Modelling Dynamic Systems with Artificial Neural Networks and Related Methods

Juš Kocijan

Nova Gorica 2023

Modelling Dynamic Systems with Artificial Neural Networks and Related Methods

Original title: Modeliranje dinamičnih sistemov z umetnimi nevronskimi mrežami in sorodnimi metodami

Author: Juš Kocijan

Edition: English edition

Reviewers of original: Prof. Aleš Belič and Prof. Igor Škrjanc

English translation: Juš Kocijan

Proofreading: LINGULA, jezikovni center, d.o.o.

Text formatting and cover layout: Juš Kocijan

Publisher: University of Nova Gorica Press, Vipavska 13, SI-5000 Nova Gorica, Slovenia

Publication year: 2023

ISBN: 978-961-7025-29-3 (PDF)

Published in PDF:

http://www.ung.si/sl/zalozba/

http://www.ung.si/en/publisher/

15 February 2023

Free e-publication.

Kataložni zapis o publikaciji (CIP) pripravili v Narodni in univerzitetni knjižnici v Ljubljani
COBISS.SI-ID 140589315
ISBN 978-961-7025-29-3 (PDF)

How to cite: Kocijan, J. (2023). Modelling Dynamic Systems with Artificial Neural Networks and Related Methods. University of Nova Gorica Press. https://www.ung.si/en/publisher/

Foreword

Modelling Dynamic Systems with Artificial Neural Networks and Related Methods can be used as a textbook for the field it covers or as an introductory textbook for more advanced literature in the field.

It is intended for undergraduate students, especially at the postgraduate level, who have sufficient knowledge of dynamic systems, as well as for professionals who wish to familiarise themselves with the concepts and views described.

The textbook is not intended to be a detailed theoretically based description of the subject but rather an overview of the identification of dynamic systems with neural networks and related methods from the perspective of systems theory and, in particular, its application. The work is intended to inform the reader about the views on this subject, which are related but treated very differently in different research fields.

I used the material for the textbook as a basis for lectures at the Faculty of Electrical Engineering of the University of Ljubljana. I would like to thank Miro Štrubelj, who created many of the figures, Dr. Gregor Gregorčič, who allowed me to use some of the pictures from his doctoral thesis, and all others who directly or indirectly influenced the creation of this work.

Ljubljana, Autumn 2007 *Juš Kocijan*

Addendum 2023

The English-language version of the textbook was created in 2022 for English-speaking students. The textbook follows its Slovenian original, except for some minor corrections and additions.

Ljubljana, Winter 2023 *Juš Kocijan*

Contents

Chapter 1

Introduction to artificial neural networks

Artificial neural networks (ANNs) are a concept and method for solving various problems that have been around for more than half a century. In this textbook, we will refer to them as 'neural networks'. Their applications are highly diverse, both in professional fields (engineering, computer science, natural sciences, social sciences, etc.) and by problem domain (regression, classification, clustering, etc.).

Neural networks are widely used for the experimental modelling of dynamic systems, especially complex and nonlinear systems, and consequently for the design of automatic control systems. In this textbook, we will highlight particular problems in this area.

It is important to bear in mind that this is an introductory overview that does not claim to be exhaustive but rather gives insight into the most common use of neural networks for the experimental modelling of nonlinear dynamic systems and some of the specific problems that arise in this context.

Neural networks are classified as one of the most commonly used methods in computational intelligence. The classification and the relationship to the methods of so-called artificial intelligence go back to the beginnings of neural networks. Artificial neural networks arose from the idea of replacing a biological neuron, as shown in Figure 1.1, with an artificial neuron. In this way, they would form the basic elements of an artificial brain.

This pattern of thinking was later surpassed by the realisation that neural networks are nothing more than basic mathematical functions for approximating an arbitrary nonlinear relation.

In the remainder of this chapter, a brief overview of the evolution of neural networks is given. This is followed by a classification of neural networks according to various criteria. Next, we will describe two types of the most commonly used neural networks: the multilayer perceptron and the radial basis-function network. We will then describe how neural networks can be used to model dynamic systems.

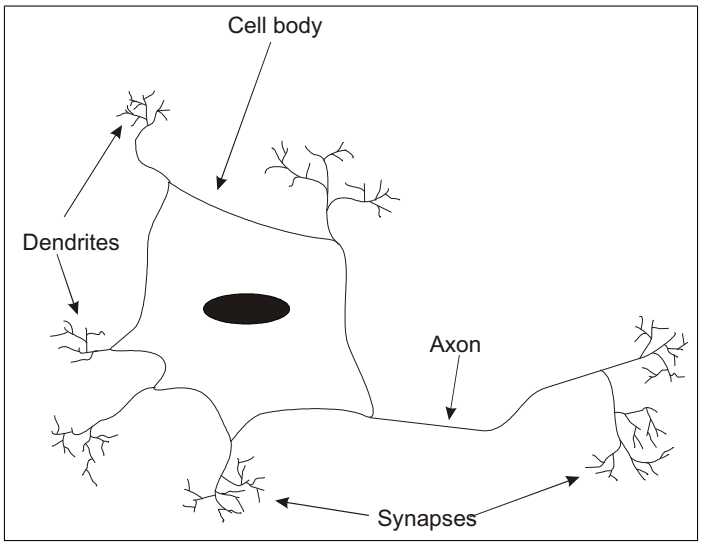

Figure 1.1: Biological model of an artificial neuron

1.1 Short overview of the development of the topic

In this subsection, we summarise the main milestones in the development of artificial neural networks. The aim is to illustrate the evolution of the perspectives that have emerged with the developments. The overview is based on [10] from a systems perspective. It is a holistic view of the neural network and its properties in terms of inputs and outputs.

1943 In this year, McCulloch and Pitts [12] presented models of biological neurons, which were the basic elements of circuits for solving computational problems. The idea was that they would replace the functions of a biological neuron.

1949 Hebb, a psychologist by profession, published a book [7] in which he described neuron learning as tuning the weights of connections between units (i.e., neurons). The various and much later developed rules for parameterising the weights in neural networks, which can be represented as units with connections, are essentially derivatives of this method.

1959 Rosenblatt [21] described a single-layer neural network consisting of elements which he called a 'perceptron'. Neural networks consisting of these elements, albeit in a modified form, are still among the most commonly used neural networks today. The basic elements of the neural network are the switching functions; as already mentioned, the network has only one layer. This means that the input signal from a particular input passes through only one layer to the output neuron. This simple structure limits the possibilities of the neural network.

1960 Widrow and Hoff [24] introduced a neural network of linear elements called Adaline (ADAptive LINear Element). This network can be represented electrically as inputs weighted by resistors and connected to a sumator. In principle, a neuron is a linear function, and the Adaline network approximates input-output mapping by using a weighted sum of linear functions. The authors also derived the first analytical method for determining the weights, which they called the delta rule. This is essentially the least-squares method for optimisation.

1963 Widrow and Smith [25] used the Adaline neural network to stabilise an inverted pendulum.

1969 In their book 'Perceptrons' [13], Minsky and Papert showed the limitations of (single-layer) perceptron neural networks. They cannot be used, for example, to separate the elements that are not linearly separable. This means that a single-layer network cannot be used to train to imitate the logical function 'exclusive or' (XOR). The authors have also demonstrated that the limitations of single-layer networks can be overcome with a two-layer network, but they did not show how to set the weights of a multilayer network.

The publication of [13] marked the end of the era of single-layer neural networks; the results led to a gap in research on neural networks.

1986 Rumelhart et al. [22] published a method for learning weights for a multi-level network. They called this learning method 'backpropagation' because it was interpreted as error propagation from the outputs of the neural network to the inputs. With this work, they enabled the use of the multilayer perceptron for the classification of functions that are not linearly separable or can represent an arbitrary nonlinearity. This finding led to a boom in the research and use of artificial neural networks.

late 1980s Artificial neural networks began to be used for automatic control, e.g., [20].

1990 Narendra and Parthasarathy [14] used neural networks to identify and control dynamic systems.

1995 Sjoeberg et al. showed [23] that neural networks can be viewed from a systems perspective as identification using nonlinear regression. They thus removed the artificial intelligence aspect from neural networks. These and similar methods were renamed 'methods of computational intelligence' to avoid unnecessary misunderstandings.

after 2010 Deep learning greatly extends the capabilities of neural networks, especially in speech, image, object recognition, and beyond.

1.2 About artificial neural networks

The basic building block of a neural network is a neuron. It is described by a function

$$y_i = f(\sum_j w_{ij} x_j + \theta_i), \tag{1.1}$$

where x_j are the inputs of the neuron, weighted by the values of w_{ij} are weighted and together with the bias values θ_i and the activation function $f(\cdot)$ determine the outputs of the neuron y_i. The artificial neuron is shown in Figure 1.2.

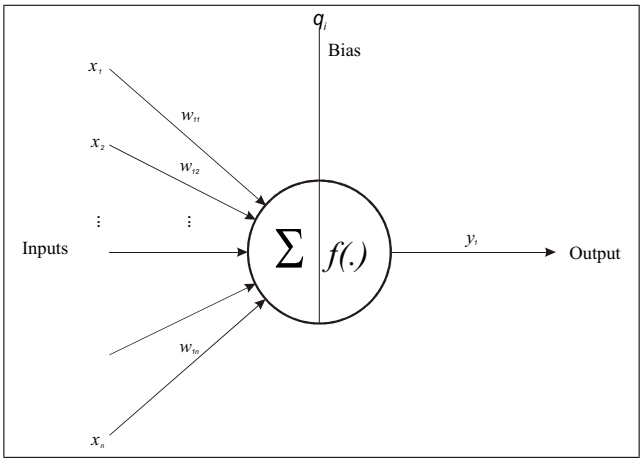

Figure 1.2: Representation of the basic element of a neural network - a neuron

This formal form of the neuron was defined by McCulloch and Pitts [12], and the form has essentially remained unchanged to the present day. The activation function can have different forms. Similarly, neurons can be connected in different ways. Neural networks have evolved in many directions and are classified in different ways. The most common classification criteria are the training, the topology, or the purpose of the neural networks.

Training of neural networks

The parameters of a neural network are usually understood to be the weights and biases that must be determined by an optimisation procedure. This parameterisation is called learning or training. There are three types

of learning or training: supervised learning, unsupervised learning and reinforcement learning.

Supervised learning starts with a set of input data and a target set of output data, and according to the chosen cost function, a neural network is trained to map between the given input and output data. The first methods were developed for single-layer neural networks. These are the perceptron learning rule (Hebb's rule)[7]:

$$\Delta w_{ij} = \gamma y_i x_j \ ,$$

where Δw_{ij} is the weight change and γ is a constant that determines the learning rate, and the delta rule (Widrow-Hoff rule)[24]:

$$\Delta w_{ij} = \gamma x (d_i - y_i) x_j \ ,$$

where $(d_i - y_i)$ is the difference between the desired value and the actual output value. This rule essentially turns out to be the least squares method of optimisation.

The oldest training method for multilayer networks is the backpropagation method. This is essentially the Delta rule, generalised for nonlinear problems. A closer look at the method shows that it is a first-order gradient optimisation method, specifically the steepest descent method. Backpropagation is used to calculate the gradient of the cost function. This is applicable to all types of nonlinear systems but differs in computational complexity depending on the system. To improve its weaknesses, which are mainly slow convergence and the possibility that the optimisation becomes stuck in a local minimum, some improved versions have been developed, such as [10]:

- learning with momentum and
- variable learning rate.

First-order gradient methods are known to be inefficient; Newton optimisation methods, which belong to second-order gradient methods, are often used to solve nonlinear optimisation problems. Among the most commonly used are the Newton optimisation methods with

- Gauss-Newton modification and
- Levenberg-Marquardt modification.

Self-organised neural networks, unlike supervised networks, use unsupervised learning. Self-organised networks are trained from input data only. They are mainly used in pattern recognition for clustering, vector quantisation and dimensionality reduction. A typical example is the Kohonen network, which is also called the Self-Organising Map (SOM) [6].

Reinforcement learning is an approach to dynamic programming for training the neural network used as a feedback controller. This means that it has a specific application. The learning consists of two phases: parameter estimation and parameter evaluation. It is shown schematically in Figure 1.3.

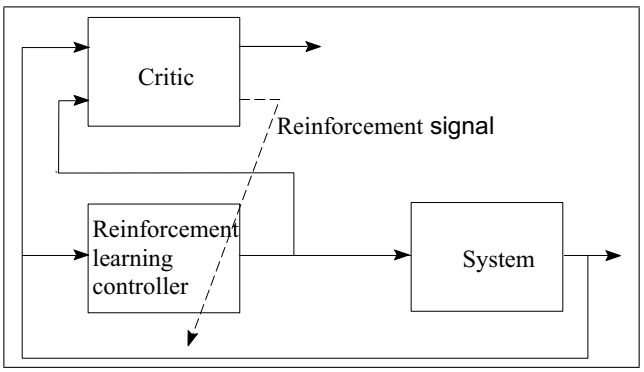

Figure 1.3: Reinforcement learning schematic

As new optimisation methods are developed, they are also used for neural network learning. Stochastic optimisation methods, such as searching for parameters and structures using evolutionary methods (genetic algorithms and genetic programming), are also popular.

Recently, deep learning [2] has been developed and has become a very popular training method.

Topology of neural networks

According to their topology, neural networks are divided into those with complete connectivity, local connectivity. and layer-wise connectivity. For example, the Hopfield neural network is a neural network with perfect connectivity.

The Hopfield neural network has a switching activation function or a linear function with saturation. Its characteristic feature is a feedback loop from outputs to inputs. It is called 'associative memory' and is mainly interesting from a theoretical point of view. It is less useful in practice. Boltzmann machines are a derivative.

Boltzmann machines are Hopfield networks with hidden layers. Their parameters (i.e., the weights) are determined using a stochastic rule of weight variation, called 'simulated annealing'.

An example of a neural network with local connectivity is the Kohonen neural network. It is also an example of a network that is mainly used for data clustering. Its main feature is that it maintains topological connectivity (mapping) between data.

An example of a neural network with multilayer mapping and connectivity is the multilayer perceptron, which will be discussed in greater detail later.

Another possible division is into feedforward and recurrent neural networks (Figure 1.4). This division is based on the direction of the signal or data flow through the network. Feedforward networks mainly have direct flow from

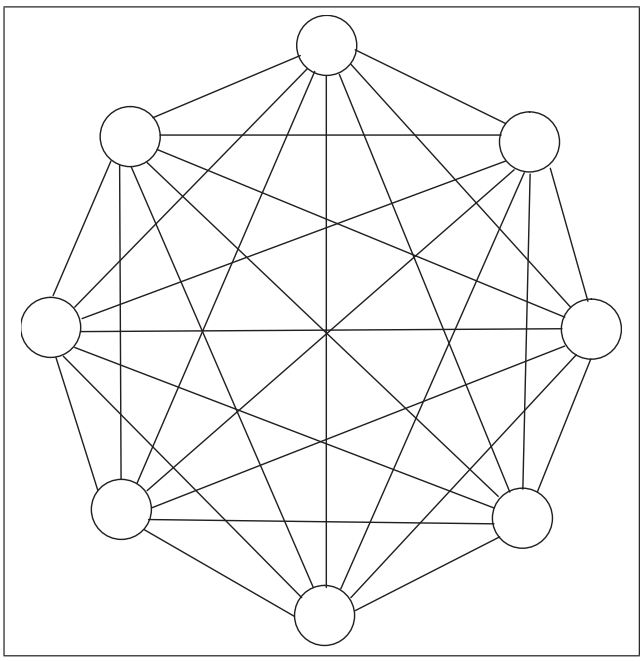

Figure 1.4: Example of recurrent network topology

input to output, while recurrent networks have full and local connectivity. Recurrent networks differ from feedforward networks, where signals flow exclusively from input to output, in that recurrent networks also contain feedback loops from the outputs.

Purpose of neural networks

Neural networks can be divided coarsely or finely according to their purpose. At a coarser level, neural networks can be divided into those for classification and those for regression.

Classification is a process in which the data are sorted into a finite number of sets. This means that mapping is done in a quantised output space. Regression is a process in which the output values can take on any value. In regression, an input-output mapping is performed in a continuous output space.

In this textbook, the focus is on using neural networks and similar approaches to model dynamic systems and using these models to design automatic control. Therefore, classification and special neural networks will not be discussed. The focus will be exclusively on regression, This means that supervised learning of mainly feedforward networks will be examined.

Main properties of neural networks

A neural network is a nonlinear mapping from the input space $\mathcal{R}^n$ to the output space $\mathcal{R}^m$ of the data.

$$f : \mathcal{R}^n \Rightarrow \mathcal{R}^m \tag{1.2}$$

with input-output data pairs $(\mathbf{x}, y)$ such that $y = f(\mathbf{x})$. Neural networks are universal approximators [8]. This means that there is always a neural network with the appropriate dimensions that approximates the nonlinear objective function with arbitrary accuracy.

It is often interpreted from the literature that neural networks have the ability to learn from examples. This essentially means that the parameters of the network are determined by optimisation with respect to the chosen cost function, for example:

$$E(k) = \frac{1}{2}\sum_{j=1}^{n}(d_j(k) - y_j(k))^2 \tag{1.3}$$

and with the chosen optimisation algorithm (learning rule).

The neural network that learns in this way can use the learned association for prediction, which is often interpreted as the ability to generalise.

There is no single definition of neural networks. There are only common properties that unite the different models. These common properties of neural networks are:

- a structure that consists of a large number of simple elements connected to each other,
- they usually have a variable topology, which means that the connections change or the values of the weights on the connections change or the number of elements in the network changes,
- the structure allows for parallel processing of information,
- the information is processed according to the latent (i.e., internal) states of the network and the inputs to the network.

There are many other common properties that distinguish neural networks from each other:

- the methodology of the processes,
- the types of networks,
- the learning rules,
- the application domains, and
- other features.

Artificial neural networks today

Neural networks have long ceased to be merely an area of research. They have been a magnet for research since 1985. Figure 1.5 shows the number of publications in the area of dynamic-systems control over the last century. Since

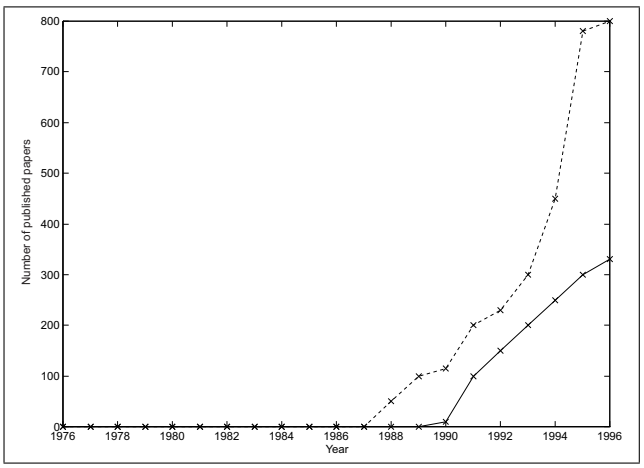

Figure 1.5: The growth of the number of publications - artificial neural networks for control (Source: G.W. Ng: Application of Neural Networks to Adaptive Control of Nonlinear Systems)[16]

then, the number of publications has continued to increase. Moreover, it is a technology that is currently used in many applications.

Domains and applications of neural networks include:

- aviation (autopilot, fault detection, etc.),
- automotive systems (automatic steering, etc.),
- banking (document recognition, etc.),
- electronics (steering, pattern recognition, etc.),
- language (recognition, etc.),
- manufacturing (guidance, prediction, etc.),
- medicine (signal analysis, etc.),
- accounting (various analyses, etc.),
- robotics (guidance, sensing, etc.),
- telecommunications (data compression, etc.),
- transportation (diagnostic systems, etc.),
- military industry (signal processing (radar, sonar, etc.),
- entertainment industry (animation, special effects, etc.),
- insurance (value optimisation, etc.),
- other.

Have artificial neural networks fulfilled the expectations that the researchers had in the initial phase of development?

In the early days, researchers were mainly interested in understanding how biological systems work with the help of mathematical models. There was a very strong connection between biological research and the research of artificial neural networks. Expectations were high. The neural network was supposed to be the beginning of the artificial intelligence. As we have already said, it turned out not to be so. On the one hand, it was a disappointment, but on the other, a new technology that is very well suited for efficient methodologies for nonlinear mappings had arrived.

One of the areas where neural networks are very useful is the modelling of dynamic systems and their use for automatic control. The main application of neural networks is the identification of nonlinear dynamic systems, but there are also implementations in automatic control. We have devoted a separate chapter to this.

The extensive literature on neural networks is also followed by software that can be used to work with neural networks. One of the better-known programs for system identification is based on the Matlab software package (e.g., [15] and [17]).

1.3 Multilayer perceptron

A multilayer perceptron, shown schematically in Figure 1.6, is described by the following equation.

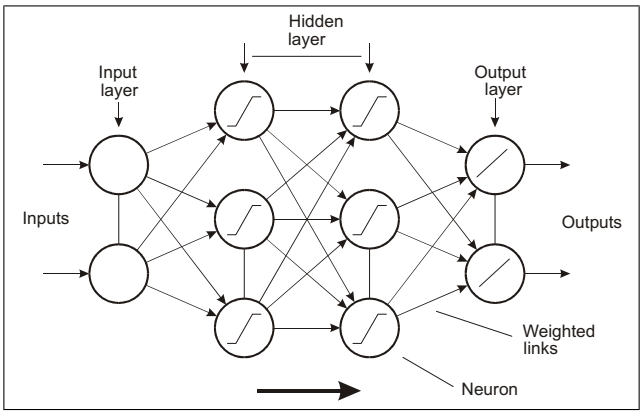

Figure 1.6: Multilayer perceptron

$$y_i = f_i(\sum_j w_{ij} f_n(\sum_k w_{jk} f_m(\sum_l w_{kl} x_l + w_{0l}^1) + w_{0k}^2) + w_{0j}^3). \tag{1.4}$$

It is called multilayer because it consists of at least one hidden layer (the activation functions f_n, f_m) and an output layer f_i. The multilayer perceptron is a universal approximator [8], which is true if it has one or more hidden layers.

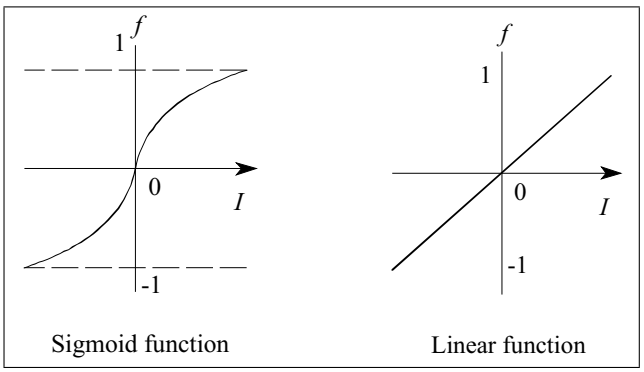

Figure 1.7: Two examples of activation functions

The most common activation functions (examples in Figure 1.7) are:

- sigmoid,
- hyperbolic tangent,
- linear,
- Gaussian,
- signum function.

The activation function in the hidden layer is often the sigmoid function.

$$f(I) = \frac{1}{1 + e^{-aI}} \ , \tag{1.5}$$

where $a, I \in \mathcal{R}$, while the most common output function is linear (Figure 1.7).

The multilayer perceptron is a typical example of a forward neural network used for regression. Let us look at two examples of how we can approximate a nonlinear function by a multilayer perceptron.

Example of approximating a function of one variable

We approximate the function

$$\begin{aligned} y &= 30\left[100(x-0.6)^2(x-0.1)(x-0.8) - 5e^{-5x}\right. \\ &+ \left. 0.9\sin(20x + \frac{\pi}{4})\right] \end{aligned} \tag{1.6}$$

where x is the input variable and y is the output variable, by a multilayer perceptron with a hidden layer. The activation function of the hidden layer is a sigmoid function, while the output is a linear function.

We approximate the function in several steps by first optimising the weights of the network with one neuron in the hidden layer, then with two, and thus increasing the number of neurons in the hidden layer until we reach the desired approximation level. We use the Levenberg-Marquardt modification of the Newtonian optimisation method, or the Levenberg-Marquardt learning method for training (i.e., optimisation). The results of the optimisation are shown in Figures 1.8 to 1.10. Finally, we obtain a neural network with five neurons in the hidden layer.

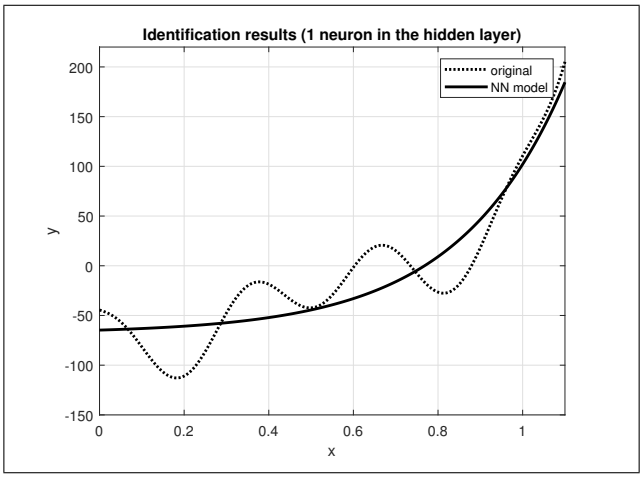

Figure 1.8: A multilayer perceptron with a neuron in the hidden layer

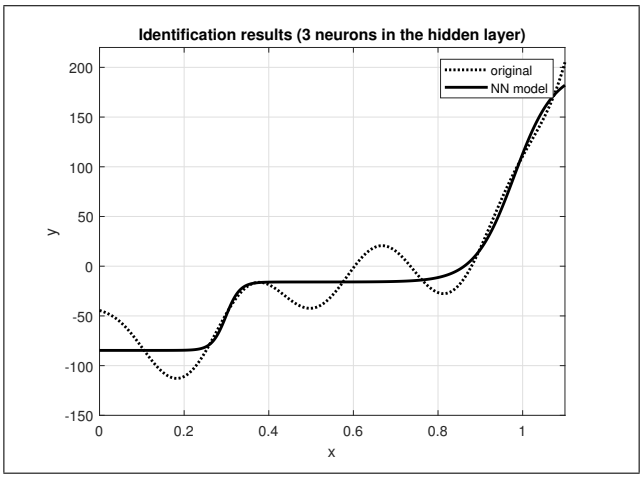

Figure 1.9: A multilayer perceptron with two neurons in the hidden layer

Example of approximation of a function of two variables

Let the function

$$z(x, y) = x\cos(2x) + y\sin(2y) - 0.75 \tag{1.7}$$

where x and y are input variables and z is the output variable and are approximated by a multilayer perceptron with a hidden layer. Again, the activation function of the hidden layer is a sigmoid function and that of the output layer is a linear function. The structure of the hidden layer again consists of one to five neurons. The results are shown in Figures 1.11 to 1.13.

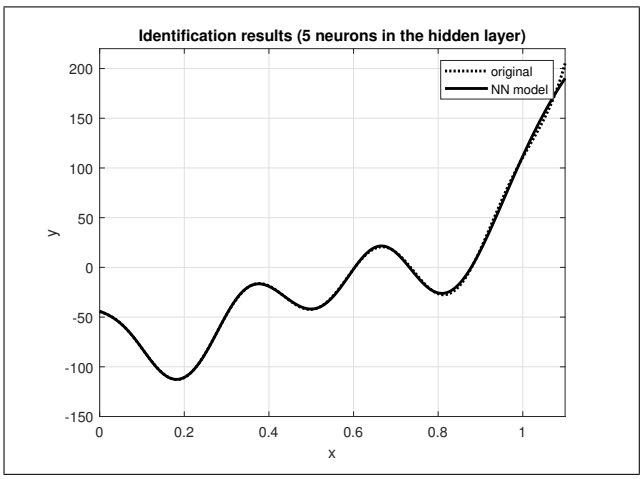

Figure 1.10: A multilayer perceptron with five neurons in the hidden layer

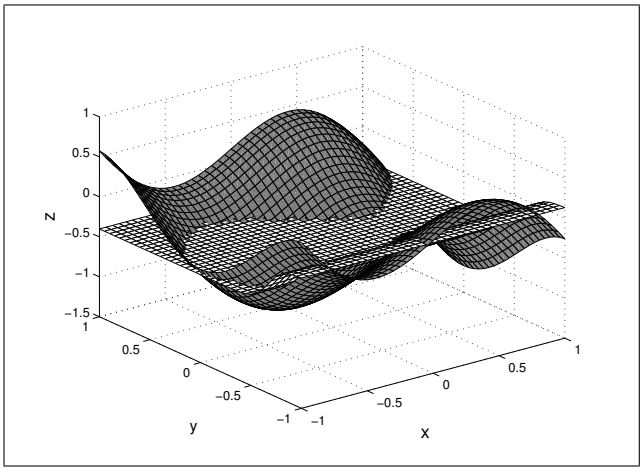

Figure 1.11: A multilayer perceptron with a neuron in the hidden layer. (target area is dark grey, learned area is white)

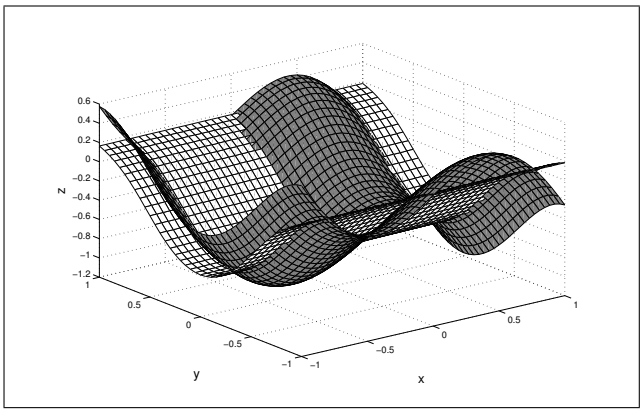

Figure 1.12: A multilayer perceptron with two neurons in the hidden layer. (target area is dark grey, learned area is white)

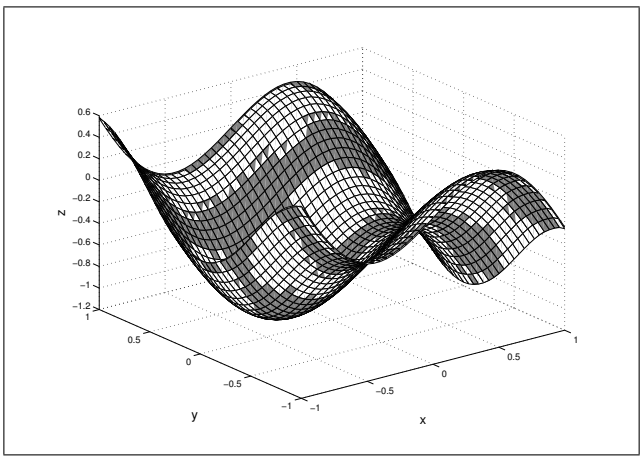

Figure 1.13: A multilayer perceptron with five neurons in the hidden layer. (target area is dark grey, learned area is white)

1.4 Radial basis-function network

The radial basis-function (RBF) is the second most commonly used neural network. The network contains nonlinear transformations of the input data, which (when weighted) sum to form the output of the network, as shown in Figure 1.14. The neural network in Figure 1.14 imple-

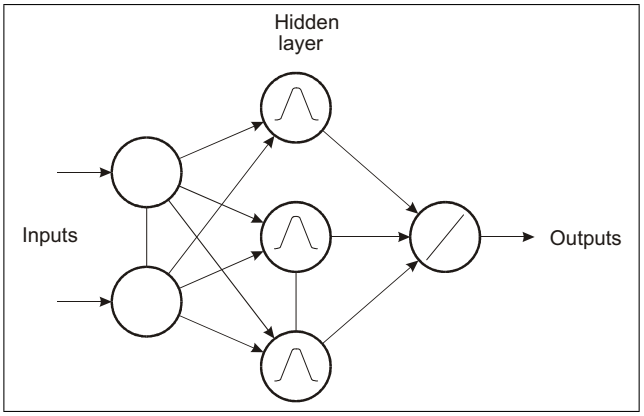

Figure 1.14: Radial basis-function network

ments the approximation function $f : \mathcal{R}^n \to \mathcal{R}$, which can be generalised to multiple outputs, described mathematically as follows:

$$y = f(\mathbf{x}) = \sum_{i=1}^{N} w_i g_i(\mathbf{x}, \gamma_i). \tag{1.8}$$

The function g is called the basis function and in the case of this neural network it is a radial function, meaning the function where the output depends on a distance between inputs r. Possible radial functions are:

- $g(r) = r$, a linear radial function,

- $g(r) = r^2$, a quadratic function,
- $g(r) = exp(\frac{-r^2}{\rho^2})$, a Gaussian function (Figure 1.15),
- $g(r) = [1 + exp(\frac{r^2}{\rho^2})^{-1}]$, logistic function,
- $g(r) = r^2 \log(r)$, various spline functions,
- $g(r) = (r^2 + \rho^2)^{\frac{1}{2}}$, polyquadratic function,
- other radial functions.

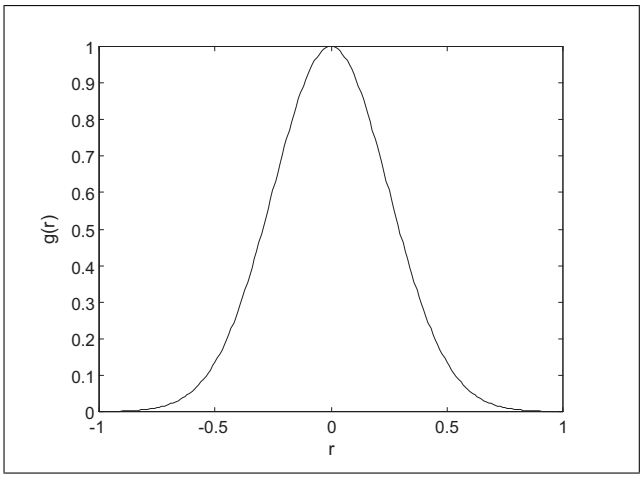

Figure 1.15: Gaussian basis function

For example, with multiple inputs, the Gaussian basis function would look like this:

$$g_i = \exp\left[-\frac{1}{2}\left(\frac{(x_1-\gamma_{i1})^2}{\rho_{i1}^2} + \frac{(x_2-\gamma_{i2})^2}{\rho_{i2}^2} + \ldots + \frac{(x_n-\gamma_{in})^2}{\rho_{in}^2}\right)\right], \quad (1.9)$$

where $x_j; j = 1, \ldots, n$ are input data, $\gamma_{ij}; j = 1, \ldots, n$ are centres of the radial function g_i and $\rho_{ij}; j = 1, \ldots, n$ are scaling factors of the radial function g_i. Radial functions often lie in N points of the input data (e.g., $\boldsymbol{\gamma}_i = \mathbf{x}_i \quad i = 1 \ldots n \leq N$). Alternatively, their position and choice of radii $\boldsymbol{\rho}_i$ are part of the optimisation process or learning. For an example of the distribution and an illustration of how the radial function is constructed, see Figures 1.16 and 1.17.

An important property of radial basis-function networks is their stability, which depends on the choice of radial basis functions. For example, since Gaussian basis functions have the property of approaching zero as they move away from the centre, this means that the output of the neural network is also bounded whenever the weights have finite values. This BIBO (bounded-input-bounded-output) stability is so important that the basis functions of RBF networks are mainly Gaussian functions. RBF networks are also universal approximators [19].

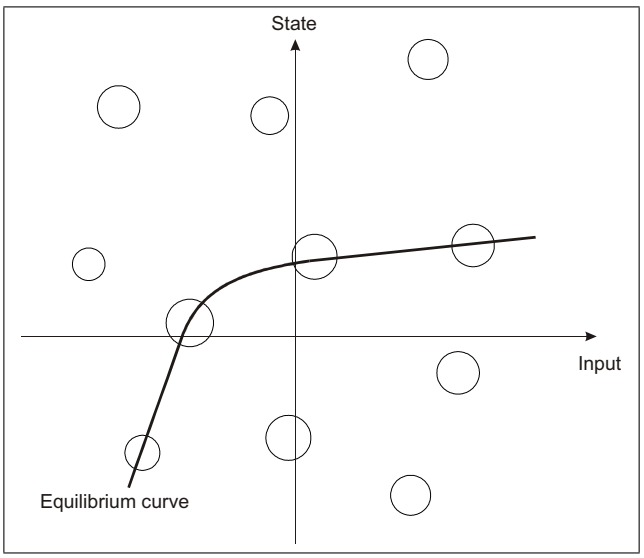

Figure 1.16: Distribution of RBF network centres

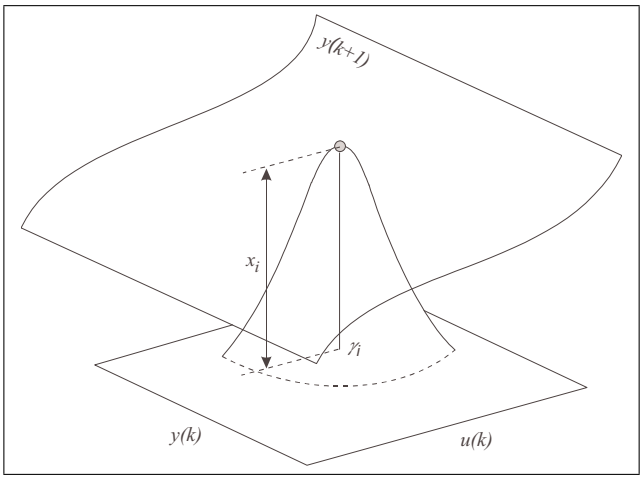

Figure 1.17: Specific RBF corresponding to the given nonlinearity

Since the weights w_i vary linearly with respect to the output, it is not difficult to find their optimal values with respect to the minimum squared error. The RBF network contains additional parameters $\boldsymbol{\gamma}_i$ and $\boldsymbol{\rho}_i$, whose values are generally unknown. If we optimise the values of these parameters in addition to the weights, the result is a nonlinear optimisation problem, like the optimisation of the multilayer perceptron was. A successful optimisation approach in this case is the orthogonal least squares (OLS) method.

