ell_1(\varepsilon) = |\varepsilon|$$

It can be noted, that given a true system in the form (5.1), where $e(t)$ are independent with the probability distribution function $f_e(x)$, the $\theta(v_j)$ that minimises (5.2) will be the *maximum likelihood estimate* if

$$\ell(\varepsilon) = -\log f_e(\varepsilon)$$

(see [63]). From this it follows, e.g., that ℓ_2 corresponds to the maximum likelihood estimator for a system where $e(t)$ has a Gaussian distribution with known variance.

Obviously, for a given φ, exactly one of the functions $\chi_{v_j}(\varphi)$ equals 1. This means that all the terms in the inner sum in (5.2) except for one are zero. Hence, if the partitioning χ_{v_j} of the state-space is known, i.e., if we know what experimental data belongs to which affine subsystem, then using 2-norm will reduce the problem of identifying piecewise affine systems of this kind to a least-squares problem like in Example 2.5, which can easily be solved using standard techniques. However, when the partitioning is unknown, and consequently both χ_{v_j} and θ should be estimated, the problem becomes much more difficult. There are two fundamental approaches to take. The first possibility is to define an a priori grid (like in Section 3.1.6), and estimate one affine subsystem per cell in the grid (see, e.g., [52]). This approach leads to a simple estimation process, but will in a sense suffer from the curse of dimensionality, since the number of cells (and hence the number of parameters) will grow exponentially with the number of dimensions, which also leads to a need

for very large datasets. The other possible approach is to estimate both the partitioning and the subsystems simultaneously or iteratively (like in, e.g., [14, 34, 51]). This means that we will in general need a smaller number of subsystems, but that the estimation process will be more complex, with potentially many local minima, that will complicate the use of local search minimisation algorithms.

Using piecewise affine systems for modelling and identification can be seen as a special case of multiple model approaches. In [72], an excellent introduction to nonlinear modelling using multiple models is given together with a survey of the field.

Some of the main issues in piecewise affine system identification are

- the ability to find a good model of the real system, preferably requiring as few parameters as possible,
- the ability to avoid getting stuck in local minima during the search for a good model, and
- the computational complexity.

In the next chapter, an identification method based on mixed-integer linear or quadratic programming will be presented in detail. As we will see, the mixed-integer programming approach guarantees that the global optimum is reached, at the cost of increased complexity. However, using a "truncated mixed-integer optimisation algorithm" might be a feasible alternative, as will be discussed in Section 6.5.

5.1 Existing Approaches

Different approaches to piecewise affine system identification can be found in many fields in the literature, as mentioned in Section 3.4. However, many of them are related. One of the crucial points that separate different groups of approaches from each other is how the partition into different regions is done. As we have seen from (5.2), as soon as we know the different regions, it is easy to identify the different affine subsystems one by one, using standard linear system identification techniques. From this viewpoint, the different approaches to identification of piecewise affine systems can be categorised as follows:

- All parameters, both the parameters determining the partition and the parameters of the submodels, are identified simultaneously. Such approaches can be found, e.g., in [4, 36, 51, 79].
- All parameters are identified simultaneously for a model class with a very simple partition, and new submodels/regions are added when needed. Examples of this kind of approach are found in [14, 33, 41, 44, 51, 79].
- The regions and submodels are identified iteratively or in several steps, each step considering either the regions or the models. The methods in [34, 42, 75, 87, 88] belong to this category. This is also the main approach in [72].

- The regions are predetermined or determined using only information about the distribution of the regression vectors, and the local submodels are identified after that. The approaches described in [18, 52, 55, 65, 91, 94, 95] fall into this category.

The following sections describe the different categories more extensively, and some advantages and drawbacks are pointed out. It should also be mentioned that there are of course alternative ways of categorising different approaches. For example, one can distinguish between online and offline algorithms.

5.1.1 Identifying All Parameters Simultaneously

Perhaps the most straightforward way of attacking the piecewise affine system identification problem is to formulate the criterion function (5.2), parametrised according to a certain model structure, and then minimise it directly, using a numerical method, e.g., a Gauss-Newton search (see Section 2.2.2). In that way both the partitioning and the submodels are estimated simultaneously. The greatest advantage with such an approach might be its simplicity, and a reasonable computational complexity (depending on what numerical method is used).

The greatest drawback with this approach is that the optimisation algorithm might get trapped in a local minimum. This drawback is shared with many other of the categories.

The result also depends on how the model is parametrised. In general, it is desirable to have as few free parameters as possible, both for numerical and complexity reasons.

One example of this approach is given by **Pucar and Sjöberg** [79] where a hinging hyperplane model, as defined in Section 3.1.5, is used. Another example is found in **Julián et al.** [51], where Chua's canonical representation (3.6) is used to approximate nonlinear functions. In both cases, the number of hinge functions is assumed to be known. Pucar and Sjöberg present a damped Newton method for estimating all parameters of the model simultaneously, while Julián et al. use a Gauss-Newton algorithm. Local convergence of the parameters is proven.

In an earlier article by **Batruni** [4], a multilayer neural network with piecewise affine functions in Chua's canonical representation (as described in Section 3.1.5) in each layer is proposed. The parameters would be estimated using backpropagation, which basically is a kind of gradient search.

An approach using a 1-norm criterion function can be found in **Gad et al.** [36]. Since the criterion function itself becomes a piecewise affine function of the parameters, they can use a simplex-like optimisation algorithm to efficiently find a local minimum. However, as for the other approaches in this category, convergence to a global minimum cannot be guaranteed.

There are several general numerical optimisation algorithms that try to reduce the risk of getting stuck in a local minimum. These include simulated annealing, genetic algorithms [26], etc. The basic idea of both simulated annealing and genetic algorithms is to modify a standard numerical optimisation algorithm, such

as the ones described in Section 2.2.2, by repeatedly using random updates of the parameter estimates. These kinds of algorithms can be used for the piecewise affine system identification problem, at the cost of increased computational complexity.

5.1.2 Adding One Partition at a Time

Instead of solving the entire optimisation problem at once, it can be divided into several different steps, to get a number of easier optimisation problems that can be solved one at a time. One way of achieving this is to start with a simple model, for instance a hinging hyperplane model with only one hinge function, and fit it to the data. Let us call this hinge function $k_1(\varphi, \theta_1)$. The criterion to be minimised is

$$V^1(\theta_1, Z^N) = \frac{1}{N} \sum_{t=1}^{N} \ell \left(y(t) - k_1(\varphi(t), \theta_1) \right)$$

If the resulting model $k_1(\varphi, \hat{\theta}_1)$ is not satisfactory, one can add a new hinge function by fitting it to the residual $\varepsilon_1(t) = y(t) - k_1(\varphi(t), \hat{\theta}_1)$; that is, we minimise

$$V^2(\theta_2, Z^N) = \frac{1}{N} \sum_{t=1}^{N} \ell \left(\varepsilon_1(t) - k_2(\varphi(t), \theta_2) \right)$$

The sum of the two models, $k_1(\varphi(t), \hat{\theta}_1) + k_2(\varphi(t), \hat{\theta}_2)$, will then be a better approximation of the system. This procedure is repeated until the value of the criterion function (5.2) does not decrease significantly anymore, or until the resulting model is satisfactory. One also has to take into consideration the risk of *overfitting*, which means that the model might start to adjust to the particular noise realisation, if given too many degrees of freedom (e.g., too many hinge functions).

In addition to this algorithm, one can include a *refitting* procedure: When having added the hth hinge function, the previously added hinge functions are refitted one by one, so that the ith function $k_i(\varphi, \theta_i)$, $i \leq h$, is fitted to the residual $y(t) - \sum_{j \neq i} k_j(\varphi(t), \hat{\theta}_j)$. This procedure can also be iterated until no further improvement can be observed. Note that the criterion function will not increase by the refitting, so there is no risk of getting stuck in limit cycles.

The advantage of this approach of adding one, e.g., hinge function at a time, compared to the previous one of identifying all parameters simultaneously, is that each optimisation problem is easier to solve. On the other hand, one has to solve several optimisation problems instead of just one. When the number of regions/hinge functions of the real system is unknown, this approach may also have an advantage compared to the previous one. Just as for the first category, there is a risk of getting stuck in a local minimum.

Breiman [14] introduced the hinging hyperplane models and the algorithms described above for use in function approximation. The algorithm for estimating one single hinge function is called the *hinge-finding algorithm*. It starts by assuming that the dataset is partitioned into two subsets, and then estimates local affine

models to each of the two sets. These affine models constitute the two submodels of a hinge function. From the fact that the hinge function should be continuous, the hinge of the hinge function can be determined. However, this new hinge may very well partition the data into two subsets which are different from the original subsets. Therefore, the procedure can be repeated using the new partition, and in this way we iterate until the same partitions are obtained in two consecutive iterations. Then a local minimum of $V^1(\theta_1, Z^N)$ is found.

The problem with this algorithm is that there is no guarantee for convergence. One can get stuck into limit cycles, or one can get least-squares problems that do not have a unique solution, which corresponds to a partition where one of the regions does not contain enough data points.

The identification algorithms in [14] were analysed further in **Pucar and Sjöberg** [79], and it was shown that the hinge-finding algorithm is a special case of a (non-damped) Newton algorithm, as presented in Section 2.2.2. It was also modified to guarantee convergence, yielding a damped Newton algorithm. Pucar and Sjöberg also give a greedy algorithm, where – just as in the previously described algorithms – one hinge function at a time is added. The model is written as a *weighted* sum of the hinge functions, and when a new function is added, its parameters are estimated together with the weights of the other hinge functions, while all other parameters are kept fixed. Pucar and Sjöberg compare the performance of this greedy function with the performance of the algorithm mentioned in the previous section for a noise-free example, and it turns out that the simultaneous estimation of the parameters performs better when the true model was within the model class (for this example).

Julián et al. [51] use a similar algorithm, but for Chua's canonical representation. They also mention the possibility of adding several hinge functions at a time.

Hush and Horne [44] use so-called *hinging sigmoid (HS)* functions for function approximation. A HS function is defined by

$$\sigma(\varphi) = \begin{cases} \theta_+ & \varphi^T\theta \geq \theta_+ \\ \varphi^T\theta & \theta_- \leq \varphi^T\theta \leq \theta_+ \\ \theta_- & \varphi^T\theta \leq \theta_- \end{cases}$$

Like in [14], one HS function is added at a time, and is fitted to the residual from the previous estimations. A refitting procedure similar to the one in [14] is also suggested. For the fitting of a single HS function three algorithms are proposed, of which one is due to Breiman and Friedman [15]. All the algorithms make use of the fact that once the partition is known, finding the parameters θ, θ_+ and θ_- is a least-squares problem with linear constraints (corresponding to the requirement that the partition should not change). A similar approach is also presented in Section 6.6.

The Breiman/Friedman algorithm is analogous to the hinge-finding algorithm in [14] mentioned above, only for a different model class. It is a Newton algorithm, where in each iteration the updated parameter estimate is directly placed at the

minimum of the local quadratic function. As for the hinge-finding algorithm, there is no guarantee for convergence for this algorithm. To avoid convergence problems, one should be able to modify the step size of the parameter update in a way similar to what is being done with the hinge-finding algorithm in [79].

The second algorithm in [44] also starts by assuming a partition. Given the partition, it solves the obtained QP problem including the linear constraints, and then detects if there are any data points at the borders of the partition. In that case, these data points are assigned to the other region (as long as this does not destroy the positive definiteness of the quadratic objective function), and the new QP problem is solved. This is repeated until the objective function does not decrease anymore.

This algorithm is also related to the Newton algorithms. Here the minimum is found within the region where the criterion function is quadratic (which corresponds to preserving the current partition of data points), and then, if this minimum is situated at a border of the region, a new step is taken into one of the adjacent regions. This means that each iteration explores exactly one new partition, and that the parameter estimates only move between adjacent regions of the piecewise quadratic criterion function.

The third algorithm described in [44] is an extension of the second, and can basically be seen as a way to give good initial values to the second algorithm. It first fits an affine function $\varphi^T\theta$ to the data. This affine function is used as the initial middle (nonconstant) part of the HS function. The data points are ordered along the direction of the gradient of the affine function, and the hinges are initially placed at the two extreme points on each side. From this initial partition, algorithm 2 is used to find a local minimum. Then one data point at a time is taken from the middle region and assigned to the upper region (where $\varphi^T\theta \geq \theta_+$) to see if improvements can be made. After this the other hinge location is optimised, and finally algorithm 2 is performed again.

In the article [33] by **Ernst**, hinging hyperplanes are used together with a tree structure, where a hinge function is added to the subregion with the worst fit. This approach is related to the model trees in [91], and is explained in more detail in Section 5.1.4.

A similar approach is taken by **Johansen and Foss** [46]. The models that are used resemble the hinging hyperplane trees, in the sense that the different subregions are partitioned hierarchically: The state-space is split by a hyperplane, and each of the two new regions are independently partitioned etc. To determine the position of the first hyperplane, their algorithm starts by performing an exhaustive search over models with $2 + k$ regions. The best of these models determines where the first hyperplane should be placed. Having fixed that, the algorithm determines the second partitioning hyperplane by an exhaustive search over models with $3 + k$ regions etc. Since the authors use interpolation between the different submodels, the resulting model is not a piecewise affine system, but the method could be used also for piecewise affine systems.

Heredia and Arce [41] use continuous piecewise affine systems with rectangular subregions with the borders parallel to the coordinate axes, and a somewhat

different representation called *Continuous Threshold Decomposition*. Each border is defined by a threshold, and the authors use the same set of thresholds for all variables. They propose a method where one threshold at a time is added using a numerical method. An Akaike criterion [63] is used to decide how many thresholds should be added.

5.1.3 Finding Regions and Models in Several Steps

The last two categories are related in the sense that identifying the different regions and the different subsystems are done separately, in different steps. Many of the methods found in the literature are constructed in a similar way:

- Local affine models are used to find the partition of the data.
- After the partition is done, the parameters of the affine subsystems can easily be identified.

The local affine models in the first step can either be used to cluster the data points into different clusters, in which all data points have similar model parameters, or they can be used together with a change detection algorithm, in order to find the mode switchings.

One example of such an approach is found in [34] by **Ferrari-Trecate et al.** They consider PWARX systems, as defined in Section 3.1.4. Their identification algorithm consists of five steps. The first step is to estimate local affine models around each data point. This is done using the c data points that are closest to the point for which the local model is estimated, where c is a constant specified by the user. Then a measure of the "confidence level" for each local estimation is computed, based on how scattered the points used for the estimation are, and on the empirical covariance matrix for the estimation. After this, a K-means algorithm is used to cluster the data points (see [10], or [29] for an overview of different clustering techniques), based both on their position and on the parameters of the local model. The two last steps are to estimate an affine submodel for each cluster, and to try to find hyperplanes that separate the clusters from each other.

Hoffmann and Engell [42, 43] consider a hybrid system model with several modes, where the dynamics of each mode is linear. When the state reaches one of several switching regions, the system switches to another mode according to a table. State jumps (see Section 3.1.8) are also allowed, and are modelled by affine functions, one for each switching region.

The identification process is divided into several steps: First the mode switches are detected with a standard change detection algorithm, where a model

$$x(k+1) = Ax(k) + Bu(k) + d(k)H(k-K)$$

is fitted to a window of the time series. When $\|d(k)\|$ exceeds a given threshold, this indicates that a mode switching has occurred.

In the second step, the switching points are clustered and classified as belonging to different switching regions, and the regions are approximated by hyperspheres.

Then a possible mode sequence is found. From this a discrete model for the different mode switchings is built.

In the last steps, the affine functions for the state jumps and the continuous dynamics of each mode are estimated using standard linear identification techniques.

One problem with the approach by Hoffmann and Engell is that the system model is only valid along the trajectories of the experimental data. This is due to the very flexible hybrid model, which does not allow any conclusions about unexplored parts of the state-space to be drawn, since there might for instance be an undiscovered switching region anywhere in these parts.

In **Skeppstedt et al.** [87, 88], an online approach is considered. The models taken into consideration are of PWARX type. A multiple-model recursive parameter estimation algorithm called XAFFM is used to estimate the current parameter value. XAFFM does not only give estimates of the parameters, but also of the covariance matrix and posterior probability of the parameter estimate.

By applying Bayes' rule, using information about the operating point and the output of the sample, it is determined to which of the affine submodels that the current sample most likely belongs. If the posterior probability of the current estimated parameter value is high enough, the parameter value of the submodel is updated. However, if the estimate of the current parameter value and the previously estimated submodel parameter value differ too much, no updating is done. Instead, if this happens repeatedly, a new submodel is added to the total model.

After the update of the dynamics, the separating hyperplanes are updated by a standard procedure for linear discrimination.

The *Switching Dynamical Systems* considered in **Petridis and Kehagias** [75] do not really belong to the model classes considered so far in this chapter. They consist of a number of subsystems, between which switchings occur with a certain probability for each time step. The dependence between the current state and mode, which is inherent in the models considered previously, is not considered in [75].

For these systems, an iterative identification algorithm related to the algorithm by Skeppstedt et al. is presented. Initially, the number of submodels K is set to one, $t = 1$ and a threshold ϵ is chosen. Then the following steps are executed iteratively:

1. For $k = 1, \ldots, K$, compute predictions $\hat{y}_k(t)$ of $y(t)$ using the current models.

2. For $k = 1, \ldots, K$, if $|y(t) - \hat{y}_k(t)| < \epsilon$, let $y(t)$ be assigned to model k and go directly to step 3. If no model satisfies the requirement, let $K = K + 1$, add one model and assign $y(t)$ to this model.

3. Estimate all submodels using the data assigned to them. Let $t = t + 1$ and go to 1.

Some convergence results are given, in the sense that the ratio between the falsely and the correctly classified data points will asymptotically approach 0 with probability 1.

