Transition control based on grey, neural states

Transition Control Based on Grey, Neural States

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven,
op gezag van de Rector Magnificus, prof.dr. M. Rem,
voor een commissie aangewezen door het College van
Dekanen in het openbaar te verdedigen op
maandag 25 november 1996 om 16.00 uur

door

Jozef Mazák

geboren te Trnava

Dit proefschrift is goedgekeurd door de promotoren:
prof.dr.ir. A.C.P.M. Backx
en
prof.dr.ir. P.P.J. van den Bosch

Copromotor: dr.ir. A.A.H. Damen

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Mazák, Jozef

Transition control based on grey, neural states / by
Jozef Mazák. - Eindhoven : Technische Universiteit
Eindhoven, 1996. - XIV, 160 p.
Proefschrift. - ISBN 90-386-0210-3
NUGI 832
Trefw.: neurale netwerken / intelligente regelsystemen /
toestandsruimte-analyse.
Subject headings: neural nets / process control /
state-space methods.

Preface

This thesis completes four and a half years of research in the area of using neural networks for transition control of partly-known nonlinear dynamic systems. This research was done at the Eindhoven University of Technology, Faculty of Electrical Engineering, at the Measurement and Control Section of the Measurement and Control Systems Department.

It was a great pleasure for me to work in the Measurement and Control Group for more than four years. It was also a great opportunity for me to experience work at a Western university and to experience life in the Netherlands.

I would like to thank to many people who supported me during this research. First of all, I would like to express my gratitude to Ton Backx, Ad Damen, and Siep Weiland for their valuable discussions, suggestions and comments concerning all parts of my research. My very special thanks go to Ad Damen for his extra encouragement and help not only in my research but also in my personal life. Next, I would like to thank to Ton Backx who enabled me to meet several people at the AspenTech/IPCOS company to obtain the necessary information concerning modelling of the polymerization reactor.

Several individuals have read the manuscript and their comments have helped me to improve the accuracy and clarity of the text. I am pleased to acknowledge the assistance of all members of the Promotion Committee, especially the assistance of my promotors prof. Ton Backx, prof. Paul van den Bosch and dr. Ad Damen and also the assistance of of prof. Joos Vandewalle from the Katholieke Universiteit Leuven, prof. H. Verbruggen from Delft University of Technology, prof. dr. ir. W. M. G. v. Bokhoven, prof. dr. ing. H. A. Preisig, TUE, dr. ir. P. J. M. Cluitmans and S. Weiland.

I wish to express my thanks to all my colleagues. I would like to thank to my room-mate Leon Ariaans who was willing to answer my questions at any time and help with all kinds of problems. Further, I wish to thank Robert Jan Gorter, Heinz Falkus and Yvo Boers for having me as a friend. I would like to thank to Udo Bartzke for the computing support and many restarts of the windows manager on my "Alpha" station, and Wim Beckers for his help in making me acquainted with the VMS operating system. I would like to thank to my students J.F. Balseiro who did a lot of work concerning of rigorous modelling of the polymer reactor and Marcel van Garderen who wrote the software for solving nonlinear optimal control problems.

I also wish to thank Barbara Cornelissen for proofreading some parts of the English text, for her valuable advice addressed to my wife in taking care of our baby and helping my family during the period of last two years of my research.

Finally, I am grateful to my wife Diana, not only for giving me a son but also for her patience and understanding during the time I devoted to my research. I am also grateful to my parents and my sister Renata supporting me for the last six years while I was abroad.

Eindhoven, October 14, 1996

Abstract

This thesis is devoted to studying the use of neural networks for the design of controllers which steer the process state between different operating points. The advantage of our approach is that by means of a single nonlinear controller we cover a broad range of the process operating conditions. For a practical application this means a speed up of transitions of the process state between different operating points while maintaining a good level of optimality.

Our approach covers all phases of a practical controller design. We consider: (1) process modelling issues in a form of nonlinear grey-box state-space neural models, (2) process state estimation issues by means of design of a nonlinear neural state observer and finally (3) control issues related to a nonlinear neural state feedback tracking controller. The design takes into account process constraints, process disturbances and measurement noise.

As a mathematical model of the process we consider a nonlinear state-space model parametrized by a combination of an a priori known analytical part and a black box neural network part. In the state vector of the model we distinguish between white, physically well defined states and black or hidden states. The model is estimated as a simulation model of the process to obtain a good simulation of the process output over a long time horizon. The neural net of the model is trained, in output error set-up, on measured process input/output data. The choice of a state-space parametrization of the model enables inclusion of a priori process knowledge into the model in a conceptually easy way. It also enables us, later on, to define proper reference signals for the controller.

The simulation model of the process is then supplemented with a nonlinear filter gain, parametrized by a static neural network, to improve the state predictions obtained by the previously estimated simulation model due to process disturbances. Various options for the filter gain parametrization are also considered in this thesis. The filter gain neural network is trained using the measured process data, independently of the simulation model parametrization, completing the second step of the proposed controller design.

The transition controller is a nonlinear static state feedback parametrized also by a neural network. The controller neural network is trained on the simulation model of the process such that the model states follow prescribed reference trajectories. A discussion of various choices of the state reference signals, including an optimal choice, is also given in the thesis. In order to remove final tracking errors an integral action is introduced into the closed loop. The process constraints are handled by proper specification of reference signals, which is done by means of the white-box part of the model and then by choosing of weighting factors in a control criterion.

All nonlinear functions being estimated at different steps of our algorithm are parametrized by sigmoidal feedforward neural networks. To train the neural network we distinguish between gradient-based deterministic optimization and stochastic optimization. A number of methods are reviewed in the thesis in order to obtain an effective combination of these two optimization techniques which is then used for neural network training.

There are a number of examples given in this thesis demonstrating both modelling issues and control issues involved in the topic of this thesis. The most important examples include: (1) a gantry crane process to demonstrate the state-space modelling procedure of a nonlinear process and (2) a fluidized bed polymerization process is used to demonstrate both modelling and transition control issues. A nonlinear state feedback controller is considered for a swing-up problem of a multi-link inverted pendulum.

Contents

Glossary

Symbols

x	state vector
u	control or input
y	output
w	process disturbance
v	measurement noise
r	reference signal
n	state dimension
m	control dimension
p	output dimension
q	system disturbance dimension
s	reference dimension
Θ	parameter vector
θ_i	ith component of θ
χ	neural network input
γ	neural network output
N_L	number of processing layers of a neural network
N_{Ni}	number of hidden nodes in the ith hidden layer
$\mathcal{D}$	data set
$\mathcal{N}(m,\sigma)$	normal distribution with a mean value m and a standard deviation σ
$\mathcal{E}\{\cdot\}$	expectation operator
$\mathcal{C}^\infty$	set of differentiable functions of any order
$\mathcal{C}^2$	set of two times differentiable functions
$\mathbb{R}$	set of real numbers
$\mathbb{R}^n$	set of column vectors of dimension n
$\mathbb{Z}$	set of integers
$\mathbb{Z}^+$	set of positive integers inclusive zero
τ	integration step size
T_s	process sampling time

Notational conventions

a^T	transposition of a vector a
$\hat{a}$	estimated value of a
$\|x\|$	absolute value of x
$\|\|x\|\|_2^2$	squared Euclidean norm of a vector $x \in \mathbb{R}^n$ $\|\|x\|\|_2^2 = x^T x$
z^{-1}	time shift operator: $z^{-1}x(k) = x(k-1)$
$\dot{x}(t)$	time derivative: $\dot{x}(t) = \dfrac{dx}{dt}(t)$
$\dfrac{\partial f[x(k),u(k)]}{\partial x(k)}$	partial derivative of a function f with respect to x evaluated at $(x(k),u(k))$: $\dfrac{\partial f[x(k),u(k)]}{\partial x(k)} := \dfrac{\partial f}{\partial x}[x,u]\Big\|_{x=x(k),u=u(k)}$
$\mathcal{J}_x^f(k)$	Jacobian matrix of a function $f \in \mathbb{R}^m$ with respect to $x \in \mathbb{R}^n$ evaluated at $x(k), u(k)$: $\mathcal{J}_x^f(k) := \begin{pmatrix} \frac{\partial f_1}{\partial x_1}[x,u] & \dots & \frac{\partial f_1}{\partial x_n}[x,u] \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1}[x,u] & \dots & \frac{\partial f_m}{\partial x_n}[x,u] \end{pmatrix}\Big\|_{x=x(k),u=u(k)}$

Abbreviations

MLP	multilayer perceptron
I/O	input/output
p.d.f.	probability density function
FBPR	fluidized bed polymerization reactor
CPU	central processing unit

1 Introduction

One of the most challenging problems of the system theory is the problem of nonlinear control of nonlinear dynamic systems. This challenge is interesting not only from a theoretical point of view where new theories can be developed but also from practical point of view as the majority of technological processes show complex nonlinear behaviour within present operating conditions. A typical example of an industrial control problem is a transition of process operating conditions from one operating point to another. This type of control will also be a general topic of this thesis. In the following section we explain why this problem is so interesting and put forward problems to be solved.

1.1 Motivations

Taking a linear approximation of the process behaviour restricts the control quality up to a certain level. It is quite logical to realize that an increase of the quality of the process control can be reached by abandoning the linearity assumptions about the process dynamics and trying to treat the control problem in a full nonlinear set-up. However, the nonlinearities in the process behaviour and the process dimensionality might be so high that we will still have to approximate the process dynamics with a less complex model but this time on a higher qualitative level than using linear models. The linear control theory was brought in the past to a very high level by mathematical proofs of optimality and global convergence. A tremendous step forward was done in the robustification of linear methods. In the nonlinear framework we are faced qualitatively with completely new problems which do not exist in the linear framework. These include multiple equilibrium points of the system, bifurcations of equilibrium points, periodic solutions, chaos, strange attractors and so on. The available analytical methods do not provide us with a practically computational methodology. In practice we have often experienced that the nonlinear dynamics of a given process can be much more complex than those assumed by existing analytical methods. These are complexities like nonlinear time-delayed feedbacks, general nonlinear dependence of the dynamics on observed and unobserved inputs, high or infinite state dimensionality.

Another important aspect concerning recent control systems is that the number

of control tasks provided by these systems is still increasing. The market demands are pushing manufacturers into producing a wider range of products and therefore into periodically switching the process production between different operating points. A pragmatic solution adopted by large industries at the present moment is a control strategy based on a set of linear controllers designed for each operating point. The transition from one operating point to another is realized by a sequence of manual control actions. It is clear that manual control is not optimal as it is done by an operator using only his process knowledge without doing any optimizations at all. Optimization and consequent automation of this operation might result in an improvement of a production performance, for instance, just by speeding up the transition. Note that the products made during the transition are usually of wide specifications type as they do not meet prescribed the high specifications.

To make our research motivations more clear we present at this point an example of a nonlinear dynamic control process, namely a fluidized bed polymerization reactor (FBPR) widely used in petrochemical industries, for example for an ethylene polymerization. This type of reactors can also be found in biotechnological and coal industries. The process itself is schematically depicted in Figure 1.1. This process will be discussed in detail later on in Chapter 6 when we will demonstrate our control algorithms. The process consists of a reactor, a heat exchanger and primary controllers. The reactor is fed at the bottom with monomer and comonomer masses in gas form. These polymerize in the reactor with the help of a catalyst. The mass in the reactor is composed of solid particles of polymer through which bubbles of gas rise. Recycled, un-reacted gases are cooled down in the heat exchanger and then added to the incoming gas flow. The temperature of the cooling water T_w in the heat exchanger is controlled by a primary PID controller to stabilize the process dynamic. The mass input flows are manipulated by a pressure controller which keeps a constant pressure at the top of the reactor through compensation of fast process disturbances. The final product is withdrawn from the reactor at a rate Q_0.

The dynamics of the FBPR are quite complex, and as we stated at the beginning of this section, this is often the case. A rigorous modelling of this process is based on a set of simplifying assumptions. Basically we take here the mass and energy balances to build up a mathematical model which is then of restricted fidelity. We assume, for example an average size of solid particles and gas bubbles in the reactor. This might not be such a bad assumption, but if it is not properly chosen, the overall mathematical model might be wrong [14]. This and similar model simplifications limit the performance of the control system being designed using these models. If the performance of a control system has to be further optimized we have to consider first the process modelling issue and try to build up a more accurate mathematical model of the process dynamic. This would enable us to increase both robustness and performance of the control system.

A practical operation of technological processes is often complicated by the fact that the process behaviour is influenced by disturbances. For instance, in the case of the polymerization reactor, it can be a fouling of the heat exchanger, impurities

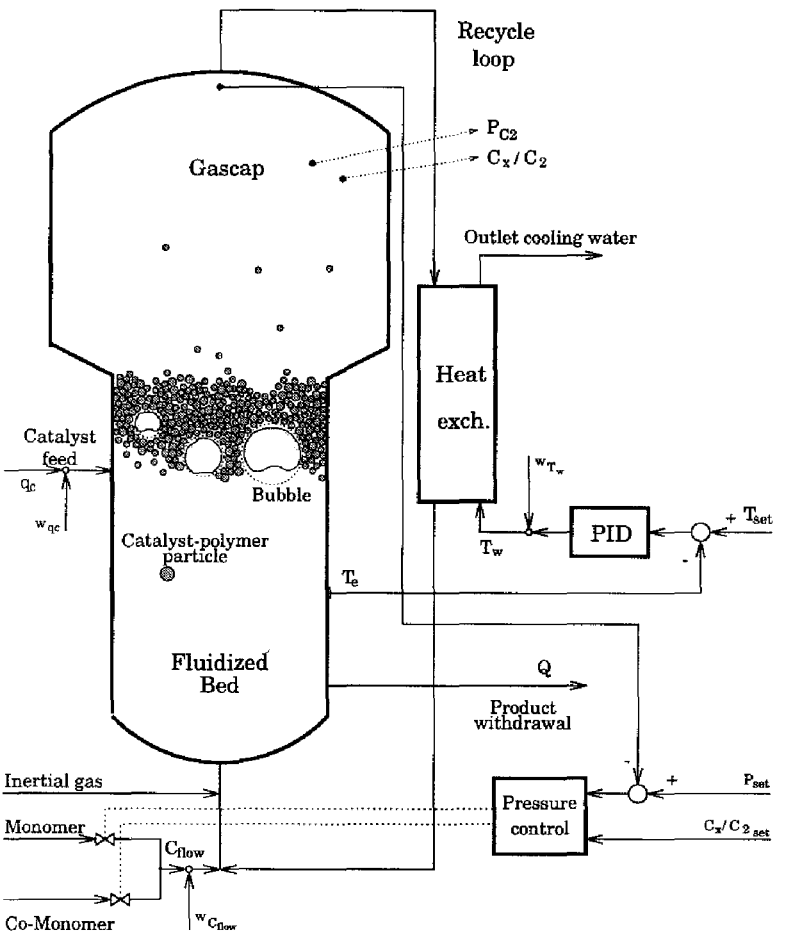


Figure 1.1: Fluidized bed polymerization rector diagram

in the input mass flows or the catalyst activity fluctuations in the reactor. The main difficulty is not in determining the stochastic characteristics of disturbances, these can be estimated rather well, but in their nonlinear effect to the process outputs. Existing theories of nonlinear stochastic systems assume additive disturbances either at the process state or at the process output. If we could model also the effect of disturbances to controlled outputs we could then further improve the quality of control.

Along with the problem of a complex nonlinear process behaviour and process disturbance we are also faced with the problem of high process dimensionality. By this we mean that the process dynamics are described by a large number of differential equations whose order can also be rather high. Often these equations contain also partial derivatives which further complicate the process description. For instance, a heat exchanger is described basically by partial differential equations, but to translate these equations to ordinary differential equations we have to consider a rather large number of equations. For such a high-dimensional nonlinear dynamic system it is very cumbersome to use analytical techniques.

In practice, we often have a good prior knowledge about the process we are going to control. The first principal dynamic relations concerning some of the process variables are usually also known. In the case of the FBPR these are for example the concentrations of the monomer and the co-monomer, as we know how they relate to the pressure. This also means that the dynamics of the process is partly known and the known part of the process dynamics can be explicitly brought into the model parametrization.

From the previous discussion it is clear that for the controller design a good mathematical model of the process dynamics and disturbance effects will be unavoidable. A good model should not be too complex, as it will be used later on in the further optimization of the controller, and it should not oversimplify the process dynamics. In general, it should be a nonlinear model. The controller, in general, should be nonlinear as well. In our design we have chosen a state-space realization of the controller. Our motivations for this choice are: (1) having a possibility of an easy way of including a prior process knowledge about the process into the model; (2) to be able to parametrize approximated nonlinearities by static nonlinear functions; (3) to be able to model the effect of disturbances at the process output; (4) the controller is a static state feedback. The state of the process is, in general, not measured directly and has to be estimated.

Therefore an immediate problem arising at this point is how to parametrize the nonlinear functions of all the components of the controller, which will be designed.

1.2 Neural networks

Let us turn our attention for a while to biological systems, e.g. the human body. In daily practice we perform a number of control tasks without noticing any problems there unless we are in a good health. Presumably, these are optimal and efficient. But do we know how are we really solving these tasks? Isn't it also possible to

mimic this approach in industrial process control? A human body is also a complex nonlinear dynamic system. The decisions and movements we make represent similar problems we meet in the fields of pattern recognition and control. We know that all these decisions and pulses leading to control actions are being determined in our brain. We also know, that our brain is a complex system of neurons, building up perhaps the most complex, most efficient system consisting of nonlinear maps performing different tasks and storing millions of bits of information. Therefore a question can be asked whether a similar artificial neural network can be used for modelling of the process behaviour and solving control tasks.

Studies of neural networks, attempts of understanding their functionality and the build up of their artificial equivalents are quite old. The use of linear threshold units as a basic unit in neural networks was pioneered by Rosenblatt [54]. The neural network mechanism proposed by Rosenblatt is called the *perceptron*. Research on the general architecture by Rosenblatt and others continued into the late 1960s. During this period, Minsky and Papert [43] published a mathematical analysis of the perceptron showing its limitations, since the perceptron's basic computational element is a linear threshold unit. Therefore the perceptron can only discriminate between linearly separable classes. The fact that a large proportion of interesting classes of patterns are not linearly separable means that the capabilities of the perceptron are very limited. After this criticism the neural networks lost some of the attention until new learning rules were discovered. That enabled using nonlinear threshold units in the perceptron replacing the original linear units. It was also proven that such a structure can approximate any nonlinear map, and this became immediately attractive not only for control engineers but also for statisticians, computer scientists and others.

Neural network structures, used in the context of this thesis, will be discussed in more detail in a specially devoted chapter. But basically a neural network can be seen as a high dimensional nonlinear static/dynamic black-box system with many inputs and many outputs while the behaviour of this box can be adjusted freely by tuning weighted interconnections inside the network. This looks very much like a modelling problem of an unknown complex system with many inputs and outputs indicated in the previous section. If we can use measured data of process inputs and outputs, we can try to teach a neural network the process dynamics from these data. However, the process dynamics can be so complex, that by trying to model it by a black-box neural network we might end up with a not very feasible problem. In such cases we can try to combine a black-box neural network with an available a priori knowledge about the process. A way of doing it will be explained in Chapter 4.

1.3 Thesis contributions

The main contributions of this thesis are summarized as follows:

- We have proposed a complete procedure for the design of a nonlinear transition controller for a partly-known multivariable nonlinear process which brings the process state from one operating point to another, while taking into the account process disturbances and process constraints. The design was verified by a large number of simulation experiments done on a rigorous simulation model of a fluidized bed ethylene polymerization reactor [37, 38] as well as on an academic example [39].

  The controller uses neural networks to parametrize unknown nonlinearities. The design is carried out in state-space domain and the design is model based. The model of the process is a prior knowledge based state-space simulation model, where a black-box part is parametrized by a static neural network. The process state is estimated by means of a design of a state observer, similar to the Kalman design, but this time is the filter gain a nonlinear function parametrized by a neural network. The controller is then a nonlinear static state feedback parametrized by a neural network.

- In this thesis we have elaborated on a concept of nonlinear state observers based on the available prior process knowledge and using neural networks as nonlinearities approximators. A state observer is a composition of a dynamic simulation model of the process and a static nonlinear filter. The model incorporates our prior process knowledge. For this purpose we have divided the process state vector into two parts:

  1. The physically known part of the state vector represents those states of the process, which physical meaning is well defined.
  2. The hidden part represents the complementary part of the state vector containing those states of the process which are unknown.

  With this respect we divided the model into an analytically known part and into a black-box part parametrized by a static neural network. The filter gain is also parametrized by a static neural network.

- The state observer is being designed in two steps:

  1. An output-error simulation model of the process is estimated using the measured process data and a prior process knowledge;
  2. A nonlinear filter gain is estimated to improve the state estimates from the previously estimated simulation model. In this step are the weights of the model neural network set to fixed values and the weights of the filter neural network are tuned independently of them.

- It is shown in this thesis that the state-space approach for modelling partly-known nonlinear dynamic processes provides us with better conceptual and algorithmic properties than similar nonlinear dynamic I/O models, mainly in terms of an easier way of incorporating different types of prior knowledge about the process dynamics at the modelling stage.

- In this thesis we have also paid attention to the training of neural networks, as it is a problem to be solved at every stage of our controller design algorithm. Neural network training is a non-convex, high dimensional minimization problem. To deal with this problem efficiently we have investigated and tested a couple of function minimization algorithms. We have also proposed a novel training algorithm based on weighted past gradients. To balance the time spent in the minimization and the risk of getting trapped in a local minimum we have proposed a combined stochastic and deterministic optimization procedure, supplemented by a number of restarts from different initial points.

- In this thesis we have also considered the computational cost and memory requirements for different problems involving a neural network training procedure. These considerations are very seldom reported in the literature though they are significant.

- We have elaborated on a simulation model of the FBPR reactor [37], which was used to test the proposed controller design algorithm.

1.4 Chapter overview

In Chapter 2 we discuss the general framework of this thesis. We define here a general concept of a nonlinear dynamic system as it will be looked at throughout this thesis. Then we state a control problem which will be the main topic of this thesis. This control problem will be concerned with a real industrial process what will then, in fact, dictate the proposed solution. As one of our ideas is to use neural networks we review the most important existing control techniques for control of nonlinear systems based on neural networks. After this we describe, in general terms, our approach. At the end of this section we review a nonlinear function approximation problem.

In Chapter 3 we describe the class of neural networks, which we use for the approximation of unknown nonlinear functions. We address both static and recurrent feedforward neural networks. As these are structures with many parameters to optimize we investigate in this chapter some of the function minimization techniques, both deterministic and stochastic and we conclude this chapter with a suggestion of a combined deterministic stochastic minimization algorithm.

Chapter 4 is devoted to a discussion concerning modelling issues related to estimation of a mathematical model of the controlled process. In this chapter we treat both prediction models and simulation models, both in I/O configuration and

state-space configuration. A state-space model is treated as a simulation model of the process and later on it is supplemented with a nonlinear filter gain resulting in a neural process state observer. In this chapter we also demonstrate, using a simple numerical example, why simulation models are to be preferred with respect to prediction models. In this chapter we also give another, more practical, numerical example and that is an identification of a grey-box state-space simulation model of a gantry crane process.

In Chapter 5 we discuss the transition control problem. We start by an introduction of a nonlinear state feedback. We then give an example of the control of a multi-link inverted pendulum using a neural network to parametrize the state feedback. Next we discuss all issues related to the state feedback design for a transition controller. These issues include: the process model, the controller objective, the choice of a reference signal and the state feedback structure. At the end of this chapter a simple numerical example demonstrating the whole controller design procedure is given.

In Chapter 6 we give a simulation example of a transition controller design for the fluidized bed ethylene polymerization reactor. First of all, we introduce the process itself and we specify the transition control problem precisely. Then we discuss the identification experiments together with the process data simulations. Next we discuss the state observer design and finally, we describe the controller state feedback design. A validation of the controller using the original simulation model of the process is also given.

2 Framework and Problem Statement

In this chapter we discuss a general framework for the type of problems we want to address in this thesis and are essential for our global goal: *Design of a transition controller for a partly known nonlinear dynamic system.* First of all, we specify mathematical descriptions of dynamical system which will be used as general models of real processes. We will treat these concepts in state-space domain, both in continuous-time domain and in discrete-time domain. A general formalism of system theory, directly related to this framework, can be found in [75, 72]. After this, we state a general problem of this thesis: The transition control problem. Next we review the most important control schemes based on neural networks and propose a scheme which will be worked out in this thesis. We conclude this chapter by a general framework for function approximation problems.

2.1 System descriptions

A broad class of nonlinear dynamic state-space systems Σ_s can be described in continuous time $t \in I\!R,\ t > 0$ by the following set of equations

$$\dot{x}(t) = f_c[x(t), u(t), w(t)] \tag{2.1a}$$
$$y(t) = h_c[x(t), u(t), v(t)] \tag{2.1b}$$

Both f_c and h are assumed to be smooth, nonlinear, multivariate functions of their arguments given by

$$f_c[x,u,w] = \begin{pmatrix} f_{c,1}[x,u,w] \\ \vdots \\ f_{c,n}[x,u,w] \end{pmatrix} \qquad h[x,u,w] = \begin{pmatrix} h_1[x,u,v] \\ \vdots \\ h_p[x,u,v] \end{pmatrix}$$

where we omitted the time arguments for time signals. The equation (2.1a) is called the state equation and describes the time evolution of the state $x \in I\!R^n$ of the system assuming that $x(0) = x_0$ is the initial condition of the state at the time $t = 0$. The other two arguments of the function f belong to the input space

of the system: $u \in \mathbb{R}^m$ is an observed system input and is called the control input or simply control, $w \in \mathbb{R}^q$ is a non-observed system input and is called the process disturbance or simply disturbance. The equation (2.1b) describes the measurement system, attached to the system evolution, and therefore we often call this equation a measurement equation. The measurements are denoted by a vector $y \in \mathbb{R}^p$. The variable $v \in \mathbb{R}^r$ is another non-observable input signal which stands for the measurement noise. It is quite usual to assume that the measurement noise is additive to the output and then the dimension of the measurement noise is equal to the dimension of the output vector, or $r = p$.

Similarly to (2.1), a linear time-invariant state-space dynamic system can be described as follows

$$\dot{x}(t) = Ax(t) + Bu(t) + Gw(t) \tag{2.2a}$$
$$y(t) = Cx(t) + Du(t) + Hv(t) \tag{2.2b}$$

where A, B, C, D, G, H are, in this case, constant matrices. The system (2.2) is a special case of a system described by (2.1). An advantage of considering the system description (2.2) is, for instance, that this description satisfies the supperpossition principle and the differential equations (2.2a) are readily solved. However, the real-world systems are nonlinear and the system description (2.2) is only valid for limited ranges of system variables.

Let a sampled version of the system Σ_s is described in the discrete-time domain by

$$x(k+1) = f_d[x(k), u(k), w(k)] \tag{2.3a}$$
$$y(k) = h_d[x(k), u(k), v(k)] \tag{2.3b}$$

where $k \in \mathbb{Z}^+$ stands for the time index or time step and the meaning of other variables is similar to those in (2.1). The state in this description evolves at equal discrete-time moments also called sampling moments. The time interval between two successive samples is the sampling time and will be denoted by T_s.

A transformation of a continuous-time system given by (2.1) to its equivalent description a discrete-time system (2.3) and vice-versa is not trivial for a general nonlinear system. In this thesis we will require only transformations from continuous-time to discrete-time domain. The reason is that the first principles of real process behaviours provide us with mathematical laws of type (2.1) while e.g. for a numerical optimization of controllers using a digital computer descriptions of type (2.3) are more convenient. In a lack of an analytical solution to a general problem of transformation of (2.1) to (2.3) we often take an approximation of this transformation. As there are many ways of constructing such an approximation and these were used later on in this thesis in different simulation examples we review some of them in Appendix A.

We know that the representation of the process given either by equations (2.1) or by equations (2.3) is unique up to modulo any state transformation of type

$$\bar{x} = \alpha[x] \tag{2.4}$$

where $\alpha : \mathbb{R}^n \to \mathbb{R}^n$ is a smooth and invertible function and x is the state, either continuous or discrete. This ambiguity of the state-space representation can be avoided by choosing a certain structure for f_c, h_c or f_d, h_d. The importance of this restriction is given by preserving the physical meaning of states under study.

Let us consider the continuous time nonlinear dynamic system Σ_s described by (2.1). We call a triple $\mu = (x_0, u_0, w_0)$ an equilibrium point of this system (2.1) if

$$0 = f_c[x_0, u_0, w_0] \tag{2.5}$$

Similarly, an equilibrium point of a discrete-time dynamic system (2.3) is defined by

$$x_0 = f_d[x_0, u_0, w_0] \tag{2.6}$$

In the following we will consider only those equilibrium points, either defined by (2.5) or by (2.6) for which $w_0 = 0$. We will call x_0 an equilibrium state and u_0 an equilibrium control.

The disturbances w entering the state equation, either in (2.1) or in (2.3) can be deterministic or stochastic or combination of these two. In this thesis we consider mainly stochastic disturbances of white noise type. Strictly speaking, one can not simulate a white noise sequence in continuous time over its full frequency range. Therefore it is often replaced in the literature (see e.g. [40]) by dw, which is then a Wiener process. This delicate problem will be overcome, when we will consider a discrete-time process description.

By restricting ourselves in this section to a specific class of nonlinear dynamic systems we are ready to state a control problem related to these systems.

2.2 The transition control problem

The control problem stated in this section will be studied in this thesis. We use the continuous time domain for its statement as this is more natural when having in mind a physical process. However, the actual implementation of the solution, we will propose later on, will be done in the discrete-time domain as a digital computer is going to be used in the place of a control device.

The next definition defines a control function of the system (2.1): A measurable function $u \in \mathbb{R}^m$ defined on an interval $\langle t_0, t_f \rangle$ is said to be a control on $\langle t_0, t_f \rangle$ if there exists a function $x(t) \in C^\infty$ defined on $\langle t_0, t_f \rangle$ such that

1. $x(t) \in \mathbb{R}^n$ for all $t \in \langle t_0, t_f \rangle$
2. $\dot{x}(t) = f[x(t), u(t), w(t)]$ on $\langle t_0, t_f \rangle$ for any measurable $w(t)$.

This definition says that $u(t)$ is any measurable function for which the system of differential equations (2.1a) has a solution provided an initial condition $x_0 \in \mathbb{R}^n$ is given. The function $x(t)$ is called a *state trajectory* corresponding to $u(t)$ called a *control trajectory*.

Let us consider now a nonlinear closed-loop dynamic system according to Figure 2.1. The variables u, y, w, v denote the already introduced control, output, process disturbance and measurement noise signals. Let the control system or simply controller be represented by a nonlinear dynamic system Σ_s^c, similarly to (2.1). A general task of any controller is to ensure a certain specific behaviour of the controlled system. To do this the controller takes the process output y and an external signal $r \in \mathbb{R}^s$ as its inputs and computes a control signal u offered to the system input. The signal r is called a reference signal. The behaviour of the closed loop is judged by the controller via a new output signal $z \in \mathbb{R}^{n_q}$ which represents an optimized output and is defined as follows

$$q(t) = d[x(t), u(t), r(t), w(t), v(t)] \tag{2.7}$$

where d is a nonlinear function, for simplicity assumed to be smooth.

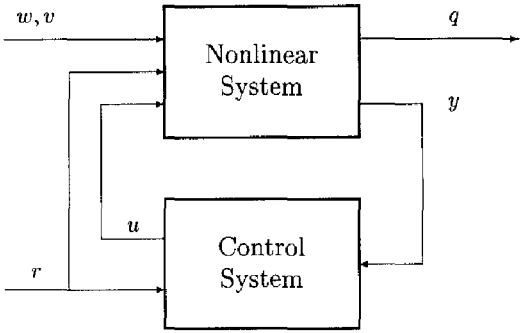


Figure 2.1: Feedback control loop

Let us assume two different equilibrium points of the controlled system Σ_s, μ_0 and μ_f. Then the transition control problem can be stated as follows:

Problem 2.1. Find a controller Σ_s^c, which steers a nonlinear dynamic system (2.1) from a neighbourhood of one equilibrium point μ_0 to a neighbourhood of another equilibrium point μ_f in an optimal way to be defined.

This problem can be mathematically described as follows: Minimize a cost functional

$$J = \Psi[q(t_f), t_f] + \int_{t_0}^{t_f} L[q(t), t]dt \tag{2.8}$$

where $\Psi[q(t_f), t_f] \in C^2$ is defined on $\mathbb{R}^{n_q} \times \mathbb{R}$ is the final-time penalty put on $q(t)$, $L[q(t), t] \in C^2$ is a Lebesgue integrable function on $\mathbb{R}^{n_q} \times (t_0, t_f)$, subject to:

a) the differential system equations (2.1) describing the controlled system dynamics;

b) the initial state of the system given by

$$x(t_0) \sim \mathcal{O}(\mu_0) \tag{2.9}$$

where $\mathcal{O}$ stands for a neighbourhood of μ_0;

c) the final state condition defined by

$$c_f[x(t_f), \mu_f, t_f] \leq 0 \tag{2.10}$$

where $c_f \in C^\infty$;

d) the control inequality constraints

$$c_u[u(t), t] \leq 0 \tag{2.11}$$

where $c_u \in C^\infty$;

e) the inequality constraints expressing the forbidden region of the state space

$$c_x[x(t), t] \leq 0 \tag{2.12}$$

where $c_x \in C^\infty$.

Discussion

- The above stated problem is a free final-time optimal nonlinear control problem with mixed state and control constraints.
- All process variables, $u(t), y(t), x(t), q(t)$, are stochastic processes due to the presence of random process disturbances and measurement noise. To be precise in the above problem formulation all these variables should be replaced by their expected values. As the problem is nonlinear, an analytical determination of expected values of these variables is, in general, not trivial.
- An initial state condition $x(t_0)$ of the controlled system is assumed to be known only approximately. This is expressed by the constraint (2.9). It tells us that the initial state of the system can be found in a neighbourhood of the initial equilibrium point μ_0. It is more likely that the initial state is closer to μ_0 than far away. Therefore the neighbourhood of the initial equilibrium $\mathcal{O}(\mu_0)$ can be probabilistically described by a normal distribution $\mathcal{N}(\mu_0, \sigma_{\mu_0})$.
- A similar reasoning as the one for the initial state of the system applies also for the final state of the system. As the process is subject to disturbances the requirement of reachability of the final equilibrium μ_f exactly is relaxed to other conditions, namely that the final state of the system $x(t_f)$ should be sufficiently close to μ_f. This problem can be also handled by the "tail" $\Psi[q(t_f), t_f]$ of the cost functional (2.8) by penalizing large deviations of the final state value from μ_f.

- In practice, all process variables have physically defined ranges of operational values. These values can not be exceeded, mainly due to safety reasons. Often also time changes of manipulated variables can not exceed certain limits. The same holds also for process states and outputs. For example, in the case of the polymerization reactor introduced in Chapter 1 (see Figure 1.1), a fast increase of the temperature can cause melting of solid particles and consequent collapse of the fluidized bed. The constraints put on process variables does not have to be only simple bounds put on magnitudes of these signals but and can be considered as a general nonlinear constraints.

- We have required in this transition control problem definition that all constraints are smooth functions. This requirement could be in fact relaxed, as the later on used numerical optimization methods require only continuity of first derivatives, that is $\{c_f, c_u, c_x\} \in \mathcal{C}^2$ might suffice.

- In practice, there is always an existing control system present in the process, e.g. primary controllers, and we can assume that there are present also some steady-state controllers designed for specific equilibrium points. A transition from one operating point to another can possibly start by switching from a steady-state controller to a nonlinear transition controller (design of which is studied in this thesis). The transition controller changes the process operating conditions to the new ones which are in the neighbourhood of the final operating point. Finally, we can let a new steady-state controller to take over. The transition controller is in principle nonlinear as it controls the process over nonlinear regions. The steady-state controllers are linear because they control the process only close to an equilibrium.

2.3 *Neural control paradigms*

Before we describe the approach presented in this thesis to approximation of a solution of the above stated control problem we review the most important control techniques based on neural networks as our approach is closely related to them. We assume, that the reader has already got a general knowledge about neural networks, so we give the following discussion without going into the details concerning neural networks as to this discussion is devoted the Chapter 4.

In the past decade a large number of, prevailing model-based control algorithms has been proposed to achieve better control quality and robust controllers. All these techniques rely strongly on the availability of a mathematical model that is a good representation of the process dynamics. However, most of these models are empirical, first principal models or linear estimates of the true process dynamics. The neural networks methodology offers an alternative for the derivation of proper dynamic models of the system. In addition, it is possible to take advantage of the potential of neural networks to device new control strategies that are impossible with conventional methods.

2.3.1 Direct inverse control

The most appealing feature of neural networks is the ability of the inversion of complex dynamic systems. The inverse system model can be generated from input/output process data and then cascaded with the controlled process such that the composed system results in an identity mapping between the desired response and the controlled process output. Clearly, the quality of such a control will be determined by the accuracy of the process inverse model. The other problem is the lack of the feedback. This affects the robustness of the direct inverse design with respect to the process disturbances or inverse model discrepancies. Provided that the process dynamics are non-minimum phase, an inverse of the process will be unstable and the control loop will be internally unstable.

A more appropriate structure of the direct inverse control is shown in Figure 2.2. This structure is in the literature usually addressed as a specialized learning [51]. The neural controller is trained to find the process input u that drives the process output y to the desired reference value r. The drawback of this procedure lies in the requirement of knowledge of the process Jacobian $(\partial y/\partial u)$. As this is in general not available heuristics and different approximations are used at this place.

The learning algorithm represents here a quasi-feedback which, if implemented on line, can contribute to the robustness, however the speed of the neural controller parameter adjustment might be too slow for compensation of fast disturbances.

2.3.2 Model reference control

The structure of this control scheme is shown in Figure 2.3. The desired performance of the closed-loop is specified by through a stable reference model which output is compared with the process output and the error signal e is then used to adjust the neural network controller. Again, the Jacobian of the process might be required in the adaptation mechanism when minimizing the squared error signal by a gradient descent. In the case when the reference model is chosen as an identity mapping the model reference control coincides with the direct inverse control.

An advantage of this structure is in better robustness and the possibility of adaptation when the neural controller is trained on-line. However, the non-convexity of the adaptation mechanism might be of a real practical limitation here.

2.3.3 Adaptive neural control

This type of control is meant for processes with time variable or changing dynamics and is suitable also for an on-line implementation. It consists of a recursive process parameter identification to track the process environmental conditions or parameters and a procedure to adjust the control parameters (see Figure 2.4). The process parameters are in fact the weights of a neural network which are being adapted.

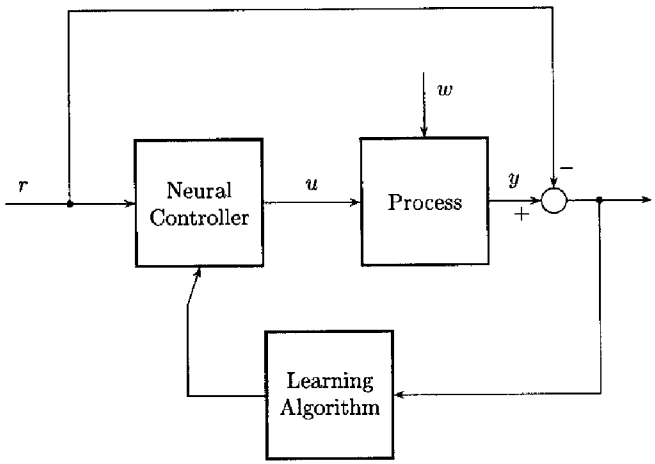


Figure 2.2: Specialized learning architecture

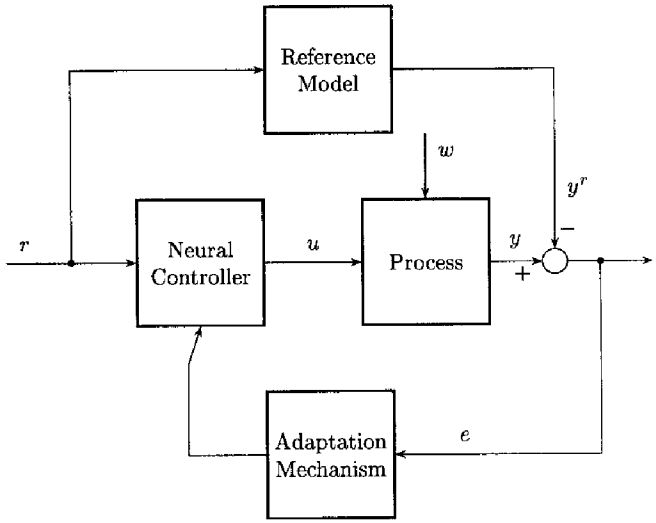


Figure 2.3: Model reference control block diagram

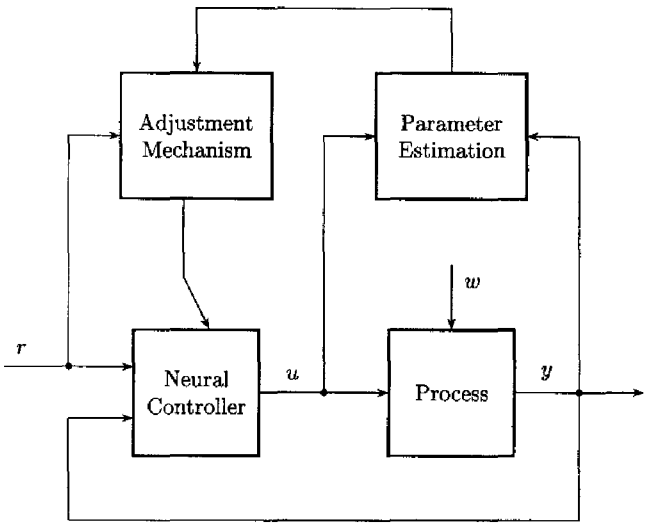


Figure 2.4: Adaptive control

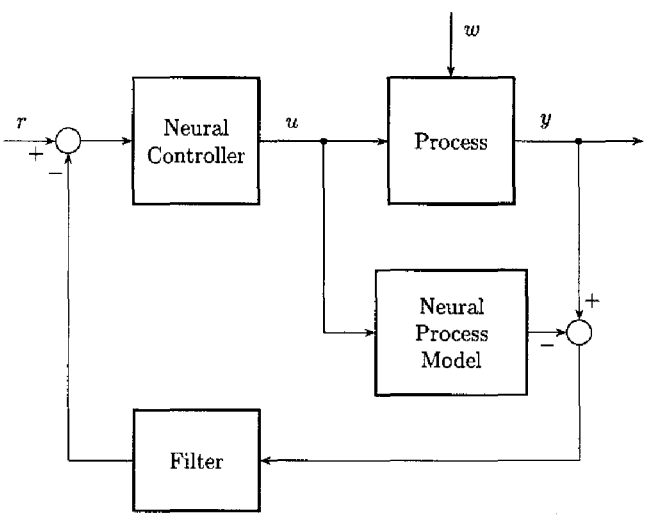


Figure 2.5: Internal model control

The estimated neural process model is then used for a neural controller parameter adjustment.

The problems of this scheme are similar to the problems arising in a "classical" linear set-up, like stability and persistent excitation of the identification algorithm. The biggest problem is a guaranteed convergence of this scheme as all being recursively solved subproblems are hard non-convex optimizations.

2.3.4 Internal model control

The internal model control structure (IMC) is illustrated in Figure 2.5. A neural model of the process is connected in parallel with the process and the difference of the process and model outputs is used for feedback. The neural controller is then related to the process inverse and therefore is this structure limited to open loop stable minimum-phase systems. The filter in the feedback is introduced due to the practically imperfect process model and measurement noise to improve the sensitivity characteristics of the closed loop.