Example of a function of two variables

Let the function

$$z(x, y) = \sin(0.4x^3 - 1.6y^2 + 0.5) \quad (1.10)$$

be approximated by an RBF network. We choose the Gaussian function as the basis function. We optimise the

weights of the grid with a different number of basis functions of width $\rho = 0.1$, which is increased until the approximation of the given function $z(x, y)$ is sufficient. The results are shown in Figures 1.18 to 1.21.

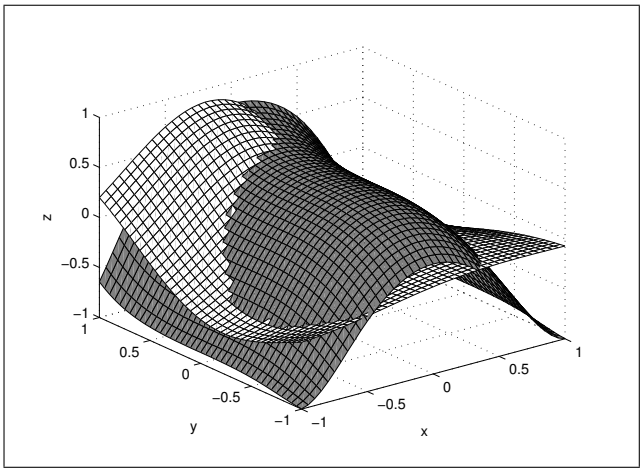

Figure 1.18: The network with three radial basis functions (the target surface is dark grey, the learned surface is white)

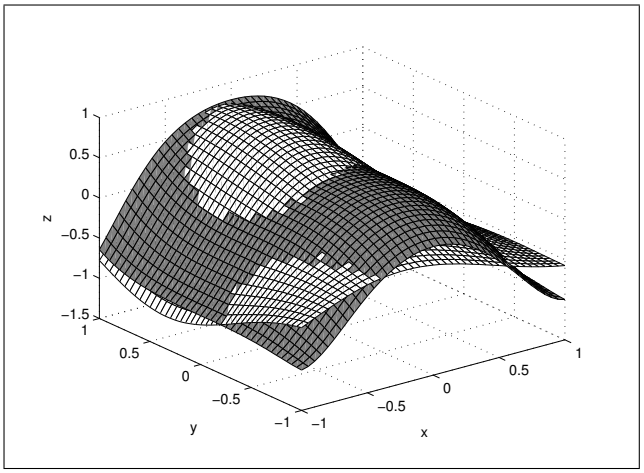

Figure 1.19: The network with five radial basis functions (the target surface is dark grey, the learned surface is white)

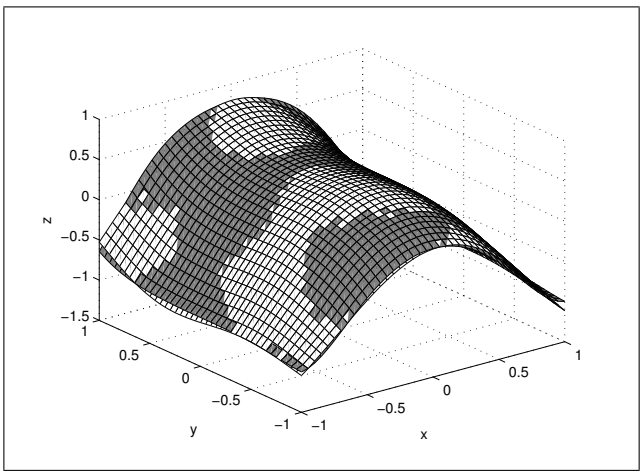

Figure 1.20: The network with eleven radial basis functions (the target surface is dark grey, the learned surface is white)

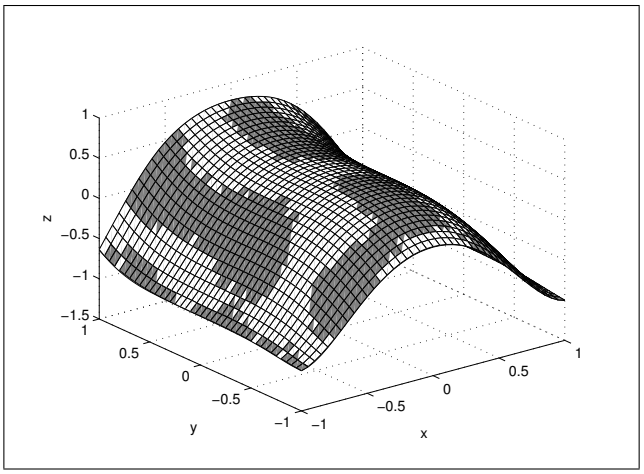

Figure 1.21: The network with thirty-one radial basis functions (the target area is dark grey, the learned surface is white)

1.5 Neural networks for modelling nonlinear dynamic systems

Neural networks, as described, are used to model static nonlinear functions. We optimise their weights to obtain the most appropriate relationship between the input and output data. When new data is given as input, the network representing the input/output mapping predicts the corresponding output values.

In dynamic systems, nonlinearities result in an output value that depends not only on the input value but also on the internal states of the system. Internal states are usually the past values of the input and output signals or their derivatives (depending on whether it is a discrete-time or a continuous-time system).

The usual representation of dynamic systems is that the memory (or derivative) is implemented outside the approximator of the static function, in the present case the neural network. Consequently, we also bring in the delayed (or differentiated) values of the inputs and outputs. In dynamic systems, we usually consider the input/output behaviour in terms of signals or time series of data, in contrast to static systems in which we deal with individual input and output values. When using static approximators, whether neural networks or other related methods, obtaining time sequences at the output means that the approximator performs its prediction separately at each time instant or step. This means that at the selected time

instant k, the neural network is optimised to predict the value of the dynamic system at the next time instant $k+1$ [3] given the values of the inputs consisting of the selected number of delayed values and a current value of the input signal u and the output signal y. This can be illustrated by Equation (1.11)

$$\hat{y}(k+1) = f(y(k), y(k-1), \ldots, u(k), \ldots), \qquad (1.11)$$

where $\hat{}$ denotes the prediction and k is the number of subsequent time instants. This is called one-step prediction. We will discuss this in more detail in the following chapters.

When the behaviour of a dynamic system is evaluated, it is done through multi-step prediction or simulation, meaning testing the behaviour of the network with the full signal, replacing the unknown output signal values with their predictions. The idea of performing a simulation of a neural network describing a dynamic system is illustrated in Figure 1.22 and can be represented by Equation (1.12)

$$\hat{y}(k+1) = f(\hat{y}(k), \hat{y}(k-1), \ldots, u(k), \ldots). \qquad (1.12)$$

Optimising a neural network involves the supervised learning of the feedforward network. However, the model of a dynamic system illustrated by this neural network is actually a recurrent network because of the feedback connections.

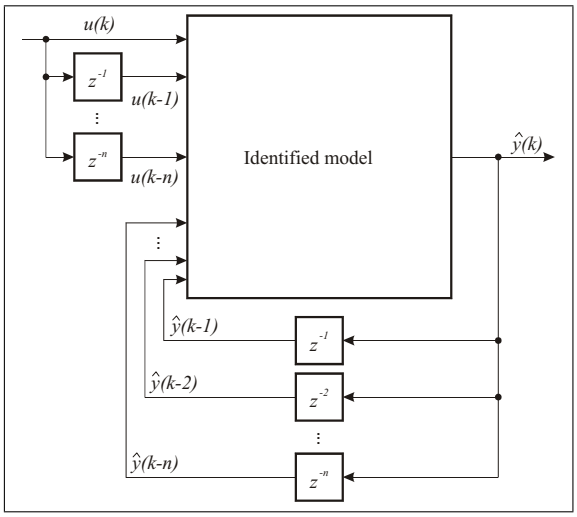

Figure 1.22: Block diagram of the simulation of the identified nonlinear dynamic system

Example of the approximation of a first-order dynamic system by an RBF network

A dynamic system can be described with the following differential equation using the RBF network:

$$y(k+1) = 0.2\tanh(y(k)) + \sin(u(k)). \qquad (1.13)$$

The input and output signals used for learning the neural network are each represented by 2207 samples. The dynamic system is modelled with a neural network containing 20 Gaussian basis functions with $\rho = 0.3$ randomly distributed in a region where the values of the input and output signals take place. You will learn more about how to identify dynamic systems with a neural network in the following chapters. Let us take a look at how the neural network models the nonlinearity of $y(k+1)$ at the inputs $y(k)$ and $u(k)$. The results are shown in Figure 1.23. At first glance, the figure shown on the right in Figure 1.23 appears very poor, but on closer inspection, it turns out to be sufficient for the domain where the neural network learning data was available. The example is very illustrative and shows how important it is to be aware of the limited possibilities of neural networks; it also shows the complexity of identifying dynamic systems with neural networks.

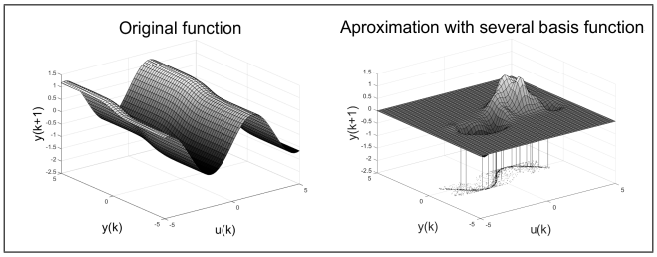

Figure 1.23: The nonlinear dynamic system (left figure) and its RBF network model (right figure) [6]

Bibliography

[1] Artificial Neural Network. http://en.wikipedia.org/wiki/Artificial_neural_network

[2] Y. Bengio, I. Goodfellow,A. Courville (2017). Deep learning Cambridge, MA, USA: MIT press.

[3] A. P. Englebrecht (2002): Computational intelligence, An introduction, Wiley and Sons, Chichester.

[4] I. Grabec, W. Sachse (1997): Synergetics of measurement, prediction and control, Springer Series in Synergetics, Springer-Verlag Berlin and Heidelberg GmbH & Co.

[5] G. Gregorčič (2004): Data-based modelling of nonlinear systems for control, PhD thesis, University College Cork, National University of Ireland, Cork.

[6] S. Haykin (1994): Neural networks: a comprehensive foundation, Macmillian Publishing Company, Inc.

[7] D. O. Hebb (1949): The organization of behaviour, Wiley, New York.

[8] K. Hornik (1989): Multilayer feedforward networks are universal approximators, Neural Networks, Vol. 2, 359-366.

[9] I. Kononenko, M. Kukar (2007): Machine learning and data mining: introduction to principles and algorithms, Horwood Publishing, Chichester.

[10] B. J. A. Kröse and P. P. van der Smagt (1994): An introduction to neural networks, Seventh edition, The University of Amsterdam.

[11] D. J. Mackay (2003): Information theory, inference and learning algorithms, Cambridge University Press.

[12] W. S. McCulloch, W. Pitts (1943): A logical calculus of the ideas immanent in nervous activity, Bulletin of Mathematical Biophysics, 5, 115-133.

[13] M. Minsky, S. Papert (1969): Perceptrons: an introduction to computational geometry, The MIT press.

[14] K. S. Narendra, K. Parthasarathy (1990): Identification and control of dynamical systems using neural networks, IEEE Transactions on Neural Networks, Vol. 1, No. 1, 4-27.

[15] Matlab (1993): Matlab. Mathworks Inc.

[16] G. W. Ng (1997): Application of neural networks to adaptive control of nonlinear systems, UMIST Control Systems Centre Series, Vol. 4, Research Studies Press, Manchester.

[17] M. Norgaard, O. Ravn, N.L.L. Poulsen(2002): NNSYSID-toolbox for system identification with neural networks. Mathematical and computer modelling of dynamical systems, Vol. 8, No.1, 1-20.

[18] J. Mikleš, M. Fikar (2007): Process modelling, identification, and control. Springer.

[19] J. Park , I. W. Sandberg (1991): Universal approximation using radial-basis-function networks, Neural Computation, Vol. 3, No. 2, 246-257.

[20] D. Psaltis, A. Sideris, A. A. Yamamura (1988): A multilayer neural network controller, IEEE Control Systems Magazine, Vol. 8, No. 2, 17-21.

[21] F. Rosenblatt (1959): Principles of neurodynamics, Spartan Books, New York.

[22] D. E. Rumelhart, J. L. McClelland (1986): Parallel distributed processing: explorations in the microstructure of cognition, The MIT press.

[23] J. Sjöberg et al. (1995): Non-linear black-box modeling in system identification: a unified overview, Automatica, Vol. 31, No. 12, 1691-1724.

[24] B. Widrow, M. E. Hoff (1960): Adaptive switching circuits. In 1960 IRE WESCON Convention Record, 96-104, DUNNO.

[25] B. Widrow, F. W. Smith (1963): Pattern recognizing control systems, computer and information sciences (COINS) Symposium Proceedings, Spartan Books, 351-357.

Chapter 2

Identification of linear dynamic systems

2.1 Summary on linear system identification

The purpose of this chapter is to summarise the basics of linear system identification in order to continue with nonlinear system identification in the following chapter. Details on the theory and practice of linear systems identification can be found, for example, in [2], [3], [4], among others.

System identification is an experimental determination of the temporal behaviour of a system based on measured signals. The temporal behaviour is defined within a class of mathematical models and represents the identified process in such a way that the differences between the system and its mathematical model are as small as possible.

The main task in system identification, which is a form of experimental system analysis, is to find an appropriate model structure that determines the class of models in which the model is sought. Estimating the parameters of this structure is the next step. A basic rule in system identification is not to identify parts of the system that are already known. Prior knowledge about the system, whether it is physical knowledge or knowledge gained from experiments, must be used. One of the classifications of mathematical models by colour code is as follows:

- 'white-box' model, the theoretical model obtained from the physical, chemical, etc. insight into the processes in a system;
- 'grey-box' model, in which there is some prior knowledge, which takes the form of
  - the full or partial structure of the model, while the parameters have to be estimated from measurements;
  - the knowledge of the combinations of measured signals or their relationships that can be used to support the modelling;
- 'black-box' model, by which the structure can be identified and the parameters estimated from the measured data, as there is no prior knowledge of the process. The system model is identified only from the input and output signals of the system (Figure 2.1).

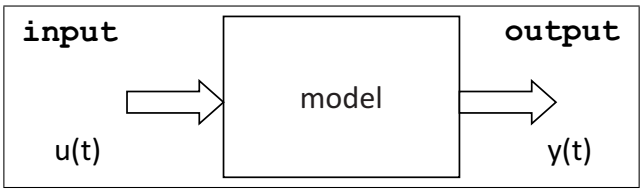

Figure 2.1: A schematic representation of a dynamic system with input and output signals

The theory and practice of identifying dynamic systems is a combination of the results of scientific knowledge and engineering practice. In addition to the technical literature, there is also a considerable amount of identification software (e.g., [5]).

System identification methods can be classified in many ways. Some of the classifications according to different aspects include:

- Class of mathematical models:
  - nonparametric models,
  - parametric models.
- Class of signals used:
  - continuous-time, discrete-time,
  - deterministic, random, pseudo-random.
- Error between process and model:
  - output error,
  - input error,
  - equation error or error-in-variables.
- Simultaneity of measurement and evaluation:
  - offline,
  - online.
- Data processing:
  - nonrecursive,

    - * direct,
    - * iterative,
  - recursive.
- Structure of the models:
  - linear models,
  - nonlinear models.

The complete procedure for system identification is described, in the literature (e.g., [2], [4]). The basic steps of system identification can be summarised as follows:

1. identification of the purpose of the model,
2. collecting prior knowledge,
3. design of experiments,
4. carrying out experiments,
5. identification of the model, and
6. evaluation of the identified model.

The whole process is highly iterative and interactive. The steps in which the purpose of the model, the data measurements, and the model evaluation are verified are particularly important for the practice of engineering identification.

When we experimentally model (identify) a system, the complexity of the problem strongly depends on whether we approach it as a static (Equation (2.1)) or dynamic (Equations (2.2) and (2.3)) system. The dynamics of the system increases the complexity of the problem.

- Static system:

$$F[u(t), y(t)] = 0, \tag{2.1}$$

  where $u(t)$ is input signal and $y(t)$ is output signal.

- Dynamic system [3]:

$$\begin{aligned} &F[t, u(t), \dot{u}(t), \ddot{u}(t), \ldots, u^{(m)}(t), \\ &y(t), \dot{y}(t), \ddot{y}(t), \ldots, y^{(n)}(t)] = 0, \end{aligned} \tag{2.2}$$

  where $\dot{u}(t), \ddot{u}(t), u^{(m)}(t)$ are the first, the second, and the m-th derivatives, respectively, of the continuous-time input signal $u(t)$ and $\dot{y}(t), \ddot{y}(t), y^{(n)}(t)$ are the first, the second and the n-th derivatives of the continuous-time output signal $y(t)$, respectively.

$$\begin{aligned} &F[k, u(k), u(k-1), u(k-2), \ldots, u(k-m), \\ &y(k), y(k-1), y(k-2), \ldots, y(k-n)] = 0, \end{aligned}$$

  where $u(k-i); i = 1 \ldots m$ are delayed samples of the discrete-time input signal u, and $y(k-i); i = 1 \ldots n$ are delayed samples of the discrete-time output signal y.

System identification methods are often classified according to model classes:

- nonparametric and
- parametric models.

Nonparametric models are primarily models of linear systems. They describe the input/output behaviour of a process in the form of value tables or curves. Typical representatives are:

- frequency responses (Bode diagrams),
- weighting functions or impulse responses, step responses,
- Fourier analysis, frequency-response analysis, correlation analysis, spectral analysis.

Parametric models are used to model both linear and nonlinear dynamic systems. These are models with explicitly expressed parameters. Most often, they are models expressed in the form of:

- differential equations,
- difference equations,
- transfer functions,
- state-space equations.

In identifying a linear model of a system, the order of the linear model must be known. This is determined by the order of the equations describing the system and the regressors. Regressors are the quantities on which the prediction of the model depends functionally. The parameters are estimated with optimisation, which uses the sum of the squares of the response errors (the least-squares method) as the cost function, although other criteria are also used, for example, the maximum likelihood of the responses (maximum-likelihood method).

Linear systems can be represented by the following equation in the z-domain [2], [3], [4]:

$$A(z^{-1})y(z) = \frac{B(z^{-1})}{F(z^{-1})}u(z) + \frac{C(z^{-1})}{D(z^{-1})}v(z), \tag{2.3}$$

where A, B, C, D, F are polynomials of the complex variable z. $y(z)$, $u(z)$, $v(z)$ are the transforms of the discrete-time output signal $y(k)$, the input signal $u(k)$ and the noise signal $v(k)$. Depending on which regressors are chosen to describe the system, some of these polynomials are equal to 0 or 1.

The methods for system identification differ depending on the regressors used and/or the optimisation cost function. In the continuation, the least-squares method is adopted for the optimisation:

- finite-impulse-response (FIR) method ($A = F = D = 1, C = 0$), i.e., the regressors are only delayed input signals,
- autoregressive model with exogenous input (ARX) ($F = C = D = 1$), i.e., the regressors are delayed input and output signals,
- output-error method (OE) ($A = C = D = 1$), i.e., the regressors are delayed values of the input signal and estimates of the output signal (predictions from the past),
- autoregressive and moving average model with exogenous input (ARMAX) ($F = D = 1$), where the regressors are delayed input, output and noise signals,
- the Box-Jenkins (BJ) method ($A = 1$), which has delayed input signals as regressors, delayed output-signal estimates, the prediction error, and the simulation error (when the output estimates are used for prediction),
- it is also possible to represent the linear system in a state space [3] where the linear system is written in the form : $\mathbf{x}(k) = \mathbf{A}\mathbf{x}(k-1) + \mathbf{B}\mathbf{u}(k-1)$, where $\mathbf{x}(k) = [x_1(k), \ldots, x_n(k)]^T$ is the vector of state variables, $\mathbf{u}$ is a vector of input variables and $\mathbf{A}, \mathbf{B}$ are matrices of constant elements,
- other possible regressors.

In addition to the methods described above, there are a large number of variants. The methods described differ mainly in the way they incorporate noise into the model. Noise is unavoidable in real systems. Let us look at some selected models of the second-order linear dynamic systems (Figures 2.2, 2.3 and 2.4):

- autoregressive model with exogenous input (ARX)

$$\begin{aligned} y(k) &= a_1 y(k-1) + a_2 y(k-2) \\ &+ b_1 u(k-1) + b_2 u(k-2) \\ &+ v(k); \end{aligned} \tag{2.4}$$

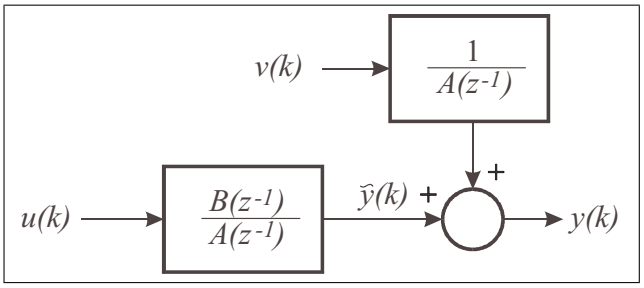

Figure 2.2: Autoregressive model with exogenous input (ARX)

- autoregressive and moving average model with exogenous input (ARMAX)

$$\begin{aligned} y(k) &= a_1 y(k-1) + a_2 y(k-2) \\ &+ b_1 u(k-1) + b_2 u(k-2) \\ &+ v(k) + c_1 v(k-1) + c_2 v(k-2); \end{aligned}$$

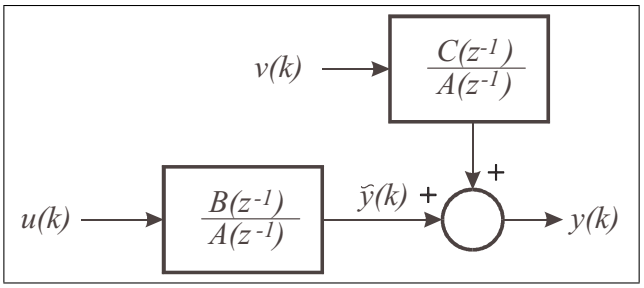

Figure 2.3: Autoregressive and moving average model with exogenous input (ARMAX)

- Output error model (OE)

$$\begin{aligned} y(k) &= a_1[y(k-1) - v(k-1)] \\ &+ a_2[y(k-2) - v(k-2)] \\ &+ b_1 u(k-1) + b_2 u(k-2) \\ &+ v(k). \end{aligned} \tag{2.5}$$

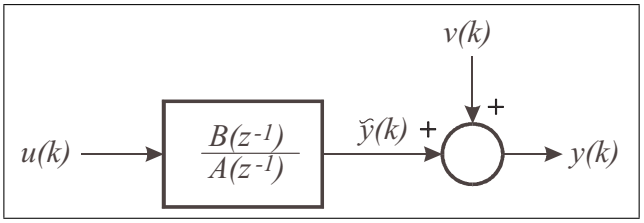

Figure 2.4: Output error model (OE)

Each of the methods also has its own recursive versions for online identification [2], [3], [4].

Example of the ARX model identification

Let us look at a simple example of the ARX model identification of the first-order system. The system we want to identify is described by the difference equation:

$$y(k) = 0.9512y(k-1) + 0.09754u(k-1) \tag{2.6}$$

or the transfer function in the z-domain

$$H(z) = \frac{0.09754z^{-1}}{1 - 0.9512z^{-1}} \tag{2.7}$$

and is not disturbed by noise.

It is a first-order system and the prediction depends on a delayed value of the output and a delayed value of the input. Thus, the two regressors are: $y(k-1)$, $u(k-1)$. The structure of the model is given by the one-step-ahead prediction equation, meaning that the prediction of values based on known values of the regressors in the previous time step:

$$y(k) = -a_1 y(k-1) + b_1 u(k-1). \tag{2.8}$$

If we insert the sampled measured-signal values, we can write Equation (2.8) in matrix form:

$$\begin{matrix} \mathbf{y} & = & \boldsymbol{\Psi\theta} \end{matrix}$$
$$\begin{bmatrix} y(2) \\ y(3) \\ \vdots \end{bmatrix} = \begin{bmatrix} -y(1) & u(1) \\ -y(2) & u(2) \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \end{bmatrix} \tag{2.9}$$

It should be noted that the order of the rows can also be permuted, as the best prediction is optimised based on the individual values of the signal and not the whole signal.

The given system of equations is solved by the method of least squares with the following analytical solution:

$$\hat{\boldsymbol{\theta}} = [\boldsymbol{\Psi}^T\boldsymbol{\Psi}]^{-1}\boldsymbol{\Psi}^T\mathbf{y}. \tag{2.10}$$

The parameters have been optimised for one-step-ahead prediction, as this is a property of model-based methods with equation errors, but they must be validated with a simulation (multi-step-ahead prediction) to see if the model satisfactorily represents the dynamics of the system.

Figures 2.5 and 2.6 present the data used and the result of the parameter estimation.

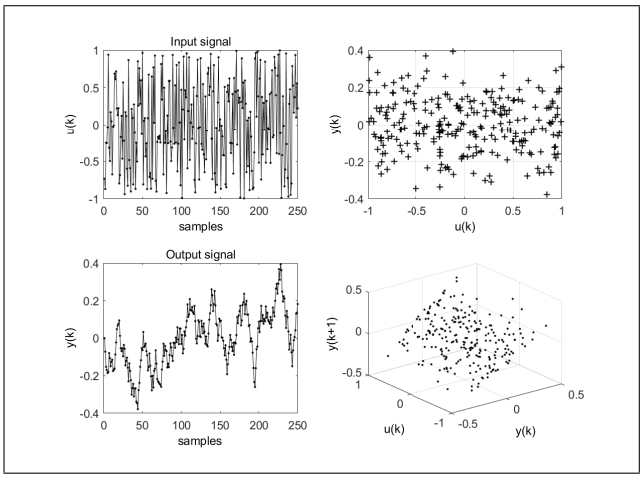

Figure 2.5: Input and output signal (left images), covering the area with the measured data (right images)

In this example, we have focused solely on the system identification method. In the following example, we will show the whole identification process.

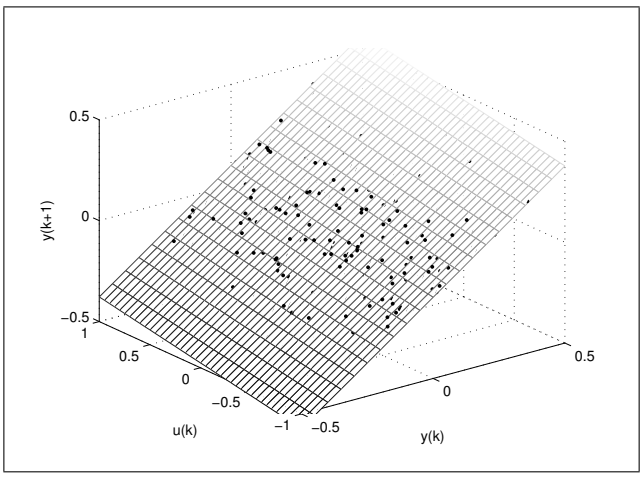

Figure 2.6: The plane showing the identified model and identifying data

2.2 Mechatronic-system identification example

The purpose of this example is to illustrate the complete identification procedure for a linear dynamic system in practice and to highlight the problems encountered in this procedure.

Description of the mechatronic system

The dynamic process to be identified is an electromechanical motor-generator laboratory assembly manufactured by ELWE [1]. It is a device designed for experimental laboratory work and training in control design. A schematic representation of the device together with the data acquisition equipment can be found in Figure 2.7.

A series-connected 100 W DC motor is mechanically coupled to a series-connected DC generator. The load of the generator is two 40 W (220 V) light bulbs. We are interested in the model of the system that has the voltage at the terminals of the motor as system input u and the voltage of the speed transducer as system output y. In order to control the system by a computer, there is a thyristor converter at the input of the system. The speed sensor outputs a voltage of 1 V at 1000 rpm, which is passed through the measuring amplifier of the transducer. The thyristor converter and the amplifier are part of the additional equipment of the motor-generator system from the same manufacturer.

A PCI -20000 converter, manufactured by Burr-Brown, contains, among other things, digital-to-analogue and analogue-to-digital converters required for the acquisition and injection of signals for system identification. The block diagram of the general measurement system for system identification is shown in Figure 2.8. The measured

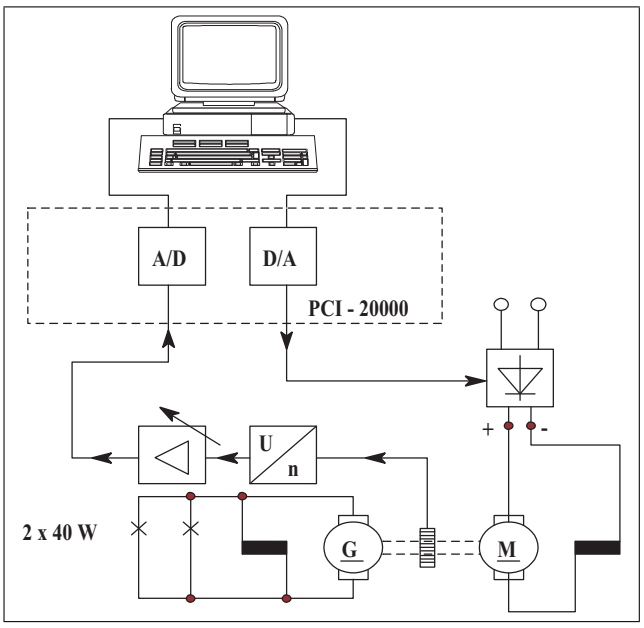

Figure 2.7: Motor-generator laboratory setup

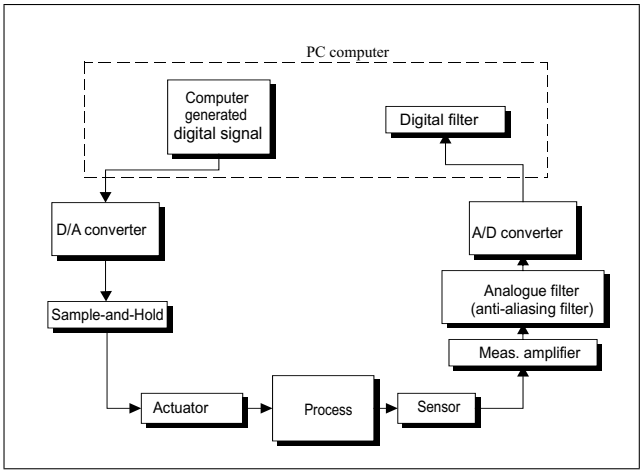

Figure 2.8: Measuring-system schematic

steady-state system characteristics are shown in Figure 2.9.

While the static model describes only the gain between the input and output signals in steady state, the dynamic model describes not only the gain in steady state, but also transient phenomena of the output variables that occur when the input variable changes and causes the output variable to change.

The speed of rotation of the system depends on the electric current through the bulbs, which varies greatly when the bulbs are ignited. This can be observed in the static characteristic curve (Figure 2.9) as a voltage drop. The operating point at which the linearised model is operated is 8.5 V. The steady-state characteristic must be measured in order to linearise the static process, which, as can be seen from Figure 2.9, is not necessary in the present case,

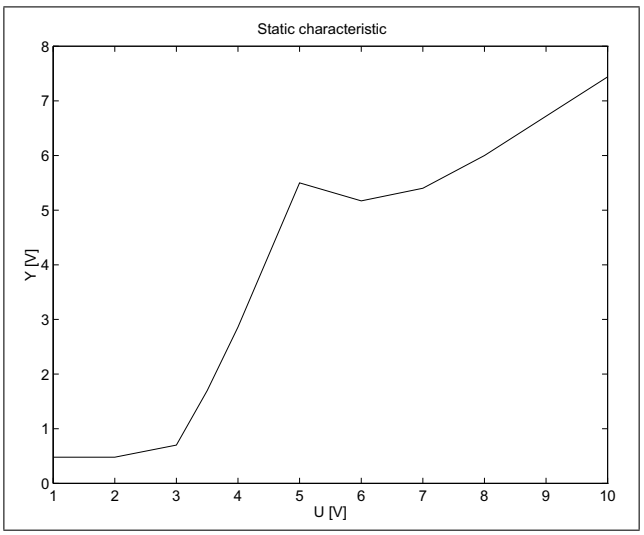

Figure 2.9: Measured static characteristic

as the steady-state characteristic is quite linear around the operating point of interest (8.5 V). To model the system over the entire operating range, it would be necessary to create a nonlinear model or settle for a linear model of low accuracy. Note that in reality linear systems do not exist or are extremely rare. To identify an unstable system, it would have to be done it in a slightly different way [4].

Sampling time, input signal, preprocessing

Input signal selection

Different identification methods require different input signals. Common to all methods is that the input signals must sufficiently excite the system. This means that the input signals must contain a sufficient number of frequency components in the range of interest. This can be achieved by a large number of different experiments or by the variability of a single input signal. A good model is obtained only in the part of the frequency range that is excited by the input signal. There are many suitable and less suitable input signals. It is not sufficient to use only one form of the signal. An input signal must be constructed from signals that each excite a different frequency range. Note that the choice of input signal is of paramount importance to the identification process. Usually, the response of the process to several input signals is recorded and then the most appropriate combination of input signal and response is selected.

In our example, we have chosen a pseudo-random binary signal (PRBS) as the input signal. This signal is an approximation of white noise.

White noise [2] is a signal with statistically independent values whose power spectral density is equal to a constant. It is therefore quite suitable for system excitation because it excites over the entire frequency range. However, it is practically infeasible because its mean power is infinite [2].

The white-noise approximation with PRBS is easy to construct and has a limited amplitude at high power density. The PRBS has amplitude $+a$ and $-a$. Sign changes occur only in discrete time instants $k\lambda, k = 1, 2, \ldots$, where λ is the length of the time interval, also called duty cycle. The position of the zeros of the spectrum depends on the duty cycle λ. Therefore, λ is determined by the frequency range of interest for system identification (i.e., the estimated system dynamics). The frequency range of interest is iteratively identified until it is determined that the input signal excites the entire frequency range of interest.

The choice of the amplitude a of the input signal should be a compromise between the largest possible value due to the signal-to-noise ratio on the one hand and the safety and economy of operation and nonlinearity of the system on the other. In the present example, nonlinearity is a highly influential factor, as can be seen from the static characteristics (Figure 2.9). We have chosen the amplitude $a = \pm 1$ V around the operating current (8.5 V) because we want to remain in the range that is as linear as possible.

If the choice of the input signal is restricted by various factors (e.g., limited experimental possibilities), we can at least determine in which frequency range the given input signal excites the system and consequently in which frequency range our model is valid.

To validate the model, we have chosen a sequence of rectangular pulses with a duration of 1 second and an amplitude of ± 1, which satisfactorily excites the process in a somewhat narrower frequency range.

Choice of sampling time

The rule of thumb for choosing the sampling time [2] is that it should be 10 % of the settling time of the system in response to a step signal. The step response with an amplitude of 1 V in the selected linear range is shown in Figure 2.10. Using the figure, we can determine the settling time, which in our example is ≈ 0.2 s, and from here we choose the sampling time $T_s = 0.02$ s. The sampling time determines the frequency range in which the signal can be observed (Shannon's theorem):

$$\begin{aligned} f_{max} &= \frac{1}{2T_s} = 25 \text{ Hz} \\ \omega_{max} &= 2\pi f_{max} = 157 \text{ rad/s}. \end{aligned} \tag{2.11}$$

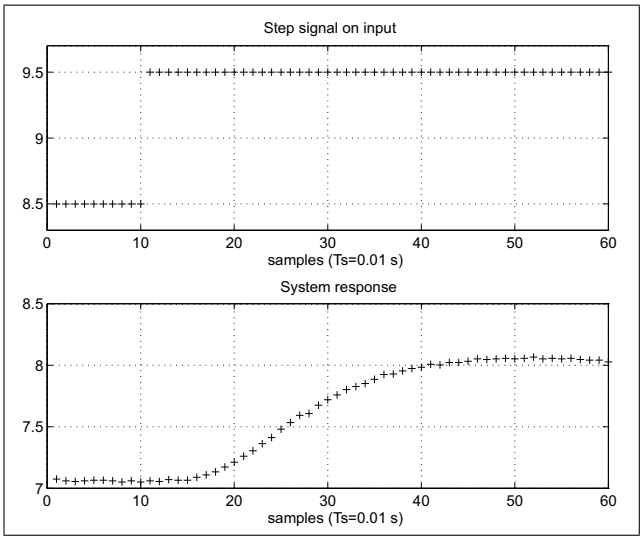

Figure 2.10: Response to step change

The PRBS input signal was generated with a digital register of seven bits and, since we chose a duty cycle of $\lambda = 5T_s = 0.1$ s, we obtain 635 samples. The total measurement time was therefore 12.7 s. The first zero of the power spectrum of the input signal was at $\omega = 62.8$ rad/s. The most useful subrange is up to the frequency at which the magnitude of the power spectrum drops to half of its low frequency value, which in our case is at 31.4 rad/s, because only in this frequency subrange is the system to be modelled satisfactorily excited.