Medeiros et al. [66, 67] use an algorithm called GRASP (Greedy Randomised Adaptive Search Procedure) to iteratively and heuristically find good partitions of the state-space. When a partition is given, the remaining parameters are estimated using least squares, and the mean squared error is taken as a measure of how good the partition was.

Related to this category are the *Local Learning* methods of [70, 71] by **Murray-Smith et al.,** also considered in [72]. They use local linear models, which are interpolated to get the total system model. The interpolation implies that the resulting systems are not piecewise affine systems. The identification process follows an iterative algorithm, where the model structure is first determined using a priori knowledge. Then the model parameters are identified locally for each submodel. After this, the algorithm determines where the model structure needs improvement, and the structure is adjusted. The last two steps are then iterated until the result is satisfactory.

Another related approach is the *Model on Demand* approach by **Stenman** [90], and similar methods [78, 85]. The philosophy behind this approach is to keep a database of old data, and to estimate a local model when needed for every point of interest. The local models are estimated using data samples from the vicinity of the state-space, and, e.g., a weighted least squares criterion. Now, for a given set of data in the database, and for a piecewise constant weight function, the resulting global model will be a piecewise affine system (even if it consists of a very large number of submodels). The shapes of the different regions and the submodels are determined by the choice of weight function, which becomes a trade-off between incorporating as much data as possible to decrease the variance of the estimate, and building a model which is as local as possible to avoid a biased model.

5.1.4 Using Predetermined Regions

One possible approach is to partition the state-space in a regular grid, with one affine subsystem for each region in the grid. Since both the partitioning of the state-space and the identification of the subsystems become very easy (given enough experimental data in each region), this approach might work well for low-dimensional systems. However, the number of regions grows exponentially with the dimensions, so both the computational complexity and the need for experimental data are exponential in the number of dimensions, making the approach infeasible for higher-dimensional systems.

One example of this kind of approach is given in [52] by **Julián**, where the author considers nonlinear function approximation using the HL CPWL class (see Section 3.1.6).

Instead of deciding on the partition completely independently of the experimental data, another option might be to take the distribution of the regression vectors into account. In this way, the state-space can be partitioned in such a way that each region gets a proper amount of experimental data to allow for the estimation of an affine subsystem, and the exponential complexity can be avoided. The approach in [18] and the model trees proposed in [91] fall into this category.

In [18] by **Choi and Choi**, a method is proposed, where the state-space is first partitioned into hypertriangles, with experimental data points in each corner. For each hypertriangle an affine subsystem is fitted. Since only $n+1$ points are used to fit each n-dimensional subsystem, this method is quite sensitive to noise. Therefore, the authors also suggest to use an unsupervised learning algorithm with fewer cells to get the partition. Then the problem of estimating the affine subsystems becomes a standard identification problem.

In the neural network literature, one can find several other examples of methods that start by clustering the data, and then build local linear models for each cluster; see, e.g., [55, 65, 94, 95].

The model trees proposed by **Strömberg et al.** [91] stem from the Classification and Regression Trees (CART) [16]. Here, a binary tree is built, where each node represents a region of the state-space, and the regions of the children nodes form a partition of the region of the parent node. Then one linear subsystem is associated with each leaf of the tree.

The obvious disadvantage with just considering the distribution of the regression vectors, and not the corresponding y values, is that a set of data which really should belong to the same subsystem might be split arbitrarily. To somewhat make up for this drawback, one option is to make the partition rather fine (with many regions), and then afterwards merge adjacent regions with similar subsystems (cf. the pruning strategy in [91]). A minor problem with this approach is that the merged regions may be of varying shapes and sizes, and it might be difficult to represent them efficiently. For models represented as a tree structure, this is no problem as long as the adjacent regions are also adjacent leaves in the tree. Then, all that needs to be done is to erase the leaves and let their parent node become a new leaf representing the merged region. However, if the regions belong to completely different branches of the tree, it may even be hard to determine whether they are adjacent or not.

One idea would be to use Breiman's Hinge Finding Algorithm (or the modified version in [79]) to find good partitions for the model trees. This would reduce the risk that two adjacent regions with similar dynamics would end up in completely different branches of the tree. This is what is done by Ernst in [33].

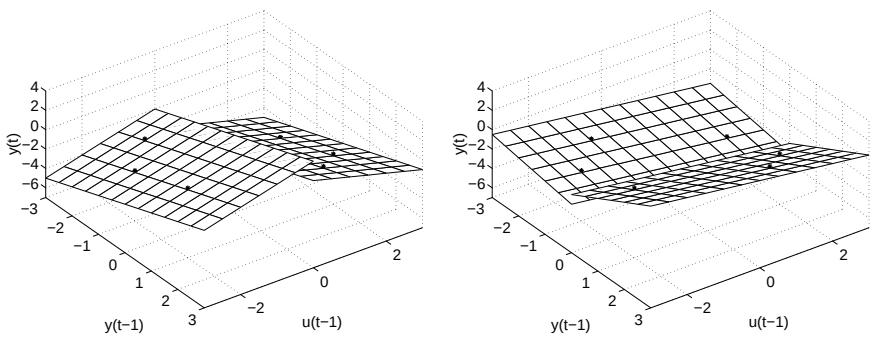
5.2 Discussion

The recurring problems with piecewise affine system identification concern how to find good solutions (a global minimum or a good local minimum) to the identification problem given a model structure, and how to keep the computational complexity and model complexity (number of model parameters) low. The computational complexity for a given algorithm is a function both of the number of regions/subsystems, the number of experimental data, and of the number of dimensions of the regression vector. Hence, there are many trade-off situations when comparing different methods and tuning parameters for given algorithms.

One such issue was mentioned already in Section 3.4: The choice between choos-

ing an a priori grid or estimating the partitioning together with the subsystems. The first option leads to a simple estimation process, but to get a good result, the numbers of regions and data needed will increase exponentially with the number of dimensions. Hence, this approach may be good for low-dimensional systems, but for systems of higher dimensions another approach might be preferred.

Let us take a closer look at the requirements on the data. To be able to identify a system satisfactory, several or all modes (depending on the system structure) have to be excited by the data, and furthermore, each mode has to be excited for a sufficiently long time. A set of data that, given the model structure and a criterion function, uniquely determines the system will here be called *persistently exciting*. For example, for the case of predetermined regions and mutually independent affine subsystems (as in a PWARX system), we need at least as many linearly independent data samples in each mode as the dimension of the regression vector. When the regions are to be estimated as well, we might even need more data, as the following example shows.



(a) True system. (b) False system.

Figure 5.1: The system that is identified in Example 5.1.

Example 5.1 (Insufficient data) *Consider the system*

$$y(t) = \begin{cases} \begin{pmatrix} 1 & -1 & 1 \end{pmatrix} \varphi(t) & \varphi_3(t) < 0 \\ \begin{pmatrix} -1 & -1 & -1 \end{pmatrix} \varphi(t) & \varphi_3(t) \geq 0 \end{cases}$$

where $\varphi(t) = \begin{pmatrix} 1 & -y(t-1) & u(t-1) \end{pmatrix}^T$ *(and hence* $\varphi_3(t) = u(t-1)$*). Suppose*

that we are given the data samples $(\varphi(t), y(t))$*:*

$$\Big\{(\begin{pmatrix}1\\-1\\2\end{pmatrix}, -2),\ (\begin{pmatrix}1\\1\\2\end{pmatrix}, -4),\ (\begin{pmatrix}1\\-2\\1\end{pmatrix}, 0),$$
$$(\begin{pmatrix}1\\-1\\-2\end{pmatrix}, 0),\ (\begin{pmatrix}1\\2\\-1\end{pmatrix}, -2),\ (\begin{pmatrix}1\\1\\-2\end{pmatrix}, -2)\Big\}$$

Given the true partition of the state-space, these points determine the system uniquely, since there are three linearly independent samples in each region (see Figure 5.1(a)). However, we could alternatively choose another, incorrect partition and get a completely different system. An example of this is shown in Figure 5.1(b), where the system

$$y(t) = \begin{cases}\begin{pmatrix}-2.5 & -1.5 & -0.5\end{pmatrix}\varphi(t) & \varphi_2(t) < 0\\ \begin{pmatrix}-3.5 & 0.5 & -0.5\end{pmatrix}\varphi(t) & \varphi_2(t) \geq 0\end{cases}$$

is shown to match the experimental data perfectly.

The need for enough data from each subsystem limits the number of subsystems that would be feasible to identify. Therefore, for larger examples it becomes advantageous to allow the regions to be formed according to the data, to allow for fewer regions and parameters.

Another trade-off issue is the one concerning the choice between the three first categories of the previous sections. The first category – simultaneous identification of all parameters – is perhaps the most straightforward and general option, since any numerical optimisation algorithm can be applied. The other categories are more heuristic, and the problem is split up into several subproblems, each of which might be easier to solve than attacking the complete problem directly. Generally, it is hard to say which strategy should be preferred, regarding the quality of the solutions and the computational complexity. Some comparisons exist, e.g., the one in [79] mentioned previously, but no extensive study comparing several methods and their performance has been found by the author.

There is also a trade-off between model complexity and the result quality. The more degrees of freedom that are built into the model structure, the closer the model can approximate the experimental data. However, since in practice data are more or less corrupted by noise, a too large degree of model flexibility might cause the model to adjust to the actual noise realisation, thereby causing overfitting. This is a general problem, occurring not only in piecewise affine system identification.

Finally, there is a trade-off between the computational complexity of the algorithm chosen, and the quality of the resulting model. This is most obvious in the algorithms where all parameters are identified simultaneously. The risk of ending up in a local minimum varies, depending on what numerical optimisation algorithm

is chosen. For example, having chosen a Gauss-Newton method, a simple way of increasing the chance of finding at least a good suboptimal model is to run the Gauss-Newton method several times, starting from different initial values. This will (on average) increase the quality of the solution, but at the same time increase the computational complexity. In the next chapter we will see another example of improving the result at the cost of increased complexity.

6

PWA Identification Using MILP/MIQP

The algorithms described in Chapter 5 all have one property in common: They cannot guarantee that the optimal solution, i.e., the global minimum of (5.2), is found. In this chapter, a type of algorithms that obtain the optimal solution in a finite number of steps is presented. The algorithms are based on MILP/MIQP, which were described in Section 2.3. The basic algorithms are described in Section 6.1, and some extensions are presented in Section 6.2.

It should be said from the beginning that the computational complexity of these algorithms is such that they are not a realistic alternative for identifying larger piecewise affine systems, or using large sets of experimental data. This subject is treated in more detail in Section 6.3. Another question is if it is really necessary to find the global optimum in practice, or if a good suboptimal solution will do. This issue becomes even more relevant when using piecewise affine systems as approximations to general nonlinear systems, and when the experimental data are corrupted by noise. Nevertheless, algorithms that guarantee convergence to the global optimum in finite time may still be of theoretical interest.

Furthermore, there are special cases where it might be feasible to use identification algorithms based on MILP/MIQP. One example is the problem of identifying a piecewise affine Wiener model, which is described in Section 6.4. Another idea is to use an MILP/MIQP solver to get a decent, suboptimal solution, which can be used as an initial value for a conventional optimisation algorithm. This is studied in Section 6.5.

The way we formulate the identification problem can also be used together with different methods similar to some of the approaches in Chapter 5. We will consider two Newton-like algorithms in Section 6.6.

6.1 Formulating the Identification Problem as an MILP/MIQP Problem

The system class we will consider in this section is the class of hinging hyperplane systems given by (3.10) and repeated here for convenience:

$$y(t) = \varphi^T(t)\theta_0 + \sum_{i=1}^{M} \pm \max\{\varphi^T(t)\theta_i, 0\} + e(t) \tag{6.1}$$

Recall that this is shorthand notation for

$$y(t) = \varphi^T(t)\theta_0 + \sum_{i=1}^{M^+} \max\{\varphi^T(t)\theta_i, 0\} - \sum_{i=M^+ +1}^{M} \max\{\varphi^T(t)\theta_i, 0\} + e(t)$$

where we assume M^+ and M to be known. As mentioned in Section 3.1.5, two properties of this class are that the piecewise affine function is continuous, and that the affine subsystems are separated by the hyperplanes $\varphi^T(t)\theta_i = 0$.

Letting $\theta = \begin{pmatrix}\theta_0^T & \dots & \theta_M^T\end{pmatrix}^T$, the predicted output $\hat{y}(t|\theta)$ becomes

$$\hat{y}(t|\theta) = \varphi^T(t)\theta_0 + \sum_{i=1}^{M} \pm \max\{\varphi^T(t)\theta_i, 0\} \tag{6.2}$$

As in Section 2.2, we are given $y(t)$, $\varphi(t)$, $t = 1, \dots, N$, and would like to minimise the prediction error according to some criterion function, i.e.,

$$\min_{\theta} \quad V(\theta, Z^N) = \sum_{t=1}^{N} \ell(\varepsilon(t, \theta)) \tag{6.3}$$

where

$$\varepsilon(t, \theta) = y(t) - \hat{y}(t|\theta) = y(t) - \left(\varphi^T(t)\theta_0 + \sum_{i=1}^{M} \pm \max\{\varphi^T(t)\theta_i, 0\}\right)$$

In this thesis, we will use

$$\ell_2(\varepsilon) = \varepsilon^2$$

and

$$\ell_1(\varepsilon) = |\varepsilon|$$

The main question of this section is: How can we rewrite these problems to fit into the mixed-integer programming (MILP/MIQP) framework? If we would be able to formulate the piecewise affine system identification problem as an MILP or MIQP, then the optimal solution could be found in a finite number of steps, e.g., by using the algorithm described in Section 2.3. There are two steps to accomplish that: To rewrite (6.3) as a linear/quadratic function of the unknowns by introducing auxiliary variables, and to rewrite the constraints of the auxiliary variables as linear inequalities plus the constraint on some of the variables to be discrete.

We will make the assumption that there are known bounds for the parameters θ_i. These bounds could be in any norm without affecting the later results, but for simplicity the Euclidean norm is chosen here, i.e.,

$$\|\theta_i\| \leq U_{\theta_i}, \quad i = 1, \ldots, M$$

This implies that the products $\varphi^T(t)\theta_i$ are bounded:

$$L_i(t) \triangleq -\|\varphi(t)\|U_{\theta_i} \leq \varphi^T(t)\theta_i \leq \|\varphi(t)\|U_{\theta_i} \triangleq U_i(t)$$

This is not a serious restriction, since the bounds can be chosen arbitrarily large. However, the tighter the bounds, the more efficiently the optimisation can be performed. Other types of bounds that could be imagined include componentwise bounds on θ_i, i.e.,

$$L_{\theta_i} \preccurlyeq \theta_i \preccurlyeq U_{\theta_i}$$

where $\preccurlyeq$ denotes componentwise inequalities, as usual. If this type of bounds is used, the definitions of $L_i(t)$ and $U_i(t)$ should be changed accordingly.

6.1.1 Reformulating the Criterion Function

Let us start with the criterion function. The problem here is that $\hat{y}(t|\theta)$ is a nonlinear function of θ. To eliminate that problem, introduce the auxiliary variables

$$z_i(t) = \max\{\varphi^T(t)\theta_i, 0\}, \quad i = 1, \ldots, M \tag{6.4}$$

and regard these as unknown variables in our minimisation. The criterion function for $\ell_2(\varepsilon)$ now takes the form

$$V_2 = \sum_{t=1}^{N} \left(y(t) - \left(\varphi^T(t)\theta_0 + \sum_{i=1}^{M} \pm z_i(t) \right) \right)^2 \tag{6.5}$$

which is quadratic in the unknowns θ_0 and $z_i(t)$. The corresponding function for $\ell_1(\varepsilon)$ is

$$V_1 = \sum_{t=1}^{N} \left| y(t) - \left(\varphi^T(t)\theta_0 + \sum_{i=1}^{M} \pm z_i(t) \right) \right| \tag{6.6}$$

This function is unfortunately neither linear, nor quadratic. However, there is a standard trick for rewriting functions like this as a linear function with linear constraints. Introduce the *slack variables* $s_1, \ldots, s_N$. Then, minimising V_1 is the same as solving

$$\begin{aligned} \min_{\theta,z,s} \quad & \sum_{t=1}^{N} s_t \\ \text{subj. to} \quad & s_t \geq y(t) - \varphi^T(t)\theta_0 - \sum_{i=1}^{M} \pm z_i(t) \\ & s_t \geq -y(t) + \varphi^T(t)\theta_0 + \sum_{i=1}^{M} \pm z_i(t) \\ & z_i(t) = \max\{\varphi^T(t)\theta_i, 0\} \end{aligned} \tag{6.7}$$

Now the criterion function is linear at the price of increasing the number of variables by N and the number of linear constraints by $2N$. However, in the context this is not a very serious complexity increase, since it is generally the discrete variables introduced in the next section that determine the complexity.

6.1.2 Reformulating the Constraints

By introducing the auxiliary variables $z_i(t)$, we have also introduced the additional nonlinear constraints (6.4). The next step in our reformulation process is to rewrite these constraints as linear constraints, with the help of additional discrete variables. To that end, define

$$\delta_i(t) = \begin{cases} 0 & \varphi^T(t)\theta_i < 0 \\ 1 & \varphi^T(t)\theta_i \geq 0 \end{cases} \quad i = 1, \ldots, M \tag{6.8}$$

This means that we can write $z_i(t)$ as

$$z_i(t) = \delta_i(t)\varphi^T(t)\theta_i$$

Notice also that geometrically, $\delta_i(t)$ tells us on what side of the ith hinge $\varphi(t)$ is positioned. Now we can use the method described in [7] to transform the constraints (6.4) and (6.8) to linear inequalities. However, there is a more efficient way, which will now be described. This transformation method is related to the method of transforming boolean expressions into linear inequalities, described in [68], and should be possible to use in many applications, where similar transformations are needed.