As the IMC structure is theoretically well understood and theoretically supported by stability and robustness proves at least in a linear set-up it is a good candidate for a general scheme of a controller for our control problem.

2.3.5 Model-based predictive control

This type of control became very popular in recent years due mainly to its ability to handle process constraints effectively. This strategy includes an optimization routine which is used to determine the optimal sequence of future controls to minimize an objective function. The receding horizon control approach is a principal concept applied here. The optimizer computes a range of future controls to minimize the cost function based on predicted future process outputs $\hat{y}(k+d)$ over a long-range time horizon $d = 1, \ldots, L_y$. However, only the first control value of the sequence is applied at the process input and the whole procedure is repeated. The structure of this structure is shown in Figure 2.6.

If the controller and predictor are assumed to be multilayer neural networks parametrized nonlinearly with respect to their outputs than the optimization problem is a non-convex one. The only way out seems to be using linear parametrizations like radial basis neural networks.

2.4 The proposed control strategy

Our approach to the controller design is a model-based design. The model serves as a simulator of the process dynamics. This is then used to optimize a state feedback controller. Our general idea is that if the model can be improved, e.g. with respect to existing models, we can possibly achieve better robustness and

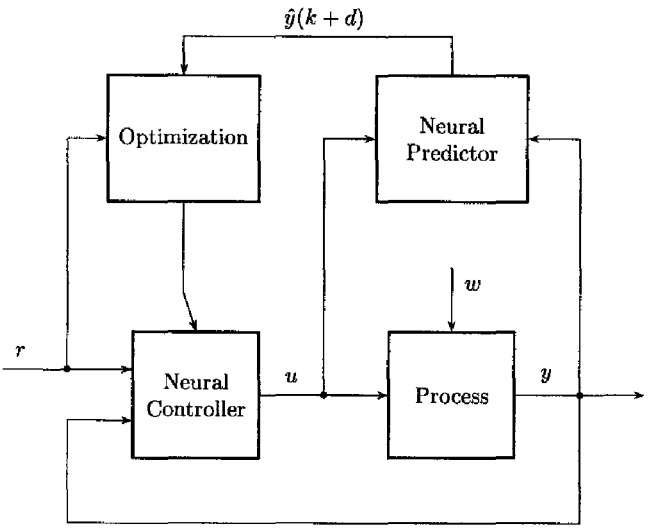


Figure 2.6: Model-based predictive control

better performance of the closed loop. A major improvement is expected from using neural networks to parametrize both the model and the controller.

A general strategy of the controller synthesis adopted in this thesis is described by these three stages:

1. Using measured process data we estimate a nonlinear, a priori knowledge-based dynamic state-space simulation model of the process which captures the control relevant dynamic relations between manipulated variables and controlled variables. The a priori knowledge is brought into the model by means of combination of known analytical part with a black-box neural network part. This allows us to give a physical meaning to a part of the estimated state vector and later on define reference signals for required transitions.

2. At the second stage we compute a static nonlinear filter gain to build up a state observer reconstructing the process state in a disturbed environment. An important point to stress here is that the filter gain is going to be computed for a fixed model preserving its simulation capabilities.

3. At the last stage we compute a nonlinear, in principle static, state feedback controller which steers the process from one operating point to another. The reference signal is defined basically in the model state-space domain but is partly physically related to the physical process states. The feedback gain is optimized such that the tracking errors are minimized.

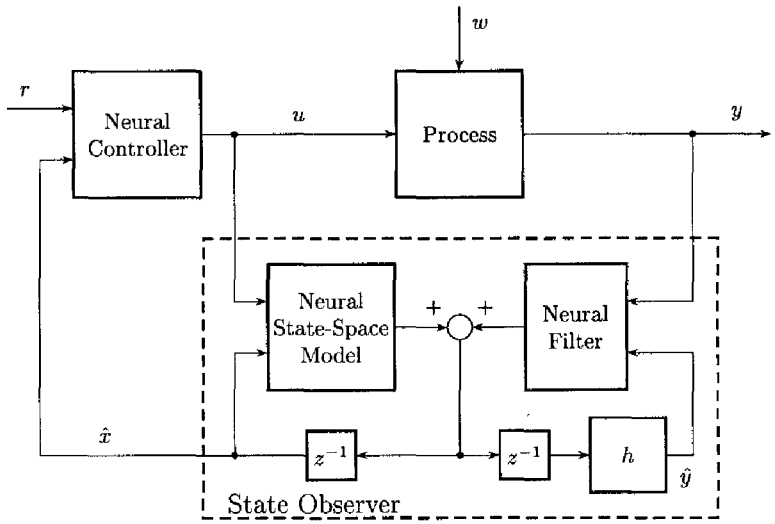


Figure 2.7: Proposed control structure

With respect to this strategy the controller will be a complex nonlinear dynamic system composed of two main blocks and two sub-blocks as shown in Figure 2.7. The main blocks are the state observer and the neural controller. The state observer is composed of two sub-blocks, which are the process simulation model and the filter gain. The block h represents a part of the process prior knowledge.

By means of this internal structuring of the controller we hope that we will gain, with respect to black-box strategies, in a better insight into the controller design problem, in an easier analysis of results and it will allow us doing the modifications of the controller in an easy way if the controller requirements changes.

2.5 Approximation theory

Later on we will see that in the vast majority of problems of a controller design appears a subproblem of approximation of an unknown nonlinear function from examples. The problem of function approximation has been treated quite extensively in literature, see e.g. [6]. For our needs the following formulation suffices:

Problem 2.2. Let $\Phi(\chi)$ be a real-valued continuous function defined on a set $\mathbb{R}^{n_\chi}$ and let $\widehat{\Phi}(\chi, \Theta)$ be a real-valued approximating function depending continuously on χ and on n_θ parameters, Θ. Given a distance function ρ, determine the parameters

$\Theta^* \in \mathbb{R}^{n_\theta}$ such that

$$\rho[\widehat{\Phi}(\chi, \Theta^*), \Phi(\chi)] \leq \rho[\widehat{\Phi}(\chi, \Theta), \Phi(\chi)] \tag{2.13}$$

for all $\Theta \in \mathbb{R}^{n_\theta}$.

The distance measure ρ is a measure of the goodness of approximation and is usually given as the L^p norm of the distance $\widehat{\Phi}(\chi, \Theta) - \Phi(\chi)$, called also error, and is defined as follows

$$\rho = L^p[\widehat{\Phi}(\chi, \Theta) - \Phi(\chi)] = \left[\int_{\mathbb{R}^{n_\chi}} |\widehat{\Phi}(\lambda, \Theta) - \Phi(\lambda)|^p d\lambda\right]^{1/p}, \qquad p \geq 1 \tag{2.14}$$

In practice, mainly $p = 1$, $p = 2$ and $p = \infty$ is used. In the first case we have to find such a Θ that the median of the approximation error is minimal. This type of approximation typically allows occasionally big approximation errors. In case $p = 2$, the distance measure (2.14) translates to the usual Euclidean vector norm. An approximation done using this distance measure allows only for small errors and gives minimum error variance. The case $p = \infty$ is the so called worst case because it minimizes a worst approximation error. The most interesting case is the choice of $p = 2$ as this leads to a differentiable optimization problem which can be solved numerically by available gradient optimization methods. As the original function Φ is not known the actual approximation is done on a set of test data produced by this function and the integral in (2.14) is replaced by a sum.

In the context of neural networks, which will be introduced in the next chapter, $\widehat{\Phi}(\chi, \Theta)$ is parameterized by a neural network and Θ is a vector of network's weights and biases.

2.6 Summary

In this chapter we defined a general framework necessary for the type of control problems we will be dealing with later on. We have also defined a control problem which will be the main subject of this thesis.

We have chose state-space descriptions of nonlinear dynamic systems, considered either in continuous time domain or in discrete-time domain. The main reason why we have chosen state-space descriptions is that in the MIMO case ($m > 1$ and/or $p > 1$) the other forms are not simpler to work with than the state-space form. Besides of this, in practice, we always can point to some variables, which do not belong to the output space of the process. These variables can be usually reconstructed from measurements and then used for feedback. This already suggests the use of an estimated process state for the feedback.

The type of control problem we are going to study include the operating point changing type of control. We use neural networks as a tool for approximations of unknown nonlinear functions. We have reviewed the most important neural control techniques using neural networks and then proposed a scheme which should be suitable for a practical application.

At the end of this chapter we addressed a problem of approximation of nonlinear functions as this problem relates to the problem of neural network training.

3 *On Neural Networks*

In this chapter we discus the concept of a neural network as we apply it in process modelling and process control. Neural networks are in our context considered purely from a mathematical point of view as universal approximators of nonlinear functions and not as biological systems. However, many notions related to biological networks are inherently used also here like "neuron" instead of a special nonlinear function and also "learning" or "training" instead of parameter estimation or mathematical optimization. Our mathematical neural networks, as will be understood later, share a lot of properties with biological neural networks, for instance a massive parallelism, hierarchy and multitasking. All these properties make the neural networks attractive in modelling and control of complex nonlinear dynamic processes.

There are multitudes of different types of neural network architectures. To list them here would be outside of the scope of this chapter. According to our experience there are well over hundreds different neural networks proposed by many researchers, but only few of them became really popular for modelling problems [1]. This is mainly because of a strong mathematical support of these popular structures. These networks include the so called multilayer feedforward neural networks [55], radial-basis functions neural networks [44, 48] and recently also wavelet networks [2]. We will concentrate in this thesis only on multilayer feedforward neural networks called also multilayer perceptrons.

3.1 *Multilayer feedforward network*

The basic element of a neural network is a simple computational or processing unit that is characterized by

1. $\theta_w \in \mathbb{R}^{n_{\theta_w}}$ – a vector of weights
2. $\theta_b \in \mathbb{R}$ – a bias or offset
3. $s : \mathbb{R} \to \mathbb{R}$ – an activation function

If $z \in \mathbb{R}^{n_z}$ is an input vector, fed to the processing unit, the activation function computes $s(\theta_w^T z + \theta_b)$ and this value is then taken as output of the unit. If we

connect a finite number of such units in parallel into a layer and subsequently connect a finite number of such unit-layers in series only by feedforward connection we end up with an architecture called multilayer feedforward neural network or a multilayer perceptron (MLP), schematically depicted in Figure 3.1.

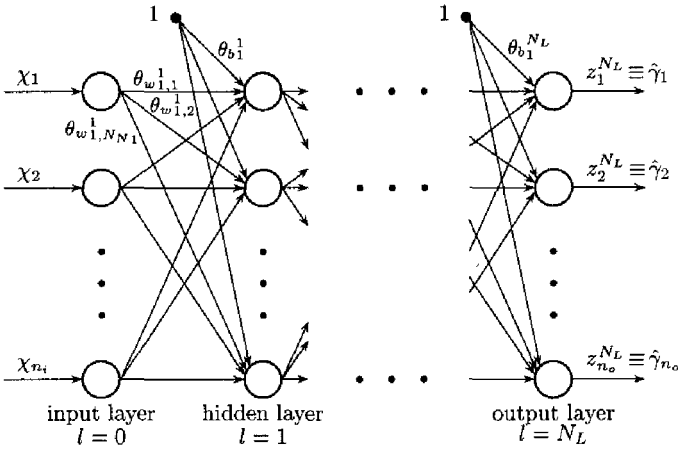


Figure 3.1: The multilayer perceptron (MLP) structure

An MLP can therefore be considered as a function

$$\widehat{\Phi} : I\!R^{n_i} \mapsto I\!R^{n_o}$$

which maps an input space of a vector dimension n_i into an output space of a vector dimension n_o. The input of the MLP is denoted by $\chi \in I\!R^{n_i}$ and the output of the MLP is denoted by $\hat{\gamma} \in I\!R^{n_o}$. Let the ith node in the lth layer of the network compute its output z_i^l according to

$$u_i^l = \sum_{j=1}^{N_{N_{l-1}}} \theta_{w_{ij}}^{l} z_j^{l-1} + \theta_{b_i}^{l} \tag{3.1a}$$

$$z_i^l = s(u_i^l) \tag{3.1b}$$

where $l = 1, 2, \ldots, N_L$ stands for the hidden layer number, $i = 1, 2, \ldots, N_{Nl}$ is the node index of the lth layer, $\theta_{w_{ij}}^{l}$ is a weighting factor of the connection between the jth node of the $(l-1)$th layer and ith node of the lth layer and $\theta_{b_i}^{l}$ is the bias of the ith node of the lth layer. The input of the processing function of the ith node in the lth layer is denoted by u_i^l and the corresponding node output is then denoted by z_i^l.

The input layer, $l = 0$, is a special layer because it provides only the distribution of inputs ξ_i, for $i = 1, \ldots, n_i$, among the nodes of the first hidden layer, $l = 1$. Mathematically it can be seen as

$$z_i^0 = \chi_i, \qquad \text{for } i = 1, 2, \ldots, n_i$$

The variable u_i^l is sometimes called an activation of the node and serves as an input of the node activation function $s(u)$. This function is usually chosen as

$$s(u) = \frac{1}{1 + e^{-u}} \tag{3.2}$$

or

$$s(u) = \tanh(u) = \frac{1 - e^{-2u}}{1 + e^{-2u}} \tag{3.3}$$

For the last hidden layer we often chose a linear processing function

$$s(u) = u \tag{3.4}$$

to allow any range for the MLP outputs $z_i^{N_L}$. The radial basis neural networks use as a processing function

$$s(u) = \exp\left(-\frac{(u - u_m)^2}{2\sigma_u^2}\right) \tag{3.5}$$

where u_m is a mean value and σ_u is a standard deviation, both chosen in advance.

A family of functions $\mathcal{F}$ that can be realized by an MLP is characterized by

1. The number of inputs and outputs n_i, n_o;
2. The number of layers N_L, inclusive the output layer;
3. The number of nodes in hidden layers N_{Nl}, $l = 1, 2, \ldots, N_L - 1$;
4. The set of weights $\theta_{w_{i,j}}^l$ and biases $\theta_{b_i}^l$;
5. The processing function $s(u)$.

We will use a notation $\mathcal{F}_{N_{N1}, N_{N2}, \ldots, N_{NN_L-1}}$ for an MLP with N_{N1}, N_{N2}, ..., N_{NN_L-1} nodes in consecutive hidden layers. We include the weights and biases of a MLP into a long vector $\Theta \in \mathbb{R}^{n_\theta}$ in the following order

```
k := 1
for l := 1 to N_L do
    for i := 1 to N_Nl do
        Θ_k = θ_{b_i}^l
        k := k + 1
        for j := 1 to N_{Nl-1} do
            Θ_k := θ_{w_{i,j}}^l
            k := k + 1
        end
    end
end
```

The question is now, that if we can choose N_L, N_{Nl}, $l = 1, 2, \ldots, N_L - 1$ and Θ freely, what kind of functions can be represented by this neural network. The answer is well known and shortly it is that a multilayer neural network can approximate arbitrarily well any continuous function on any compact set provided that the network contains sufficiently many hidden nodes and the activation function $s(u)$ is continuous, bounded and non-constant. This is a well known result proven by many authors [13, 16, 20, 24, 23, 29], as already indicated in Section 2.5.

A special class of MLPs is an MLP with only one hidden layer or a two layer perceptron $N_L = 2$. It was shown in the literature [13, 16] that such a neural network is sufficient to approximate any continuous function. There is always a question whether it is better to use a MLP with only one hidden layer or with more hidden layers for a particular approximation problem. This question was also noticed in [32]. One can think that by using more hidden layers we might possibly need less nodes to approximate a complex nonlinearity. But, in general, it is difficult to say which structure is better as we also do not know a priori how many nodes do we have to put into the MLP to obtain a certain approximation accuracy.

If we look back to the approximation problem we see that the approximation is generally done by means of a set of training examples or simply by data

$$\mathcal{D} = \{(\chi(1), \gamma(1)), (\chi(2), \gamma(2)), \ldots, (\chi(N), \gamma(N))\}$$

computed by the true function $\gamma(k) = \Phi[\chi(k)]$ for $k = 1, 2, \ldots, N$ and N denotes then the length of the data set $\mathcal{D}$. The pair $(\chi(k), \gamma(k))$ will be also called a pattern, $\chi(k)$ is the input pattern and $\gamma(k)$ is the output pattern. Let us assume that the test inputs $\chi(k)$ are for the moment chosen freely, e.g. at random. A suitable set of weights and biases of the MLP which approximates this function is then found by the so called supervised learning of the neural network. This means, that the output of the network $z^{N_L}(k)$, which also represents the approximated value $\hat{\gamma}(k)$ of the desired output $\gamma(k)$ evaluated for a certain input pattern $\chi(k)$, is compared to the data and the error

$$\varepsilon(k) = \gamma(k) - \hat{\gamma}(k) \tag{3.6}$$

is then used to adjust the weights and biases. More precisely, we define a cost function or an error measure to measures the goodness of the approximation as in Problem 2.2. Let this measure be chosen as a sum of squares of errors (3.6) as follows

$$J(\Theta) = \sum_{k=1}^{N} \bar{J}(k, \Theta) = \frac{1}{2N} \sum_{k=1}^{N} \varepsilon(k)^T \varepsilon(k) \tag{3.7}$$

where $\bar{J}(k, \Theta) = \varepsilon(k)^T \varepsilon(k)$. The function (3.7) is then minimized with respect to the network parameters Θ, inclusive the number of layers N_L and the number of nodes N_{Nl} in each layer l. Let us assume that the network complexity, in terms

of N_L and N_{Nl}, was chosen in advance. A natural way of weights adjustment is the direction of the negative gradient of the error function (3.7), that is

$$\Theta(j+1) = \Theta(j) - \rho_j \frac{\partial J(\Theta)}{\partial \Theta(j)} \tag{3.8}$$

where j is the iteration index, $\frac{\partial J(\Theta)}{\partial \Theta(j)}$ is the gradient of the cost function with respect to the weights and $\rho_j > 0$ is a step size taken at the jth iteration. We usually start the iteration process (3.8) from an initial guess $\Theta(0)$ which is generated randomly. In principle, we have two options how to perform the update (3.8):

1. either we perform the update after each single pattern using as an update direction gradient
$$\frac{\partial \bar{J}(k,\Theta)}{\partial \Theta(k)}$$
2. or we perform the update after collecting errors of a number of patterns, let us say M, computing the gradient as follows
$$\sum_{k=c+1}^{c+M} \frac{\partial \bar{J}(k,\Theta)}{\partial \Theta(k)}$$
where $c = 0, 1, \ldots, M-1$

An update of weights according to the second option is also called a batch learning. If the order of patterns in the data set does not play any role we can be choosing patterns in these two cases at random. As it will become clear later on, this is not always the case, for instance when the data are being produced by a dynamic system and we are interested in a simulation model of the process. Then we have to do the batch learning, often having chosen $M = N$. The weight update (3.8) is iteratively repeated until a sufficient minimum of the cost function is found, usually checked by using testing data, different from training data.

To do be able to perform the iteration process (3.8) efficiently we need analytical expressions of gradients of (3.7) with respect to the weights of the MLP. These are defined as follows

$$\frac{\partial J}{\partial \theta_{wi,j}^{l}} = \frac{1}{N} \sum_{k=1}^{N} \varepsilon(k)^T \frac{\partial \varepsilon(k)}{\partial \theta_{wi,j}^{l}} \tag{3.9}$$

where

$$\frac{\partial \varepsilon(k)}{\partial \theta_{wij}^{l}} = \frac{\partial \varepsilon(k)}{\partial u_i^l(k)} \frac{\partial u_i^l(k)}{\partial \theta_{wij}^{l}} \tag{3.10}$$

Let use denote

$$\delta_i^l(k) = \frac{\partial \varepsilon(k)}{\partial u_i^l(k)} \tag{3.11}$$

and observing from (3.1a) that

$$\frac{\partial u_i^l(k)}{\partial \theta_{wij}^l} = z_j^{l-1}(k)$$

the equation (3.10) can be written in the following form

$$\frac{\partial \varepsilon(k)}{\partial \theta_{wij}^l} = \delta_i^l(k) z_j^{l-1}(k)$$

Using the chain rule the expression (3.11) can be rewritten as follows

$$\delta_i^l(k) = \frac{\partial \varepsilon(k)}{\partial z_i^l(k)} \frac{\partial z_i^l(k)}{\partial u_i^l(k)} = \frac{\partial \varepsilon(k)}{\partial z_i^l} s'(u_i^l(k)) \tag{3.12}$$

where $s'(u)$ is the derivative of the processing function with respect to its argument. It can be easily verified that for (3.2) holds

$$s'(u) = s(u)(1 - s(u)) \tag{3.13}$$

and for (3.3) holds

$$s'(u) = 1 - s(u)^2 \tag{3.14}$$

For an output layer composed of linear units we have

$$z_i^{N_L} = u_i^{N_L} \qquad s'(u) = 1$$

and

$$\delta_i^{N_L}(k) = -1$$

For an output layer composed of nonlinear units, let us assume s(u) given by (3.2), we have

$$z_i^{N_L} = s(u_i^{N_L}) \text{ and } \delta_i^{N_L}(k) = -s(u)(1 - s(u))$$

Observing, that for the lth hidden layer of the MLP, $l < N_L$, holds

$$\frac{\partial \varepsilon(k)}{\partial z_i^l} = \sum_{j=1}^{N_{N\,l+1}} \frac{\partial \varepsilon(k)}{\partial u_j^{l+1}} \frac{\partial u_j^{l+1}}{\partial z_i^l} = \sum_{j=1}^{n_{l+1}} \delta_j^{l+1}(k) w_{ji}^{l+1} \tag{3.15}$$

we can compute (3.12) recursively as follows

$$\delta_i^l(k) = s'\left(u_i^l(k)\right) \sum_{j=1}^{N_{N\,l+1}} \delta_j^{l+1}(k) w_{ji}^{l+1} \tag{3.16}$$

where l is iterated backwards from $l = N_L - 1$ until $l = 1$.

The gradient of the cost function with respect to the MLP's biases is determined in the same way as for weights. We can assume in (3.1a) for the moment that the bias $\theta_{b_i^l}$ is represented by a weight $\theta_{w_{j,0}^l}$ while defining $z_0^l = 1$ and letting the sum in (3.1aa) to run from $j = 0$. Then the gradient of the cost function according to the bias is given by

$$\frac{\partial \varepsilon}{\partial \theta_{b_i^l}} = \delta_i^l(k) \tag{3.17}$$

The variable $\delta_i^l(k)$ represents the error sensitivity of the ith unit in the lth hidden layer for the kth data point. The equation (3.16) defines the back-propagation of this error sensitivity through the network starting at the output layer, and therefore is the algorithm (3.9)–(3.17) called *backpropagation*.

An important issue to keep in mind is that due to the finiteness of the set of training examples is the original function $\gamma = \Phi[\chi]$ tested only on a limited range of the input space of the function Φ, that means $\chi \in \mathcal{X} \subset \mathbb{R}^{n_i}$, which is then mapped into a bounded subset of the output space $\mathcal{Y} = \{\gamma \in \mathbb{R}^{n_o} : \gamma = \Phi[\chi],\ \chi \in \mathcal{X}\}$. As we assume Φ to be a general nonlinear function the extrapolation ability of the MLP can be very poor or, in general, it will be meaningless. With this respect the approximation will always show increasing errors towards the boundary of the input space $\mathcal{X}$, depending on the distribution of χ within this set and the number of weights of the neural network.

3.2 Recurrent feedforward network

By a recurrent feedforward neural network we mean a structure based on a MLP in which some of the outputs are fed back to the input of the network, usually through a number of delays. We do not assume in this structure any recurrent connections inside of the MLP, e.g. between hidden layers. That means that all recurrent connections appear outside of the MLP. Such a recurrent MLP, though it is still static, represents a nonlinear dynamic map which can be used to approximate nonlinear dynamic system, e.g. given by (2.3). How is this done exactly, will be treated in detail in the next chapter. At this moment it is important to realize that training of a recurrent MLP from input/output examples by a minimization of criterion (3.7) becomes more complicated because the error (3.6) does not only depend on the MLP weights Θ but also on those inputs, which stand for past MLP outputs, as these are also a function of Θ.

Therefore the backpropagation algorithm has to be revised. This revision is incorporated through a correction term added to the backpropagation formula (3.10) as follows

$$\frac{\mathrm{d}\varepsilon(k)}{\mathrm{d}\theta_i} = \frac{\partial \varepsilon(k)}{\partial \theta_i} + \sum_j \frac{\partial \varepsilon(k)}{\partial \chi_j(k)} \frac{\partial \chi_j(k)}{\partial \theta_i} \tag{3.18}$$

where the index j runs through those inputs, which are function of the past network outputs. The last term has to be determined with respect to the actual recurrent feedback configurations. To compute

$$\frac{\partial \varepsilon(k)}{\partial \chi_j(k)}$$

we can still use the backpropagation algorithm. Observe that $\chi \equiv z^0$ so that from (3.15) we can see that by performing one extra backpropagation step we obtain required partial derivatives.

3.3 Iterative MLP learning algorithms

In the previous section we have seen that training of a MLP is in fact a minimization of criterion (3.7). This is nothing else then a general unconstrained nonlinear optimization problem. As it is not a trivial to solve problem we will pay in the following more attention to it.

First of all, we have to notice, that the minimization of (3.7) with respect to the weights of a MLP can not be done analytically. Therefore it must be done by an iterative numerical procedure. Actually, we have already introduced one such an iterative procedure and that was the backpropagation algorithm (3.9)–(3.17) together with the weights update (3.8).

A neural network learning problem is in fact nothing else than a nonlinear optimization problem which can be stated as follows

Problem 3.1. Given a real-valued function $J : \mathbb{R}^{n_\theta} \to \mathbb{R}$ defined on a set $\mathbb{R}^{n_\theta}$ and a bounded subset $\mathcal{C} \subset \mathbb{R}^{n_\theta}$, by the optimization problem

$$\begin{aligned} &\text{minimize} && J(\Theta) \\ &\text{subject to} && \Theta \in \mathcal{C} \end{aligned} \tag{3.19}$$

we mean a problem of finding an element $\Theta^* \in \mathcal{C}$ such that

$$J(\Theta^*) \leq J(\Theta), \quad \text{for all } \Theta \in \mathcal{C} \tag{3.20}$$

Such a Θ^* we call a globally optimal solution or simply a global minimum. The existence of such an element is in fact guaranteed by the boundedness of $\mathcal{C}$. When $\mathcal{C}$ is not finite then existence of a minimizing point is only guaranteed if $J : \mathbb{R}^{n_\theta} \to \mathbb{R}$ is a continuous function, $\mathcal{C} = \mathbb{R}^{n_\theta}$, and $J(\Theta) \to +\infty$ if $\|\Theta\| \to +\infty$.

Necessary and sufficient conditions for optimality are readily available when J is a differentiable function on $\mathbb{R}^{n_\theta}$ and $\mathcal{C}$ is a convex subset of $\mathbb{R}^{n_\theta}$. Then we have $\nabla J(\Theta^*)^T(\Theta - \Theta^*) \geq 0$, for all $\Theta \in \mathcal{C}$ where $\nabla J(\Theta^*)$ is the gradient of J evaluated at Θ^*

$$\nabla J(\Theta^*) = \left(\frac{\partial J(\Theta)}{\partial \theta_1}, \frac{\partial J(\Theta)}{\partial \theta_1}, \ldots, \frac{\partial J(\Theta)}{\partial \theta_{n_\theta}} \right)^T \Bigg|_{\Theta=\Theta^*}$$

In case $\mathcal{C} = \mathbb{R}^{n_\theta}$ (unconstrained case) this is equivalent with $\nabla J(\Theta^*) = 0$. When J is in addition twice continuously differentiable and $\mathcal{C} = \mathbb{R}^{n_\theta}$, an additional necessary condition is that the Hessian matrix $\nabla^2 J(\Theta^*)$ be positive definite at Θ^*. If J is a non-convex function of Θ, i.e. the condition $J[\alpha\Theta_1 + (1-\alpha)\Theta_2] \leq \alpha J(\Theta_1) + (1-\alpha)J(\Theta_2)$ is not satisfied for every $\Theta_1, \Theta_2 \in \mathcal{C}$ and every scalar α, $0 \leq \alpha \leq 1$ then the above stated conditions of optimality have only local character for some neighbourhood of Θ^*.

When the function J is defined by the MLP it becomes non-convex. This creates a serious obstacle to finding a global solution of the optimization problem Problem 3.1. The parameter dimension is usually in the order of a few tens or even hundreds what further complicates the problem.

A method of finding a minimizer Θ^* of the function J is then called an optimization method. These methods perform a certain iteration process on Θ, similar to (3.8). Such an iteration process can be based either on a gradient of the optimized function or the updates can be generated randomly. In the following sections we will discuss both of these approaches.

3.4 Gradient optimization

In the following we consider search methods which use the gradient of the minimized function as well as the function values. We will concentrate on a feasibility of using these methods for neural network training.

3.4.1 The method of steepest descent

One of the reasons, why neural networks became so popular was the promotion of the backpropagation training algorithm [74, 55, 21]. Purely seen from the mathematical point of view it is an algorithm of analytical evaluation of gradients of the MLP error function with respect to weights and then application of this gradient in a steepest descent minimization procedure for an iterative update of weights.

The negative gradient $-\nabla J(\Theta(j))$ in (3.8) shows the direction of the most rapid decrease of the function at the point $\Theta(j)$. This is where the name of this method, steepest descent, comes from. The step size ρ_j determines the speed of convergence of this method or in another words the learning rate of the network. Under certain conditions this is a globally convergent method converging to a local minimum of the cost function J. If ρ_j is too small the method converges very slowly. On the other hand, if ρ_j is too big, the method starts to oscillate. The choice of a constant $\rho_j = \rho$ causes also oscillations of the method near a local minimum. Often is (3.8) combined with a one dimensional search or line optimization as follows

$$\rho_j^* = \arg\min_{\rho_j} \quad J\Big(\Theta(j) - \rho_j \nabla J(\Theta(j))\Big) \tag{3.21}$$

with respect to $\rho_j \in \mathbb{R}$. The actual update (3.8) uses then a value ρ_j^* obtained as a result of this optimization problem. An analysis shows, that we then iterate

our search along orthogonal increments. This feature, in fact, makes the steepest descent method slow and computationally inefficient. This inefficiency becomes even more obvious if the dimensionality of Θ becomes very large which is the case of neural network training.

Many suggestions are reported in the literature of speeding up the backpropagation training algorithm. All these improvements are based on some heuristics which are closely related to the application and therefore it is hardly possible to generalize these results. However, they can be found useful for large problems being solved on small machines with a limited memory when the steepest descent algorithm is the only one applicable.

A most common improvement of (3.8) concerns the extension of this recurrence in an extra, so called, momentum term [55] giving us the following iteration process

$$\begin{aligned}\Theta(1) &= \Theta(0) - \rho_0 \nabla J(\Theta(0)) \\ \Theta(j+1) &= \Theta(j) - \rho_j \nabla J(\Theta(j)) + \beta_j(\Theta(j) - \Theta(j-1)), \quad j \geq 1\end{aligned}$$

A complete mathematical proof of a global convergence of this interesting algorithm together with conditions put on ρ_j and β_j can be found in [42, 49]. Intuitively, the benefit of this method lies in a sort of filtering out zigzag changes of the gradient along steep valleys. It can potentially also escape from shallow local minimum, because when $\nabla J(\Theta(j))$ becomes zero, $\Theta(j) - \Theta(j-1)$ probably was not. The main drawback of this method lies in a reliable choice of steps ρ_j and β_j which could provide a good convergence speed.

Other improvements of the algorithm (3.8) concerns an adaptive choice of the step size ρ_j during the learning process by monitoring the speed of the descent progress [59, 52, 26, 73]. Our experience and tests of some of these methods show that the error functions formed by neural networks together with data are much too complex that we could benefit out of these methods. Another weak point of all these proposals is that they were not tested on more examples and compared to other methods, for instance on the same data.

Another often seen modification of the backpropagation algorithm is an extension of the basic cost function into a term penalizing superfluous weights [30, 60]. This is expressed by an Euclidean norm put on weights resulting in the following criterion

$$\tilde{J}(\Theta) = \frac{1}{2}\sum_{k=1}^{N} \varepsilon(k)^T \varepsilon(k) + \varrho \frac{1}{2}\sum_{i=1}^{n_\theta} \theta_i^2 \tag{3.23}$$

where $\varrho \in I\!R$, $\varrho > 0$ is a so called regularization parameter. We can immediately notice, that in this way we penalize also nonzero weights and therefore we introduce here some bias with respect to the optimal solution. The gradient of (3.23) is readily available as a combination of the backpropagation gradient and an extra term as follows

$$\nabla \tilde{J}(\Theta) = \nabla J(\Theta) + \varrho\Theta$$

We have found this method useful for early stages of any optimization to penalize weights with very big values which might cause node saturations and consequently numerical problems in optimization. The regularization term in fact pre-conditions the Hessian of the cost function and in this way makes the optimization easier.

3.4.2 Conjugate gradient optimization methods

A method which outperforms the steepest descent algorithm and has a precise mathematical interpretation is the method of conjugate gradients proposed originally in [22] and then reconsidered for a non-convex optimization in [15]. This method is already close to the second order gradient optimization techniques which are subject of the next section. The algorithm assumes that the optimized function is quadratic and implicitly uses the Hessian matrix in its derivation. In the iteration process is this matrix actually not updated and therefore this method is also called as memoryless quasi-Newton.

The search direction $p(k)$ is in this method generated as follows

$$p(k) = -\nabla J(\Theta(k)) + \beta(k)p(k-1)$$

where $\beta(k)$ is computed by different formulas (see e.g. [56])

$$\beta(k) = \frac{\nabla J(\Theta(k))^T \nabla J(\Theta(k))}{\nabla J(\Theta(k-1))^T \nabla J(\Theta(k-1))}$$

or

$$\beta(k) = \frac{(\nabla J(\Theta(k-1)) - \nabla J(\Theta(k)))^T \nabla J(\Theta(k))}{\nabla J(\Theta(k-1))^T \nabla J(\Theta(k-1))}$$

For quadratic functions this method finds the minimum in a finite number of steps. When this method is used for optimization of non-quadratic functions the search direction is periodically re-initialized to the steepest descent direction by choosing $\beta(k) = 0$ for $k = 0, n_\theta, 2n_\theta, \ldots$. We have experienced that when using this method for neural network training, after a few iterations the method gets stuck and as a consequence the conjugate direction generation has to be restarted more often than after every n_θ iterations. To explain this assume that $\Theta(k+1) \approx \Theta(k)$. As for this method holds that $p(k) \perp \nabla J(\Theta(k+1))$ and $\nabla J(\Theta(k+1)) \perp \nabla J(\Theta(k))$, we can easily verify, that the search direction becomes almost orthogonal to the gradient direction, where marginal improvement can be expected. For this reason at least the second formula for $\beta(k)$ should be used, which automatically "resets" the search direction to the steepest descent one when this occurs. But then we often proceed the optimization mainly in the inefficient steepest descent steps.

Some modification of the basic algorithm of conjugate gradients are discussed in [46], [57]. Mainly the problem of inexact line searches addressed in [57] could be of particular interest here, as the line search is very crucial for maintaining the mutual conjugacy of the search directions. As we are dealing with complex nonlinear functions benefits of these improvements might be masked by the problem complexity.

3.4.3 Second-order gradient optimization methods

It turns out, that for some applications, mainly concerning pattern recognition problems, is the backpropagation learning method sufficient. For system identification for control purposes, where accurate approximations are natural this is not enough at all. It is well known, that for quadratic functions is the steepest descent not the best search direction. This is due to the curvature of the cost function, which is not taken into the account or in other words the steepest descent does not considers the successive gradient changes of the cost function. These changes can be brought into the optimization implicitly or explicitly. In both approaches we relay on a strictly quadratic landscape of the optimized function with respect to optimized parameters, in our case these are the weights of the neural network Θ. Then we reach the global minimum within at most n steps if n is the searched space dimension. The error landscapes produced by neural networks, even they are quadratic in terms of the error ϵ, are very complicated in shape. They are non-convex functions of its parameters Θ and often resulting in an ill-conditioned optimization problem. By ill-conditioning we mean, that the spectrum of eigenvalues of the Hessian can be very wide at different points of $J(\Theta)$.

The current state-of-the-art in non-convex optimization based on quadratic approximation of the optimized function is the well known Broyden-Fletcher-Goldfalb-Shanno (BFGS) quasi-Newton method [58]. This method starts with an initial guess $\Theta(0)$ and initial positive definite matrix Q_0 and iterates:

- a line search (3.21) in $\rho_j > 0$ for

$$\Theta(j+1) = \Theta(j) - \rho_j Q_j^{-1} \nabla J(\Theta(j))$$

- and an update

$$Q(j+1) = Q(j) - \frac{Q(j)p(j)d(j)^T + d(j)p(j)^T Q(j)}{d(j)^T p(j)} + \left(1 + \frac{p(j)^T Q(j) p(j)}{d(j)^T p(j)}\right) \frac{d(j)d(j)^T}{d(j)^T p(j)}$$

where $d(j) = \Theta(j+1) - \Theta(j)$ and $p(j) = \nabla J(\Theta(j+1)) - \nabla J(\Theta(j))$.

Similar method to BFGS is an older method of Davidon-Fletcher-Powell (DFP) method [7]. This method updates Q^{-1} and therefore gives the search direction directly while the BFGS must solve a linear system of equations. But test examples shows that the DFP method has a tendency to produce a sequence of matrices Q_j which are not positive definite because of computer round-off errors. The numerical stability of BFGS method is usually further increased by updating Cholesky factors of the approximated Hessian Q_j.

The main advantage of quasi-Newton methods is in their fast convergence close to the minimum. When used for non-quadratic functions they still show superlinear rate of convergence in a neighbourhood of a non-singular minimum point. The

main drawback of these methods is in the need to store and update an $n_\theta \times n_\theta$ matrix Q_j which can be significant for large sets of optimized parameters. Basically, these methods do not handle situations when the true Hessian matrix of J becomes singular or negative definite. In such a situations we have to switch to other optimization methods.

A method specifically designed for minimizing a sum-of-squares error is the Levenberg-Marquardt algorithm. This method balances a Newton update and a standard gradient descent. However, our tests did not show a clear superiority of this method with respect to the quasi-Newton method.

3.4.4 Optimization by optimal filtering techniques

The problem of weights adjustment can be transformed into a problem of the state estimation of a nonlinear dynamic system from noisy data using optimal filtering techniques. The estimate can be done by the extended Kalman filter [31, 61].

Let the weights of the neural network constitute a state of the following discrete-time nonlinear dynamic system

$$\begin{aligned} \Theta(k+1) &= \Theta(k) && \text{(3.24a)} \\ \gamma(k) &= \widehat{\Phi}(\chi(k), \Theta(k)) + v(k) && \text{(3.24b)} \end{aligned}$$

where $\gamma(k)$ is the given output pattern and $\hat{\gamma}(k) = \widehat{\Phi}(\chi(k), \Theta(k))$ is the output of the neural network at time instant k and ξ is the neural network input pattern. The vector $v(k) \in \mathbb{R}^{n_o}$ is assumed to be a Gaussian white noise. Let

$$\begin{aligned} \mathcal{E}\{v(k)\} &= 0 \\ \mathcal{E}\{v(k)v(j)^T\} &= R_k \delta_{kj} \end{aligned}$$

where δ_{kj} is the Dirac function, $\mathcal{E}\{\cdot\}$ denotes the expectation operation and R_k is a positive definite covariance matrix of the noises $v(k)$.

To apply the extended Kalman filter on the nonlinear dynamic system (3.24), we linearize the nonlinearity in the output equation (3.24b) around the current estimate of the state vector, which is in fact the current estimate of weights of the neural network. Then the filter equations are

$$\begin{aligned} \hat{\Theta}(k+1) &= \hat{\Theta}(k) + K(k)[\gamma(k) - \widehat{\Phi}(\chi(k), \hat{\Theta}(k))] && \text{(3.25a)} \\ P(k+1) &= P(k) - K(k)\, H(k)\, P(k) && \text{(3.25b)} \\ K(k) &= P(k)\, H(k)^T\, [R(k) + H(k)\, P(k)\, H(k)^T]^{-1} && \text{(3.25c)} \end{aligned}$$

where

$$H(k) = \left. \frac{\partial \widehat{\Phi}(\chi(k), \Theta(k))}{\partial \chi(k)} \right|_{\Theta(k)=\hat{\Theta}(k)}$$

The equation (3.25a) defines the weights update and the equation (3.25b) defines the weights covariance matrix $P(k)$ update, both are being updated after presenting a single input pattern $\chi(k)$ of the data to the neural network input. Notice, that $P(k) \in \mathbb{R}^{n_\theta} \times \mathbb{R}^{n_\theta}$ is a square matrix of dimension of Θ and to compute the Kalman gain $K(k)$ we have to invert a matrix of dimension n_o, which is the number of neural network outputs.

This method was tested on simulated data from the following system

$$y_d(k+1) = \frac{y_d(k)y_d(k-1)(1-u(k-1))+u(k)}{1+y_d(k-1)^2+y_d(k-2)^2}$$
$$y(k) = y_d(k) + e(k) \qquad e(k) \sim \mathcal{N}(0, 0.01)$$

Using as an input $u(k)$ a sequence of uniformly distributed random samples from an interval $\langle -1, 1\rangle$ we generated a sequence of 500 output points $y(k)$ produced by this system. A neural network was then let to approximate the nonlinearity of this process. To construct an approximation problem, we defined the neural network input as follows

$$\chi(k) = (u(k-1), u(k-2), y(k-1), y(k-2))^T$$

The neural network was a MLP with one hidden layer with 10 nodes, 4 inputs and 1 output computing $\hat{\gamma}$. The true output pattern $\gamma(k)$ was represented by the simulated system output $y(k)$, $\gamma(k) = y(k)$. This example was taken from another study where we were comparing prediction and simulation models. At this point we want to demonstrate the performance of the extended Kalman filter used for neural network training regardless obtaining a biased estimate of the system transfer function. In Figure 3.2 is shown the cost function value against the iteration number. We can immediately recognize the superiority of the EKF against the backpropagation. The slow convergence of the backpropagation algorithm is quite remarkable. As the EKF is computationally more involved its advantage in cost reduction speed is weakened by this problem. However, the overall CPU time spent in 5000 iterations of the backpropagation was 3799 seconds and reached cost function value was 0.241, while 200 iterations of the EKF took 2354 seconds of CPU and the reached cost function value was 0.0255.

3.4.5 *Other gradient optimization methods*

When we have tested the quasi-Newton optimization method for its suitability for neural network training we have observed a couple of convergence difficulties. The main convergence difficulty was related to a situation when the Hessian matrix of the optimized function has negative eigenvalues or is almost singular. The landscape of the optimized function has there eccentric curved valleys or flat ravines. Since the method uses a quadratic approximation of the optimized function, which can happened to be very poor resulting in slow convergence rate of the optimization. To overcome these problems we have tested different optimization methods.

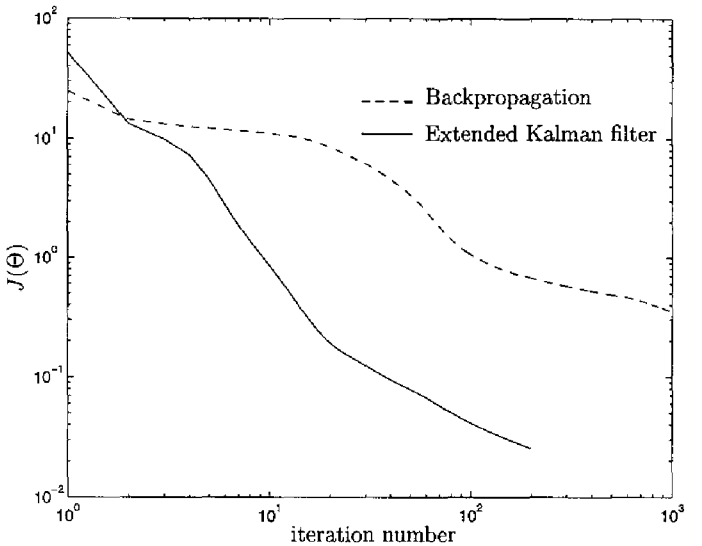


Figure 3.2: EKF versus backpropagation neural network training

In the following we give a review of some of these methods as they can be used in combination with the quasi-Newton optimization.