The input signal and its magnitude spectrum density (i.e., the magnitude spectrum of the aperiodic signal) is shown in Figure 2.11. The magnitude density spectrum of the

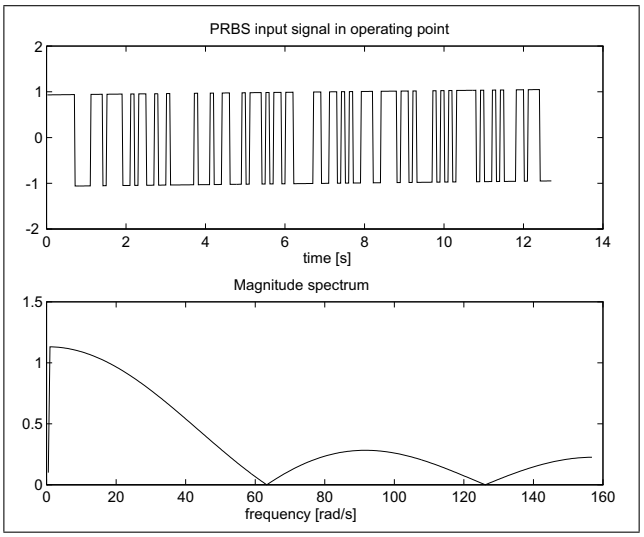

Figure 2.11: Input signal and its magnitude spectrum density

input signal does not contain the component at 0 rad/s in Figure 2.11 because the DC component ($\omega = 0$) has been eliminated. This will be discussed later. We did not choose just one input signal. The signal described was chosen from ten signals with similar properties (PRBS signals with different lengths and duty cycles, (i.e., with different magnitude spectrum densities)) for which we made measurements and selected the most appropriate one based on model validation. The number of samples should be large, because the larger the number of samples, the better the result of the system identification. An offset voltage was added to the input signal generated with a computer so that its average value corresponds to an operating current of 8.5 V. The response of the system at the operating point

(i.e., without the DC component) is shown in Figure 2.12.

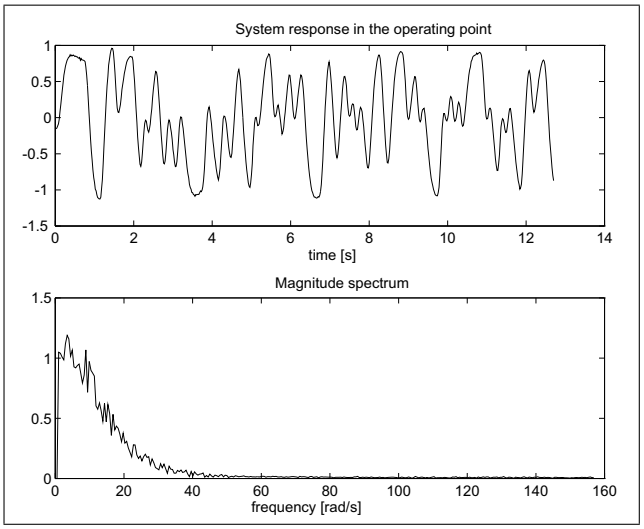

Figure 2.12: Response and its magnitude spectrum density

An analogue filter can be implemented upstream of the analogue-to-digital converter (Figure 2.8) and prevent frequency aliasing [2]. In the present example, this was done using a high-quality measurement amplifier that has a frequency characteristic of a low-pass filter. Due to the large bandwidth of the amplifier, the aliasing effect was not completely eliminated. However, this residual effect cannot be separated from the disturbance noise.

Before the identification process, the recorded signals should be processed accordingly. We need to remove the DC component from the signals that is due to operation at the operating point. It leads to a jump in the amplitude density spectra in Figures 2.11 and 2.12. Also, if necessary, the effects of the noise must be reduced, especially in the estimation of the parameters (this will be discussed later), to avoid biased results. This can be done by filtering, but care must be taken with the properties of the chosen system identification method. We will therefore turn to the question of filtering once we have selected the identification method.

We can identify nonparametric models for which we do not need to make any assumptions about the structure of the model in advance other than linearity, or we can identify parametric models where we need to determine the structure. In system identification, these modelling methods complement each other and, as we will see in our example, we can derive the information for structure identification from one model to another. Nonparametric models can also be used to evaluate parametric models.

Model-structure selection

The choice of model structure is very important in system identification because parameter estimation methods can only identify parameter values. The choice of the model of the system depends on the purpose of the model. In our example, we want a linear parametric model that will be used for system simulation as accurately as possible around the operating point. If the model has a large order, it describes the dynamics better, but it is also more complex and has poles and zeros that converge.

We have chosen from among the methods described in [2], which have already been mentioned here. In identifying linear systems, we have to decide on the order of the model. A first estimate of the model order can be obtained by analysing the information matrix or by identifying a nonparametric model: the frequency response.

The information matrix [2] can be used as follows. The value of the determinant $\det(\frac{[\mathbf{\Psi}^T\mathbf{\Psi}]^{-1}}{N})$, where $\mathbf{\Psi}^T\mathbf{\Psi}$ is the information matrix, $\mathbf{\Psi}$ is the matrix of regression vectors and N is the number of regression vectors in the matrix $\mathbf{\Psi}$. The determinant of the information matrix makes a value drop in the order of the information matrix with respect to the order of the model, which already describes the system sufficiently well. The values of the determinant of the information matrix in our example were: for the first-order model 0.3194, for the second-order model 0.0012, for the third-order model $4.6501 \cdot 10^{-7}$, for the fourth-order model $1.4037 \cdot 10^{-10}$ and so on. The greatest loss of value is between the second and third-order models. The identified system is therefore of the second order.

The second way to determine the order of the model is to use a nonparametric system identification [2]. Since we have chosen an aperiodic input signal, we can determine the frequency response with several methods: Fourier analysis, correlation analysis, or spectral analysis. All methods should generally give the same results. All require an aperiodic input signal, while correlation analysis requires white noise as an input signal. As mentioned earlier, PRBS is an approximation of white noise.

Fourier analysis is an otherwise straightforward method, but it is very sensitive to noise. Fourier transforms are calculated using a discrete or fast Fourier transform.

The frequency response can also be calculated using spectral analysis. Like Fourier analysis, spectral analysis gives very poor results in the presence of noise. The variance of the correlation function estimates at large shifts in τ is large. The solution to both problems is smoothing, for example, with the Hamming window [2].

Smoothing is a filtering method with variable weights (i.e., frequency or time window) that can be used to remove noise. We have to be very careful in choosing the width of the smoothing window, as it has to be narrow compared to the measurement time and wide enough to preserve the essential information. The details of the smoothing can be found in [2]. Figure 2.13 shows the magnitude and phase frequency response in the frequency range of interest without smoothing, obtained by Fourier analysis. The

response would be the same if the spectral analysis were performed without smoothing. Also, Figure 2.13 shows both responses of the spectral analysis: without smoothing and with smoothing using the Hamming window of 35 samples.

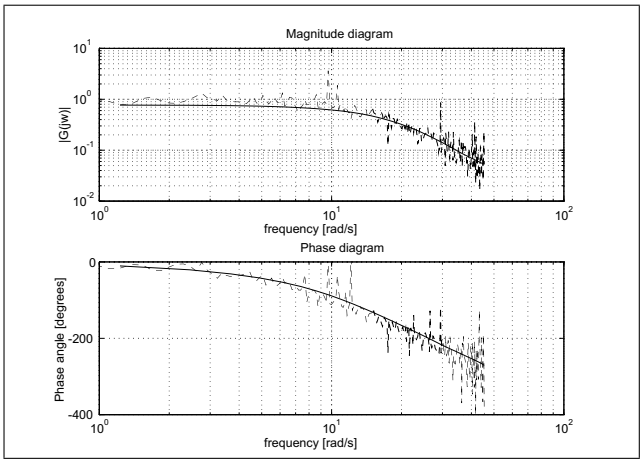

Figure 2.13: Magnitude and phase response without smoothing (dashed curve) and with smoothing (full curve)

Note that smoothing can blur peaks in the frequency response. Nevertheless, the resulting frequency response is satisfactory for obtaining an initial estimate of the order of the system. Using the frequency response shown in Figure 2.13, we used the slope of the magnitude response and the phase response to confirm that the system can be modelled as a second-order system. The slope of the phase response can also be used to estimate the dead time.

When determining the order of the system model, we must be aware that it is an iterative process and that a final decision can only be made in the validation phase. The order of the system determined thus far can only be used as an initial value.

The selection of dead time, except when dealing with a first-order system, can be relatively tricky. A practical approach to this problem is to estimate the parameters at different dead times and take the result where the model fits the data best. In our example, we arrived at a dead time of 0.04 s using the procedure described above. The resulting dead time was checked against the phase response.

The selection of the parameter-estimation method depends not only on the system to be identified, but also on the form of the disturbance or noise. We always have to identify several models. With the validation procedure, we test the models and decide which model fits best. It is very difficult to determine in advance which of the methods should be used for parameter estimation. We usually test several methods and decide again after validating the models. We have chosen one of the methods described in the relevant literature [2, 4].

In our case, we chose the simplest model (i.e., the nonrecursive ARX model), which was optimised using the least-squares method. This method gives biased results, apart from a special form of noise filter. The bias can only be reduced by a suitable filtering, which is found iteratively. Such methods are similar to other methods for parameter estimation in terms of bias (generalised least squares, maximum likelihood).

The least-squares method is essentially about calculating the solution to a given system of equations. When estimating the parameters of an unknown system disturbed by noise, we minimise the square of the difference between the output of the system to be identified and the model.

The equation of the perturbed process in the z-domain is

$$y(z) = \frac{B(z^{-1})}{A(z^{-1})} z^{-d} u(z) + n(z), \tag{2.12}$$

where $A(z^{-1})$ is the denominator of the transfer function in the z-domain, $B(z^{-1})$ - the denominator of the transfer function in the z-domain, y - output signal, u - input signal and n - noise,

correspondingly in the time domain

$$\begin{aligned} y(k) &= a_1 y(k-1) - a_2 y(k-2) - \ldots - a_n y(k-n) \\ &+ b_1 u(k-d-1) + \ldots + b_n u(k-d-n) \\ &+ a_1 n(k-1) - a_2 n(k-2) - \ldots - a_n n(k-n). \end{aligned} \tag{2.13}$$

The vector notation of Equation (2.13) is

$$y(k) = \boldsymbol{\psi}^T(k)\boldsymbol{\theta}, \tag{2.14}$$

where

$$\boldsymbol{\theta} = [a_1, \ldots, a_n, b_1, \ldots, b_n] \tag{2.15}$$

is a vector of the parameters to be estimated. The solution of the least-squares problem, the derivation, and the conditions are described in detail in [2], is

$$\hat{\boldsymbol{\theta}} = [\boldsymbol{\Psi}^T\boldsymbol{\Psi}]^{-1}\boldsymbol{\Psi}^T\mathbf{y}, \tag{2.16}$$

where $\mathbf{y}$ is the output signal vector and $\boldsymbol{\Psi}$ is the matrix consisting of vectors $\boldsymbol{\psi}$ for the corresponding element values of vector $\mathbf{y}$.

Filtering is a method that can be used to obtain results that are as unbiased as possible. The process described by Equation (2.12), can also be written as follows.

$$A(z^{-1})y(z) = B(z^{-1})z^{-d}u(z) + A(z^{-1})n(z). \tag{2.17}$$

The least-squares method gives unbiased results, when $A(z^{-1})n(z) = v(z)$ is a white noise, i.e., if we get $n(z)$ from $v(z)$ via the filter $\frac{1}{A(z^{-1})}$. This condition is often not met in practice, and it was not met in the present example, so noise filtering is required. If we decide to filter, we must filter the input and output signals with the same filter.

We divide Equation (2.17) by the filter $F(z^{-1})$ and obtain

$$A(z^{-1})\frac{y(z)}{F(z^{-1})} = B(z^{-1})z^{-d}\frac{u(z)}{F(z^{-1})} + \frac{A(z^{-1})}{F(z^{-1})}n(z). \tag{2.18}$$

Unbiased identification results when the error of Equation (2.18) (i.e., its last part) is white noise.

Suppose $n(z)$ is white noise. In this case, we obtain unbiased results when $A(z^{-1}) = F(z^{-1})$. This means that we filter signals with an unknown denominator of the transfer function yet to be identified.

In practice, of course, $n(z)$ is not white noise. However, it can approximate white noise filtered by $\frac{1}{A(z^{-1})}$. Therefore, we use a form of filter (e.g., the Butterworth filter), which must be such that the last part of Equation (2.18) is as close as possible to white noise, which is iteratively checked by evaluating the results.

Following the procedure described above, we have identified models of different orders. After validation, the steps of which are described in the following section, we chose a second-order model. This model had two options, which we will consider below in order to select the most suitable one for our purpose.

If we do not filter the input and output signal, the transfer function of the model is

$$\hat{G}_{pn}(z) = \frac{0.0281z^{-1} + 0.0165z^{-2}}{1 - 1.6379z^{-1} + 0.6890z^{-2}}z^{-2}. \tag{2.19}$$

This is a second-order transfer function in the z-domain with stable poles and phase nonminimum zeros and an additional delay of two samples (i.e., 0.04 seconds).

After an iterative procedure in which the input and output signals were filtered with $\frac{1}{A(z^{-1})}$, we obtained a second-order model given by the transfer function

$$\hat{G}_{pf}(z) = \frac{0.0554z^{-1} - 0.00215z^{-2}}{1 - 1.743z^{-1} + 0.7816z^{-2}}z^{-2}. \tag{2.20}$$

Validation is used for model selection. The main elements of validation are described in the following section.

Model validation

The part of the system identification procedure that indicates how good the model is and what needs to be changed (order, method, input signal, sampling time) is the most important part of the procedure. This is the validation of the model, or the evaluation of the model. It is carried out **iteratively**. In validation, we take into account our observations and judgement and do not just rely on the partial results that the computer provides. Which model of the system we choose depends not only on the fulfilment of the requirements, but also on the purpose of the model. To test the validity of the model, we will use several different procedures. We will show the results of the validation for the selected models (2.19) and (2.20).

Prediction-error test

In the least-squares method, the prediction error should be zero. The following tests are available:

- The error signal $e(t)$ should have normal distribution with zero mean. From the left parts of Figures 2.14 and 2.15, we see that the error satisfies this test but does so better for the model with filtered signals.

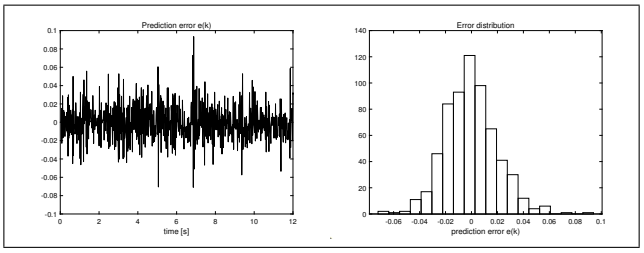

Figure 2.14: Prediction error $e(k)$ between the responses of the transfer function $\hat{G}_{pn}$ and the system used for identification (left image), and the distribution of the error signal $e(k)$ (right figure)

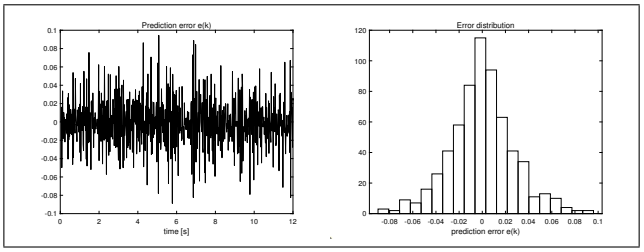

Figure 2.15: Prediction error $e(k)$ between the responses of the transfer function $\hat{G}_{pf}$ and the system used for identification (left image), and the distribution of of the error signal $e(k)$ (right figure)

- The distribution of the error signal $e(k)$ must be normal with zero mean. This can be seen from the right-hand parts of Figures 2.14 and 2.15.

- The error signal $e(k)$ must be independent of the past inputs ($\phi_{eu}(\tau) = 0$) for $\tau < 0$. For open loop operation, the error must be independent of all inputs. Figures 2.16 and 2.17 show the curves of the correlation between the error and the input signal, and we see that their value is small everywhere.

- The autocovariance function of the prediction-error signal $e(k)$ must be a delta impulse. From Figures 2.18 and 2.19, we see that the the autocovariance function has its highest value exactly at shift (or delay) τ=0 and is small at the other shifts. From this, we can deduce a similarity with the delta impulse.

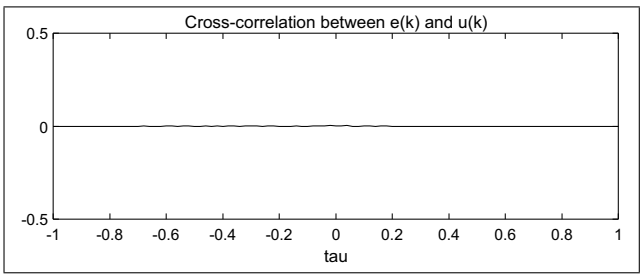

Figure 2.16: Cross-correlation between the error $e(k)$ and the input signal $u(k)$ for transfer function $\hat{G}_{pn}$

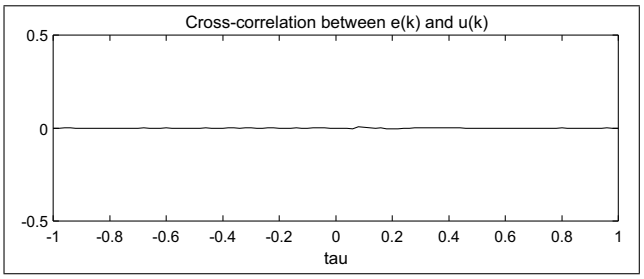

Figure 2.17: Cross-correlation between the error $e(k)$ and the input signal $u(k)$ for transfer function $\hat{G}_{pf}$

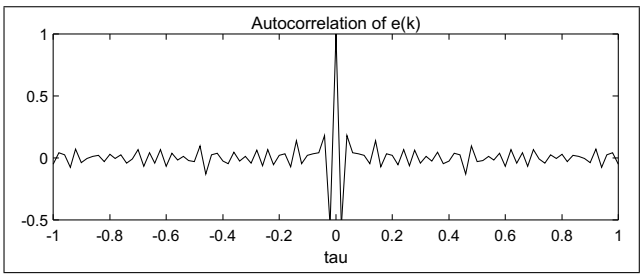

Figure 2.18: Autocorrelation function of the prediction-error signal $e(k)$ for transfer function $\hat{G}_{pn}$

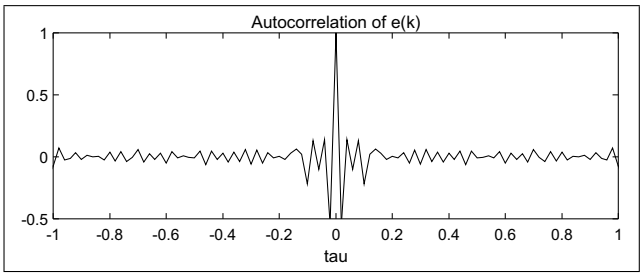

Figure 2.19: Autocorrelation function of the prediction-error signal $e(k)$ for transfer function $\hat{G}_{pf}$

For all the above tests, we can see from the figures that both transfer functions satisfy the conditions, but the transfer function obtained from filtered signals is better.

Consistency of input/output behaviour

The model is validated by testing how it responds to input signals for which the system response is known. There are two possibilities:

- The validation is done with the data we used to identify the model. This is called verification.
- The validation is performed with the data we did not use to identify the model, but which was collected at the same working point. This is called validation or cross-validation. This type of validation tells much more about the validity of the model than validation with data that has been used to identify it.

Figures 2.20 and 2.21 show the input-output response results for both data sets for the two transfer functions selected. Figure 2.20 shows a comparison of the time responses of the two models to the input signal used for identification and the error between the measured response and the simulated response of the model $\hat{G}_{pn}$ and between the measured response and the simulated response of the model $\hat{G}_{pf}$ for the data we used for identification. Figure 2.21 shows a comparison of the time responses of the two models to a sequence of rectangular pulses and error

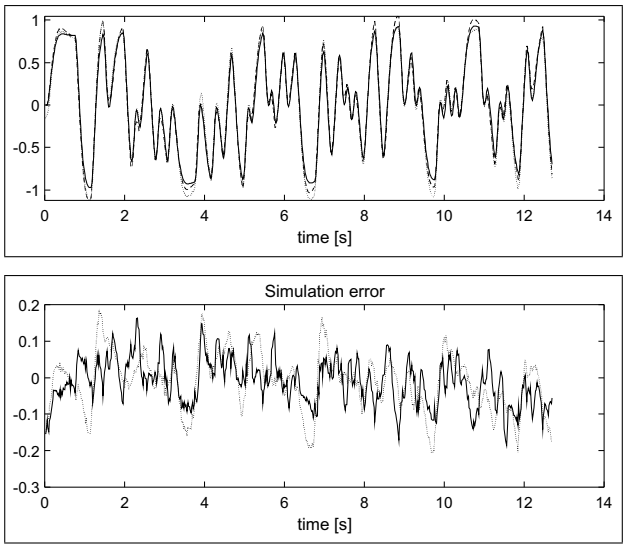

Figure 2.20: Identification signal: upper figure – the measured system response (dotted curve), the model simulation response $\hat{G}_{pn}$ (dashed curve), the model simulation response $\hat{G}_{pf}$ (solid curve); bottom figure – simulation-response error of $\hat{G}_{pn}$ (dotted curve), simulation-response error of $\hat{G}_{pf}$ (dashed curve).

between measurements and the simulated response of the model $\hat{G}_{pn}$ and between the measurements and the simulated response of the model $\hat{G}_{pf}$ for data not used for identification. From the figures, it can be seen that the model errors of $\hat{G}_{pf}$ are on average smaller than for the model $\hat{G}_{pn}$.

From the responses in Figures 2.20 and 2.21, it can be seen that the transfer function identified from the filtered signals better describes the behaviour of the process.

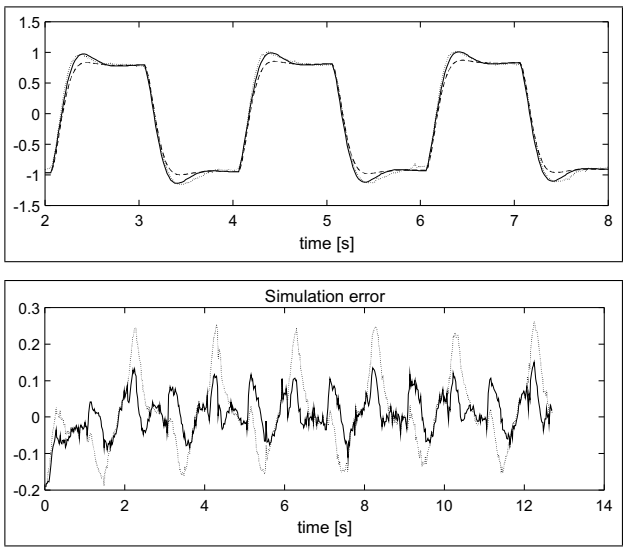

Figure 2.21: Validation signal: upper figure – the measured system response (dotted curve), the model simulation response $\hat{G}_{pn}$ (dashed curve), the model simulation response $\hat{G}_{pf}$ (solid curve); bottom figure – simulation-response error of $\hat{G}_{pn}$ (dotted curve), simulation-response error of $\hat{G}_{pf}$ (dashed curve).

Frequency consistency

The consistency between the frequency response obtained from the Fourier analysis and the frequency response of the transfer function $\hat{G}_{pf}$ is shown in Figure 2.22. The time responses must be accompanied by a fit of the frequency response and vice versa.

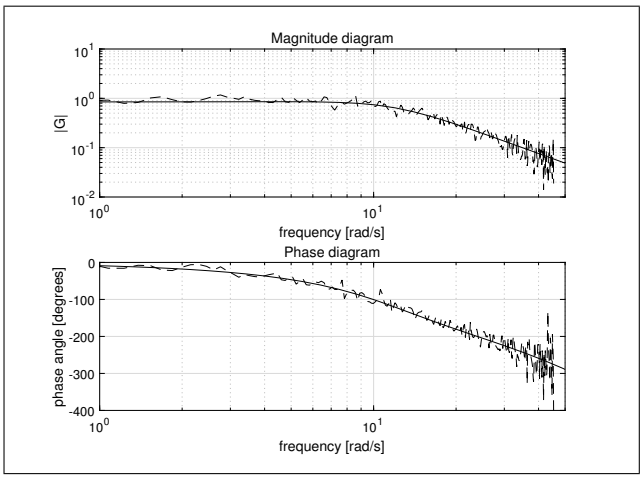

Figure 2.22: Test of the frequency response of the transfer function $\hat{G}_{pf}$ (response $\hat{G}_{pf}$ - solid curve, frequency response for comparison - dashed curve)

In Figure 2.22, it can be seen that the negative magnitude of the phase angle increases with frequency. This is due to the dead time, which is also consistent with a model that has a delay of 0.04 seconds.

Parameter meaningfulness, model reduction, covariance matrix, parameter errors, model utility

The meaningfulness of the discrete-time model of the continuous-time system can be tested by considering the number of poles on the negative real axis in the z- domain. The number of poles must not be odd. In our example, there are no poles on the negative real axis. Furthermore, a model must agree with the identified system in terms of stability.

The standard deviation of the estimated parameters is determined as the square root of the diagonal elements of the covariance matrix of the parameter error [2]. This is a quantitative measure of the confidence in the parameter estimates. The greater the covariance, the greater the deviations, and the lower the reliability and validity of the model.

The covariance matrix is expressed as

$$\mathrm{cov}[\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}] = \sigma_e^2 E\{[\boldsymbol{\Psi}^T \boldsymbol{\Psi}]^{-1}\}, \tag{2.21}$$

where σ_e^2 is the variance of the error of the model [2]. The standard deviations of the individual parameters of the transfer function $\hat{G}_{pn}$ are as follows:

$$\begin{aligned} a1 &= -1.6379; \sigma = 0.0117; \\ a2 &= 0.6890; \sigma = 0.0112; \\ b1 &= 0.0281; \sigma = 0.0013; \\ b2 &= 0.0165; \sigma = 0.0017; \end{aligned}$$

and for the transfer function $\hat{G}_{pf}$, the standard deviations are derived:

$$\begin{aligned} a1 &= -1.7430; \sigma = 0.0040; \\ a2 &= 0.7817; \sigma = 0.0035; \\ b1 &= 0.0555; \sigma = 0.0013; \\ b2 &= 0.0022; \sigma = 0.0017. \end{aligned}$$

It can be observed that the standard deviations of the model parameters obtained from the filtered signals are generally smaller than in the case of the model obtained from unfiltered signals. From all the estimation results shown, it can be concluded that the model obtained from the filtered signals more accurately describes the behaviour of the system we have identified.

We know that we cannot derive the position of the poles in the continuous-time domain from the position of the poles in the discrete-time domain. Zeros at infinity of the s-domain normally transfer to poles on the negative real axis, or at least near the zero coordinate of the z-domain. From all of this, we might conclude that a continuous-time system that we are identifying, assuming that it is of second order, has no finite zeros. This is confirmed by Figure 2.22, which shows that the amplitude response at high frequencies has a slope of 40 dB/decade. The identified second-order model has a finite zero, so its response at

high frequencies does not optimally match the frequency response. If we repeat the identification procedure with a discrete-time model, it has a zero at $z = 0$, we obtain the following model

$$\hat{G}_p(z) = \frac{0.04019z^{-1}}{1 - 1.7040z^{-1} + 0.7494z^{-2}}z^{-2}. \qquad (2.22)$$

The resulting frequency response (Figure 2.23) shows very good agreement. The agreement with the simulation response (Figure 2.24) is not perfect but sufficient for most control implementations.

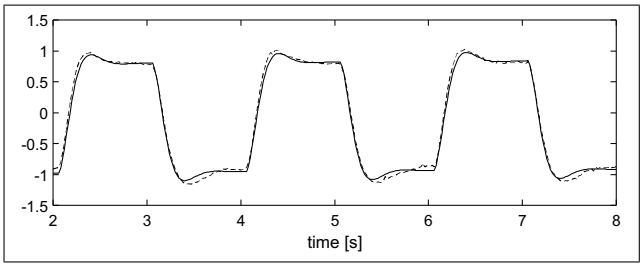

Figure 2.23: Part of the measured system response (dashed curve) and the model simulation response $\hat{G}_p$ (solid curve) to a sequence of rectangular pulses

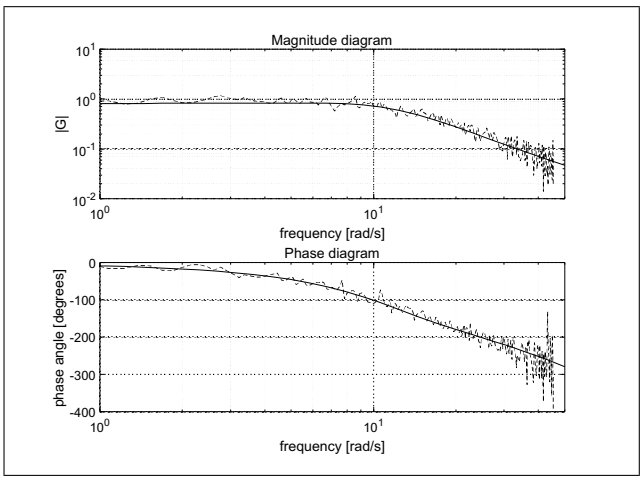

Figure 2.24: The test of frequency behaviour of the $\hat{G}_p$ transfer function (response $\hat{G}_p$ - solid curve, frequency response for comparison - dashed curve)

If we want more accurate information about the model, especially about its behaviour at high frequencies, we would need to perform additional experiments or repeat the identification procedure. The required accuracy of the model primarily depends on the purpose of the model. This assessment depends on the user's judgement. Models can be used for a variety of purposes, such as designing control algorithms, simplifying complex systems, designing simulators, designing fault diagnosis algorithms, and similar. In our example, we have demonstrated the identification process from a practical point of view and obtained a linear model describing the dynamics of the mechatronic system around the operating point. After its validation, the identified model is sufficiently accurate for the design of control systems as well as for the simulation of the dynamics of the mechatronic system, taking into account the standard deviations of the parameters, as well as other purposes. The final validation of the model would be through its application for the design of control algorithms or for use as a simulator.

Bibliography

[1] ELWE (1988): ELWE Educational systems, Experimental manual U9, ELWE, Cremlingen.

[2] R. Isermann, M. Muenchhof (2011): Identification of dynamic systems: an introduction with applications (Vol. 85). Berlin: Springer.

[3] J. Mikleš, M. Fikar (2007): Process modelling, identification, and control. Springer.

[4] L. Ljung (1987): System identification, Theory for the user, Prentice-Hall, Englewood Cliffs, NJ.

[5] L. Ljung (1992): System identification toolbox, user's guide, The MathWorks, Inc., Natick, MA.

[6] J. Sjöberg et al. (1995): Non-linear black-box modeling in system identification: a unified overview, Automatica, Vol. 31, No. 12, 1691-1724.

Chapter 3

Identification of nonlinear dynamic systems

3.1 General information on the identification of nonlinear systems

As mentioned in Chapter 1, researchers in the 1990s came to the realisation that training neural networks to behave like nonlinear dynamic systems was, in fact, nonlinear system identification. With this understanding, it was possible to incorporate and take into account knowledge from system theory and practise [2]. In this chapter, we will look at the basic guidelines for the identification of nonlinear dynamic systems as given by [4].

The main problem in system identification, as mentioned in the previous section, is to find a suitable model structure that can be used to describe the behaviour of a nonlinear system sufficiently well. Estimating the parameter values for the chosen structure is a minor problem in most cases. The basic rule we must always follow is that we do not identify parts of the system that we already know. This means that we have to use the knowledge about the system that we already have.

Model colour coding is also used for nonlinear systems. This coding denotes the background knowledge used for modelling: the model is a white box in theoretical modelling, a black box in system identification, and a grey box if the modelling is one of the various combinations of the two.

Commonly used experimental models of nonlinear dynamic systems are artificial neural networks, fuzzy models or networks of local models, models based on Gaussian process models, wavelet models, and many other models [3].

If the problem of system identification is approached in a standard statistical way, it is also necessary to further harmonise the terminology. Neural networks and other similar methods have also been given their own terminology to be used. Some of the most commonly used terms and their equivalents are as follows.

- Terms (systems theory term = neural network or machine learning term):
  - estimate, identify = learn, train,
  - validate = generalise,
  - model structure = network,
  - estimation data, identification data = learning set, training set,
  - validation data = generalisation set, test set,
  - overfitting = overtraining.

Practical aspects of the identification of nonlinear systems

The identification procedure must not and cannot be fully automated as it involves too many subjective decisions. For a successful procedure, not only knowledge about the identification procedure is necessary but also suitable software (e.g., [6], [7]) and, above all, input and output data that contain sufficient information about the dynamic behaviour of the system to be identified.

The identification of nonlinear dynamic systems is a much more comprehensive problem than the identification of linear dynamic systems. The notion of nonlinearity encompasses an infinite variety of different forms of nonlinearity and experience with one type of nonlinear dynamic system is generally not applicable to other types. It is therefore advisable to gain practical experience of modelling a selected nonlinear process by simulating similar examples with a computer. This is one way to use prior knowledge of the system to be identified.

The problem of identifying nonlinear systems can be described as follows. The individual samples of the output signals of the system to be modelled can be represented as a function of the delayed samples of the input signal u and the output signal y disturbed by noise:

$$y(k) = g(y(k-1), y(k-2), \ldots, u(k-1), u(k-2), \ldots) + n(k), \tag{3.1}$$

where k is the consecutive number of the sample defining a time instant. System identification means finding the function $g(\cdot)$

$$\hat{y}(k|\boldsymbol{\theta}) = g(\boldsymbol{\psi}(k), \boldsymbol{\theta}), \tag{3.2}$$

whereas:

$\boldsymbol{\psi}(k)$= $\boldsymbol{\psi}(u(k-1), u(k-2), \ldots, y(k-1), y(k-2), \ldots)$ the vector of regressors also regression vector,

$\boldsymbol{\theta}$ the vector of parameters, and

$\hat{y}$ the output of the model.

The problem can be divided into two subproblems:

- selection of regressors $\boldsymbol{\psi}(k)$, and
- selection of the nonlinear mapping $g(\boldsymbol{\psi})$.

Regressors

If the structure is linear, the system is described in the following form

$$A(z^{-1})y(z) = \frac{B(z^{-1})}{F(z^{-1})}u(z) + \frac{C(z^{-1})}{D(z^{-1})}v(z), \tag{3.3}$$

which can be written simply as

$$\hat{y}(k) = \boldsymbol{\psi}^T(k)\boldsymbol{\theta}. \tag{3.4}$$

As described in the previous chapter, models of linear systems differ according to the regressors.

Similarly, the models for nonlinear systems are named according to the different regressors $\boldsymbol{\psi}$ used to represent the nonlinearity

$$\hat{y}(k) = g(\boldsymbol{\psi}(k), \boldsymbol{\theta}). \tag{3.5}$$

Regressors $\boldsymbol{\psi}$ in different nonlinear models:

- nonlinear finite impulse response (NFIR) model, which means that the regressors only take delayed input signals ($u(k-i)$);
- nonlinear autoregressive model with exogenous input (NARX), which means that the regressors are delayed input and output signals ($u(k-i), y(k-i)$);
- nonlinear output error model (NOE), which means that the regressors take delayed values of the input signal and estimates of the output signal (the predictions from the past)past) ($u(k-i)$, $\hat{y}(k-i)$);
- nonlinear autoregressive and moving average model with exogenous input (NARMAX), for which the regressors are delayed input, output and noise signals signals ($u(k-i), y(k-i), \varepsilon(k-i) = y(k-i) - \hat{y}(k-i)$);
- nonlinear Box-Jenkins (NBJ) model that has delayed input signals as regressors, delayed output signal estimates, the prediction error and the simulation error (if the output estimates were used for prediction) ($u(k-i), \hat{y}(k-i), \varepsilon(k-i), \varepsilon_u(k-i) = y(k-i) - \hat{y}_u(k-i)$);
- it is also possible to represent the nonlinear system in state space written in the form: $\mathbf{x}(k) = \mathbf{F}(\mathbf{x}(k-1)\mathbf{u}(k-1))$;
- other possible regressors.

Nonlinear mappings

Nonlinear mappings are often represented as a weighted sum of k basis functions $g_k(\boldsymbol{\psi})$:

$$g(\boldsymbol{\psi}(k), \boldsymbol{\theta}) = \sum_k \alpha_k g_k(\boldsymbol{\psi}), \tag{3.6}$$

although, as will be seen later, other forms of representation are possible when modelling with Gaussian processes.

For example, a well-known scalar example of a weighted sum representation is the evolution of a function with a Fourier series. Typical examples of nonlinear mappings that are represented by a weighted sum are

- wavelets,
- nearest neighbours,
- B-splines,
- ARTIFICIAL NEURAL NETWORKS,
  - multilayer perceptron,
  - radial basis-function networks,
  - others,
- FUZZY MODELS.

Let us look at the notations of some well-known basic functions:

- neural network with sigmoidal activation function

  $$g_k(\boldsymbol{\psi}) = \sigma(\beta_k \boldsymbol{\psi} + \gamma_k),$$

  where β is the dilation factor and γ is the translation factor;
- radial basis-function network

  $$g_k(\boldsymbol{\psi}) = r(\beta_k(\boldsymbol{\psi} - \gamma_k));$$

- fuzzy model

  $$g_k(\boldsymbol{\psi}) = \sum_j \alpha_j (\prod_k \mu_A(\beta_k(\boldsymbol{\psi} - \gamma_k)));$$

- recurrent network,

  $$\boldsymbol{\psi}(k) = g(\boldsymbol{\psi}(k-i), \boldsymbol{\theta})$$

- and many others.