Consider $z_i(t)$ and $\delta_i(t)$ for a given i and t. In Figure 6.1, the feasible region for the two constraints (definitions) (6.4) and (6.8) is drawn as two solid lines. What we would like is to be able to express that region as the intersection between a polyhedron (the linear constraints) and the set $\delta_i(t) = \{0, 1\}$. The simplest

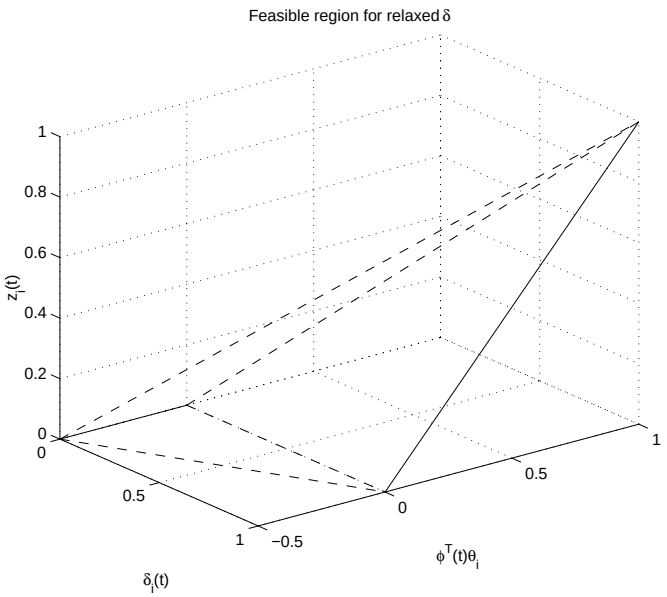


Figure 6.1: The feasible region for the constraints (6.4) and (6.8) in the $(\delta_i(t), \varphi^T(t)\theta_i, z_i(t))$ space (solid lines), and its convex hull (the dashed tetrahedron). Here we have assumed that $L_i(t) = -0.5$, $U_i(t) = 1$.

way of doing this is to construct the convex hull (see Section 2.1) of the feasible region (marked with dashed lines in Figure 6.1). This will both give the smallest polyhedron including the original feasible region and the smallest set of linear inequalities*. The first property is a general property of convex hulls, while the second property follows since four linear inequalities (together with the requirement $\delta \in \{0, 1\}$) are needed to define each of the two line segments of the feasible region. By choosing the linear inequalities to the ones defining the complex hull, both line segments are defined by the same inequalities.

At first sight, the convex hull seems to be defined by

$$\begin{cases} z_i(t) \geq 0 \\ z_i(t) \leq U_i(t)\delta_i(t) \\ \varphi^T(t)\theta_i \leq z_i(t) \\ (1 - \delta_i(t))L_i(t) + z_i(t) \leq \varphi^T(t)\theta_i \end{cases} \tag{6.9}$$

However, there is one problem in using these inequalities. According to (6.8), the point $(\delta_i(t), \varphi^T(t)\theta_i, z_i(t)) = (0, 0, 0)$ should not be part of the feasible region.

*Note that this only holds as long as we neglect the influence that variables from different timepoints have on each other, or as long as the bounds $L_i(t)$ and $U_i(t)$ are as tight as possible. Otherwise, the bounds on $\varphi^T(t_0)\theta_i$ may be reduced if bounds on $\varphi^T(t)\theta_i$ for other timepoints limit the possible values of θ_i. This would cause the feasible region to be smaller.

This means that the convex hull is described by the union of all regions obtained by different positive μ in the following inequalities

$$\begin{cases} z_i(t) \geq 0 \\ z_i(t) \leq U_i(t)\delta_i(t) \\ (1-\delta_i(t))\mu + \varphi^T(t)\theta_i \leq z_i(t) \\ (1-\delta_i(t))L_i(t) + z_i(t) \leq \varphi^T(t)\theta_i \end{cases} \qquad \mu > 0 \tag{6.10}$$

which is a description that might be awkward to work with, due to the non-fixed parameter μ. Actually, this is more of a theoretical than a practical problem. The reason for this is best understood geometrically: When $\varphi^T(t)\theta_i = 0$, it means that $\varphi(t)$ is lying on the ith hinge, and then it does not really matter to what side we assign it, i.e., we can let $\delta_i(t)$ be 0 or 1. Therefore, we can let $\mu = 0$.

Putting (6.5) and (6.9) together now gives us the following problem:

$$\begin{aligned} \min_{\theta,z,\delta} \quad & V_2 = \sum_{t=1}^{N} \left(y(t) - \left(\varphi^T(t)\theta_0 + \sum_{i=1}^{M} \pm z_i(t) \right) \right)^2 \\ \text{subj. to} \quad & z_i(t) \geq 0 \\ & z_i(t) \leq U_i(t)\delta_i(t) \\ & \varphi^T(t)\theta_i \leq z_i(t) \\ & (1-\delta_i(t))L_i(t) + z_i(t) \leq \varphi^T(t)\theta_i \\ & \delta_i(t) \in \{0,1\} \end{aligned} \tag{6.11}$$

This is an MIQP problem, and we are done with the reformulation.

In the same way, minimising V_1 can be reformulated as an MILP problem using (6.7) and (6.9):

$$\begin{aligned} \min_{\theta,z,\delta,s} \quad & \sum_{t=1}^{N} s_t \\ \text{subj. to} \quad & s_t \geq y(t) - \varphi^T(t)\theta_0 - \sum_{i=1}^{M} \pm z_i(t) \\ & s_t \geq \varphi^T(t)\theta_0 + \sum_{i=1}^{M} \pm z_i(t) - y(t) \\ & z_i(t) \geq 0 \\ & z_i(t) \leq U_i(t)\delta_i(t) \\ & \varphi^T(t)\theta_i \leq z_i(t) \\ & (1-\delta_i(t))L_i(t) + z_i(t) \leq \varphi^T(t)\theta_i \\ & \delta_i(t) \in \{0,1\} \end{aligned} \tag{6.12}$$

6.1.3 Restricting the Search Space

We have now reached the MILP/MIQP formulations, so in principle, we are ready to apply an MILP/MIQP solver to find the optimal parameter values. However, as noted in Section 3.1.5, the parameter values are not uniquely determined in the expressions above, which means that there will be several optimal solutions, all describing the same system. To avoid this, we can restrict the search space by requiring that $0 \leq w^T\theta_1 \leq \cdots \leq w^T\theta_{M^+}$ and $0 \leq w^T\theta_{M^++1} \leq \cdots \leq w^T\theta_M$ for a given constant (nonzero) vector w, just as in Section 3.1.5.

An interesting choice of w is $w = -\varphi(t)$ for some t, say, $t = 1$. This gives the inequalities

$$\begin{aligned} &\varphi^T(1)\theta_{M^+} \leq \cdots \leq \varphi^T(1)\theta_1 \leq 0 \\ &\varphi^T(1)\theta_M \leq \cdots \leq \varphi^T(1)\theta_{M^++1} \leq 0 \end{aligned} \tag{6.13}$$

that should be included in (6.11) and (6.12). In particular, we can set

$$\begin{cases} \delta_i(1) = 0 \\ z_i(1) = 0 \end{cases} \qquad i = 1, \ldots, M$$

which means that we can exclude these $2M$ variables from the optimisation, thus getting a somewhat smaller optimisation problem.

It should be emphasised that this reduction of the search space not necessarily leads to a faster computation time, even if it often helps. This is something that needs to be investigated more thoroughly. See [97] for a discussion of the relations between number of variables, number of constraints, and computation time.

6.1.4 Some Examples

Let us now look at some examples of using MILP/MIQP for piecewise affine system identification.

Example 6.1 *In the following example, we consider a noiseless system described by*

$$\begin{aligned} y(t) = y(t-1) - u(t-1) + 2 + \\ + \max\{y(t-1) + u(t-1) - 1, 0\} - \\ - \max\{u(t-1) - 1, 0\} \end{aligned} \tag{6.14}$$

which we can rewrite as

$$y(t) = \varphi^T(t)\theta_0 + \max\{\varphi^T(t)\theta_1, 0\} - \max\{\varphi^T(t)\theta_2, 0\} \tag{6.15}$$

with

$$\varphi(t) = \begin{pmatrix} 1 \\ -y(t-1) \\ u(t-1) \end{pmatrix} \tag{6.16}$$

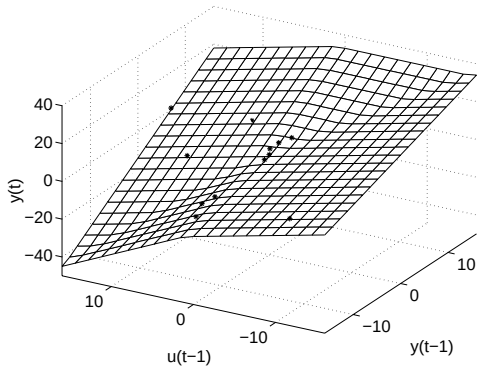


Figure 6.2: The function defined in (6.14) and the experimental data.

and

$$\theta_0 = \begin{pmatrix} 2 \\ -1 \\ -1 \end{pmatrix}, \quad \theta_1 = \begin{pmatrix} -1 \\ -1 \\ 1 \end{pmatrix}, \quad \theta_2 = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \tag{6.17}$$

which are the parameters to be identified. The set of experimental data consists of $(u(t), y(t))$, $t = 0, \ldots, 12$, *and is persistently exciting. In Figure 6.2, the function to be identified and the data are plotted.*

Let us now formulate this system in a form suitable for the MILP/MIQP identification algorithms. Following (6.4) *and* (6.8)*, we start by introducing*

$$z_i(t) = \max\{\varphi^T(t)\theta_i, 0\}, \quad i = 1, 2$$

and

$$\delta_i(t) = \begin{cases} 0 & \varphi^T(t)\theta_i < 0 \\ 1 & \varphi^T(t)\theta_i \geq 0 \end{cases}, \quad i = 1, 2$$

Table 6.1 shows the correct values of $\delta_i(t)$*, which show that the data are well distributed over the different modes. Now, depending on what criterion function we would like to use, we can formulate the identification problem exactly as in* (6.11) *or* (6.12)*. We can also add the restrictions* (6.13)*. Then a conventional MILP/MIQP solver can be used to identify the parameters. The computation time for this experiment was a few seconds, and a few hundred nodes in the MILP/MIQP search trees were visited before the correct parameter values were found (100-800 nodes, depending on implementation and strategy of the MILP/MIQP solvers).*

The following example is taken from [8].

t	$u(t-1)$	$y(t-1)$	$\delta_1(t)$	$\delta_2(t)$
1	0	0	0	0
2	0	2	1	0
3	5	5	1	1
4	16	7	1	1
5	10	0	1	1
6	2	-8	0	1
7	3	-9	0	1
8	2	-12	0	1
9	-10	-13	0	0
10	0	-1	0	0
11	0.5	1	1	0
12	-1	3	1	0

Table 6.1: The experimental data, and the correct values of $\delta_i(t)$.

Example 6.2 *Consider the system*

$$\begin{aligned} y(t) = 0.8y(t-1) + 0.4u(t-1) - 0.1 + \\ + \max\{-0.3y(t-1) + 0.6u(t-1) + 0.3, 0\} \end{aligned} \tag{6.18}$$

The model is identified on the data reported in Figure 6.4(a), by solving an MILP with 66 variables (of which 20 integers) and 168 constraints. For this example, the translation into inequalities given by [7] was used instead of the method in (6.9)*. The problem was solved by using* Cplex *[45] (1014 LP solved in 0.68 s on a Sun Ultra 10 running* Matlab *5.3), and, for comparison, was solved again using BARON [84] (73 LP solved in 3.00 s, same machine), which results in a zero output prediction error (Figure 6.4(b)). The fitted hinging hyperplane model is shown in Figure 6.3. By adding a random Gaussian noise with zero mean and variance 0.1 on the measured output $y(t)$, the following model*

$$\begin{aligned} y(t) = 0.8315y(t-1) + 0.3431u(t-1) - 0.2014 + \\ + \max\{-0.3391y(t-1) + 0.6205u(t-1) + 0.3977, 0\} \end{aligned} \tag{6.19}$$

was identified in 1.39 s (3873 LP solved) using Cplex *(7.86 s, 284 LP using BARON) on the estimation set reported in Figure 6.5(a), and produces the validation data reported in Figure 6.5(b). For comparison, a linear ARX model was identified on the same estimation data, with the following result*

$$y(t) = 0.8250y(t-1) + 0.7217u(t-1) \tag{6.20}$$

The results for validation data are found in Figure 6.6 (higher order ARX models did not produce significant improvements). Clearly, the error generated by driving

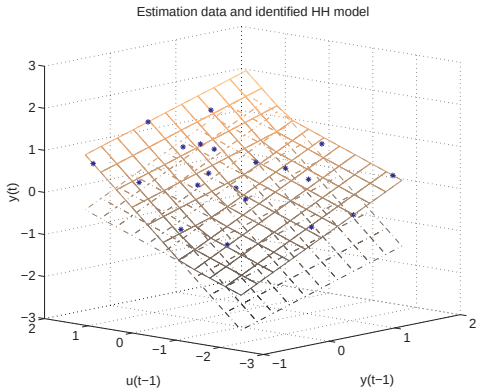
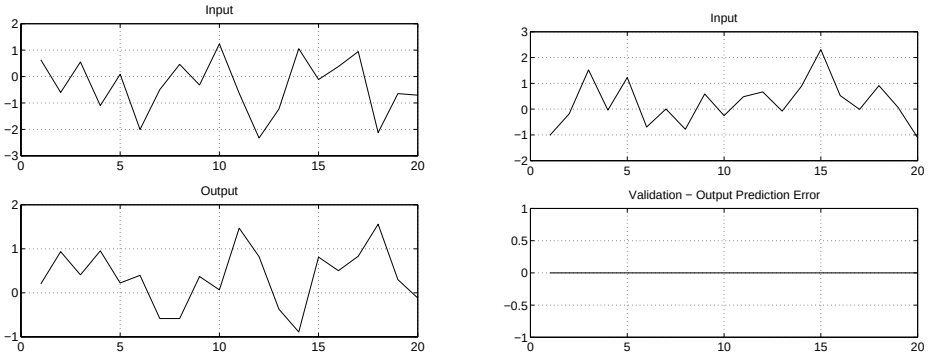


Figure 6.3: Identification of model (6.18) – noiseless case. Identified hinging hyperplane model.



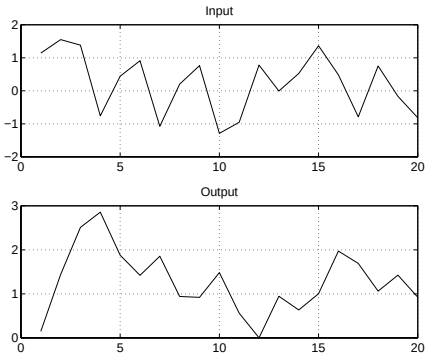
(a) Estimation data.

(b) Validation data.

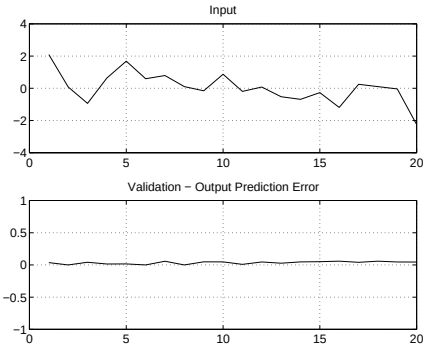
Figure 6.4: Identification of model (6.18) – noiseless case.

the ARX model in open-loop with the validation input $u(t)$ is much larger, and would not make (6.20) *suitable for instance for formal verification tools, where a good performance of open-loop prediction is a critical requirement.*

In Figure 6.7 we compare the performance in terms of LP/QPs and total computation time of the linear criterion (6.6) *versus the quadratic criterion* (6.5)*. The reported numbers are computed on a Sun Ultra 60 (2×360 MHz) running* MATLAB *5.3 and the solver BARON [84], by averaging the result of ten estimation data sets generated by feeding random Gaussian inputs $u(t)$ and zero output noise to system* (6.18)*.*



(a) Estimation data.



(b) Validation data.

Figure 6.5: Identification of model (6.18) – noisy case, $y(t)$ is perturbed by a random Gaussian noise with zero mean and variance 0.1.

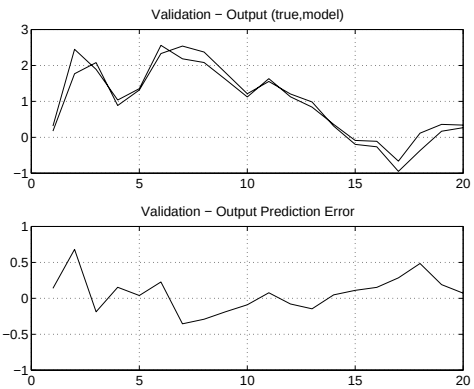
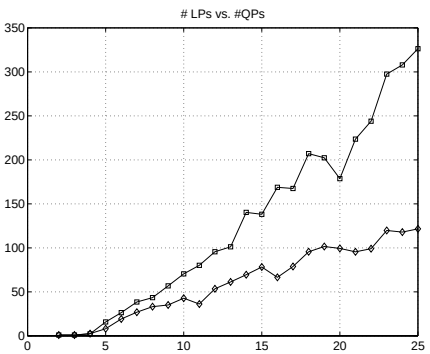


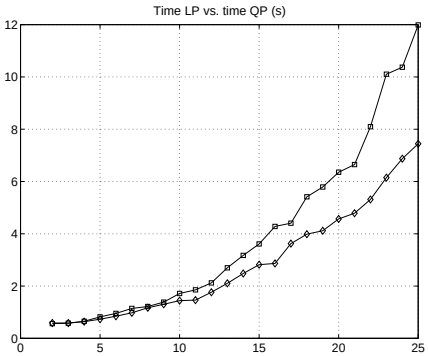
Figure 6.6: Identification of a linear ARX model – same estimation and validation data as in Figure 6.5.