Except the quasi-Newton method, these methods are usually not included in currently available software packages and must be programmed separately.

Optimization by weighted past gradients

In this section we discuss an optimization method which we proposed when tackling the problem of following steep, curved, high dimensional valleys of $J(\Theta)$. The steepest descent method will be getting stuck in such a valley after a few iterations by taking small orthogonal steps and the quasi-Newton method will be affected by not strictly positive definite Hessian of $J(\theta)$. In these cases the optimization is proceeded by small orthogonal increments resulting in jumping from side to side of a multidimensional valley.

The defined by the following formulas

$$\begin{aligned} \Theta(k+1) &= \Theta(k) + Q(k, \lambda^*) \\ Q(k, \lambda^*) &= \sum_{i=k-q}^{k} \lambda_i^* \nabla J(\Theta(i)) \end{aligned}$$

where k is the iteration number, $Q(k, \lambda^*)$ is a weighted sum of past q gradients

inclusive the current one and $\lambda^* \in \mathbb{R}^{q+1}$. The weighting factors λ_i^* are chosen at each iteration by solving the following optimization problem

$$\lambda^* = \arg\min_{\lambda} J(\Theta(k) + Q(k, \lambda))$$

The last optimization problem can be easily solved by a quasi-Newton algorithm. Compared to the quasi-Newton method used to minimize $J(\Theta)$, The convergence rate of this method was only better than the convergence rate of the quasi-Newton method only in the earlier stages of the optimization. It seems that the dimensionality of the valley can be so high that it is impossible to follow the valley curvature only by using a few past gradients.

However, the main advantage of this method remains in the reduction of the searched parameter space dimension compared to the quasi-Newton optimization method as $q \ll n_\theta$.

Optimization along curved lines

The reason to investigate the "curved search methods" for training of the neural network is their potential improvement of convergence on error landscapes featured by curved valleys. The method derived in [3] performs a one dimensional search along a quadratic curve

$$\Theta(k+1) = \Theta(k) + \alpha_k p(k) + \frac{1}{2}\alpha_k^2 q(k) \tag{3.26}$$

where $q(k)$ is the steepest descent direction and $p(k)$ is the quasi-Newton direction. We have experienced occasionally a faster convergence using this type of methods but in longer terms these methods did not outperform the quasi-Newton method. The reason for this may be that the curvature of the valley is so complex that the quadratic curve can only fit this curvature in a very small neighbourhood of the current point. In fact, our implementation of this method was not optimal as the Hessian matrix was only estimated numerically.

Optimization by nonlinear coordinates transformation

Another class of gradient methods which considers non-quadratic shape of optimized cost function are described in [53, 41]. As we have already mentioned, using second order gradient optimization methods, like the quasi-Newton, usually results in a poor performance because the supposed quadratic approximation of the optimized function does not describe the behaviour of the optimized function accurately, e.g. when the Hessian matrix of $J(\Theta)$ has negative eigenvalues. In such cases it is interesting to use a nonlinear transformation of coordinates $\Theta = \psi(\bar{\Theta})$. The Hessian matrix in the transformed coordinates is given by

$$\nabla^2 J(\psi(\bar{\Theta})) = \nabla\psi^T(\bar{\Theta})\nabla^2 J(\Theta)\nabla\psi(\bar{\Theta}) + \sum_{i=1}^{n_\theta} \nabla^2\psi_i(\bar{\Theta})\nabla J(\Theta) \tag{3.27}$$

and can be made positive definite. This means that in transformed coordinates the approximation of the optimized function by some quadratic form is more accurate than in the original coordinates.

For the nonlinear transformation of coordinates we took the form proposed in [53]. We implemented this method using numerical estimation of required gradients and Hessian matrix components. The problem with this method is that to construct a suitable coordinate transformation function we have to know the sensitivity of the eigenvalues of the Hessian matrix with respect to optimized parameters and that requires determination of third derivatives. Just for curiosity, this method finds the minimum of the well known Rosenbrock's "banana" function in the second iteration.

3.5 Stochastic optimization

This broad class of optimization techniques is very well suited for non-convex optimization problems with many local minima. We can distinguish deterministic and stochastic global optimization methods. The deterministic optimization methods try to locate all local minima and then choose the best one as a global minimum. However, there is no test available for a general non-convex function to verify whether there exists another local minimum other than already found. Therefore methods like [11] won't work in practice unless we can exploit the optimized function analytically. Even if our problem is a global optimization problem on a bounded set the covering methods [71], working with certain grids, are not interesting for us because of very large grid points to be considered even for problems of moderate dimensions. On top of this, these methods assume limited rate of change of the optimized function given by the Lipschitz constant which is in practice also very hard to find.

Later on we rather concentrate on stochastic methods which seem to give better results [70]. These methods usually guarantee a convergence to a global minimum in a probabilistic sense as the number of trials increases

$$Pr\{\lim_{k\to\infty} \Theta(k) = \Theta^*\} = 1 \tag{3.28}$$

where Θ^* is a global optimizer and Pr is the probability operator. However, in practice we do sacrifice the possibility of global convergence otherwise such a method would be found lacking efficiency. At the same time we are loosing the reliability of these methods and therefore some trade-off between these two issues is always necessary.

A basic algorithm describing this class of methods is as follows

1. Choose an initial point Θ_0, step size α_0, set $k = 0$, and $\Theta(0) = \Theta_0$.
2. Generate a search trial $p(k) \in \mathbb{R}^{n_\theta}$ at random
3. If $J(\Theta(k) + \alpha_k p(k)) < J(\Theta(k))$ update $\Theta(k+1) = \Theta(k) + \alpha_k p(k)$

4. set $k = k + 1$ and go to step 2.

Different methods differ namely in steps 2 and 3 of this scheme, that is how to generate a search trial and how to perform the update or to choose the step size α_k. To satisfy (3.28) α_k must obey the following conditions

$$(i) \quad \sum_{k=0}^{\infty} \alpha_k = \infty \qquad (ii) \quad \sum_{k=0}^{\infty} \alpha_k^2 < \infty \qquad (iii) \quad \lim_{k \to \infty} \alpha_k = 0$$

These are simple conditions to fulfill theoretically, e.g. by a choice $\alpha_k = \frac{1}{k}$. An obvious problem here is that computationally we can perform only a finite number of iterations. However, for simple functions of modest dimension of Θ, say $n_\theta < 10$, which can be evaluated in a relatively short time and we usually can perform a sufficient number of iterations, these conditions are practically satisfied. For functions of high dimensionality, say $n_\theta > 50$, this is a real limitation. In the next part we will review those stochastic search methods which we were found very useful for neural network training and which also showed better efficiency than the basic method discussed just above.

3.5.1 Controlled random search

Controlled random search methods are characterized by controlling some of the parameters of the generating probability density function (p.d.f.) of search trials $p(k)$. Most of the time, a trial is chosen from a Normal p.d.f. which variance is controlled with respect to the progress in optimization. The method proposed by [27] was used by many researchers in different modifications. For instance, in [63] is in the case of success the variance increased by a constant ratio and in the case of failure decreased by a constant ratio. In [69] is proposed a controlled random search strategy which belongs to a class of reinforcement search algorithms. It means that a successful trial is used also in the future to adjust the search parameters as there is a good chance that the same trial will also appear in the future.

The algorithm of [69] generates trials $\overline{\Theta}$ by

$$\overline{\Theta}(k) = \Theta(k) + p(k) \tag{3.29}$$

where $p(k) \in \mathbb{R}^{n_\theta}$ is a vector of stochastic increments which are chosen from an uniform distribution on an interval $\langle -\sigma_k, \sigma_k \rangle$, $\sigma_k \in \mathbb{R}^{n_\theta}$. If a trial is successful, that is $J(\overline{\Theta}(k)) < J(\Theta(k))$, we perform the following update

$$\Theta(k+1) = \overline{\Theta}(k) \tag{3.30a}$$

$$\sigma_{k+1} = \alpha\sigma_k + (1-\alpha)\frac{1}{2P_a}|p(k)| \tag{3.30b}$$

where P_a is a user defined parameter specifying the probability of success. If a trial is not successful, that is $J(\overline{\Theta}(k)) \geq J(\Theta(k))$, we perform the following update

$$\Theta(k+1) = \Theta(k) \tag{3.31}$$

$$\sigma_{k+1} = \alpha\sigma_k \qquad 0 < \sigma < 1 \tag{3.32}$$

The decay factor α defines the speed of the algorithm convergence and typical values of this factor are 0.9, 0.99. The coefficient P_a defines expected probability of a successful trial. A typical value for P_a is from interval $\langle 0.1, 0.01 \rangle$. It is claimed in [69] that the parameter ranges will stabilize to effective values. Actually, this will only happen when an expectation of a successful trial will approach value P_a. As this is seldom the case, unless we choose a very small value for P_a, the searched parameter ranges converge exponentially to zero. We have used this method mainly due to its simplicity and almost no computational overhead.

3.5.2 Simulated annealing

Simulated annealing is a stochastic optimization technique that can optimize any cost function possessing arbitrary degree of nonlinearity, discontinuities or stochasticity, including arbitrary constraints imposed on these cost functions. From the statistical point of view it guarantees finding a globally optimal solution. Simulated annealing is an optimization procedure which probabilistically samples different points of the function landscape, called also energy function. This method maintains a parameter called the temperature. As the temperature is reduced the likelihood that lower local minima are sampled rather than higher ones increases. Finally, when the temperature is at zero the global minimum is found.

The method of simulated annealing consists of three functional relationships:

1. The generating probability density function $f_a(\Theta)$ of the parameter space.
2. The probability function P_a for acceptance of a new cost function given just the previous value.
3. A schedule $T(k)$ of annealing the temperature T in annealing-time steps k.

Basically, the method generates random trials with a probability density function $f_a(\Theta)$ and successful trials represent new points. The key feature of the simulated annealing optimization is that a not successful trial is treated probabilistically: the probability that this trial is accepted is P_a. The standard simulated annealing optimization, generally specified as Boltzman annealing, uses as the acceptance probability a function, which is based on changes in the cost function value in two successive trials,

$$P_a(\Delta J) = \exp(-\Delta J/T) \tag{3.33}$$

where ΔJ represents the difference between the present and previous values of the cost function, i.e. $\Delta J = J(k+1) - J(k)$.

As a generating function $f_a(\Theta)$, the Boltzman annealing uses a Gaussian p.d.f.

$$f_a(\theta_i) = (2\pi T)^{-1/2} \exp(-\Delta\theta_i^2/(2T)) \tag{3.34}$$

where $\Delta\theta_i = \theta_i(k+1) - \theta_i(k)$ is a single parameter deviation of $\Theta(k+1)$ from the currently accepted point $\Theta(k)$ and T represents the temperature in the system. In [18] has been shown that for the Boltzman annealing with the generating

function (3.34) the optimization procedure can find a global minimum of $J(\Theta)$ if the annealing temperature is reduced at the rate of

$$T(k) = \frac{T_0}{\ln k}$$

or slower. As we can see from the last equation, the logarithmic decrease of the temperature will in general lead to a very slow optimization.

A faster annealing schedule can be obtained using a Cauchy distribution as the generating probability density function of the parameter space (see [67]). The Cauchy distribution given by

$$f_a(\theta_i) = \frac{T}{(\Delta\theta_i^2 + T^2)^{(n_\theta+1)/2}} \tag{3.35}$$

has a fatter tail than the Gaussian distribution of the Boltzman annealing and this permits easier access to test local minima in the coarse of the search. To guarantee that the system will statistically find the global minimum, the annealing schedule for Cauchy distribution is

$$T(k) = \frac{T_0}{k}.$$

Both the Boltzman annealing and the Cauchy annealing have the distribution functions, which sample infinite ranges and there is no provision for considering different annealing schedules for different parameters. It would be also convenient to sample a bounded search space rather than an infinite space. Also, there is no quick algorithm for calculating a n_θ-dimensional Cauchy random generator. One might choose a n_θ-product of one-dimensional Cauchy distributions for which a few quick algorithms exists. This could also permit different T_0's to take into account different parameter sensitivities. The required annealing schedule looks in this case as

$$T_i(k) = \frac{T_0}{k^{1/n}}$$

which, although faster than Boltzman annealing, is still quite slow. This sort of annealing was proposed in [25]. Though this is a very sophisticated version of simulated annealing, it introduces relatively high computational overhead which may not allow us to perform sufficient number of cost function evaluations within a reasonable time.

Less computational overhead is introduced in the simulated annealing of [12]. This method uses (3.33) as the acceptance probability function and an uniform probability density function of the parameter space. During the optimization are the ranges of parameters adaptively adjusted for each parameter dimension independently such that the averaged percentage of accepted moves is about one-half of the total moves. Basically, the parameter range is extended if the success rate is too high or decreased if the success rate is too small.

3.6 Summary and conclusions

While the problem of a neural network training belongs to the class of non-convex optimization problems we have to consider it as a global optimization problem.

In principle, we have two options for a choice of an optimization method and that is either a deterministic or a stochastic optimization method. The deterministic methods try to solve this problem by locating all local minima. No such method, however, can guarantee that all local minima will be found for a general non-convex function. Far better results - both theoretically and computationally - are obtained by stochastic methods. We have seen that these methods are reliable under mild conditions, that is they converge almost certainly to the global minimum. However, a strictly global method is usually found lacking in efficiency. Therefore we do sacrifice the possibility of an absolute guarantee of the global minimum and we only expect to find a good minimum.

The advantages and disadvantages of local and global optimization algorithms are judged mainly by the number of iterations needed to solve the problem and computer time needed to complete this task. Some methods are very sophisticated and may need fewer iterations than simpler methods. On the other hand, more complex methods need more computer time per iteration but after all, the practice shows that increased computational costs are always compensated by substantially fewer number of iterations required than for simple methods. This is the case of the quasi-Newton method compared to conjugate gradients of Fletcher-Reeves or weighted past gradients proposed in this chapter. Local optimizations methods usually converge very fast but only to a local minimum and are very sensitive to the shape of the energy function. Global optimization methods are insensitive to the shape of the energy function and converge to a global minimum but very slowly and theoretically in infinite time. As in many practical problems, a trade-off between complexity and reliability of overall optimization will take place.

Our approach to neural network training is based on a combination of deterministic and stochastic methods resulting in the following algorithm:

```
for j := 1 to N_Θ do
    set Θ_j(0) to either random or user-specified values
    call stochastic search routine
    call quasi-Newton routine
end
```

whrere $\Theta_j(0)$ is a starting point of the optimization. That means that we iterate a stochastic search followed by a quasi-Newton optimization for a number of starting points N_Θ until sufficient performance is obtained which is judged manually. The stochastic search is either the controlled random search algorithm using formulas (3.30) or a Boltzman simulated annealing of [12]. The quasi-Newton routine starts then from the best point found by the stochastic search. The returned minimum is stored and later on, after a whole batch of N_Θ optimizations was completed we decide either to restart them, possibly with replacement of worst solutions with

new random guesses, or we accept the best solution. This decision process was not automated yet and was done manually. The number of iterations performed either by a stochastic search or by the quasi-Newton search are specified in advance as input parameters.

A C code of the simulated annealing of [25] we obtained from the author himself. A FORTRAN code of the simulated annealing of [12], which is publicly available on the INTERNET computer network, was re-programmed in C. The other stochastic search routines were programmed in C. As a quasi-Newton routine we were using FORTRAN routines E04KBF, E04JBF, E04UCF of the NAG library [47].

4 Grey-Box Neural Network Models

This chapter deals with the modelling issues involved in the controller design procedure discussed in the next chapter. Provided, that there is available a mathematical simulation model of the process we can test different control strategies and synthesize such a strategy which is the best with respect to our requirements. Therefore, to have a feasible simulation model of the process *is of importance.* We consider only parametric models which means that the model is a mathematical function of a finite number of tunable parameters which allow the model to compute a specific function.

Neural networks offer an excellent approximation possibility and are recently often used to approximate functions that define the plant input/output dynamics. The main feature of our approach is that the process dynamics is modeled by embedding as much available a priori knowledge about the process dynamics into the neural network model as possible.

4.1 Black-box modelling

The general idea behind black-box modelling is to assume measured input/output data of the given system, usually assumed in a form of multivariate time series

$$\mathcal{D} = \{[u(k), y(k)],\ u \in \mathbb{R}^m,\ y \in \mathbb{R}^p,\ k = 1, 2, \ldots, N\} \tag{4.1}$$

where u is the input and y is the output and to approximate the output of the system $y(k)$ by a relationship $\hat{y}(k) = \hat{f}(\phi, \Theta)$ where ϕ is the usual regression vector and Θ denotes the parameters of the model. In our context, the function $\hat{f}$ is in our context approximated by an MLP and the regression vector is typically composed from either past inputs and past outputs. The values of parameters Θ are then typically obtained by minimizing a summed squared error between the true system outputs $y(k)$ and the modelled output $\hat{y}(k)$.

In the following we will be treating two classes of dynamic black-box models: input/output models and state-space models. Each of these models will implicitely assume a certain structure of the modeled system.

4.1.1 I/O models

Consider the following prediction form of a nonlinear input/output dynamic system in the discrete-time domain

$$y(k) = f_p[y(k-1), \ldots, y(k-n_y), u(k), \ldots, u(k-n_u)] + e(k) \tag{4.2}$$

where $u(k) \in \mathbb{R}^m$ and $y(k) \in \mathbb{R}^p$ are observed inputs and outputs, respectively, $k \in \mathbb{Z}^+$ is the discrete-time index and $e(k) \in \mathbb{R}^p$ is a noise sequence of mutually independent identically distributed random samples which are independent of inputs $u(k)$ and outputs $y(k)$. The order of this system is defined by the number of output delays $n_y \in \mathbb{R}^m$ and by the number of input delays $n_u \in \mathbb{R}^p$. Let the measured I/O data of this system be denoted by $\mathcal{D}_p$ and let a model of this system be parameterized as follows

$$\hat{y}(k) = \hat{f}_p[y(k-1), \ldots, y(k-\widehat{n_y}), u(k), \ldots, u(k-\widehat{n_u}), \Theta_{\hat{f}_p}] \tag{4.3}$$

In (4.3) the arguments of the approximation function $\hat{f}_p$ are the d elayed true process outputs $y(k)$ and process inputs $u(k)$, $\widehat{n_y}$ and $\widehat{n_u}$ define the model order similarly to the definition of the given system order in (4.2). The parameter vector $\Theta_{\hat{f}_p}$ is a finite dimensional vector of unknown parameters. Let the approximating function $\hat{f}_p$ be a member of a family of approximation functions $\mathcal{F}_{n_1,n_2,\ldots,n_{N_L}}$, i.e. $\hat{f}_p \in \mathcal{F}_{n_1,n_2,\ldots,n_{N_L}}$ represented by a multilayer perceptron consisting of hidden layers of n_1, n_2, $\ldots$, n_{N_L} hidden nodes. The parameter vector $\Theta_{\hat{f}_p} \in \mathbb{R}^{n_\theta}$ then contains all network's weights and biases and n_θ denotes the dimension of the vector $\Theta_{\hat{f}_p}$. The prediction error $e(k)$ is defined by

$$e(k) = y(k) - \hat{y}(k), \qquad \text{for } k = 1, \ldots, N \tag{4.4}$$

The model (4.3) is in the literature called a nonlinear auto-regressive model with an exogenous signal or NARX [8, 9]. If the noise sequence $e(k)$ does not satisfy the previously required assumptions we have to consider a generalization of (4.2), namely

$$\begin{aligned} y(k) = f_p^e[&y(k-1), \ldots, y(k-n_y), u(k), \ldots, u(k-n_u), \\ &e(k-1), \ldots, e(k-n_e)] + e(k) \end{aligned} \tag{4.5}$$

where $e(k) \in \mathbb{R}^p$ is a white noise sequence. A model of this system is then parameterized as follows

$$\begin{aligned} \hat{y}(k) = \hat{f}_p^e[&y(k-1), \ldots, y(k-\widehat{n_y}), u(k), \ldots, u(k-\widehat{n_u}), \\ &e(k-1), \ldots, e(k-\widehat{n_e}), \Theta_{\hat{f}_p^e}] \end{aligned} \tag{4.6}$$

where $e(k)$ are the prediction errors computed by (4.4), $\hat{f}_p^e \in \mathcal{F}_{n_1,n_2,\ldots,n_{N_L}}$ is a neural network and $\Theta_{\hat{f}_p^e}$ are its weights. Compared to the NARX model (4.3),

the model (4.6) is extended in a moving average part of the order $\widehat{n_e}$ and is often called as NARMAX model [8].

Another way of modelling a nonlinear dynamic I/O system is to consider a model described by the following equation

$$\hat{y}(k) = \hat{f}_s[\hat{y}(k-1), \ldots, \hat{y}(k-\widehat{n_y}), u(k), \ldots, u(k-\widehat{n_u}), \Theta_{\hat{f}_s}] \tag{4.7}$$

Here, as the arguments of the approximation function $\hat{f}_s$, $\hat{f}_s \in \mathcal{F}_{n_1,n_2,\ldots,n_{N_L}}$, we have used past values of the model output $\hat{y}(k)$ and the past values of the process input $u(k)$. The free parameters $\Theta_{\hat{f}_s}$ contain the neural network weights and biases. This model assumes that a data generating system being described by

$$y_s(k) = f_s[y_s(k-1), \ldots, y_s(k-n_y), u(k), \ldots, u(k-n_u)] \tag{4.8a}$$

$$y(k) = y_s(k) + e(k) \tag{4.8b}$$

where $e(k)$ is an output noise or measurement noise assumed to be uncorrelated with past inputs and with past outputs. Let us denote the measured input/output data of this system by $\mathcal{D}_o$.

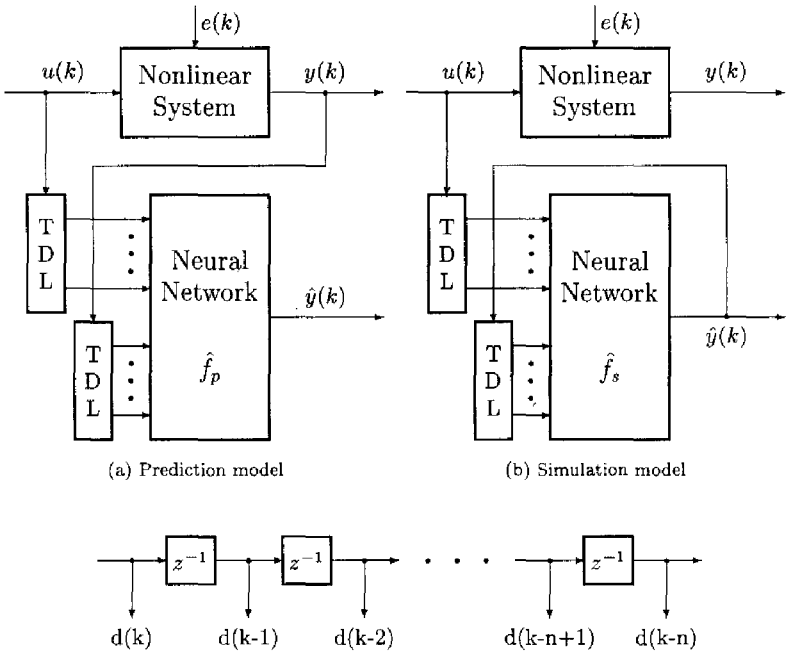
Although both models (4.3) and (4.7) predict the next value of the process output $y(k)$ and they are quite similar in their structure, they state different assumptions and they have also different mathematical properties. These properties become more obvious if the prior assumptions, in this case (4.2) and (4.8), are violated, e.g. in practice.

The first of these two models, (4.3) or (4.6), is called a prediction error model or an equation error model. The second model (4.7) is called a simulation model or an output error model. The first type of model is usually optimized towards a best prediction of the next process output value given past system outputs and inputs, while the second one is optimized for longer time predictions which are based exclusively on past system inputs. A schematic diagram of both of these two models is depicted in Figure 4.1. We frequently use a subscript "p" when talking about a prediction model and a subscript "s" when talking about a simulation model.

The question is now how to choose the unknown parameters of the above proposed models. Typically, an estimate of either $\Theta_{\hat{f}_p}$ or $\Theta_{\hat{f}_s}$ is obtained by minimizing the following criterion

$$J(\Theta_{\hat{f}}) = \frac{1}{2N} \sum_{k=1}^{N} \|y(k) - \hat{y}(k, \Theta_{\hat{f}})\|_2^2 \tag{4.9}$$

where $\|\cdot\|_2$ is the usual Euclidean vector norm, $\hat{f}$ stands either for $\hat{f}_p$ or $\hat{f}_s$, the model orders $\widehat{n_u}$, $\widehat{n_y}$ and possibly $\widehat{n_e}$ are chosen beforehand together with the size of the neural network. As $\hat{y}(k)$ is parametrized nonlinearly it is not in general possible to minimize (4.9) analytically. Moreover, this criterion defines a non-convex function of parameters $\Theta_{\hat{f}}$. The minimized value of $J(\Theta_{\hat{f}})$ gives us some



(a) Prediction model

(b) Simulation model

(c) Taped delay line (TDL)

Figure 4.1: Input/output model parametrizations

idea about the accuracy of the estimated model. It is convenient to compute an index

$$\vartheta = \frac{\sum_{k=1}^{N} \|y(k) - \hat{y}(k, \Theta_f)\|_2^2}{\sum_{k=1}^{N} \|y(k)\|_2^2} \cdot 100\% \tag{4.10}$$

relating the approximation error magnitudes to the magnitudes of the true process output. The accuracy of the approximation is determined by many factors. Basically, we have to distinguish between an approximation accuracy of process nonlinearities and an approximation accuracy of the process dynamics. These two aspects have to be treated together as their separation is complicated by the nonlinearity of the problem. The most important factors influencing the *approximation accuracy* are discussed bellow:

- First of all, it is the value of the criterion (4.9) we find during its minimization. Since we are dealing with a non-convex problem it is hard to access

this issue. We rely here on the optimization routine used to minimize the cost function. Usually a priori knowledge about the physical nature of the problem and a noise level in the system might help us to reject poor solutions and continue the optimization procedure trying to find better ones.

- The accuracy of process nonlinearities approximation will depend not only on the smallest value of the criterion we find in the criterion (4.9) but also on how densely the observation points fill the input space of the approximated transfer function. That implies that the length of the data set will grow very fast with the input dimension if one wants to maintain a certain level of accuracy. This might badly affect the optimization process which will be slowed down due to the increased computational costs.

- As the approximation is concerned with dynamic systems, the bandwidth of the system, understood as a frequency range between the smallest and highest eigen frequency of the system, will also influence the accuracy of approximation. In general, the process bandwidth limits the maximum sampling T_s, used to sample the process inputs and outputs and also the length N of the used set. If the process dynamics are stiff, that is the bandwidth is very broad, the data set can be quite long. The longer the data set the more time is required for minimization of the criterion (4.9). Besides of the choice of a proper data set the accuracy of approximation of the system dynamics will be given by the choice of model orders $\widehat{n_y}$ and $\widehat{n_y}$.

- The accuracy of the approximation will also be determined by the complexity of approximated nonlinearities and the size of the neural network we chose. If the complexity of the neural network is too small than the approximation will be, in general, poor. If the complexity of the neural network is too high one run into overparametrization problems.

- The process noise $e(k)$ can not be omitted from the approximation accuracy discussion. If the noise level in the system is significant, we can expect poor models. As the minimization of (4.9) is a non-convex problem, in general, it might become very hard to separate the effect of input signal $u(k)$ from the effect of the noise signal $e(k)$ in the output signal $y(k)$.

To minimize (4.9) we will use an iterative numerical procedure proposed in Chapter 3, which was a combination of stochastic search (e.g. simulated anneling) and a quasi-Newton search. For an effective quasi-Newton optimization we have to provide the software computer routines with analytical expressions of gradients. These are given in the next sections, both for prediction and simulation I/O models.

4.1.2 *Gradient computations*

To perform an efficient numerical optimization of the criterion (4.9) we need to evaluate the gradients of this cost function with respect to the network weights,

preferably analytically. These gradients are in general defined by

$$\nabla J(\Theta_{\hat{f}}) = \left(g(\theta_{\hat{f}_1}), g(\theta_{\hat{f}_2}), \ldots, g(\theta_{\hat{f}_{n_\theta}})\right)^T \tag{4.11}$$

where

$$g(\theta_{\hat{f}_i}) = \frac{\partial J(\Theta_{\hat{f}})}{\partial \theta_{\hat{f}_i}} \qquad \text{for} \quad i = 1, 2, \ldots, n_\theta \tag{4.12}$$

For the criterion (4.9) then holds

$$g(\theta_{\hat{f}_i}) = -\frac{1}{N}\sum_{k=1}^{N}\left(y(k) - \hat{y}(k, \Theta_{\hat{f}})\right)^T \frac{\partial \hat{y}(k, \Theta_{\hat{f}})}{\partial \theta_{\hat{f}_i}} \tag{4.13}$$

In the following we will distinguish two situations. The first situation is concerned with the prediction error model and the second one is concerned with the output error model.

Prediction model

If the process is modeled by a prediction model (4.3) then the past input and past output arguments of the transfer function are independent of parameters $\theta_{\hat{f}_i}$ and therefore in this case holds

$$\frac{\partial \hat{y}(k, \Theta_{\hat{f}_p})}{\partial \theta_{\hat{f}_{p\,i}}} = \frac{\partial \hat{f}_p[.]}{\partial \theta_{\hat{f}_{p\,i}}} \tag{4.14}$$

The partial derivatives of the approximation function $\hat{f}_p$ with respect to $\theta_{\hat{f}_i}$ parameters are computed by the backpropagation algorithm (3.9–3.17).

Simulation model

The situation is quite different in case of the simulation model. Because this model takes as arguments, besides the past inputs, the past model outputs rather than the past true process outputs these arguments become dependent on parameters $\theta_{\hat{f}_i}$ through the recursive evaluation of the model transfer function. Therefore we differentiate the modeled output with respect to $\Theta_{\hat{f}_s}$ as follows

$$\frac{\partial \hat{y}(k, \Theta_{\hat{f}_s})}{\partial \theta_{\hat{f}_{s\,i}}} = \frac{\partial \hat{f}_s[.]}{\partial \theta_{\hat{f}_{s\,i}}} + \sum_{j=1}^{\widehat{n_y}} \frac{\partial \hat{f}_s[.]}{\partial \hat{y}(k-j)} \frac{\partial \hat{y}(k-j, \Theta_{\hat{f}_s})}{\partial \theta_{\hat{f}_{s\,i}}} \tag{4.15}$$

The last expression is in fact includes the expression (4.14) plus a dynamic summation term, which stands for the network weights dependency on part of the network inputs.

The first term in the equation (4.15) is a static term computed by the backpropagation algorithm (3.9–3.17) as in case of (4.14) and the second term is a dynamic term where

$$\frac{\partial \hat{f}_s[.]}{\partial \hat{y}(k-j)}$$

has to be still evaluated. This expression represents a differentiation of the neural network output with respect to part of its inputs. This operation can be performed by the backpropagation algorithm as already shown in Section 3.2. The formula (4.15) together with the backpropagation algorithm (3.9–3.17) is called *dynamic backpropagation* as proposed in [45].

The equation (4.15) is a recurrent equation evaluation of which may be quite cumbersome for a digital computer. This is determined by four factors:

1. $\widehat{n_y}$ - the number of estimated output delays,
2. p - the output dimension,
3. n_θ - the total dimension of the parameter vector $\Theta_{\hat{f}_s}$,
4. N - the length of the data set.

The approximate number of memory locations $p(\widehat{n_y}+1)n_\theta$, required by a computer program, computing the equation (4.15), can be quite high. The same holds also for the number of arithmetic operations involved in this computation. We have to consider also the length of the data set. These kinds of issues have to be taken into account when formulating this type of modelling problems, otherwise the computing time can be rather high when using a small computer machine.

4.1.3 Black-box state-space models

Let us assume a sampled dynamic system described by

$$x(k+1) = f[x(k), u(k), w(k)] \tag{4.16a}$$
$$y(k) = h[x(k), u(k)] + v(k) \tag{4.16b}$$

Here $w(k)$ is the process disturbance and $v(k)$ is the measurement noise. Assume, that $w(k)$ is a sequence of mutually independent random samples which are not correlated to either $x(k)$ or $u(k)$. The measurement noise $v(k)$ is assumed to be white.

Then a modelling problem can be stated as follows: Given a data set $\mathcal{D}$ of type (4.1), generated by the system (4.16), find a discrete-time state-space model

$$\hat{x}(k+1) = \hat{f}[\hat{x}(k), u(k), \Theta_{\hat{f}}] \tag{4.17a}$$
$$\hat{y}(k) = \hat{h}[\hat{x}(k), u(k), \Theta_{\hat{h}}] \tag{4.17b}$$

approximating the true system. The unknown nonlinear functions of the true system (4.16), f and h, are parametrized by MLPs with weights $\Theta_{\hat{f}}$ and $\Theta_{\hat{h}}$, respectively. The dimension of the state of this model $\hat{x}(k)$ is assumed to be $\hat{n}$, that means that it should be estimated as well unless it is assumed to be known.

If we compare the formulas of the proposed model (4.17) with the formulas of the assumed system dynamics (4.16) we can see that the model does not include a disturbance input $w(k)$. As this signal is not assumed to be available for the observation, the model (4.17) approximates only that part of f in (4.16a) which is a function of $x(k)$ and $u(k)$. The model (4.17) does not use the true system output $y(k)$ either. It is using only the system input $u(k)$ and the internal state $\hat{x}(k)$ to compute its output $\hat{y}(k)$ and therefore belongs to the class of simulation models. A prediction form of a state-space model of a nonlinear process will be discussed later on when we will talk about neural state observers (Section 4.3).

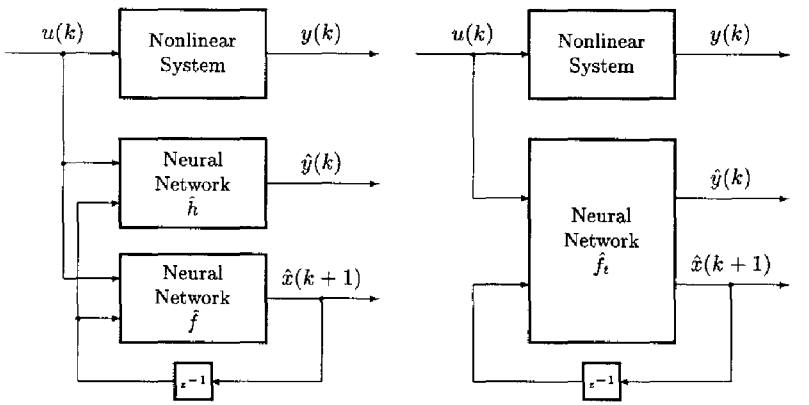
We can see, that both equations (4.17a) and (4.17b) take the same arguments and therefore we can combine these two equations into one equation as follows

$$\begin{bmatrix} \hat{x}(k+1) \\ \hat{y}(k) \end{bmatrix} = \begin{bmatrix} \hat{f}[\hat{x}(k), u(k), \Theta_f] \\ \hat{h}[\hat{x}(k), u(k), \Theta_h] \end{bmatrix} =: \hat{f}_t[\hat{x}(k), u(k), \Theta_{f_t}] \tag{4.18}$$

The nonlinear map $\hat{f}_t$ is now parametrized by a single MLP with weights $\Theta_{\hat{f}_t}$.

The model parametrization (4.18) is a good option for a black-box model. If we have some knowledge about f and/or h maps of the original system (4.16), which can be brought into the model, then the model parametrization will be probably closer to the structure of (4.17). This will be discussed and explained in Section 4.2.

The structures of both neural state-space models (4.17) and (4.18) are depicted in Figure 4.2. The neural network $\hat{f}$ of the model (4.17) is a recurrent neural network as its full output is fed back through a one step delay to its input. The $\hat{h}$ neural network is an ordinary static MLP. Also the $\hat{f}_t$ neural network belongs to the class of recurrent neural networks as a part of its output is fed back to the input. The neural network weights are in all cases obtained by minimizing a criterion (4.9) in which $\Theta_{\hat{f}}$ is replaced either by a concatenation of $\Theta_{\hat{f}}$ and $\Theta_{\hat{h}}$ or by $\Theta_{\hat{f}_t}$. What concerns the approximation accuracy of the model and minimization of the criterion (4.9), the same discussion as the one on page 48 holds also here. The gradient evaluation of the above proposed state-space models for the computer optimization routines is done using similar rules as those we have used in case of I/O models. However, algorithmically the gradient evaluation for state-space models is easier. For completeness of our discussion, we give these gradient formulas in the next two sections.



(a) Structured model (b) Unstructured model

Figure 4.2: State-space model parametrizations

4.1.4 Gradient computations – structured model

Let the $\mathcal{J}_{\hat{x}}^{\hat{f}}(k) \in \mathbb{R}^{\hat{n} \times \hat{n}}$ be the Jacobian matrix of the neural network $\hat{f}$ given by

$$\mathcal{J}_{\hat{x}}^{\hat{f}}(k) = \frac{\partial \hat{f}[\hat{x}(k), u(k), \Theta_f]}{\partial \hat{x}(k)} \tag{4.19}$$

and $\mathcal{J}_{\hat{x}}^{\hat{h}}(k) \in \mathbb{R}^{m \times \hat{n}}$ be the Jacobian matrix of the neural network $\hat{h}$ given by

$$\mathcal{J}_{\hat{x}}^{\hat{h}}(k) = \frac{\partial \hat{h}[\hat{x}(k), u(k), \Theta_h]}{\partial \hat{x}(k)} \tag{4.20}$$

where both Jacobians are evaluated for values of $\hat{x}(k)$ and $u(k)$ at the time instance k. Then the gradient of the error function with respect to weights is given by

$$\frac{\partial J(\Theta_{\hat{f}}, \Theta_{\hat{h}})}{\partial \theta_{\hat{f}_i}} = -\frac{1}{N} \sum_{k=1}^{N} (y(k) - \hat{y}(k))^T \left(\mathcal{J}_{\hat{x}}^{\hat{h}}(k) \frac{\partial \hat{x}(k)}{\partial \theta_{\hat{f}_i}} \right) \tag{4.21}$$

$$\frac{\partial J(\Theta_{\hat{f}}, \Theta_{\hat{h}})}{\partial \theta_{\hat{h}_j}} = -\frac{1}{N} \sum_{k=1}^{N} \left(y(k) - \hat{y}(k) \right)^T \frac{\partial \hat{h}[\hat{x}(k), u(k), \Theta_{\hat{h}}]}{\partial \theta_{\hat{h}_j}} \tag{4.22}$$

where $\theta_{\hat{f}_i}$ and $\theta_{\hat{h}_j}$ are components of the neural network's weights $\Theta_{\hat{f}}$, $\Theta_{\hat{h}}$, respectively, with indexes i and j running through all weights in corresponding neural

networks. The partial derivative of $\hat{x}(k)$ with respect to $\theta_{\hat{f}_i}$ weights has to be computed recursively by

$$\frac{\partial \hat{x}(k)}{\partial \theta_{\hat{f}_i}} = \frac{\partial \hat{f}[\hat{x}(k-1), u(k-1), \Theta_{\hat{f}}]}{\partial \theta_{\hat{f}_i}} + \mathcal{J}_{\hat{x}}^{\hat{f}}(k-1)\frac{\partial \hat{x}(k-1)}{\partial \theta_{\hat{f}_i}} \tag{4.23}$$

The first term in the last expression is being evaluated by the backpropagation algorithm (3.9–3.17). The same algorithm is used to evaluate the Jacobian matrix $\mathcal{J}_{\hat{x}}^{\hat{f}}(k-1)$ which is necessary for evaluation of the second term of (4.23). The last formula is a recurrent relation which has to be evaluated recursively in time.

4.1.5 Gradient computations – unstructured model

The parameter gradient computation of the cost (4.9) for the model (4.18) is similar to the one shown in the previous section. By the backpropagation algorithm we compute the partial derivatives of $\hat{f}_t$ with respect to weights and a Jacobian matrix

$$\mathcal{J}_{\hat{x}}^{\hat{f}_t}(k) = \frac{\partial \hat{f}_t[\hat{x}(k), u(k), \Theta_{f_t}]}{\partial \hat{x}(k)} \tag{4.24}$$

Now we partition this matrix row-wise with respect to the vector dimensions of $\hat{y}(k)$ and $\hat{x}(k+1)$, respectively. Let the upper part, corresponding to differentiation of $\hat{y}(k)$ outputs, be denoted by $\mathcal{J}_{\hat{x}}^{\hat{f}_t,y}$ and the lower part, corresponding to differentiation of $\hat{x}(k+1)$ outputs, be denoted by $\mathcal{J}_{\hat{x}}^{\hat{f}_t,x}$. Then it holds

$$\frac{\partial J(\Theta_{\hat{f}_t})}{\partial \theta_{\hat{f}_{ti}}} = -\frac{1}{N}\sum_{k=1}^{N}(y(k)-\hat{y}(k))^T \frac{\partial \hat{y}(k)}{\partial \theta_{\hat{f}_{ti}}}$$

$$\frac{\partial \hat{y}(k)}{\partial \theta_{\hat{f}_{ti}}} = \frac{\partial \hat{f}_t^y}{\partial \theta_{\hat{f}_{ti}}} + \mathcal{J}_{\hat{x}}^{\hat{f}_t,y}\frac{\partial \hat{x}(k)}{\partial \theta_{\hat{f}_{ti}}}$$

$$\frac{\partial \hat{x}(k+1)}{\partial \theta_{\hat{f}_{ti}}} = \frac{\partial \hat{f}_t^x[\hat{x}(k), u(k), \Theta_{\hat{f}_t}]}{\partial \theta_{\hat{f}_{ti}}} + \mathcal{J}_{\hat{x}}^{\hat{f}_t,x}(k)\,\frac{\partial \hat{x}(k)}{\partial \theta_{\hat{f}_{ti}}}$$

and $\hat{f}_t^y$ denotes that part of the neural network $\hat{f}_t$ which outputs compute $\hat{y}(k)$ and similarly, $\hat{f}_t^x$ denotes that part of $\hat{f}_t$ which outputs compute $\hat{x}(k+1)$.

4.1.6 Model complexity

The minimization procedure, which we usually perform on the sum-squared output error cost function (4.9) to find suitable weights and biases for different neural

networks, does not provide us with optimal structure (or size) of the neural network. It neither tells us the correct model orders. It is clear that the complexity of the model has a strong impact on its performance in a particular application.

When talking about the model complexity we have to distinguish between the order of the dynamics and the complexity of the neural network. These determine the complexity of the model. The order of the model dynamics is defined as follows:

1. In case of an I/O model it is the number of delayed system inputs and delayed outputs, either measured or produced by the model, and possibly also the number of delayed prediction errors (in the case of NARMAX model) specified by

$$\widehat{n_y},\ \widehat{n_u},\ \widehat{n_e}$$

2. In case of a state-space model it is the number of model states specified by

$$\hat{n}$$

The complexity of the neural network, used for the parametrization of the transfer function in the case of an I/O model is specified by the number of hidden layers and the number of nodes in each hidden layer, that is by numbers

$$N_L,\ N_{N1}, \ldots, N_{NN_L}$$

In the case of a state-space model we have to consider either one or two neural networks, depends whether we consider either the paramterization (4.17) or the parametrization (4.18). Then we have to consider either two or one set of parameters, like $N_L,\ N_{N1}, \ldots, N_{NN_L}$ to define the neural network complexity.