Structure identification

First, we start with the systematic selection of regressors, from the simplest possible combination to the more complex ones:

- $u(k)$ - static nonlinearity,
- $u(k-i)$ - NFIR,
- $u(k-i), y(k-i)$ - NARX,
- other.

When determining the order of the model or the number of sample delays, it is useful to know Takens' theorem (its form for excited systems can be found in [9]), which states how a model of a continuous dynamic system can be reconstructed from sampled input and output signals. The sufficient condition for the reconstruction of the system is:

$$n > 2p, \tag{3.7}$$

where n is the order of the discrete model and p is the order of the original continuous system. In practice, it turns out that, depending on the system, a lower order model is often sufficient:

$$p \leq n \leq 2p+1. \tag{3.8}$$

Once we have chosen the model order and the regressors, we start choosing the basis functions. There are different ways of choosing, both objective and subjective. There are no concrete rules because the basis functions we have listed are capable of representing the nonlinearity with any degree of accuracy. Some directions that might be considered are [4]:

- Radial basis functions are chosen when dealing with a small number of regressors (e.g., wavelets for a maximum of 3 regressors),
- ridge basis functions are chosen when we are dealing with a larger number of regressors (e.g., sigmoid neural networks),
- fuzzy models are used when a priori heuristic knowledge is available.

The cross-validation procedure is used to select the different structural elements. Cross-validation is a method from statistics for evaluating models in which the data are divided into at least two parts. The basic form of cross-validation is k-fold cross-validation. We usually use $k-1$ parts for training and the remaining part for validation. The training and validation data must cross-over in successive rounds.

If we choose basis functions and thus a nonlinear model structure, we must also estimate their parameters, which are usually the weights of the individual basis functions and, depending on the structure chosen, some other parameters. This can be done with any known and suitable deterministic or stochastic optimisation method. Among the deterministic methods, the Gauss-Newton algorithms are known to be efficient, while the first-order gradient methods are generally time-consuming. It is possible to estimate the parameters offline or with the so-called recursive identification online. The optimisation criteria usually depend on the model error.

Model error

The optimisation criteria are different. They depend on the purpose of the model and the optimisation method. The cost function used to write the optimisation criterion is expressed as a quantitative measure of the quality of the model. We usually want to achieve the smallest value of the cost function, which is usually a function of the error of the model. A commonly used example is the cost function that depends on the square of the model error:

$$\begin{aligned} \bar{V}(\boldsymbol{\theta}) &= \mathbb{E} \parallel y(k) - g(\boldsymbol{\psi}(k), \boldsymbol{\theta}) \parallel^2 \\ &= \sigma_n^2 + \mathbb{E} \parallel g_0(\boldsymbol{\psi}(k)) - g(\boldsymbol{\psi}(k), \boldsymbol{\theta}) \parallel^2, \end{aligned}$$

where $g_0(\boldsymbol{\psi}(k))$ is the original system and σ_n^2 is the variance of the noise.

When optimising the model with respect to the error square, there are generally three possible sources of error that affect the quality of the model:

- noise,
- bias, which can be represented as follows

  $$V = \mathbb{E} \parallel g_0(\boldsymbol{\psi}(k)) - g(\boldsymbol{\psi}(k), \hat{\boldsymbol{\theta}}(m))) \parallel^2,$$

  where $\hat{\boldsymbol{\theta}}(m)$ is a vector of estimated values is the parameter with the chosen length m,
- is the variance of the parameter values depending on the variance of the measurement noise σ_n^2

  $$V \approx \sigma_n^2 \frac{dim\,\{\boldsymbol{\theta}\}}{N}.$$

The quality of the identified model depends on the information content of the measured input/output data. The variance of the parameter values and the bias are correlated as shown in Figure 3.1.

Tips for carrying out identification

Given the complexity of the problem of identifying nonlinear dynamic systems, it is difficult to suggest a fixed procedure. In general, the procedure follows the same steps as for the identification of linear systems, which were described in the previous section. However, we can give some tips that may increase efficiency.

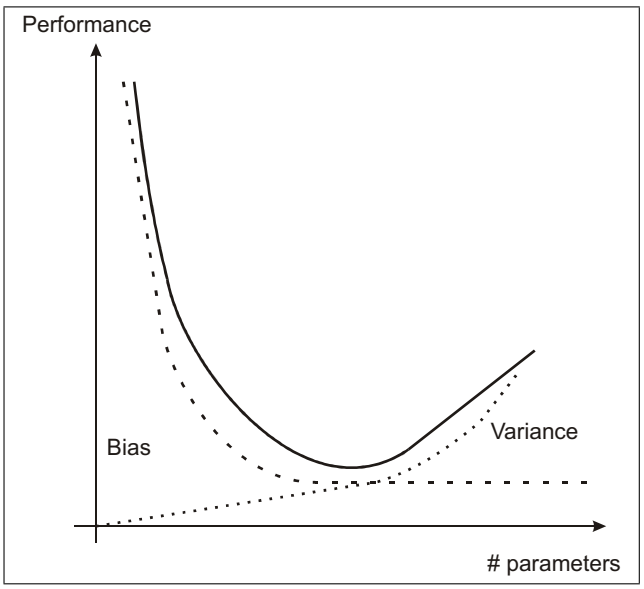

Figure 3.1: Bias-variance relationship

1. Look at the measured input/output data. From this you can deduce whether the system is strongly nonlinear and which 'time constants' (using the technical term for linear systems) it has.

2. Try simple things first, i.e., simple structures and dimensions of models: linear models first and a small number of regressors, basis functions and parameters for the estimation.

3. Determine the physical background of the dynamic process, as this can give you an idea for the choice of regressors.

4. Divide the data into those for identification and those for model validation. In the machine learning literature, the division of data subsets is referred to in different ways, although the purpose is the same. The data for identification are split into data for parameter estimation in nonlinear identification and data for monitoring performance in cross-validation. The parameter estimation data in machine learning are called 'training data' and the performance monitoring data are called 'validation data', while the data used to validate the final model are called 'test data'. It should be noted that this division is also found in the literature describing system identification.

5. Normalise the data (transform the different data types into one size class) and set the mean of the data equal to 0.

6. Monitor the bias/variance ratio throughout the optimisation by cross-validation, indicating an unwise increase in the number of parameters.

7. When modelling a system with a neural network, use regularisation. It is described, for example, in [4].

8. Investigate the efficiency characteristics of the number of parameters, as the model should be as simple as possible but not too simple.

9. When choosing the sampling time, the same rule as for linear systems should apply, even though it is much more difficult to follow: the sampling time should be chosen in such a way that it captures the entire dynamics of the system to be modelled.

10. If you have an input signal to choose from, choose one that has an input/output data distribution and an amplitude distribution that is as rich as possible in the range of operation that is limited by the physical constraints of the process. An example of the distribution is shown in Figures 3.2 and 3.3.

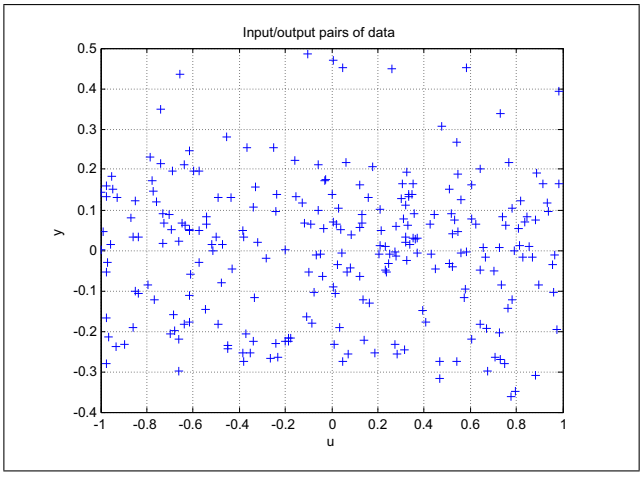

Figure 3.2: Example of input/output data distribution

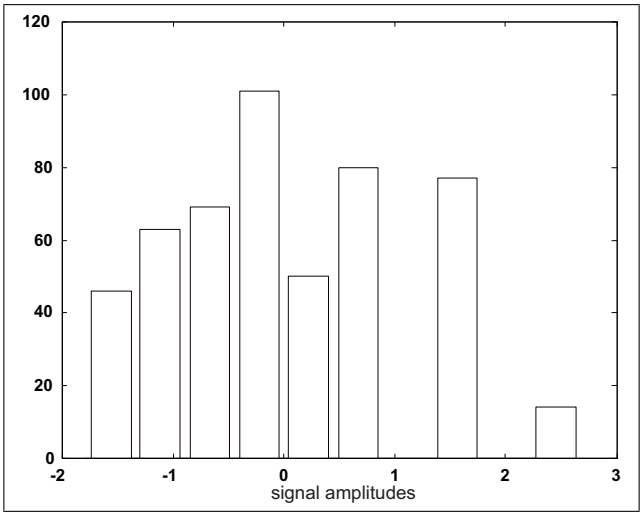

Figure 3.3: Example of the amplitude distribution of the input signal

Model validation

Model verification and validation is, as already stated, one of the most important steps in modelling. The nonlinearity of dynamic systems does not make the validation process any easier.

As seen and described by Equation (3.2), modelling is mainly for one-step prediction. The dynamic model system is most suitable for its purpose when it allows predictions over longer horizons and simulations. A typical example of such a purpose is the use of the model for control design or system performance evaluation. This means that the model has to be evaluated for one-step-ahead prediction, but also for simulation.

- One-step-ahead prediction:

$$\hat{y}(k+1) = g(y(k), y(k-1), \ldots, u(k), u(k-1), \ldots).$$

- Simulation:

$$\hat{y}(k+1) = g(\hat{y}(k), \hat{y}(k-1), \ldots, u(k), u(k-1), \ldots).$$

How the model is simulated is shown in Figure 1.22 in Chapter 1.

The analysis of the consistency of the input/output behaviour with the one-step-ahead prediction and simulation (Figure 3.4) is usually performed on the data used for identification and on the data intended for validation (test data).

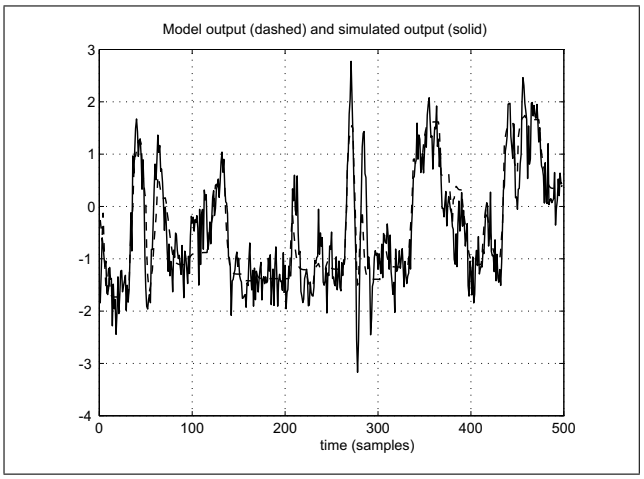

Figure 3.4: Test of consistency of input/output behaviour

In addition to the consistency test of input/output behaviour, various statistical tests can be used, also for nonlinear systems. These include different correlations to test whether the prediction or simulation error is independent of all input and output signals. An example is testing the simulation error with auto- and cross-correlation and

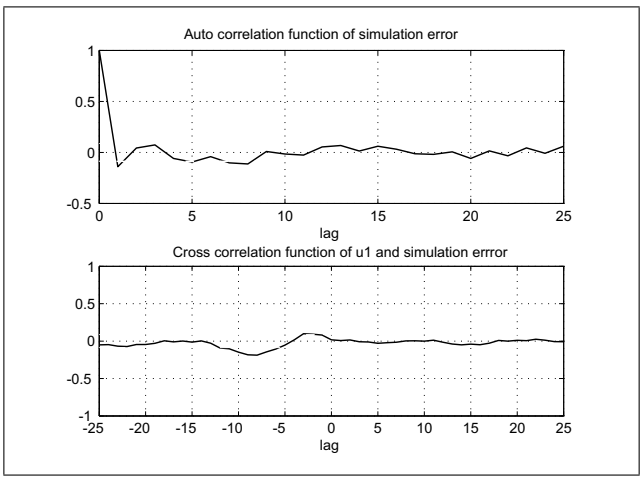

Figure 3.5: Test of prediction error

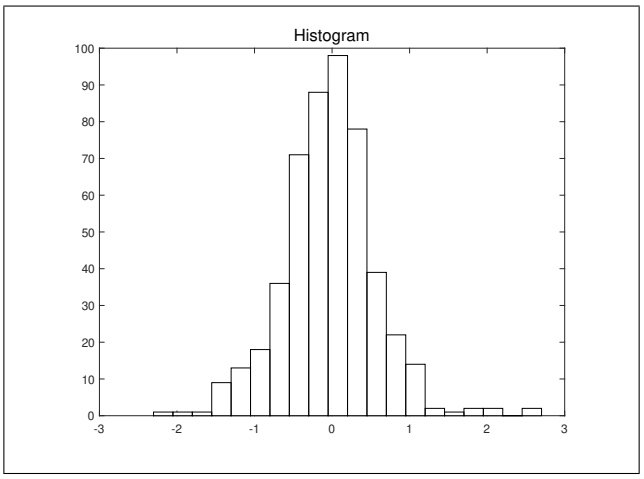

Figure 3.6: Histogram of prediction errors

analysing the histogram of identification and validation error (examples in Figures 3.5 and 3.6).

Other forms of statistical tests are estimates of the average prediction error. These include estimating the error with various criterion functions, such as the mean square or the Akaike criterion function. For more details on estimators with statistical tests, see [5].

Model reduction or pruning [5] is also part of validation. An example of a neural network after pruning can be found in Figure 3.7. The procedures for reduction depend on the structure and the modelling method. More on these procedures can be found in the literature describing the individual procedures (e.g., specific neural networks) and in various software. We will not go into detail about these procedures; more details can be found in, for example, [1].

The most important test of the dynamic model, regardless of its structure and whatever the method of modelling, is always to test the model in terms of its purpose. The

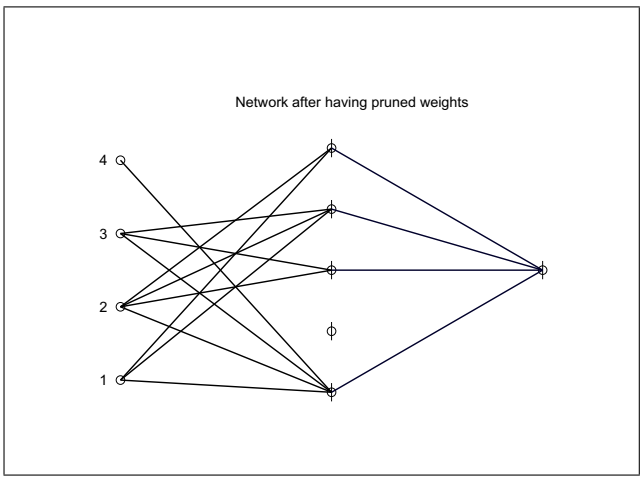

Figure 3.7: Example of a pruned artificial neural network structure

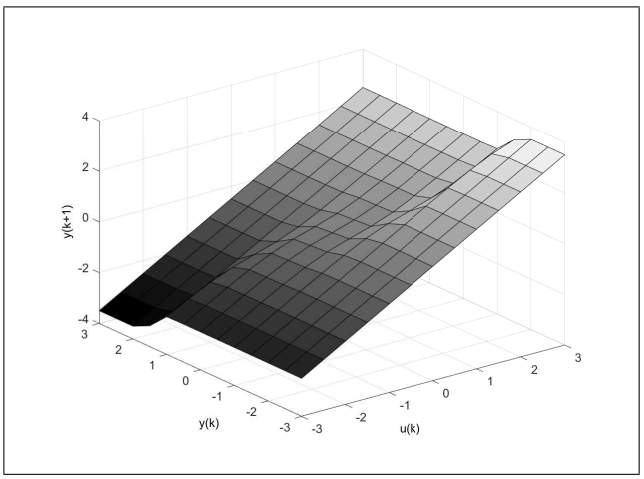

Figure 3.8: 3D-representation of the nonlinearity of the system

model is good enough if it is useful for the purpose for which it was developed.

This was only an overview of modelling. Better insight can be found in the literature that describes neural network identification in more detail (e.g., [5]).

Example of identification of nonlinear autoregressive model with exogenous input (NARX)

We will illustrate the identification process with an example of a first-order dynamic nonlinear system with an artificial neural network. The mathematical model of the process is described by the following nonlinear differential equation:

$$y(k) = y(k-1) - 0.5\tanh(y(k-1) + u^3(k-1)), \quad (3.9)$$

where u is the input and y is the output of the system. This is one of the discrete-time equivalents of the continuous-time system

$$\dot{y} = -\tanh(y + u^3) \quad (3.10)$$

for a sampling time of 0.5 s.

Since this is a first order system, the nonlinearity can be represented graphically in three dimensions (Figure 3.8). The system was excited with a random signal with variable amplitude in the range between -1.2 and +1.2. The identification signal and the response of the system are shown in Figure 3.9. The validation signal must be different from the identification signal so that we can be certain that the model fits. A randomly varying signal in the same amplitude range as the identification signal is chosen. The input signal for validation and the system response are shown in Figure 3.10. It is common for a signal to be divided into the identification part (the training and validation set) and the validation part (the test set). The length of the signals can be arbitrary, but it is useful to have a longer identification signal to contain as much information as possible about the dynamic behaviour of the process.

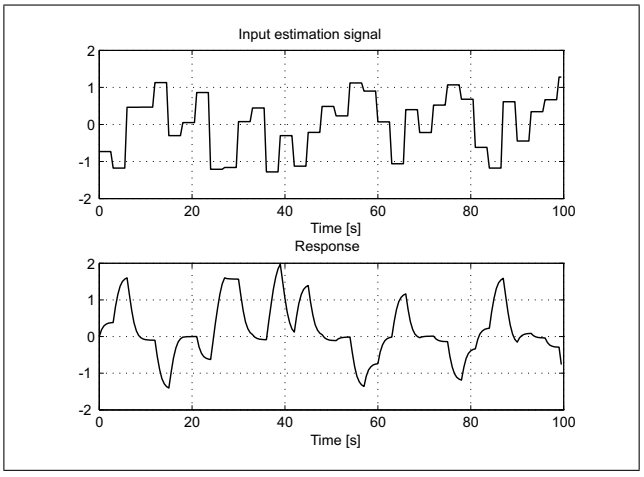

Figure 3.9: Identification signal and response of the system

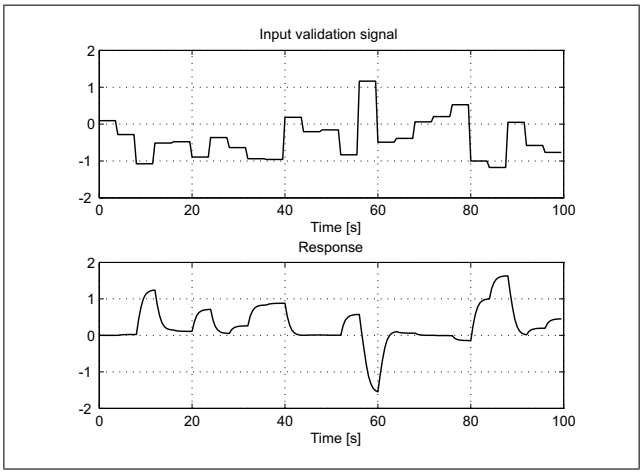

Figure 3.10: Signal for validation and system response

The histograms of the amplitude distributions of the selected input signals are shown in Figure 3.11. It can be seen that the amplitude distribution in the whole selected region is not perfect, but the signal for identification has a better amplitude distribution. How the identification

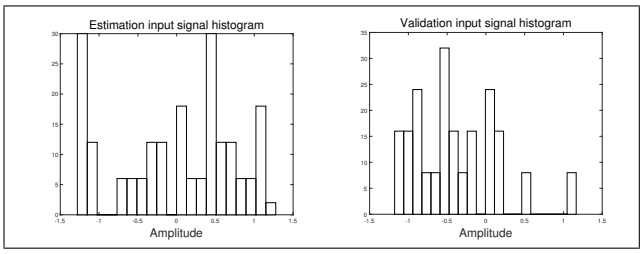

Figure 3.11: Histogram of amplitude distributions of input signals for identification and validation

input/output data are distributed according to the nonlinearity of the process is shown in Figure 3.12. For a denser distribution of samples, more samples are needed, which also means a longer signal. It is very important to check what input/output data are available for identification and validation so that what can be achieved by model identification can be estimated. If there are no data for identification, the model cannot learn the system behaviour.

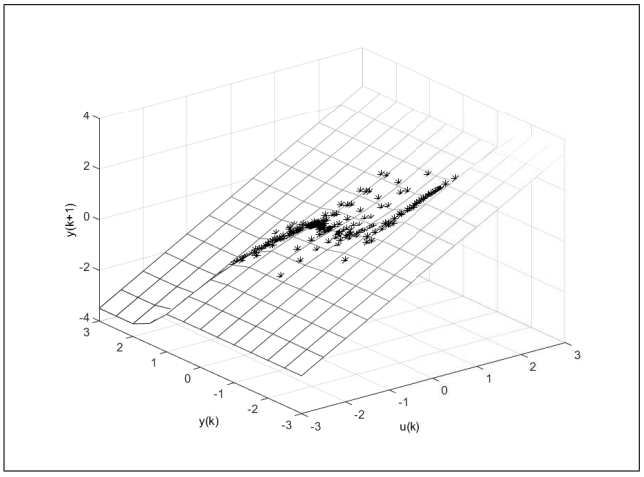

Figure 3.12: Input/output data distribution according to nonlinearity of the system to be identified

In the next step, we will select the structure of the neural network and the regressors and estimate the parameters of the network.

For the system identification we used the software NNSYSID Toolbox for Matlab [7]. We chose a neural network with a multilayer perceptron, but we could have chosen any other neural network or model that is a universal approximator or can adequately represent this system. The regressors chosen were $y(k-1)$ and $u(k-1)$. The structure selected was the NARX and the least-squares optimisation method, respectively. Following the cross-validation procedure, we chose a neural network with one hidden layer and five neurons in it. The structure of the network is shown schematically in Figure 3.13.

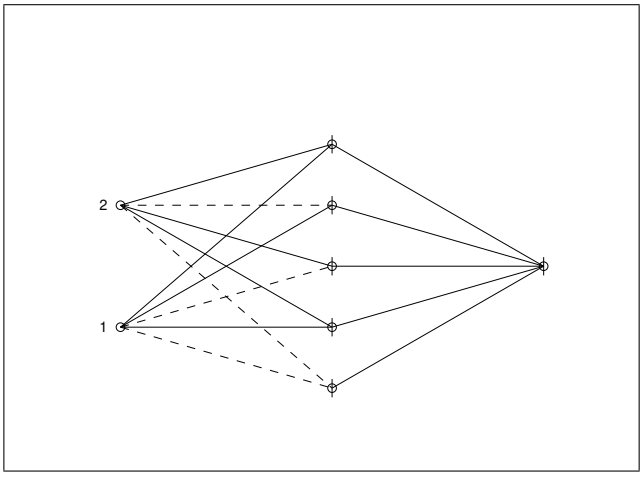

Figure 3.13: Network structure

The parameters (weights) of the network were determined using the Levenberg-Marquardt method, which we chose because of its high efficiency in nonlinear optimisation problems. The parameter values described by Equation (1.4) can be fitted with a hidden layer matrix/vector $\mathbf{W}_1 = [w_{jk}|w_{0k}]$ and the matrix/vector of the output layer $\mathbf{W}_2 = [w_{ij}|w_{0j}]$. When the optimisation was complete (i.e., the error was so small that the weights no longer changed noticeably), we obtained the following results:

$$\mathbf{W}_1 = \begin{bmatrix} -0.5588 & -2.0621 & -1.9530 \\ 0.5155 & 0.0499 & -0.8670 \\ -1.5149 & 0.3190 & 0.4768 \\ 0.3366 & -1.2029 & 1.8379 \\ 0.8411 & 1.3841 & 1.7123 \end{bmatrix},$$

$$\mathbf{W}_2 = \begin{bmatrix} 1.2054 & 1.7784 & 0.0810 & 1.1704 & 1.4048 & -0.0580 \end{bmatrix}.$$

The weight matrix W_2 has one more element than there are connections between the hidden and output layers, because it contains the values for the output offset or bias in the last position w_{0j}.

Let us take a look at the model validation that must be performed in the identification process every time we receive a new model and want to evaluate its quality of fit. We run the identification process until the model is good enough for its final purpose. We will only show the results for the final selected model. In our case, the model served to illustrate the identification process.

Figure 3.14 shows how the neural network represents the nonlinearity of the system based on the identification data. The prediction error between the nonlinearity of the system and the nonlinearity of the model (also called residuals) can be seen in Figure 3.15.

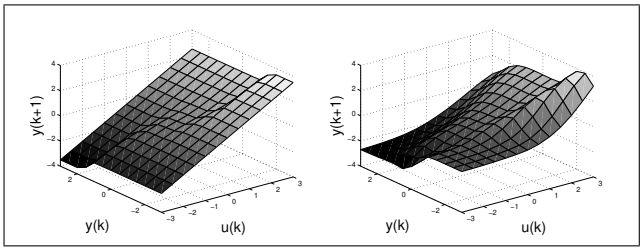

Figure 3.14: One-step-ahead prediction: system (left), model (right)

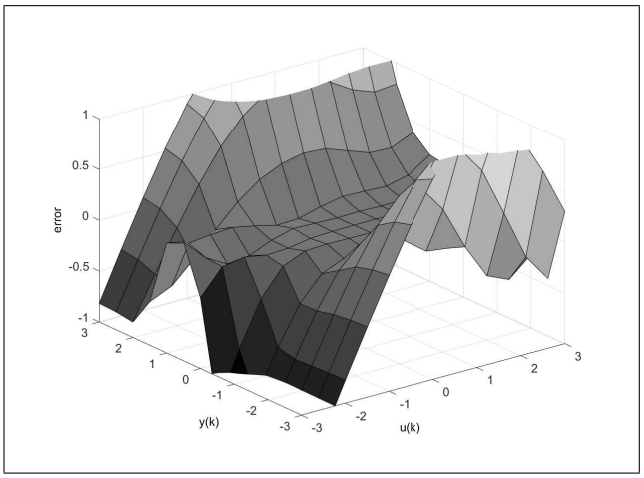

Figure 3.15: Validation of prediction error

If the model structure is suitable, the model response after optimisation differs from the process response only by white noise. The amplitude distribution of the prediction error for the validation signal, which should be normal, is shown in Figure 3.16. It can be interpreted as a rather narrow Gaussian curve. Another statistical tool that can

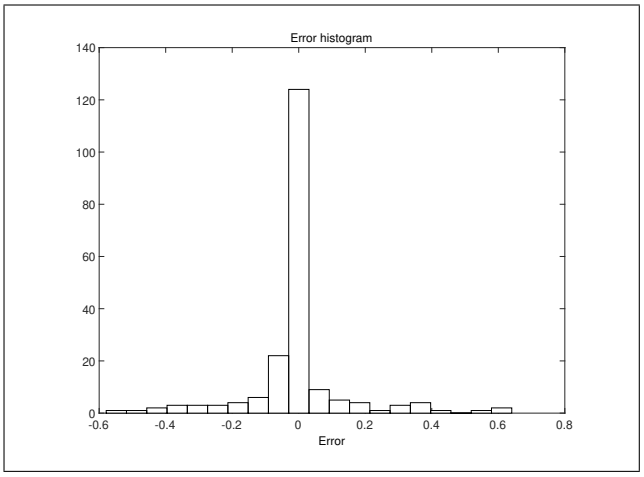

Figure 3.16: Distribution of model prediction error on validation signal

be used to check the statistical properties of the prediction error is the autocorrelation of the error (Figure 3.17), which confirms the previous observation, and the cross-correlation between the prediction error and the input signal (Figure 3.18), which shows a low correlation between the error and the input signal.

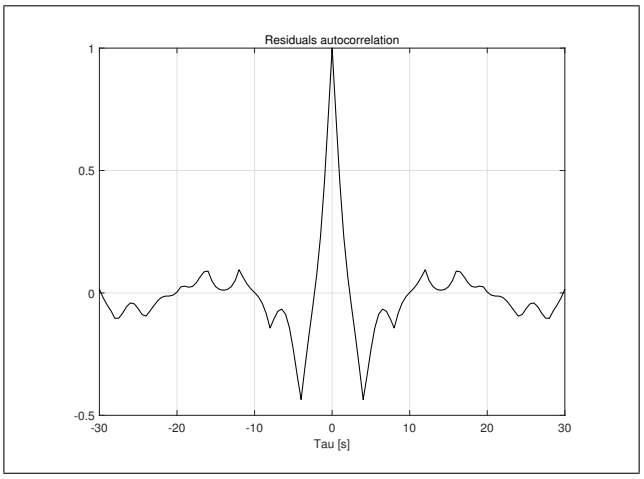

Figure 3.17: Prediction-error autocorrelation

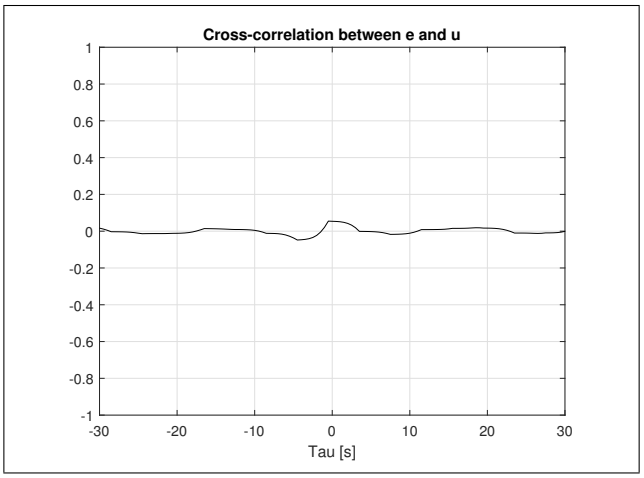

Figure 3.18: Cross-correlation of the prediction error with the input signal

One of the more informative steps is usually a qualitative or visual check of the input/output behaviour by comparing the responses of the system and model simulation with the validation signal. In Figure 3.19, which shows such a comparison, there is a clear correspondence between the two responses. Our goal of illustrating the identification process of a nonlinear process was satisfactorily achieved with the obtained model, so we completed the highly iterative process.

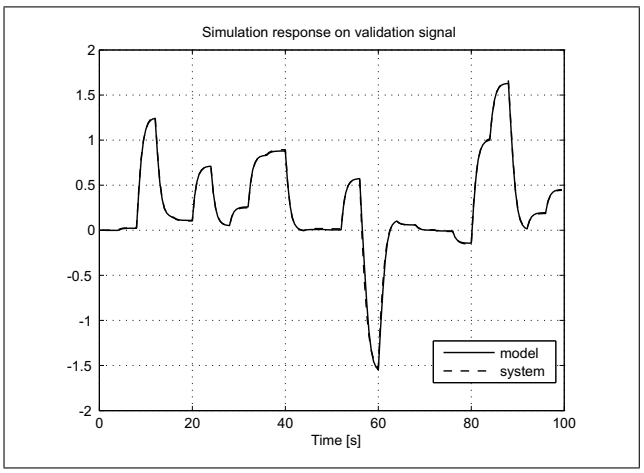

Figure 3.19: Consistency of input/output behaviour on validation signal

3.2 Example of the identification of a pH-neutralisation process

In the illustrative example in the previous section, we demonstrated the procedure for identifying the system. The demonstration was much easier because we identified a first order system by which the nonlinearity can be shown graphically. For higher order systems, this type of evaluation is more difficult. In the case of the identification of the pH neutralisation process, we will demonstrate the identification of a higher order system on a simulated system that reflects a realistic chemical process.

A simplified schematic of the pH-neutralisation process, which is often used in the literature as a benchmark for various modelling and design methods, is shown in Figure 3.20. The purpose of this process is to chemically neutralise the input fluid. The process has three input flows: acid (Q_1), input liquid (Q_2) and base (Q_3), which are mixed in a vessel T_1. Before mixing, the acid enters the vessel T_2, which gives the system additional dynamics. The flow of the acid and base is controlled by automatic control valves, while the incoming liquid is measured only by a flow meter, in particular a rotameter. The measured output variable is the acidity, meaning the pH (pH) of the mixture. Since the pH sensor is located at the outlet of the tank T_1, there is also some dead time in measuring the acidity. In this example, the input to the process is the flow rate of the base, not the flow rate of the input liquid. A more detailed description of the process can be found in [8].

A theoretical dynamic model of the pH neutralisation process was derived from chemical equilibrium equations. The derived model also includes the dynamics of the sensors and valves as well as the dynamics of the hydraulics of the outlet flow. The modelling assumed ideal mixing of the liquids, their constant density, and the complete solubility of the ions. The theoretical model of the pH neutralisation of the chemical process is described by the system of Equations (3.12) and the data given in Table 3.1. Instead of performing the measurements on the inaccessible process, we used the simulated data to identify a surrogate model. The theoretical model used contains several nonlinearities, including the implicitly calculated and strongly nonlinear titration curve, which is a typical element of all pH-neutralisation processes.

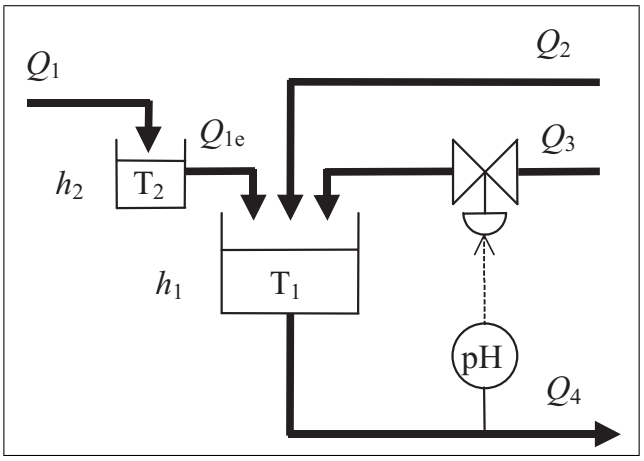

Figure 3.20: Schematic of the pH-neutralisation process

$$
\begin{aligned}
\dot{x} &= f(x) + g(x)Q_3 + p(x)Q_2 \\
c(x,y) &= 0
\end{aligned}
\tag{3.11}
$$

$$
\begin{aligned}
f(x) &= \left[\frac{q_1}{A_1 x_3}(W_{a1} - x_1)\frac{q_1}{A_1 x_3}(W_{b1} - x_2)\right. \\
&\quad \left.\frac{1}{A_1}(q_1 - C_{v4}(h_1 + z)^n)\right]^T \\
g(x) &= \left[\frac{1}{A_1 x_3}(W_{a3} - x_1)\frac{1}{A_1 x_3}(W_{b3} - x_2)\frac{1}{A_1}\right]^T \\
p(x) &= \left[\frac{1}{A_1 x_3}(W_{a2} - x_1)\frac{1}{A_1 x_3}(W_{b2} - x_2)\frac{1}{A_1}\right]^T \\
c(x,y) &= 0 = x_1 + 10^{y-14} - 10^{-y} \\
&\quad + x_2\frac{1 + 2\cdot 10^{y-pK_2}}{1 + 10^{pK_1 - y} + 10^{y-pK_2}} \\
pH &= y
\end{aligned}
\tag{3.12}
$$

Figure 3.21 shows the simulation scheme for the pH neutralisation process. The nonlinearity of the process can also be shown by plotting the responses of the system to the same input signal in different operating regions. One such example is a sequence of rectangular pulses with increasing amplitude. The response of our nonlinear system to this input signal is shown in Figure 3.23. The selected identification signal, the validation signal, and the system responses to it are shown in Figure 3.24. A comparison of the signals in Figure 3.24 and the amplitude distributions in Figures 3.25 and 3.26 shows that the identification

Table 3.1: Parameters of the pH-neutralisation process at the operating point [8]

$\|q_2\|$=0.03 M $NaHCO_3$	q_2=0.55 ml/s
$\|q_3\|$=0.003 M NaOH,	q_3=15.6 ml/s
0.0005 M $NaHCO_3$	q_{1e}=16.6 ml/s
A_1=207 cm^2	q_4=32.8 ml/s
A_2=42 cm^2	$W_{a1}=3 \cdot 10^{-3}$ M
z=11.5 cm	W_{b1}=0 M
p=0.607	W_{a2}=-0.03 M
$K_{a1} = 4,47 \cdot 10^{7}$	W_{b2}=0,03 M
$K_{a2} = 5.62 \cdot 10^{-11}$	$W_{a3} = 3.05 \cdot 10^{3}$ M
$K_w = 1.00 \cdot 10^{14}$	$W_{b3} = 5.00 \cdot 10^{5}$ M
Δt=15 s	h_1=14 cm
Δt_c=1 s	h_2=3 cm
τ_{pH}=15 s	$W_{a4} = 4.32 \cdot 10^{-4}$ M
τ_h=15 s	$W_{b4} = 5.28 \cdot 10^{-4}$ M
τ_v=6 s	pH=7.0
θ=10 s	

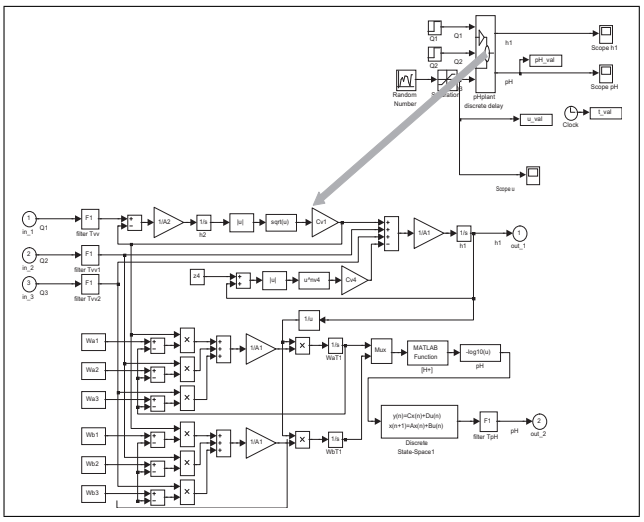

Figure 3.21: Simulink simulation scheme of the pH-neutralisation process

signal is more dynamic and has a richer distribution of amplitudes. This also generally means that the response to the identification signal contains a greater amount of information about the dynamics of the chemical process.