6.2 Extensions

The MILP/MIQP formulations in Section 6.1 assumed that the system could be described by hinging hyperplane functions with a known number of max functions (M known), and known signs of the max functions (M^+ known). In this section, several extensions will be considered. Firstly, the case when M^+ is not known will be discussed. Secondly, we will extend the hinging hyperplane models by allowing various forms of discontinuities. We will also consider a kind of robust hinging hyperplane models. In the last extension, the identification problem for general PWARX systems (described in Section 3.1.4) will be considered. The extensions in Sections 6.2.2 and 6.2.3 are taken from [8].



(a) Average number of LPs and QPs.



(b) Average computation time (Sun Ultra 60 (2 × 360 MHz) running Matlab 5.3 and the solver BARON).

Figure 6.7: Identification of model (6.18) – MILP vs MIQP (results are averaged on ten estimation data sets generated by random Gaussian inputs $u(t)$ and zero output noise). The horizontal axes show the number of estimation data samples. The results from MILP are marked with diamonds, and the results from MIQP are marked with squares.

Only the quadratic criterion function V_2 will be considered. The reformulation of V_1 follows immediately, by using slack variables as in (6.7).

6.2.1 Unknown Number of Positive Max Functions

When M^+ is unknown, the sign of each max function has to be determined. Here, two ways of doing this will be described. In the first method, we introduce M discrete variables $\sigma_i \in \{0, 1\}$, and rewrite the predicted output as

$$\hat{y}(t, \theta, \sigma) = \varphi^T(t)\theta_0 + \sum_{i=1}^{M}(2\sigma_i - 1)\max\{\varphi^T(t)\theta_i, 0\} \tag{6.21}$$

Using the same definition of $z_i(t)$ as before (see (6.4)) yields

$$\hat{y}(t, \theta, \sigma) = \varphi^T(t)\theta_0 + \sum_{i=1}^{M}(2\sigma_i - 1)z_i(t) \tag{6.22}$$

This expression is still not linear in the unknowns, which means that the criterion functions will not be so either. Therefore we define

$$\zeta_i(t) = \sigma_i z_i(t), \quad i = 1, \ldots, M \tag{6.23}$$

and get

$$\hat{y}(t,\theta,\sigma) = \varphi^T(t)\theta_0 + \sum_{i=1}^{M}(2\zeta_i(t) - z_i(t)) \tag{6.24}$$

The criterion function V_2 becomes

$$V_2 = \sum_{t=1}^{N}\left(y(t) - \left(\varphi^T(t)\theta_0 + \sum_{i=1}^{M}(2\zeta_i(t) - z_i(t))\right)\right)^2 \tag{6.25}$$

which is quadratic in θ_0, $\zeta_i(t)$ and $z_i(t)$.

What remains now is to formulate the definitions of $\sigma_i(t)$, $z_i(t)$, and $\zeta_i(t)$ as linear inequalities. For this we need the auxiliary discrete variables $\delta_i(t)$, defined as in (6.8). Following a similar line of reasoning as in Section 6.1, one arrives at the inequalities

$$\begin{cases} \zeta_i(t) \geq 0 \\ \zeta_i(t) \leq z_i(t) \\ z_i(t) \leq U_i(t)\delta_i(t) \\ \zeta_i(t) \leq U_i(t)\sigma_i \\ (1-\delta_i(t))L_i(t) + z_i(t) \leq \varphi^T(t)\theta_i \\ \varphi^T(t)\theta_i \leq z_i(t) \\ z_i(t) - \zeta_i(t) \leq U_i(t)(1-\sigma_i) \end{cases} \tag{6.26}$$

which can easily be checked by testing all possible values of $(\delta_i(t), \sigma_i)$.

Since M^+ is unknown, we cannot include the inequalities (6.13) right away. However, we can restrict the search space to some extent by including

$$\varphi^T(1)\theta_i \leq 0, \quad i = 1, \ldots, M \tag{6.27}$$

Furthermore, we can let

$$\sigma_{i+1} \leq \sigma_i, \quad i = 1, \ldots, M-1 \tag{6.28}$$

to order the max functions, so that the positive max functions come first.

An alternative way of dealing with the problem of unknown M^+ is to write the predicted output as

$$\hat{y}(t|\theta) = \varphi^T(t)\theta_0 + \sum_{i=1}^{M}(\max\{\varphi^T(t)\theta_i^+, 0\} - \max\{\varphi^T(t)\theta_i^-, 0\}) \tag{6.29}$$

where we require that

$$\varphi^T(t)\theta_i^+ \geq 0 \quad \Leftrightarrow \quad \varphi^T(t)\theta_i^- \geq 0 \tag{6.30}$$

This can be regulated with the help of the discrete variables $\delta_i(t)$, where

$$\delta_i(t) = 1 \quad \Leftrightarrow \quad \varphi^T(t)\theta_i^+ \geq 0 \quad \Leftrightarrow \quad \varphi^T(t)\theta_i^- \geq 0 \tag{6.31}$$

Furthermore, we introduce the auxiliary variables

$$z_i(t) = \max\{\varphi^T(t)\theta_i^+, 0\} - \max\{\varphi^T(t)\theta_i^-, 0\} = \delta_i(t)\varphi^T(t)(\theta_i^+ - \theta_i^-) \tag{6.32}$$

We also need the lower and upper bounds

$$\begin{aligned} L_i^+(t) &\leq \varphi^T(t)\theta_i^+ \leq U_i^+(t) \\ L_i^-(t) &\leq \varphi^T(t)\theta_i^- \leq U_i^-(t) \\ L_i(t) &\leq \varphi^T(t)(\theta_i^+ - \theta_i^-) \leq U_i(t) \end{aligned}$$

The problem can now be written as

$$\begin{aligned} \min_{\theta^{\pm},z,\delta} \quad & V_2 = \sum_{t=1}^{N} \left(y(t) - \left(\varphi^T(t)\theta_0 + \sum_{i=1}^{M} z_i(t) \right) \right)^2 \\ \text{subj. to} \quad & \varphi^T(t)\theta_i^+ \geq L_i^+(t)(1-\delta_i(t)) \\ & \varphi^T(t)\theta_i^+ \leq U_i^+(t)\delta_i(t) \\ & \varphi^T(t)\theta_i^- \geq L_i^-(t)(1-\delta_i(t)) \\ & \varphi^T(t)\theta_i^- \leq U_i^-(t)\delta_i(t) \\ & z_i(t) \geq L_i(t)\delta_i(t) \\ & z_i(t) \leq U_i(t)\delta_i(t) \\ & \varphi^T(t)(\theta_i^+ - \theta_i^-) \geq z_i(t) + L_i(t)(1-\delta_i(t)) \\ & \varphi^T(t)(\theta_i^+ - \theta_i^-) \leq z_i(t) + U_i(t)(1-\delta_i(t)) \\ & \delta_i(t) \in \{0,1\} \end{aligned} \tag{6.33}$$

Here the method from [7] is used to transform the definitions of $z_i(t)$ and $\delta_i(t)$ into inequalities. The first four inequalities establish the relation between θ_i^+, θ_i^-, and $\delta_i(t)$, while the last four inequalities define $z_i(t)$.

The advantage of this method compared to the first alternative is that we need fewer variables. The drawback is that θ_i^+ and θ_i^- cannot be uniquely determined; if $(\theta_i^{+*}, \theta_i^{-*})$ is a solution, so is, e.g., $(\theta_i^{+*} + \lambda\theta_i^{+*}, \theta_i^{-*} + \lambda\theta_i^{+*})$.

To restrict the search space, we can include the following inequalities that correspond to (6.13):

$$\begin{cases} \varphi^T(1)\theta_i^+ \leq 0 & i = 1, \ldots, M \\ \varphi^T(1)(\theta_1^+ - \theta_1^-) \leq \cdots \leq \varphi^T(1)(\theta_M^+ - \theta_M^-) \end{cases}$$

which makes the variables $\delta_i(1)$, $z_i(1)$ redundant.

6.2.2 Discontinuous Hinging Hyperplane Models

In hinging hyperplane models, the output $y(t)$ is a continuous function of the regressor $\varphi(t)$. On the other hand, hybrid systems often consist of piecewise affine discontinuous mappings. In order to tackle discontinuities, we can modify the hinging hyperplane model (6.2) in the form

$$\hat{y}(t|\theta) = \varphi^T(t)\theta_0 + \sum_{i=1}^{M}(\varphi^T(t)\theta_i + a_i)\delta_i(t) \tag{6.34a}$$

$$\delta_i(t) = \begin{cases} 0 & \varphi^T(t)\theta_i < 0 \\ 1 & \varphi^T(t)\theta_i \geq 0 \end{cases} \tag{6.34b}$$

where a_i, $i = 1, \ldots, M$ are additional free parameters, $a_i^- \leq a_i \leq a_i^+$. This modification allows a discontinuity of constant size along each hinge. A more general class is obtained by using

$$\hat{y}(t|\theta) = \varphi^T(t)\theta_0 + \sum_{i=1}^{M}\varphi^T(t)\theta_i\delta_i(t) \tag{6.35a}$$

$$\delta_i(t) = \begin{cases} 0 & C_i\varphi(t) < 0 \\ 1 & C_i\varphi(t) \geq 0 \end{cases} \tag{6.35b}$$

where C_i, $i = 1, \ldots, M$ are additional free vectors of parameters, controlling the partitioning of the state-space independently of θ_i. By introducing new continuous variables $z_i(t)$

$$z_i(t) = (\varphi^T(t)\theta_i + a_i)\delta_i(t)$$

or

$$z_i(t) = \varphi^T(t)\theta_i\delta_i(t)$$

respectively, both the problems of identifying (6.34) and (6.35) can again be recast as an MILP/MIQP. Using the transformations from [7], from (6.35) we get

$$\begin{aligned} \min_{\theta,z,\delta,C} \quad & V_2 = \sum_{t=1}^{N}\left(y(t) - \left(\varphi^T(t)\theta_0 + \sum_{i=1}^{M} z_i(t)\right)\right)^2 \\ \text{subj. to} \quad & C_i\varphi(t) \geq L_{C_i}(t)(1-\delta_i(t)) \\ & C_i\varphi(t) \leq U_{C_i}(t)\delta_i(t) - (1-\delta_i(t))\mu \\ & z_i(t) \geq L_i(t)\delta_i(t) \\ & z_i(t) \leq U_i(t)\delta_i(t) \\ & \varphi^T(t)\theta_i \geq z_i(t) + L_i(t)(1-\delta_i(t)) \\ & \varphi^T(t)\theta_i \leq z_i(t) + U_i(t)(1-\delta_i(t)) \\ & \delta_i(t) \in \{0,1\} \end{aligned} \tag{6.36}$$

where μ is a small number (just as in (6.10)), which we in practice can choose, e.g., to the machine precision; and $L_{C_i}(t)$ and $U_{C_i}(t)$ are lower and upper bounds on $C_i\varphi(t)$.

Note that the problem (6.36) mostly does not have a unique solution. For instance, once θ_i, $z_i(t)$, and $\delta_i(t)$ have been fixed, in general there exist infinitely many vectors C_i satisfying the constraints (6.35b) (cf. the general PWARX systems in Section 6.2.4).

6.2.3 Robust Hinging Hyperplane Models

As discussed in Chapter 4, in formal verification methods, model uncertainty needs to be handled in order to provide safety guarantees. Typically, the model is associated with a bounded uncertainty. In the context of symbolic solvers for timed automata, differential inclusions $a \leq \dot{x} \leq b$ are handled, for instance by the solver HyTech [40]. As another example, for piecewise affine and MLD systems, the verification algorithm proposed in [9] handles model uncertainty as additive input disturbances entering a nominal model.

In the present context of hinging hyperplane models, we wish to find an uncertainty description of the form

$$\varphi^T(t)\theta_0^- + \sum_{i=1}^{M} \pm \max\{\varphi^T(t)\theta_i^-, 0\} \leq y(t) \leq \varphi^T(t)\theta_0^+ + \sum_{i=1}^{M} \pm \max\{\varphi^T(t)\theta_i^+, 0\} \tag{6.37}$$

(that should hold for all t) for an inclusion-type of description, or the form

$$y(t) = \varphi^T(t)\theta_0 + \sum_{i=1}^{M} \pm \max\{\varphi^T(t)\theta_i, 0\} + e(t), \quad e^- \leq e(t) \leq e^+ \tag{6.38}$$

for an additive-disturbance-type of description. With a finite number of data samples, it is naturally impossible to guarantee the inclusions of (6.37) or (6.38) to hold for any kind of input and initial conditions, so what will be proposed here are methods for making the inclusions hold at least for the experimental data.

If we consider the inclusion-type of uncertainty description (6.37), a pair of extreme models with parameters θ^-, θ^+ can be obtained by solving (6.11) or (6.12) with the additional linear constraints

$$y(t) \geq \varphi^T(t)\theta_0^- + \sum_{i=1}^{M} \pm z_i(t), \quad t = 1, \ldots, N \tag{6.39}$$

for estimating θ^-, and

$$y(t) \leq \varphi^T(t)\theta_0^+ + \sum_{i=1}^{M} \pm z_i(t), \quad t = 1, \ldots, N \tag{6.40}$$

for estimating θ^+, respectively. Turning to the additive-disturbance description of the form (6.38), it can computed in two alternative ways:

1. First, identify the model parameters $\hat{\theta}$ by solving (6.11) or (6.12), and then compute

$$\begin{aligned} e^+ &\triangleq \max_{t=1,\ldots,N} y(t) - \hat{y}(t,\hat{\theta}) \\ e^- &\triangleq \min_{t=1,\ldots,N} y(t) - \hat{y}(t,\hat{\theta}) \end{aligned} \tag{6.41}$$

2. Modify the MILP (6.12) by using only one slack variable s, i.e., replace all entries of s_t in (6.12) by s. The objective function that should be minimised simply becomes s. The corresponding optimum then provides a nominal model such that the bound on the norm of the additive disturbance $e(t)$ is minimised.

6.2.4 General PWARX Systems

In Section 3.1.4, PWARX systems were defined as being of the following type:

$$y(t) = \varphi^T(t)\theta(v) + e(t) \tag{6.42}$$

where the key vector v is a function of $\varphi(t)$, $\varphi(t)$ consists of old inputs and outputs plus a constant element, and the regions of the different subsystems are polyhedral. We assume that the regions are separated by M hyperplanes $\{\varphi \in \mathbb{R}^n \mid C_i\varphi = 0\}$, $i = 1, \ldots, M$, where $C_i \in \mathbb{R}^{1\times n}$ are unknown normal vectors of the hyperplanes, and n is the dimension of φ. In order to better fit into the framework of this chapter, v will be defined slightly different compared to Chapter 4:

$$v_i = \begin{cases} 0 & \text{if } C_i\varphi < 0 \\ 1 & \text{if } C_i\varphi \geq 0 \end{cases}$$

We can notice directly that in most cases, it will be impossible to determine C_i uniquely from the experimental data, since there is a continuum of hyperplanes which will split the set of vectors $\varphi(t)$ in the same partitions.