The above definition of the complexity of the neural network includes only parameters which are free to choose. There are, in fact, other two parameters which might be also included into the neural network complexity consideration, namely the number of neural network inputs and the number of neural network outputs. However, these two parameters are defined by the dimension of system inputs m and the dimension of system outputs p and the order of the model.

It is very hard to say a priori, what the neural network complexity should be and in practice it is estimated from data and using the prior knowledge about the process. A pragmatic approach to the neural network complexity optimization is to train different neural networks of different complexities and by checking out their performance on validation data we decide about optimal complexity. To do such a search systematically, we can consider, for instance, a set of neural networks with only one hidden layer and to vary the number of hidden nodes N_{N1}. Thus, we have to train a set of neural networks having a range of values for N_{N1} and finally we select one which gives the best validation results. This approach requires tremendous computational effort, is extremely time consuming and yet we often search only through a very small set of neural network complexities. If we try to extend the searched set of neural networks, e.g. for neural networks with two

hidden layers, then we soon find out that we do not have enough computational resources to complete our search. And even if we keep the set of neural network sizes small enough we still have to put a lot of computational effort to train a particular neural network. As training of a neural network is a non-convex problem, the comparison of performance of different neural networks might not be that informative. It means, that we can not say, that one complexity is a better choice than the other, as we can not be sure, that the minimum we have found, in case of the worse neural network is a good one. A partial way out of this problem is to train the same neural network for couple of times, by means of starting the training from different initial starting points and check the similarity of solutions.

The choice of the model order will be in this thesis based on the available prior process knowledge. Otherwise, we would have to vary during the neural network training not only its complexity but also the model order.

4.1.7 Model validity

If a model of a system is adequate then the residues or predictor errors should be unpredictable from past inputs and past outputs. This statement is equivalent to a saying that the prediction errors should be uncorrelated with all linear and nonlinear combinations of past inputs and past outputs. Checking of model validity in a framework of nonlinear systems is complicated because of the system nonlinearity. That means, that the validity of the model should be considered in two directions:

1. accuracy of approximation of the system nonlinearities;
2. the order of the model.

The validity of the estimated model is often being verified by examining the residuals. These are computed both for an estimation data set and for a validation data set. To do this analysis we have a few options which are discussed below:

1. Plot of residues and visual judgment. This is usually the first step in a residual analysis. Basically we can check, for instance, if the mean value of errors is about zero, if the errors are symmetric or asymmetric with respect to the mean value, or we can check for some abrupt behaviours of errors.

2. Plot of residual spectra. When estimating a prediction-error model, assuming a system (4.2) and $e(k)$ to be a white noise, a correct model should show a flat spectrum of residuals. If this is not the case, we have to reconsider either the neural network complexity or the model orders.

   In case of assuming a system configuration (4.8) while estimating a simulation model the residual spectrum should be equal to the spectrum of the output noise, provided that it is not correlated with u and y.

   If the noise is not strictly additive to the output then it can be seen as a colored noise additive to the output filtered by the process dynamic. It is

hard to say, in general, how the spectrum of residuals should look like, as both prediction and simulation model will be biased.

3. Correlation tests. These tests can detect un-modeled nonlinearities by examining cross-correlation functions operating on $e(k)$ and $u(k)$. This approach is described in [5]. In case of the prediction-error model we basically check for correlation of the prediction errors $e(k)$ with the input signal $u(k)$ and mutual correlation of prediction errors $e(k)$. For more proper verification of the model validity we should follow all tests proposed in [5].

Our experience shows that on simple test examples, as the one which will be treated bellow, the correlation and cross-correlation tests are satisfied if the estimated model gives similar performance also on the validation data. As the data generating system becomes more complex, as for instance the polymerization reactor shown in Figure 1.1, the assumed model parametrization will not describe the process such, that these criteria will be satisfied. However, they can be "almost" satisfied, depending on how complex our model is. The main issue here is to decide, whether the model accuracy is sufficient for the particular application. If this is not the case, we have to re-parametrize the model and repeat the estimation phase, possibly until satisfactory results are obtained.

4.1.8 Prediction or simulation – an example

To demonstrate some aspects of the previous discussion we present at this place a simple numerical example.

As a process we consider a system proposed in [45] but extended with a process noise. The system was given as follows

$$y(k) = \frac{y(k-1)y(k-2)y(k-3)u(k-2)(y(k-3)-1)+u(k-1)}{1+y(k-2)^2+y(k-3)^2} \tag{4.25}$$

where $u(k)$ denotes the input sequence and $y(k)$ denotes the output sequence. We included into the simulation of this system a process disturbance $w(k)$ and a measurement noise $v(k)$ while using a state-space description of this system. The new given system is then given by

$$\begin{aligned}
x_1(k+1) &= \frac{x_1(k)x_2(k)x_3(k)x_4(k)(x_3(k)-1)+u(k)}{1+x_2(k)^2+x_3(k)^2} + w_1(k) \\
x_2(k+1) &= x_1(k) + w_2(k) \\
x_3(k+1) &= x_2(k) + w_3(k) \\
x_4(k+1) &= u(k) + w_4(k) \\
y(k) &= x_1(k) + v(k)
\end{aligned}$$

We simulated this system with $u(k)$ being a combination of a sweeped sine wave and a uniformly distributed zero-mean random signal with maximum amplitude 1, shown in the Figure 4.3, all $w_i(k)$ and $v(k)$ were also simulated as uniformly

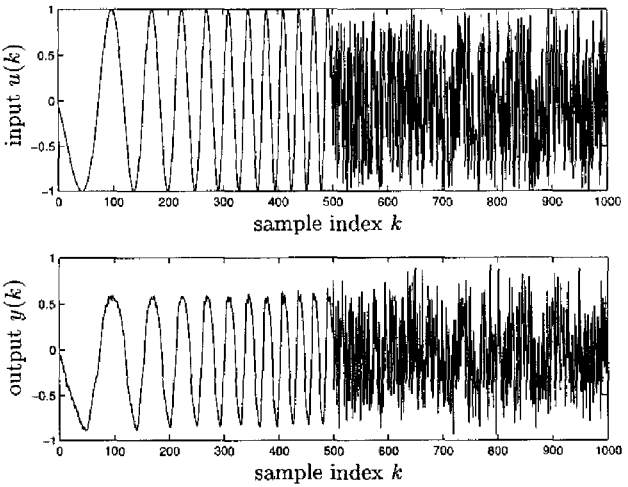


Figure 4.3: Estimation data set

distributed zero-mean random signals, but with maximum amplitude 0.02. We generated two data sets $\mathcal{D}$, $N = 1000$, of simulated I/O data points. The first one was used for the estimation of a model and the second one for its validation.

The idea of combining a random signal with a sine wave signal is that the random signal does not sufficiently excite the system in the nonlinear region. The nonlinearity of this system becomes visible only when y(k) stays in magnitude close to one for a couple of time instances otherwise the product of the last three past samples of $y(k)$ in the numerator of (4.25) decay to zero very fast and the remaining term in the numerator $u(k)$ becomes dominant.

We estimated a set of prediction models parametrized by (4.3) and a set of simulation models parametrized by (4.7). Approximated nonlinear process maps $\hat{f}_p$ and $\hat{f}_s$ were parametrized by a MLP with one hidden layer. In case of the prediction error model, the neural network input was defined as

$$\chi = [y(k-1), y(k-2), y(k-3), u(k-1), u(k-2)]^T$$

and in the case of the simulation model, the neural network input was defined as

$$\chi = [\hat{y}(k-1), \hat{y}(k-2), \hat{y}(k-3), u(k-1), u(k-2)]^T$$

This also means, that we implicitly assumed that we know the correct process orders $\widehat{n_y}$, $\widehat{n_u}$.

The neural network weights were in all cases optimized by minimization of the following criterion

$$J(\Theta_{\hat{f}}) = \frac{1}{2}\sum_{k=1}^{1000}(y(k) - \hat{y}(k, \Theta_{\hat{f}}))^2 \tag{4.26}$$

similarly to (4.9). The number of hidden nodes N_{N1} in the hidden layer was varied from 8 to 19 and the total number of parameters to estimate, given by

$$n_\theta = N_{N1}(5+1) + N_{N1} + 1 = 7N_{N1} + 1$$

was varying from 57 to 134. Note, that 134 is already quite a considerable number of parameters to estimate. To optimize the neural network weights we first performed $100n_\theta$ simulated annealing iterations and then $500n_\theta$ quasi-Newton iterations. We restarted the optimization of each neural network configuration five times, always from different starting point. We evaluated the model performance on the validation data set. The results of prediction error model estimation and its validation are summarized in Figure 4.4(a). In this Figure we have also shown the value of the Akaike's criterion

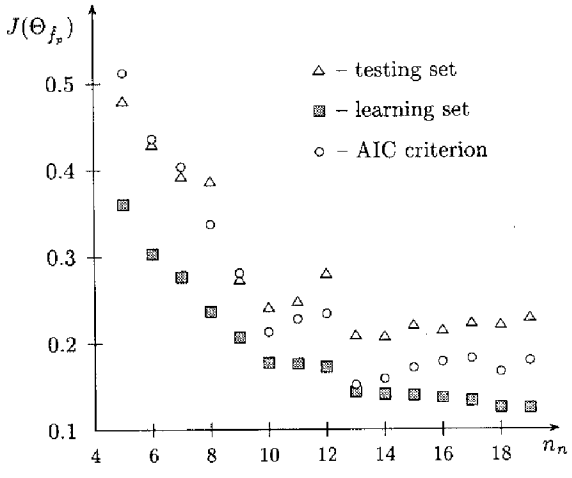
$$AIC(n_\theta) = -2\log(J(\Theta_{\hat{f}_p})/N) + 2n_\theta$$

as proposed in [35]. Only scaled values are shown in this graph as the actual values of this criterion are not that important as the location of its minimum with respect to n_θ. The results of simulation model estimation and validation are summarized in Figure 4.4(b).

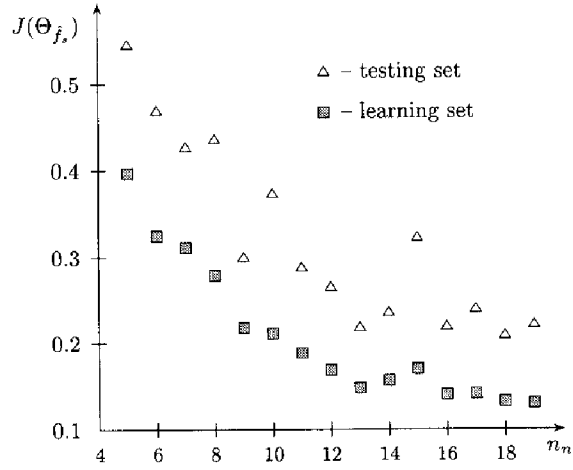
From the Figure 4.4(a) we can see that on estimation data is the cost function monotonically decreasing with increasing number of nodes in the neural network's hidden layer. While testing the model performance on validation data (still in the prediction-error set-up) we can find a turning point from which the cost function value increases. We can also observe that the Akaike's criterion is consistent with testing the model performance on validation data. In case of estimation of the simulation model (Figure 4.4) we can observe similar characteristics of the estimation process: a decrease of the cost function value with respect to the number of hidden nodes on estimation data and an initial decrease of the cost on validation data and than its increase. We can also notice from these results that optimization of a output-error model is more difficult as decrease of the cost function on estimation data is not that gradual as in the case of prediction-error model optimization.

In this example we actually did not succeeded to manage a monotonic decrease of the cost function value on estimation data for neural networks with 14 and 15 nodes. It is very hard to say in this case why, but it may be due to a very complex cost function landscape.

In both experiments, prediction model estimation and simulation model estimation, we have found out that a neural network with 13 hidden nodes might be a good choice. If we compute the predictions and simulations using the true system equations with and without the disturbance and we compute the criterion (4.26)

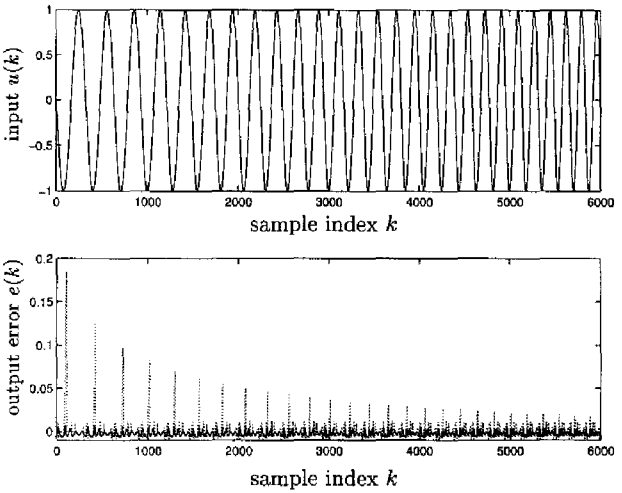


(a) Prediction model optimization

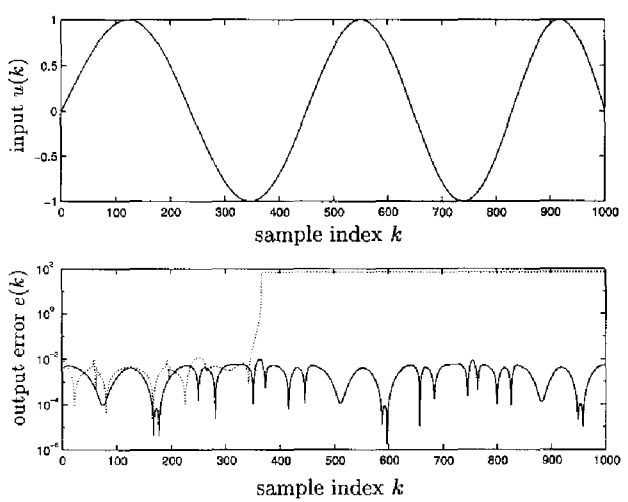


(b) Simulation model optimization

Figure 4.4: I/O neural network model identification results



(a) increasing frequency from $0.003f_0$ to $0.0006f_0$



(b) increasing frequency from $0.002f_0$ to $0.0006f_0$

Figure 4.5: Validation with a sine wave signal

using these data we obtain cost function values 0.1703 and 0.1680, respectively. These values are in fact reached by 13 hidden nodes neural in the neural network both for prediction and simulation models and that is consistent with our already made decision about the neural network complexity.

For further validation of these models we have used a sweeped sine wave signal as an input. In Figure 4.5(a) we can see that the output error in case of the prediction model is decreasing towards higher frequencies, what is to be expected, while the output error in case of the simulation model remains small over the whole input signal spectrum. Testing both models for even smaller frequencies, shown in the Figure 4.5(b), the prediction model is very likely to fail in predicting the next output sample simply by getting stuck outside of estimated region of nonlinearities of the true transfer function. The simulation model is still showing reliable performance. Actually, in this particular example, the process shows most of its nonlinearities when the input signal stays close to 1 for a couple of samples. As the prediction model tends to predict the next output sample by using mainly previous output samples, while underestimating the importance of inputs, its validation by simulation often results in errors similar to the one shown in Figure 4.5(b).

To compute the results shown in Figure 4.4 we needed about one month of computing time on the DEC 3000 computer.

4.1.9 I/O models versus state-space models

When dealing with nonlinear systems it is difficult to say if it is better to model a given process by an I/O model or a state-space model. In case of linear systems the mathematical transformation between both models is trivial, while in the case of nonlinear system this is a nontrivial problem.

In our work we have developed computer software to handle both I/O and state-space models. The I/O model can be estimated either in the prediction configuration (see Figure 4.1(a)) or in the simulation configuration (see Figure 4.1(b)). The state-space models (see Figure 4.2(a) and Figure 4.2(b)) are always estimated as simulations models.

The decision about estimating either an I/O model or a state-space model should be based on the particular purpose of the estimated model. For us the model purpose is to design a controller for a real-world process. I/O models are often used as SISO, one step ahead predictors. Estimation of I/O multivariable models is a much more difficult problem. The number of a neural network inputs grows very fast with the process dimensionality and the order of the system dynamics. If we are interested in a prediction error model the problem difficulty might be still acceptable, but in case of estimating a simulation model we are almost always faced with a difficult optimization problem. In case of a state-space model the number of neural network inputs is smaller and the number of outputs is given either by the state dimension or output dimension. This means that in case of a MIMO system we might end up with less parameters to estimate.

The advantage of state-space models is also their simpler implementation and handling in software programs for a digital computer where we can gain some

algorithmic advantages with respect to the I/O models. These advantages concern mainly the gradient computation where in the case of I/O simulation model the dynamic term in the backpropagation is summed from one to the highest delay used in tapped delay lines while in case of state-space models this sum reduces into just one term. From a conceptual point of view the state-space model parametrization allows us a better understanding and formulation of problems and, what is most, important it allows us also to incorporate our a priori knowledge into the modelling in a straightforward way.

4.2 Grey-box modelling

The grey-box modelling is characterized by the a priori available partial information about the process dynamic and structure which may be used for the model parametrization. This knowledge can be expressed during modelling in many ways. One way can be the choice of parameter values, another way is to elaborate the model structure. As it was already said, the available a priori information does not contain full dynamic description of the process. Consequently, the total dynamical model must contain a white-box part that reflects the a priori knowledge and as well as black-box part to model the complementary process dynamics.

In the following sections we further exploit the state-space models in such a direction that we can effectively incorporate the available a priori process knowledge. Before we start to do this we classify different kinds of a priori knowledge which is usually available.

4.2.1 A priori information in process modelling

In practice we are always dealing with physical processes in which we are able to point out exactly some of the process's physical states. These can be, for instance in different type of processes, represented by following variables:

1. technological processes

   temperatures, pressures, flows, volumes, levels, concentrations

2. mechanical systems

   positions, angles, speeds, angular speeds, accelerations, forces, torques

3. electrical systems

   voltages, currents, powers, fluxes

These variables always obey physical laws which relate them to other variables in the process. For instance the Newton's law relates the acceleration and force in a mechanical system, the acceleration is a time derivative of the speed. In electrical systems, it is Ohm's law or Faraday's law which relates currents, voltages and fluxes to each other. In chemical processes, for instance, pressures or concentrations

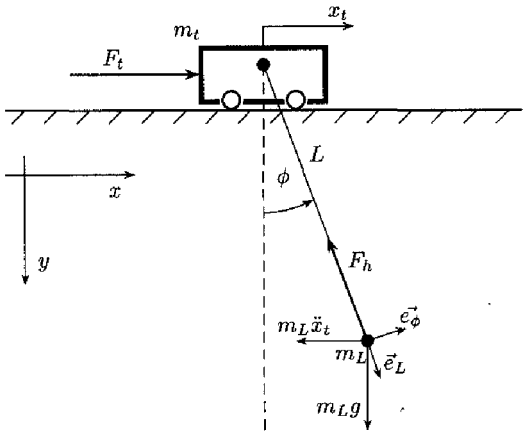


Figure 4.6: Gantry crane schematic diagram

depend on the temperature also by known relations coming clearly from energy balances in the process.

To express mathematically a combination of a white, a priori known part of the model, and a black-box part we will now consider two situations.

Known static relations

A static relation is a relation which does not contain time derivatives or delays of the considered variables. Such a relation can be brought into the model parametrization simply by defining the output map $\hat{h}$ of the model (4.17). To illustrate this case, let us consider the gantry crane process, depicted in Figure 4.6.

Without going into the details of the dynamics of this process, as it will be given precisely later on, assume that we can measure a position of the load (x_l, y_l) in a Cartesian coordinate system, a position x_t of the trolley, the actual length L of the rope holding the load and an angle ϕ of the rope with respect to the vertical direction. If we are interested in modelling the load behaviour in the load coordinates then we can fix the output map $\hat{h}$ of the model (4.17) as follows

$$\begin{aligned} x_l &= x_t + L\sin(\phi) \\ y_l &= L\cos(\phi) \end{aligned}$$

and the complementary system dynamics is given by a state equation

$$\hat{x}(k+1) = \hat{f}[\hat{x}(k), u(k), \Theta_f]$$

where $\hat{f}$ stands for a neural network and the state vector $\hat{x}$ is defined as

$$\hat{x} = (x_t,\ \phi,\ L,\ \ldots)^T$$

where the dots mean that we still have to fill in couple of states to model the process dynamics properly. The first three state components have then a physical meaning while the rest of state components will not have a physical interpretation.

Known dynamic relations

Dynamic relations are those that include time derivatives or time delays. Let us take the gantry crane example again. It is quite straightforward to distinguish in the process behaviour three different subsystems with approximate dynamics:

1. Pendulum dynamics

$$\ddot{\phi} + \frac{g}{L}\sin(\phi) = 0 \tag{4.27a}$$

2. Mass translation dynamics of trolley

$$m_t \ddot{x}_t = F_t \tag{4.27b}$$

3. Mass lift dynamics

$$m_L \ddot{L} = m_L g \cos(\phi) - F_h \tag{4.27c}$$

where F_h is the hoisting force.

Assuming that these relations are known beforehand we can form the following model of this process

$$\hat{x}(k+1) = \bar{f}[\hat{x}(k), u(k)] + \hat{f}_N[\hat{x}(k), u(k), \Theta_f] \tag{4.28a}$$

$$\hat{y}(k) = h[\hat{x}(k)] \tag{4.28b}$$

where the state, control and output vectors are defined as follows

$$x = \begin{pmatrix} x_t \\ \phi \\ L \\ \dot{x}_t \\ \dot{\phi} \\ \dot{L} \end{pmatrix} \qquad u = \begin{pmatrix} F_t \\ F_h \end{pmatrix} \qquad y = \begin{pmatrix} x_l \\ y_l \end{pmatrix}$$

and the known nonlinear functions $\bar{f}$ and h are given by

$$\bar{f}[x,u] = \begin{pmatrix} x_1 + \tau x_4 \\ x_2 + \tau x_5 \\ x_3 + \tau x_6 \\ x_4 + \tau m_t^{-1} u_1 \\ x_5 + \tau g \sin(x_2)/x_3 \\ x_6 + \tau m_L{}^{-1}(m_L g \cos(x_2) - u_2) \end{pmatrix}$$

$$h[x] = \begin{pmatrix} x_1 + x_3 \sin(x_2) \\ x_3 \cos(x_2) \end{pmatrix}$$

where τ is the sampling time of the process. The discretization of dynamical relations (4.27) was, in this case, done by taking a first-order Taylor series approximation of their solutions and therefore the sampling time τ should be kept small.

The model (4.28) is composed of a known part represented by $\bar{f}$ and an unknown part to be estimated, represented by a neural network $\hat{f}_N$.

4.2.2 State partitioning

In the last two sections we could see that the state vector always contained components with either an exact physical meaning or directly related to the outputs or to other states by known relations. The other part of the state vector was composed of states without a specific physical meaning but which are necessary for modelling of the process dynamics. Therefore we propose the following state partitioning of the considered system (4.16)

$$x = \begin{pmatrix} x^0 \\ x^1 \end{pmatrix} \tag{4.29}$$

where $x^0 \in \mathbb{R}^{n^0}$ represents unknown system state components and $x^1 \in \mathbb{R}^{n^1}$ are known state components, physically well defined and later on they will represent our a priori knowledge about the system dynamics.

In the hidden state component x^0 we can possibly further distinguish more types of states. For instance, in case of assuming process disturbances which are not white, x^0 will also include states modelling the coloring of the disturbances.

The above discussion brings us directly to grey-box modelling problem discussed in the following sections.

4.2.3 Grey-box state-space models

A grey-box state-space model in discrete time will be from now on understood as a model of type (4.17) in which we assume:

1. that the process output map h is known exactly and set in the model by defining $\hat{h} = h$;
2. that part of the true system state map f can also be known and the model state map is then a combination of an analytically known part $\bar{f}$ and an unknown part $\hat{f}_N$ approximated by a neural network;
3. that the state vector is partitioned according to (4.29).

The resulting grey-box state-space model is then parametrized as follows

$$\hat{x}(k+1) = \hat{f}[\hat{x}(k), u(k), \Theta_{\hat{f}}] \tag{4.30a}$$

$$y(k) = h[\hat{x}^1(k), u(k)] \tag{4.30b}$$

where the function $\hat{f}$ denotes here a combination of an analytical part $\bar{f}$ and a neural network approximated part $\hat{f}_N$ and $\hat{x}^1$ is the known part of the state vector carrying our process a priori knowledge. The weights $\Theta_{\hat{f}}$ of the neural network $\hat{f}$ are estimated by a minimization of a criterion (4.9) with respect to $\Theta_{\hat{f}}$.

4.2.4 Gradient computations – fixed output map

The partial derivatives of the criterion (4.9) with respect to the free parameters $\Theta_{\hat{f}}$ are for the model (4.30) determined in the same way as described in Section 4.1.4. The only difference is in replacement $\hat{h}$ by h, skipping (4.22), the neural network part of $\hat{f}$ is handled by the backpropagation algorithm and the analytical part $\bar{f}$ has to be differentiated from problem to problem independently.

4.2.5 Computational costs

The overall procedure for gradient evaluations of the cost function (4.9) may be time consuming mainly for large values of N. In Table 4.1 we give some figures of time needed to evaluate the cost function (4.9) by different computers. These figures correspond to an identification problem done on the FBPR (see Figure 1.1). The number of inputs in this experiment was $m = 4$, the number of outputs was $p = 4$ and the length of the data set was $N = 7201$. The model was parametrized by a neural state-space model of type (4.30), the order was chosen as $\hat{n} = 6$ and $\hat{f}$ was parametrized by an MLP with one hidden layer containing 12 nodes. This yields an estimation of

$$(10 + 1) \times 12 + (12 + 1) \times 6 = 210$$

parameters.

The first column of Table 4.1 shows the type of used computer hardware. The second column shows an average CPU time needed for the cost function (4.9) evaluation and the third column shows a timing for the gradient computations. These figures demonstrate that optimization of the cost function is, in general, a time consuming process and that it is quite essential to have a fast computer available.

Imagine, for instance, that we want to optimize 6 different neural network configurations, each containing one hidden layer, and we vary the number of nodes from 10 to 15. We want to estimate also models of different orders, let's say ranging from 5 to 8. Because we are essentially using only local optimization routines, we want to restart every optimization let's say 5 times, what is still a very modest number compared to the dimensionality of the parameter space. However, we already end up with 120 optimization problems of parameter dimensionality in a range from 176 parameters for the smallest model up to 320 parameters for the largest model. Let us assume, that we want to perform 5000 stochastic search iteration and that we limit the number of gradient evaluations in a gradient search also by 5000. Then the estimated time to complete such an optimization task will

Table 4.1: CPU time spent in evaluation of the cost function (4.9) and its gradients

Hardware	cost CPU time [sec]	gradient CPU time [sec]
Intel 386/25MHz	12.5	741.4
VAXstation 3100	10.3	332
VAXstation 3100-M76	4.7	165
Pentium/90MHz	0.37	30.6
DEC 3000 M300X	0.28	13.0

not be less than about 90 days, because the computation overhead of optimization routines still has to be added.

4.2.6 *Initial state condition estimation*

To start a simulation of the state-space model (4.17) we have to specify the value of the state vector $\hat{x}(0)$ at the time instance $k = 0$. Often we choose as an initial condition $\hat{x}(0) = 0$. In the case of a fully parametrized model (4.17) the minimization routine will choose such a state coordinates that the transient from a wrong initial condition will be minimal. In the case of an estimation of a grey-box state-space model (4.30) it is possible to specify more precisely an initial condition for the known part of the state vector $\hat{x}^1$ and choose a zero initial condition for the hidden state components $\hat{x}^0$. The initial condition can also be estimated from the data together with the neural network weights. Then the estimation problem could be formulated as a minimization of the following criterion

$$J_0(\Theta_{\hat{f}}, \hat{x}(0)) = \frac{1}{2N} \sum_{k=1}^{N} \|y(k) - \hat{y}(k, \Theta_{\hat{f}}, \hat{x}(0))\|_2^2 \tag{4.31}$$

with respect to $\Theta_{\hat{f}}$ and $\hat{x}(0)$. If we assume a model parametrization (4.30), the gradient of the criterion (4.31) with respect to $\Theta_{\hat{f}}$ is evaluated in the same way as already shown in Section 4.2.4. The only problem is to determine the partial derivatives of (4.31) with respect to the initial state $\hat{x}(0)$. These are computed as follows

$$\frac{\partial J_0(\Theta_{\hat{f}}, \hat{x}(0))}{\partial \hat{x}_i(0)} = \frac{1}{N} \sum_{k=1}^{N} (y(k) - \hat{y}(k))^T \frac{\partial \hat{y}(k, \Theta_{\hat{f}}, \hat{x}(0))}{\partial \hat{x}_i(0)} \tag{4.32}$$

and

$$\frac{\partial \hat{y}(k, \Theta_{\hat{f}}, \hat{x}(0))}{\partial \hat{x}_i(0)} = \mathcal{J}_{\hat{x}}^h(k) \frac{\partial \hat{x}(k, \Theta_{\hat{f}}, \hat{x}(0))}{\partial \hat{x}_i(0)}$$

where $\mathcal{J}_{\hat{x}}^{h}(k)$ is the Jacobian matrix of the output map h with respect to $\hat{x}(k)$ and $i = 1, \ldots, n$. The partial derivatives of $\hat{x}$ with respect to $x(0)$ are computed by the following recurrsive formula

$$\frac{\partial \hat{x}(k)}{\partial \hat{x}_i(0)} = \frac{\partial \hat{f}[\hat{x}(k-1), u(k-1)]}{\partial \hat{x}(k-1)} \frac{\partial \hat{x}(k-1)}{\partial \hat{x}_i(0)} \tag{4.33}$$

for $i = 1, \ldots, n$.

The evaluation of gradients of the cost function with respect to the initial condition requires a computation of the recursive formula (4.33), which might be time consuming. If we do not have enough computing power and we fix the initial condition to a constant value we can often observe an abrupt behaviour of simulated states during the first few simulated samples. To overcome this problem we can skip a requirement of an optimal estimation of $x(0)$ by minimizing (4.32) and we can simply start the optimization from a chosen initial condition and after a few iterations, when the state coordinates were chosen, we compute a mean value of simulated states by

$$\hat{x}(0) = \frac{1}{N - N'} \sum_{k=N'}^{N} \hat{x}(k) \tag{4.34}$$

where $N' > 1$ is used to skip the transients, and use this value as an initial condition for further optimization of (4.9). This method was experimentally proven to give satisfactory results.

4.3 Nonlinear neural state observers

The so far discussed state-space models represented simulation models of the true system and they used only the system input u to compute the model state $\hat{x}$ and output y evolutions. As the true system is also subject to a disturbance input w, there will always be a difference in the simulated output $\hat{y}$ and the true output y due to the fact that the model does not take into the account the effect of the disturbance. The purpose of treating the problem of state estimation is to improve the state estimates $\hat{x}$ computed by the simulation model for the disturbance effects.

Let the system be again described in the discrete-time domain by equations (4.16). The state estimation problem is concerned with estimating a state of this system at time k given all measurements up to the time index k. This estimate is denoted by $\hat{x}(k \mid k)$. A prediction involves an estimation of the state at some future time $k + l$, $l > 0$ and the corresponding estimate is then denoted by $\hat{x}(k + l \mid k)$.

Definition 4.1. By a single-stage ahead state predictor we denote a state estimator estimating the value of the state at time $k + 1$ denoted by $\hat{x}(k + 1 \mid k)$ given all measurements up to the time moment k.

Definition 4.2. By a current-stage state filter we denote a state estimator estimating the value of the state at time k denoted by $\hat{x}(k \mid k)$ given all measurements up to and inclusive the time moment k.

4.3.1 A single-stage ahead state predictor

An optimal nonlinear discrete-time single-stage predictor, in general, is required to minimize an estimation error criterion

$$J = \text{trace}\, \mathcal{E}\{\tilde{x}(k \mid k-1)\tilde{x}(k \mid k-1)^T\} \tag{4.35}$$

The estimation error $\tilde{x}$ is given by

$$\tilde{x}(k \mid k-1) = x(k) - \hat{x}(k \mid k-1) \tag{4.36}$$

where $x(k)$ is the true system state and $\hat{x}(k \mid k-1)$ is its estimated value. The main problem is now how to generate such a sequence of state estimates $\hat{x}(k \mid k-1)$ which would minimize (4.35).

Let us consider as a candidate for such an estimator the following system

$$\begin{aligned}
\hat{x}(k+1 \mid k-1) &= \hat{f}[\hat{x}(k \mid k-1), u(k)] && \text{(4.37a)}\\
\hat{y}(k \mid k-1) &= h[\hat{x}^1(k \mid k-1), u(k)] && \text{(4.37b)}\\
\hat{x}(k+1 \mid k) &= \hat{x}(k+1 \mid k-1) + \hat{g}[y(k) - \hat{y}(k \mid k-1)] && \text{(4.37c)}\\
\hat{x}(0 \mid -1) &= \hat{x}_0 \qquad \text{is given}
\end{aligned}$$

where $\hat{f}$ and h represent an estimated model of the given system in a form (4.30) and $\hat{g}$ is a correction to be designed. The first equation (4.37a) defines an a priori state prediction at the time instance k based on knowledge of $u(k)$ and a state estimate $\hat{x}(k \mid k-1)$ estimated on the previous stage from data up to the time instance $k-1$. The second equation (4.37b) computes an output estimate $\hat{y}(k \mid k-1)$ based on a state estimate $\hat{x}(k \mid k-1)$ available at this stage. The estimated output is then compared to the true process output. The output prediction error, containing information about the influence of the disturbance $w(k)$ and the noise $v(k)$ and given as

$$e(k) = y(k) - \hat{y}(k \mid k-1) \tag{4.38}$$

is then used by the $\hat{g}$ correction term in the third equation (4.37c) to correct the a priori state estimate for the disturbance effect. The nonlinear function $\hat{g}$ must be chosen such, that (4.35) is minimal. The correction term $\hat{g}$ will be later referred to also as a nonlinear static filter gain or simply filter gain.

In general, the filter gain $\hat{g}$ in (4.37c) is a complex nonlinear function which analytical synthesis is a very hard problem and in fact, in general, there is no analytical solution to this problem. Therefore we parametrize this gain by a static neural network and estimate its weights from data. Let us denote this approximation by $\hat{g}_f[y(k) - \hat{y}(k), \Theta_{\hat{g}}]$ and let $\Theta_{\hat{g}}$ denotes a vector of all network weights and biases. From now on, the state estimation problem translates to a parameter estimation problem where a neural network parameters $\Theta_{\hat{g}}$ are estimated such that (4.35) is minimal. However, this minimization would require knowledge of the true process state $x(k)$. Because of a lack of this knowledge we can minimize the

output error (4.38) and hope that the state estimation error will be small as well. The parameters of the neural network $\Theta_{\hat{g}}$ are the obtained by a minimization of the following criterion

$$J(\Theta_{\hat{g}}) = \frac{1}{2N} \sum_{k=1}^{N} \|e(k)\|_2^2 \tag{4.39}$$

where $e(k)$ is given by (4.38). This criterion coincides with the criterion (4.9), used for the model estimation. The state estimates obtained by a minimization of the criterion (4.39) are in fact weighted state estimates, where the weighting factor is the output map h.

4.3.2 Current-stage state filter

An optimal nonlinear discrete-time current-stage state filter is required to minimize an estimation error criterion

$$J = \text{trace } \mathcal{E}\{\tilde{x}(k \mid k)\tilde{x}(k \mid k)^T\} \tag{4.40}$$

The estimation error $\tilde{x}$ is given by

$$\tilde{x}(k \mid k) = x(k) - \hat{x}(k \mid k)$$

Let us consider a nonlinear current-stage state filter described by the following set of equations

$$\begin{aligned}
\hat{x}(k \mid k) &= \hat{x}(k \mid k-1) + \hat{g}_c[y(k) - \hat{y}(k \mid k-1),\, \Theta_{\hat{g}_c}] && \text{(4.41a)}\\
\hat{x}(k \mid k-1) &= \hat{f}[\hat{x}(k-1 \mid k-1), u(k)] && \text{(4.41b)}\\
\hat{y}(k \mid k-1) &= h[\hat{x}^1(k \mid k-1)] && \text{(4.41c)}\\
\hat{x}(0 \mid 0) &= \hat{x}_0 \qquad \text{is given}
\end{aligned}$$

where the nonlinear function $\hat{g}_c$ is again parametrized by a static neural network and the vector $\Theta_{\hat{g}_c}$ contains all its weights. An optimal set of weights is determined similarly as in the previous Section, that is by minimization of the criterion (4.39). The prediction error $e(k)$ is again defined by (4.38), but $\hat{y}(k \mid k-1)$ is computed by the system (4.41).

4.3.3 Discussion

The proposed two structures of the state observer are depicted in Figure 4.7 and in Figure 4.8, respectively. From a comparison of these two diagrams we can see that the predictor corrects the model state prediction before the delay and the filter makes this correction after the delay. The choice between these two schemes should be based on the controller implementation. When we assume, that the controller should at the time instance k deliver a control sample $u(k)$ to the process input,

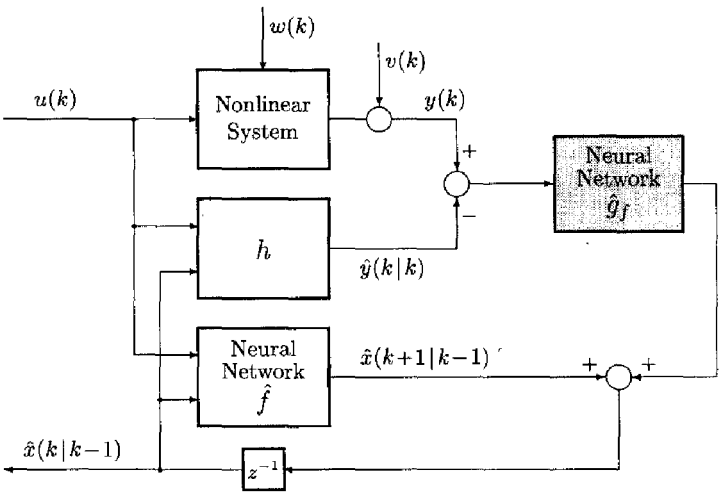


Figure 4.7: A single-stage state estimator diagram

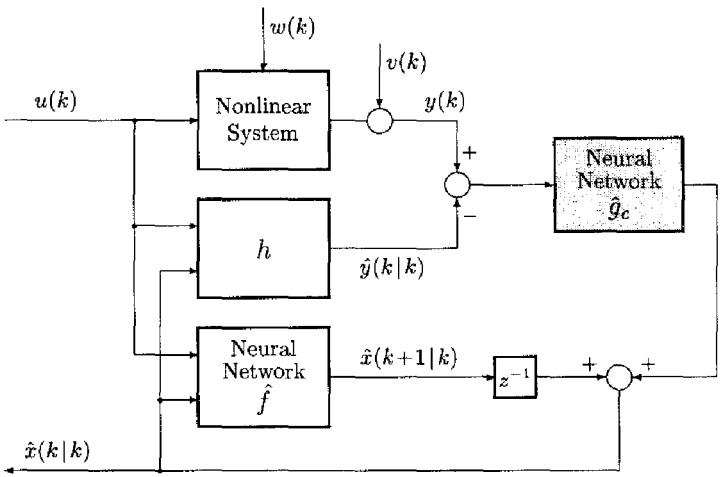


Figure 4.8: A current-stage state estimator diagram

what means that it must be computed before the time moment k using a state estimate $\hat{x}(k \mid k-1)$ we have to use the predictor schema. If the control sequence is being computed using the filter, the control output sample $u(k)$ will be delayed by the computing procedure.

The parametrization of the gain $\hat{g}$ assumed as an input the innovation sequence (4.38). This type of parametrization was proposed also in [64] and in fact it gains a lot from an inspiration of a state estimation in linear dynamic systems. As we are dealing with nonlinear systems, proper parametrization of $\hat{g}$ would be

$$\hat{g}[\hat{y}(k), y(k), \hat{x}(k), u(k), \Theta_{\hat{g}}] \tag{4.42a}$$

That means that, in general, the filter gain is also a function of the system state $\hat{x}(k)$ and $u(k)$ because it is very unlikely that in different regions of the system nonlinearity an optimal gain will be locally the same. Also the values of $y(k)$ and $\hat{y}(k)$ should be used independently, because the output map h is in general assumed to be a nonlinear function.

If the system nonlinearity is not too complex the gain $\hat{g}$ parametrization can be considered in a simpler form. The most interesting modifications include:

1. replacement of $y(k)$ and $\hat{y}(k)$ by their difference

$$\hat{g}[y(k) - \hat{y}(k), \hat{x}(k), u(k), \Theta_{\hat{g}}] \tag{4.42b}$$

2. assuming only the prediction error

$$\hat{g}[y(k) - \hat{y}(k), \Theta_{\hat{g}}] \tag{4.42c}$$

3. assuming only $\hat{y}(k)$ and $y(k)$, but independently

$$\hat{g}[y(k), \hat{y}(k), \Theta_{\hat{g}}] \tag{4.42d}$$

Taking a different parametrization of the gain $\hat{g}$ we can obtaine better or worth approximation of an optimal gain g, that means better or worth state estimates.

4.3.4 Gradient computations

In this section we give the gradient computation of the criterion (4.39) assuming the single-stage ahead state predictor given by (4.37) and assuming that the filter gain $\hat{g}$ has the form (4.42c). The gradient of the cost function (4.39) with respect to weights $\Theta_{\hat{g}}$ is in this case given by

$$\frac{\partial J(\Theta_{\hat{g}})}{\partial \theta_{\hat{g}_i}} = \frac{1}{N} \sum_{k=1}^{N} \Big(y(k) - \hat{y}(k)\Big)^T \frac{\partial \hat{y}(k)}{\partial \theta_{\hat{g}_i}}$$

for $i = 1, 2, \ldots, n_\theta$ and

$$\frac{\partial \hat{y}(k)}{\partial \theta_{\hat{g}_i}} = \mathcal{J}^h_{\hat{x}}(k) \frac{\partial \hat{x}(k)}{\partial \theta_{\hat{g}_i}}$$

where $\mathcal{J}_{\hat{x}}^{h}(k)$ is again the Jacobian matrix of the output map h with respect to $\hat{x}$. The partial derivative of states $\hat{x}$ with respect to the weights are computed by the following formula

$$\frac{\partial \hat{x}(k)}{\partial \theta_{\hat{g}_i}} = \frac{\partial \hat{g}[y(k-1) - \hat{y}(k-1), \Theta_{\hat{g}}]}{\partial \theta_{\hat{g}_i}} + \mathcal{J}_{\hat{x}}^{\hat{f}}(k-1)\frac{\partial \hat{x}(k-1)}{\partial \theta_{\hat{g}_i}} - \mathcal{J}_{\hat{y}}^{\hat{g}}(k-1)\frac{\partial \hat{y}(k-1)}{\partial \theta_{\hat{g}_i}}$$

$\mathcal{J}_{\hat{x}}^{\hat{f}}(k-1)$ is the Jacobian matrix of the system map $\hat{f}$ with respect to the state given by (4.19) and $\mathcal{J}_{\hat{y}}^{\hat{g}}(k)$ is the Jacobian matrix of the neural network $\hat{g}$ given by

$$\mathcal{J}_{\hat{y}}^{\hat{g}}(k) = \frac{\partial \hat{g}[y(k) - \hat{y}(k), \Theta_{\hat{g}}]}{\partial \hat{y}(k)}$$

4.4 Linear MIMO state-space identification

In this section we are concerned with identification of linear time-invariant models for multivariable data. We again assume that a measured data set (4.1) is available and we want to find a linear time-invariant state-space model

$$\hat{x}(k+1) = \hat{A}\hat{x}(k) + \hat{B}u(k) \tag{4.43a}$$

$$\hat{y}(k) = \hat{C}\hat{x}(k) \tag{4.43b}$$

where $\hat{x}(k), u(k), y(k)$ are the state, control, output vectors, respectively. For the case of simplicity, we assume that the modelled system does not contain the direct feed-through. The objective is to find such constant matrices $\hat{A}, \hat{B}, \hat{C}$ that the simulated output $\hat{y}$ will be close to the real output y in the usual least-squares sense. Let all the unknown matrices $\hat{A}, \hat{B}, \hat{C}$ in the model (4.43) be fully parametrized and let Θ_L denotes a column vector of all entries of matrices $\hat{A}, \hat{B}, \hat{C}$. This is expressed by the following criterion

$$J = \frac{1}{2N}\sum_{k=1}^{N} \|\hat{y}(k, \Theta_L) - y(k)\|_2^2 \tag{4.44}$$

which is then in an optimal situation minimal with respect to $\hat{A}, \hat{B}, \hat{C}$. Because of the non-quadratic dependency of the value of (4.44) on the model parameters, there is no analytical solution to this problem and again a numerical minimization has to be employed here.