As in the illustrative example in the previous section, we used the same software and selected the multilayer perceptron neural network for model identification.

Using an iterative procedure, we identified the regressors that yielded the best model according to the cost function and with the most favourable relationship between the quality of the model and the smaller number of regressors that determines the complexity of the model. We chose a model with the following regressors: $y(k-1), y(k-2), y(k-3), y(k-4), u(k-1), u(k-2), u(k-3), u(k-4)$.

For the structure, we chose the NARX model structure

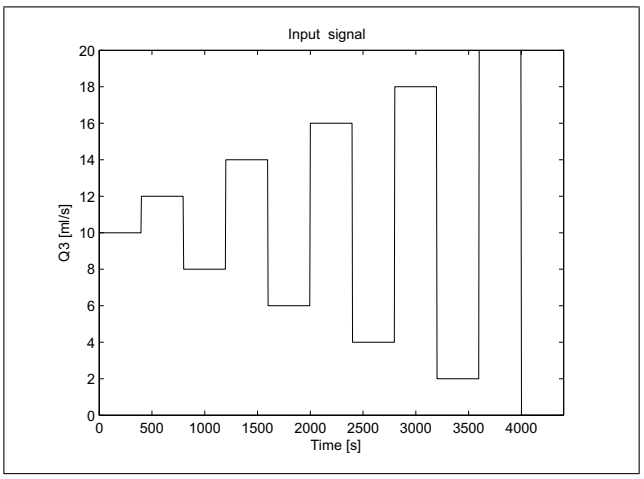

Figure 3.22: Input signal to demonstrate the nonlinearity of the system

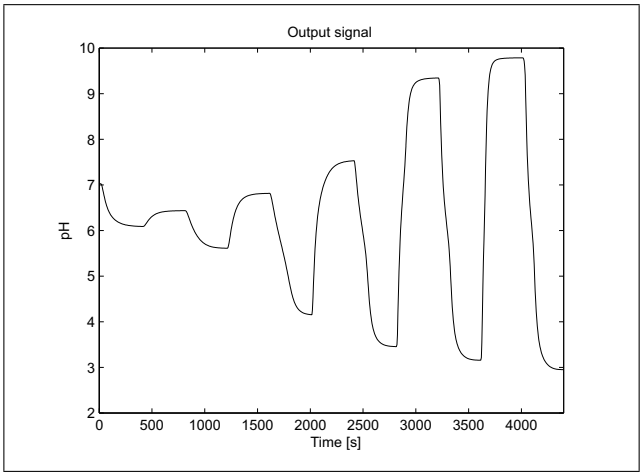

Figure 3.23: Response of the nonlinear system (T_s = 25 sec)

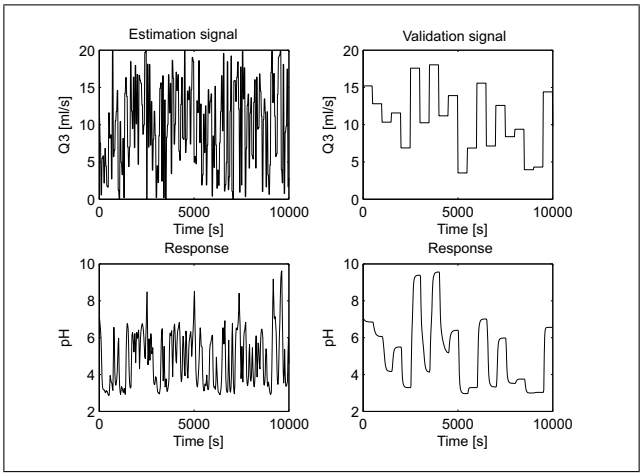

Figure 3.24: Identification signal and validation signal and their response

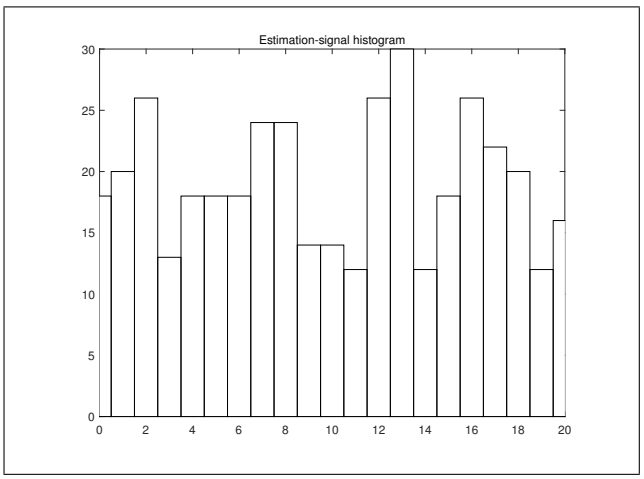

Figure 3.25: Histogram of input signal amplitudes for identification

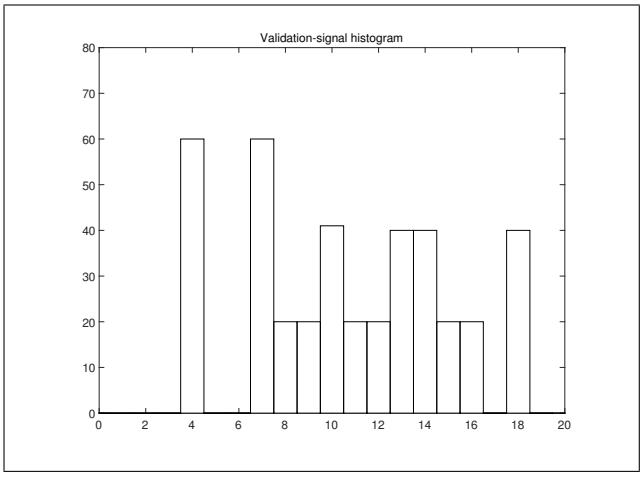

Figure 3.26: Histogram of input signal amplitudes for validation

and a hidden layer with 10 neurons using cross-validation. Note that the number of hidden layers (usually one for a so-called shallow network), the number of neurons, and the regressors are not chosen sequentially, but simultaneously. This is a cyclical process (i.e., cross-validation) in which the individual elements are changed and the cost function is validated: for the NARX model, this is the sum of the squares of the error between the process response and the model response. The parameters were optimised using the Levenberg-Marquardt method. After completing the optimisation, we also used the pruning method built into the software to remove superfluous weights, of which there were very few. A schematic of the network structure is shown in Figure 3.27.

Figures 3.28 and 3.29 show the autocorrelation of the prediction error (residuals) between the process and model responses to the signal for validation and the cross-correlation between the prediction error of the responses and the input signal. In particular, the model is evaluated by simulation even if it is essentially identified for one-step prediction.

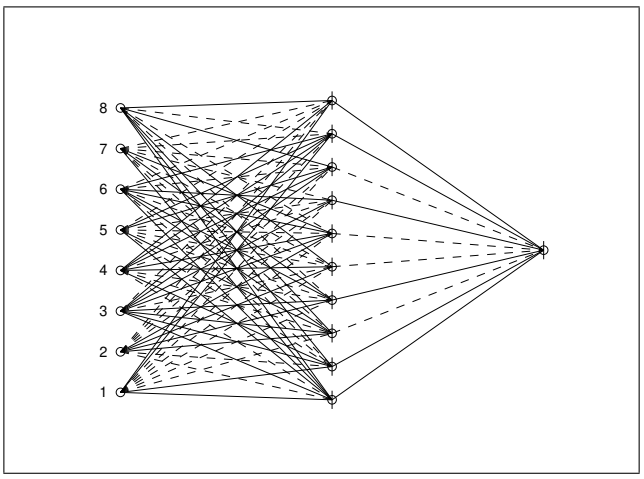

Figure 3.27: Network structure

By simulating the system and model responses to the validation signal in Figure 3.30, we can confirm what has already been done by the statistical tools in Figures 3.28 and 3.29. We have succeeded in approximating the dynamic behaviour of the pH neutralisation process relatively well with a neural network.

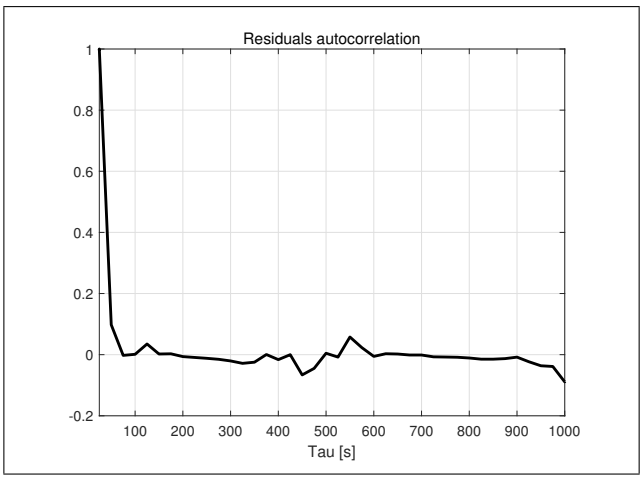

Figure 3.28: Output error autocorrelation

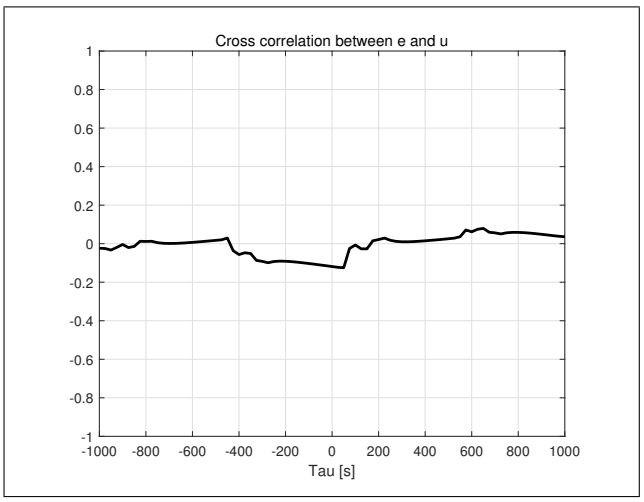

Figure 3.29: Cross-correlation of input signal and output error

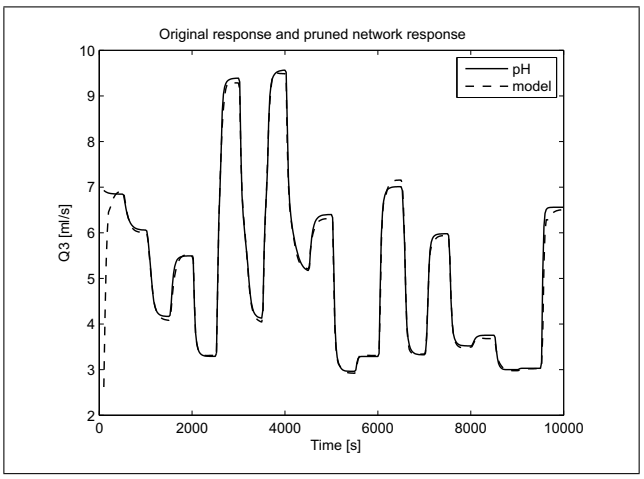

Figure 3.30: Consistency of input-output behaviour on validation signal

Bibliography

[1] A. P. Englebrecht (2002): Computational intelligence, An introduction, Wiley and Sons, Chichester.

[2] M. Gevers (2006): A personal view of the development of system identification: A 30-year journey through an exciting field, IEEE Control System Magazine, Vol. 26, No. 6, 93-105.

[3] G. B. Giannakis, E. Serpedin (2001): A bibliography on nonlinear systems identification, Signal Processing, Vol. 81, No. 3, 533-580.

[4] J. Sjöberg et al. (1995): Non-linear black-box modeling in system identification: a unified overview, Automatica, Vol. 31, No. 12, 1691-1724.

[5] M. Nørgaard, O. Ravn, N. K. Poulsen, L. K. Hansen (2000): Neural networks for modelling and control of dynamic systems, Springer, London.

[6] Deep learning toolbox for Matlab `http://www.mathworks.com/products/deep-learning.html`

[7] The NNSYSID toolbox for use with Matlab `https://www.mathworks.com/matlabcentral/fileexchange/87-nnsysid`
in
M. Nørgaard (1995): Neural network based system identification toolbox user's guide, Technical Report 95-E-773, Institute of Automation, Technical University of Denmark, Lyngby.

[8] M. A. Henson and D. E. Seborg (1994): Adaptive nonlinear control of a ph neutralization process, IEEE Trans. Control System Technology, Vol. 2, No. 3, 169-183.

[9] J. Stark, D. S. Broomhead, M. E. Davies, J. Huke (2003): Delay embeddings of forced systems: II Stochastic forcing, Journal of Nonlinear Science, Vol. 13, No. 6, 519-577.

Chapter 4

Control with artificial neural networks

4.1 Neural networks in control systems

Neural networks are, at their core, experimentally-derived mappings from input data to output data, from which the physical or other backgrounds of these mappings are not apparent. As already noted, they are representatives of black-box models. You should also bear this in mind when considering their role in the development of the automatic control of dynamic systems. Their role is multifaceted, although not as much as the role of theoretical models of dynamic systems. Neural networks are used as models of whole systems or only subsystems, as substitutes for certain nonlinearities, for control, as a model of the controller or the inverse of the system, and similar [5].

A systematic breakdown of the various uses of neural networks in the design and implementation of automatic control systems can be found in a review article [1], which also contains references to literature describing various applications. We will not go into detail but only list the applications described in [1]. The breakdown of the applications is shown in Figure 4.1.

The applications are divided into the use of neural networks as a tool for control design and as part of a controller.

Neural network only as a tool

The roles of the neural network are:

- Neural network as a modelling aid:
  - is based on an indirect target;
    - neural network identifies unknown parts in the model (Figure 4.2),
      - for control with feedback linearisation (Figure 4.3),
      - for control with instantaneous linearisation (Figure 4.4),
    - neural network predicts the controlled variable,
      - neural network model for controllers that do not need a process model but only a prediction of the controlled variable,
  - based on the control objective;
    - neural network is a model of the controlled variable (Figure 4.5),
    - neural network is a model of the reference system,
    - neural network is the model of cost function or performance index,
      - neural network helps to optimise conventional controllers,
- neural network as a supervisory aid:
  - signal combinations;
- neural network as a control implementation tool:
  - neural network maps unknowns in the controller;
    - variable controller parameters modelled by a neural network,
    - control with inverse model of part of the process,
    - predictive control (the neural network provides a model of the system to be controlled),
  - the neural network represents the solution of the implicit control law (Figure 4.6);
    - the neural network replaces the solving of computationally intensive operations (e.g., solving the Riccati equation).

Neural network as controller

The roles of the neural network are:

- learning neural network based on u:
  - neural network mimics human operator;
    - mapping the actions of the human operator,
  - neural network mimics other controller;

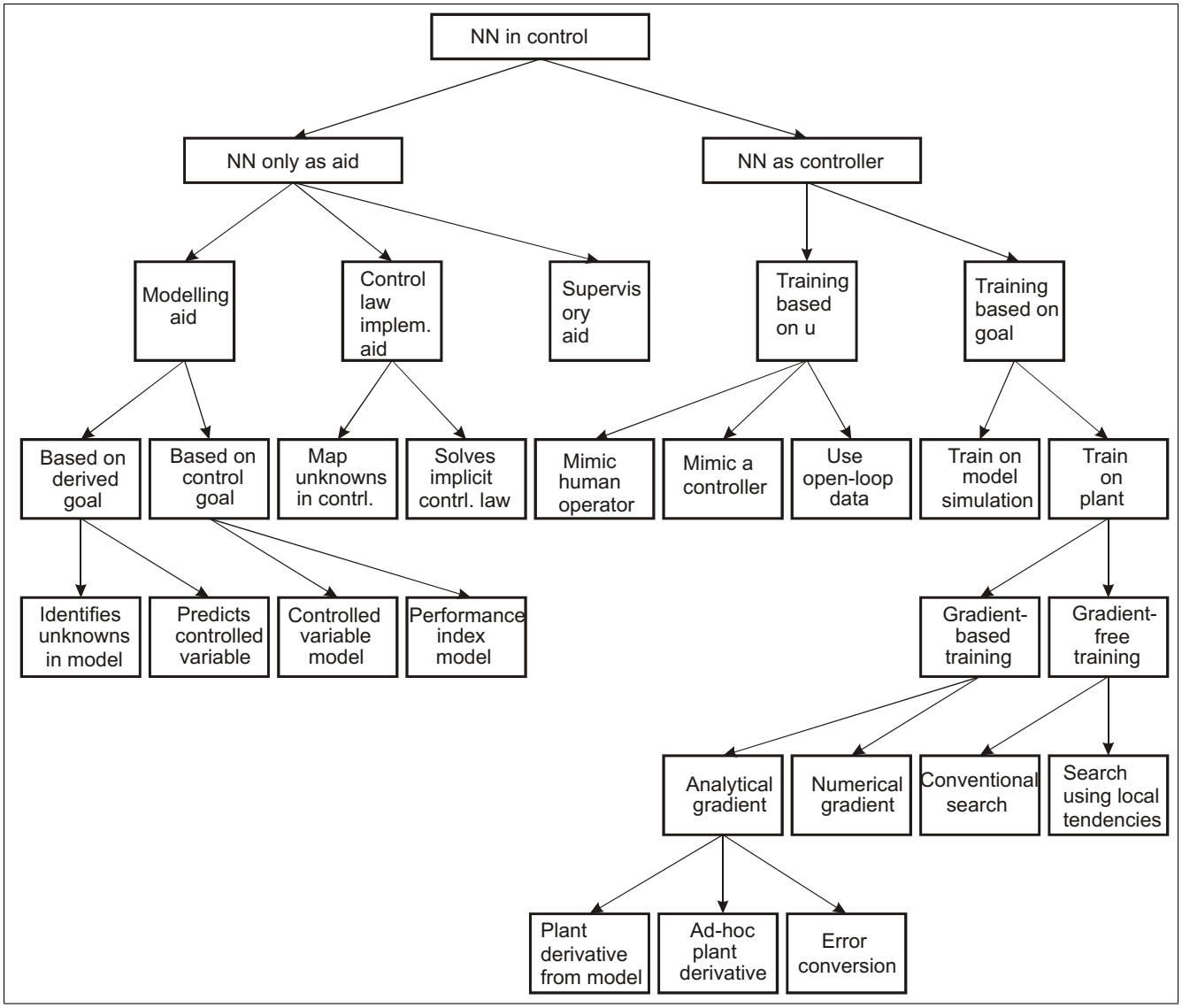

Figure 4.1: Classification of artificial-neural-network (ANN) uses in control [1]

  - ∗ mapping the characteristics of conventional controllers (Figure 4.7),
- neural network uses open loop data (Figure 4.8);
  - ∗ adaptive control with the inverse model,

- learning based on the control goal:
  - learning a neural network by simulation on a model;
  - learning on a plant;
    - ∗ learning based on the gradient of the cost function (numerical, analytical),
    - ∗ learning without cost-function gradient (search algorithms).

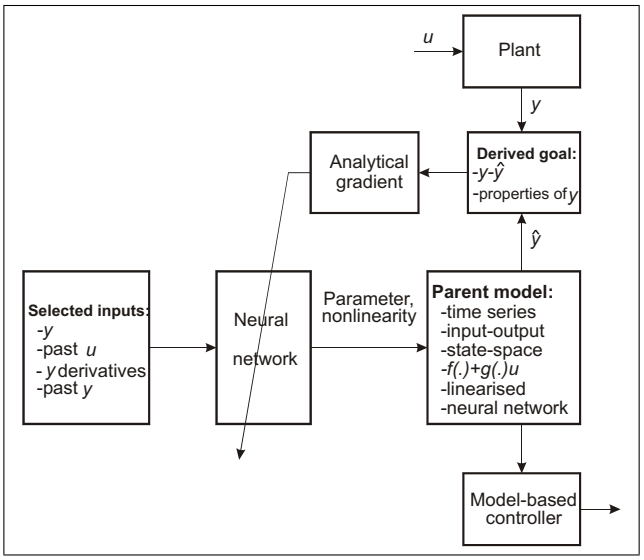

Figure 4.2: Neural network identifies unknown parts in the model [1]

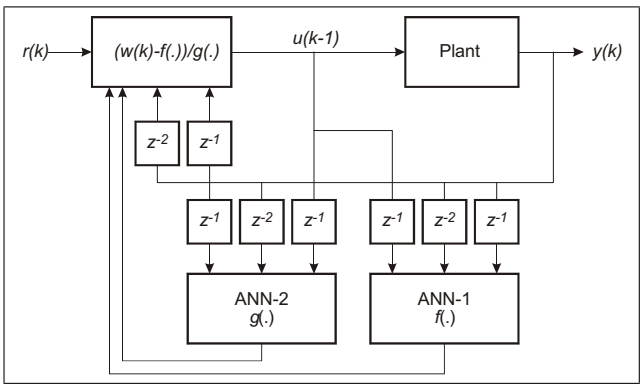

Figure 4.3: Control with feedback linearisation

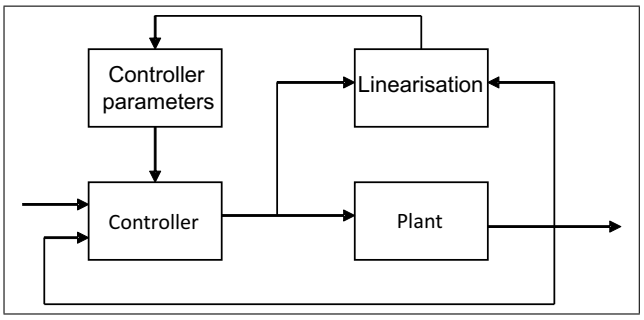

Figure 4.4: Control with instantaneous linearisation

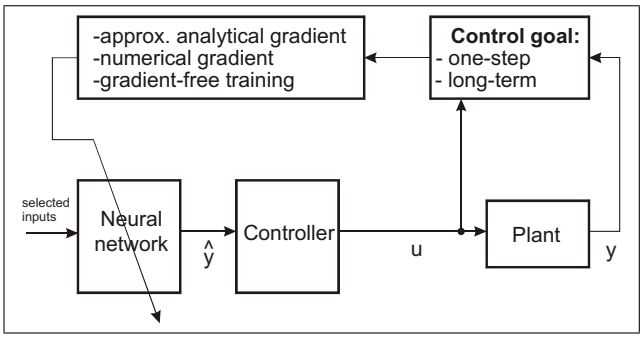

Figure 4.5: Controlled-variable model based on control goal [1]

Frequently used control methods

In the literature review describing the use of neural networks to control dynamic systems, we note that the most commonly encountered applications are:

- various forms of predictive control (which are quite useful in engineering practice), and
- various forms of adaptive control (which are somewhat less accepted in engineering practice).

Because of their applicability, especially for various black-box models, we will examine predictive control in more detail in the next subsection, in particular at how neural networks can be used in predictive control.

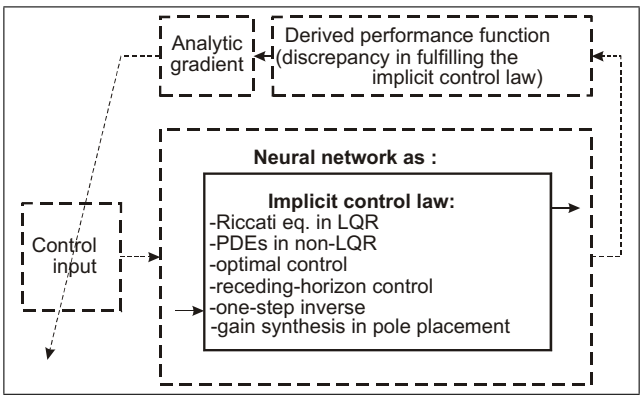

Figure 4.6: The neural network solves implicit control law [1]

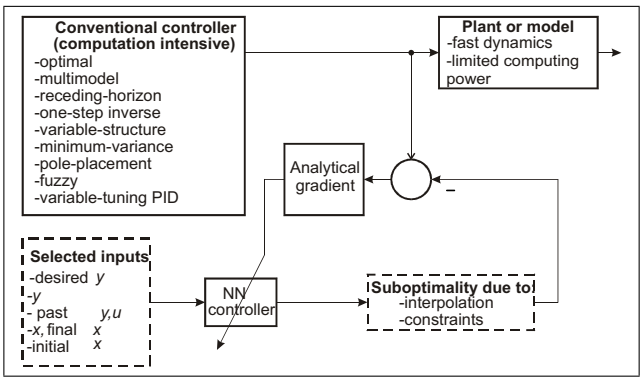

Figure 4.7: Imitating the characteristics of conventional controllers [1]

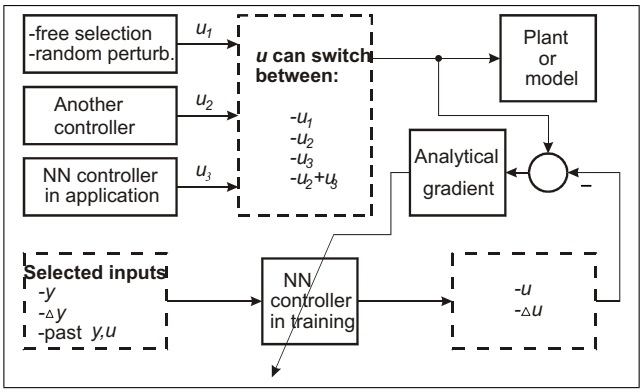

Figure 4.8: The neural network uses open-loop data [1]

4.2 Predictive control

Basic concept of predictive control

The description of the basic concept is taken from [4]. Model-Based Predictive Control (MPC, MBPC) is one of

the few advanced control methods that have gained acceptance in industrial practice, especially in the field of process control. The reasons for this can be found in the simple and easy-to-understand principle of the method, with the simple integration of constraints on the control and controlled variables, the simple design and tuning of both univariable and multivariable control for linear and nonlinear systems. Predictive control is based on the explicit use of a process model to predict the future output of the process. The control signal is based on minimising a cost function of the difference between the predicted output and the reference trajectory for a given horizon in the future. The field of predictive control includes many different algorithms with similar control principles and different models and minimisation mechanisms of cost functions derived from a model (e.g., Model Algorithmic Control (MAC), Dynamic Matrix Control (DMC), Generalised Predictive Control (GPC), Predictive Functional Control (PFC), Unified Predictive Control (UPC), etc.). The general notation when using nonlinear models is Nonlinear Model-based Predictive Control (NMPC).

Regarding the control design, the main advantages of using predictive control methods are the following:

- it is suitable for the control of systems with more complex dynamics,
- it is suitable for systems possessing dead time or minimum phase,
- its general concept allows the control of both univariable and multivariable processes,
- it allows feedforward compensation of measurable disturbances,
- captures system constraints in the design process,
- can incorporate size constraints and rate of change constraints,
- considerable freedom in design, with design parameters considered as control specifications,
- a predictive controller can form the control signal in advance if the future setpoint curve is known,
- the methods do not include explicit signal derivation so that measurement noise does not cause problems,
- the methods do not contain explicit integration so that there is no problem of integral wind-up,
- the principle is easy to understand.

The two main disadvantages of using predictive methods are:

- a good model of the system to be controlled is necessary because the quality of the control depends directly on the quality of the model,
- the computational complexity of the methods can become problematic when controlling faster systems.

The basic principles of predictive control are reflected in the following steps (Figure 4.9):

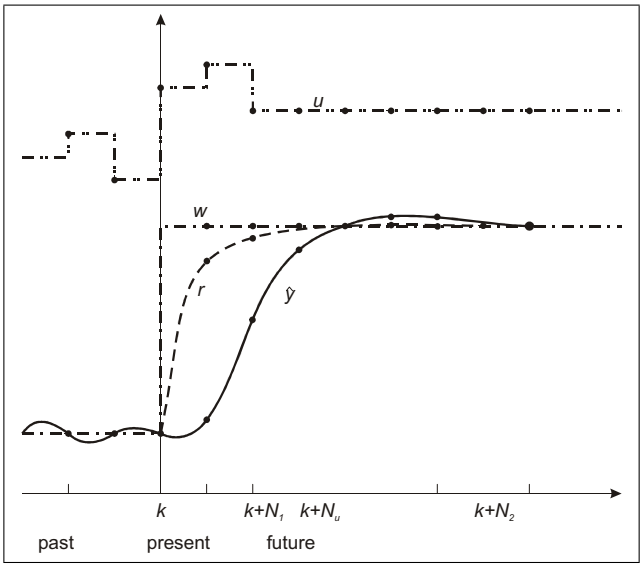

Figure 4.9: Basic concept of predictive control. w - setpoint trajectory, r - reference trajectory, u - control signal, $\hat{y}$ - model response, N_1, N_2 - start and end of prediction horizon, N_u - control horizon

- Prediction of the output signal of the system based on the system model.
  At each time instant k, we calculate the trajectory of the output signal $y(k+j)$ for the horizon in the future $j = (N_1, \ldots, N_2)$. N_1 and N_2 denote the lower and upper values of the prediction horizon, which determines the coincidence horizon within which we want the output signal to conform to the prescribed behaviour. The predicted values of the output signal of the system, expressed in terms of the system model, denoted $\hat{y}(k+j \mid k)$, represent the j-step prediction of the model. The predicted values depend on the future control scenario $u(k+j \mid k); j = 0, \ldots, N_u - 1$, which we want to use from time instant k onwards.
- Creation of a reference trajectory.
  By defining a reference trajectory $r(k+j \mid k); j = N_1, \ldots, N_2$, we determine the desired time course of the system from the current value $y(k)$ to the given setpoint value $w(k)$.
- Determination of future control signal.
  The vector of the future control signal $u(k+j \mid k); j = 0, \ldots, N_u - 1; N_u \leq N_2$ is calculated by minimising the corresponding cost function by which we minimise the error between the $r(k+j \mid k)$ and $\hat{y}(k+j \mid k)$. The determination of the future signal is

based on the application of the open loop optimality criterion in a certain interval in the future.

- Use of the first element of the control signal vector to control the system. Only the first element $u(k \mid k)$ of the optimal vector of the control signal $u(k+j \mid k); j = 0, \ldots, N_u - 1$ is used.

At the next time instant, when we have a new measurement of system output, we repeat the whole process. This principle is called 'the receding-horizon strategy'. Many different solutions are proposed for each step, which makes the different predictive control methods different from each other.

The basic scheme of a closed-loop predictive control system is shown in Figure 4.10. While the basic model may

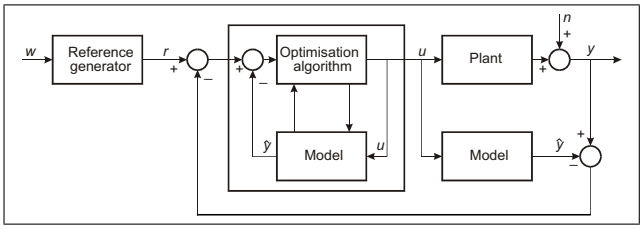

Figure 4.10: Common concept of proposed predictive control principles

already include a model of measurable disturbances, the feedback loop is incorporated into the algorithm to eliminate one-step-ahead model prediction errors, due to unmodeled dynamics and other (nonmeasurable) disturbances in the system. The error is expressed as the difference between the system output and the model output at the time of sampling:

$$e(k) = y(k) - \hat{y}(k). \tag{4.1}$$

When predicting the future output of a system, the error $e(k)$ is added to the model prediction $\hat{y}(k+j \mid k)$, for each $j = N_1, \ldots, N_2$. Assuming a constant error $e(k)$ for the entire prediction horizon, we compensate for the errors of the steady-state model and the constant disturbances in the system. The prediction algorithm is based on the deviations between the model output and the measurements and estimates the future effects of unmeasurable disturbances acting on the process. Ideally, the model matches the process and no disturbances are present. In this case, the feedback loop is not functional and the control is an example of open-loop optimal control.

To predict the output of the system, the use of the system model is essential. It can be used to calculate the prediction of the output signal several steps into the future. In principle, we can use any linear or nonlinear model. This includes both theoretical and experimental models. Examples of such models of systems include neural networks, fuzzy models, Gaussian-process models, among others. The models used are usually, but not exclusively, written in the discrete-time domain.

Different predictive algorithms with a nonlinear model can be divided according to the way the nonlinear control problem is solved:

- The direct nonlinear approach writes the control problem in the form of a nonlinear programming problem and solves it using iterative optimisation methods. This approach follows directly from the idea of using a nonlinear model in prediction. However, due to the solution of the nonconvex optimisation problem with additional constraints, it is computationally intensive.
- The linearisation approach is more complex due to the simplification of the optimisation problem and the use of linear predictive control algorithms for the linearised system. The linearisation methods used are feedback linearisation or inverse nonlinear mapping, online model linearisation at the operating point, using a set of local linear models, and some other methods.

All predictive design methods assume knowledge of the trajectory of the setpoint signal in the future, but this assumption is not always met. Usually, we specify how the system output approaches the setpoint signal, which is determined by forming the reference trajectory $r(k+j \mid k); j = N_1, \ldots, N_2$. The reference trajectory can be considered as the internal reference of the predictive controller, which determines the desired closed-loop behaviour and from which the vector of the future control signal is determined. When forming a reference trajectory in relation to the knowledge of the setpoint signal, two situations are distinguished:

1. The setpoint signal $w(k+j)$ is known in advance for all $j = 1, \ldots, N_2$. Due to the principle of predictive control, which predicts future behaviour, the predictive controller initiates the appropriate control action before the setpoint signal changes. This compensates for dead times and large time delays in the system. In robotic systems, tracking systems, batch processes, and similar, the setpoint signal may be known in advance.
2. The setpoint signal $w(k+j)$ is not known in advance. As the best possible prediction of the setpoint signal, we take its current value $w(k+j) = w(k)$.

Given the starting point of the reference trajectory, we distinguish two situations:

1. We use the current measured value of the output signal $r(k) = y(k)$. In this way, we introduce an additional feedback loop into the system, the effect of which on the overall closed-loop system is difficult to evaluate because it is not the result of optimising the cost function. In some cases, it may even destabilise the control loop.

2. We use the current value of the reference trajectory $r(k)$.

The open-loop-optimal calculation of the system's response at a certain interval in the future is the main feature of predictive control. In this respect, there are similarities with the linear-quadratic (LQ) controller [4], but the LQ controller uses an infinitely long prediction horizon. Choosing the appropriate cost function is the first step in determining the vector of the future control signal $u(k+j \mid k); j = 0, \ldots, N_u - 1$. Here we tend to choose the cost functions that are easy to compute and whose minimum can be found analytically or by optimisation algorithms. Since the desired behaviour of the system in the future is determined by the reference trajectory, the logical choice of the cost function is the difference between the predicted response of the system and the reference trajectory, for example:

$$J = \sum_{j=N_1}^{N_2} (r(k+j) - \hat{y}(k+j))^2. \tag{4.2}$$

The parameters N_1 and N_2 determine the fitting horizon in which we want the predicted output of the system to match the reference trajectory as closely as possible. By increasing the parameter N_1, we omit the influence of control errors in the near future, especially for phase-minimal or dead-time systems, resulting in a more steady and smoother control. The extended form of the cost function is described by equation:

$$J = \sum_{j=N_1}^{N_2} \alpha_j (r(k+j) - \hat{y}(k+j))^2. \tag{4.3}$$

Weight vector $\boldsymbol{\alpha} = [\alpha_{N_1}, \ldots, \alpha_{N_2}]$ can be used to further influence the significance of the errors at each time instant in the prediction horizon. It is also common to include a measure of the variation in the control signal in the cost function. Through this measure, we seek to reduce the variation of the control signal by the cost of increasing the deviation between the predicted output of the system and the reference trajectory:

$$J = \sum_{j=N_1}^{N_2} (r(k+j) - \hat{y}(k+j))^2 + \sum_{j=0}^{N_u} \beta (\Delta u(k+j))^2. \tag{4.4}$$

The above cost function contains the vector of changes in the control signal $u(k)$. In many methods, we use the optimal vector of variations of the control signal $\Delta u(k+j \mid k); j = 0, \ldots, N_u - 1$ instead of the absolute values $u(k+j|k); j = 0, \ldots, N_u - 1$.

The predicted future signal $\hat{y}(k+j|k); j = 1, \ldots, N_2$ depends on the predicted vector of the control signal in the future $u(k+j|k); j = 0, \ldots, N_u - 1; N_u \leq N_2$. In general, the elements of the vector are arbitrary and independent of each other, which increases the computational complexity and also the time consumption of the optimisation enormously by increasing N_u. The control signal can also become rich in unwanted high-frequency components as a result. In practice, we always choose to structure the vector of the control signal by introducing relationships between the elements of the vector. Such a decision also increases the robustness of the predictive control. Due to the principle of the receding horizon and the use of only the first element of the vector of the control signal, the actual control signal is not limited by the introduction of structuring. The most commonly used techniques for structuring the control signal [4] are:

- The introduction of a control horizon after the transient response assumes a constant control signal assuming a constant setpoint signal in the future. Control horizon $N_u; N_u \leq N_2$ represents the time from which the control signal will remain constant. This reduces the number of optimisation variables, specifically the elements of the vector of the control signal. The simplest and most frequently used value in practice is $N_u = 1$, which also provides good results with a varying setpoint signal.