Let us now consider the special case of $M = 2$ for the remainder of the section. The reformulation for other values of M is completely analogous. First we can write $\hat{y}(t|\theta)$ as

$$\begin{aligned} \hat{y}(t|\theta) &= \varphi^T(t)\theta(v) = \\ &= \varphi^T(t)\Big[\theta(\begin{pmatrix}0\\0\end{pmatrix})(1-v_1(t))(1-v_2(t)) + \theta(\begin{pmatrix}1\\0\end{pmatrix})v_1(t)(1-v_2(t)) + \\ &\qquad + \theta(\begin{pmatrix}0\\1\end{pmatrix})(1-v_1(t))v_2(t) + \theta(\begin{pmatrix}1\\1\end{pmatrix})v_1(t)v_2(t)\Big] \end{aligned}$$

Rearranging the terms gives

$$\begin{aligned}\hat{y}(t|\theta) = \varphi^T(t) \Bigg[& \theta(\begin{pmatrix}0\\0\end{pmatrix}) + v_1(t) \left(-\theta(\begin{pmatrix}0\\0\end{pmatrix}) + \theta(\begin{pmatrix}1\\0\end{pmatrix}) \right) + \\ & + v_2(t) \left(-\theta(\begin{pmatrix}0\\0\end{pmatrix}) + \theta(\begin{pmatrix}0\\1\end{pmatrix}) \right) + \\ & + v_1(t)v_2(t) \left(\theta(\begin{pmatrix}0\\0\end{pmatrix}) - \theta(\begin{pmatrix}1\\0\end{pmatrix}) - \theta(\begin{pmatrix}0\\1\end{pmatrix}) + \theta(\begin{pmatrix}1\\1\end{pmatrix}) \right) \Bigg] \triangleq \\ \triangleq \varphi^T(t) \Big[& \tilde{\theta}_0 + v_1(t)\tilde{\theta}_1 + v_2(t)\tilde{\theta}_2 + v_1(t)v_2(t)\tilde{\theta}_3 \Big]\end{aligned}$$

where the new variables $\tilde{\theta}_0, \ldots, \tilde{\theta}_3$ have been introduced as different linear combinations of $\theta(v)$ for the different values of v. Now we can identify $\tilde{\theta}_0, \ldots, \tilde{\theta}_3$, and then solve for $\theta(v)$, but first the expression for $\hat{y}$ must be linear in the unknown variables. To accomplish that, introduce

$$\begin{aligned} z_1(t) &= v_1(t)\varphi^T(t)\tilde{\theta}_1 \\ z_2(t) &= v_2(t)\varphi^T(t)\tilde{\theta}_2 \\ z_3(t) &= v_2(t)\varphi^T(t)\tilde{\theta}_3 \\ z_4(t) &= v_1(t)z_3(t) \end{aligned}$$

to get

$$\hat{y}(t|\theta) = \varphi^T(t)\tilde{\theta}_0 + z_1(t) + z_2(t) + z_4(t)$$

and the quadratic criterion function

$$V_2 = \sum_{t=1}^{N} \Big(y(t) - \Big(\varphi^T(t)\tilde{\theta}_0 + z_1(t) + z_2(t) + z_4(t) \Big) \Big)^2$$

What is left now is to rewrite the definitions of $z_j(t)$ as linear inequalities. Since the values of $v_j(t)$ do not depend on $\tilde{\theta}_k$, but on C_i, we cannot use the same procedure as in Section 6.1.2. Instead, the method in [7] is used again, which yields

$$\begin{cases} C_i\varphi(t) \geq L_{C_i}(t)(1 - v_i(t)) \\ C_i\varphi(t) \leq U_{C_i}(t)v_i(t) - (1 - v_i(t))\mu \end{cases} \quad i = 1, 2$$

$$\begin{cases} z_j(t) \leq U_{\tilde{\theta}_j}(t)v_j(t) \\ z_j(t) \geq L_{\tilde{\theta}_j}(t)v_j(t) \\ \varphi^T(t)\tilde{\theta}_j \geq z_j(t) + L_{\tilde{\theta}_j}(t)(1 - v_j(t)) \\ \varphi^T(t)\tilde{\theta}_j \leq z_j(t) + U_{\tilde{\theta}_j}(t)(1 - v_j(t)) \end{cases} \quad j = 1, 2$$

$$\begin{cases} z_3(t) \le U_{\tilde{\theta}_3}(t) v_2(t) \\ z_3(t) \ge L_{\tilde{\theta}_3}(t) v_2(t) \\ \varphi^T(t)\tilde{\theta}_3 \ge z_3(t) + L_{\tilde{\theta}_3}(t)(1 - v_2(t)) \\ \varphi^T(t)\tilde{\theta}_3 \le z_3(t) + U_{\tilde{\theta}_3}(t)(1 - v_2(t)) \end{cases}$$

$$\begin{cases} z_4(t) \le U_{\tilde{\theta}_3}(t) v_1(t) \\ z_4(t) \ge L_{\tilde{\theta}_3}(t) v_1(t) \\ z_3(t) \ge z_4(t) + L_{\tilde{\theta}_3}(t)(1 - v_1(t)) \\ z_3(t) \le z_4(t) + U_{\tilde{\theta}_3}(t)(1 - v_1(t)) \end{cases}$$

Here, μ is a small number, e.g., the machine precision. $U_{C_i}(t)$, $L_{C_i}(t)$, $U_{\tilde{\theta}_i}(t)$, and $L_{\tilde{\theta}_i}(t)$ are upper and lower bounds on $C_i\varphi(t)$ and $\varphi^T(t)\tilde{\theta}_i$, respectively, which can be derived from bounds on C_i and $\theta(v)$ given a priori.

To restrict the search space a bit more, we can also, e.g., require that

$$C_i\varphi(1) = -1, \quad i = 1, 2$$

which also implies that $v_1(1) = v_2(1) = 0$ and $z_j(1) = 0$, $j = 1, \ldots, 4$.

6.3 Computational Complexity and Theoretical Aspects

As mentioned in Section 2.3, the general MILP or MIQP problem is $\mathcal{NP}$-hard, so that, in worst case, every combination of δ values has to be tested. However, in practice not all combinations of δ values are feasible, which improves the complexity a bit. This is the subject of the following section. We will only consider the systems of Section 6.1, although some of the discussion is relevant also to the extensions of Section 6.2. No rigorous complexity analysis will be made; instead some aspects of it will be discussed.

6.3.1 Number of Feasible δ Combinations

Since the identification problem is easily solved once we know the correct partitioning of the data (it reduces to an LP or a convex QP problem), we can look at the optimisation process as trying to find the best way of partitioning the set of vectors $\varphi(t)$, i.e., to partition the regression vector space into different regions. As we have also seen, $\delta_i(t)$ describes on what side of the ith hinge that $\varphi(t)$ is positioned. In other words, for each t, $\delta_1(t), \ldots, \delta_M(t)$ together work as a key vector, telling us in what region $\varphi(t)$ is situated. Each leaf in the MILP/MIQP search trees corresponds to a certain combination of δ values that is to be tested. However, in all cases of interest, many combinations of values of the $\delta_i(t)$ variables will be infeasible, i.e., they will not correspond to a possible partitioning of the state-space. For example, if a regression vector $\varphi(t_0)$ belongs to the convex hull of some other regression vectors $\varphi(t_1), \ldots, \varphi(t_k)$ (see Figure 6.8), and we assign the

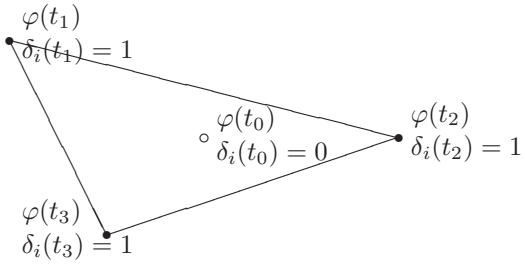


Figure 6.8: An infeasible assignment of $\delta_i(t)$ values. No hyperplane can separate $\varphi(t_0)$ from $\varphi(t_1)$, $\varphi(t_2)$, and $\varphi(t_3)$, so if $\delta_i(t_1) = \delta_i(t_2) = \delta_i(t_3) = 1$, then we must also have $\delta_i(t_0) = 1$.

values $\delta_i(t_1) = \delta_i(t_2) = \cdots = \delta_i(t_k) = 1$, then $\delta_i(t_0) = 1$ by necessity, since no hyperplane $\varphi^T\theta_i = 0$ can separate $\varphi(t_0)$ from $\varphi(t_1), \ldots, \varphi(t_k)$. The natural question arises: How many feasible combinations of $\delta_i(t)$ values are there?

If the dimension of $\varphi(t)$ equals $n+1$, the question can also be formulated as: In how many ways can we group N points in $\mathbb{R}^n$ (that is, the points composed by elements number 2 to $n+1$ of $\varphi(t)$, $t = 1, \ldots, N$) by separating them with M hyperplanes? This question has already been answered in Corollary 2.2. Note that $\varphi(t)$ corresponds exactly to the vectors used in the proof of Corollary 2.1, since the first element of $\varphi(t)$ always equals 1.

To be able to use Corollary 2.2, we need the data to be in general position, but since – according to Section 2.4 – for an arbitrarily chosen set of points in $\mathbb{R}^n$, this condition should be satisfied with probability one, the restriction should not be too serious, especially if we assume white noise in our experimental data.

We assume that M and M^+ are known, and that the data set is persistently exciting. This implies that trivial partitions like having one hinge going outside the data set, or several hinges positioned at the same place, are – albeit feasible – unnecessary to try.

We also have to remember that some of the hinge functions should be positive and some negative. This will increase the number of δ combinations that have to be tested, since at each possible position for a hinge, both a positive and a negative hinge function has to be tested.

We summarise the previous discussion in the following corollary:

Corollary 6.1
Assume that N samples of data in general position are given, and that a hinging hyperplane model with M^+ positive and $M^- = M - M^+$ negative hinge functions should be fitted to it, using (6.11) *or* (6.12)*. Then the number of combinations of values of $\delta_i(t)$ that have to be tested equals*

$$f_\delta(n, N, M, M^+) = \binom{f(n+1, N)/2 - 1}{M} \cdot \binom{M}{M^+} \tag{6.43}$$

	(n, M)						
N	$(1,1)$	$(1,2)$	$(1,3)$	$(2,1)$	$(2,2)$	$(2,3)$	$(3,1)$
20	19	171	969	190	17955	1125180	1159
50	49	1176	18424	1225	749700	$3.1 \cdot 10^8$	19649
100	99	4851	156849	4950	$1.2 \cdot 10^7$	$2.0 \cdot 10^{10}$	161799
200	199	19701	1293699	19900	$2.0 \cdot 10^8$	$1.3 \cdot 10^{12}$	1313599
500	499	124251	$2.1 \cdot 10^7$	124750	$7.8 \cdot 10^9$	$3.2 \cdot 10^{14}$	$2.1 \cdot 10^7$

Table 6.2: Number of feasible δ combinations for some different values of n, M, and N. Here we assume $M^+ = M$.

where

$$f(n, N) = 2 \sum_{k=0}^{n-1} \binom{N-1}{k}$$

Proof According to Corollary 2.2, there are $\binom{f(n+1,N)/2-1}{M}$ nontrivial ways in which a set of N points in $\mathbb{R}^n$ can be partitioned by M hyperplanes. For each partition, we should decide which of the M hyperplanes that should be hinges of the positive hinge functions. In other words, out of M hinge functions, we should choose M^+ to be positive. □

A MILP/MIQP search tree with only the feasible, nontrivial solutions as leaves, and the other cut away, would therefore have $f_\delta(n, N, M, M^+)$ leaves, and consequently contain $2f_\delta(n, N, M, M^+) - 1$ nodes altogether. Table 6.2 lists some values of $f_\delta(n, N, M, M^+)$. However, one should keep in mind that the computation of $f_\delta(n, N, M, M^+)$ just considers the feasibility of different δ combinations. Since the standard MILP/MIQP algorithms (such as branch-and-bound) can throw away obviously bad solutions, much fewer nodes will probably be tested in practice. The following examples illustrate this. It should be noted that in the MILP cases in these examples – like all MILP examples in this chapter – the Cplex MILP solver [45] was used. This is a general-purpose MILP solver, and does not (to the author's knowledge) explicitly make use of the feasibility properties special to this problem.

Example 6.3 *For the system in Example 6.1, the number of feasible combinations of δ values is $f_\delta(2, 12, 2, 1) = 4290$, so an MILP/MIQP search tree could contain $2 \cdot 4290 - 1 = 8579$ nodes. However, as previously mentioned only a few hundred nodes were actually visited. The total number of δ combinations (feasible and infeasible) is $2^{MN} = 2^{24} = 1.7 \cdot 10^7$.*

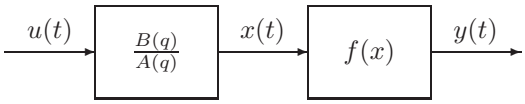


Figure 6.9: Wiener process with piecewise affine static output mapping.

Example 6.4 *A noiseless hinging hyperplane system described by*

$$\begin{aligned} y(t) = {} & 0.6312 + 0.0518y(t-1) - 0.2881u(t-1) + \\ & + \max\{-0.6019 + 0.8529y(t-1) + 0.3759u(t-1), 0\} - \\ & - \max\{0.1238 - 0.5255y(t-1) + 1.8081u(t-1), 0\} \end{aligned} \tag{6.44}$$

was identified from 300 data samples, using a 1-norm criterion function and the Cplex *MILP solver. Although the number of feasible δ combinations equals $f_\delta(2, 300, 2, 1) = 2.0 \cdot 10^9$, the problem was solved after having visited about 1000 nodes, taking about 5 minutes on a SunBlade100 machine. The total number of δ combinations is $2^{600} = 4.1 \cdot 10^{180}$.*

Although the results of the previous examples are encouraging, the systems are rather simple, and the algorithms might not be feasible to use for larger examples. Another problem is that the algorithms seem to slow down when trying to identify systems with noise. However, some systems have a structure which can be exploited to restrict the search space further, and to speed up the algorithm. In Section 6.4, piecewise affine Wiener models are considered.

Another option is to interrupt the algorithm before it is completed, and use the best solution found so far, or use it as an initial value for, e.g., a Gauss-Newton algorithm. This is described in Section 6.5.

6.4 Piecewise Affine Wiener Models

As already mentioned, using MILP/MIQP directly on the problem formulations in Section 6.1 might be too complex to be used in realistic examples. However, there are some model classes where the special model structure allows us to simplify the computations considerably. Let us now turn to the class of *piecewise affine Wiener models*. These models consist of a linear system, followed by a static nonlinearity. The model structure is shown in Figure 6.9, and can be described by the relations

$$\begin{aligned} A(q)x(t) &= B(q)u(t) \\ y(t) &= f(x(t)) \end{aligned} \tag{6.45}$$

where $A(q) = 1 + \sum_{h=1}^{n_a} a_h q^{-h}$, $B(q) = \sum_{k=1}^{n_b} b_k q^{-k}$, and q^{-1} is the delay operator, $q^{-1}x(t) = x(t-1)$. For simplicity of notation, we assume that $n_a \geq n_b$ (this has no effect on the results). We assume that $f(x)$ is a piecewise affine, continuous,

invertible function (without restrictions we can assume that f is strictly increasing), and parametrise its inverse as

$$x(t) = y(t) - \alpha_0 + \sum_{i=1}^{M} \pm \max\{\beta_i y(t) - \alpha_i, 0\} \tag{6.46}$$

Both signs + and − are allowed in order to be able to represent nonconvex functions, and we use the shorthand notation introduced for hinging hyperplane models in (3.10); i.e., (6.46) is equivalent to

$$x(t) = y(t) - \alpha_0 + \sum_{i=1}^{M^+} \max\{\beta_i y(t) - \alpha_i, 0\} - \sum_{i=M^+ +1}^{M} \max\{\beta_i y(t) - \alpha_i, 0\}$$

where M^+ and M are assumed to be known. To avoid overparametrisation, we assume that the coefficient of the linear part in (6.46) equals 1 (this can be compensated for in the $B(q)$ polynomial). As $\max\{-p, 0\} = -p + \max\{p, 0\}$ for all $p \in \mathbb{R}$, without loss of generality we can also assume $\beta_i \geq 0$.

Note that, in general, the piecewise affine Wiener models do not belong to the class of hinging hyperplane models, and therefore we cannot directly use the same kind of problem reformulations as in Section 6.1. To see this, we can rewrite the system using

$$x(t) = (1 - A(q))x(t) + B(q)u(t) \tag{6.47}$$

to get

$$\begin{aligned} y(t) &= f\left(\left[1 - A(q)\right] x(t) + B(q)u(t)\right) = \\ &= f\left(\left[1 - A(q)\right] f^{-1}(y(t)) + B(q)u(t)\right) \end{aligned}$$

From Lemma 2.1 (see Section 2.5, we get an expression for f given (6.46). We can then see that the expression for $y(t)$ contains nested max functions. In fact, the model class is no subset of the class of hinging hyperplane models, but of Güzeliş' model class (see (3.7)).

Like before, we would like to minimise a criterion function as in (6.3), with $\ell = \ell_2$ or $\ell = \ell_1$. We assume that there are known bounds for the values of the unknown parameters a_h, b_k, α_i, and β_i. We also introduce

$$\theta = \left(a_1 \ \ldots \ a_{n_a} \ b_1 \ \ldots \ b_{n_b} \ \alpha_0 \ \ldots \ \alpha_M \ \beta_1 \ \ldots \ \beta_M\right)^T$$

In the case of noisy systems, we assume that the noise enters the system according to the following equation

$$\begin{aligned} A(q)x(t) &= B(q)u(t) + e(t) \\ y(t) &= f(x(t)) \end{aligned} \tag{6.48}$$

where $E[e(t)] = 0$ and $e(t)$ is independent of $\{u_1^{t-1}, x_1^{t-1}, y_1^{t-1}\}$. However, the algorithm may work well also when the noise enters the system differently, as illustrated in Example 6.5.

The problem of identification of Wiener models have been considered previously, e.g., in [17, 38, 96].