Notice, that the linear model (4.43) can be seen as a neural state-space model with a very simple neural network containing just an output layer of n linear nodes (see the Figure 4.9). Then we can use the same concept of gradient computation as the one given in Section 4.1.4. The resulting gradient computation algorithm

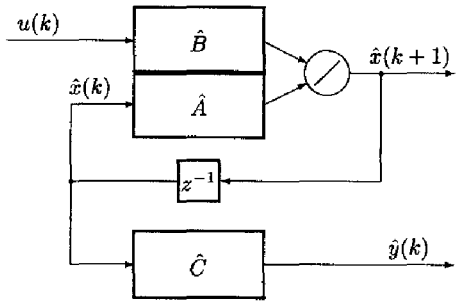


Figure 4.9: Linear state-space identification set-up: seen as a neural network only with one linear node

is as follows

$$\frac{\partial J(\Theta_L)}{\partial \theta_{L_i}} = \sum_{k=1}^{N} \Big(\hat{y}(k) - y(k)\Big)^T \frac{\partial \hat{y}(k)}{\partial \theta_{L_i}}$$

where for the parameters of Θ_L corresponding to the $\hat{C}$ matrix elements holds

$$\frac{\partial \hat{y}(k)}{\partial c_{i,j}} = I_i x_j(k)$$

i and j indices run through all elements of the $\hat{C}$ matrix and I_i is a zero column vector of dimension of $\hat{y}$ with unit entry on ith row. For the parameters of Θ_L corresponding to the $\hat{A}$ and $\hat{B}$ matrices components hold

$$\frac{\partial \hat{y}(k)}{\partial b_{i,j}} = \hat{C}\frac{\partial \hat{x}(k)}{\partial b_{i,j}} \qquad \frac{\partial \hat{y}(k)}{\partial a_{i,j}} = \hat{C}\frac{\partial \hat{x}(k)}{\partial a_{i,j}}$$

where i and j indices run through all elements of the $\hat{A}$ matrix and the $\hat{B}$ matrix, independently. The partial derivatives of states $\hat{x}$ are computed by the following recurrent formulas, similar to the dynamic backpropagation rule (4.23)

$$\frac{\partial \hat{x}(k)}{\partial b_{i,j}} = I_i u_j(k-1) + \hat{A}\frac{\partial \hat{x}(k-1)}{\partial b_{i,j}}$$

where i and j indices run through all elements of the $\hat{B}$ matrix and

$$\frac{\partial \hat{x}(k)}{\partial a_{i,j}} = I_i \hat{x}_j(k-1) + \hat{A}\frac{\partial \hat{x}(k-1)}{\partial a_{i,j}}$$

i and j indices run through all elements of the $\hat{A}$ matrix. In this scheme we have to estimate $n^2 + n(m+p)$ parameters. From a theoretical point of view this is

an overparametrized identification problem and by considering e.g. observability canonical form we could remove n^2 parameters. For detailed discussion see [34] and references therein. Main purpose of linear identification in this thesis is to have a kind of a reference value for the nonlinear identification. Therefore we are mainly interested in the performance of the best least-squares linear simulation model.

Note that the minimization of (4.44) with respect to Θ_L is a non-convex optimization problem. For reliability of this minimization we use the same techniques as for neural network training, that is a combination of the simulated annealing and the quasi-Newton optimization. To start the minimization of (4.44) we have used a random guess for $\hat{A}, \hat{B}, \hat{C}$ matrices. It is clear that if this results in an unstable system than it is not possible to evaluate the cost function. Therefore we incorporated into the simulated annealing a simple stability test to be able to refuse unstable trials. The optimization was started by an simulated annealing search and then, after a sufficient optimization of the cost function, we switched to the quasi-Newton search.

4.5 Gantry crane identification - A case study

4.5.1 Equations of motion

To illustrate the previously described modelling issues we present here an example of model identification of a nonlinear MIMO process. The considered process represents a gantry crane. The gentry crane is a machine for lifting and lowering a load and moving it horizontally, with the hoisting mechanism an integral part of the machine. The trolley is a device which travels along the horizontal direction and carries the hoisting mechanism. The process is schematically depicted in the Figure 4.6. Assume the following notation: m_t - the total mass of the trolley, x_t - the position of the trolley, F_t - the driving force on the trolley, m_L - the mass of the load, L - the length of the rope, ϕ - the angle of the rope with respect to the vertical axis measured anticlockwise.

The equation of motion are derived from kinematics laws valid for plane motions of rigid bodies [66] and Newton's laws of motion. Referring to Figure 4.6, it holds for the velocity and acceleration of the load in a vector notation

$$\vec{v} = \dot{L}\vec{e_L} + L\dot{\phi}\vec{e_\phi}$$

$$\vec{a} = (\ddot{L} - L\dot{\phi}^2)\vec{e_L} + (L\ddot{\phi} + 2\dot{L}\dot{\phi})\vec{e_\phi}$$

where $\vec{e_L}$ is is the unit radial vector and $\vec{e_\phi}$ is the unit transverse vector. Applying the Newton's laws we can write for the forces acting on the trolley an equation

$$m_t\ddot{x}_t = F_t + F_h \sin(\phi)$$

For the forces acting on the load in the radial and transverse directions hold equations

$$m_l L\ddot{\phi} + 2m_L \dot{L}\dot{\phi} = -m_L g \sin(\phi) - m_L \ddot{x}_t \cos(\phi)$$

$$m_L \ddot{L} - m_L L\dot{\phi}^2 = m_L g \cos(\phi) - F_h - m_L \ddot{x}_t \sin(\phi)$$

Considering also friction forces the equations of motion of the system should be extended for damping factors and the final equations of motion are

$$\ddot{x}_t = \frac{F_t}{m_t} + \frac{F_h}{m_t}\sin(\phi) - \frac{d_t}{m_t}\dot{x}_t \tag{4.45}$$

$$\ddot{\phi} = -\frac{g}{L}\sin(\phi) - \frac{1}{L}\ddot{x}_t \cos(\phi) - \frac{2}{L}\dot{L}\dot{\phi} - \frac{d_\phi}{m_L}\dot{\phi} \tag{4.46}$$

$$\ddot{L} = g\cos(\phi) - \frac{F_h}{m_L} + L\dot{\phi}^2 - \ddot{x}_t \sin(\phi) - \frac{d_L}{m_L}\dot{L} \tag{4.47}$$

where d_t, d_ϕ and d_L are the damping constants. Notice, that in the equation (4.46) and (4.47) we did not substitute for $\ddot{x}_t$ the right-hand side of the equation (4.45), for a compactness of the notation. A linearization of the process dynamics shows two pure integrators, one in the motion of the trolley and the second one in the lifting and lovering of the load action. To have a well posed identification problem we prestabilized this process by a static feedback

$$F_t = -(x_t + L\sin(\phi) - x_{ref}) - 2\dot{x}_t + 2\dot{\phi} \tag{4.48}$$

$$F_h = 14.7 + 4(L\cos(\phi) - 1 - y_{ref}) \tag{4.49}$$

computed by an LQ design for a linearized system around an equilibrium point given by

$$x_t = \phi = \dot{x}_t = \dot{\phi} = \dot{L} = 0, \quad L = 1, \quad F_t = 0, \quad F_h = m_L g \tag{4.50}$$

In the feedback law (4.48)-(4.49) we fed back the load coordinates

$$x_l = x_t + L\sin(\phi) \qquad y_l = L\cos(\phi)$$

instead of the trolley position x_t and the rope length L to create reference inputs which are directly related to the future control problem. The identification problem can then be formulated as to estimate a dynamical relation between the load reference position and the actual load position.

The above described model of a gantry crane process, used for a nonlinear neural identification, was proposed in [19].

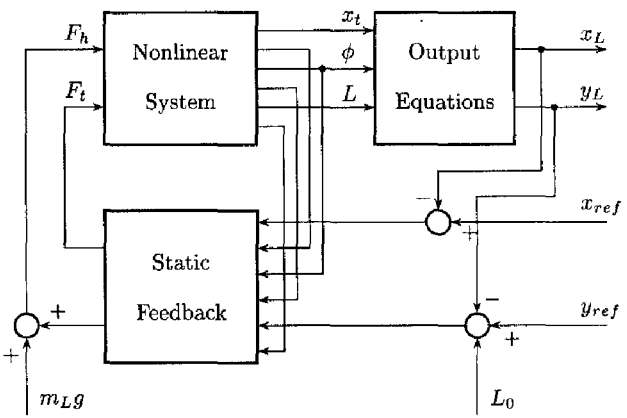


Figure 4.10: Gantry crane identification set-up

4.5.2 *Identification experiment*

The performance of the final I/O neural model was found in [19] to be unsatisfactory. This was mainly due to a difficult quasi-Newton optimization problem resulting from minimization of an output error criterion. Here we parametrized the estimated model in a state-space form and added the simulated annealing optimization to the minimization algorithm.

We defined the inputs and outputs as follows

$$u = \begin{pmatrix} x_{ref} \\ y_{ref} \end{pmatrix} \qquad y = \begin{pmatrix} x_L \\ y_L \end{pmatrix}$$

We simulated equations (4.45)-(4.47),(4.48),(4.49) by a numerical integration using a Merson form of the Runge-Kutta method. For the process parameters we have chosen the following values: $m_t = 3.5\text{kg}$, $m_L = 1.5\text{kg}$, $d_t = 0.1\text{Ns/m}$, $d_\phi = 0.01\text{Ns/rad}$, $d_L = 10\text{Ns/m}$ and $g = 9.8\text{m/s}^2$.

As a testing input signal we have chosen a sequence of uniformly distributed random samples with maximal amplitudes

$$\max|x_{ref}| = 5 \qquad \max|y_{ref}| = 2$$

followed by a zero-order hold with a sampling time of 0.2 seconds. From physical construction of the gantry crane reference values for the vertical position of the load above 1 do not make sense as then the physical length of the rope L would become negative. However, as the test signal was of white noise type, it was possible to use a higher range for y_{ref} to excite sufficiently the process dynamics

in its nonlinear regions. We simulated the process twice for 500 seconds. After re-sampling output signals with a sampling time $T_s = 0.2$s we created two data sets of 2501 data points, one used for estimation purposes and the other one for validation purposes. The process model was parametrized by a nonlinear neural state-space structure according to 4.2(a) where the output map h was chosen such that the two process outputs were taken as the first two components of the state vector, that is

$$h[\hat{x}] = (\hat{x}_1, \hat{x}_2)^T$$

giving them a physical meaning of a position of the load. The rest of the state components was treated as a black-box part of the model. The dimension of the state vector was chosen 6 what can be a priori assumed from the physical structure of the process. The complexity of the neural network $\hat{f}$, approximating the state map of the process f, was estimated together with the network weights in a large optimization batch job.

Initially, we have tried all optimization methods discussed in Chapter 3 to find a proper set of weights for this process. The main problem was to capture by a model the dynamic of the load swing as this should be later on suppressed by the controller. As the impulse responses of the second output show faster dynamic than the impulse responses of the first output, we think that the optimized cost function landscape is characterized by curved valleys. This is a typical example when gradient methods show slow convergence and stochastic search methods are time expensive.

Finally, we have performed a number of optimization experiments on this problem. Each optimization was carried out for different complexity of the network while for each networks complexity we performed three optimizations started from different initial points. The cost function was optimized first by by simulated annealing procedure to eliminate local solutions with very high values of the cost function and then followed by the quasi-Newton method. Just for curiosity the CPU time consumed by this optimization was 144 hours 33 minutes and 36 seconds on the DEC 3000 workstation.

The results of these experiments are shown in the Table 4.2. In this table are shown cost function values for both the estimation data set J_e and the validation data set J_v. The number of nodes in the hidden layer of the neural network and the corresponding number of weights is also shown in this table. For each complexity of the neural network we performed three different minimizations. The dash "-" symbol denotes cost function values bigger than 100, considered as very poor local minima.

In the Figure 4.11 we show the learning curves obtained from the quasi-Newton optimization. We can observe that the optimization is getting stuck mainly at two different values of the cost function, either 100 or about 4. The first value corresponds to the situation with a good fit of the second output and much worse fit of the first one. This happens due to the fact that the impulse response of the second output is shorter than the impulse response of the first output, as already mentioned.

Table 4.2: Neural net optimization results: N_{N1}-number of nodes, n_{θ_f}-number of weights, $N\,J_e$-a scaled value of the estimation cost function, J_v-a scaled value of the validation cost function

N_{N1}	n_{θ_f}	$N\,J_e$			$N\,J_v$		
6	96	6.81	74.85	7.57	15.61	80.91	-
7	111	0.73	96.46	0.76	2.00	-	2.34
8	126	0.59	4.82	8.09	1.72	21.54	-
9	141	60.96	4.43	0.23	-	-	0.75
10	156	75.39	0.15	6.91	-	0.41	47.22
11	171	5.13	7.30	3.67	-	43.55	-
12	186	3.69	0.07	0.16	-	0.18	-
13	201	3.55	0.02	0.13	-	0.11	0.69
14	216	3.72	3.55	83.56	35.97	-	-
15	231	42.68	62.95	3.54	-	-	-

The second value, where the optimization is getting stuck corresponds to the situation of again a very good fit of dynamics of the second output while only slow dynamics of the first output are reasonably good fitted. At this point the unmodelled part of the process dynamics corresponds to the load swing. As the total power of the output is about 8, the approximation error, evaluated by (4.10), gets below 1% for most of the cases. But to model properly also the load swinging it turns out that we need to reach much better performance as the contribution of the load swing is masked by process nonlinearities.

From the Figure 4.11 can be seen that the neural network learning procedures with the best performances did not reach a minimum yet. We therefore continued the minimization from the best result, that was the configuration with 13 nodes and a cost 0.02, for an additional $500n_w$ iterations which consumed 36 hours and 58 seconds of the CPU time. After this experiment we found the performance of the model on estimation data 3.19e-03 and on validation data it was 1.53e-02. In the Figure 4.12 are shown spectra of final output errors on the estimation data set. Even these errors are very low there are still small approximation errors at low frequencies and in case of the first output x_l the spectrum shows increased errors around the resonance frequency of the load swinging. For a better validation of the model we evaluated also impulse responses of the estimated model. In the Figure 4.13 are shown only the two most interesting ones out of four possible. In the top left plot are shown impulse responses from x_{ref} input to x_l output and in the top right plot are shown impulse responses from y_{ref} input to y_l output. As the difference between the true impulse response and the model impulse response are hardly visible, below these two plots are shown corresponding errors.

For a comparison, we estimated also a 6th order linear state-space model of

the process parametrized by (A, B, C) triple. The value of the cost function we found here was 9.18. The performance of the best linear model we found is shown in Figure 4.14. Actually, the impulse response test does not excite the process nonlinearities sufficiently, and the linear model performs here quite well on this test.

4.5.3 Discussion

The dynamics of the gantry crane process is composed of both fast dynamics, caused by the pendulum swing and slow dynamics, caused by the movement of the trolley. A numerical estimation of the simulation model for this process, parametrized by a neural network, was found to be a difficult task, mainly because of enormously time consuming minimization procedures. To speed-up the minimization we have considered the following issues:

- Different optimization methods. Here we have experienced, that the optimization will remain difficult as we are probably dealing with error landscapes composed of multi-dimensional valleys. These are probably caused by the fact that the impulse response of the second output is shorter than the impulse response of the first output and that the contribution of the load swing to the final error is rather small.

- Initial guess of weights. We have also used as an initial guess of weights of an estimated linear state-space model which, of course, were properly scaled and biases adjusted such that the linear portion of the sigmoidal function was used by the neural network. This approach led only to a marginal improvement as the weights of the neural network stayed in the neighbourhood of the initial guess.

- A parallel combination of a linear model with the neural network. We have also parametrized the model as a parallel combination of a linear model estimated around the system equilibrium point (4.50) and a neural network. We have observed, when training the neural network, that the neural network tried to compensate for the parallel linear model and then it tried to fit the process dynamic.

- Non-uniform sampling time. One of the problems, why the neural network training is time consuming, is the excessive number of data points used for identification as the prosess dynamics have a broad range. To cope with this problem we sampled the process inputs and outputs non-uniformly, using a random sampling time. The sampling time intervals were chosen as uniformly distributed random numbers from an interval $\langle 0.1, 1 \rangle$ seconds. The state map $\hat{f}$ was extended for an extra input and that was the sampling time. In this way we reduced the length of the data set what led to a better conditioned optimization problem but as we validated the estimated model with a constant sampling time this model did not outperform the one being

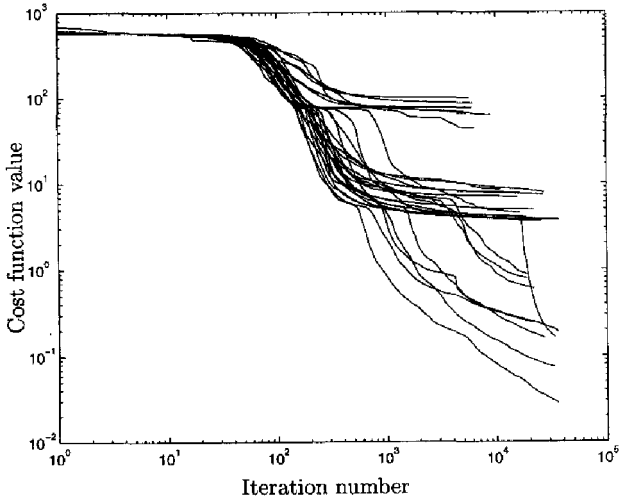


Figure 4.11: Quasi-Newton optimization

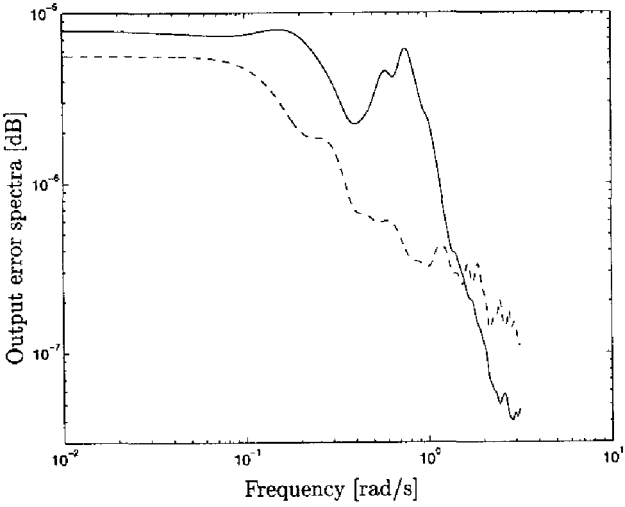


Figure 4.12: Estimated output error spectra: solid line - x_t, dashed line - y_t

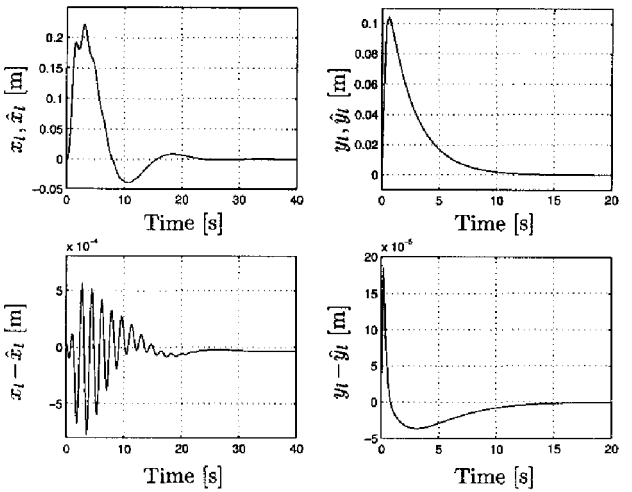


Figure 4.13: Process versus estimated neural model impulse responses

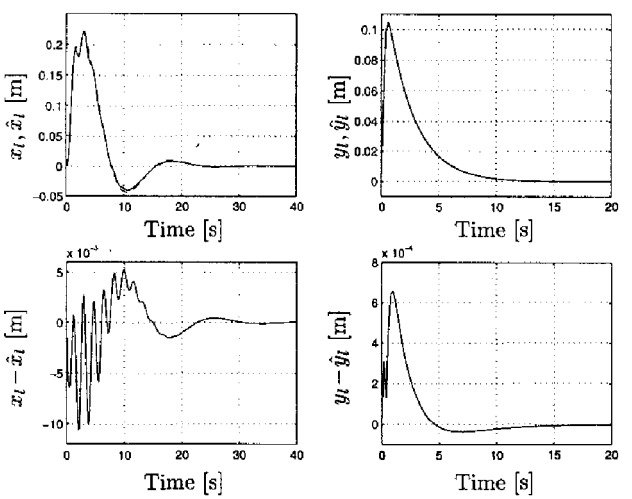


Figure 4.14: Process versus estimated linear model impulse responses

estimated specially with this constant sampling time. It was also not clear, how such a variable sampling time model could be used for the controller design.

- Variable sampling time for states. Another solution to the problem how to deal with broad process dynamics could be in using different sampling time for the internal states of the model. Preliminary tests were done in [4] but these were not completed due to the lack of computing power. The advantage of this approach could be that a controller design for this model makes more sense than in the previous case.

- Periodic rescaling of neural network weights. Often we could observe that some of the nodes of the neural network saturated during the optimization in the flat reagion of the node activation function. This caused a slow down of the optimization due to the bad numerical properties of the model. To minimize these problems we have periodically monitored the inputs and outputs of the activation function $s(\nu_i^l(k)) = \lambda_i^l(k)$ of all nodes, that is for all i and l and for all k. We have computed a variance of the node outputs $\sigma_{\lambda_i^l}$, a minimum value of the node input $\underline{\nu}_i^l = \min|u(k)|$ and a maximum value of the node input $\underline{\nu}_i^l = \max|u(k)|$ for all k. A node saturation was detected as a small variance of the node output $\sigma_{\lambda_i^l}$ and $\overline{\nu}_i^l$ was large. Then the weights corresponding to this node were rescaled as follows:

  1. the bias of the node

  $$^{\text{new}}\theta_{b_i}^{\ l} = \theta_{b_i}^{\ l} - (\overline{\nu}_i^l - \underline{\nu}_i^l)/2$$

  2. the weights

  $$^{\text{new}}\theta_{w_{ji}}^{\ l} = \theta_{w_{ji}}^{\ l}/(5(\overline{\nu}_i^l - \underline{\nu}_i^l))$$

  for $j = 1, \ldots, N_{Nl}$

  3. the biases of next layer nodes

  $$^{\text{new}}\theta_{b_i}^{l+1} = \theta_{b_i}^{l+1} + s((\overline{\nu}_i^l - \underline{\nu}_i^l)/2)$$

  4. the weights at the output of the node

  $$^{\text{new}}\theta_{w_{ij}}^{l+1} = \theta_{w_{ij}}^{l+1}(5(\overline{\nu}_i^l - \underline{\nu}_i^l))s'((\overline{\nu}_i^l - \underline{\nu}_i^l)/2)/s'(0)$$

  for $j = 1, \ldots, N_{Nl+1}$

4.6 Summary

In this chapter we discussed both black-box and grey-box neural models. Black-box models were considered both in I/O configuration and in state-space configuration. We proposed the state-space configuration as being a better form for including

a priori process knowledge into the model. Therefore the grey-box models are considered exclusively as state-space models.

Next the state-space model of the process is completed with a nonlinear filter gain, similarly to a Kalman gain from the linear estimation theory. This filter stands for improved state predictions by the simulation model due to the fact that the process is subject to non-observed disturbances and measurement noise. We have considered both single-stage ahead state estimates and current-stage estimates.

The unknown nonlinearities are parametrized by neural networks whose weights are estimated by numerical minimization of a quadratic error function. For all proposed modelling structures we gave formulas for the evaluation of gradients.

In this chapter we have trated two numerical examples. The first one was intended to show a different performance obtained from an optimized prediction model and an optimized simulation model. The second one was originally used to test different optimization methods for their feasibility for neural network training. It shows a typical set-up of an identification problem in a state-space domain using neural networks.

5 *Neural State Transition Control*

In this chapter we will discuss in detail our approach to the solution of the transition control problem as it was stated in Chapter 2 by the Problem 2.1. In this chapter we will tackle this control problem in a discrete-time state-space domain, as the controller will be realized by a digital computer.

Before we start addressing all issues involved in the controlled design, we discuss a general nonlinear state-space control problem and give an academic example of control of a multi-link inverted pendulum. A static neural network will be used to approximate the nonlinear state feedback. After a discussion of a transition controller design, applicable in process control, we give a more realistic example to illustrate some of the ideas of the transition controller design. The proposed transition controller design algorithm will be fully demonstrated in the last chapter of this thesis.

5.1 Operating point changing

Theoretically "operating point changing" means steering the system to another equilibrium point. Practically that means changing values of process variables, e.g. pressures, temperatures, flows, speeds etc., to new values such that a product with different properties can be produced or new functionality is attributed to the process. If the system is linear, this operation presents no problem because of a global character of the system properties. However, in nonlinear systems, this type of operation has many unsolved problems, which include:

- problems of capturing and exploiting process nonlinearities;
- stability of the closed loop along transition trajectories;
- level of optimality of the transition;
- robustness with respect to process disturbances;
- process conditions and constraints handling.

In general, the operating point changing type of control belongs to the most complex control problems, mainly because we have to go through nonlinear regions of the process dynamic. If the transition is done manually, it requires a lot of understanding of the process functionality and behaviour. This knowledge is usually obtained from the first physical, chemical, mechanical or other principles. The actual operating point change is then carefully scheduled through a sequence of intermediate equilibria by slow manipulation of process set-points and possibly also by sequentially switching between in advance designed linear controllers computed using linearized process dynamic along the transition path. If we would be able to model the process dynamics and to design a single nonlinear controller the operating point change could possibly be done in a faster, simpler, smoother, more reliable and effective way.

There have been many studies dealing with nonlinear system control problems, but it seems that there is no conclusive method. To control a nonlinear system we often use the "Optimal control theory" as a general method for treating nonlinear control problems to devise concrete algorithms for a specific control problem. Unfortunately, a vast amount of numerical calculations is required in this approach with many trials and errors. We will demonstrate on numerical examples that a careful set-up of the controller synthesis procedure is important to eliminate these problems to a certain extent.

5.2 General considerations

Let a sampled version of the controlled nonlinear dynamic system Σ_s be now given by

$$x(k+1) = f[x(k), u(k), w(k)] \tag{5.1a}$$
$$y(k) = h[x^1(k), u(k)] + v(k) \tag{5.1b}$$

with a similar notation as in (2.3) on page 10 and $x^1(k)$ denotes the known part of the state vector as proposed in (4.29). The system (5.1) can be seen as a model of the process for which we want to design the controller.

Let the control system attached to the system (5.1), see also Figure 2.1, be described by the following nonlinear state-space dynamic system

$$z(k+1) = \alpha[z(k), x(k), r(k)] \tag{5.2a}$$
$$u(k) = \beta[z(k), x(k), r(k)] \tag{5.2b}$$

where α and β are smooth nonlinear functions, $z(k) \in \mathbb{R}^{n_z}$ is the state of the controller, n_z is the dimension of the reference and $r(k) \in \mathbb{R}^n$ is a reference signal. The controlled system state $x(k)$ and the reference signal $r(k)$ represent the controller input and $u(k)$, the controller output becomes the controlled system input. A composition of (5.1) and (5.2) is shown in Figure 5.1 and is shortly called a dynamic state feedback. The nonlinear functions α and β should be chosen such that the system state $x(k)$ will follow $r(k)$, preferably as good as possible.

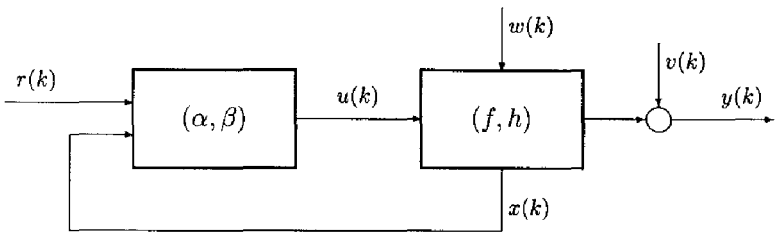


Figure 5.1: Dynamic state feedback

The proposed controller (5.2) includes a regular static state feedback just by assuming in (5.2a) that $z(k+1) = z(k)$ for all k. But what is more important it also includes the idea of adding integrators into the state feedback. In this case the function α could be defined as

$$\alpha[z(k), x(k), r(k)] = z(k) + (x(k) - r(k))$$

provided that the reference signal $r(k)$ is a state reference signal. If the dynamics of the controller are fixed, in other words α is fixed, only the output map of the controller β has to be designed. In such cases we consider only static neural networks used for approximation of the controller. This also makes the controller design easier than using a dynamic neural network for the controller parametrization.

5.3 Example: A multi-link inverted pendulum

To check the feasibility of using neural networks for a parametrization of state-feedback controllers we took as a test example a problem of swinging-up a multi-link inverted pendulum. We took a rather strong assumptions: the state of the system is fully available for the feedback, the system dynamics are fully known and there are no disturbances acting in the system.

The inverted pendulum is considered as a two dimensional mechanical system of $n_l + 1$ degrees of freedom shown in Figure 5.2. A frictionless trolley of mass M moves in the horizontal direction under the action of a force F_t. We attached to the trolley a series of n_l ideal links each of a length $l_1, l_2, \ldots, l_{n_l}$, and a mass $m_1, m_2, \ldots, m_{n_l}$. Each joint is assumed to be a frictionless revolute hinge. The system is at any moment described in terms of $n_l + 1$ coordinates $(x_t, \xi_1, \ldots, \xi_{n_l})^T$ and their first-order time derivatives. The position of the trolley with respect to some freely chosen reference point is denoted by x_t and ξ_i is an angle of the base of the ith link against the vertical axis, measured positive in the clockwise direction.

Let us assume in the following an equal length of all links, $l_i = l$ and and also equal mass of all links $m_i = m$, for $i = 1, \ldots, n_l$. Then the total energy of the

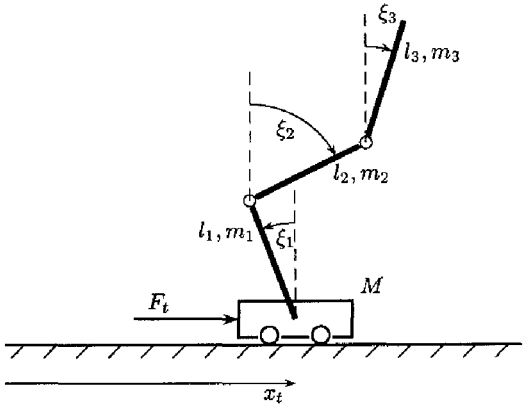


Figure 5.2: Three-link inverted pendulum

system, given as a sum of the kinetic and the potential energy is according to [28] given by the system Lagrangian as follows

$$
\begin{aligned}
&L(x_t, \xi_1, \ldots, \xi_{n_l}, \dot{x}_t, \dot{\xi}_1, \ldots, \dot{\xi}_{n_l}) = \\
&\quad \frac{1}{2}\left(M + n_l m\right)\dot{x}_t^2 + \\
&\quad \frac{1}{2}\sum_{k=1}^{n_l}\Big((1/3 + n_l - k)ml^2\dot{\xi}_k^2 + (1 + 2(n_l - k))ml\cos(\xi_k)(\dot{x}_t\dot{\xi}_k - g)\Big) + \\
&\quad \frac{1}{2}\sum_{k=1}^{n_l-1}\sum_{j=k+1}^{n_l}(1 + 2(n_l - k))ml^2\cos(\xi_j - \xi_k)\dot{\xi}_j\dot{\xi}_k
\end{aligned}
\tag{5.3}
$$

Equations of motion of the system are then given by the Lagrange method as

$$
\begin{aligned}
\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} &= F_t \\
\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\xi}_k}\right) - \frac{\partial L}{\partial \xi_k} &= 0 \qquad\qquad k = 1, \ldots, n_l
\end{aligned}
$$

Substituting (5.3) into the equations (5.4a) and (5.4a) yields for the x_t coordinate an equation

$$
\begin{aligned}
\left(M + n_l m\right)\ddot{x}_t + \frac{1}{2}\sum_{k=1}^{n_l}(1 + 2(n_l - k))ml\cos(\xi_k)\ddot{\xi}_k = \\
F_t + \frac{1}{2}\sum_{k=1}^{n_l}(1 + 2(n_l - k))ml\sin(\xi_k)\dot{\xi}_k^2
\end{aligned}
$$

and for the coordinate $\xi_k,\ k = 1, \ldots, n_l$ yields an equation

$$(1/2 + n_l - k)ml\Big(\cos(\xi_k)\ddot{x}_t + \sum_{j=1}^{k-1} l\cos(\xi_k - \xi_j)\ddot{\xi}_j\Big)$$
$$+ (1/3 + n_l - k)ml^2\ddot{\xi}_k + l^2 m \sum_{j=p+1}^{n_l} (1/2 + n_l - j)\cos(\xi_j - \xi_k)\ddot{\xi}_j$$
$$= l^2 m \sum_{j=k+1}^{n_l} (1/2 + n_l - j)\sin(\xi_j - \xi_k)\dot{\xi}_j^2$$
$$- (1/2 + n_l - k)ml\Big(l\sum_{j=1}^{n_l}\sin(\xi_j - \xi_k)\dot{\xi}_k^2 - g\sin(\xi_k)\Big)$$

These equations are used for simulation of this system. From now on let $M = 1,\ m = 0.1,\ l = 1$ and $g = 10$.

A mathematical model of the above described system can be brought into a nonlinear state-space dynamic form affine in $u(t)$ and given as follows

$$\dot{x}(t) = f_1[x(t)] + f_2[x(t)]u(t) \tag{5.5}$$

where $x \in \mathbb{R}^{2n_l+2}$ is the state vector of the system consisting of the position and speed of the trolley and angles and angle speeds of links attached to he trolley, that is

$$x = (x_t, \dot{x}_t, \xi_1, \dot{\xi}_1, \xi_2, \dot{\xi}_2, \ldots, \xi_{n_l}, \dot{\xi}_{n_l})^T$$

and $u \in \mathbb{R}$ is the control input representing the force F_t applied on the trolley. An exact analytical form of both f_1 and f_2 becomes very complicated for systems with more than two links, because of the necessity of inversion of a $n_l + 1$ dimensional matrix. To evaluate f_1 and f_2 in (5.5) analytically for different number of links of the pendulum we have used the MAPLE software package for symbolic computations. The final computer program code was optimized for a minimum of floating point operations also by this package.

With respect to this system we formulate the control problem as a synthesis of such a nonlinear state-feedback control law

$$u(t) = \beta[x(t), \Theta_\beta]$$

which satisfies

$$-u_{max} \le u(t) \le u_{max}$$

and which brings the system from a given initial equilibrium point

$$x(t_0) = (0, 0, \pi, 0, \pi, 0, \ldots)^T$$

that is $\xi_i = \pi$ for all i and the rest of the state vector components are zero, to the final zero equilibrium point

$$x(t_f) = (0, \ldots, 0)^T$$

in a finite time, possibly small. The controller nonlinearity β is approximated by an MLP with weights Θ_β. As the control signal is required to be bounded we simply define the output node of the neural network to be nonlinear, namely $u_{max} \tanh(.)$.

An approximate solution to the above stated control problem can be obtained by a direct numerical minimization of the following cost functional

$$J(\Theta_\beta) = x(t_f)^T x(t_f) \tag{5.6}$$

with respect to the weights of the neural network Θ_β. The final time t_f is chosen in advance. To compute the value of the criterion (5.6) we integrated the closed-loop system numerically using the Heun method (A.6). We estimated the controller for a one, two and three link inverted pendulum.

After completing many optimization experiments, we have found a couple of solutions of a swing-up of the one-link and two-link inverted pendulum. In case of the three link inverted pendulum the computing costs became so high that we were not able to complete the full optimization of the controller due to the lack of computing resources. Most of the solutions were showing an irregular control and state trajectories. The type of the solution found during the neural network training was determined mainly by the length of the simulated state trajectory of the closed-loop system, that is by the choice of the final time t_f. The integration step size τ, used in the Heun method (A.6) to integrate the closed-loop system, defined then the accuracy of the integration and consequently a discrete-time system for which the controller is computed. Basically, we started the optimization of the controller with a sufficiently large final time to be sure that a solution exists. When we found a solution we decreased the final time by some factor, usually a few percent, and we continued the optimization of the neural network using the last set of weights. In this way we tried to optimize the control interval to the minimum. We have also observed that starting the neural network training using random initial weights and the minimal control interval did not lead to a solution. A logical explanation is that in this case we have to find a very particular solution while on longer control intervals there exist more solutions to this control problem which are easier to find. Note also, that due to the non-convexity of the problem not every initial solution on a longer time interval led to a minimal solution while using the strategy of gradual decrease of the the final time t_f. Usually, we ended up this procedure when we found a few similar solutions showing about the same type of behaviour and resulting in about the same minimal control interval. This provided us with certain level of confidence in the obtained solution.

While experimenting with this control problem we propose a new criterion explaining the control objective and that is a minimization of the kinetic energy of this system with simultaneous maximization of the potential energy of the system in the final equilibrium point. Geometrically, this means the pendulum keeps

staying in an up-right position. Mathematically, the new control objective can be formulated as a minimization of a modified Lagrangian of this system $\bar{L}$, where $\bar{L}$ is L given by (5.3) where the minus sign next to the gravity term g is reversed. If we also require that the position of the trolley x_t and the control input u in the final point must be zero, we have to solve a nonlinear constrained optimization problem. To do this we used the penalty function approach converting the constrained optimization problem to an unconstrained one. Then the control objective is given as follows

$$\min_{\Theta_\beta} \quad \bar{L}(x(t_f)) + \rho_k(x_t(t_f)^2 + u(t_f)^2), \qquad \rho_k > 0, \qquad \rho_k \to \infty \tag{5.7}$$

with respect to the system dynamics (5.4).

One link inverted pendulum results

This is the easiest situation and $n_l = 1$. To parametrize the controller we have chosen a neural network with one hidden layer consisting of 8 sigmoidal nodes. The integration step-size was chosen $\tau = 0.01$, the final time $t_f = 1.6$ and $u_{max} = 20$. The controller neural network was optimized by minimization of (5.7) by performing 40,000 simulated annealing iterations followed by 100,000 quasi-Newton iterations. Some of these results are shown in Figure 5.3. In Figure 5.4 are shown the final time optimized state and control trajectories. These were found by taking the fourth solution from the previous experiment and gradually decreasing the final time t_f during the neural network training to its final value $t_f = 1.18$.

Two-link inverted pendulum results

In this case is $n_l = 2$. The complexity of the neural network was chosen as 6 nodes in one hidden layer resulting in 49 weights. The final time was chosen $t_f = 2$, the integration time was $\tau = 0.005$ and $u_{max} = 40$. As a maximum number of simulated annealing iterations we chose $2,000,000$ and as a maximum of quasi-Newton iterations we chose $100,000$. Figure 5.5 shows five solutions of this control problem. In the first column is shown $\dot{x}_t$ against x_t, in the second column is shown ξ_2 against ξ_1, in the third column is shown $\dot{\xi}_2$ against $\dot{\xi}_1$ and in the fourth column is shown u against the time t. These are typical solutions when tackling this problem by a numerical unconstrained optimization of a black-box neural network controller. The last of these solutions, shown at the bottom of Figure 5.5, seems to give an acceptable solution to the problem and could possibly be further optimized, to obtain a smaller final time t_f.

Three-link inverted pendulum results

In this case $n_l = 3$. The complexity of the neural network was chosen as 8 nodes in one hidden layer. The final time was chosen $t_f = 2$, the integration time was $\tau = 0.005$ and $u_{max} = 40$. We have optimized in this case in total 81 weights

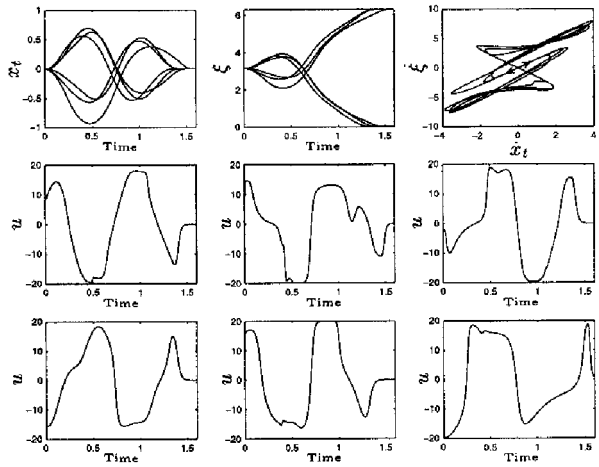


Figure 5.3: Swing-up of a one link inverted pendulum

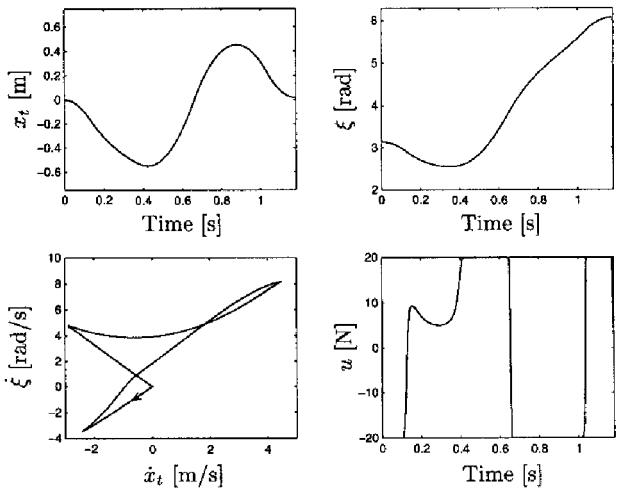


Figure 5.4: Swing-up of a one-link inverted pendulum with optimized final time t_f

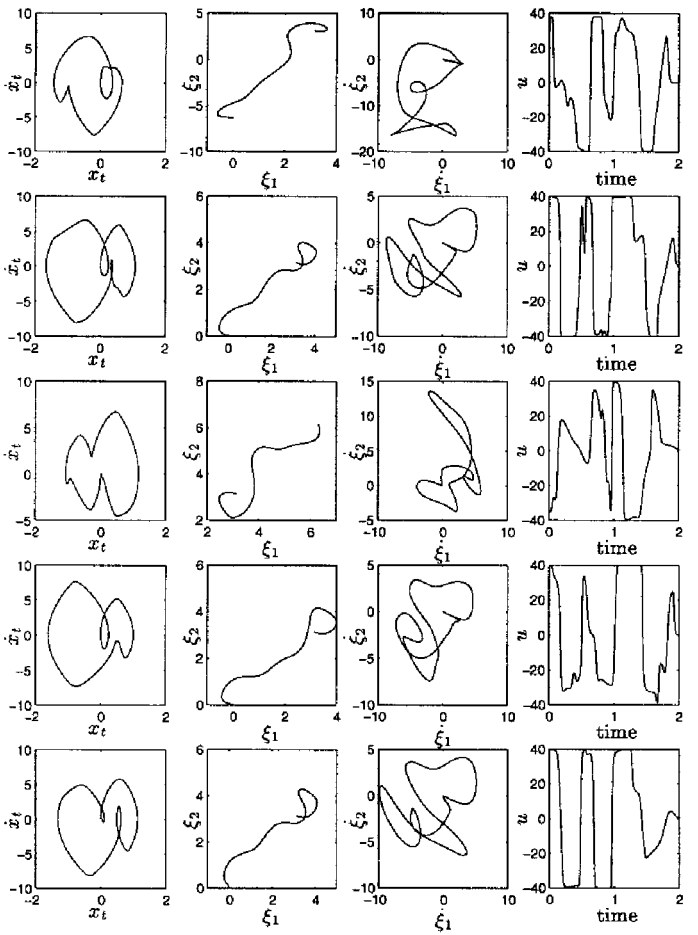


Figure 5.5: Two-link inverted pendulum swing-up results

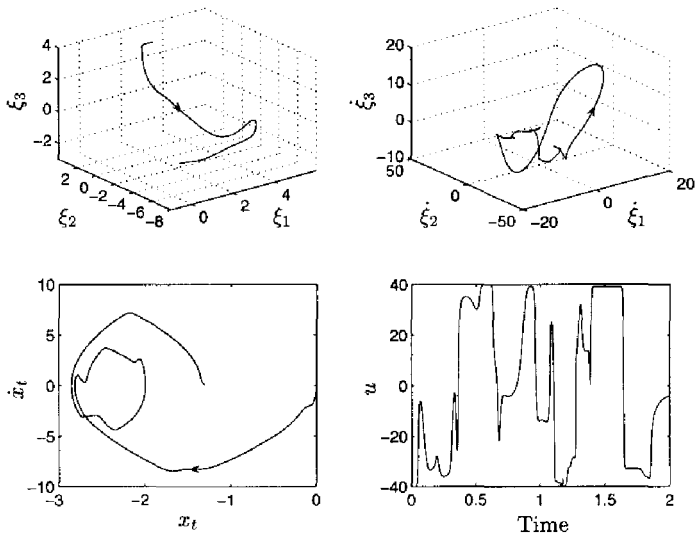


Figure 5.6: Three-link inverted pendulum swing-up results

by performing 100,000 simulated annealing iterations followed by 200,000 quasi-Newton iterations. The best solution we have found is shown in the Figure 5.6. We can see that the pendulum is approaching the final position, in this case $\xi_1 = 0$, $\xi_2 = -\pi$, $\xi_3 = 0$, but is not yet there exactly. The penalizing factor ρ_k in (5.7) was chosen very small, just to make the problem easier, resulting in a final position error of the trolley.