- The clustering technique assuming $N_u = N_2$ divides the entire prediction horizon into a fixed number of segments within which it assumes a constant control signal.

- The basis function representation algorithm structures the control signal as a linear combination of independent, predetermined basis functions (linear, polynomial, etc.). The choice of basis functions depends on the system and the setpoint signal. The calculation of the control signal is thus reduced to the calculation of the selected parameters of the basis functions.

After selecting the form of the cost function and the structure of the vector of the future input signal, control design is followed by the determination of the values of the elements of this vector. The solution is found by determining the minimum of the convex optimisation problem when linear models are used, without using constraints. The solution can often be determined analytically, meaning that the output of the process depends linearly on the past values of the inputs and outputs of the system, or by one-step optimisation methods, for example, the method of least squares. Imposing constraints on the minimisation of the cost function using linear models requires finding the minimum of the cost function using iterative optimisation algorithms. The optimisation problem remains convex in this case. The use of nonlinear models usually results in the optimisation problem not being convex; consequently, time-consuming and costly iterative algorithms have to be used. Convergence to the true solution is no longer guaranteed within the prescribed time or number of optimisation iterations. The additional inclusion of constraints for system variables in the chosen optimisation algorithm can even lead to the minimum of the cost function being indeterminate with respect to the constraints (e.g., in the

case of unpredictably large disturbances). Both possibilities are unacceptable for the implementation of nonlinear predictive control in real systems. Several approaches have been proposed in the literature when such a situation occurs [4]. Such optimisation methods, for example, interior point methods [4], are concerned with taking into account time constraints and constraints on the result in each optimisation step and providing at least an approximate solution. If necessary, the optimisation process may be terminated prematurely due to time constraints or the optimisation process may not find a solution to the optimisation problem.

An interesting possibility is to use 'soft' constraints, by which we assume that the constraints represent limits that should not be exceeded. Using the structure of the cost function, we introduce a mechanism that allows this to be done in an emergency. To do this, it is necessary to define two sets of constraints:

1. constraints imposed for physical, safety and similar reasons (e.g., the range and rate of change of the control variable are limited by the actuator used) and must not be softened,
2. other constraints that may be softened (e.g., in the cases in which we may violate the limits that define acceptable product quality).

Soft constraints are implemented by adding an additional soft constraint function to the criterion used, which is not zero only in the case where the constraints are violated. In this way, the optimisation algorithm only violates constraints when an emergency situation occurs (e.g., in the case of unforeseen large disturbances [4]).

Again, only the first element $u(k|k)$ of the optimal vector of the control signal $u(k+j|k); j = 0, \ldots, N_u - 1$ is always implemented. Due to the receding horizon strategy and the finite length of the prediction, the response of the control system generally does not correspond to the optimal open-loop response based on which the optimal vector of the control signal was determined.

Illustrative example of control with a predictive controller

The following is an example of control with a predictive controller that uses a neural network for prediction.

The controller is defined for a system mathematically described by the nonlinear difference equation (3.10):

$$y(k) = y(k-1) - 0.5 \tanh(y(k-1) + u^3(k-1)),$$

where u is the input and y is the output of the system. The design process can be summarised in three steps:

Step 1 - identification of the system in the operating region region: Since this is the same system that we have already identified in Chapter 3, here only the most important features are summarised:

- regressors: $y(k-1)$, $u(k-1)$,
- structure: NARX model,
- a hidden layer with five neurons, multilayer perceptron,
- Levenberg-Marquardt optimisation method.

Model validation was also shown in Chapter 3. This time, we only consider the contour plot of the one-step ahead error predictions in Figure 4.11, which shows the operating range in which the model is useful.

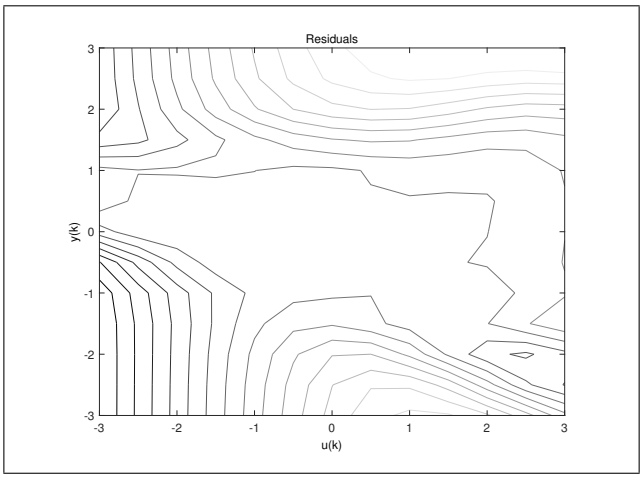

Figure 4.11: Response-error-validation (one-step-ahead prediction)

Step 2 - Design of a predictive control:

- We have selected Predictive Functional Control (PFC) as one of the simplest forms of MPC [4]. A feature of most PFC methods is the reduction of the prediction horizon to a few points. In the case of a single point, the prediction horizon (also called the coincidence horizon) is called the coincidence point. A more detailed description of the method used PFC can be found, for example, in [4] and [3].
- The PFC cost function we use is:

$$J = \min_{\mathbf{U}(k)} [r(k+P) - \hat{y}(k+P)]^2 . \qquad (4.5)$$

- As random points of coincidence, we have experimentally determined the value of eight samples in advance. The reference trajectory was chosen so that the value of the control signal approaches the reference value exponentially with a time constant of 10 samples.

- Some important properties to consider when developing a predictive control:
  - computational complexity,
  - robustness of closed-loop control to differences between the controlled system and its model,
  - consideration of the constraints of the different variables we are dealing with.

Step 3 - Evaluation of the designed control system: the response of the closed-loop control system and the corresponding control signal are shown in Figure 4.12. From this, it can be seen that the predictive control agrees with the reference curve trajectory, as long as the closed-loop system operates in the range in which the model describes the controlled system well. However, at the last step change, the closed-loop system falls outside the range of the good description of the model, resulting in a response that does not correspond to the specified requirements. This can be avoided by constraining the control to the range restricted to the range in which the model is sufficient.

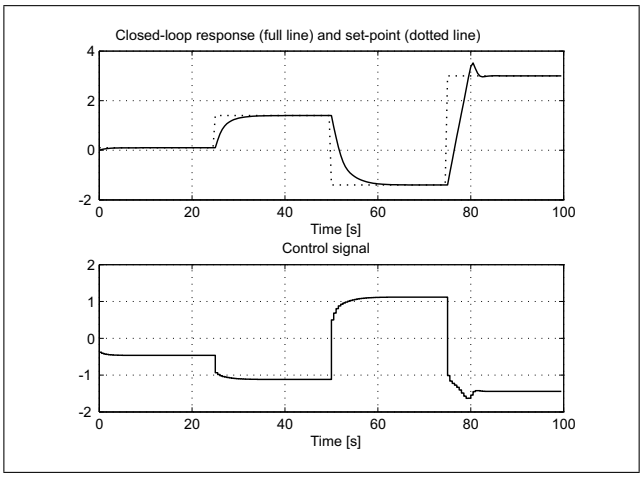

Figure 4.12: Response of predictive control based on neural network model

Bibliography

[1] M. Agarwal (1997): A systematic classification of neural-network-based control, IEEE Control Systems Magazine, Vol. 17, No. 2, 75-93.

[2] W. S. Levine (1996): The control handbook, CRC Press, Boca Raton, Fl.

[3] J. Kocijan, R. Murray-Smith, C. E. Rasmussen, B. Likar (2003): Predictive control with Gaussian process models. In The IEEE Region 8 EUROCON 2003. Computer as a Tool. (Vol. 1, pp. 352-356). IEEE.

[4] J. M. Maciejowski (2002): Predictive control with constraints, Pearson Education Limited, Harlow.

[5] M. Nørgaard, O. Ravn, N. K. Poulsen, L. K. Hansen (2000): Neural networks for modelling and control of dynamic systems, Springer, London.

[6] M. Nørgaard (1995): Neural network based control system design toolkit user's guide, Technical Report 96-E-830, Institute of Automation, Technical University of Denmark,Lyngby.
`https://www.mathworks.com/matlabcentral/fileexchange/86-nnctrl`

Chapter 5

Local-model networks and blended multiple-model systems

'Multiple-model systems' is a common term for various modelling and control design methods for nonlinear systems composed of less complex subsystems. They are called different names depending on the field of research from which they originate. The best-known multiple-model systems are fuzzy Takagi-Sugeno models, blended multiple-model systems, multiple-model switching systems, Markov mixtures of experts, among others. An overview of multiple-model modelling systems can be found in [15], and some later developments in the special issue of a journal dedicated to the topic [7].

Since we have dealt with neural networks in the previous chapters, let us now start from the point of view of neural networks, from which can look at multiple-model systems as networks of local models, which we will tentatively describe in the following chapter. The description is the same for other forms of multiple-model systems.

Hierarchical networks

As we have seen, neural networks can be divided into those with a ridge structure and those with a radial structure. A typical representative of a neural network with a ridge structure is the multilayer perceptron (Figure 5.1), in which the nonlinearity of the modelled system is approximated by a ridge basis function:

$$z_i = f_i \left(\sum_{j=1}^{m} w_{ij} x_j + w_{0j} \right), \qquad (5.1)$$

where x_j is the j^{th} input, w_{ij} is the weight at node ij, w_{0j} is the bias for the j^{th} input, f_i is a ridge basis function for the i^{th} output, z_i is the i^{th} output.

Representative for neural networks with radial structures are radial basis-function networks (Figure 5.2), where the nonlinear hyperplane of the modelled system is approximated by differently weighted basis-function centres γ, specifically by points scattered over the space of the regressors:

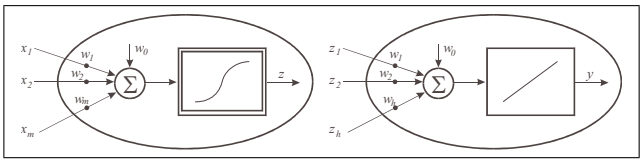

Figure 5.1: Multilayer perceptron

$$z_i = \exp\left(-\frac{1}{2} \sum_{j=1}^{m} \frac{(x_j - \gamma_{ij})^2}{\rho_{ij}^2} \right), \qquad (5.2)$$

where ρ_{ij} is the scaling factor of the radial function.

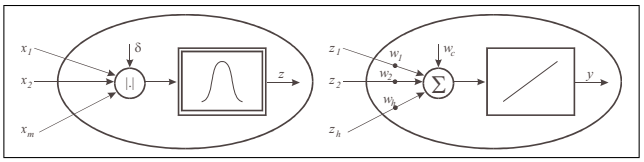

Figure 5.2: Radial basis-function networks

These centres are more or less evenly distributed across the area of interest (Figure 1.16). They are typically also located in the operating points of the nonlinear system, where there usually is a lot of measurement data available to identify the system.

Local-model networks and blended multiple-model systems

The centres of the radial basis functions are usually selected as the individual points defined by the input regressors, which are the centres of the measured data sets. These data sets can be replaced by local models obtained from these data. In this way, the amount of data for modelling can be significantly reduced without sacrificing information content. Instead of weighing the data centres to obtain the best fit of the modelled nonlinearity,

we weigh the local sub-models and combine them into a global model of the modelled nonlinearity. This structure is called a 'Local-Model Network' (LMN). We optimise the parameters, usually the weights, in a similar way to other neural networks. Local models, which can be obtained in any way, are combined in different ways to form a global model. We can weigh the outputs of the individual local models or, if the local models have the same structure, we can weigh the parameters of the local models into a single structure. The simple principle of a network with local-model network is shown in Figure 5.3.

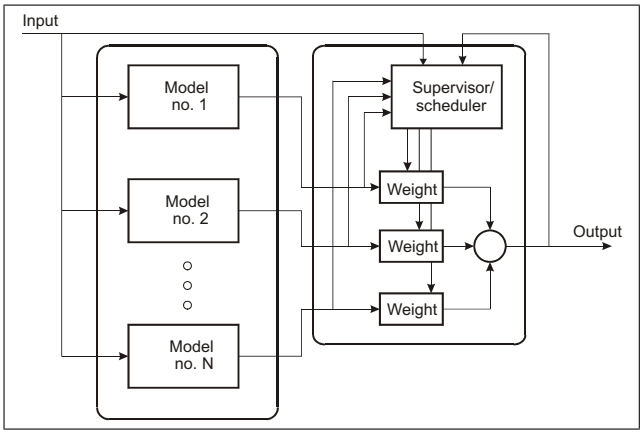

Figure 5.3: A principle of local-model network with weights on local-model outputs

Dynamic systems

If the described principle works well for describing static systems, care must be taken when modelling dynamic systems.

A typical approach for both modelling and designing the control of dynamic systems is the so-called 'divide and conquer' approach. This means that the problem (a nonlinear dynamic system being modelled or the global nonlinear controller being designed) is usually divided into linear subproblems, because linear methods are well established. We then solve the individual linear subproblems, which are valid only in a small domain, and combine them into a global nonlinear solution. Figure 5.4 shows a possible implementation of control based on a network of local models.

Linearisation of the nonlinear system is the core of the modelling method described. The usual type of linearisation is the first-order Taylor expansion around the operating point or identification around this point. We should be aware that such an approach is only valid in the neighbourhood of the chosen operating point.

The standard method for modelling with local-model networks is to treat linearised models around a representative number of equilibrium points. The equilibrium points

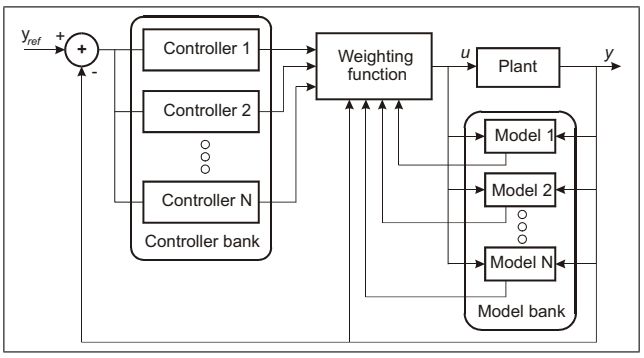

Figure 5.4: Block diagram illustrating a closed-loop adaptive controller in the form of blended local controllers whose parameters vary according to the variation of the local system models

(Figure 5.5) are those to which the nonlinear dynamic system tends. In the time domain, these points often represent constant equilibrium states where a nonlinear system can settle itself. The operating points are usually, but not

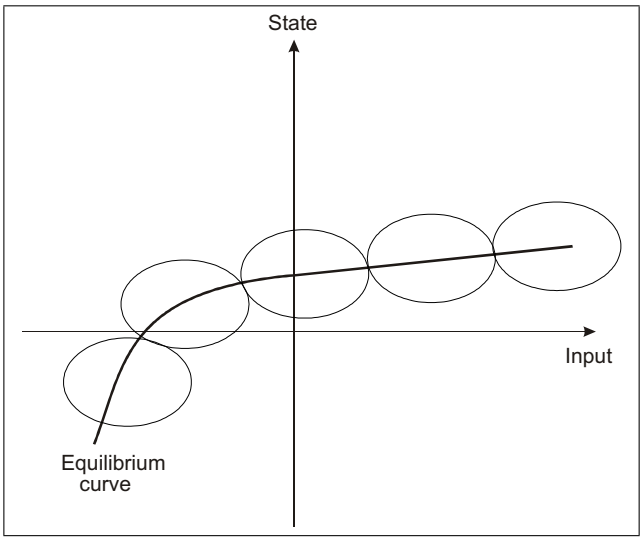

Figure 5.5: Model of a nonlinear system as a family of linear systems; linearisations only in equilibrium points

necessarily, the equilibrium points.

The question arises as to how we divide the entire nonlinear domain of a nonlinear system into subdomains. There are several possibilities. They are systematically described in [15], and a description of the solutions can also be found in [2], [18], [14].

One possible approach is to evenly divide the domain of action or nonlinearity being modelled. This is useful for problems of lower complexity and is illustrated in Figures 5.6 and 5.7.

It is also possible to divide the workspace into work subspaces according to the operating regimes of the nonlinear system. This is particularly useful when modelling, for

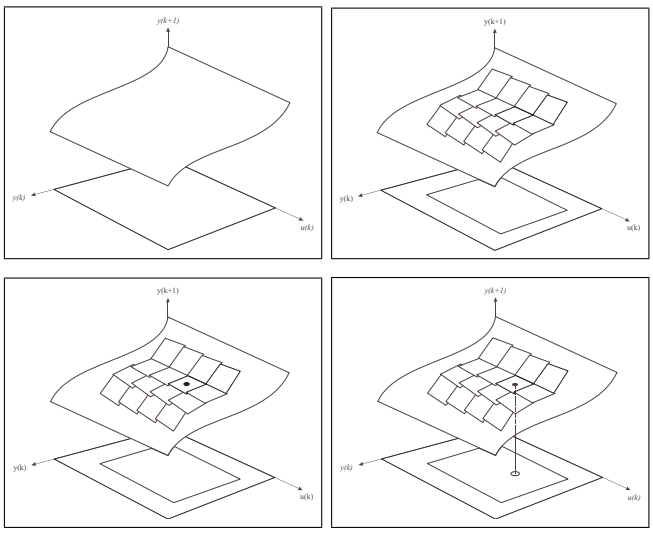

Figure 5.6: Modelling a nonlinear dynamic system with uniformly distributed local linear models

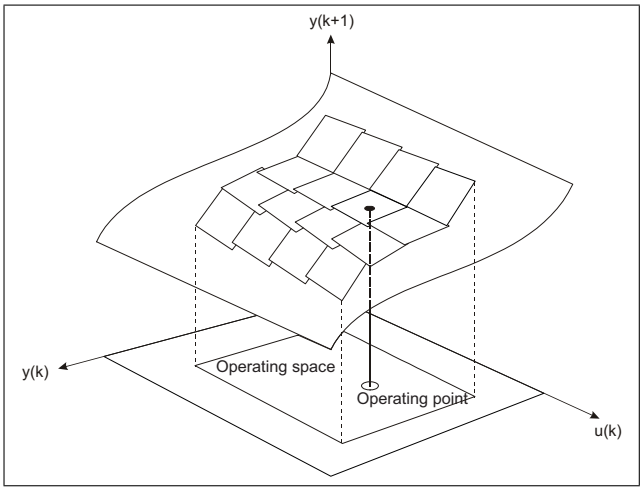

Figure 5.7: Modelling a nonlinear dynamic system with uniform distributed local linear models - detail from Figure 5.6

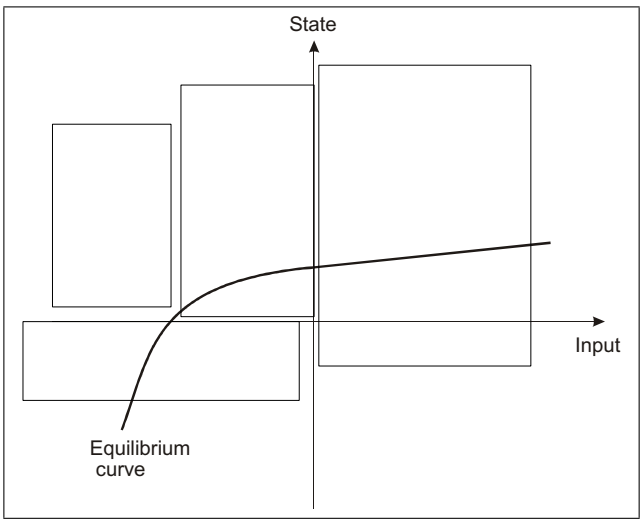

Figure 5.8: Division of the operating region of a nonlinear system into operating sub-regions according to similar operating characteristics

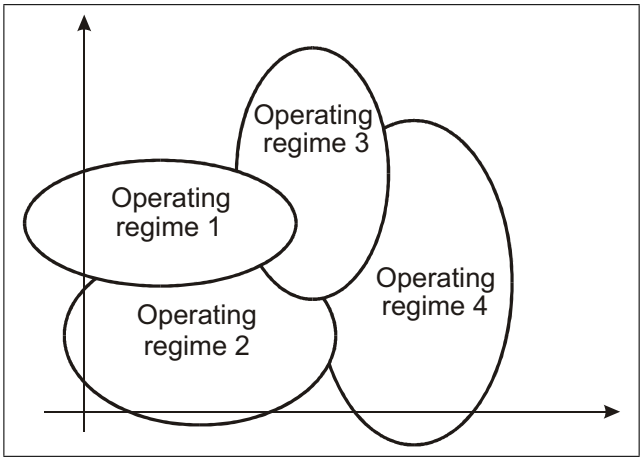

Figure 5.9: Alternative partition of the operational region of the nonlinear system

example, an industrial system that has different operating regimes and shows different dynamic behaviours in each of them. Again, this approach is only useful for a limited number of applications and is illustrated in Figures 5.8 and 5.9.

Various ways of modelling local-model networks are described in addition to the publications already mentioned in [1], [4] with the software described in [5], the control applications in [6], [17], [19] and many others.

Topics of interest

When linearising a nonlinear system $\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x}, \mathbf{u})$, where $\mathbf{x}$ is the vector of system states and $\mathbf{u}$ is the vector of inputs, with the Taylor expansion at the operating point $(\mathbf{x}_0, \mathbf{u}_0)$ from a known nonlinear system, one does not obtain a linear system but an affine system that is linear in parameters.

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{F}(\mathbf{x}, \mathbf{u}) \text{ is linearised into} \\ \dot{\mathbf{x}} &= \mathbf{F}(\mathbf{x}_0, \mathbf{u}_0) + \nabla_x \mathbf{F}(\mathbf{x}, \mathbf{u})(\mathbf{x} - \mathbf{x}_0) + \\ &\quad \nabla_u \mathbf{F}(\mathbf{x}, \mathbf{u})(\mathbf{u} - \mathbf{u}_0) + \text{higher order terms} \end{aligned}$$

A constant element defining the operating point of the system ($\mathbf{F}(\mathbf{x}_0, \mathbf{u}_0)$) means that the superposition condition is not satisfied because the system is not linear, even if reformulated to look linear. This constant element can be very large, which means that the linearised model cannot be a linear system with a small constant disturbance. When we move from one working point to the next, the constant element changes. This means that it is connected to the

dynamics of the system; therefore, it cannot be treated as an external disturbance or even neglected.

This is just one of the problems that arise when modelling nonlinear dynamic systems with a local-model network.

Another problem is the description of the dynamics of a system in nonequilibrium regions [8],[16], which results from the 'tendency' of states of stable dynamic systems to converge to equilibrium regions. The system is thus only briefly in regions away from the equilibrium curve (Figure 5.10) and only a small amount of data exists to describe these regions. The problem would not be so great if the systems could be arbitrarily excited in practice to obtain information describing the entire operating region. However, due to the nature of the system and other constraints on the excitation signal, the design of the experiment is usually limited.

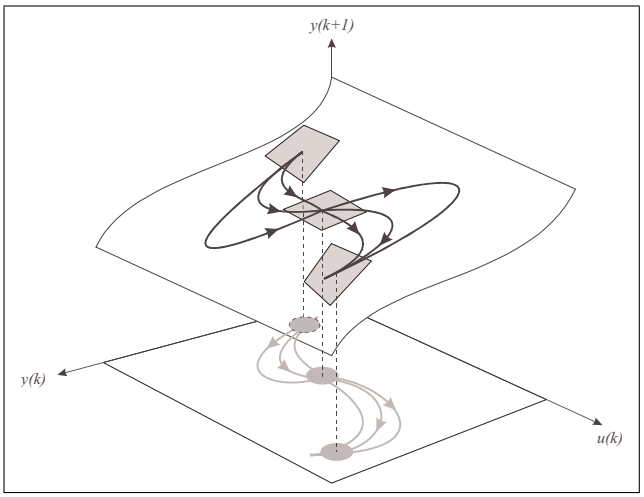

Figure 5.10: Switching from one local model to another

The next problem is to optimise the parameters that determine the selected dynamics to be described by the LMN. Two approaches exist [16]:

- The first option is to describe the global dynamics of the unknown system. In this case, we optimise all the parameters of the local models that make up the LMN jointly and at once using all the available training data.
- Another option is to describe the local dynamics by optimising the parameters of the individual local models of the LMN. We use only the data describing the relevant domain.

The optimisation process of the global dynamics usually leads to a globally better description of the system, as the free parameters of the local models of nonequilibrium regions can be adjusted to increase the validity of the local models over a wider range. The implication of better global behaviour is the loss of information about the local dynamics, since the parameters of the local models no longer reflect it [8], [9], [16]. In contrast, if the local models are optimised in the second way (i.e., locally), they represent the local dynamics and are thus more transparent and useful for the analysis and design of the control system. However, at the same time, local models are only valid for the domain in which they have been identified, and this is usually small. Consequently, some areas of system operation may remain unmodelled, leading to the problem of modelling regions away from the equilibrium curve.

The problem of some regions not being modelled can also be avoided by using a smaller vector of *scheduling variables* [8]. Scheduling variables are those that determine the position of a local model in the nonlinear hyperspace. The smaller the vector of scheduling variables or the shorter the scheduling vector, the larger the range of validity of the local models (Figure 5.11), but there is a problem with the blending of the local models in areas 'far' from the equilibrium curve. The blending can become uneven or even discontinuous (Figures 5.12 and 5.13), which can also lead to unstable control loops when used for control design [3].

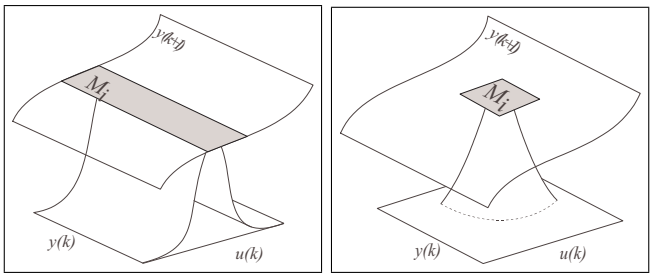

Figure 5.11: The interpretation of the vector of scheduling variables

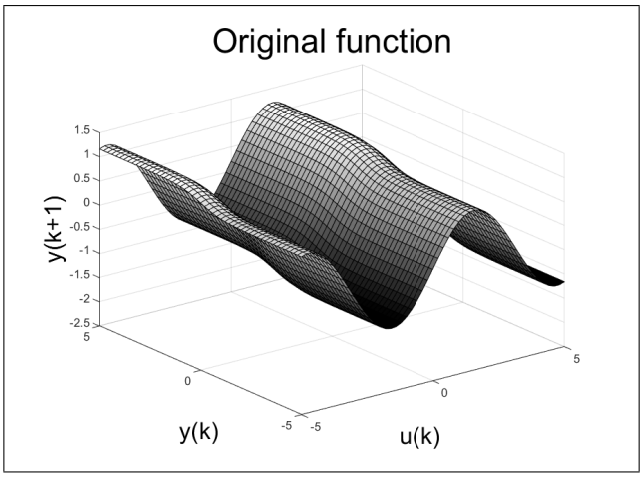

Figure 5.12: Original system [2]

5.1 Velocity-based linearisation

In the previous section, we mentioned that the Taylor expansion is not an adequate method to obtain a suit-

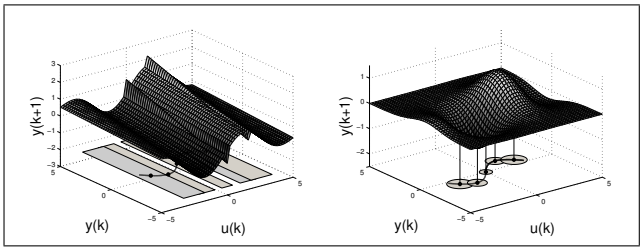

Figure 5.13: Model with a reduced and complete vector of scheduling variables [2]

able linear partial model. Better suited is its derivative: velocity-based linearisation.

The velocity-based linearisation is a generalisation of the usual linearisation around the operating point: the local linear system belongs to each operating point of the whole space of the operation of the nonlinear system (Figure 5.14) and not only to the equilibrium points. The theory of velocity-based linearisation is discussed in [10] and [11], and the demonstration software can be found in [13].

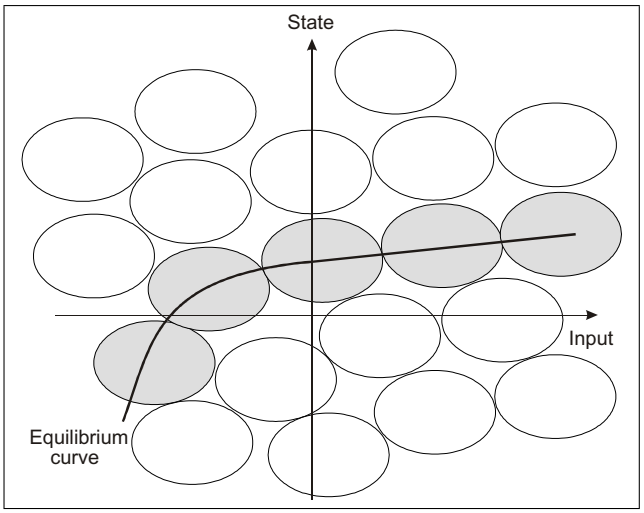

Figure 5.14: Validity regions of local models

The velocity-based linearisation at any point, which may be arbitrarily far from equilibrium given the value of the scheduling vector, is equal to the velocity-based linearisation at one of the equilibrium points determined by the same value of the scheduling vector. This fact is shown in Figure 5.15.

Velocity-based linearisation

Let us look at how a known nonlinear system is described by a velocity-based linearisation.

A nonlinear dynamic system is described by a system of nonlinear differential equations

$$\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x}, \mathbf{u})$$

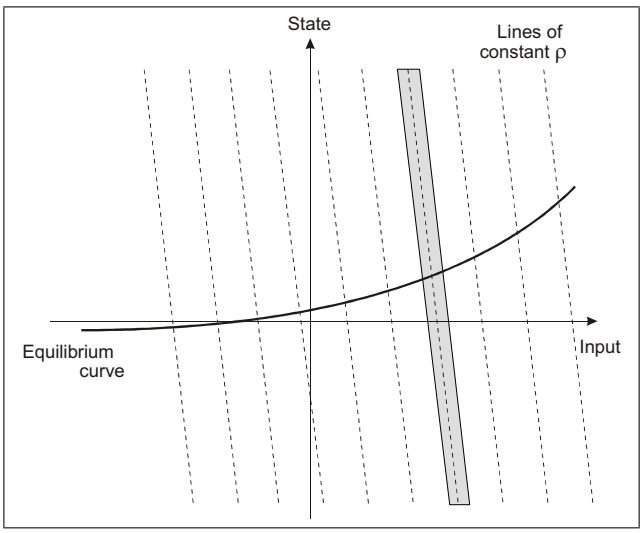

Figure 5.15: The effect of velocity-based linearisation out of equilibrium

$$\dot{\mathbf{y}} = \mathbf{G}(\mathbf{x}, \mathbf{u}), \tag{5.3}$$

where $\mathbf{x} \in \mathcal{R}^n$, $\mathbf{u} \in \mathcal{R}^m$, $\mathbf{y}$ is the vector of outputs, and $\mathbf{F}, \mathbf{G}$ are matrices of function dependences.

This system can equivalently be described in the form in which the linear and nonlinear parts have been separated:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{f}(\boldsymbol{\rho}) \\ \mathbf{y} &= \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} + \mathbf{g}(\boldsymbol{\rho}) \\ \boldsymbol{\rho} &= \boldsymbol{\rho}(\mathbf{x}, \mathbf{u}),\ \text{z}\ \nabla_{\mathbf{x}}\boldsymbol{\rho},\ \text{constants}, \end{aligned} \tag{5.4}$$

where $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ are correspondingly large constant matrices. The nonlinear functions $\mathbf{f}(\cdot)$ and $\mathbf{g}(\cdot)$ depend on the vector of scheduling variables $\boldsymbol{\rho} = \boldsymbol{\rho}(\mathbf{x}, \mathbf{u}) \in \mathcal{R}^q$, $q \leq m+n$, which expresses the nonlinear dependence of the dynamics of the system on its state and input with constants $\nabla_x \boldsymbol{\rho}$, $\nabla_u \boldsymbol{\rho}$. When this system is differentiated, we obtain

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{w} \\ \dot{\mathbf{w}} &= (\mathbf{A} + \nabla\mathbf{f}(\boldsymbol{\rho})\nabla_{\mathbf{x}}\boldsymbol{\rho})\mathbf{w} + (\mathbf{B} + \nabla\mathbf{f}(\boldsymbol{\rho})\nabla_{\mathbf{u}}\boldsymbol{\rho})\dot{\mathbf{u}} \\ \dot{\mathbf{y}} &= (\mathbf{C} + \nabla\mathbf{g}(\boldsymbol{\rho})\nabla_{\mathbf{x}}\boldsymbol{\rho})\mathbf{w} + (\mathbf{D} + \nabla\mathbf{g}(\boldsymbol{\rho})\nabla_{\mathbf{u}}\boldsymbol{\rho})\dot{\mathbf{u}}. \end{aligned}$$

If a result is 'frozen' at a value of the scheduling vector $\boldsymbol{\rho}_1$, we obtain a form based on the velocity-based linearisation at that particular point, which is a linear dynamic system:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{w} \\ \dot{\mathbf{w}} &= (\mathbf{A} + \nabla\mathbf{f}(\boldsymbol{\rho}_1)\nabla_{\mathbf{x}}\boldsymbol{\rho})\mathbf{w} + (\mathbf{B} + \nabla\mathbf{f}(\boldsymbol{\rho}_1)\nabla_{\mathbf{u}}\boldsymbol{\rho})\dot{\mathbf{u}} \\ \dot{\mathbf{y}} &= (\mathbf{C} + \nabla\mathbf{g}(\boldsymbol{\rho}_1)\nabla_{\mathbf{x}}\boldsymbol{\rho})\mathbf{w} + (\mathbf{D} + \nabla\mathbf{g}(\boldsymbol{\rho}_1)\nabla_{\mathbf{u}}\boldsymbol{\rho})\dot{\mathbf{u}}. \end{aligned}$$

The basic features of velocity-based linearisation can be summarised in the following points:

- Velocity-based linearisation is associated with each operating point (both equilibrium and nonequilibrium).

- The properties of the model obtained by velocity-based linearisation are a completely accurate representation of the properties of a nonlinear system in the immediate vicinity of the relevant operating point.
- The family of models obtained by velocity-based linearisation around *all* operating points has no loss of information with respect to the nonlinear dynamics and is an alternative representation of the nonlinear system.

Velocity-based linearisation addresses many of the shortcomings of ordinary linearisation theory around equilibrium points in terms of analysis and control design:

- it does not contain a constraint on the action around equilibrium points;
- it provides a direct link between the solution obtained in the form of a velocity-based linearisation and the local solution of the nonlinear system;
- solutions obtained by velocity-based linearisation can be composed into a global solution of the nonlinear system;
- no prior knowledge of equilibrium points is required.

It should be noted that the method raises new problems, such as the derivation of the input signal, but these are relatively easy to solve in both analysis and system design.

Modelling example

Let us look at the velocity-based linearisation for a damped pendulum [12] (Figure 5.16), which can be described by the equation

$$\ddot{\theta} = Q\dot{\theta} - Q\sin\theta + bF, \tag{5.5}$$

where $Q = 29.46$ and $b = 1.21$, $\theta \in [0, \pi]$. In a state space of the form (5.4), where linear and nonlinear parts are separated, the notation of the system is as follows:

$$\begin{aligned}\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ 0 & -Q \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ b \end{bmatrix} u \\ &\quad + \begin{bmatrix} 0 \\ -Q\sin(x_1) \end{bmatrix} \\ y &= \begin{bmatrix} 1 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \end{aligned} \tag{5.6}$$

where the state x_1 is θ, the state x_2 is $\dot{\theta}$ and the input u is F.

The system of equations presented in the form of a velocity-based linearisation is as follows:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

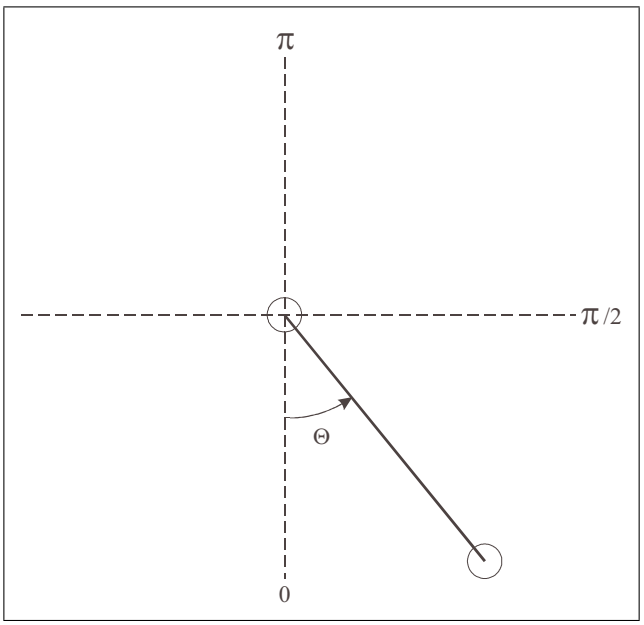

Figure 5.16: Schematic of a pendulum

$$\begin{aligned}\begin{bmatrix} \dot{w}_1 \\ \dot{w}_2 \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ -Q\cos(x_1) & -Q \end{bmatrix}\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}\dot{u} \\ \dot{y} &= \begin{bmatrix} 1 & 0 \end{bmatrix}\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}. \end{aligned} \tag{5.7}$$

The linearisation, for example, at the point (x_{1_0}, x_{2_0}, u_0) is obtained by 'freezing' the equation at this point. In our case, it is sufficient to insert only the value x_{1_0} into the equation. In this way, we obtain from the system of equations (5.7) a completely linear system at the chosen point and not an affine system as we would obtain using the Taylor expansion on the system described with Equations (5.6).