6.4.1 Reformulating the Identification Problem

Like in the hinging hyperplane formulation, the first thing to do is to get rid of the max functions. This is done by introducing the discrete variables

$$\delta_i(t) = \begin{cases} 0 & \beta_i y(t) - \alpha_i < 0 \\ 1 & \beta_i y(t) - \alpha_i \geq 0 \end{cases} \quad i = 1, \ldots, M \tag{6.49}$$

One additional problem for this model structure is that we will get products between the coefficients a_h of the $A(q)$ polynomial and the coefficients inside the max functions, β_i and α_i. Furthermore, since a_h may very well be negative, the inequalities in the definition (6.49) of $\delta_i(t)$ may change directions if we multiply by a_h. This is not desirable, so to get rid of these problems, first define $a_h = a_h^+ - a_h^-$, where $a_h^+, a_h^- \geq \gamma$, and $\gamma > 0$ is any positive scalar. Here we let $\gamma = 1$. Then

$$\begin{aligned} a_h \max\{\beta_i y(t-h) - \alpha_i, 0\} = \\ = (a_h^+ - a_h^-) \max\{\beta_i y(t-h) - \alpha_i, 0\} = \\ = \max\{a_h^+ \beta_i y(t-h) - a_h^+ \alpha_i, 0\} - \max\{a_h^- \beta_i y(t-h) - a_h^- \alpha_i, 0\} = \\ = \max\{c_{ih}^+ y(t-h) - d_{ih}^+, 0\} - \max\{c_{ih}^- y(t-h) - d_{ih}^-, 0\} \end{aligned}$$

where

$$\begin{aligned} c_{ih}^{\pm} &\triangleq a_h^{\pm} \beta_i, \quad i = 1, \ldots, M, \ h = 1, \ldots, n_a \\ d_{ih}^{\pm} &\triangleq a_h^{\pm} \alpha_i, \quad i = 1, \ldots, M, \ h = 1, \ldots, n_a \end{aligned}$$

Let also

$$\begin{aligned} c_{i0} = c_{i0}^+ = c_{i0}^- &\triangleq \beta_i, \quad i = 1, \ldots, M \\ d_{i0} = d_{i0}^+ = d_{i0}^- &\triangleq \alpha_i, \quad i = 1, \ldots, M \\ d_{0h} &\triangleq a_h \alpha_0, \quad h = 1, \ldots, n_a \\ d_{00} &\triangleq \alpha_0, \end{aligned}$$

and

$$\bar{d}_0 \triangleq \sum_{h=0}^{n_a} d_{0h} = \left(1 + \sum_{h=1}^{n_a} a_h\right) \alpha_0 \ .$$

As $a_h^+, a_h^- > 0$, from (6.49) it now follows

$$\delta_i(t) = \begin{cases} 0 & a_h^{\pm} \beta_i y(t) - a_h^{\pm} \alpha_i < 0 \\ 1 & a_h^{\pm} \beta_i y(t) - a_h^{\pm} \alpha_i \geq 0 \end{cases} \quad i = 1, \ldots, M, \ h = 1, \ldots, n_a$$

and hence

$$\delta_i(t) = \begin{cases} 0 & c_{ih}^{\pm} y(t) - d_{ih}^{\pm} < 0 \\ 1 & c_{ih}^{\pm} y(t) - d_{ih}^{\pm} \geq 0 \end{cases} \quad i = 1, \ldots, M, \ h = 0, \ldots, n_a \tag{6.50}$$

Introduce the auxiliary continuous variables

$$\begin{aligned} z_{ih}(t) &= \delta_i(t-h)\left((c_{ih}^+ - c_{ih}^-)y(t-h) - (d_{ih}^+ - d_{ih}^-)\right),\ h = 1,\dots,n_a \\ z_{i0}(t) &= \delta_i(t)(c_{i0}y(t) - d_{i0}) \end{aligned} \tag{6.51}$$

for $i = 1,\dots,M$, $t = n_a+1,\dots,N$.

To rewrite the criterion function (6.3), we first try to express $y(t)$ as a linear function of our unknowns. First we note that, by (6.46) and (6.51),

$$\begin{aligned} x(t) &= y(t) - d_{00} + \sum_{i=1}^{M} \pm z_{i0}(t) \\ a_1 x(t-1) &= a_1 y(t-1) - d_{01} + \sum_{i=1}^{M} \pm z_{i1}(t) \\ &\vdots \\ a_{n_a} x(t-n_a) &= a_{n_a} y(t-n_a) - d_{0n_a} + \sum_{i=1}^{M} \pm z_{in_a}(t) \end{aligned} \tag{6.52}$$

Now, using (6.48) and (6.52), we get

$$\begin{aligned} x(t) &= y(t) - d_{00} + \sum_{i=1}^{M} \pm z_{i0}(t) = \\ &= -\sum_{h=1}^{n_a}\left(a_h y(t-h) - d_{0h} + \sum_{i=1}^{M} \pm z_{ih}(t)\right) + \sum_{k=1}^{n_b} b_k u(t-k) + e(t) \end{aligned}$$

which provides the relation

$$y(t) = -\sum_{h=1}^{n_a} a_h y(t-h) + \sum_{k=1}^{n_b} b_k u(t-k) + \bar{d}_0 - \sum_{i=1}^{M}\sum_{h=0}^{n_a} \pm z_{ih}(t) + e(t) \tag{6.53}$$

and therefore

$$\hat{y}(t|\theta) = -\sum_{h=1}^{n_a} a_h y(t-h) + \sum_{k=1}^{n_b} b_k u(t-k) + \bar{d}_0 - \sum_{i=1}^{M}\sum_{h=0}^{n_a} \pm z_{ih}(t) \tag{6.54}$$

This means that the criterion function (6.3) using the ℓ_2 norm can be written as

$$\begin{aligned} V_2(\theta, Z^N) = \sum_{t=n_a+1}^{N} \Bigg(& y(t) + \sum_{h=1}^{n_a} a_h y(t-h) - \sum_{k=1}^{n_b} b_k u(t-k) - \\ & -\bar{d}_0 + \sum_{i=1}^{M}\sum_{h=0}^{n_a} \pm z_{ih}(t) \Bigg)^2 \end{aligned} \tag{6.55}$$

If 1-norm is used, we can rewrite the criterion function similarly and then introduce slack variables, analogously to what was done in (6.7).

The next step is to translate the definitions (6.51) of $z_{ih}(t)$ and (6.50) of $\delta_i(t)$ into linear inequalities. This is done using the method in [7]. The resulting inequalities are

$$\begin{cases} c_{ih}^{\pm} y(t) - d_{ih}^{\pm} \geq L_{ih}^{\pm}(t)(1-\delta_i(t)) \\ c_{ih}^{\pm} y(t) - d_{ih}^{\pm} \leq U_{ih}^{\pm}(t)\delta_i(t) \end{cases} \quad i=1,\dots,M,\ h=0,\dots,n_a,\ t=1,\dots,N$$

$$\begin{cases} z_{i0}(t) \geq L_{i0}(t)\delta_i(t) \\ z_{i0}(t) \leq U_{i0}(t)\delta_i(t) \\ c_{i0}y(t) - d_{i0} \geq z_{i0}(t) + L_{i0}(t)(1-\delta_i(t)) \\ c_{i0}y(t) - d_{i0} \leq z_{i0}(t) + U_{i0}(t)(1-\delta_i(t)) \end{cases} \quad i=1,\dots,M,\ t=n_a+1,\dots,N$$

$$\begin{cases} z_{ih}(t) \geq L_{ih}(t)\delta_i(t-h) \\ z_{ih}(t) \leq U_{ih}(t)\delta_i(t-h) \\ (c_{ih}^{+}-c_{ih}^{-})y(t-h) - (d_{ih}^{+}-d_{ih}^{-}) \geq z_{ih}(t) + L_{ih}(t)(1-\delta_i(t-h)) \\ (c_{ih}^{+}-c_{ih}^{-})y(t-h) - (d_{ih}^{+}-d_{ih}^{-}) \leq z_{ih}(t) + U_{ih}(t)(1-\delta_i(t-h)) \end{cases} \quad h=1,\dots,n_a \tag{6.56}$$

where $U_{ih}^{\pm}(t)$, $L_{ih}^{\pm}(t)$, $U_{ih}(t)$, and $L_{ih}(t)$ are upper and lower bounds on $c_{ih}^{\pm}y(t)-d_{ih}^{\pm}$ and $(c_{ih}^{+}-c_{ih}^{-})y(t-h)-(d_{ih}^{+}-d_{ih}^{-})$, respectively, computed from the known upper and lower bounds on a_h, b_k, α_i, and β_i. Note also that if $c_{ih}^{\pm}y(t)-d_{ih}^{\pm}=0$, $\delta_i(t)$ is allowed to be either 0 or 1. This is analogous to what was discussed in Section 6.1.2, and corresponds to a data sample lying on a breakpoint of the nonlinearity.

Beside the previously mentioned reformulations, that are very similar to the ones in the hinging hyperplane case, we can also introduce some extra constraints that will help the MILP/MIQP solver to discard infeasible solutions, thereby reducing the complexity. Without loss of generality, we can assume that the breakpoints for the positive max functions in the piecewise affine output nonlinearity are ordered, and similarly for the negative max functions. Therefore,

$$\delta_i(t) = 1 \quad \Rightarrow \quad \delta_j(t) = 1 \tag{6.57}$$

should hold for all $i,j \leq M^{+}$ such that $j<i$, and for all $i,j > M^{+}$ such that $j<i$. Each constraint (6.57) is translated into

$$\delta_i(t) - \delta_j(t) \leq 0, \quad j<i \tag{6.58}$$

Now, since

$$\delta_i(t)-\delta_j(t)\leq 0,\ \delta_j(t)-\delta_k(t)\leq 0 \quad \Rightarrow \quad \delta_i(t)-\delta_k(t)\leq 0$$

we only need to include the inequalities (6.58) for pairs of consecutive indices i,j.

Not only the breakpoints, but also the output data $y(t)$ can easily be ordered. This means that we can also get additional relations on $\delta_i(t)$ by using (6.49). In

fact, if $\delta_i(t_0) = 1$ and $y(t_1) \geq y(t_0)$, it must follow that $\delta_i(t_1) = 1$. Like above, we can translate these relations into

$$\delta_i(t_0) - \delta_i(t_1) \leq 0, \quad \forall t_1 \neq t_0 : y(t_1) \geq y(t_0), \quad t_0, t_1 = 1, \ldots, N \tag{6.59}$$

The final MIQP problem that we need to solve is

$$\min \sum_{t=n_a+1}^{N} \left(y(t) + \sum_{h=1}^{n_a} a_h y(t-h) - \sum_{k=1}^{n_b} b_k u(t-k) - \right. \tag{6.60a}$$
$$\left. - \bar{d}_0 + \sum_{i=1}^{M} \sum_{h=0}^{n_a} \pm z_{ih}(t) \right)^2$$
$$\text{subj. to constraints (6.56), (6.58), (6.59)} \tag{6.60b}$$

with respect to the continuous variables $\{a_h, b_k, c_{i0}, d_{i0}, \bar{d}_0, c_{ih}^{\pm}, d_{ih}^{\pm}, z_{ih}(t)\}$ and the binary variables $\{\delta_i(t)\}$; $h = 1, \ldots, n_a$, $i = 0, \ldots, M$, $k = 1, \ldots, n_b$, $t = 1, \ldots, N$. The solution to (6.60) provides the optimal parameters $\hat{a}_h$, $\hat{b}_h$, and

$$\hat{\alpha}_0 = \frac{\hat{\bar{d}}_0}{1 + \sum_{h=1}^{n_a} \hat{a}_h} \tag{6.61}$$
$$\hat{\alpha}_i = \hat{d}_{i0} \tag{6.62}$$
$$\hat{\beta}_i = \hat{c}_{i0} \tag{6.63}$$

Finally, since the nonlinearity $f(x)$ is invertible, we can obtain the estimate $\hat{f}(x)$ by using the result of Lemma 2.1.

Example 6.5 *The following example is taken from [8]. A Wiener model, consisting of a first-order linear system and a nonlinearity with two breakpoints, is identified using $N = 20$ estimation data points. The system is first identified using noiseless data, and then using noisy measurements $\tilde{y}(t) = y(t) + e(t)$, where $e(t)$ are independent random variables, uniformly distributed on a symmetric interval around zero. The resulting estimated parameters are shown in Table 6.3.*

Apparently, the estimated parameters are overall very close to the true values, the closer the lower the intensity of the output noise, as should be expected.

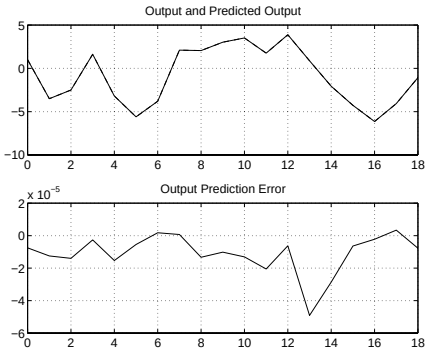
The estimated model was also tested on a set of validation data, and in Figure 6.10 the resulting one-step-ahead predicted output and output error are plotted. Note that such a good performance cannot be achieved by using standard linear identification techniques.

6.4.2 Complexity Analysis

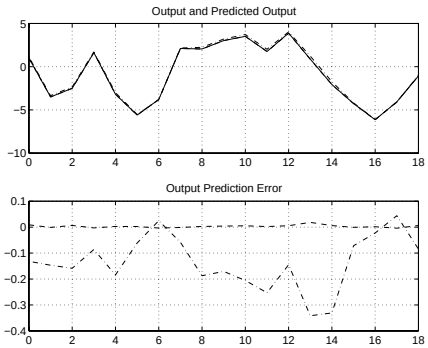
The complexity of the problem is tightly connected to the number of discrete variables, which equals MN. However, by imposing the constraints expressed by

Parameter	True value	Noiseless data	$\lvert e(t)\rvert < 0.01$	$\lvert e(t)\rvert < 0.1$
a_1	-0.5	-0.5000	-0.4990	-0.5360
b_1	2	2.0000	2.0024	2.0003
α_0	-2	-2.0000	-2.0001	-1.7748
α_1	0.5	0.5000	0.5095	0.5509
α_2	-1.5	-1.5000	-1.4924	-1.4999
β_1	0.5	0.5000	0.5016	0.5028
β_2	0.5	0.5000	0.4988	0.4876

Table 6.3: Estimation results.



(a) System estimated with noiseless data.



(b) System estimated with output noise $|e(t)| \leq 0.01$ (dashed line), and $|e(t)| \leq 0.1$ (dot-dashed line).

Figure 6.10: Validation results.

(6.58) and (6.59), the degrees of freedom for the integer variables are reduced considerably. If we would only have positive (or only negative) max functions, instead of having to test 2^{MN} different cases (the number of feasible and infeasible combinations of δ values) in the worst case, only $\binom{M+N}{M}$ combinations would have to be tested. Furthermore, of these combinations only $\binom{N-1}{M}$ are nontrivial (so that every affine region of the nonlinearity contains at least one data sample). If we have M^+ positive and $M^- = M - M^+$ negative max functions, this number must be multiplied by a factor $\binom{M}{M^+}$ to account for the fact that the positive and negative max functions could be ordered in different ways. For example, for $N = 20$ and $M = 2$ this means that the number of possible combinations of integer variables decreases from approximately 10^{12} to 231 (in the case of only positive max functions) or 462 (for one positive and one negative max function). Of these, 171 combinations (or 342, respectively) are nontrivial. In general, for a fixed M the worst-case complexity grows as N^M. Note that this simplification is possible since the nonlinearity is one-dimensional, which allows an ordering of the breakpoints

and of the output data. In fact, the number of nontrivial combinations equals $f_\delta(1, N, M, M^+)$, calculated in Corollary 6.1.

6.5 Using Suboptimal MILP/MIQP solutions

One advantage with the MILP/MIQP algorithms is that they give intermediate results, which are suboptimal, but get better and better the longer the algorithm runs. This feature could be used for complex MILP/MIQP problems, where the time to find the optimal solution (and then prove that it really is optimal) might be extremely long. Instead of waiting for the optimal solution, one can abort the computations and use the best solution found so far.

In the case of piecewise affine system identification, one possibility is to use the suboptimal solutions obtained from the MILP/MIQP solver as initial values for a standard local minimisation algorithm, e.g., Gauss-Newton (see Section 2.2.2). Finding good initial values for minimisation algorithms is an important issue, and hopefully, using the intermediate solutions from MILP/MIQP as initial values gives better results than just using randomly picked initial values. Experiments show that this is often the case. However, it might sometimes take a little while until the MILP/MIQP solver finds suboptimal solutions that are worth using. The following examples illustrate this.

Example 6.6 *Consider once again the system in Example 6.1 and the 2-norm criterion function. The intermediate results from a simple, straightforward* MATLAB *implementation of an MIQP solver are shown in Table 6.4. The parameters of these suboptimal solutions were then used as initial values of the* MATLAB *optimisation functions* `fminsearch` *and* `lsqnonlin`*, and the resulting criterion function values are also found in Table 6.4.*

As comparison, a number of random values were given as initial parameters to `fminsearch` *and* `lsqnonlin`*. In Figure 6.11, the values obtained by using the intermediate results from MIQP as initial values are compared to the mean result of using random initial values. We can see that already after about 250 visited nodes, the combined MIQP/*`fminsearch` *and MIQP/*`lsqnonlin` *algorithms perform better than using only* `fminsearch` *and* `lsqnonlin`*, respectively.*

Example 6.7 *Consider the system in Example 6.4, but with white noise added to $y(t)$ in* (6.44)*. The variance of the noise is 0.1, while the input is white noise with unit variance. The system was identified using 100 such data samples and the 1-norm criterion function. The problem was reformulated according to* (6.12) *and given to the* CPLEX *MILP solver. The intermediate solutions were used as initial values to the* MATLAB *function* `fminsearch`*. Just as in Example 6.6, a number of random initial values to* `fminsearch` *were also used as comparison. Table 6.5 shows the resulting values of the criterion function. In Figure 6.12, the values of the criterion function are shown as a function of the approximative computation*

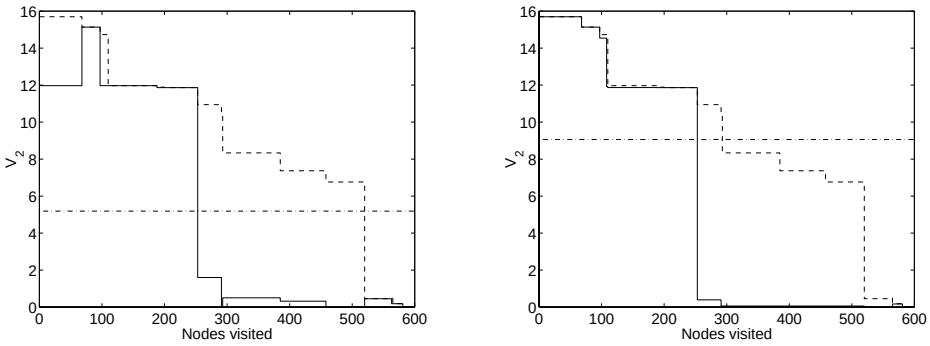
Node	MIQP	`fminsearch`	`lsqnonlin`
23	15.6967	11.9709	15.6967
68	15.1356	15.1356	15.1356
97	14.7320	11.9709	14.5412
100	14.7320	11.9709	14.5412
108	14.5007	11.9709	11.9132
110	11.9711	11.9709	11.8683
144	11.9711	11.9709	11.8683
188	11.8921	11.8629	11.8640
200	11.8632	11.8631	11.8632
253	10.9457	1.5977	0.3908
291	10.4377	0.0145	0.0564
293	8.3370	0.5000	0.0564
300	8.3370	0.5000	0.0564
385	7.3687	0.3188	0.0564
391	7.3687	0.3188	0.0564
458	6.7620	0.0000	0.0564
520	0.4540	0.4496	0.0000
523	0.4540	0.4496	0.0000
560	0.4514	0.4496	0.0000
563	0.4796	0.1817	0.0000
565	0.1834	0.1817	0.1669
581	0.1122	0.0000	0.0000
583	0.0044	0.0000	0.0000

Table 6.4: Intermediate results for the system in Examples 6.1 and 6.6. The first column shows the number of the node where the result was found (e.g., the first result was found in the 23rd node visited). The second column contains the values of the criterion function V_2 for the different solutions obtained by the MIQP solver. The third and fourth columns contain the values of V_2 after the solutions have been improved by running `fminsearch` and `lsqnonlin`, respectively.

time (on a SunBlade 100). Here, the combined MILP/`fminsearch` performs better than using random initial values from the very first intermediate result produced by the MILP solver.