Comments

In this test example we have found out that the choice of the optimized control cost function can be quite crucial for an easiness of solvability of this control problem. The minimization of an energy function of the system (5.7) led to a solution in less iterations than minimization of the usual quadratic function (5.6). Intuitively, this can be explained by a more proper match of the energy function to the system dynamics compared to a general quadratic function. Another reason of a better convergence when using the energy function may also be found in the fact, that when optimizing the quadratic function (5.6), we are searching for a very particular solution, while a minimization of the energy function allows all modulo 2π solutions for link angles, so that we have many more global minima.

Notice, that all solutions we have found are valid only for a very specific state initial point. Validation of the controller with a slightly perturbed initial state

condition did not result in a swing-up of the pendulum in any case. To make the controller more robust with this respect we minimized a new cost function consisting of a sum of cost functions of type (5.7), but each one computed by starting the simulation of the closed-loop from different initial state condition, close to the downwards position of the pendulum. Though these results are not shown here, this should be a more proper way of design of a controller in this example.

We did some preliminary work to implement a neural network controller for a real inverted pendulum. The main problems we have to deal with include:

- The system state is not fully observable. We can only measure the angles and the speeds must be reconstructed from measurements.
- The links and joints are not ideal anymore and extra dynamics have to be considered. These dynamics include coulomb and viscous friction terms which represent non-differentiable dynamics.
- The presence of non-differentiable dynamics makes the numerical optimization of either (5.6) or (5.7) using gradient methods impossible due to the lack of a sufficient smoothness of the minimized function.

5.4 Controller design considerations

In the design of the transition controller we will assume that

1. the process dynamics are only partly known;
2. the state of the process is not measured;
3. the process is subject to process disturbances and measurement noise.

These assumptions immediately suggests that we will have to estimate a model of the process dynamics. In the Chapter 4 we proposed a grey-box state-space modelling concept (Section 4.2 on page 63) which will be followed also in this chapter. The unknown state of the process will be reconstructed by a design of a neural state observe while following the concepts proposed in Section 4.3 on page 69. The remaining part of the complete transition controller design consists of a design of the state feedback. The issues concerned with this design are discussed next.

5.4.1 Process model

The state feedback shown in the Figure 5.1 assumes direct measurements of the full state vector. However, in practice, the state of the controlled process is most of the time not measured directly, at least not the whole state vector. To deal with this problem we reconstruct a state of the process from measured data using the

neural state-space observer proposed by (4.37). Again, the state-space observer is composed of an estimated simulation model of the process $\hat{f}$ and a filter gain $\hat{g}$, both approximated by static neural networks. That is

$$\hat{x}(k+1) = \hat{f}[\hat{x}(k), u(k), \Theta_{\hat{f}}] + \hat{g}[y(k), \hat{y}(k), \Theta_{\hat{g}}] \quad (5.8a)$$

$$y(k) = h[\hat{x}^1(k), u(k)] \quad (5.8b)$$

As the design of the controller is carried on off-line, what means that $y(k)$ is not available, the controller will be designed for the model (5.8) and later, on during the actual controller implementation, the model will be replaced by the real process.

5.4.2 Controller objective

In this section we will refer to the control problem state in Chapter 2 on page 11.

Let the controlled system Σ_s model be given by

$$\hat{x}(k+1) = \hat{f}[\hat{x}(k), u(k), \Theta_{\hat{f}}] \quad (5.9a)$$

$$y(k) = h[\hat{x}^1(k), u(k)] \quad (5.9b)$$

Recall, that h is a known nonlinear function and $\hat{f}$ stands for a static MLP $\hat{f}_N$ combined with an a priori known analytical part $\bar{f}$.

Let us assume that

$$\mathbb{E} := \{\mu \in \mathbb{R}^{m+p+n},\ \mu = (u_e, y_e, x_e) \mid x_e = f[x_e, u_e, 0], y_e = h[x_e^1, u_e]\}$$

be a finite set of equilibrium points of the system (5.9) and x_e^1 denotes the known part of the equilibrium state with respect to the state partitioning proposed by (4.29). Let us define a set of all trajectories of the system (5.9) by

$$\mathcal{T} := \{(u(k), y(k), x(k)) \in \mathbb{R}^{m+p+n},\ k \in \mathbb{Z}^+ \mid (5.1) \text{ holds}\}$$

while assuming that k evolves on a finite time interval $\langle t_0, t_f \rangle$. Let t denotes an element of $\mathcal{T}$. With respect to the equilibrium set $\mathbb{E}$ of the system (5.9) we denote by $\mathrm{t}_{i,j} \subset \mathcal{T}'$ those trajectories, which start at time t_0 at an equilibrium point μ_i, $\mu_i \in \mathbb{E}$ and terminate at time t_f, $t_f > t_0$ in an equilibrium point μ_j, $\mu_j \in \mathbb{E}$. Then $\mathcal{T}'$, $\mathcal{T}' \subset \mathcal{T}$ is defined by

$$\mathcal{T}' := \{\forall\, \mathrm{t}_{i,j} \in \mathcal{T},\ \mu_i \in \mathbb{E},\ \mu_j \in \mathbb{E},\ \mu_i \neq \mu_j\}$$

The trajectories $\mathrm{t} \in \mathcal{T}'$ will also be called transitions or state transitions.

The controller objective was defined in Chapter 2 by a criterion (2.8) on page 12. We will discuss now the two most interesting versions of this criterion.

Time-optimality

It is quite logical to require a time optimality from a transition. For instance, in the case of the fluidized bed polymerization reactor, the production during the

transition between different operating points is either a wide-specification product or off-specification product. A faster transition will also mean less losses of the production. This requirement can be translated into the following criterion

$$J(\mathrm{t}) = t_f - t_0 \tag{5.10}$$

where $\mathrm{t} \in \mathcal{T}'$ and $\langle t_0, t_f \rangle$ is the control interval. initial time can often be considered zero and then the criterion (5.10) reduces to the length of the control sequence $u(k)$, in discrete time. To be consistent with (2.8) on page 12 we can choose either $\Psi = t_f,\ L = 0$ or $\Psi = 0,\ L = 1$ to obtain the criterion (5.10).

Minimum energy

Besides the transition time we also would like to optimize other process conditions. Often the transition can be a priori prescribed by reference trajectories of physically defined states and/or outputs of the process. These states should follow these trajectories as close as possible. Often we want to minimize also the control effort. In case of the polymerization reactor this can be, for instance, the amount of the catalyst fed into the reactor.

All these and similar requirements can be translated into the following criterion

$$J(\mathrm{t}) = \sum_{k=t_0}^{t_f} \left(\|\hat{x}(k) - r(k)\|_Q^2 + \|u(k)\|_R^2 \right) \tag{5.11}$$

where the norms are weighted Euclidean vector norms, Q is a semi-positive definite matrix and R is a positive definite matrix. The criterion (5.11) is evaluated along a single transition $\mathrm{t} \in \mathcal{T}'$ on a discrete-time interval $\langle t_0, t_f \rangle$. In the criterion (5.11) we assume a reference signal $r(k) \in \mathbb{R}^n$, defined for all states of the model. From the physical point of view, the reference signal $r(k)$ can only be defined for $\hat{x}^1$ part of the state vector. The choice of the reference signal for the black states $\hat{x}^0$ is a subject of the design and will be discussed later on.

Often we are interested in having an integral action in the state feedback to guarantee zero final tracking errors. Then we have to consider the following criterion

$$J(\mathrm{t}) = \sum_{k=t_0}^{t_f} \left(\|\hat{x}(k) - r(k)\|_Q^2 + \|\Delta u(k)\|_R^2 \right) \tag{5.12}$$

where $\Delta u(k) = u(k) - u(k-1)$ and $\Delta u(t_0) = 0$.

5.4.3 Constraints

The minimization of either (5.11) or (5.12) is subject to a set of constraints introduced already in (2.8) on page 12. The first obvious constraint is given by the

model (5.9) defining the process dynamics and therefore also the set of all possible trajectories of the system T.

The initial state uncertainty constraint (2.9) on page 13 is handled by considering in the control criterion different realizations of the transition each of them starting at different initial points close to μ_0.

The final state constraint given by (2.10) can be easily handled by including a penalty term Ψ into the control criterion, e.g. $\Psi = \|\hat{x}(t_f) - \mu_f\|_2^2$. It can also be handled by the criterion (5.11) or (5.12) by defining a sufficiently long control interval and a reference value with $r(k) = \mu_f$ for $k \in \langle t', t_f \rangle$, where $t_0 < t' < t_f$.

The state and control constraints are often considered as simple bounds, constraining operating ranges of corresponding variables. These constraints are then given by

$$\underline{x} \leq x(k) \leq \overline{x} \tag{5.13a}$$

$$\underline{u} \leq u(k) \leq \overline{u} \tag{5.13b}$$

where $\underline{x} \in I\!R^n, \underline{u} \in I\!R^m$ are the lower bounds and $\overline{x} \in I\!R^n, \overline{u} \in I\!R^m$ are the upper bounds, respectively. The consideration of the constraints (5.13) is not only important for a proper process operations but also to define the model validity ranges.

Other type of constraints, often met in practice, are the limited rates of change of process variables, given as follows

$$\underline{\Delta x} \leq \Delta x(k) \leq \overline{\Delta x} \tag{5.14a}$$

$$\underline{\Delta u} \leq \Delta u(k) \leq \overline{\Delta u} \tag{5.14b}$$

where $\underline{\Delta x} \in I\!R^n, \underline{\Delta u} \in I\!R^m$ are lower bounds and $\overline{\Delta x} \in I\!R^n, \overline{\Delta u} \in I\!R^m$ are upper bounds, respectively.

5.4.4 Reference signal

The reference signal is considered either as a constant value specifying the final equilibrium point of the system or as a time varying signal specifying also the path in the state space. Notice that the model (5.9) provides us only with a part of the state vector $\hat{x}(k)$ which has a physical interpretation and that the rest of the state vector is meaningless in this respect. Therefore a prior specification of a reference signal can be concerned from physical point of view only with x^1 state components. In practice, an operating point of the process will be given by specific values of inputs u_e and outputs y_e. Based on y_e we can determine the equilibrium values of x^1 state components knowing that $y_e = h[x_e^1]$ and h is known. Equilibrium values of other states, namely x_e^0 are given by the model equation $x_e = \hat{f}[x_e, u_e]$ corresponding to (5.9a), which has to be solved for the unknown x_e^0. It might happen that there is no feasible solution for x_e^0 as its value is bounded by the validity of the estimated neural network. Then a new model has to be estimated using either more data or different complexity of the model.

When defining the state reference path signal between two equilibrium points we have to consider the reachability of the state-space of the model. As the controlled system is being considered as a general nonlinear system it is hard to investigate this issue. As the estimated model is a grey-box model, with physically well defined part of the state vector, we can use our a priori knowledge about the process dynamics to define a transition path for the states x^1. For the rest of the state components, namely x^0, is the transition path obtained as a result of the controller optimization.

In the following we discuss three approaches of a choice of a reference signal.

Step signal

Let us assume that a new operating point of the system is required to be $\mu_f = (\hat{x}_e, u_e)^T = (\mu_f^x, \mu_f^u)^T$. A most straightforward way of defining a reference signal is using a step signal given by

$$r(k) = \mu_f^x, \quad \text{for } k = t_0, \ldots, t_f \tag{5.15}$$

where μ_f^x is a value of states in the final equilibrium point.

Filtered step signal

As an alternative to a simple step reference signal we often use a low pass filtered step signal, given by

$$r_i(k) = \mu_{0_i} + (\mu_{f_i} - \mu_{0_i}) \exp(k/k_i') \quad \text{for } i = 1, \ldots, \hat{n} \tag{5.16}$$

where $\hat{n}$ is the dimension of the model state vector $\hat{x}$, $k = t_0, \ldots, t_f$ is the discrete time and k_i' is a discrete-time constant chosen in advance. For the known part of the state vector $\hat{x}^1$ is this constant chosen with respect to the prior process knowledge. For the other states, namely $\hat{x}^0$, we can use a guess.

Optimized state trajectory

A better way, and also a more difficult way, to determine the reference trajectories is to use the optimal state, and possibly also control, trajectories, provided that we are able to solve the optimal control problem (2.1) in function space. That means that if we would be able to find an optimal control input $u^*(k)$ as a time function resulting in an optimal state $\hat{x}^*(k)$ trajectory, this trajectory could be used as a reference signal for a later on design of a state-feedback controller.

For this purpose, we have developed in [17] numerical techniques for solving nonlinear optimal control problems with control and/or state constraints.

5.4.5 The feedback structure

In general, the controller is assumed to be a dynamic state feedback (5.2). The controller dynamic will be from now on assumed as follows

$$z(k+1) = az(k) + \hat{x}^{1,a}(k) - r(k) \tag{5.17}$$

where we partitioned the $\hat{x}^1$ part of the state vector into two components

$$\begin{pmatrix} \hat{x}^{1,a} \\ \hat{x}^{1,b} \end{pmatrix}$$

This partitioning defines a set of states $\hat{x}^{1,a}$ which will be tracked with a zero final error and the other states $\hat{x}^{1,b}$ may be tracked with nonzero final errors. Assuming $\hat{x}^{1,a}(k) \in I\!R^{n^{1,a}}$, then the dimension of both $z(k)$ and $r(k)$ is assumed to be $n^{1,a}$.

The constant $a > 0$ is a constant equal to 1 in case of assuming pure integrators, but often chosen in our algorithm slightly smaller than one, e.g. $a \in \langle 0.9, 0.999 \rangle$, which improves a numerical stability of the controller optimization procedure. This will be demonstrated in Chapter 6 on a simulation example.

The nonlinearity of the controller β in (5.2) will be approximated by a static MLP and will be denoted as follows

$$u(k) = \hat{k}[z(k), \hat{x}(k), r(k), \Theta_{\hat{k}}] \tag{5.18}$$

where $\Theta_{\hat{k}}$ stands for the weights of the neural network.

The controller structure given by equations (5.17) and (5.18) can be also considered in a simpler form, for instance

$$u(k) = \hat{k}[\hat{x}(k), \hat{x}(k-1), r(k), r(k-1), \Theta_{\hat{k}}] \tag{5.19a}$$

$$u(k) = \hat{k}[u(k-1), \hat{x}(k), r(k), \Theta_{\hat{k}}] \tag{5.19b}$$

which is a sort of I/O dynamic parametrization of a dynamic state feedback.

5.5 Controller synthesis

The controller synthesis, which will now be described, is based on a separation of the state estimation and the controller design. This is a proven optimal controller synthesis for linear systems. In case of a general nonlinear system there is no evidence yet that this should be also an optimal controller synthesis. However, we adopt this principle also in the case of a nonlinear controller design while expecting only an approximation of an optimal synthesis which we do not know.

The structure of the controller is schematically depicted in Figure 5.7. The state feedback $\hat{k}$ to be designed takes as inputs a reference signal $r(k)$, an estimated state $\hat{x}(k)$ and an integrator state $z(k)$ computed by the block Σ_I representing the equation (5.17).

Prior to the actual numerical optimization of the neural network parameters Θ_k we compute the equilibrium set $\mathbb{E}$ of the estimated model (5.9) and we define a reference signal for a couple of transitions within this set. To compute $\mathbb{E}$ we have to solve the following set of nonlinear equations

$$\hat{f}[\hat{x}_e, u_e] - \hat{x}_e = 0 \tag{5.20}$$

It is clear, that to determine all equilibrium points of $\mathbb{E}$ would require to find all solutions of (5.20). The equilibrium value for the known part of state vector components x^1 will be given in practice and therefore the equation 5.20 will have to be solved only with respect to the black states $\hat{x}^0$. Let us assume for the next discussion that we have defined a couple of equilibrium points. If the reference signal was defined, e.g. by following the discussion in Section 5.4.4, then Θ_k is computed by minimization of the following criterion

$$J_T = \sum_{i,j} J(\mathrm{t}_{i,j}) \tag{5.21}$$

with respect to the weights of the controller neural network. Basically, we can simulate the closed loop for all $\mathrm{t}_{i,j}$, evaluate the criterion J_T together with its gradients with respect to Θ_k and then use any numerical minimization method to minimize this criterion. The function J is defined either by (5.12) or by (5.10) or by (5.11).

A block diagram of the closed loop set-up is depicted in Figure 5.7. As it was mentioned earlier, the design is carried out off-line. That means, that we can not dispose of the real output $y(k)$. A simulation of this closed loop can be, in principle, considered in three situations:

1. The filter gain $\hat{g}$ is completely omitted from the simulation of the closed loop.
2. The filter gain $\hat{g}$ is included into the closed loop with zero inputs, that means $e_{ref}(k) = 0$.
3. The filter gain $\hat{g}$ is included assuming $e_{ref} \neq 0$.

We can test the closed loop using a reference signal $e_{ref}(k)$ in place of the true innovation sequence $e(k) = y(k) - \hat{y}(k)$ in the filter gain. The statistical properties of the $e_{ref}(k)$ sequence (type of distribution, mean, variance) can be estimated from the state observer design results. As the estimation is based on a minimization of a least-squares type of criterion $e(k)$ shows most of the time a Gaussian distribution and therefore it is straightforward to generate an $e_{ref}(k)$ sequence.

As the minimization of (5.21) with respect to Θ_k is performed numerically it is useful to determine the required cost function gradients analytically. These are given in the following section. As this computation is quite involved and badly influenced by the simulation time of the closed loop, we have often used numerical evaluation of these gradients.

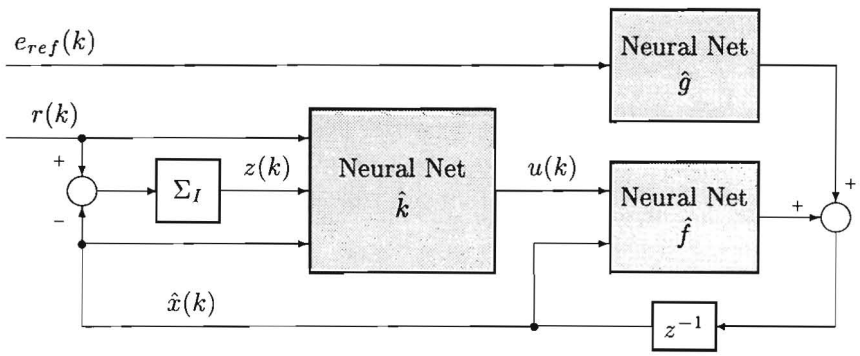


Figure 5.7: Controller set-up

The value of the parameter a in the controller dynamic (5.17) has been changed during the controller optimization sequentially starting from a value less than one, let us say 0.9, to a final value 1. This was done mainly due to obtain the stability of the closed loop. Notice, that the controller is estimated numerically and in general we can not expect a stable closed loop when taking random initial weights for the controller neural network $\Theta_{\hat{k}}$. As we minimize a cost function on a finte time interval, in general, there is no guarantee for stability of the closed loop. We have experienced on the polymerization reactor, that if we started the optimization from an unstable closed-loop system, it remained unstable, no matter how long the control interval was. As the control interval was always finite the controller neural network could not learn the stability of the closed-loop system. A choice of $a < 1$ helped us to start the optimization from a stable closed-loop system.

Training of the controller neural network is in fact done by an unconstrained minimization and all the constraints are handled by suitable choice of reference signals $r(k)$ and weighting matrices Q and R in (5.12).

5.6 Gradient computation

The gradient computation for a numerical minimization of (5.21) depends on the parametrization of the state observer, on the structure of the state feedback and also on the choice of the cost function. As an example we show here a gradient computation for only one situation. Let us assume a minimization of the following cost function

$$J(\Theta_k) = \sum_{k=1}^{N_c} \left(\|\hat{x}(k) - r(k)\|_Q^2 + \|\Delta u(k)\|_R^2 \right) \tag{5.22}$$

where $\Delta u(k) = u(k) - u(k-1)$ and let us assume state observer equations

$$\begin{aligned} \hat{x}(k+1) &= \hat{f}[\hat{x}(k), u(k), \Theta_{\hat{f}}] + \hat{g}[y(k), \hat{y}(k), \Theta_{\hat{g}}] \\ \hat{y}(k) &= h[\hat{x}^1(k)] \end{aligned}$$

and let us assume the state feedback be parametrized by

$$\begin{aligned} z(k+1) &= az(k) + \hat{x}(k) - r(k) \\ u(k) &= \hat{k}[\hat{x}(k), z(k), r(k), \Theta_{\hat{k}}] \end{aligned}$$

Then the gradient computation of (5.22) with respect to the weights of $\hat{k}$ is given by following formulas

$$\begin{aligned} \frac{\partial J(\Theta_{\hat{k}})}{\partial \theta_{\hat{k}i}} &= \sum_{k=1}^{N_c} (\hat{x}(k) - r(k))^T Q \frac{\partial \hat{x}(k)}{\partial \theta_{\hat{k}i}} + \sum_{k=1}^{N_c} \Delta u(k)^T R \frac{\partial \Delta u(k)}{\partial \theta_{\hat{k}i}} \\ \frac{\partial \Delta u(k)}{\partial \theta_{\hat{k}i}} &= \frac{\partial u(k)}{\partial \theta_{\hat{k}i}} - \frac{\partial u(k-1)}{\partial \theta_{\hat{k}i}} \\ \frac{\partial \hat{x}(k+1)}{\partial \theta_{\hat{k}i}} &= \mathcal{J}_{\hat{x}}^{\hat{f}}(k) \frac{\partial \hat{x}(k)}{\partial \theta_{\hat{k}i}} + \mathcal{J}_{u}^{\hat{f}}(k) \frac{\partial u(k)}{\partial \theta_{\hat{k}i}} + \mathcal{J}_{\hat{y}}^{\hat{g}}(k) \frac{\partial \hat{y}(k)}{\partial \theta_{\hat{k}i}} \\ \frac{\partial \hat{y}(k)}{\partial \theta_{\hat{k}i}} &= \mathcal{J}_{\hat{x}}^{h}(k) \frac{\partial \hat{x}(k)}{\partial \theta_{\hat{k}i}} \\ \frac{\partial u(k)}{\partial \theta_{\hat{k}i}} &= \frac{\partial \hat{k}[\hat{x}(k), z(k), r(k), \Theta_{\hat{k}}]}{\partial \theta_{\hat{k}i}} + \mathcal{J}_{\hat{x}}^{\hat{k}}(k) \frac{\partial \hat{x}(k)}{\partial \theta_{\hat{k}i}} + \mathcal{J}_{z}^{\hat{k}}(k) \frac{\partial z(k)}{\partial \theta_{\hat{k}i}} \\ \frac{\partial z(k+1)}{\partial \theta_{\hat{k}i}} &= a \frac{\partial z(k)}{\partial \theta_{\hat{k}i}} + \frac{\partial \hat{x}(k)}{\partial \theta_{\hat{k}i}} \end{aligned}$$

As we can immediately observe that an analytical gradient evaluation is rather involved. In our controller estimation experiments we often replaced this procedure by a numerical estimation of gradients using a finite difference approach

$$\frac{\partial J(\Theta_{\hat{k}})}{\partial \theta_{\hat{k}i}} = \frac{J(\theta_{\hat{k}i} + \epsilon) - J(\theta_{\hat{k}i} - \epsilon)}{2\epsilon}$$

where ϵ is a small number chosen usually in a neighbourhood of the square root of the machine precision. Our experience is that using numerically estimated gradients in training of neural networks it is better to set the value of ϵ in advance rather than to use a computer program based adjustment. The reason for this is that at the initial point, the computer estimated ϵ might be numerically optimal but later on in the course of the optimization it can become a very bad choice, the optimization is consequently getting stuck and requires a restart with a new value for ϵ.

5.7 Numerical example

In this section we give a simple numerical example to demonstrate the proposed controller design scheme using the methodology proposed in this chapter and in Chapter 4. Let us consider a nonlinear dynamic system described by the following state-space equations in discrete time

$$x_1(k+1) = \frac{x_1(k)}{1+x_2(k)x_2(k)} + u(k) + w_1(k) \tag{5.23}$$

$$x_2(k+1) = 1 + \frac{x_1(k)x_2(k)}{1+x_2(k)x_2(k)} + w_2(k) \tag{5.24}$$

$$y(k) = x_2(k) + v(k) \tag{5.25}$$

where $w_1(k)$, $w_2(k)$ and $v(k)$ are the process disturbances and the measurement noise, respectively, simulated as a zero mean uniformly distributed random sequences with maximal amplitude 0.1. To test the dynamics of this system we used as an input $u(k)$ a sequence of uniformly distributed, zero mean random samples with maximum amplitude 1. It means that nonlinearities of this system are excited in a very specific range. We generated a data set of $N = 2000$ input/output data points in a form of (4.1) on page 45.

The first step of the algorithm is an estimation of a state-space simulation model of the system from data while using the available a priori knowledge. To simulate the methodology proposed in Section (4.2), we assumed that we know that the second component of the system state vector x_2 is directly measured and we also assumed that the order of the process $n = 2$ is also known. Then the state vector of the model is defined as $\hat{x} = (\hat{x}_1, \hat{x}_2) = (\hat{x}^0, \hat{x}^1)$. The first component $\hat{x}^0$ stands for the black-box part of the model while the second component $\hat{x}^1$ represents the prior process knowledge. The model is then parametrized by (4.30) where the output map is given by

$$\hat{y} = \hat{x}_2$$

and the state map was parametrized by a neural network with one hidden layer for which the number of nodes was varied from 2 to 8. For each neural network configuration we have used one half of the data set to estimate the network's parameters $\Theta_{\hat{f}}$ and the rest of data was used for the validation of the estimated model. The results of these experiments are shown in Table 5.1. A good choice for the complexity of the neural network model of the system dynamics could be a configuration with 4 hidden nodes as configurations with more nodes tend to fit the noise in data.

In the second step of the algorithm we estimated a filter gain $\hat{g}$ introduced by (4.37) while we parametrized $\hat{g}$ by (4.42d). The nonlinearity of $\hat{g}$ was approximated by a neural network with one hidden layer containing 4 nodes and 22 weights. This choice was actually based on an optimization of a set of neural networks with different number of one hidden layer nodes while choosing the one with the best

Table 5.1: Output error identification results, N_{N1} is the number of nodes, n_θ is the number of estimated parameters, J_e is the estimation cost function value and J_v is the validation cost function value.

N_{N1}	n_θ	J_e	J_v
2	14	$1.585e-02$	$1.668e-02$
3	20	$9.669e-03$	$1.266e-02$
4	26	$9.018e-03$	$1.221e-02$
5	32	$8.863e-03$	$1.300e-02$
6	38	$8.361e-03$	$2.921e-02$
7	44	$8.261e-03$	$1.261e-02$
8	50	$8.028e-03$	$1.636e-02$

performance on the validation data set. The best cost function value we found during this experiment was $8.7699e-03$.

Figure 5.8(a) shows the spectrum of the output error signal obtained from the identification experiment together with the spectrum of the validation error. The spectra of output error signals obtained from identification experiments clearly shows the effect of process disturbance which is not described by the model. The estimated state observer then gives a flat spectrum of the output error signal shown in the Figure 5.8(b).

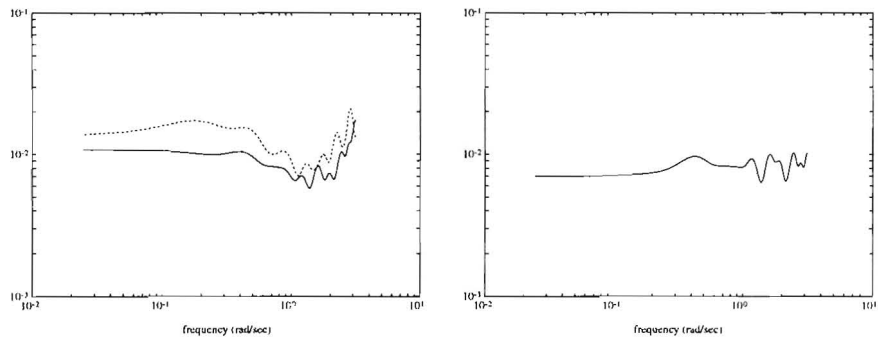
The nonlinear static state-feedback controller parametrized as

$$u(k) = \hat{k}[\hat{x}(k), r(k), \Theta_{\hat{k}}]$$

was optimized for the estimated simulation model in a noise free situation and then validated on the true description of the process given by (5.23)–(5.25) including the noise. The controller nonlinearity $\hat{k}$ was approximated by an MLP with one hidden layer containing 4 hidden nodes resulting in a design of 21 weights. As a reference signal we used a sequence of random steps uniformly distributed in the interval $(0.5, 1.5)$ of a total length 500 samples. The range of the reference signal was chosen with respect to the simulated range of the system output. The weights of the controller neural network were obtained from an optimization of the criterion (5.11) where the weighting factors Q and R were chosen as follows: $Q = 1$ and $R = 0.1$. The value of R was experimentally chosen to bound the amplitude of the control input by $-1 < u < 1$ as it defines the validity of the estimated model. The results of the controller design are plotted in Figure 5.9. The steady-state errors in the tracking of the state component $\hat{x}^1 = \hat{x}_2$ are due to the penalty we put on u to limit its range. The steady-state errors where removed by including an extra state into the controller neural network as follows

$$u(k) = \hat{k}_d[u(k-1), \hat{x}(k), r(k), \Theta_{\hat{k}_d}]$$

which should emulate an integral action in the closed loop. The weights of this dynamic state feedback were estimated by a minimization of the criterion (5.12).



(a) solid line - estimation error, dashed line - validation error

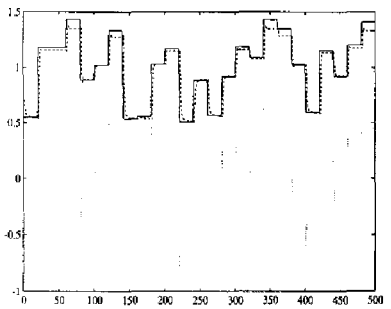
(b) state observer validation

Figure 5.8: Output error spectra

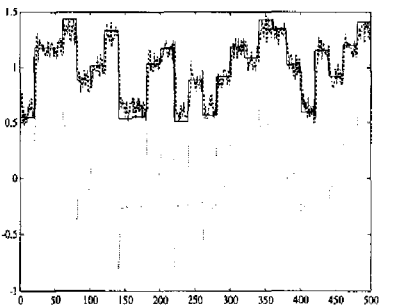
The plot of these results is shows in Figure 5.9(c). We have shown here only a noise free situation to see clearly that the tracking errors were removed. A validation of the dynamic controller with noise shows similar behaviour to results shown in Figure 5.9(b) only the steady-state errors are removed. Notice also that there are some irregularities in tracking of reference values close to the magnitude 0.5. This is due to the fact that these magnitudes are close to the nonlinear validity of the estimated model.

5.8 Summary

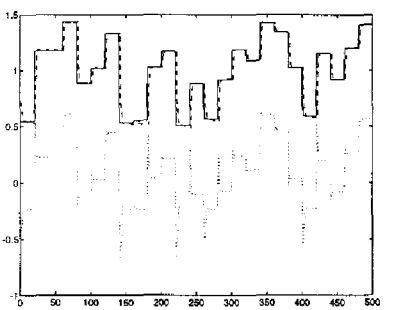
As a general nonlinear control problem is not analytically solvable we have considered in this chapter its numerical solution. The control function is, in general, parametrized as a static state feedback approximated by a multilayer feedforward neural network while the weights of the network are computed by minimization of an optimality criterion, defined in advance. We have shown by an example (multi-link inverted pendulum) that the choice of this criterion can influence the solvability of a considered control problem. The controller synthesis is based on separation of a state estimation problem and a state-feedback controller design problem. By doing so we expect an approximation of the optimal synthesis which we do not know. The numerical example, presented at the end of this chapter, shows feasibility of this approach.



(a) Static controller optimization



(b) Controller validation



(c) Dynamic controller estimation

Figure 5.9: State tracking controller results: solid line - state reference signal $r(k)$; dashed line - state $\hat{x}^1(k)$; dotted line - control signal $u(k)$

6 *Transition Control of a Polymerization Reactor*

In this chapter we will demonstrate on a large simulation example the applicability of the transition controller design as it was proposed in the above chapters. The controlled process will be a rigorous simulation model of the fluidized bed polymerization reactor shown already in Figure 1.1 on page 3. This system was chosen for our tests because a transition type of control is extensively used for this process and it also shows a complex nonlinear behaviour with process disturbances.

6.1 Process simulation model

To test the ideas presented in previous chapters we developed a rigorous mathematical model of the fluidized bed polymerization reactor [36] based on first principal analysis of material and energy balances in the reactor [10],[50]. The complete description of this model requires an introduction of more than hundred variables and is therefore outside the scope of this section. Therefore we describe only that part of the process which is relevant for presentation of our results.

A schematic diagram of the process is shown in Figure 6.1. The process consists of a reactor and a heat exchanger which are connected through a recycling loop. The reactor is fed at the bottom with a monomer (ethylene), co-monomer (propane, butane) and inertial gases (N_2,H_2). As the inertial gas is not consumed by the reaction it is added into the recycling loop once for all. In our rigorous model we did not manipulate the amount of H_2 fed into the reactor though it will be manipulated in the reality as it is used as a "chain stopper" in the polymerization reaction. The monomer and co-monomer polymerize in the reactor into a polymer which is then withdrawn from the reactor as a final product. The reactor is continuously supplied with a monomer and a co-monomer such that the total mass flow into the reactor C_{flow} and the ratio of the mass concentration of co-monomer to monomer $(C_x/C_2)_{set}$ in the mixed flow are being manipulated.

The lower part of the reactor is called a "fluidized bed" and consists of solid particles of polymer and catalyst called an "emulsion phase". The catalyst is fed into the reactor at a rate q_c. Through the emulsion phase bubbles of gas rise what

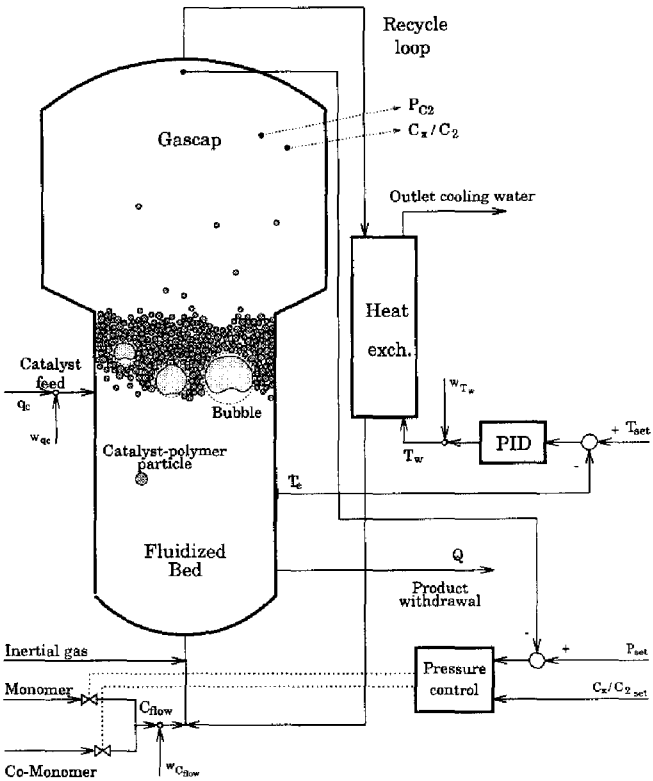


Figure 6.1: Fluidized bed polymerization rector diagram

is called as "bubble phase". It is important to operate the reactor at a gas velocity which is above the so called minimum fluidized velocity to maintain the fluidized bed of solid particles and bubbles of gas.

Above the bed is a free space called "gascap" which prevents the solid particles to get into the gas recycling loop. Recycled unreacted gases are cooled down in the heat exchanger and then added to the incoming fresh gas flow at the bottom of the reactor. The temperature of the cooling water temperature T_w in the heat exchanger is manipulated by a primary PID controller which stabilizes the process dynamics. The process instability is in fact caused by an exothermic type of the chemical reaction taking place in the reactor. The temperature of the emulsion phase T_e is controlled by this primary PID controller to a given set-point value T_{set}.

The final product (polyethylene) is withdrawn from the reactor at such a rate Q that the height of the bed remains constant.

The total length of the recycling loop is significant so that the unreacted gases are added to the input flow with a delay of about 1 minute. This delay together with the dynamics of the heat exchanger create slow dynamics of the process.

The gas in the gascap is analyzed in analyzers to measure the partial pressure of monomer P_{C2} and a ratio ζ of the partial pressure of co-monomer to monomer $\zeta = P_{C_x}/P_{C_2}$. As this measurement takes about five minutes this will be our bottleneck for the choice of the sampling time for the control system. Besides, the measured samples of P_{C2} and ζ will be also delayed five minutes. Also the production rate Q measurement requires some chemical analysis and these samples are also available at a rate of one sample per five minutes with a delay of five minutes.

The total pressure in the gascap P which relates to the product quality is basically kept at a certain constant value. The problem is that the five minutes sampling interval of P_{C_2} and ζ might be too slow for a compensation of fast perturbations of the pressure P. The total pressure P can be measured on-line at a much faster rate than one sample per 5 minutes. Therefore a primary pressure PID controller is designed here to suppress the fast pressure perturbations in the gascap. The structure of this controller is shown in Figure 6.2. The pressure set-point P_{set} is compared with the measured pressure P, the error signal is then filtered by the PID controller which manipulates the total mass flow into the reactor C_{flow}. The PID controller is then followed by a block of nonlinear functions which compute the actual monomer and co-monomer input flows by using their ratio set-point $(C_x/C_2)_{set}$.

Both the temperature PID primary controller and the pressure PID controller were tuned experimentally by a visual judgment of either water temperature disturbance or pressure disturbance step responses.

To demonstrate the process behaviour we show here a couple of step experiments obtained by simulation of considered types of process disturbances. Figures 6.3-6.5 show effects of perturbations of process variables: C_{flow} - the total process mass input flow, T_w - the cooling water temperature and X_{cat} - the mass fraction of the catalyst with respect to the solid particles. From a positive C_{flow} disturbance

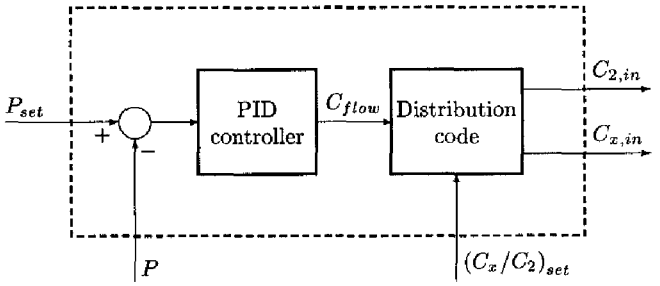


Figure 6.2: Pressure inner-loop controller

simulation (see Figure 6.3) can be seen that the production rate, the temperature and the monomer partial pressure initially raise due to increased reaction rates in the reactor. The ratio ζ initially drops due to the smaller reactivity of the co-monomer with respect to the monomer. We can also observe that except of the ratio ζ the settling time of all shown process variables is about 2 hours. The settling time of ζ is much longer and in fact converges slowly to a steady-state value due to the already mentioned slow reactivity of the co-monomer. The overshoot is caused in all responses by the delay of the recycling loop what is confirmed by simulation experiments shown in Figure 6.4. Due to the delay in the recycling loop, the water temperature PID controller responds to the increase of the bed temperature T_e with a delay and therefore there is a rather large initial increase of the bed temperature T_e. Due to the increased temperature in the reactor the production rate also increases, the pressure drops due to the faster consumption of monomer and co-monomer and the ratio increases, because the monomer is consumed faster than the co-monomer. The water temperature disturbance is fully compensated by the controller and the process stabilizes to its original steady-state values. The catalyst activity perturbation responses, shown in Figure 6.5, result in a similar behaviour. The only difference is that some of the process variables converge to different steady-state values. In this case an adjustment of the $(C_x/C_2)_{set}$ set-point is required.