If we simulate the model in terms of a velocity-based linearisation for analysis purposes, we need $\dot{y}$ integrated, and $\dot{u}$ can be obtained via a filter that is the implementation of the derivative. How we get rid of the derivative, which mainly comes into play in control design, is discussed in the next section. Figure 5.17 shows the correspondence between the two system notations. The small discrepancy is due to the approximation of the derivation and other numerical reasons.

* * *

Velocity-based linearisation can also be applied to discrete-time systems, although discretisation should not be used for the derivatives and integrals required to implement velocity-based linearisation, as the error is accumulated by integrating the output signal.

Let us imagine a nonlinear dynamic system represented by sampled data in the form of a discrete-time system:

$$\begin{aligned}\mathbf{x}(k+1) &= \mathbf{F}\left(\mathbf{x}(k), \mathbf{u}(k)\right) \\ \mathbf{y}(k) &= \mathbf{G}\left(\mathbf{x}(k), \mathbf{u}(k)\right), \end{aligned} \tag{5.8}$$

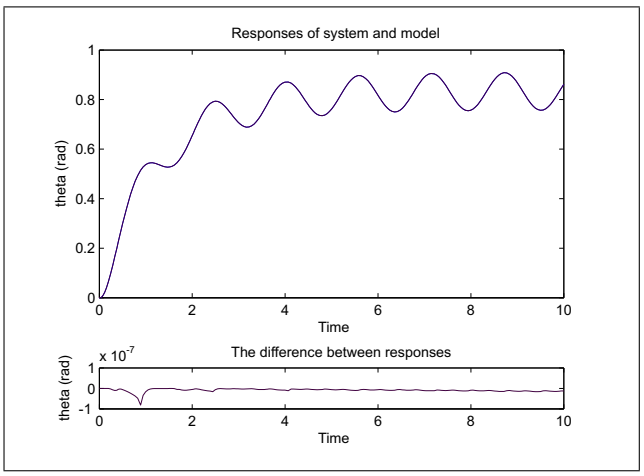

Figure 5.17: Response of the model and the pendulum, which are very close, as shown in the bottom figure, which illustrates the difference between the two responses

where the sampling time T is chosen small enough that the description (5.8) captures all the nonlinear dynamics of interest in the original system. We want to describe this system by a linear model with variable parameters.

In order to be able to use a velocity-based linearisation, the discrete-time system (5.8) is represented in the continuous-time domain in terms of a delayed continuous-time signal notation:

$$\begin{aligned} \mathbf{x}(t+T) &= \mathbf{F}\left(\mathbf{x}(t), \mathbf{u}(t)\right) \\ \mathbf{y}(t) &= \mathbf{G}\left(\mathbf{x}(t), \mathbf{u}(t)\right), \end{aligned} \tag{5.9}$$

where the values of the states $\mathbf{x}$, outputs $\mathbf{y}$ and inputs $\mathbf{u}$ of the discrete-time and continuous-time systems at sampling times $t = kT, \quad k = 1, 2, \ldots$, are equal.

System (5.9) can be expressed in terms of an extended local linear equivalence ELLE [10]:

$$\begin{aligned} \mathbf{x}(t+T) &= \tilde{\mathbf{A}}\mathbf{x}(t) + \tilde{\mathbf{B}}\mathbf{u}(t) + \mathbf{f}(\boldsymbol{\rho}), \\ y(t) &= \tilde{\mathbf{C}}\mathbf{x}(t) + \tilde{\mathbf{D}}\mathbf{u}(t) + \mathbf{g}(\boldsymbol{\rho}), \end{aligned} \tag{5.10}$$

where $\mathbf{x}(t) \in \mathcal{R}^n$, $\mathbf{u}(t) \in \mathcal{R}^m$ and $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}, \tilde{\mathbf{D}}$ are correspondingly large constant matrices. The nonlinear functions $\mathbf{f}(\cdot)$ and $\mathbf{g}(\cdot)$ depend on a vector of scheduling variables $\boldsymbol{\rho} = \boldsymbol{\rho}(\mathbf{x}(t), \mathbf{u}(t)) \in \mathcal{R}^q$, $q \leq m + n$, which expresses the nonlinear dependence of the dynamics of the system on its state and input with constant $\nabla_x \boldsymbol{\rho}$, $\nabla_u \boldsymbol{\rho}$ [10].

The first equation of (5.10), relative to the operating point $(\mathbf{x}, \mathbf{u}) = (\mathbf{x}_0, \mathbf{u}_0)$, can be written using

$$\begin{aligned} \delta\mathbf{x}(t) &= \mathbf{x}(t) - \mathbf{x}_0, \\ \delta\mathbf{u}(t) &= \mathbf{u}(t) - \mathbf{u}_0, \end{aligned}$$

as:

$$\mathbf{x}(t+T) = \tilde{\mathbf{A}}\left(\mathbf{x}_0 + \delta\mathbf{x}(t)\right) + \tilde{\mathbf{B}}\left(\mathbf{u}_0 + \delta\mathbf{u}(t)\right) + \mathbf{f}(\boldsymbol{\rho}). \tag{5.11}$$

Assuming local linearity near the operating point $(\mathbf{x}_0, \mathbf{u}_0)$ follows:

$$\begin{aligned} \mathbf{x}(t+T) - \mathbf{x}_0 + \mathbf{x}_0 &= \tilde{\mathbf{A}}\mathbf{x}_0 + \tilde{\mathbf{A}}\delta\mathbf{x}(t) + \tilde{\mathbf{B}}\mathbf{u}_0 + \tilde{\mathbf{B}}\delta\mathbf{u}(t) \\ &+ \mathbf{f}_0 + \mathbf{f}_{x0}\delta\mathbf{x}(t) + \mathbf{f}_{u0}\delta\mathbf{u}(t), \end{aligned} \tag{5.12}$$

$$\begin{aligned} \delta\mathbf{x}(t+T) + \mathbf{x}_0 &= \left(\tilde{\mathbf{A}}\mathbf{x}_0 + \tilde{\mathbf{B}}\mathbf{u}_0 + \mathbf{f}_0\right) \\ &+ \left(\tilde{\mathbf{A}} + \mathbf{f}_{x0}\right)\delta\mathbf{x}(t) + \left(\tilde{\mathbf{B}}\right. \\ &+ \left.\mathbf{f}_{x0}\right)\delta\mathbf{u}(t), \end{aligned} \tag{5.13}$$

$$\begin{aligned} \delta\mathbf{x}(t+T) &= (\mathbf{F}_0 - \mathbf{x}_0) + \left(\tilde{\mathbf{A}} + \mathbf{f}_{x0}\right)\delta\mathbf{x}(t) \\ &+ \left(\tilde{\mathbf{B}} + \mathbf{f}_{u0}\right)\delta\mathbf{u}(t), \end{aligned} \tag{5.14}$$

where $\mathbf{F}_0 = \mathbf{F}(\mathbf{x}_0, \mathbf{u}_0)$, $\mathbf{f}_0 = \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0)$, $\mathbf{f}_{x0} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_0, \mathbf{u}_0)$ and $\mathbf{f}_{u0} = \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(\mathbf{x}_0, \mathbf{u}_0)$.

Differentiating Equation (5.14) over time, we obtain:

$$\dot{\mathbf{x}}(t+T) = \left(\tilde{\mathbf{A}} + \mathbf{f}_{x0}\right)\dot{\mathbf{x}}(t) + \left(\tilde{\mathbf{B}} + \mathbf{f}_{u0}\right)\dot{\mathbf{u}}(t) \tag{5.15}$$

$$\dot{\mathbf{x}}(t+T) = \mathbf{A}(\boldsymbol{\rho})\dot{\mathbf{x}}(t) + \mathbf{B}(\boldsymbol{\rho})\dot{\mathbf{u}}(t), \tag{5.16}$$

where $\mathbf{A}(\boldsymbol{\rho}) = (\tilde{\mathbf{A}} + \mathbf{f}_{x0})$ and $\mathbf{B}(\boldsymbol{\rho}) = (\tilde{\mathbf{B}} + \mathbf{f}_{u0})$. Under the same conditions, the second equation of (5.10) can be written as:

$$\dot{y}(t) = \mathbf{C}(\boldsymbol{\rho})\dot{\mathbf{x}}(t) + \mathbf{D}(\boldsymbol{\rho})\dot{\mathbf{u}}(t), \tag{5.17}$$

where $\mathbf{C}(\boldsymbol{\rho}) = (\tilde{\mathbf{C}} + \mathbf{g}_{x0})$ in $\mathbf{D}(\boldsymbol{\rho}) = (\tilde{\mathbf{D}} + \mathbf{g}_{u0})$, z $\mathbf{g}_{x0} = \frac{\partial \mathbf{g}}{\partial \mathbf{x}}(\mathbf{x}_0, \mathbf{u}_0)$ in $\mathbf{g}_{u0} = \frac{\partial \mathbf{g}}{\partial \mathbf{u}}(\mathbf{x}_0, \mathbf{u}_0)$.

In the way described above, we can represent the operation of the system at any point defined by the vector of scheduling variables $\boldsymbol{\rho}$, with a vector of states $\mathbf{x}(t)$ and input $\mathbf{u}(t)$. System (5.10) can thus be written:

$$\begin{aligned} \dot{\mathbf{x}}(t+T) &= \mathbf{A}(\boldsymbol{\rho})\dot{\mathbf{x}}(t) + \mathbf{B}(\boldsymbol{\rho})\dot{\mathbf{u}}(t) \\ \dot{\mathbf{y}}(t) &= \mathbf{C}(\boldsymbol{\rho})\dot{\mathbf{x}}(t) + \mathbf{D}(\boldsymbol{\rho})\dot{\mathbf{u}}(t), \end{aligned}$$

where the matrices are

$$\begin{aligned} \mathbf{A}(\boldsymbol{\rho}) &= \tilde{\mathbf{A}} + \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}(t), \mathbf{u}(t)), \\ \mathbf{B}(\boldsymbol{\rho}) &= \tilde{\mathbf{B}} + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(\mathbf{x}(t), \mathbf{u}(t)), \\ \mathbf{C}(\boldsymbol{\rho}) &= \tilde{\mathbf{C}} + \frac{\partial \mathbf{g}}{\partial \mathbf{x}}(\mathbf{x}(t), \mathbf{u}(t)), \\ \mathbf{D}(\boldsymbol{\rho}) &= \tilde{\mathbf{D}} + \frac{\partial \mathbf{g}}{\partial \mathbf{u}}(\mathbf{x}(t), \mathbf{u}(t)), \end{aligned}$$

depending on the vector of scheduling variables $\boldsymbol{\rho}$.

Let us consider an example of modelling a discrete-time system with velocity-based linearisation.

Example of modelling a discrete-time system

We are interested in a velocity-based linearisation for a nonlinear discrete-time system [17], described by the equation

$$y(k+1) = \frac{y(k)}{1+y^2(k)} + u(k)^3 \tag{5.18}$$

and with sampling time $T = 1$ s. In the state space, the system (5.18) can be written as:

$$\begin{aligned} x(k+1) &= \frac{x(k)}{1+x^2(k)} + u(k)^3 \\ y(k) &= x(k). \end{aligned} \tag{5.19}$$

The representation of the system (5.19) in the continuous-time domain is

$$\begin{aligned} x(t+T) &= \frac{x(t)}{1+x^2(t)} + u(t)^3 \\ y(t) &= x(t), \end{aligned} \tag{5.20}$$

which is not an equivalent system but has the same response at the moments of sampling. If we were to discretise the system (5.20) with the sampling time T, we would obtain the system (5.18).

The system (5.20) in the form of a velocity-based linearisation is

$$\begin{aligned} \dot{x}(t+T) &= (1-x(t)^2)/(1+x(t)^2)^2\dot{x}(t) + 3u(t)^2\dot{u}(t) \\ \dot{y}(t) &= \dot{x}(t). \end{aligned} \tag{5.21}$$

Figure 5.18 shows the matching responses of the systems (5.18) and (5.21). The difference between the responses of

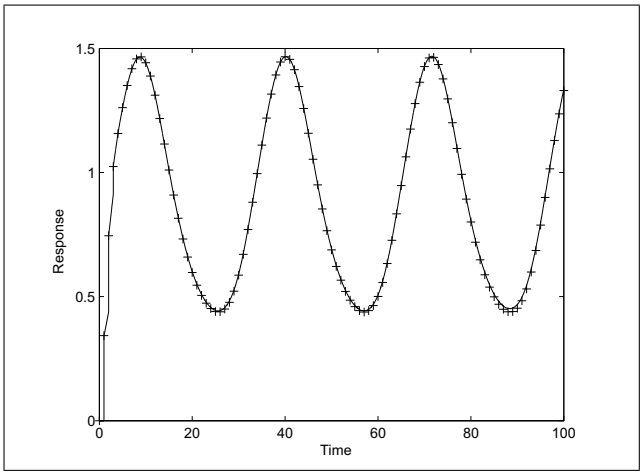

Figure 5.18: Response of the discrete-time system and the continuous-time model in the velocity-based form

the two systems can be seen if observed carefully. The difference is due to the larger approximations, but it is much smaller than if we tried to replace the derivatives and integrals of the velocity-based linearisation implementation with discrete-time equivalents or an incremental discrete-time model.

5.2 Blended multiple-model systems

The velocity-based linearisation method as described is a useful tool for linearising nonlinear systems with an infinitely large family of linearised submodels. In practice, we would also like to have a finite number of parameterised models. A velocity-based linearisation for when a nonlinear system is modelled by blending a finite number of linear submodels is described in detail in [12].

We select a small number of representative family members and interpolate between them to replace the missing ones: the (approximate) finite parametrisation, which is essentially a new form of blended multiple-model representation. In contrast to the usual blended systems (local-model network), velocity-based linearised systems have the following advantages:

- they use linear local models - true linear, not affine;
- they establish a direct link between the dynamics of the local models and the dynamics of the overall blended system;
  - the behaviour and properties of the whole system are approximated locally by the behaviour and properties of a weighted linear combination of local models;
  - better still, the behaviour of the whole system is approximated locally by the weighted combination of the behaviour and properties of the local models.

Example of damped pendulum modelling

We will approximate a nonlinear system (5.5) and a family of velocity-based linearised models by blending local models at angles 0, $\pi/2$ and π. We will use Gaussian weighting functions denoted by μ, as in Figure 5.19.

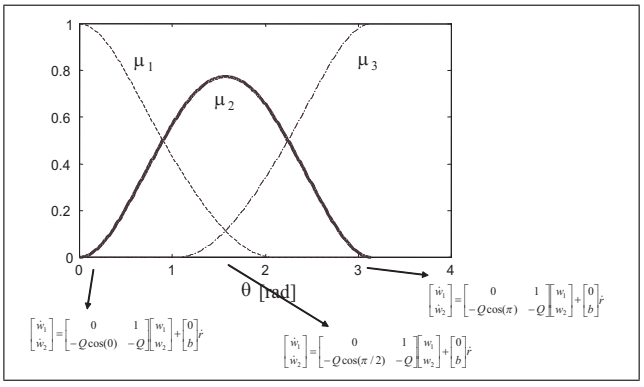

Figure 5.19: Weighting functions

The model based on the velocity-based linearisation at intermediate operating point of the approximated system is obtained by blending the velocity-based linearised models at the operating points corresponding to angles 0, $\pi/2$ and π.

1.

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{w}\\ \dot{\mathbf{w}} &= (\mathbf{A}+\nabla\mathbf{f}(\rho_1)\nabla_{\mathbf{x}}\rho)\mathbf{w}+(\mathbf{B}+\nabla\mathbf{f}(\rho_1)\nabla_u\rho)\dot{u},\end{aligned}$$

2.

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{w}\\ \dot{\mathbf{w}} &= (\mathbf{A}+\nabla\mathbf{f}(\rho_2)\nabla_{\mathbf{x}}\rho)\mathbf{w}+(\mathbf{B}+\nabla\mathbf{f}(\rho_2)\nabla_u\rho)\dot{u},\end{aligned}$$

3.

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{w}\\ \dot{\mathbf{w}} &= (\mathbf{A}+\nabla\mathbf{f}(\rho_3)\nabla_{\mathbf{x}}\rho)\mathbf{w}+(\mathbf{B}+\nabla\mathbf{f}(\rho_3)\nabla_u\rho)\dot{u}.\end{aligned}$$

Weighted combination of solutions:

$$\tilde{\mathbf{w}} = \sum_{i=1}^{3} w_i\mu_i(\rho).$$

$$\begin{array}{cl} & \text{solution of weighted combination}\\ = & \text{solution of velocity-based linearisation}\\ \approx & \text{solution for the nonlinear system (localy}\\ & \text{to the relevant operating point)}\end{array}$$

It is easy to see how few linearised systems (local models) have been taken. Only three local models are sufficient to cover the entire operating range $[0, \pi]$:

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{w}\\ \dot{\mathbf{w}} &= \sum_{i=1}^{3}\left[(\mathbf{A}+\nabla\mathbf{f}(\rho_i)\nabla_{\mathbf{x}}\rho)\mathbf{w}\right.\\ &+ \left.(\mathbf{B}+\nabla\mathbf{f}(\rho_i)\nabla_u\rho)\dot{u}\right]\mu_i(\rho).\end{aligned}$$

We can show that:

$$\begin{aligned}\tilde{\mathbf{w}} &\approx \mathbf{w}, \text{ solution of weighted combination}\\ &\approx \text{solution for the nonlinear system (locally)}\end{aligned}$$

This is a direct link between the dynamics of the local models and the dynamics of the nonlinear blended system and a consequence of the use of the velocity-based realisation and does not apply to the usual local-model networks.

The correspondence between the responses of the original and the blended model is shown by Figures 5.20, 5.21, and 5.22.

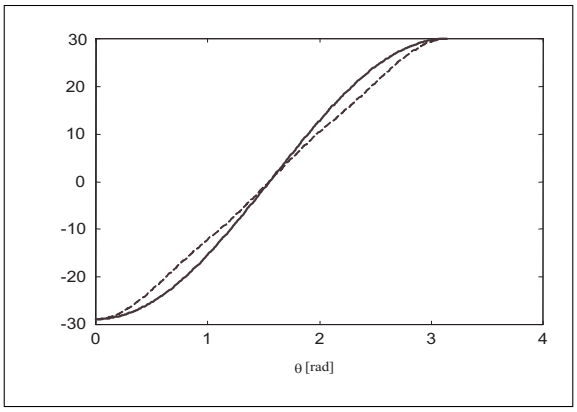

Figure 5.20: Fitting the nonlinearity of the blended model ($\dot{w}_2$) based on the velocity-based linearisation and the nonlinearity of the original system: full curve - nonlinear system, dashed curve - blended model

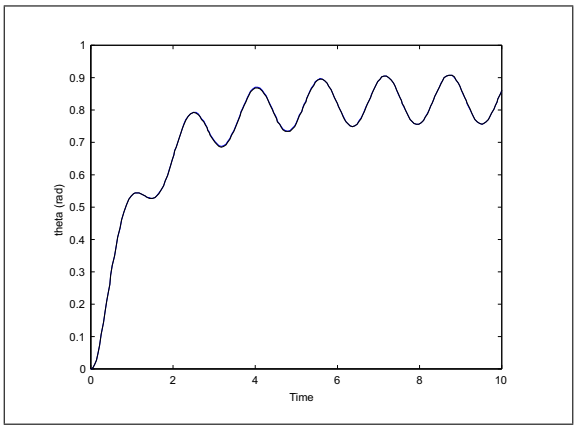

Figure 5.21: Response of a nonlinear system and the blended model to a given input signal - the working range is about $\pi/4$ rad, which is the most problematic range in terms of blending

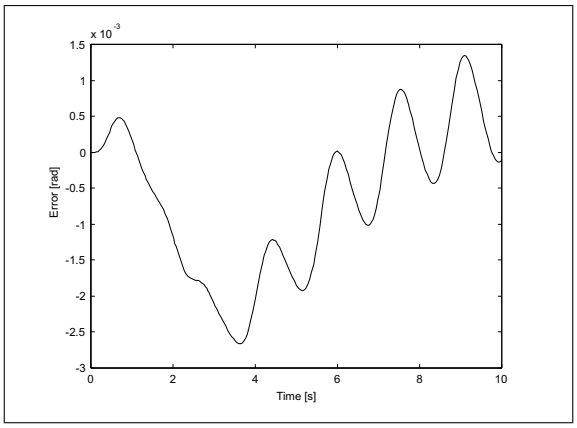

Figure 5.22: The difference between the output of the nonlinear system and the blended model

Example of modelling a discrete-time system

The same procedure as in the previous example can be used to model a discrete-time system. The system (5.18) was approximated in its continuous form (5.20) with seven uniformly distributed local models, which were blended with Gaussian weighting functions.

The correspondence between the responses of the original system and the blended model is shown in Figure 5.23.

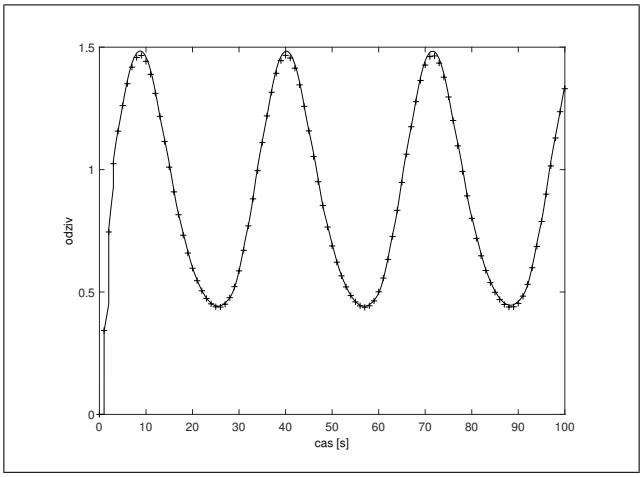

Figure 5.23: Response of a nonlinear discrete-time system (5.18) and the response of the blended model (5.22)

Bibliography

[1] R. Babuška (1998): Fuzzy modeling for control, Norwell, MA: Kluwer.

[2] G. Gregorčič (2004): Data-based modelling of nonlinear systems for control, Ph.D. Thesis, University College Cork, National University of Ireland.

[3] G. Gregorčič, G. Lightbody (2003): From multiple model networks to the Gaussian processes prior model, Proceedings in IFAC ICONS conference, Faro, 149-154.

[4] T. A. Johansen, B. A. Foss (1993): Constructing NARMAX models using ARMAX models, Int. J. Control, Vol. 58, 1125-1153.

[5] T. A. Johansen, B. A. Foss (1998): ORBIT - operating regime based modeling and identification toolkit, Control Engineering Practice, Vol. 6, 1277-1286.

[6] T. A. Johansen, K. Hunt and H. Fritz (1998): A software environment for gain scheduled local controller network design, IEEE Control Systems Maganzine, 48-60.

[7] T. A. Johansen, B. A. Foss [Eds.] (1999): Int. J. Control, Special issue: Multiple model approaches to modelling and control, Vol. 72, No. 7/8.

[8] T. A. Johansen, R. Shorten and R. Murray-Smith (2000): On the interpretation and identification of dynamic Takagi-Sugeno fuzzy models, IEEE Trans. on Fuzzy Systems, Vol. 8, No. 3, 297-313.

[9] T. A. Johansen,R. Babuška (2003): Multiobjective identification of Takagi-Sugeno fuzzy models, IEEE Trans. Fuzzy Systems, Vol. 11, No. 6, 847-860.

[10] D. J. Leith, W. E. Leithead (1998): Gain-Scheduled & Nonlinear Systems: Dynamic analysis by velocity-based linearisation families. Int. J. Control, Vol. 70, 289-317.

[11] D. J. Leith, W. E. Leithead (1998): Gain-scheduled controller design: an analytic framework directly incorporating non-equilibrium plant dynamics, Int. J. Control, Vol. 70, 249-269.

[12] D. J. Leith, W. E. Leithead (1999): Analytic framework for blended multiple model systems using linear local models, Int. J. Control, Vol. 72, 605-619.

[13] D. J. Leith, W. E. Leithead (2007): Velocity-based gain-scheduling demo, `https://www.hamilton.ie/doug/demof.htm`

[14] R. Li, Z. Zhao, X.-B. Li (2005): General model-set design methods for multiple-model approach, IEEE Trans. Auto. Control, Vol. 50, No. 9, 1260-1276.

[15] R. Murray-Smith and T. A. Johansen [Eds.] (1997): Multiple model approaches to modelling and control, Taylor & Francis, London.

[16] R. Murray-Smith, T. A. Johansen, R. Shorten (1999): On transient dynamics, off-equilibrium behaviour and identification in blended multiple model structures, Proceedings in European Control Conference, Karslruhe, BA-14.

[17] K. S. Narendra, J. Balakrishnan, M. K. Ciliz (1995): Adaptation and learning using multiple models switching and tuning, IEEE Control System Magazine, Vol. 15, No. 3, 37-51.

[18] O. Nelles (2001): Nonlinear system identification, from classical approaches to neural networks and fuzzy Models, Springer Verlag, Berlin.

[19] T. Takagi and M. Sugeno (1985): Fuzzy identification of systems and its application to modeling and control, IEEE Trans. Syst., Man, Cybern., Vol. 15, 116Â–132.

Chapter 6

Design of gain-scheduling control

The dynamic systems that surround us predominantly exhibit nonlinear dynamics. Unfortunately, the methods for their analysis and design are still relatively undeveloped. In contrast, well-developed and well-known tools for linear dynamic systems exist. This means that the 'divide and conquer' approach, in which a nonlinear problem is divided into a series of linear problems, is attractive.

Gain-scheduling control [11] is a typical 'divide and conquer' control method used in many areas of automatic control: from aeronautics to process engineering. The conventional control design by gain scheduling consists of the following steps:

1. the linearisation of a nonlinear system around a selected number of equilibrium points;
2. the design of linear controllers for each of the linear submodels;
3. the integration of the linear controllers into a global nonlinear controller by scheduling the parameters or signals of the controllers using scheduling variables.

A conventional gain-scheduling controller is limited to operating near the equilibrium points. The scheduling variables used to schedule the parameters or signals of linear controllers must be slow in conventional gain-scheduling control. This means that their rate of change must be slower than the rate of change of the output signal of the closed-loop system. The theory supporting this very widely used approach is relatively poorly developed. For a more detailed consideration of this topic, see [9].

6.1 The design with velocity-based linearisation

Velocity-based linearisation, which we described in the previous chapter, solves a number of drawbacks of conventional gain scheduling. In particular, it generalises the control design by incorporating information about the dynamics of the system even at nonequilibrium points. As mentioned in the previous chapter, this form of a model is valid throughout the domain and is not limited to the vicinity of the equilibrium points. This means that we can also deal with fast and large-scale transitions between distant operating regions of the system. Gain-scheduling-control design based on a velocity-based linearisation looks like this (Figure 6.1):

1. the development of the family of velocity-based linearised local models for the nonlinear system;
2. the design of linear controllers for each of the linear local models;
3. the integration of a nonlinear controller from a family of velocity-based controllers designed in the previous step.

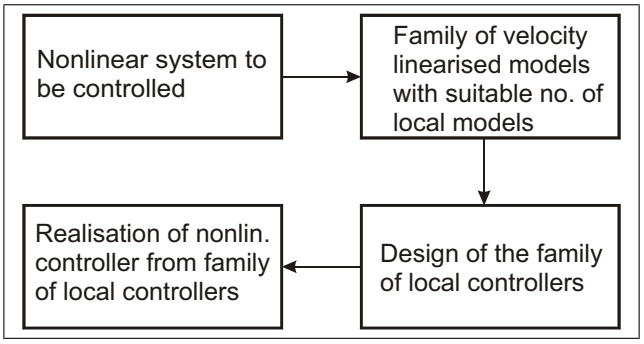

Figure 6.1: The method of gain-scheduling control design with velocity-based linearisation

The advantages of velocity-based linearisation for gain-scheduling control are as follows.

- The gain-scheduling controller designed with velocity-based linearisation works well even with large changes in the setpoint and at large distances from the equilibrium state.
- Such a controller does not require any of the signals to change slowly. This means that it can also handle the global inversion of the dynamics, which could be described in a simplified way as the direct cancellation of the poles and zeros of the controlled system at the operating points.

- The advantage of the design procedure is that the knowledge of the design of linear systems can be used.

For more details on velocity-based gain-scheduling control, see [6], [7], with the demonstration software at [10]. Examples of use are [2], [3], [4].

Implementation of controllers based on velocity-based linearisation

The form of the system using velocity-based linearisation (Figure 6.2) also contains an input-signal derivative instead of the input signal itself and an output signal derivative instead of the output signal itself. The input signal derivative is not desirable when implementing control, but it can be avoided.

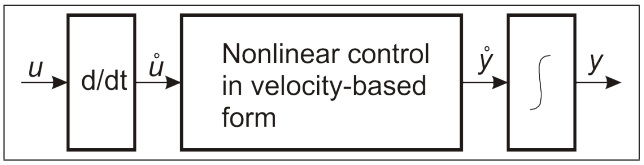

Figure 6.2: A system in the velocity-based linearisation form

We can consider the fact that almost all controllers also contain an integral part. In this case, the system can be configured as shown in Figure 6.3.

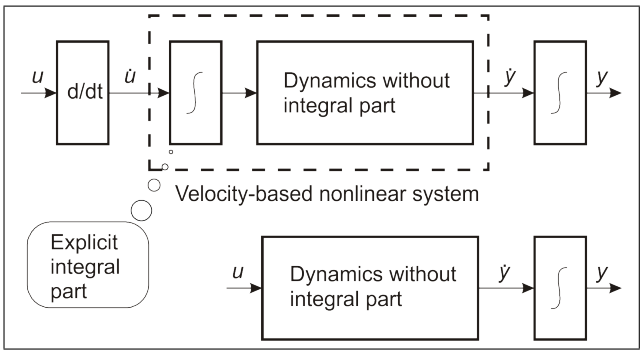

Figure 6.3: Block diagrams showing input signal derivation bypass

Example of PI controller

Let us illustrate the implementation for bypassing the signal derivation with a simple proportional-integral (PI) controller:

$$K(s,\rho) = (K_0(\rho) + \frac{K_1(\rho)}{s})\frac{1}{(s+50)}, \qquad (6.1)$$

where $K(s,\rho)$ is the transfer function of the PI controller, s is a complex variable, ρ is a scheduling variable, and $K_0, K_1 \in \mathcal{R}$ are controller parameters. Figure 6.4 shows the basic and undesirable form of implementation; Figure 6.5 shows how to avoid the derivative.

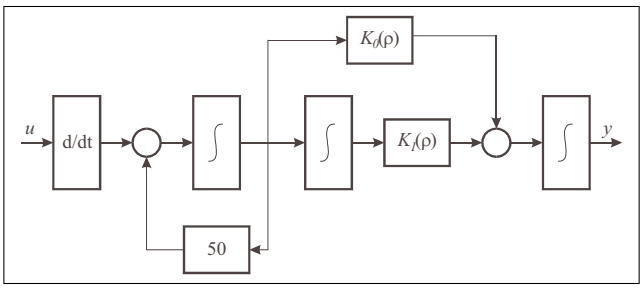

Figure 6.4: PI controller in velocity-based linearisation form that cannot be implemented

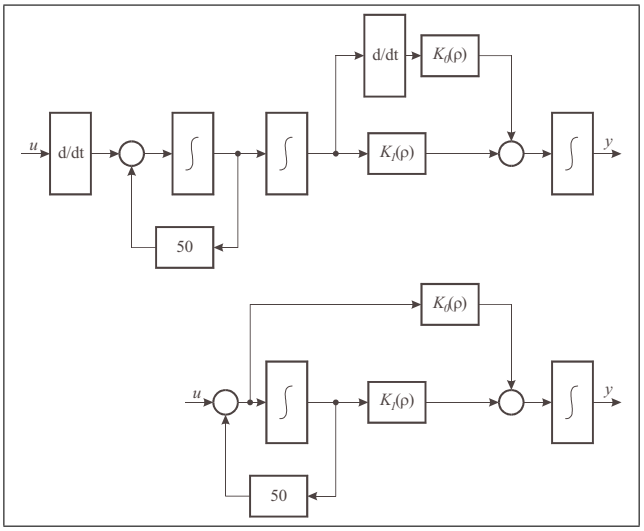

Figure 6.5: The final implementation of the PI controller

The implementation of the controller in the form of a simulation schematic of the package Matlab/Simulink for the values $K_0 = 11$ and $K_1 = 55$ is shown in Figure 6.6.

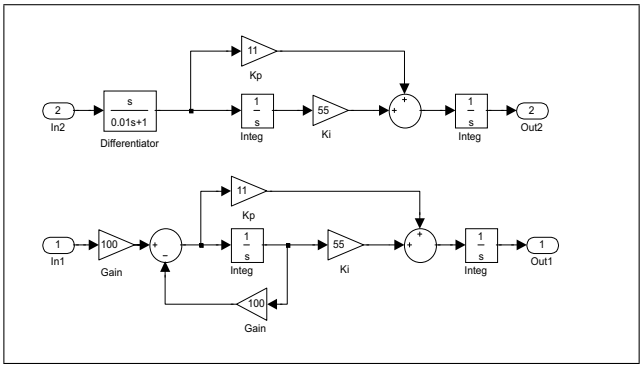

Figure 6.6: An example of the controller implementation of the controller $K(s,\rho_1) = (11 + \frac{55}{s})\frac{1}{(0.01s+1)}$ in Simulink

Example of two differently implemented gain-scheduling controllers

Let us consider the design of a controller in the velocity-based form and in the conventional gain-scheduling form for the control of the nonlinear system with input u and output y:

$$\dot{y} = \tanh(u - 10y) + 0.01(u - 10y). \qquad (6.2)$$

For both controllers, the nonlinearity of the controller parameters is the same, but they are implemented differently. The simulation schematics are shown in Figure 6.7 and the closed-loop responses are shown in Figure 6.8. From the responses, it can be seen that the conventional gain-scheduling control cannot keep up with large changes in the setpoint signal.

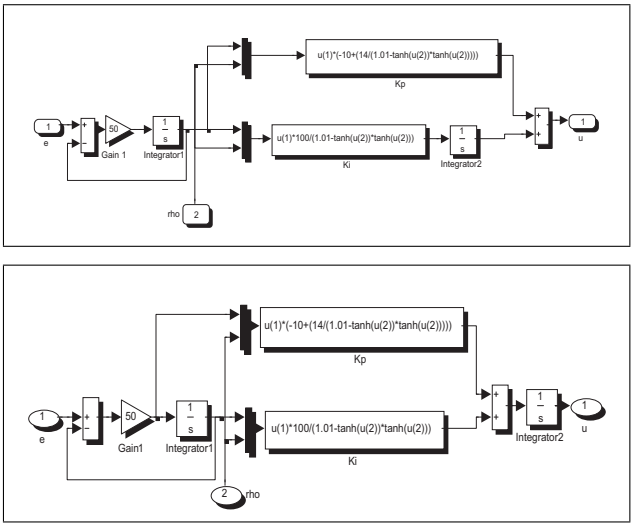

Figure 6.7: Simulink simulation schematic for the controller in the conventional form (top), and in the velocity-based form (bottom)

* * *

Thus far, we have considered families of velocity-based linearised models with an infinite number of links representing the modelled nonlinear system. In practice, we often encounter the desire to approximate a nonlinear system by a model consisting of a finite number of linear submodels.

Velocity-based linearisation also allows us to approximate a model by interpolating a small number of linearised local models. Furthermore, there is a direct relationship between the solutions of these 'local' control problems and the solution of the control problem for the nonlinear blended system.

On this basis, we can design the gain-scheduling control according to the following procedure:

- Design a linear controller for each of the linearisations of the nonlinear system, meaning for the local models that are represented by a small number of linear local models.
- Use the same type of blending functions as in the approximated system model for blending the family of controllers based on velocity-based linearisation.
- Provided that the blended system model for the controller design is sufficiently accurate and the controller is sufficiently robust, the family of controllers is also suitable for the control of the system. This condition applies to all design procedures using the system model. It should be noted that for a successful controller design, the system must be approximated by a larger number of models than a minimum number of local models that satisfactorily describe the nonlinear dynamics. The reason for this is that in the case of gain-scheduling we reproduce the inverse dynamics of the process to some extent.
- Integration of a global nonlinear controller from the designed family of local velocity-based controllers.

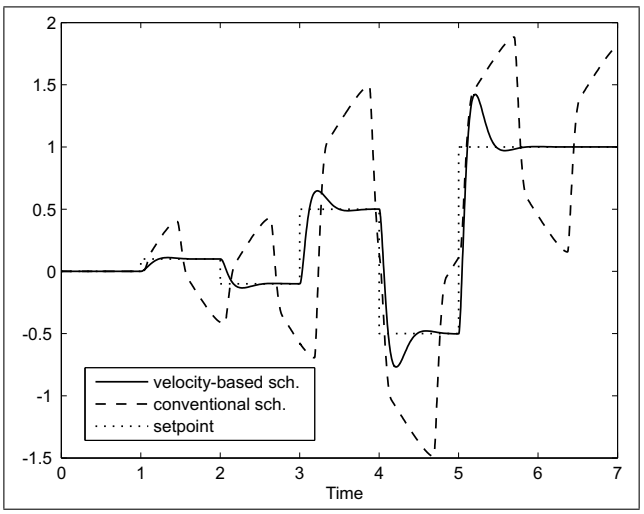

Figure 6.8: Responses in closed loop with different control implementations

Summary of lessons on gain-scheduling design based on a velocity-based linearisation:

- When we design a controller with gain scheduling, we must make a nonlinear controller with a well-defined family of local models obtained with velocity-based linearisation, which is relatively straightforward.
- The derivation of noisy input signals is not required.
- We can obtain a nonlinear controller directly connected to the family of linear controllers without the need for additional parameter tuning.
- The family of velocity-based linearised system models has an infinite number of elements, but we often

want to work with a finite and smaller set of elements. We can design controllers for a small number of linearised system models and then mix them with blending functions to create a global nonlinear controller for the entire nonlinear system.