Example 6.8 *The noiseless system*

$$\begin{aligned} y(t) = {} & 1.1515 - 0.5557y(t-1) - 0.3370y(t-2) + 1.3795u(t-1) + \\ & + \max\{-0.4898 + 0.0672y(t-1) + 1.9245y(t-2) - 0.3428u(t-1), 0\} + \\ & + \max\{-0.0336 + 0.4396y(t-1) + 0.1561y(t-2) - 1.3756u(t-1), 0\} - \\ & - \max\{-0.3871 - 1.6273y(t-1) + 0.3843y(t-2) + 1.6882u(t-1), 0\} - \\ & - \max\{1.0491 - 0.0629y(t-1) + 0.9997y(t-2) - 1.3911u(t-1), 0\} \end{aligned}$$



(a) Using `fminsearch`.

(b) Using `lsqnonlin`.

Figure 6.11: The intermediate results of Example 6.6 plotted against node number. The dashed lines show the V_2 values of the results from MIQP, the solid lines show the V_2 values after improving the solutions with `fminsearch` (a) and `lsqnonlin` (b), and the dash-dotted lines show the mean values of V_2 after using `fminsearch` (a) and `lsqnonlin` (b) on random initial values.

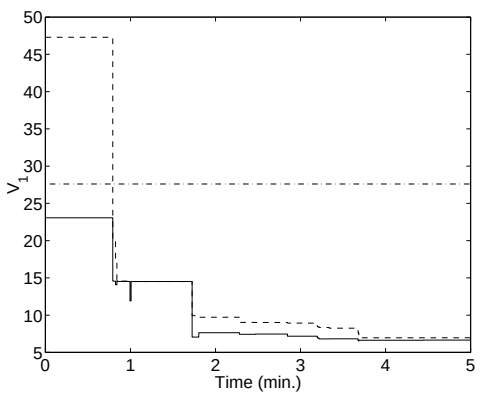


Figure 6.12: The criterion function values of the different solutions in Table 6.5 (Example 6.7) as a function of the computation time: The criterion function values after using only MILP (dashed line), after improving the results by using `fminsearch` (solid line), and the mean result from using `fminsearch` with random initial values (dash-dotted line).

	Node	Time	MILP	`fminsearch`
1	0	0.1	47.2966	23.0627
2	2310	0.8	19.9396	14.5632
3	2439	0.8	16.9958	14.0697
4	2500	0.8	14.7300	14.6152
5	2526	0.8	14.5641	14.5168
6	3100	1.0	14.5217	11.9073
7	3150	1.0	14.5137	14.5137
8	5940	1.7	9.9354	7.0514
9	6230	1.8	9.7185	7.6424
10	8010	2.3	9.0228	7.4318
11	8750	2.5	9.0206	7.4650
12	10220	2.8	8.9276	7.1749
13	11640	3.2	8.4372	6.9383
14	11717	3.2	8.3458	6.8154
15	12190	3.3	8.2480	6.8169
16	13590	3.7	7.6014	6.5391
17	13628	3.7	6.9740	6.6377
18	13700	3.7	6.9599	6.6272
19	16840	4.4	6.9572	6.6519

Table 6.5: Identification of the system in Example 6.7. Results of using intermediate solutions from an MILP algorithm as input to `fminsearch`: The number of visited nodes, the computation time (in minutes, using a SunBlade 100 machine), and the values of V_2 for the solutions before and after using `fminsearch`.

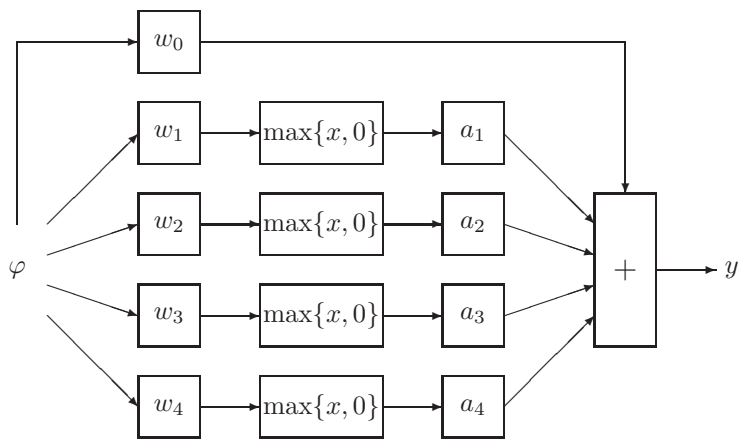


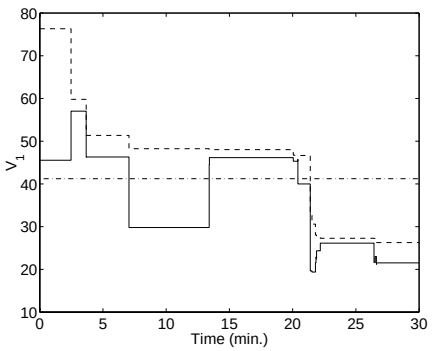
Figure 6.13: The structure of the neural network used in Example 6.8.

	Node	Time	MILP	`fminsearch`	NN
1	0	0.1	76.3178	45.5419	19.3366
2	2270	2.5	59.7858	57.0291	11.1362
3	3510	3.7	52.9559	46.2431	19.0352
4	3520	3.7	51.3513	46.2964	10.6999
5	7040	7.1	48.2320	29.7991	18.8824
6	13780	13.4	48.0376	44.5297	18.6923
7	13796	13.4	48.0274	46.1497	10.6867
8	21690	20.1	46.7244	45.3083	19.2758
9	22110	20.4	46.6540	45.7513	19.2948
10	22120	20.4	46.6491	39.9945	18.6721
11	23330	21.4	32.6644	19.6507	0.1239
12	23490	21.5	30.5823	19.3618	0.2066
13	23820	21.8	28.0669	21.5232	0.6236
14	23880	21.8	28.0649	22.8920	16.8954
15	23910	21.9	27.8096	22.4603	0.2265
16	24930	21.9	27.7681	24.3475	14.7952
17	24390	22.2	27.2844	26.1267	0.3424
18	29930	26.4	27.2509	21.6040	0.5424
19	30060	26.5	27.0350	23.0246	11.4028
20	30210	26.6	26.3121	21.1705	19.0251
21	30230	26.6	26.2591	21.5190	0.5986

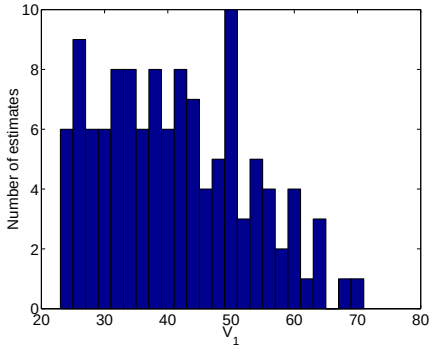
Table 6.6: Identification of the system in Example 6.8. Results of using intermediate solutions of an MILP algorithm as input to `fminsearch` and the neural network in Figure 6.13: The number of visited nodes, the computation time (in minutes, using a SunBlade 100 machine), the values of V_2 for the MILP solutions, and the values of V_2 after improving the MILP solutions using `fminsearch` and the neural network (NN), respectively.

was identified using 100 data samples and the 1-norm criterion function. The Cplex *MILP solver was used to solve the MILP problem obtained, and the intermediate solutions were used as initial values to the* Matlab *function* `fminsearch`*. They were also used as initial values to a neural network with the structure shown in Figure 6.13. The network was then trained using a backpropagation algorithm with variable learning rate (*`traingdx` *in the Neural Network Toolbox in* Matlab*).*

Table 6.6 shows the resulting values of the criterion function. The computation time shown in the table (and in subsequent figures) is the time for finding the intermediate solution with the MILP solver (on a SunBlade 100 machine). Running `fminsearch` *once took about 15 seconds on average, while the time for training the neural network was around 3-25 minutes. The results from the combined MILP/*`fminsearch` *solver are visualised in Figure 6.14(a), where the values of the criterion function are shown as a function of the computation time, and are compared to the mean result using random initial values. After about 20 minutes,*



(a) The criterion function values after using only MILP (dashed line), after improving the results by using `fminsearch` (solid line), and the mean result from using `fminsearch` with random initial values (dash-dotted line).



(b) The distribution of results using `fminsearch` with random initial values for 30 minutes.

Figure 6.14: (a) The criterion function values of the different solutions in Table 6.6 (Example 6.8) as a function of the computation time (using `fminsearch`), and (b) the results from using `fminsearch` during 30 minutes.

the combined solver works better than just using `fminsearch` *with random initial values. An interesting thing to note here is that the* 11*th intermediate result – where the combined solver starts working better – is the first solution where both positive and negative hinge functions are used. Figure 6.14(b) shows the distribution of results when* `fminsearch` *was run repeatedly during 30 minutes with random initial values. As we can see, the combined MILP/*`fminsearch` *solver performed better (the best result when using* `fminsearch` *with random initial values was* 23.3*, while the best result from the combined solver was* 19.4*).*

The results from the neural network are shown in Figure 6.15, also here with the criterion function value as a function of the computation time. We can see that also in this case, using the intermediate results from the MILP solver as initial values for the network training mostly gives better results than the average result for using random initial values.

6.6 Related Approaches

The formulation of the piecewise affine system identification problem using discrete variables $\delta_i(t)$ can also be used as a basis for other algorithms than those described in this chapter. To highlight the relation between the MILP/MIQP approach and other, previously proposed algorithms, two Newton-like methods using $\delta_i(t)$ will be

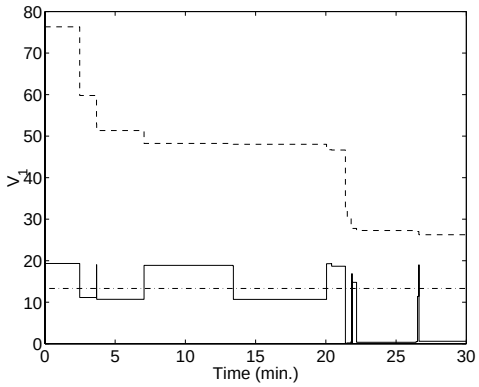


Figure 6.15: The criterion function values of the different solutions in Table 6.6 (Example 6.8) as a function of the computation time (using the neural network in Figure 6.13): The criterion function values after using only MILP (dashed line), after improving the results by using a neural network (solid line), and the mean result from training the neural network with random initial values (dash-dotted line).

discussed here. The second algorithm has, to the author's knowledge, not appeared previously in the literature.

Consider once more the predicted output (6.2), repeated here for convenience:

$$\hat{y}(t|\theta) = \varphi^T(t)\theta_0 + \sum_{i=1}^{M} \pm \max\{\varphi^T(t)\theta_i, 0\} \tag{6.64}$$

Using the definition (6.8) of $\delta_i(t)$, we can rewrite (6.64) as

$$\begin{aligned} \hat{y}(t|\theta) &= \varphi^T(t)\theta_0 + \sum_{i=1}^{M} \pm\delta_i(t)\varphi^T(t)\theta_i = \\ &= \underbrace{\begin{pmatrix}\varphi^T(t) & \pm\delta_1(t)\varphi^T(t) & \dots & \pm\delta_M(t)\varphi^T(t)\end{pmatrix}}_{\Phi^T(t,\delta(t))} \underbrace{\begin{pmatrix}\theta_0 \\ \theta_1 \\ \vdots \\ \theta_M\end{pmatrix}}_{\theta} \end{aligned} \tag{6.65}$$

where we also introduce $\delta(t) = \begin{pmatrix}\delta_1(t) & \dots & \delta_M(t)\end{pmatrix}$ for notational convenience. The criterion function V_2 from (6.5) can now be written

$$V_2 = \sum_{t=1}^{N} (y(t) - \Phi^T(t, \delta(t))\theta)^2 \tag{6.66}$$

This form can be used as the starting point for several different identification algorithms not using MILP/MIQP, often related to the approaches presented in Chapter 5. Here, some of the possible approaches will be discussed.

For a given set of $\delta(t)$, (6.66) would be minimised by

$$\hat{\theta} = \left(\sum_{t=1}^{N} \Phi(t, \delta(t)) \Phi^T(t, \delta(t)) \right)^{-1} \sum_{t=1}^{N} \Phi(t, \delta(t)) y(t)$$

However, the fact that $\delta(t)$ is determined by θ and $\varphi(t)$ has been neglected here. What one could do is to compute the value of $\delta(t)$ corresponding to $\hat{\theta}$. Let us call this value $\hat{\delta}^{(0)}(t)$. Now we can iterate, letting

$$\hat{\theta}^{(k)} = \left(\sum_{t=1}^{N} \Phi(t, \hat{\delta}^{(k-1)}(t)) \Phi^T(t, \hat{\delta}^{(k-1)}(t)) \right)^{-1} \sum_{t=1}^{N} \Phi(t, \hat{\delta}^{(k-1)}(t)) y(t) \tag{6.67a}$$

and

$$\hat{\delta}_i^{(k)}(t) = \begin{cases} 0 & \varphi^T(t) \hat{\theta}_i^{(k)} < 0 \\ 1 & \varphi^T(t) \hat{\theta}_i^{(k)} \geq 0 \end{cases} \tag{6.67b}$$

When, for some k, $\hat{\theta}^{(k+1)} = \hat{\theta}^{(k)}$, we have reached a local minimum.

This algorithm is a variant of the hinge-finding algorithm due to Breiman, described in [14]. The difference between them is that this version considers all hinge functions simultaneously, instead of fitting one hinge function at a time. The last property makes it belong to the first category of approaches listed in Chapter 5. Like the hinge-finding algorithm, there is no guarantee for convergence, but the algorithm can be modified to guarantee convergence. In this case it becomes the same algorithm as in [79].

Another option, inspired by one of the algorithms proposed in [44], is to solve the convex QP

$$\begin{aligned} \min_{\theta} \quad & V_2 = \sum_{t=1}^{N} (y(t) - \Phi^T(t, \delta(t))\theta)^2 \\ \text{subj. to} \quad & \begin{cases} \varphi^T(t)\theta_i \leq 0 & \text{if } \delta_i(t) = 0 \\ \varphi^T(t)\theta_i \geq 0 & \text{if } \delta_i(t) = 1 \end{cases} \end{aligned} \tag{6.68}$$

for given $\delta_i(t)$. The algorithm is as follows:

Algorithm 6.1

1. *Assign initial values to $\delta_i(t)$ and solve the corresponding QP* (6.68).
2. *There are two possibilities for the resulting $\hat{\theta}$:*

- *If $\hat{\theta}$ does not satisfy $\varphi^T(t)\hat{\theta}_i = 0$ for any $t = 1, \ldots, N$, $i = 1, \ldots, M$, this means that $\hat{\theta}$ lies in the interior of the feasible region of* (6.68)*. In other words, no data sample $\varphi(t)$ lies on a hinge. This means that a local minimum for the original problem* (6.66) *is found, and we stop.*
- *Otherwise, we collect all pairs (t, i) such that $\varphi^T(t)\hat{\theta}_i = 0$, and change the values of the corresponding $\delta_i(t)$ (or a subset of them). This is the same as assigning some of the data points $\varphi(t)$ lying on a hinge to the region on the other side of the hinge. Then go to 3.*

3. *Solve* (6.68) *for the new set of $\delta_i(t)$ values. If $\varphi^T(t)\hat{\theta}_i = 0$ for the same pairs (t, i) as before, go to 4. Otherwise, go to 2.*
4. *If not all possible assignments of values to $\delta_i(t)$ for the pairs (t, i) with $\varphi^T(t)\hat{\theta}_i = 0$ are tested, choose a new assignment and go to 3. Otherwise, we have found a local minimum, and stop.*

Note that the value of V_2 will never increase, since the optimal solution of the previous problem is a feasible solution to the new problem as well. This means that the algorithm will converge.

This algorithm is almost a damped Newton algorithm, as explained in Section 5.1.2 for the corresponding algorithm in [44]. The differences compared to the algorithm in [44] are that hinging hyperplanes are used instead of hinging sigmoids, and that all parameters are updated simultaneously.

6.7 Conclusions

In this chapter, an approach to piecewise affine system identification using mixed-integer programming has been considered. The theoretical advantage of this approach, compared to, e.g., using standard local minimisation tools, is that we are guaranteed to find the global optimum in a finite number of steps. Furthermore, an upper bound on the number of steps can be given. In practice, however, the computational complexity is too large to make the approach applicable to medium-sized or large identification problems of general piecewise affine systems.