We can observe that the settling time of the process is about 2 hours for first three outputs and about 8 hours for the last output. The slow dynamics of the process is caused mainly by the recycling loop. It causes that the concentrations of monomer and co-monomer in the reactor reach their steady-state values in the order of tenth of hours. We can also see the effect of primary controllers which stabilize the process dynamics. An C_{flow} perturbation is partly handled by the pressure controller which controls the total pressure in the reactor. However, the partial pressure P_{C_2} and the ratio ζ stabilizes around different equilibrium points. The water temperature disturbance T_w is completely compensated by the

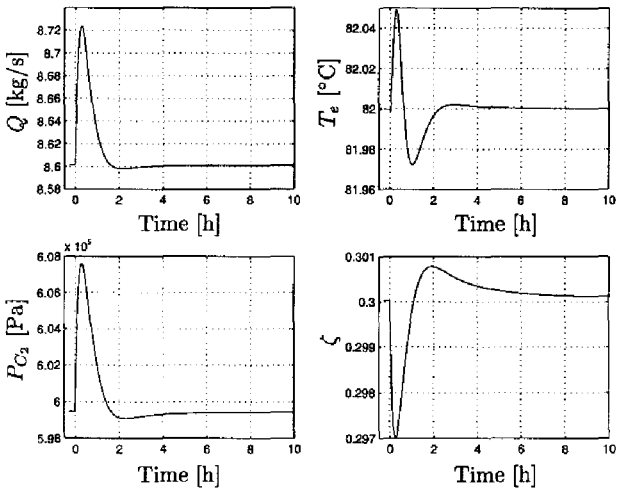


Figure 6.3: Step experiment of C_{flow} perturbation by a disturbance 0.02

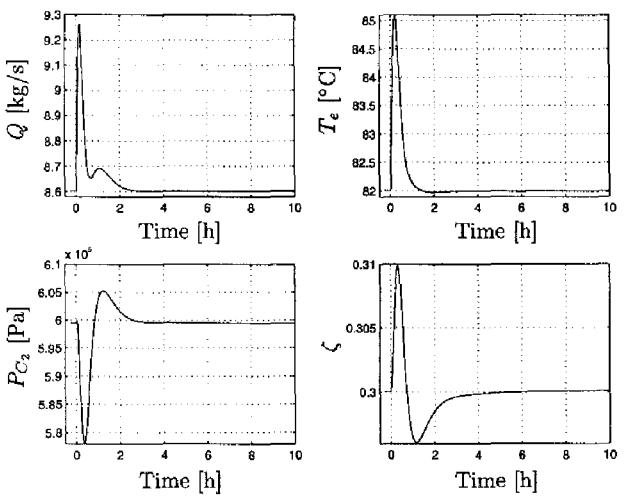


Figure 6.4: Step experiment of T_w perturbation by a disturbance 0.03

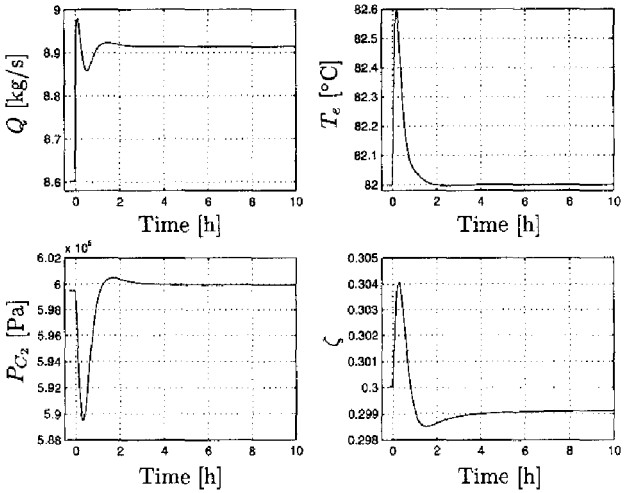


Figure 6.5: Step experiment of X_{cat} perturbation by a disturbance 1.5e05

corresponding PID controller. The catalyst activity disturbance seems to affect the process dynamics more seriously. To return the process conditions to its original operating conditions requires manipulation of all process inputs. From the Figure 6.5 we can realize that the production rate Q is much more sensitive to the C_{flow} disturbance than the bed temperature T_e and the partial pressure P_{C_2}. This is due to the effects of primary controllers.

6.2 The control problem

The grade of the produced polymer is characterized by properties like: melt index, density, molecular weight and distribution, chain structure and others. These characteristics are influenced by the selected operating conditions of the reactor. Let these conditions be characterized by values of the following four process variables:

1. Q - production rate,
2. T_e - temperature of the emulsion phase, also called bed temperature,
3. P_{C_2} - partial pressure of the monomer in the bubble phase,
4. ζ - ratio of co-monomer to monomer partial pressure in the bubble phase $P_{C_x} = \zeta P_{C_2}$.

All these variables are directly measured and define measured process outputs as follows

$$y = (Q,\, T_e,\, P_{C_2},\, \zeta)^T$$

A different combination of these values corresponds to a different type of product being produced. When switching the process production to a production of a different type of product we have to change these values to new ones. This "change-over" type of operation is supposed to be performed by manipulating the following input variables:

1. T_{set} - set-point for the bed temperatureT_e,
2. P_{set} - set-point for the total pressure P in the gascap,
3. $(C_x/C_2)_{set}$ - set-point for the ratio of co-monomer to monomer concentration in the reactor input flow,
4. q_c - catalyst input rate.

These variables define then the process control inputs

$$u = (T_{set},\, P_{set},\, (C_x/C_2)_{set},\, q_c)^T$$

The most important disturbances acting on the process include:

1. w_{T_w} - water temperature disturbance,
2. w_{qc} - catalyst flow disturbance reflecting impurities in the catalyst and irregularities in its activity,
3. $w_{C_{flow}}$ - disturbance acting on the total input flow of monomer and co-monomer into the reactor reflecting impurities and temperature changes of the input mass.

These variables define then the process disturbance

$$w = (w_{T_w},\, w_{qc},\, w_{C_{flow}})^T$$

The measurement noise v is assumed to be additive to outputs y and is represented by a four dimensional vector of measurement errors.

A transition from one operating point to another should be fast to minimize either the wasted production or production of wide specifications type.

The rate of change of all process inputs is assumed to be limited to a few percent of their nominal values, usually about 10%, for safety reasons. A faster change of process inputs results in physically non-feasible values of some variables of the used simulation model of the reactor, e.g. the required input flow can become negative or the required cooling water temperature in the heat exchanger can also become negative. Faster changes of the bed temperature T_e could in reality cause melting of solid particles an a collapse of the fluidized bed.

6.3 Identification

At the first stage of the controller design we estimate a simulation model of the process. Before being able to do this we have to prepare and perform measurement experiments on the process. The classical theory of system identification (see e.g. [33]) gives us properties of test signals for linear systems. In case of the linearity assumptions of the process dynamic and a parametrization of estimated model either a gaussian white noise signal or PRBS (pseudo-random binary sequence) are typical test signals. As the process dynamics are nonlinear it is not clear what an optimal test signal should be. Often we use a similar type of signals as in the case of linear system identification. Due to the process nonlinearity we often use uniformly distributed random signals to excite the process in very specific ranges. When testing a practical process we are very seldom, or better to say newer, allowed to excite the process by input signals of white noise type. Usually we are restricted to use very specific signals, responses to which are well predictable.

To create a data set we have used in our tests a low-pass filtered uniformly distributed white noise signal. However, in practice, the measured data will consist of a set of operating point changes performed on a real process. As these operations use a very specific control inputs, like step changes of set-points, the estimated model will be valid only in a very small region around the transition trajectories and will be of a very low pass character. Nevertheless, such a model could be very useful to improve the model accuracy around existing transition trajectories.

6.3.1 The data

To create a data set for identification purposes, the process inputs were excited using a discrete-time low-pass filtered uniformly distributed random signal followed by a zero-order hold with a sampling time $t_s = 300$ seconds. As a low-pass filter we used a fourth order Butterworth filter with a cutoff frequency $\omega_n = (1/16)\omega_s$, were ω_s was the sampling frequency in rad/s. The cut-off frequency of the filter was chosen such that the rate of change of test inputs was only a few percent of their nominal values. In this case not more than 10%.

The range of test signals was chosen as big as possible with respect to the physical ranges of process variables simulated using the available simulation model of the process. That means such ranges of process inputs that the process does not operate in physically impossible states. The physical ranges of simulated input/output process variables are given in Table 6.1. In case of the temperature T_{set} and the pressure set-point P_{set} these ranges were chosen as 10% of the nominal values of corresponding variables. The concentration ratio $(C_x/C_2)_{set}$ was changed for about 90% and catalyst input flow for about 20% of their nominal values, respectively. These ranges were chosen with some margin to be able to simulate also process disturbances without violating the already mentioned impossible physical process states. The disturbance variations were chosen 5% of the nominal values of signals to which they were applied. These ranges of simulated disturbances are given in Table 6.2.

Table 6.1: Process input and output description

input	nominal value	minimum value	maximum value
T_{set} [°C]	355.00	390.50	319.5
P_{set} [10^5Pa]	20.99	18.89	23.08
$(C_x/C_2)_{set}$	0.30	0.04	0.57
q_c [kg/h]	2.83	2.30	3.40

(a) Test input description

output	nominal value
Q [kg/s]	2.39
T_e [°C]	355.00
P_{C_2} [10^5Pa]	5.99
C_x/C_2	0.30

(b) Output nominal values

Table 6.2: Disturbance input description

disturbance	magnitude	disturbed variable	nominal value
w_{T_w} [°C]	10.40	T_w	207.32
$w_{C_{flow}}$ [kg/s]	0.12	C_{flow}	2.39
$w_{X_{cat}}$ [10^{-4}]	0.16	X_{cat}	3.28

We simulated the process model in SIMULINK for 601 process hours using the standard Adams-Gear numerical integration method with a variable step size. Collected process data were low-pass filtered by a continuous time fourth order Butterworth anti-aliasing filter with a cutoff frequency $\omega_n = (1/4)\omega_s/2$ in rad/s and then sampled with a sampling time $T_s = 300$ seconds. The spectra of the simulated process outputs are shown in the Figure 6.6 as solid lines. The process was simulated also without the disturbance using only the process inputs. By subtracting these simulated outputs from the simulated outputs with disturbances we obtained a spectra plots of the effect of disturbances at the output (see Figure 6.6, dashed lines). The dotted lines in Figure 6.6 show the spectrum of measurement noise added to the outputs.

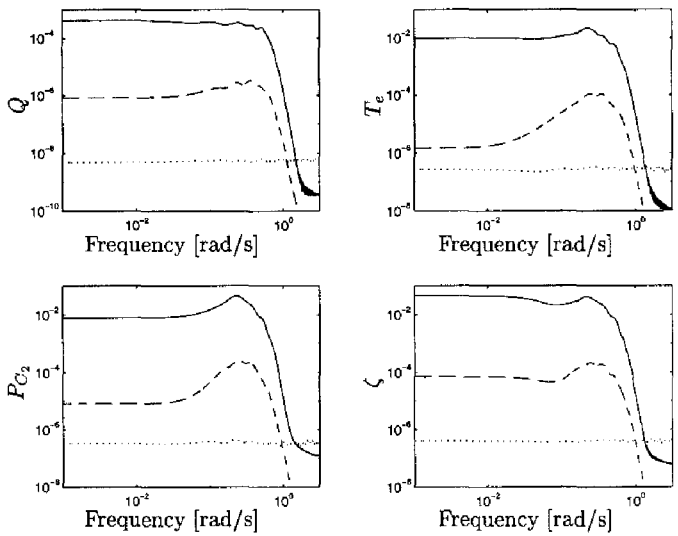


Figure 6.6: Spectra of simulated signals relative to the half of the sampling rate; solid line - simulated output spectrum; dashed line - spectrum of a difference between simulated output with disturbance and without disturbance and measurement noise; dotted line - spectrum of measurement noise.

6.3.2 *The grey-box model prametrization*

To parametrize the process model we will follow the methodology proposed in the Section 4.2.3. Let us assume that the partial pressures of the monomer P_{C_2} and the co-monomer P_{C_x} in the gascap relate to the concentration of the monomer C_{2g} and the co-monomer C_{xg}, respectively, according to the following correlations

$$\frac{R_g}{M_{C_2}} C_{2g} T_g = P_{C_2}$$
$$\frac{R_g}{M_{C_x}} C_{xg} T_g = P_{C_x}$$

where R_g [J mol^{-1}K^{-1}] is the known universal gas constant and M_{C_2}, M_{C_x} [kg mol^{-1}] are known molecular weights of monomer and co-monomer, respectively, which are known. The temperature of gases in the gascap T_g can be assumed to be equal to the temperature of the emulsion T_e, which is directly measured. This assumption is valid due to the high heat transfer coefficient between the solid particles and the bubbles of gas.

Let us define the physically known part of the state vector with respect to

(4.29) as follows

$$\hat{x}^1 = \begin{pmatrix} Q \\ T_e \\ \frac{R_g}{M_{C_2}} C_2 \\ \frac{R_g}{M_{C_x}} C_x \end{pmatrix} = \begin{pmatrix} \hat{x}_1^1 \\ \hat{x}_2^1 \\ \hat{x}_3^1 \\ \hat{x}_4^1 \end{pmatrix}$$

This means that $\hat{x}^1 \in I\!R^4$ will be represented by the first four components of the state of the model. The other state vector components will represent the hidden part of the state vector $\hat{x}^0$ which dimension will be subject of identification. The output map h in the model of this process is with respect to (4.30) on page 66 analytically defined as follows

$$h[\hat{x}(k), u(k)] = \begin{pmatrix} \hat{x}_1(k) \\ \hat{x}_2(k) \\ \hat{x}_2(k)\hat{x}_3(k) \\ \frac{\hat{x}_4(k)}{\hat{x}_3(k)} \end{pmatrix}$$

We also know that the catalyst, being fed into the reactor, becomes active in the reactor after about one hour. Assuming a sampling time of 5 minutes, this fact translates to a delay of 12 samples at the fourth control input. This a priori knowledge was brought into the estimated state map $\hat{f}$ as a known analytical part in a form of a tapped-delay line of 12 delay units, all with sampling period 5 minutes. In this way we have inserted into the model additional 12 known states, which were not estimated. In fact, the known part of the state map was realized by a time shift of the data sequence at the fourth control input, so $\bar{f}$ was skipped from the next discussion. The black-box part of the model $\hat{f}$ was then approximated by an MLP with one hidden layer with weights $\Theta_{\hat{f}}$. The dimension of $\hat{x}^0$ was considered either 1, 2 or 4. The input of the neural network $\hat{f}$ was then defined as

$$\chi \equiv (u(k)^T, \hat{x}(k)^T)^T = (u(k)^T, \hat{x}^1(k)^T, \hat{x}^0(k)^T)^T$$

and the output was defined as

$$\hat{\gamma} \equiv \hat{x}(k+1) = (\hat{x}^1(k+1)^T, \hat{x}^0(k+1)^T)^T$$

In $\hat{f}$ we have used linear output nodes corresponding to $\hat{x}^1$ states and sigmoidal output nodes corresponding to $\hat{x}^0$ states. The results of the optimization of the neural network's weights $\Theta_{\hat{f}}$ are discussed in the next section.

6.3.3 *Model parameter optimization*

The length of the data set, used for optimization of the neural network weights $\Theta_{\hat{f}}$, as well as for validation of the model, was $N = 7200$. The number of nodes

in the hidden layer of the neural network N_{N1} was varied from 8 to 14. We have performed a number of optimization experiments for a total state dimension $\hat{n} \in \{5, 6, 8\}$ and $\hat{x}(k) \in \mathbb{R}^{\hat{n}}$, that means defining one, two and four extra hidden states $\hat{x}^0$.

We observed that taking 5 states of the model led to a model of poor performance, while a model with 8 states, compared to a model with 6 states, gave us only a marginal improvement. Therefore we decided to continue further experiments with the state dimension $\hat{n} = 6$. This choice was also driven by the effort to limit the dimension of Θ_f with respect to the consumed CPU time by the numerical minimization routines.

We started the optimization of each neural network from the zero initial state condition and we performed $20n_{\theta_f}$ simulated annealing iterations followed by $20n_{\theta_f}$ quasi-Newton iterations to minimize the criterion (4.9). After that we replaced the zero initial condition by a mean value of simulated states, for each model complexity independently, and we performed additional $20n_{\theta_f}$ simulated annealing iterations followed by $20n_{\theta_f}$ quasi-Newton iterations. The results of this experiment are summarized in the Table 6.3. In the first column is shown the number of the hidden layer nodes and in the second column is shown the number of optimized weights. We can see, that already for the smallest model this number is quite significant. In the third and fourth columns are shown the final values of the cost function for both estimation and validation data. For each complexity of the neural network we restarted the optimization from five different random initial points. In the last two columns is shown the total computing time spent in the optimization routines, namely in the simulated annealing and in the quasi-Newton optimization. Comparing to the number of iterations, we have performed, to the number of weights these figures are quite significant. Note, that the simulated annealing was always terminated at the maximum number of cost function evaluations, while the quasi-Newton optimization was in some cases terminated due to impossibility of finding a better minimizing point and in the other cases by reaching the maximum number of iterations.

As a good choice for the model of identified process we chose a model with 12 hidden nodes resulting in an estimated cost function value $1.5498e-5$, which gives us an output approximation accuracy of 0.3%, computed by (4.10) on page 48. We estimated also a linear state-space model of 6th order in a form of (4.43) on page 74 and it gave us an approximation accuracy of about 16–17%. This result demonstrates that the process behaviour shows significant nonlinearities. In Figure 6.7 are shown spectra plots of all four output error signals obtained with the chosen model. It can be seen from these plots that there is quite a good consistency in estimated results except of the solution with the worst cost function value. The remaining dynamics in the output errors are most likely caused by the process disturbance.

Table 6.3: Identification results: N_{N1}-number of nodes, n_{θ_f}-number of weights, J_e-estimation cost function, J_v-validation cost function, SA - total time spent in the simulated annealing optimization, QN - total time spent in the quasi-Newton optimization (the large computation time figures in the first row of this table were caused by execution of a higher priority job).

N_{N1}	n_{θ_f}	J_e (10^{-5})	J_v (10^{-5})	SA [sec]	QN [sec]
8	142	3.2657	3.2495	33852	90233
		1.9874	1.9275	3780	28351
		2.1022	2.1005	3985	51122
		2.0257	1.9812	3052	26150
		1.9727	1.9171	4300	46301
9	159	2.6389	2.6236	3710	22838
		2.5106	2.2996	2173	27790
		2.0429	1.9782	1767	13934
		3.0581	2.9221	2052	28513
		2.2956	2.2063	1770	13551
10	176	1.6828	1.6320	2060	48338
		1.8158	1.8409	5156	35474
		2.2748	2.2614	2389	56271
		1.7987	1.8388	5156	35474
		1.9857	1.9321	2389	56271
11	193	1.6699	1.6579	2476	24537
		1.7841	1.7734	2378	22178
		1.6925	1.6889	2447	23203
		1.6157	1.5747	3061	23606
		1.6424	1.6538	2537	21270
12	210	2.4552	2.4751	7591	43386
		1.5953	1.5792	4547	53573
		1.6806	1.6022	4704	30174
		1.5721	1.5529	4872	35692
		1.5498	1.5740	4794	30104
13	227	1.5747	1.6028	5370	36628
		1.6092	1.6034	5408	36124
		1.5731	1.5392	5417	40071
		1.6409	1.6230	5636	39506
		1.5531	1.5703	5384	36450
14	244	1.7161	1.7011	6206	48889
		1.5644	1.5849	5946	93622
		1.5209	1.5587	7758	94500
		1.4925	1.5504	5962	53676
		1.5727	1.5528	6091	164523

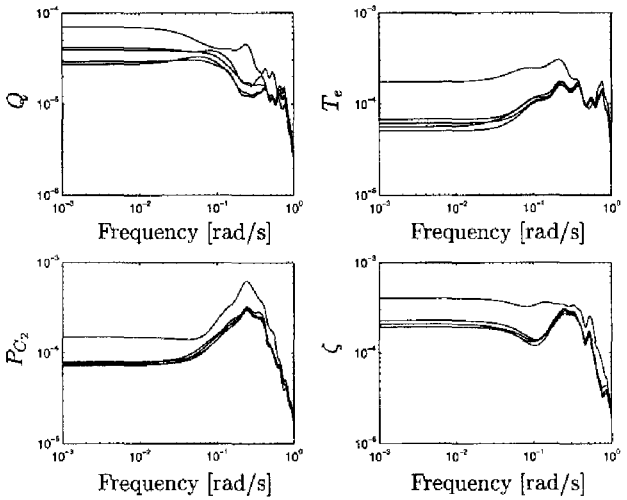


Figure 6.7: Estimated model output error spectra

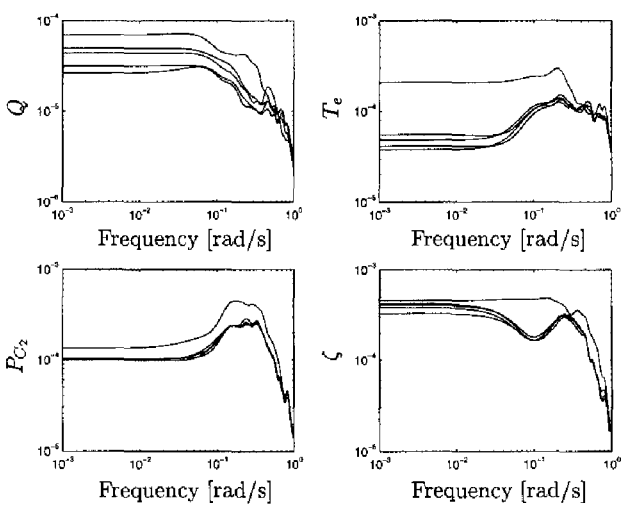


Figure 6.8: Estimated model output error spectra for validation data

6.4 State estimation

The second step in our controller synthesis algorithm is the state observer design. That is done via an estimation of a nonlinear filter gain $\hat{g}$ which is added to the previously estimated simulation model of the process. We parametrized the state observer as a single-stage ahead state predictor (4.37) with a gain $\hat{g}$ parametrized by (4.42e) on page 73. We assumed, that the process nonlinearity is not that big that we should take a more complex parametrization of the filter gain $\hat{g}$ to obtain a sufficient accuracy of the state estimates. The nonlinearity of $\hat{g}$ was approximated by a neural network with one hidden layer with variable number of nodes. The number of neural network inputs was 4 and the number of neural network outputs was 6. By varying the number of hidden nodes from 4 to 9 we optimized an additional from 50 to 105 weights $\Theta_{\hat{g}}$, to the 210 weights $\Theta_{\hat{f}}$ of the simulation model which were fixed during this optimization.

These optimization results are summarized in Table 6.4. Initial values of parameters $\Theta_{\hat{g}}$ were generated at random. The dash "-" symbol in this table means, that the optimization got stuck in a local minimum with a very high value. We can see, that variations of the performance of the observer given by different neural network complexities are not that big and we also can see that the estimated and validated cost values are consistent with each other. The smaller values of the

Table 6.4: Observer design results: N_{N1}-number of nodes, $n_{\theta_{\hat{g}}}$-number of weights, J_e-estimation cost function, J_v-validation cost function

N_{N1}	$n_{\theta_{\hat{g}}}$	J_e (10^{-6})	J_v (10^{-6})
4	50	2.3444	2.1063
		2.3610	2.1234
		2.3299	2.1028
5	61	2.3506	2.2149
		2.2676	2.0220
		2.3245	2.1349
6	72	-	-
		2.2979	2.0411
		2.2743	2.0334
7	83	-	-
		2.2449	2.0011
		2.2413	2.0100
8	94	2.2199	1.9914
		2.2410	2.0151
		2.2187	1.9901
9	105	2.2212	1.9793
		2.2295	1.9992
		2.2247	1.9835

validation cost function compared to the estimation cost function were seen also in Table 6.3 and are probably caused by a lower level of the noise in the validation data set. As a good approximation of the observer gain $\hat{g}$ we have chosen a neural network with 6 nodes with a final value of the cost function $2.2743e-6$. The spectrum of predicted output errors $y(k)-\hat{y}(k)$ produced by this state observer are shown in Figure 6.9. In the case of the first output Q, there are still some low pass effects present in the prediction errors. These may be caused by the model errors. A better solution would require a re-estimation of the process simulation model with a higher complexity of the neural network. We could also try to further optimize the filter gain $\hat{g}$ parameters or to increase the complexity of the neural network.

Finally, we have checked for distributions of estimated observer prediction errors $y(k)-\hat{y}(k)$, because they are going to be used later on in the controller design stage. In Figure 6.10 are shown estimates of probability density functions of innovation sequences by means of histograms (solid lines). In the same plots are shown also Gaussian probability density functions (dashed lines). The mean value is zero in all cases and the standard deviations are $\sigma = (1.6, 2.1, 2.2, 2.3)10^{-3}$, estimated by

$$\sigma_i = \sqrt{\frac{\sum_{k=1}^{N} e_i(k)^2}{N-1}}$$

for $i = 1, 2, 3, 4$. We can see that the estimated p.d.f. of prediction error sequences are close to those of the Normal distribution.

6.5 Controller design and validation

The last step of the transition controller design algorithm is the design of a state-tracking feedback controller. The design proposed in the Chapter 5 will be followed and demonstrated.

First of all, we define a set of equilibrium points $\mu = (\hat{x}_e, u_e)$ of the estimated model by computing a couple of solutions of the equation

$$\hat{f}[hx_e, u_e] - \hat{x}_e = 0$$

Fror this porpose we have used the `MATLAB` function `fsolve.m` and we have found 20 different equilibrium points, defining the set $\mathbb{E}$. Next, we have chosen 3 operating points out of this set, namely (OP_1, OP_2, OP_3), which define a set of six possible state transitions among them. The state reference signal r was chosen as a sequence of low pass filtered unit steps scaled to the proper initial and final equilibrium values to specify a sequence of transitions

$$OP_1 \rightarrow OP_2 \rightarrow OP_3 \rightarrow OP_1 \rightarrow OP_3 \rightarrow OP_2 \rightarrow OP_1$$

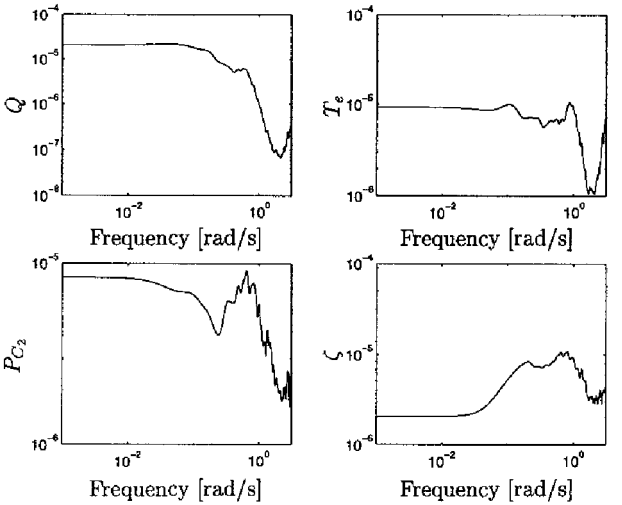


Figure 6.9: Observer validation results: Spectra plots of prediction errors for all four outputs.

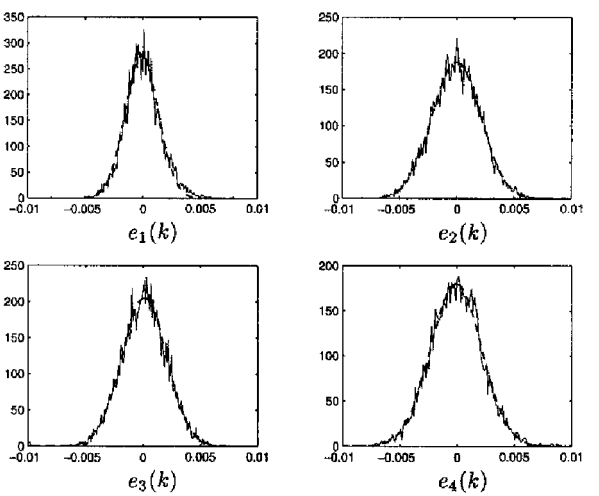


Figure 6.10: Probability density function estimation of prediction errors for all four outputs

Each reference input signal was filtered by a first-order filter given by

$$P_i(z) = \frac{1 - \exp(-1/\tau_i)}{z - \exp(-1/\tau_i)} \quad \text{for } i \in \{1, 2, 3, 4, 5, 6\}$$

where the index i stands for the reference input. The time constants τ_i were a priori chosen with respect to the process dynamic as follows

$$\tau_1 = 8,\ \tau_2 = 8,\ \tau_2 = 12,\ \tau_2 = 30,\ \tau_2 = 10,\ \tau_2 = 20$$

The first two reference signals stand for the state components representing the production rate Q and the temperature T_e. These outputs show faster dynamics then other state components, mainly due to effects of primary controllers. The third and fourth state component, describing the concentrations of the monomer C_2 and the co-monomer C_x in the gascap show slower dynamics. Moreover, the dynamic of co-monomer is much slower. The last two states are hidden states without a physical interpretation and therefore are their responses judged using the estimated model. Notice also, that the process was tested in a low pass band and therefore the choice of reference signal must be done also in this respect.

The eigen frequency of filters used to pre-filter the test input signals was 0.08 rad/min which corresponds to a time constant about 2.5 samples. The lowest time constant, used for pre-filtering of the reference signals, is chosen with this respect and a safety margin is taken as worse model performance is to be expected at higher frequencies.

The controller was optimized using the set-up shown in the Figure 5.7. We minimized the criterion (5.21) combined with (5.12). We have chosen $t_0 = 0$ and $t_f = 75$ hours resulting in 900 samples in a single transition. The controller $\hat{k}$ was parametrized by an MLP with one hidden layer.

At the first instance we performed a few optimization experiments using different complexities of the controller neural network and different penalty matrices Q and R in (5.12). We also varied the integration time constant a in (5.17) within an interval $\langle 0.99, 1\rangle$, starting wit a value 0.99. All these experiments where done with a choice of $e_{ref} = 0$. At the first instance we had observed, that for smaller values of t_f and a choice of $a = 1$ the closed loop response tend to show an unstable pole in the local linearization of the closed loop around the final equilibrium point. A typical result showing this kind of solutions is shown in Figure 6.11. It becomes clear, at this point, why we have introduced an integration time constant $a < 1$ in (5.17) on page 102. By performing a couple of minimization experiments we have experienced that a random initial guess for the controller neural network parameters with $a < 1$ led more likely to a stable behaviour of the closed loop. By a gradual increase of a to a final value $a = 1$ we managed to obtain a stable closed loop also for the choice of $a = 1$.

The final results of the controller design are shown in Figure 6.12 and Figure 6.13. These results were obtained using weighting matrices

$$Q = \text{diag}(0.5, 0.1, 0.1, 0.02, 0.01, 0.01) \tag{6.1}$$

$$R = \text{diag}(0.1, 10, 2, 0.01) \tag{6.2}$$

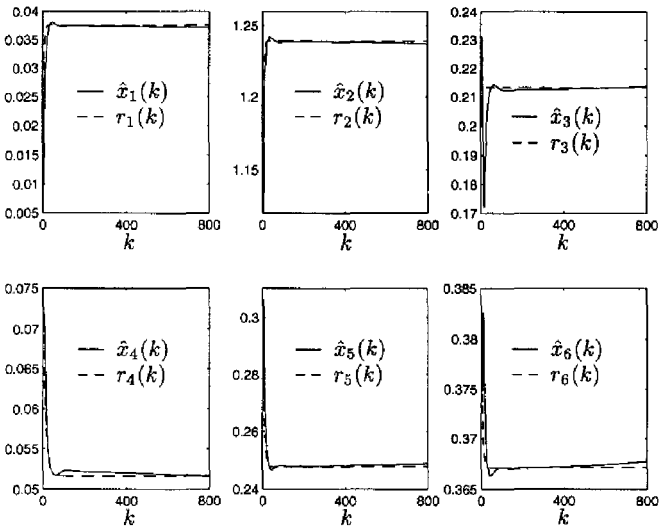


Figure 6.11: Unstable state transitions due to a "bad" choice of $t_f = 800$ and $a = 1$

It was also found out to be advantageous to add an extra penalty term to the cost function (5.12) in the earlier stages of the optimization based on the computed control equilibrium values. This term had the following form

$$\|u(k) - u_r(k)\|^2_{R_r}$$

where $u_r(k)$ is a control reference signal chosen as a sequence of unit steps similarly to the choice of the state reference, but this time without pre-filtering. The weighting matrix R_r was chosen as follows

$$R_r = \text{diag}(0.001, 0.001, 0.001, 0.0001)$$

This extra term helps to keep the simulation of the closed loop in estimated validity bounds of the neural network of the model. Note, that the extrapolation of neural network models is in general poor and a random initial guess for the controller neural network weights will, in general, not guarantee that all signals in the closed loop stay in their proper regions.

In Figures 6.14 and 6.15 we have shown zoomed estimated trajectories of all six transitions for both the control input and states, respectively. Each column in these figures corresponds to a particular control input or to a model state and each row corresponds to one of six estimated transitions.

Recall, that the previous optimization of the controller was done with $e_{ref} = 0$ (see Figure 5.7). The last step of the controller estimation was a further optimization of the controller using a nonzero $e_{ref}(k)$ signal. Now, we defined a control criterion with three examples of each of the six considered transitions for a different realization of $e_{ref}(k)$. The total length of computed closed-loop signal sequences was in this case 17100 data points. Due to high computation costs we performed only a modest number of random search iterations.

The final controller was validated with the original process by a continuous time simulation of the closed loop in SIMULINK. The simulation scheme used for this experiment is briefly discussed in Appendix B. The process was simulated with process disturbances and measurement noise. The final results are shown in Figures 6.16, 6.17 and 6.18. Basically we can observe from plots in Figures 6.12 and 6.16 that the time response of the controller was increased and that is in a sort of de-tuning of the controller when optimizing it with noise. This suggests that there will always be a trade-off between the disturbance reduction and the tracking accuracy. We can also observe an offset in steady-state values of the first output, which was the production rate Q. The cause of this may be in a poor performance of the filter gain $\hat{g}$ with respect to this output which was showing a rather poor performance at this output already before (see Figure 6.9).

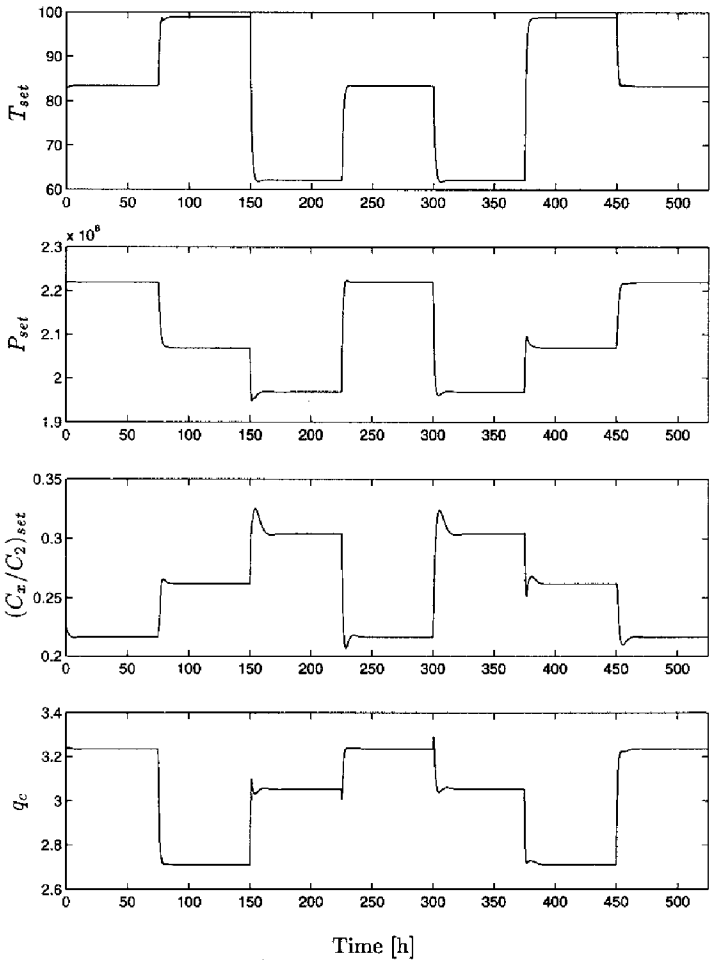


Figure 6.12: Estimated control inputs $u(k)$ for the neural model without the disturbance ($e_{ref} = 0$)

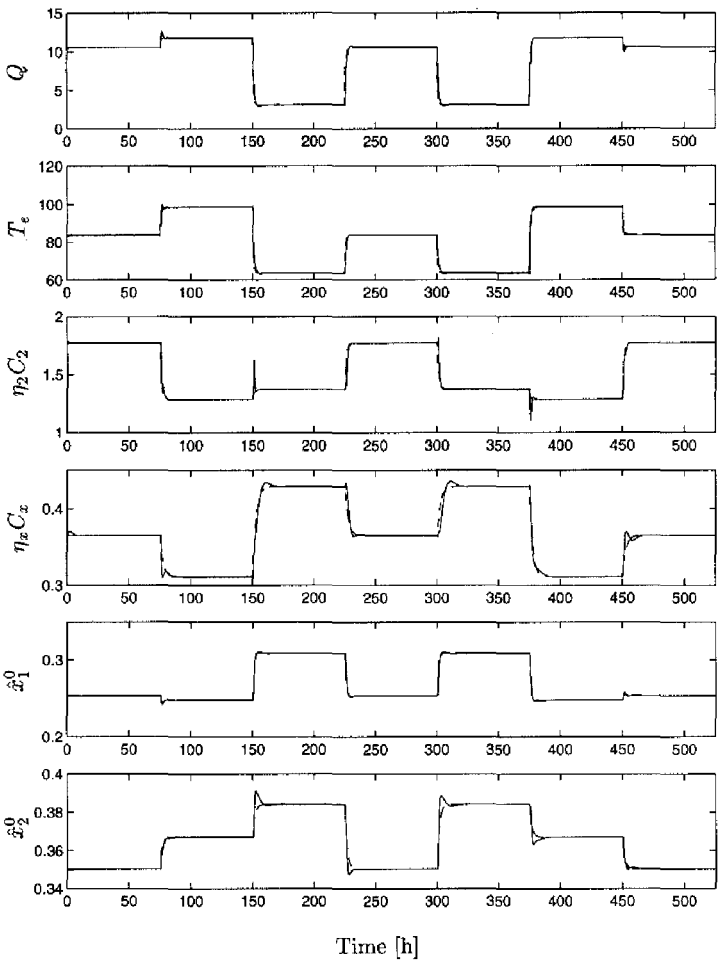


Figure 6.13: Estimated state transitions without the disturbance ($e_{ref} = 0$): dashed line - reference signal $r(k)$, solid line - estimated model state $\hat{x}(k)$

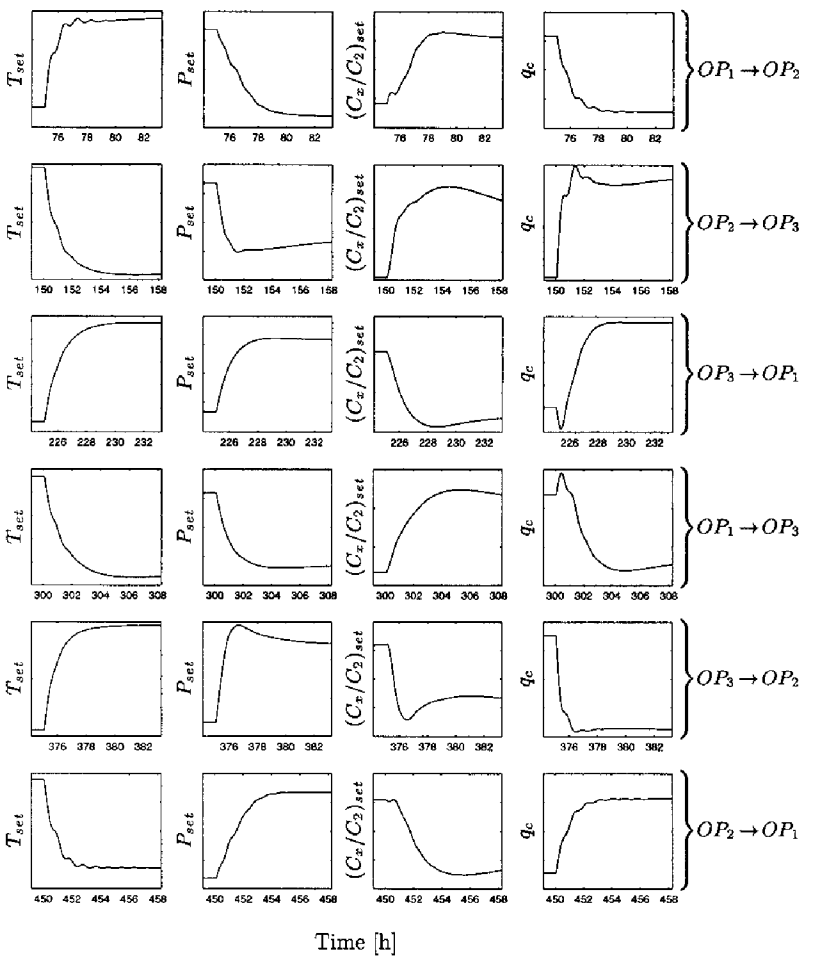


Figure 6.14: Zoomed control transitions at first 100 samples. Columns stand for 4 control inputs and rows for 6 different transitions

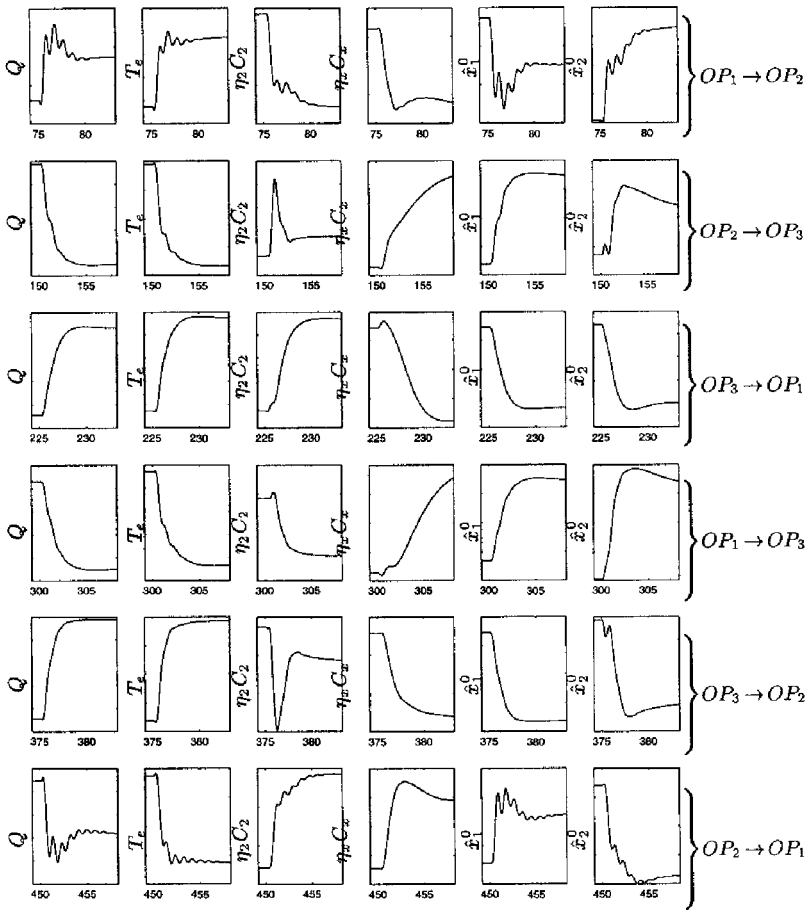


Figure 6.15: Zoomed model state transitions at first 100 samples. Columns stand for 6 states and rows for 6 different transitions

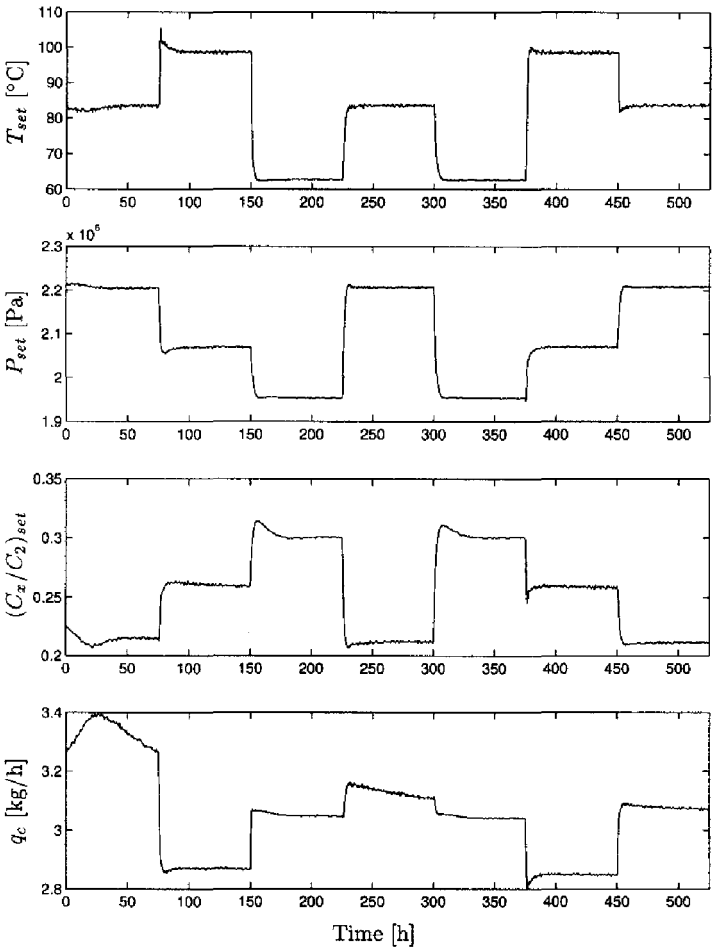


Figure 6.16: Validated control input on a "real" process with disturbances and a measurement noise

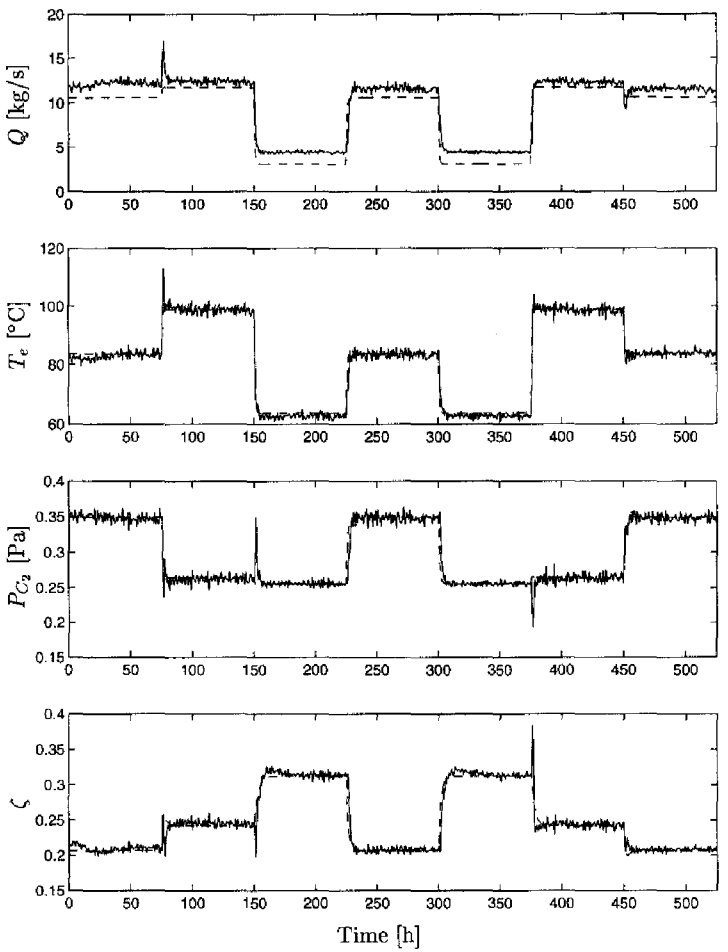


Figure 6.17: Validated outputs of the real process: dashed line - assumed output reference, solid line - true process output

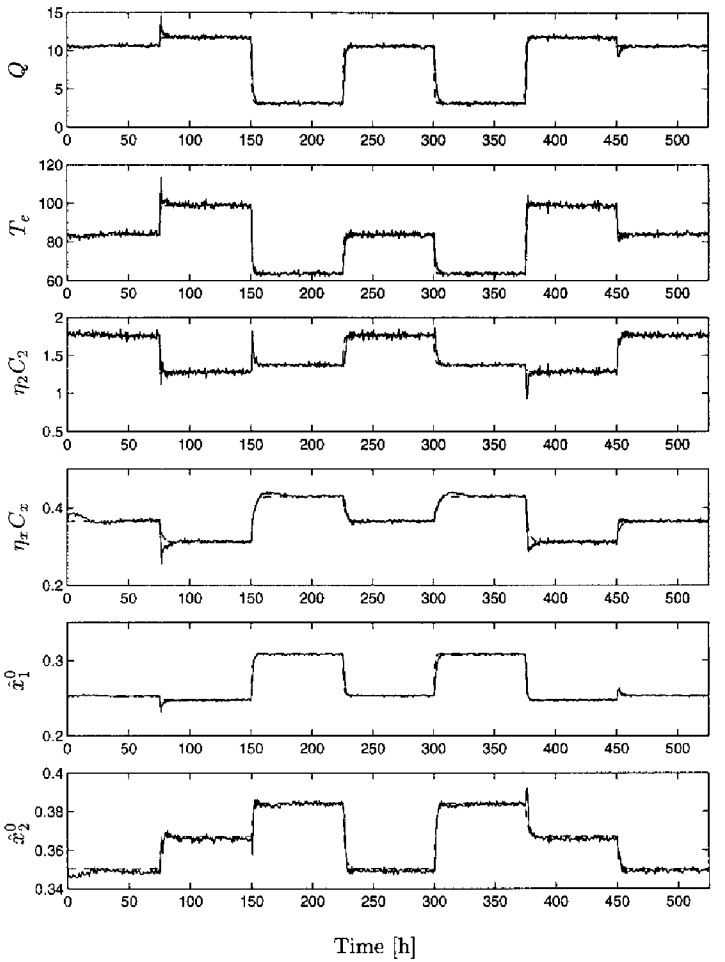


Figure 6.18: States of the process model in the controller during its validation

6.6 Summary

The simulation example presented in this chapter clearly showed strong and weak points of the the proposed controller synthesis. The strong points are:

1. A carefully estimated and validated simulation model of the process guarantees a good level of robustness of the controller with respect to modelling errors and process disturbances.

2. The process disturbances are handled very well by the designed neural state observer.

3. The controller is a nonlinear static state feedback where the process estimated state is extended with integrator states.

4. The process constraints are handled by the choice of the reference signals and by the choice of weighting matrices in the control criterion.

The weak point of this controller design is that the all involved design steps are translated to a minimization problem of a non-convex function. This results in a large computational time.

7 Conclusions and Recommendations

7.1 Conclusions

Recently, a large variety of control techniques concerning industrial process control based on neural networks was proposed. Often authors claim that their method is suitable for control of an unknown nonlinear system. They still implicitly make strong assumptions on the available a priori information. Frequently the assumption on the availability of a full process state for the feedback is made, see e.g. [68]. More realistic are approaches based on predictive control techniques, but authors often present simulations of these techniques without considering process disturbances [62]. A state-space approach to modelling and control using neural networks was recently elaborated on [65, 64].

Our approach to controller design differs from these techniques mainly due to following: We estimate a *grey-box state-space simulation* model of the process with a physical meaning of a part of state components; a neural state observer is parametrized and *estimated independently* of the process model so that the controller can cope with *process disturbances*. The reference signal specification can be based either on available a priori knowledge or can be optimized for the estimated simulation model. The final controller is based on dynamic output feedback with a structure similar to a linear LQG design providing us with better insight into the controller functionality than a complete black-box design.

In the following we summarize the conclusions of this thesis:

- According to the simulations we have done, we can conclude, that the control strategy proposed in this thesis is a powerful tool for controlling complex nonlinear systems while achieving a high level of performance. It shows robustness with respect to process disturbances and process uncertainties, mainly due to the fact that the controller is based on a grey-box simulation model of the process supplemented with a filter gain to predict the next process state. This is also supported by the state-space approach we adopted as it allows us to include a priori process knowledge into the model parametrization. Furthermore it allows us to do a more proper estimation and

validation of the model as part of the state vector has a physical meaning. The controller is completed with a neural nonlinear static state feedback, if necessary supplemented by a dynamic component in the form of integrators.

- Another important aspect of the proposed controller design is its structure and the resulting systematic design. The structure of the controller is dictated by:

  1. Grey-box state-space model estimation based on measured process data and parametrized by a combination of a *known analytical part* and a *black-box neural network*. The estimated model is a nonlinear simulation model of the process dynamics optimized to predict the process outputs over a *long horizon* terms.
  2. Neural state observer design to estimate a process state from disturbed process data. The observer takes the information from the estimated process model and adds a correction to model state predictions processed by means of a nonlinear filter gain from an error between the true and predicted process outputs. The filter gain is parametrized by a static neural network. Two types of a state observer are considered in this thesis: a single stage ahead predictor and a current state filter. An important aspect here is that the filter gain is parametrized and estimated *independently* of the simulation model and therefore it preserves the simulation capabilities of the state observer.
  3. The dynamic state-feedback design is being optimized such that the model states follow a priori defined trajectories to steer the system from one operating point to another. Provided that the model is accurate enough, the process outputs will converge also to correct values. A presence of physically defined states in the model, provides us with an easier way of reasonning when modifying the controller parametrization and/or the feedback structure, e.g. in order to obtain better convergence of the optimization algorithm. At the same time we can avoid a plenty of trials and errors in the design.

- In general, a neural network training task leads to a non-convex optimization problem. Structuring of the controller design allows us better access to this problem as we do it in three, "relatively simple" steps: model estimation, filter gain estimation and state feedback estimation. In each step, a static feedforward neural network is trained where all the dynamics are put outside of the neural network.

- The design is carried out in *state-space* domain. This approach was found to be *conceptually and algorithmically* more attractive than adopting input/output domain. By conceptual advantages of the state-space approach we mean the opportunity of incorporation of a priori process knowledge into the model parametrization, better access to control problem definition and also dealing with the process constraints. By algorithmical advantages we

mean an easy and effective way of translation of the design algorithms into digital computer programs. The state-space approach provides us also with more freedom as we usually control more process states (estimated) than process outputs.This provides more freedom in specifying reference signals. The controller is a static neural network and consequently, it is *easier to estimate* such a controller by numerical optimization techniques.

- When dealing with a control problem related to a practical process we have to face a *set of constraints* coming from physical process limitations and from safety considerations. These constraints concern both the modelling issues and controller design issues. Handling constraints is always a difficult task, especially if the problem is of a non-convex type. Moreover, as we parametrize all unknown nonlinearities by neural networks, we are faced with another set of constraints that define validity of these nonlinear approximations. We include these constraints into the design via an optimization of reference signals in time-domain by means of solving a suitable nonlinear optimal control problem off-line. The constraints can also be handled by using a priori knowledge in specification of proper reference signals which satisfy the process constraints with some safety margins. The controller then has to guarantee that the process remains close to these trajectories.

- In practice, the proposed controller synthesis will differ from the one demonstrated in Chapter 6, as we will not have such a freedom in the choice of testing signals to generate the data for the model estimation. The type of data, which are available in practice, consist of a large number of transitions performed on the process. A model, estimated using these data, will be tuned to very specific nonlinear regions of the process dynamics where the later transitions will take place. We may expect that the controller designed for the specific nonlinear reagions of the process dynamics will perform better than one being designed for a broad range of the process nonlinearities.

- We did some preliminary experiments with nonlinear multi-rate sampled neural state-space models discussed in Section 4.5.3 on page 81. These are novel neural state-space models based on different time intervals for updates of states of the model. These models could be applicable for an easier identification of stiff dynamics.

- Neural network training remains is the most time consuming part of the proposed controller design algorithm. This is caused by the non-convexity of this problem and by a high dimensionality of the neural network's weight space. For example, the controller designed in Chapter 6 for the FBPR, contains in total 366 weights, which had to be tuned.

7.2 Recommendations

- What concerns the controller design, presented in Chapter 6, we would rec-

ommend to perform a couple of transitions with the designed controller, record them and repeat the design using these data to see if there is any improvement in the performance.

- As the proposed controller design leads to non-convex minimization problems, one should take some precautions to make these minimizations more accessible:

  1. Include a priori knowledge into the model parametrization.
  2. Estimate all initial conditions as well, thich means both for the estimated state-space model and integral state of the controller, provided that it is included into the feedback.
  3. To save some computing time while to have good level of confidence in the chosen neural network complexity, one should always perform a few training iterations on a wider range of neural network complexities Then he can decide about the most promissing neural network complexity which parameters can be further optimized.
  4. Periodic re-scaling of neural network weights to improve numerical conditioning of the minimization (see Section 4.5.3, page 84).

- We would advise building up an expert system for neural network optimization. Our experience is, that at different stages of neural network training, different minimization methods are required for speeding up convergence. Moreover, every method requires a set up of optional parameters which determine performance, e.g. line search accuracy for the quasi-Newton, a speed of temperature reduction in simulated annealing and many more. However, we do not have a conclusive recommendation in this respect. We found out, that trying high accuracy for line search in quasi-Newton leads to extensive computing time whereas it could be more effective using a few extra iterations. To optimize all these parameters, e.g. by simulated annealing, we soon found out that the available computing power was too small.

- Some attention should be given also to the further improvement of the software routines we developed for simulation and estimation purposes of this work. Though some of these routines were carefully optimized, like the forward neural network evaluation path and the backpropagation path, there are still places which were not programmed efficiently. These concern mainly data storage and update during the neural network training. Notice, that when dealing with multivariable systems and large data sets, this issue should not be underestimated. At present, the available software is a bundle of undocumented `C` functions and `MATLAB` macros to carry on the design at different steps. The data transfer from one program to the next one is done manually. Also the analysis of all intermediate results is done manually. Automation of these steps would speed-up the design and avoids mistakes in data transfer.

A System Transformations

In this appendix we describe methods we frequently use to approximation a continuous time system by a discrete-time systems for computer simulation purposes.

The transformation of a continuous time system to a discrete-time system can be determined by the approximation of the first derivative involved in (2.1a). If we approximate the first derivative by a forward difference

$$\dot{x}(t) \approx \frac{x(t+\tau) - x(t)}{\tau}$$

where $\tau > 0$ is small with respect to the time variation of $x(t)$, and we substitute this approximation into (2.1a) we obtain an approximation

$$x(t+\tau) = x(t) + \tau f[x(t), u(t), w(t)] \tag{A.1}$$

When τ is considered as being the sampling time T_s and we index $x(t)$ as $x(k)$ and $x(t+\tau)$ as $x(k+1)$ we immediately obtain a description of type (2.3).

A more accurate derivative approximation is obtained when using its central difference approximation

$$\dot{x}(t) \approx \frac{x(t+\tau) - x(t-\tau)}{2\tau}$$

A discretized version of (2.1a) then gets the following form

$$x(k+1) = x(k-1) + 2\tau f[x(k), u(k), w(k)] \tag{A.2}$$

while assuming an indexing in a similar way to the previous case. To rewrite the last formula into the form of (2.3a) we have to define a new state vector consisting of $x(k)$ and $x(k+1)$. This observation brings us to the following remark.

Remark. The state dimension of a discretized system does not necessarily have to coincide with the state dimension of its continuous time system.

In general we can expand the solution of (2.1a) at a certain time moment into a Taylor series as follows

$$x(t+\tau) = x(t) + \dot{x}(t)\tau + \ddot{x}(t)\tau^2/2 + \ldots + x^{(j)}(t)\tau^j/j! \tag{A.3}$$

where j is the order of the series. By consecutive differentiation of the right hand site of (2.1a) with respect to time t we can obtain all the necessary time derivatives of the state vector for the expression (A.3). The speed of convergence of the Taylor series (A.3) is given by the choice of the step τ and the accuracy of the approximation is controlled then by τ and j.

In all of the introduced system disretizations the accuracy of the discrete-time model is given by the choice of τ. To ensure certain accuracy we relate τ to the system sampling time T_s by $T_s = \tau l$, where $l \in I\!N, l > 1$.

For the case of simplicity and the ease of programming is for computer simulations often chosen in (A.3) $j = 1$ which results in the well known Euler method of integration of ordinary differential equation or $j = 2$ which seems to be sufficient for most simulations provided that the step size τ is properly chosen. If the system nonlinearity is complex it might be to cumbersome to evaluate by hand analytically higher order derivatives of $x(t)$. Using some of the symbolic mathematical calculation software package (e.g. `MAPLE`) can greatly help here. We have observed some advantages of this approach which are listed bellow

1. Easiness of the siscretization accuracy control, namely by τ and j. Notice, that the analytical formulas for time derivatives of $x(t)$ are independent of τ.

2. A symbolic mathematical calculation language, like `MAPLE`, was shown to be a very powerful tool to compute analytically Taylor series expansions of nonlinear functions and export them as a high level computer language (e.g. `C`, `C++`) subroutines and then link them with simulation or optimization programs.

Another approach to the problem of siscretization of the system (2.1) is to approximate the integral in the following expression

$$x(t+\tau) = x(t) + \int_{t}^{t+\tau} f_c[x(t'), u(t'), w(t')]dt' \tag{A.4}$$

The most straightforward way to approximate this integral is to compute

$$\tau f_c[x(t), u(t), w(t)]$$

while obtaining back the result (A.1). This simple integration method is often modified as follows

$$x(t+\tau) = x(t) + \frac{\tau}{2}\Big(f_c[x(t), u(t), w(t)] + f_c[x(t+\tau), u(t), w(t)]\Big) \tag{A.5}$$

which is an implicit nonlinear equation for $x(t+\tau)$. To solve this equation we can start from an Euler estimate $a = x(t) + \tau f_c[x(t), u(t), w(t)]$ and iterate (A.5) as follows

$$\begin{aligned} a(l+1) &= x_l(t) + \frac{\tau}{2}\Big(f_c[x(t), u(t), w(t)] + f_c[a(l), u(t), w(t)]\Big) \\ &\quad \text{for } l = 0, 1, \ldots \end{aligned}$$

until a convergence is obtained and then put $x(t+\tau) = a$. If we would perform only one iteration of (A.5) we obtain an explicit siscretization rule as follows

$$\begin{aligned} \bar{x}(t) &= x(t) + \tau f_c[x(t), u(t), w(t)] \\ x(t+\tau) &= x(t) + \frac{\tau}{2}\Big(f_c[x(t), u(t), w(t)] + f_c[\bar{x}(t), u(t), w(t)]\Big) \end{aligned} \tag{A.6}$$

known as Heun method.

To obtain a more accurate discretizations we can consider for instance the Runge-Kutta methods. The simpler methods are applicable for computer simulations when the computing time has to be short, e.g. in minimization algorithms. More complex methods are applicable mainly for off-line simulations, e.g. in SIMULINK.

B Simulation Model of the FBPR

In this Appendix we show the SIMULINK scheme of the simulated process and the controller, to give the reader an impression about the complexity of the problem we were treated in Chapter 6.

The main simulation scheme is shown in Figure B.1. One can easily recognize there the reactor block, the heat exchanger, the inner-loop pressure controller and the outer-loop transition controller. The inputs of this scheme include: the reference input, the process disturbance input and the measurement noise. The water temperature primary PID controller is hidden inside of the heat exchanger block.

The reactor model is shown in Figure B.2. It contains three integrators `x1`, `x6`, `x3` in the main diagram, then three integrators in a linear state-space model of the gascap and two integrators in the block of the catalyst activity model. There is also a delay of one hour at the sixth input of this diagram. The `f(u)` blocks contain different nonlinear terms.

In the Figure B.3 is shown the transition controller implementation which resembles the proposed controller structure given in Figure 5.7. In this diagram `f_hat`, `g_hat` and `k_hat` stand for the process state-space model, for the filter gain and for the state feedback, respectively. The output map `h[x_hat(k)]` is in the diagram fixed to a known analytical relation.

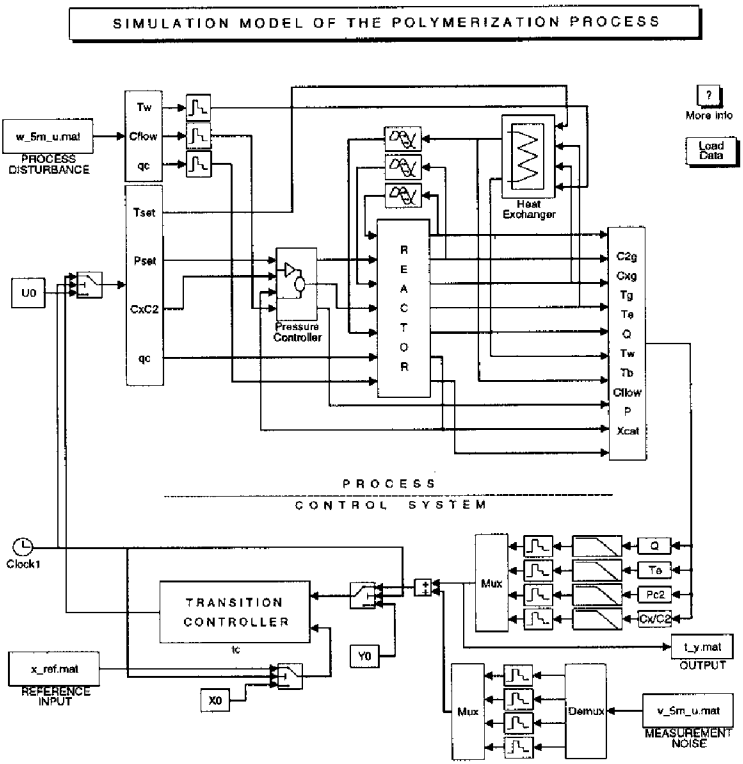


Figure B.1: A global simulation diagram

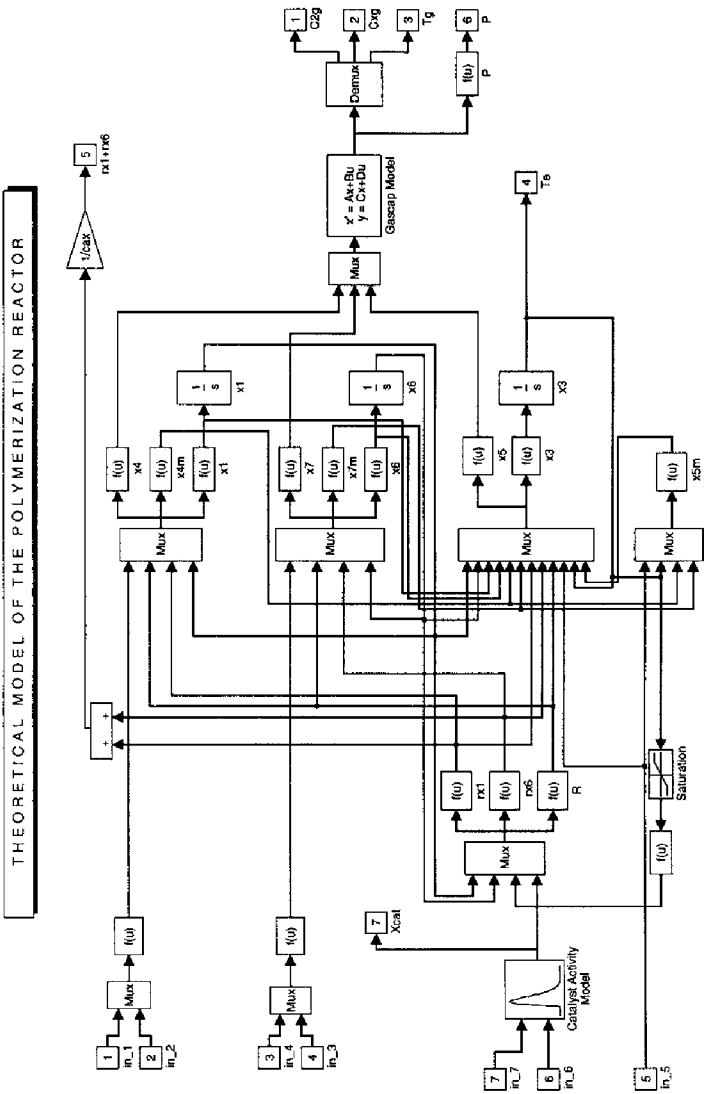


Figure B.2: Diagram of the reactor

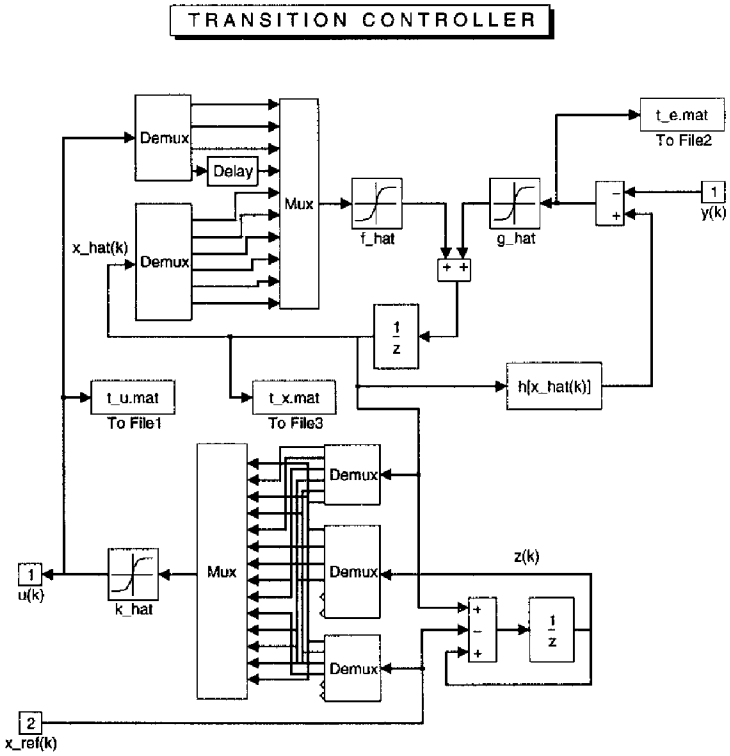


Figure B.3: Diagram of the controller with three neural networks

Bibliography

[1] P. E. An, M. Brown, S. Chen, and C. J. Harris. Comparative aspects of neural network algorithms for on-line modelling of dynamic processes. *Journal of Institute of Mechanical Engineering*, 207:223–241, 1993.

[2] B. R. Bakshi and G. Stephanopoulos. Wave-net: a multiresolution, hierarchical neural network with localized learning. *AIChE Journal*, 39(1):57–81, 1993.

[3] A. Ben-Tal, A. Melman, and J. Zowe. Curved search methods for unconstrained optimization. *Optimization*, 21(5):669–695, 1990.

[4] M. L. A. Bierman. Identification of a non-linear dynamic system using recurrent multirate sampled data neural networks. Report on practical training period, Eindhoven University of Technology, Measurement and Control Group, Eidhoven, August 1996.

[5] S. A. Billings and W. S. F. Voon. Correlation based model validity tests in the identification of nonlinear systems. *International Journal of Control*, 44, 1986.

[6] D. Braess. *Nonlinear approximation theory.* Springer, Berlin, 1986.

[7] B. D. Bunday and G. R. Garside. *Optimization methods in PASCAL.* Edward Arnold Ltd, London, 1987.

[8] S. Chen and S. A. Billings. Representations of nonlinear systems: the narmax model. *International Journal of Control*, 49(3):1013–32, 1989.

[9] S. Chen and S. A. Billings. Neural networks for nonlinear dynamic system modeling and identification. *International Journal of Control*, 56(2):319–46, 1992.

[10] K. Y. Choi and W. H. Ray. Fluidized bed reactors for ethylene and propylene polymerization. *Chemical Engineering Science*, 40(12):2261–2279, 1985.

[11] O. L. Chua. Global optimization: A naive approach. *IEEE Transactions on Circuits and Systems*, 37(7):966–969, 1990.

[12] A. Corana, M. Marchesi, C. Martini, and S. Ridella. Minimizing multimodal functions of continuous variables with the simulated annealing algorithm. *ACM Trans. on Mathematical Software*, 13(3):262–280, 1987.

[13] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

[14] J. F. Davidson, D. Harrison, and R. Clift. *Fluidization*. Academic Press, London, 1985.

[15] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *Computer Journal*, 7:149–154, 1964.

[16] K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183–192, 1989.

[17] M. J. M. Garderen. Numerical solutions for nonlinear constrained optimal control problems in discrete time. Master's thesis, Eindhoven University of Technology, Faculty of Electrical Engineering, 1995.

[18] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, November 1984.

[19] P. C. Goosen. Simulation of an overhead crane using neural networks. Master's thesis, Eindhoven University of Technology, Faculty of Electrical Engineering, 1992.

[20] R. Hecht-Nielsen. Theory of the backpropagation neural network. In *Proceedings of the International Joint Conference on Neural Networks*, volume 1, pages 657–664, San Diego, 1989.

[21] R. Hecht-Nielsen. *Neurocomputing*. Addison-Westley, Amsterdam, 1990.

[22] M. R. Hestens and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Research Journal of the National Bureau of Standards*, 49:409–436, 1952.

[23] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4:251–257, 1991.

[24] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[25] A. L. Ingber. Very fast simulated re-annealing. *Mathematical and Computer Modelling*, 12(8):967–973, August 1989.

[26] R.A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307, 1988.

[27] R.A. Jarvis. Adaptive global search by the process of competitive evolution. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-5(3), 1975.

[28] P. J. Larcombe. The dynamic equations of motion for a horizontally translating two dimensional it N-link inverted pendulum: A symbolic derivation using lagrangeś method. Control Group Research Report R-90/12, University of Glasgow, Department of Mechanical Engineering, Glasgow, May 1990.

[29] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6:861–867, 1993.

[30] S. Li. An optimized backpropagation with minimum norm weights. In *IJCNN International Joint Conference on Neural Networks*, volume 1, pages 697–702, San Diego, 1990.

[31] D. F. Liang. Comparison of nonlinear recursive filters for systems with non-neglible nonlinearities. In C. T. Leondes, editor, *Control and Dynamic Systems*, volume 20, pages 341–401. Academic Press, 1983.

[32] R. P. Lippmann. An introduction to computing with neural nets. *IEEE Acoustics, Speech and Signal Processing Magazine*, pages 4–22, april 1987.

[33] L. Ljung. *System identification: theory for the user*. Prentice-Hall, Englewood Cliffs, 1987.

[34] J. M. Maciejowski and R. J. Ober. Balanced parametrizations and canonical forms for system identification. In *Identification and system parameter estimation*, volume 2, pages 701–8, Beijing, 1988.

[35] T. Matsuoka and T. J. Ulrych. Information theory measures with application to model identification. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(3):511–517, 1986.

[36] J. Mazák and J. L. F. Balseiro. Modeling of a fluidized bed reactor for ethylene polymerization. Research project, Eindhoven University of Technology, Dept. of Electrical Engineering, Eindhoven, The Netherlands, February 1996.

[37] J. Mazák, A. A. H. Damen, and A. C. P. M. Backx. Neural state transition controller for a polymerization reactor. *Journal A*, 37(3):34–9, 1996.

[38] J. Mazák, A. A. H. Damen, and A. C. P. M. Backx. Nonlinear transition controller design using neural networks tested on a polymerization reactor. In *CDC*, 1996. to be published.

[39] J. Mazák, A. A. H. Damen, Weiland. S., and A. C. P. M. Backx. State-tracking control of non-linear systems with unknown dynamics in the presence of disturbances. In *Proceedings of the American Control Conference*, volume 6, pages 4270–4274, 1995.

[40] T. P. McGarty. *Stochastic systems and state estimation*. John Wiley & Sons, Inc., New York, 1974.

[41] V. S. Mikhalevich, N. N. Redkovskij, and A. A. Antonyuk. Minimization methods for smooth nonconvex functions. *Cybernetics*, 24(4):395–403, 1988.

[42] B. C. Mikhalevitch, A. M. Gupal, and B. I. Nopkin. *Nonconvex optimization methods.* Nauka, Moskva, 1987. in russian.

[43] M. Minsky and S. Papert. *Perceptrons : an introduction to computational geometry.* M.I.T. Press, Cambridge, 1969.

[44] J. Moody and C. J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–94, 1989.

[45] K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, 1990.

[46] L. Nazareth. A conjugate direction algorithm without line searches. *Journal of optimization theory and applications*, 23(3):373–87, 1977.

[47] The Numerical Algorithm Group Limited. *The NAG Fortran Library Manual - Mark 13*, 1988.

[48] J. Park and I. W. Sandberg. Universal approximation using radial-basis-functions. *Neural Computation*, 3:246–57, 1991.

[49] B. T. Polyak. *Introduction to optimization.* Optimization Software, New York, 1987.

[50] M. Post. Modelling and control of a fluidized bed reactor for ethylene polymerization. Technical Report R92.177/MP, IPCOS b.v., 1992.

[51] D. Psaltis, A. Sideris, and A. A. Ymamura. A multilayer neural network controller. *IEEE Control Systems Magazine*, 8(2):17–21, 1988.

[52] G. Qiu, M. R. Varley, and T. J. Terrell. Accelerated training of backpropagation networks by using adaptive momentum step. *Electronics Letters*, 28(4):377–8, 1992.

[53] N. N. Redkovskij. A minimization method with nonlinear transformation of coordinates. *Dokl. Akad. Nauk SSSR*, 288(3):556–560, 1986.

[54] F. Rosenblatt. *Principles of neurodynamics: perceptrons and the theory of brain mechanisms.* Spartan Books, 1962.

[55] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol.1 Foundations.* Cambridge, MA: MIT Press, 1986.

[56] L. E. Scales. *Introduction to Non-linear Optimization.* MacMillan, London, 1985.

[57] D. F. Shanno. Conjugate gradient methods with inexact searches. *Mathematics of operations research*, 3(3):244–56, 1978.

[58] D. F. Shanno and K. H. Phua. Algorithm 500: Minimization of unconstrained multivariate functions [E4]. *ACM Transactions on Mathematical Software*, 2(1):87–94, 1976.

[59] F. M. Silva and L. B. Almeida. Speeding up backpropagation. In R. Eckmiller, editor, *Advanced Neural Computers*, pages 151–58. North-Holland, Amsterdam, 1990.

[60] J. Sjoberg and L. Ljung. Overtraining, regularization, and searching for minimum in neural networks. In L. Dugard, M. M'Saad, and I.D. Landau, editors, *Adaptive Systems in Control and Signal Processing 1992. Selected Papers from the 4th IFAC Symposium*, pages 73–8. Pergamon Press, 1992.

[61] H. W. Sorenson. Kalman filtering techniques. In Harold W. Sorensen, editor, *Kalman Filtering: Theory and Application*, pages 90–126. IEEE Press, 1985.

[62] H. T. Su and T. J. McAvoy. Neural model predictive control of nonlinear chemical processes. In P.K. Simpson, editor, *Neural networks applications*, New York, 1996. IEEE.

[63] J. Sun, W. I. Grosky, and M. H. Hassoun. A fast algorithm for finding global minima of error functions in layered neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, volume 1, pages 715–20, San Diego, CA, 1990.

[64] J. Suykens. *Artificial neural networks for modelling and control of nonlinear systems*. PhD thesis, Katolieke Universiteit Leuven, Leuven, 1995.

[65] J. Suykens, B. De Moor, and J. Vandewalle. Static and dynamic stabilizing neural controllers applicable to transition between equilibrium points. *Neural Networks*, 7(5):819–831, 1994.

[66] J. L. Synge and B. A. Griffith. *Principles of mechanics*. MacGraw-Hill, Kogakusha, 1970.

[67] H. Szu and R. Hartley. Fast simulated annealing. *Phys. lett. A*, 122:157–162, 1987.

[68] J. Tao. Control completely unknown dynamical systems using critic adaptive neural networks. In H.C. Cihan H. Dagli, M. Akay, and P. [et al.] Chen, editors, *Intelligent engineering systems through artificial neural networks*, volume 5, pages 949–954, St. Louis, 1995.

[69] H. J. M. Telkamp and A. A. H. Damen. Neural network learning in nonlinear system identification and control design. In *Proc. of the European Control Conf.*, Groningen, The Netherlands, 1993.

[70] G. T. Timmer. *Global optimization: A stochastic approach.* PhD thesis, Centrum for wiskunde en informatica, Amsterdam, 1984.

[71] A. Törn and A. Žilinskas. *Global Optimization.* Lecture Notes in Computer Science. Springer-Verlag, 1989.

[72] S. Weiland. *Theory of approximation and disturbance attenuation for linear systems.* PhD thesis, Rijksuniversiteit Groningen, Groningen, 1991.

[73] M. K. Weir. A method for self-determination of adaptive learning rates in back propagation. *Neural Networks*, 4:371–79, 1991.

[74] P. J. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[75] J. C. Willems. Paradigms and puzzles in the theory of dynamical systems. *IEEE Transactions on Automatic Control*, 36(3), 1991.

Samenvatting

Dit proefschrift is gewijd aan een studie van het gebruik van neurale netwerken voor het ontwerp van regelaars die de proces toestand van het ene werkpunt naar het andere sturen. Het voordeel van onze aanpak is, dat we met behulp van één enkele niet-lineaire regelaar een breed bereik van process werkpunten kunnen bestrijken. Voor praktische toepassingen betekent dit een versnelling van de overgang van de procestoestand tussen verschillende werkpunten terwijl toch een goed prestatieniveau wordt gehandhaafd.

Onze aanpak bestrijkt alle stadia van een praktisch regelaarontwerp. We beschouwen: (1) proces modellering in een vorm van grey-box neurale modellen in toestandsruimte beschrijving (2) proces toestandsschatting door middel van het ontwerp van een niet-lineaire neurale toestandsobserver en tenslotte (3) regelingsaspecten met betrekking tot een niet-lineaire neurale regelaar met behulp van toestandsterugkoppling.

Als mathematisch model voor het proces wordt een niet-lineair toestandsruimte model beschouwd, geparametriseerd door een combinatie van een a priori bekend analytisch deel en een black-box neuraal netwerk deel. In de toestandsvector van het model onderscheiden we witte, fysische goed gedefinieerde toestanden en zwarte of "verborgen" toestanden. Het model van het proces wordt geschat als een simulatiemodel om een goede simulatie van de procesuitgang te krijgen over een lange horizon. Het neurale net van het model wordt getraind, in een output-error configuratie, met behulp van gemeten ingangs- en uitgangsdata. Door de keuze van een toestandsruimteparametrisatie van het model kan a-priori kennis over het proces op een conceptueel eenvoudige manier worden opgenomen. Hierdoor kunnen we ook, in een later stadium, goede referentie signalen voor de regelaar definiëren.

Het simulatiemodel van het proces wordt dan aangevuld met een niet-lineair filter, geparametriseerd door een statisch neuraal netwerk, om zo de toestandsvoorspellingen verkregen door het eerder geschatte simulatiemodel te verbeteren. Verschillende manieren voor de parametrisatie van het filter worden beschouwd in dit proefschrift. Het neurale netwerk voor de filtering wordt getraind met behulp van gemeten procesdata, onafhankelijke van de simulatiemodelparametrisatie. Dit completeert de tweede stap van het voorgestelde regelaarontwerp.

De transitie-regelaar is een niet-lineaire statische toestandsterugkoppeling, eveneens geparametriseerd door een neuraal netwerk. Het regelaar-netwerk wordt getraind op het simulatiemodel van het proces zodanig dat de modeltoestanden voorgeschreven referentietrajectoriën volgen. Dit proefschrift bevat een vergelijking tussen verschillende keuzes voor de toestandreferentiesignalen, inclusief een optimale keuze. Om eindfouten in het volggedrag te elimineren wordt een inte-

gratieactie opgenomen in de gesloten lus configuratie. Aan de randvoorwaarden voor bedrijving van het proces wordt voldaan door een juiste specificatie van de referentiesignalen. Deze worden verkregen met behulp van het witte gedeelte van het model en een aangepaste keuze van weegfactoren in een regelcriterium.

Alle niet-lineaire functies die in verschillende stadia van ons algoritme worden geschat zijn geparametriseerd door sigmoïdale feedforward neurale netwerken. Voor de training van de neurale netwerken maken we onderscheid tussen gradiëntgebaseerde, deterministische optimalisatie en stochastische optimalisatie. Een aantal methoden wordt in dit proefschrift besproken om tot een effectieve combinatie van deze twee optimalisatietechnieken te komen. Deze gecombineerde techniek wordt vervolgens gebruikt voor de training van de neurale netwerken.

In dit proefschrift wordt een aantal voorbeelden gegeven die zowel de modelleringsaspecten als de regelaspecten van dit proefschrift demonstreren. De belangrijkste voorbeelden zijn: (1) een portaalkraan. Hiermee wordt de modelleringsprocedure voor niet-lineaire processen gedemonstreerd. (2) een fluidized bed polymerisatie proces. Hiermee worden zowel modellerings- als regelaspecten gedemonstreerd. (3) een niet-lineaire toestandsterugkoppeling wordt beschouwd voor een regeling voor een inverse slinger met meerdere links.

Curriculum Vitae

Jozef Mazák was born on March 19^{th}, 1963 in Trnava, Slovakia. In 1981 he started his university study at the Faculty of Electrical Engineering of the Slovak Technical University. He studied at the Department of Automatic Control Systems for Technological Processes where he successfully graduated in 1986. In 1987 he started his postgraduate study in the field of control theory. After the political changes in Slovakia he left the country in 1991 and started to study telecommunications at the Eindhoven International Institute. In May 1992 he received his master's degree in telecommunications. Immediately after that he started his Ph.D. research at Eindhoven University of Technology in the Measurement and Control Section, Faculty of Electrical Engineering.

Jozef Mazák is married to Diana and has a son Dominik.

STELLINGEN

behorende bij het proefschrift:

Transition Control Based on Grey, Neural States

van

Jozef Mazák

Eindhoven, 25 november 1996

I

There would be no learning, either natural or artificial, without the notion of an error:

Neural net (1949): a computer architecture in which a number of processors are interconnected in a manner suggestive of the connections between neurons in a human brain and which is able to learn by a process of trial and error – called also neural network. (*Merriam Webster's Collegiate Dictionary*, 10-th edition, Springfield, 1993)

II

Searching for a local minimum of a non-convex function formed by a neural network training problem is like searching for a needle in a haystack. Finding a global minimum is even worse. However, it is enough to find sufficiently low function values.

III

A structure with less than thousand neurons should not be called a Neural Network as it is much too simple. It could be called a "quasi Neural-Network" or let's say a "σ-function".

IV

Increasing the complexity of black boxes (e.g. neural nets) for the approximation of certain functions can be helpful in finding global minima which, in fact, already exist for low complexity: the increased complexity may offer more ways with an easier access towards the global minima.

V

The future development of general control design methods should reflect mainly the ideas and skills of the designer, rather than the brilliant theoretical methods currently described in the text books, as the majority of them can not be computed in practice, no matter how big and fast the computer is.

VI

The term "A global optimization method" is misleading. Its meaning is twofold: It denotes either an optimization method converging to a global minimum or a globally convergent method converging to a local minimum.

VII

Preventing computer crashes on the part of the user in the first place is a better idea than asking either the computer manufacturer or the system manager for an assistance solving them.

VIII

A unification of the "Vysegrad" countries with EC has to happen as soon as possible, not only in the interests of the Vysegrad countries, but also in the interests of Western countries who will benefit from the economic prosperity and the growth of household demand in the Vysegrad countries.

IX

Photography has a lot in common with neural networks. There are some basic formulas which always apply, the rest is based on experience obtained by trial and error.

X

As soon as there is an expert in the neighbourhood people tend to stop either reading manuals or thinking and ask the expert first.