In spite of this, using MILP/MIQP for identification might be a feasible option in some cases. As described in Section 6.4, some model structures, like the piecewise affine Wiener models, allow a significant reduction of the complexity, due to the fact that the nonlinearity is one-dimensional. Another, potentially attractive, way of utilising the MILP/MIQP formulation was described in Section 6.5, where intermediate solutions from the MILP/MIQP solver were used as initial values for a local minimisation algorithm. One problem with the MILP/MIQP formulations is that the computational complexity increases quite rapidly with the number of experimental data. However, if we are only interested in obtaining good initial values for another algorithm, we might very well choose a subset of the experimental dataset, consisting of, say, a few hundred data samples, that are well distributed over the state-space, and use these for the MILP/MIQP algorithm.

Finally, the relation between the mixed-integer programming approach and some Newton-like methods was pointed out, and an extension of an algorithm proposed in [44] was given.

There are still issues that need to be investigated. One interesting topic is what conditions can be given to ensure that the experimental data is persistently exciting, given different model structures. For continuous-time piecewise affine systems in Chua's canonical form, some work has been done on this [49]. The MILP/MIQP solving strategies could probably also be specialised to exploit the structure of this kind of problems. More experiments could be done to compare the performance of different algorithms. In particular, the algorithms of Section 6.5 should only be seen as a first step in exploring the approach of using intermediate results from an MILP/MIQP solver. These algorithms could probably be developed substantially.

Bibliography

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[2] R. Alur, T. A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22:181–201, 1996.

[3] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, July 2000.

[4] R. Batruni. A multilayer neural network with piecewise-linear structure and back-propagation learning. *IEEE Transactions on Neural Networks*, 2(3):395–403, May 1991.

[5] A. Bemporad, G. Ferrari-Trecate, and M. Morari. Observability and controllability of piecewise affine and hybrid systems. *IEEE Transactions on Automatic Control*, 45(10):1864–1876, Oct. 2000.

[6] A. Bemporad and D. Mignone. *MIQP.M: A Matlab function for solving mixed integer quadratic programs*. ETH Zurich, 2000. Code available at `http://control.ethz.ch/~hybrid/miqp`.

[7] A. Bemporad and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35:407–427, 1999.

[8] A. Bemporad, J. Roll, and L. Ljung. Identification of hybrid systems via mixed-integer programming. In *The 40th IEEE Conference on Decision and Control*, 2001. To appear.

[9] A. Bemporad, F. D. Torrisi, and M. Morari. Optimization-based verification and stability characterization of piecewise affine and hybrid systems. In B. Krogh and N. Lynch, editors, *Hybrid Systems: Computation and Control*, volume 1790 of *Lecture Notes in Computer Science*, pages 45–58. Springer Verlag, 2000.

[10] C. M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.

[11] S. Boyd and L. Vandenberghe. Convex optimization. Course Reader for EE364, Introduction to Convex Optimization with Engineering Applications, Stanford University, May 3, 1999.

[12] M. S. Branicky. Multiple Lyapunov functions and other analysis tools for switched and hybrid systems. *IEEE Transactions on Automatic Control*, 43 (4):475–482, Apr. 1998.

[13] M. S. Branicky, V. S. Borkar, and S. K. Mitter. A unified framework for hybrid control: model and optimal control theory. *IEEE Transactions on Automatic Control*, 43(1):31–45, 1998.

[14] L. Breiman. Hinging hyperplanes for regression, classification, and function approximation. *IEEE Transactions on Information Theory*, 39(3):999–1013, May 1993.

[15] L. Breiman and J. H. Friedman. Function approximation using ramps. In *Snowbird Workshop. Machines that Learn*, 1994.

[16] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification And Regression Trees*. Wadsworth & Brooks/Cole Advanced Books & Software, 1984.

[17] J. Bruls, C. T. Chou, B. R. J. Haverkamp, and M. Verhaegen. Linear and nonlinear system identification using separable least-squares. *European Journal of Control*, 5(1):116–128, 1999.

[18] C.-H. Choi and J. Y. Choi. Constructive neural networks with piecewise interpolation capabilities for function approximations. *IEEE Transactions on Neural Networks*, 5(6):936–944, Nov. 1994.

[19] L. O. Chua and A.-C. Deng. Canonical piecewise-linear representation. *IEEE Transactions on Circuits and Systems*, 35(1):101–111, Jan. 1988.

[20] L. O. Chua and S. M. Kang. Section-wise piecewise-linear functions: Canonical representation, properties and applications. *Proceedings of the IEEE*, 65:915–929, 1977.

[21] A. Chutinan. *Hybrid System Verification Using Discrete Model Approximations*. PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, May 1999.

[22] A. Chutinan and B. H. Krogh. Computing approximating automata for a class of linear hybrid systems. In *Hybrid Systems V, Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[23] A. Chutinan and B. H. Krogh. Computing polyhedral approximations to flow pipes for dynamic systems. In *The 37th IEEE Conference on Decision and Control: Session on Synthesis and Verification of Hybrid Control Laws (TM-01)*, 1998.

[24] T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, 14(3):326–334, June 1965.

[25] Dash Associates. *XPRESS-MP User Guide*, 1999.

[26] L. Davis. *Genetic Algorithms and Simulated Annealing*. London: Pitman; Los Altos, Calif.: Kaufmann, 1987.

[27] R. A. DeCarlo, M. S. Branicky, S. Pettersson, and B. Lennartsson. Perspectives and results on the stability and stabilizability of hybrid systems. *Proceedings of the IEEE*, 88(7):1069–1082, July 2000.

[28] D. den Hertog. *Interior Point Approach to Linear, Quadratic and Convex Programming: Algorithms and Complexity*. Kluwer Academic, 1994.

[29] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973.

[30] V. Einarsson. On verification of switched systems using abstractions. Licentiate Thesis, Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden. Thesis No. 705, 1998.

[31] V. Einarsson. *Model Checking Methods for Mode Switching Systems*. PhD thesis, Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden, 2000.

[32] S. Engell, S. Kowalewski, C. Schulz, and O. Stursberg. Continuous-discrete interactions in chemical processing plants. *Proceedings of the IEEE*, 88(7): 1050–1068, July 2000.

[33] S. Ernst. Hinging hyperplane trees for approximation and identification. In *The 37th IEEE Conference on Decision and Control*, volume 2, pages 1266–1271, 1998.

[34] G. Ferrari-Trecate, M. Muselli, D. Liberati, and M. Morari. A clustering technique for the identification of piecewise affine systems. In *Hybrid Systems: Computation and Control. Fourth International Workshop*, 2001.

[35] C. A. Floudas. *Nonlinear and Mixed-Integer Optimization: Fundamentals and Applications*. Oxford University Press, 1995.

[36] E. F. Gad, A. F. Atiya, S. Shaheen, and A. El-Dessouki. A new algorithm for learning in piecewise-linear neural networks. *Neural Networks*, 13(4-5): 485–505, 2000.

[37] C. Güzeliş and İ. C. Göknar. A canonical representation for piecewise-affine maps and its applications to circuit analysis. *IEEE Transactions on Circuits and Systems*, 38(11):1342–1354, Nov. 1991.

[38] A. Hagenblad. Aspects of the identification of Wiener models. Licentiate Thesis, Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden. Thesis No. 793, 1999.

[39] W. P. M. H. Heemels, B. D. Schutter, and A. Bemporad. Equivalence of hybrid dynamical models. *Automatica*, 2001. To appear.

[40] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: a model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.

[41] E. A. Heredia and G. R. Arce. Piecewise linear system modeling based on a continuous threshold decomposition. *IEEE Transactions on Signal Processing*, 44(6):1440–1453, June 1996.

[42] I. Hoffmann. *Identifikation hybrider dynamischer Systeme*. PhD thesis, Universität Dortmund, 1999.

[43] I. Hoffmann and S. Engell. Identification of hybrid systems. In *American Control Conference*, pages 711–712, 1998.

[44] D. R. Hush and B. Horne. Efficient algorithms for function approximation with piecewise linear sigmoidal networks. *IEEE Transactions on Neural Networks*, 9(6):1129–1141, Nov. 1998.

[45] ILOG, Inc. *CPLEX 7.0 User's Manual*. Gentilly, France, 2000.

[46] T. A. Johansen and B. A. Foss. Identification of non-linear system structure and parameters using regime decomposition. *Automatica*, 31(2):321–326, Feb. 1995.

[47] M. Johansson. *Piecewise Linear Control Systems*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, Lund University, Box 118, SE-221 00 Lund, Sweden, 1999.

[48] M. Johansson and A. Rantzer. Computation of piece-wise quadratic Lyapunov functions for hybrid systems. *IEEE Transactions on Automatic Control*, 43 (4):555–559, 1998.

[49] M. A. Jordán and O. A. A. Orqueda. Persistency of excitation and richness in continuous-time piecewise linear systems. In *1st International Conference on Control of Oscillations and Chaos*, volume 2, pages 234–237, Aug. 1997.

[50] P. Julián, A. Desages, and O. Agamennoni. High level canonical piecewise linear representation using a simplicial partition. *IEEE Transactions on Circuits and Systems*, 46:463–480, 1999.

[51] P. Julián, M. Jordán, and A. Desages. Canonical piecewise-linear approximation of smooth functions. *IEEE Transactions on Circuits and Systems*, 45(5): 567–571, May 1998.

[52] P. M. Julián. *A High Level Canonical Piecewise Linear Representation: Theory and Applications*. PhD thesis, Departamento de Ingeniería Eléctrica, Universidad Nacional del Sur, 1999.

[53] C. Kahlert and L. O. Chua. The complete canonical piecewise-linear representation — part i: The geometry of the domain space. *IEEE Transactions on Circuits and Systems*, 39:222–236, 1992.

[54] A. M. Kang and L. O. Chua. A global representation of multidimensional piecewise-linear functions. *IEEE Transactions on Circuits and Systems*, CAS-25:938–940, Nov. 1978.

[55] T. Koskela, M. Varsta, J. Heikkonen, and K. Kaski. Recurrent SOM with local linear models in time series prediction. In *6th European Symposium on Artificial Neural Networks*, pages 167–172, Apr. 1998.

[56] X. D. Koutsoukos, P. J. Antsaklis, J. A. Stiver, and M. D. Lemmon. Supervisory control of hybrid systems. *Proceedings of the IEEE*, 88(7):1026–1049, July 2000.

[57] S. Kowalewski, O. Stursberg, M. Fritz, H. Graf, I. Hoffmann, J. Preußig, M. Remelhe, S. Simon, and H. Treseler. A case study in tool-aided analysis of discretely controlled continuous systems: the two tanks problem. In *Hybrid Systems V*, volume 1567 of *Lecture Notes in Computer Science*, pages 163–185. Springer-Verlag, 1999.

[58] C.-A. Lehalle and R. Azencott. Piecewise affine neural networks and nonlinear control. In *International Conference on Artificial Neural Networks*, volume 2, pages 633–638, Sept. 1998.

[59] C.-A. Lehalle and R. Azencott. Piecewise affine neural networks and nonlinear control: Stability results. In *International Conference on Artificial Neural Networks*, pages 608–612, Sept. 1999.

[60] J.-N. Lin and R. Unbehauen. Canonical piecewise-linear approximations. *IEEE Transactions on Circuits and Systems – I: Fundamental Theory and Applications*, 39(8):697–699, Aug. 1992.

[61] J.-N. Lin, H.-Q. Xu, and R. Unbehauen. A generalization of canonical piecewise-linear functions. *IEEE Transactions on Circuits and Systems – I: Fundamental Theory and Applications*, 41(4):345–347, Apr. 1994.

[62] C. Livadas, J. Lygeros, and N. A. Lynch. High-level modeling and analysis of the traffic alert and collision avoidance system (TCAS). *Proceedings of the IEEE*, 88(7):926–948, July 2000.

[63] L. Ljung. *System Identification: Theory for the User*. Prentice Hall, 2nd edition, 1999.

[64] J. Lygeros, C. Tomlin, and S. Sastry. Controllers for reachability specifications for hybrid systems. *Automatica*, 35(3):349–370, 1999.

[65] T. M. Martinetz, S. G. Berkovich, and K. J. Schulten. "Neural-gas" network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, 4(4):558–569, July 1993.

[66] M. C. Medeiros, M. G. C. Resende, and A. Veiga. Piecewise linear time series estimation with GRASP. Technical report, AT&T Labs Research, Florham Park, NJ 07932 USA, 1999.

[67] M. C. Medeiros, A. Veiga, and M. G. C. Resende. A combinatorial approach to piecewise linear time series analysis. Technical report, AT&T Labs Research, Florham Park, NJ 07932 USA, 1999.

[68] D. Mignone, A. Bemporad, and M. Morari. A framework for control, fault detection, state estimation, and verification of hybrid systems. In *American Control Conference*, pages 134–138, June 1999.

[69] T. Moor and J. Raisch. Discrete control of switched linear systems. In *European Control Conference*, Aug.–Sept. 1999.

[70] R. Murray-Smith. Local model networks and local learning. In *Fuzzy Duisburg*, pages 404–409, Feb. 1994.

[71] R. Murray-Smith and H. Gollee. A constructive learning algorithm for local model networks. In *IEEE Workshop on Computer-intensive methods in control and signal processing*, pages 21–29, 1994.

[72] R. Murray-Smith and T. A. Johansen, editors. *Multiple Model Approaches to Modelling and Control*. Taylor & Francis, 1997.

[73] S. Nadjm-Tehrani and J.-E. Strömberg. Formal verification of dynamic properties in an aerospace application. *Formal Methods in System Design*, 14(2): 135–169, Mar. 1999.

[74] D. L. Pepyne and C. G. Cassandras. Optimal control of hybrid systems in manufacturing. *Proceedings of the IEEE*, 88(7):1108–1123, July 2000.

[75] V. Petridis and A. Kehagias. Identification of switching dynamical systems using multiple models. In *The 37th IEEE Conference on Decision and Control: Session on System Identification I (WA-07)*, 1998.

[76] S. Pettersson. *Analysis and Design of Hybrid Systems*. PhD thesis, Control Engineering Laboratory, Department of Signals and Systems, Chalmers University of Technology, SE-412 96 Göteborg, Sweden, 1999.

[77] P. Philips, M. Weiss, and H. A. Preisig. Control based on discrete-event models of continuous systems. In *European Control Conference*, Aug.–Sept. 1999.

[78] C. Pizzi and P. Pellizzari. Adaptive local linear models for financial time series. *Journal of Computational Intelligence in Finance*, Jan. 1998.

[79] P. Pucar and J. Sjöberg. On the hinge-finding algorithm for hinging hyperplanes. *IEEE Transactions on Information Theory*, 44(3):1310–1319, May 1998.

[80] J. Roll. Invariance of approximating automata for piecewise linear systems. Technical Report LiTH-ISY-R-2178, Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden, 1999.

[81] J. Roll. Invariance of approximating automata for piecewise linear systems with uncertainties. In *Hybrid Systems: Computation and Control. Third International Workshop*, Lecture Notes in Computer Science 1790, pages 396–406. Springer-Verlag, 2000.

[82] J. Roll. Robust verification of piecewise affine systems. Submitted to 15th IFAC World Congress on Automatic Control, 2002.

[83] M. Rubensson. Discrete-time stability analysis of hybrid systems. Licentiate Thesis, Department of Signals and Systems, Chalmers University of Technology, Sweden, 2000.

[84] N. V. Sahinidis. BARON — Branch And Reduce Optimization Navigator. Technical report, University of Illinois at Urbana-Champaign, Dept. of Chemical Engineering, Urbana, IL, USA, 2000.

[85] A. C. Singer, G. W. Wornell, and A. V. Oppenheim. Codebook prediction: A nonlinear signal modeling paradigm. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages 325–328, 1992.

[86] J. Sjöberg, Q. Zhang, L. Ljung, A. Benveniste, B. Delyon, P. Y. Glorennec, H. Hjalmarsson, and A. Juditsky. Nonlinear black-box modeling in system identification: a unified overview. *Automatica*, 31(12):1691–1724, 1995.

[87] A. Skeppstedt. Construction of composite models from large data-sets. Licentiate Thesis, Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden. Thesis No. 149, 1988.

[88] A. Skeppstedt, L. Ljung, and M. Millnert. Construction of composite models from observed data. *International Journal of Control*, 55(1):141–152, 1992.

[89] E. D. Sontag. Interconnected automata and linear systems: A theoretical framework in discrete-time. In R. Alur, T. A. Henzinger, and E. D. Sontag, editors, *Hybrid Systems III - Verification and Control*, number 1066 in Lecture Notes in Computer Science, pages 436–448. Springer-Verlag, 1996.

[90] A. Stenman. *Model on Demand: Algorithms, Analysis and Applications*. PhD thesis, Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden, 1999.

[91] J.-E. Strömberg, F. Gustafsson, and L. Ljung. Trees as black-box model structures for dynamical systems. In *European Control Conference*, pages 1175–1180, Grenoble, France, July 1991.

[92] O. Stursberg and S. Kowalewski. Approximating switched continuous systems by rectangular automata. In *European Control Conference*, Aug.–Sept. 1999.

[93] S. A. Vavasis. *Nonlinear Optimization: Complexity Issues*. Oxford University Press, 1991.

[94] J. Vesanto. Using the SOM and local models in time-series prediction. In *Workshop on Self-Organizing Maps*, pages 209–214, Espoo, Finland, June 1997.

[95] J. Walter, H. Ritter, and K. Schulten. Non-linear prediction with self-organizing maps. In *International Joint Conference on Neural Networks*, volume 2, pages 747–752, 1990.

[96] T. Wigren. Recursive prediction error identification using the nonlinear Wiener model. *Automatica*, 29(4):1011–1025, 1993.

[97] H. P. Williams. *Model Building in Mathematical Programming*. John Wiley & Sons, 4th edition, 1999.

[98] L. A. Wolsey. *Integer Programming*. Wiley, 1998.