- When implementing controllers that are based on velocity-based linearisation, care must be taken to preserve the advantages obtained.
- Blending (i.e., interpolation) depends on the properties of the nonlinear system.
- Conventionally implemented blended controllers have greater deviations from the desired behaviour with increasing distance from the equilibrium points.
- The direct relationship between properties of velocity-based linearised models and properties of the blended nonlinear model/controller greatly facilitates analysis and design. This is a peculiarity of velocity-based linearisation and does not apply to the usual forms of systems with multiple models (local-model networks, fuzzy models, etc.).
- It should be borne in mind that the successful implementation of control depends on the correct choice of the scheduling variable and the number of blending functions.

6.2 Example of control design: a single-segment robot manipulator

The purpose of this example, described in more detail in [1], is to illustrate the process of designing a control with gain-scheduling based on the final number of linearised submodels and to show the importance of correct implementation.

We illustrate the control design process and evaluate it with a computer simulation using a single-segment robot manipulator as an example. A segment of the robot manipulator is approximated by a homogeneous rod as shown in Figure 6.9:

$$\begin{aligned} M &= J\ddot{\Phi} + mg\sin\Phi\frac{r}{2} + k_v\dot{\Phi} \\ y &= r(1-\cos\Phi), \end{aligned} \tag{6.3}$$

where y is the vertical deviation of the rod tip, Φ is the angular deviation, M is the torque, J is the momentum, m is the mass of the rod, g is the acceleration due to gravity, r is the length of the rod, and k_v is the viscous friction coefficient.

This is a nonlinear system, which is unlikely to pose any major problems for the design of the controller, regardless of the design method used.

The design process is described in the following steps:

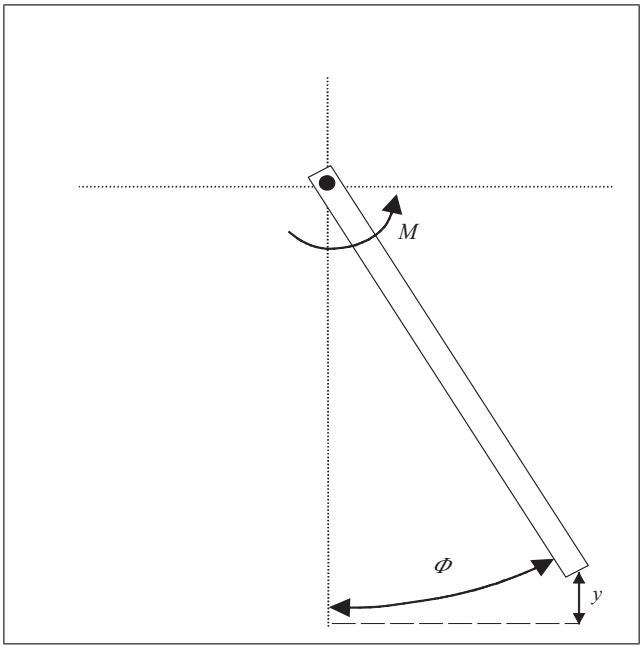

Figure 6.9: Schematic of a single-segment robot manipulator

- The system is represented by 3 local models at 3 operating points, $r = 0, 0.2$ and 0.7, corresponding to $\Phi = 0, 36$ and 72 angular degrees.
- For the local models, we design local PI controllers with the requirement that the closed-loop response is free from overshoot and is the same over the entire operating range.
- The local controllers are suitably blended into a nonlinear controller.
- Set the three blending functions in the form of a Gaussian bell curve.
- The output variable y (i.e., the vertical deviation of the rod tip) is the scheduling variable.
- A pole of the transfer function is added to attenuate high-frequency noise and deviation and to facilitate the implementation of the controller.
- We will compare two different designs of the blended controller.

Conventional implementation of the gain-scheduling controller

A conventional implementation of a blended controller with output u, input e, states $\mathbf{x}_c$ and blending functions μ_i looks like this:

$$\begin{aligned} \dot{\mathbf{x}}_{c_i} &= \sum_i \mathbf{F}_i(\mathbf{x}_c, e)\mu_i(\boldsymbol{\rho}) \\ u_i &= \sum_i \mathbf{G}_i(\mathbf{x}_c, e)\mu_i(\boldsymbol{\rho}) \end{aligned}$$

$$\boldsymbol{\rho}(\mathbf{x}_c, e) = \nabla_x \boldsymbol{\rho} \mathbf{x}_c + \nabla_e \boldsymbol{\rho} e, \tag{6.4}$$

where normally

$$\mathbf{F}_i(\mathbf{x}_c, e) = \alpha_i + \mathbf{A}_i \mathbf{x}_c + \mathbf{B}_i e, \tag{6.5}$$
$$\mathbf{G}_i(\mathbf{x}_c, e) = \beta_i + \mathbf{C}_i \mathbf{x}_c + \mathbf{D}_i e \tag{6.6}$$
$$\tag{6.7}$$

with the constants $\alpha_i, \mathbf{A}_i, \mathbf{B}_i, \beta_i, \mathbf{C}_i, \mathbf{D}_i$, which represents the linearisation of the nonlinear system at an operating point.

An example of a conventional implementation of a nonlinear PI controller is shown as a block diagram in Figure 6.10. The disadvantages of such an implementation can be summarised in two observations:

- Predictable behaviour exists only near the equilibrium points.
- The scheduling variable must be changed slowly.

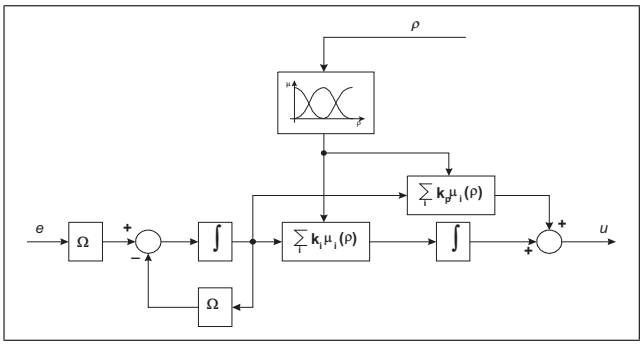

Figure 6.10: Conventional design of a blended PI controller

The scheduling variable is the output of the process y, which means that the scheduling variable does not change slowly. Implementation by blending control signals instead of parameter blending has the following disadvantages in addition to those already mentioned. The parallel dynamic controllers can increase the order of the nonlinear controller proportionally to the number of active controllers, which affects the dynamics of the closed loop. The exception is when the local controllers have identical poles when the controller parameters are blended, which is the case in our example. Therefore, closed-loop systems behave the same with conventionally implemented controllers regardless of whether output signals of controllers or controller parameters are blended.

Implementation of a gain-scheduling controller with velocity-based linearisation

The implementation of blended controllers with velocity-based linearisation proceeds as follows:

$$\begin{aligned}
\dot{\mathbf{x}}_c &= \mathbf{w}_c \\
\dot{\mathbf{w}}_c &= \left(\sum_i (\boldsymbol{\rho})\right) \mathbf{w}_c + \left(\sum_i \mathbf{B}_i \mu_i(\boldsymbol{\rho})\right) \dot{e} \\
\dot{u} &= \left(\sum_i \mathbf{C}_i \mu_i(\boldsymbol{\rho})\right) \boldsymbol{w}_c + \left(\sum_i \mathbf{D}_i \mu_i(\boldsymbol{\rho})\right) \dot{e}.
\end{aligned} \tag{6.8}$$

An example of a nonlinear PI controller in velocity-based form is shown in the block diagram in Figure 6.11. The disadvantage of the velocity-based linearisation implementation is that we have to be careful to implement it correctly.

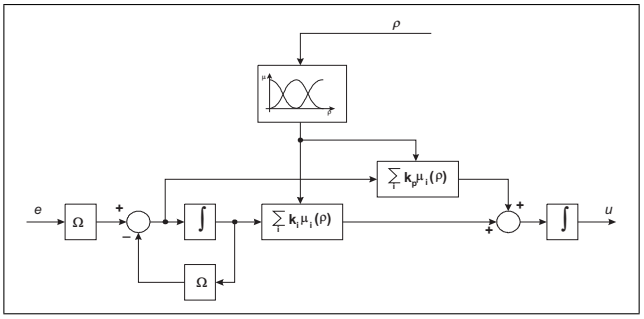

Figure 6.11: Implementation of blended controller with velocity-based linearisation

The filter at the input of the controller, which we exploited to avoid the derivative of the input signal, contains an additional pole Ω to attenuate high frequencies, which also limits the amount of noise in the signal.

A comparison of the closed-loop responses with differently implemented controllers is shown in Figure 6.12. It is clear that a closed-loop system with a conventionally implemented controller does not behave as it should with large changes in the setpoint, where the system quickly transitions from one area of nonlinearity to another.

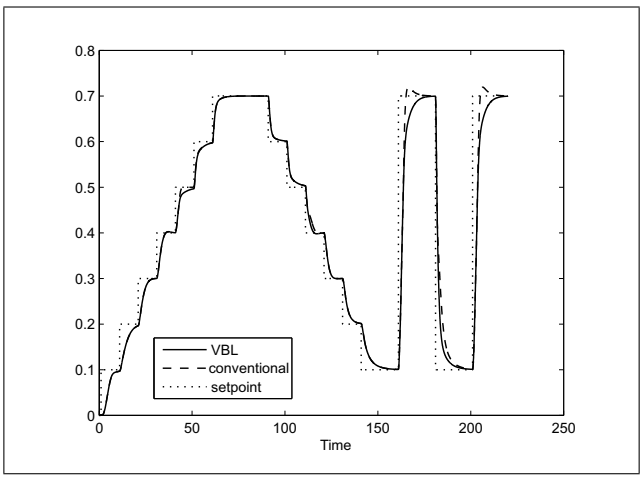

Figure 6.12: Simulation results

6.3 Example of the control design of a gas-liquid separator

The purpose of this example is to show a practically implemented gain-scheduling control in a way that is close to an engineer who normally designs process control. The task was to design a pressure control for a semi-industrial process plant for gas treatment in the process laboratory of the Systems and Control Department at the Jožef Stefan Institute. The schematic is shown in Figure 6.13. The problem is described in more detail in [5]. The control requirements are as follows:

- The behaviour of the closed-loop response should be as uniform as possible over the whole operating range.
- The control should be implemented with programmable logic controllers (PLCs) as typical representatives of industrial control hardware.
- The chosen control algorithm should be understandable to an engineer who normally works with PLCs.
- For simplicity, gain scheduling should be done by blending PI controllers and blending with triangular weighting (membership) functions.
- The control of the tank pressure must function with different liquid levels in the tank in such a way that the response is the same over the entire operating range.
- We are solving a nonlinear problem because the dynamics of the system change when the liquid level changes.

The control structure is shown in Figure 6.14, from the system-theory point of view by the block diagram in Figure 6.15, and from the implementation point of view by the block diagram in Figure 6.16.

The design process was the same as described in the previous sections. Some features of this design can be described as follows:

- We defined a suitable number of equilibrium points in which we have placed the centres of the blending functions by finding a compromise between good closed-loop behaviour (requiring as many functions as possible) and simple implementation (requiring as few functions as possible) due to the simplicity of the hardware.
- Tuning of the local controllers at equilibrium points was implemented with engineering tuning rules based on the responses to very small steps of positive and negative amplitude change around each equilibrium point.

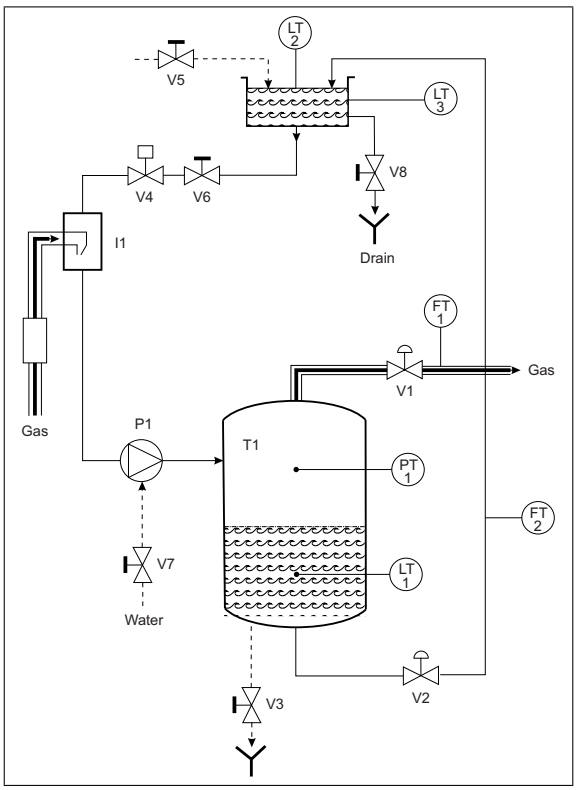

Figure 6.13: The technological schematic of the gas-liquid separator

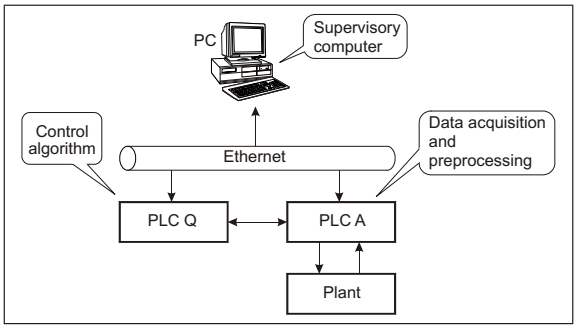

Figure 6.14: Control system for the gas-liquid separator

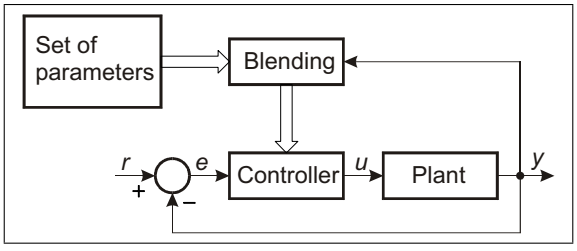

Figure 6.15: Block diagram of closed loop

- The whole design process is iterative and interactive, until the requirements are met. The design process would have been easier if we had used a mathematical model of the process, but that would have required additional effort and consequently cost in the

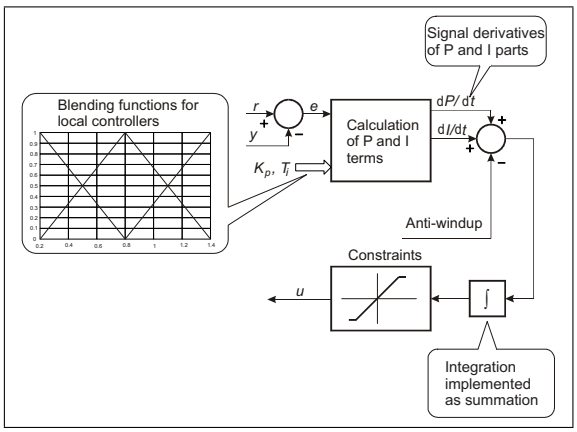

Figure 6.16: Implementation diagram of a blended gain scheduling controller for PLC

theoretical modelling of the plant, which is avoided by engineers in practice whenever possible.

- Due to the physical background, the scheduling variable was the height of the liquid, which is slower than the closed-loop response of the gas pressure. This means that the performance is not critical in any case. In reality, we would have to work with a vector of scheduling variables, but the analysis in [5] shows that the vector of scheduling variables can be simplified by one dominant variable: the fluid level in the vessel.

The simulation results, shown in Figures 6.17 and 6.18, show that the closed-loop system behaves differently with upward and downward steps due to the nonlinearity of the valve. Otherwise, we can conclude that we have a satisfactory closed-loop response over the entire range of interest with only three blended controllers.

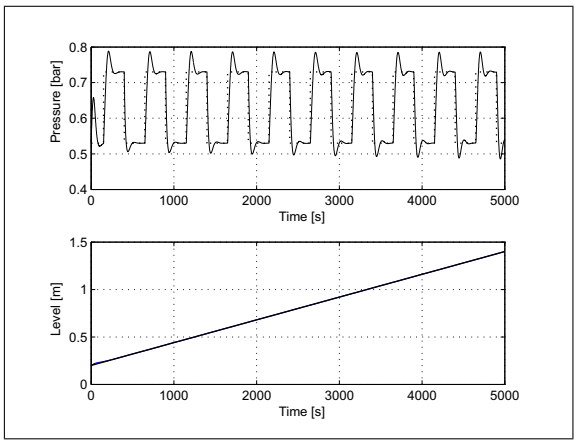

Figure 6.17: Simulation results

The next step was to test the designed controller with a PLC connected to a computer that simulated the dynamics of the plant (Hardware-in-the-Loop (HIL) simulation),

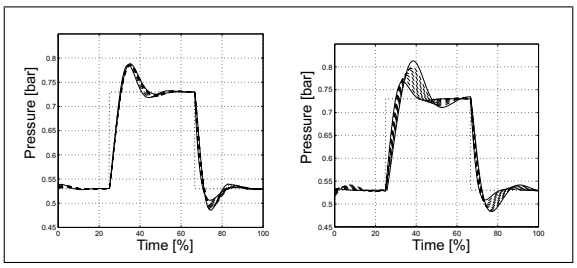

Figure 6.18: Simulation results - comparison of normalised signals (all signals in one image): closed-loop response with blended controller (left image), closed-loop response with a single PI controller

as shown in Figure 6.19 The simulation results can be seen in Figure 6.20.

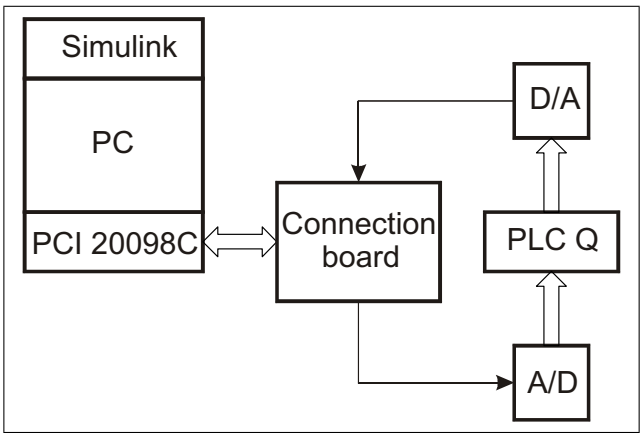

Figure 6.19: Hardware-in-the-Loop-simulation implementation

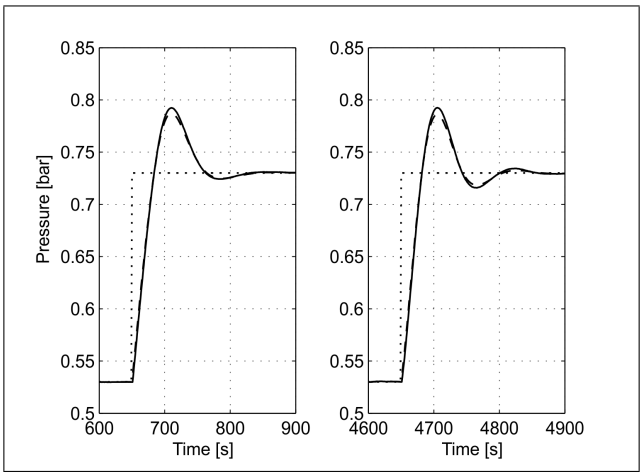

Figure 6.20: Comparison of the closed loop responses at the operating edge. The solid line represents the HIL response and the dashed line represents the computer simulation response

Finally, the controller is evaluated at the plant. In Figure 6.21, we see that the controller controls the unit satisfac-

torily. The differences between the measured and simulated behaviour can be seen in Figure 6.22. The difference between the rise times of the measurements and the simulation is due to the fact that the mathematical model we used for the simulation does not represent the dynamics of the gas-liquid unit well enough. Regardless of the observed discrepancies, it can be concluded that we were able to achieve similar dynamic behaviour over the entire operating range of the nonlinear system using a simple algorithm and industrial hardware.

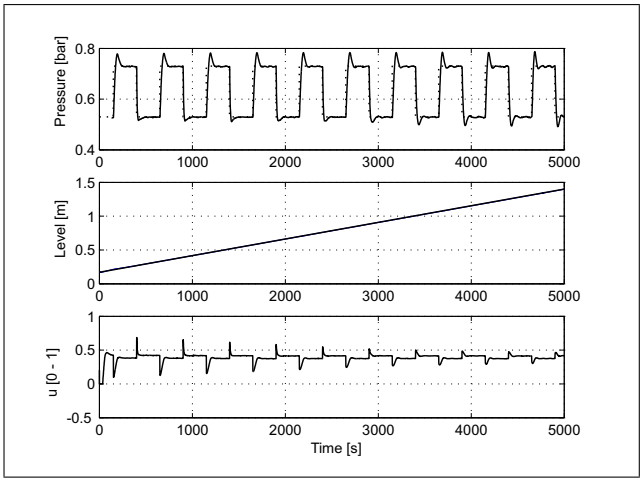

Figure 6.21: Measurements of closed-loop response, scheduling variable and control signal

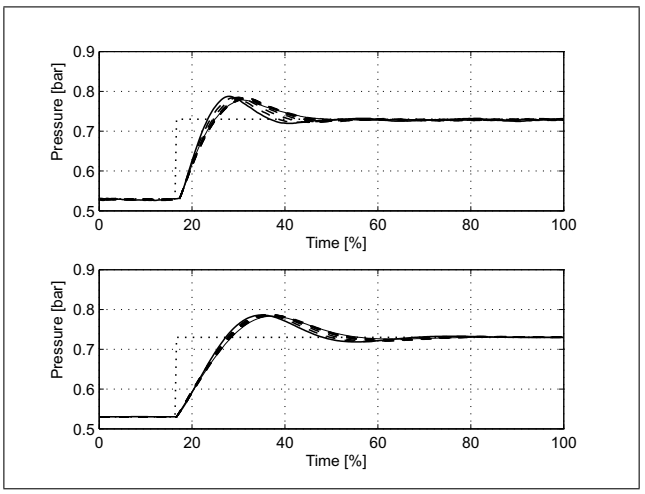

Figure 6.22: Close-up: comparison of the normalised signals between the measurements at the plant (upper image) and the computer simulation (lower image)

Bibliography

[1] B. Čokan, J. Kocijan (2001): Some realisation issue of fuzzy gain-scheduling controllers: a robotic manipulator case study, 6th On-line World Conference on Soft Computing in Industrial Applications, September 10-24, 2001. In: Roy, R. (edt.). Soft computing and industry: recent applications. New York: Springer, 2002, 191-199.

[2] J. Kocijan, D. J. Murray-Smith (1999): Robust nonlinear control for ship steering. In: Mastorakis, N. E. (edt.). Progress in simulation, modeling, analysis and synthesis of modern electrical and electronic devices and systems, World Scientific and Engineering Society Press, 235-240.

[3] J. Kocijan, D. Vrančić (2000): Multiple blended controller design for bilinear systems. In: Rudas, I. J. (edt.), Tar, J. K. (edt.). IFAC Symposium on Artificial Intelligence in Real Time Control, AIRTC-2000, October 2-4, 2000, Budapest, Hungary. Preprints. Budapest: Budapest Polytechnic, 187-192.

[4] J. Kocijan, N. Hvala, S. Strmčnik (2000): Multimodel control of wastewater treatment reactor. In: Mastorakis, N. (edt.). System and control: theory and applications, (Electrical and computer engineering series). World Scientific and Engineering Society, 2000, 49-54.

[5] J. Kocijan, G. Žunič, S. Strmčnik, D. Vrančić (2002): Fuzzy gain-scheduling control of a gas-liquid separation plant implemented on a PLC. Int. J. Control, Vol. 75, 1082-1091.

[6] D. J. Leith, W. E. Leithead (1998): Gain-scheduled & nonlinear systems: dynamic analysis by velocity-based linearisation families. Int. J. Control, Vol. 70, 289-317.

[7] D. J. Leith, W. E. Leithead (1998): Gain-scheduled controller design: an analytic framework directly incorporating non-equilibrium plant dynamics. Int. J. Control, Vol. 70, 249-269.

[8] D. J. Leith, W. E. Leithead (1999): Analytic framework for blended multiple model systems using linear local models, Int. J. Control, Vol. 72, 605-619.

[9] D. J. Leith, W. E. Leithead (2000): Survey of gain-scheduling analysis and design, Int. J. Control, Vol. 73, 1001-1025.

[10] D. J. Leith, W. E. Leithead (2007): Velocity-based gain-scheduling demo, `https://www.hamilton.ie/doug/demof.htm`

[11] W. J. Rough, J. S. Shamma (2000): Research on gain scheduling, Automatica, Vol. 36, 1401-1425.

Chapter 7

Identification of nonlinear systems with Gaussian processes

7.1 Gaussian Process

The Gaussian process is a stochastic process, also known as a 'random process'. A stochastic process is a generalisation of a random variable to a domain of independent variables. If the value of the random variable at each point in this domain is distributed according to normal (i.e., Gaussian) distribution, such a stochastic process is called a *Gaussian process* (GP) [16]. Alternatively, if the input to a stochastic process is a vector of independent variables $\mathbf{x}$, this process is Gaussian if the distribution of the values of the function $f(\mathbf{x})$ is Gaussian for each input vector $\mathbf{x}$.

The Gaussian-process model (abbreviated to 'GP model') is usually referred to as a nonparametric probabilistic model [23]. In this way, we allow *a priori* the description of an unknown system with an infinite set of functions instead of the usual restriction to a class of parameterised functions (i.e., basis functions). In doing so, we assign a higher probability to functions whose occurrence we consider more likely in the description of the system (e.g., smooth, stationary, periodic).

The basics of modelling with a GP model are described in [27] or the opening chapters in [23]; a more detailed explanation can be found in [23, 22].

It should be noted that the input and output of the GP model are not signals. As with other data-driven modelling methods, the input to the model is a vector of sampled values of independent variables $\mathbf{x}$, and the output of the GP model is a probability distribution of the output $f(\mathbf{x})$ for a given input vector. This follows from the Bayesian modelling [20], which introduces the probability aspect into the modelling. Let us look at the details of the modelling with GP.

For any set N of input vectors x_i, $i = 1, \ldots, N$, the GP is given by the vector of mean values of the $\mathbf{m} = [m_1(\mathbf{x}_1) \ldots m_N(\mathbf{x}_N)]^T$ and the covariance matrix $\mathbf{K}$,

$$\mathbf{K} = \begin{bmatrix} K_{11} & \ldots & K_{1N} \\ \vdots & \ddots & \vdots \\ K_{N1} & \ldots & K_{NN} \end{bmatrix}, \tag{7.1}$$

where:

$$m_i(\mathbf{x}_i) = \mathbb{E}[f(\mathbf{x}_i)] \tag{7.2}$$

and the elements of the covariance matrix K_{ij} are defined as:

$$\begin{aligned} K_{ij} &= \operatorname{cov}(f(\mathbf{x}_i), f(\mathbf{x}_j)) \\ &= \mathbb{E}[(f(\mathbf{x}_i) - m(\mathbf{x}_i)(f(\mathbf{x}_j) - m(\mathbf{x}_j)], \end{aligned} \tag{7.3}$$

usually obtained by a *covariance function* $C(\mathbf{x}_i, \mathbf{x}_j)$:

$$\operatorname{cov}(f(\mathbf{x}_i), f(\mathbf{x}_j)) = C(\mathbf{x}_i, \mathbf{x}_j). \tag{7.4}$$

If the distribution of a set of variables is Gaussian, the distribution of any subset of the elements of that set is also Gaussian, which is called the consistency requirement. This property is important for the performance of a GP model when the elements of the covariance matrix of the GP are obtained by a covariance function [23].

Covariance function

The value of the covariance function $C(\mathbf{x}_i, \mathbf{x}_j)$ expresses the correlation between the values of the outputs $f(\mathbf{x}_i)$ and $f(\mathbf{x}_j)$ based on the values of the input vectors $\mathbf{x}_i$ and $\mathbf{x}_j$. The covariance function can be of arbitrary form if it forms a nonnegative definite covariance matrix $\mathbf{K}$ for any set N of input vectors $\mathbf{x}_i, i = 1, \ldots, N$. The covariance functions can be stationary (depending only on the distance between the data), nonstationary, periodic, etc., and are given in more detail in [23]. The covariance function defining the form of the unknown function $f(\mathbf{x})$ is usually not known in advance but can be obtained from knowledge of the general properties of the function $f(\mathbf{x})$.

The most commonly used covariance function is the squared exponential or Gaussian covariance function, which expresses two general properties:

- smoothness, which indicates that the output of the function being modelled changes relatively little when the change in the input value is small. The correlation is higher for two output values whose input values are close to each other.

- stationarity, where the covariance between two input vectors depends only on their distance from each other and not on their absolute position in the domain.

It is often used when there is no *a priori* knowledge about the structure of the function to be modelled. Here, the covariance between two outputs $y_i = f(\mathbf{x}_i)$ and $y_j = f(\mathbf{x}_j)$:

$$C_{SE}(\mathbf{x}_i, \mathbf{x}_j) = v_1 \exp\left[-\frac{1}{2} \sum_{d=1}^{D} w_d (x_i^d - x_j^d)^2 \right]; \qquad (7.5)$$

D is the length of the input vector $\mathbf{x}$ or the number of independent variables of the model. The parameters v_1 and w_d, $d = 1, \ldots, D$ of the covariance function are arbitrary. They are defined as *hyperparameters*[1] [22, 19] to emphasise that these are parameters of an otherwise nonparametric model [2], which determine the shape of the unknown function $f(\mathbf{x})$. The parameter v_1 is the scaling coefficient of the covariance, and the parameters w_d reflect the importance of each component of the input vector: the larger the parameter w_d, the more influential the change in the component vector x^d is on the value of the output. For a given covariance function to form a positive definite covariance matrix, all parameters of the squared-exponential covariance function must be greater than zero.

Modelling

The operation of the GP model is best illustrated by an example. Suppose we want to describe a system

$$y = f(\mathbf{x}) + v, \qquad (7.6)$$

where v is white Gaussian noise with mean 0 and variance v_0, $v \sim \mathcal{N}(0, v_0)$. Based on N input-output samples (i.e., pairs of vectors) $(\mathbf{x}_i, y_i)$ collected in the set $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$, we want to determine the unknown value of the output y^* given the values of the input vector $\mathbf{x}^*$. In the sequel, we refer to the matrix $\mathbf{X}$ of dimension $N \times D$ together with the vector $\mathbf{y}$ of dimension $N \times 1$ as a *training dataset*, since we use them to train or learn the GP model. A particular input/output pair $(\mathbf{x}_i, y_i)$ from this dataset is also called a 'training data pair' or 'training point'. The pair $(\mathbf{x}^*, y^*)$ is referred to as the test dataset or *test input/output* pair.

The training outputs y_i, $i = 1, \ldots, N$ represent the values of the random variables resulting from the Gaussian process. We assume that the output of the system is smooth and the system is stationary and use the Gaussian covariance function (7.5) with parameters unknown at the beginning, to form a covariance matrix $\mathbf{K}$.

$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \mathbf{K})$, where the elements of the covariance matrix $K_{ij} = C(\mathbf{x}_i, \mathbf{x}_j) = C_{SE}(\mathbf{x}_i, \mathbf{x}_j) + v_0 \delta_{ij}$. $C_{SE}(\mathbf{x}_i, \mathbf{x}_j)$ are the elements of the covariance matrix given by the covariance function (7.5) and $v_0 \delta_{ij}$ describe the effect of the noise at the output of the system, where δ_{ij} is the Kronecker operator. Since we have assumed white noise, its values are only correlated with themselves.

Since the (as yet unknown) output y^* is an embodiment of the same system as the training outputs $\mathbf{y}$, we can write [3]: $\mathbf{y}_{N+1} = \begin{bmatrix} \mathbf{y} \\ y^* \end{bmatrix} \sim \mathcal{N}(0, \mathbf{K}_{N+1})$. The joint covariance matrix $\mathbf{K}_{N+1}$ of the vector $\mathbf{y}_{N+1}$ can be decomposed:

$$\mathbf{K}_{N+1} = \begin{bmatrix} \begin{bmatrix} & \mathbf{K} & \end{bmatrix} & \begin{bmatrix} \mathbf{k}(\mathbf{x}) \end{bmatrix} \\ \begin{bmatrix} \mathbf{k}(\mathbf{x})^T \end{bmatrix} & \begin{bmatrix} k(\mathbf{x}) \end{bmatrix} \end{bmatrix}. \qquad (7.7)$$

The matrix $\mathbf{K}$ is the covariance matrix of the training data, $\mathbf{k}(\mathbf{x}^*)$ is the vector of covariances between the training data and the test data, $k(\mathbf{x}^*)$ is the autocovariance of the test data.

According to Bayes' principle [20], the probability distribution of the output y^* can be divided into two parts: the part that determines the probability of the learning outputs given the learning inputs (i.e., the marginal part): $p(\mathbf{y}|\mathbf{X}) \sim \mathcal{N}(\mathbf{0}, \mathbf{K})$, and the conditional part that, given the first part and the input $\mathbf{x}^*$, predicts the probability distribution of the output y^*. In formal terms, the calculation of the probability distribution of the output response y^* is as follows [16]:

$$p(y^*|\mathbf{x}^*, \mathbf{y},\ XX) = \int p(y^*|\mathbf{x}^*, \boldsymbol{\theta}, \mathbf{y}, \mathbf{X}) p(\boldsymbol{\theta}|\mathbf{y}, \mathbf{X}) d\boldsymbol{\theta}. \quad (7.8)$$

Normally this integral is analytically intractable, but two approximations [16] are available. The first, more common, is to approximate the integral using the most likely values of the unknown hyperparameters $\boldsymbol{\theta}$:

$$p(y^*|\mathbf{x}^*, \mathbf{y},\ XX) \approx p(y^*|\mathbf{x}^*, \boldsymbol{\theta}, \mathbf{y}, \mathbf{X}). \qquad (7.9)$$

We use those values of the hyperparameters $\boldsymbol{\theta}$, for which the probability of training outputs $\mathbf{y}$ is greatest given the values of the training inputs $\mathbf{X}$ and the covariance function $C(.,.)$. These are determined by searching for the maximum marginal likelihood. To avoid constrained optimisation, we use the logarithm of the marginal likelihood when maximising the marginal likelihood:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}) &= \log(p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})) \\ &= -\frac{1}{2}\log(|\mathbf{K}|) - \frac{1}{2}\mathbf{y}^T \mathbf{K}^{-1} \mathbf{y} - \frac{N}{2}\log(2\pi), \end{aligned} \qquad (7.10)$$

[1] Neal [19] has shown that a forward neural network with a hidden layer having an infinite number of neurons corresponds to a GP model. Hyperparameters determine the distribution of the neural network's parameter values

[2] The model is nonparametric because in addition to the hyperparameters and the covariance function, we need information about the behaviour of the function $f(\mathbf{x})$ in the form of the input/output data used in modelling

where $\boldsymbol{\theta}$ is a vector of parameters, $\boldsymbol{\theta} = [w_1 \dots w_D \; v_1 \; v_0]^T$ and $\mathbf{K}$ covariance matrix for the training data $\mathcal{D}$. If the optimisation is performed with the conjugate-gradient method or another gradient method, a further calculation requires the derivation of the log likelihood after all hyperparameters:

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \theta_i} = -\frac{1}{2}\mathrm{Tr}\left(\mathbf{K}^{-1}\frac{\partial \mathbf{K}}{\partial \theta_i}\right) + \frac{1}{2}\mathbf{y}^T\mathbf{K}^{-1}\frac{\partial \mathbf{K}}{\partial \theta_i}\mathbf{K}^{-1}\mathbf{y}. \quad (7.11)$$

At each optimisation step, the inverse of the covariance matrix $\mathbf{K}^{-1}$ must be calculated, which is computationally challenging for large N.

The alternative to approximate the integral (7.8) is numerical integration over the entire hyperparameter distribution using the Markov Chain Monte Carlo method [16].

The method for determining the model hyperparameters is the cross-validation method [23]. The values of the hyperparameters are searched as usual, except that we divide the training data into k parts for k-fold cross-validation. We use $k-1$ parts for training and the remaining part for validation. Repeat the process k times, each time with different data for validation. An extreme example is leave-one-out cross-validation (LOO cross-validation) with only one data point for validation. The biggest problem with this procedure is the computational complexity, as we have to learn k models from [23].

Prediction

The joint distribution $p(\mathbf{y}_{N+1})$ is Gaussian. Therefore, the conditional distribution is also Gaussian $p(y^*|\mathbf{x}^*, \mathbf{y}, \mathbf{X}) = \frac{p(\mathbf{y}_{N+1})}{p(\mathbf{y}|\mathbf{X})}$. By simplification [16, 27], the predicted output of the system (7.6) is a Gaussian distribution:

$$p(y^*|\mathbf{x}^*, \mathbf{X}, \mathbf{y}) = \mathcal{N}\left(\mu(\mathbf{x}^*), \sigma^2(\mathbf{x}^*)\right) \quad (7.12)$$

with mean $\mu(\mathbf{x}^*)$ and variance $\sigma^2(\mathbf{x}^*)$:

$$\mu(\mathbf{x}^*) = \mathbf{k}(\mathbf{x}^*)^T\mathbf{K}^{-1}\mathbf{y} \quad (7.13)$$

$$\sigma^2(\mathbf{x}^*) = k(\mathbf{x}^*) - \mathbf{k}(\mathbf{x}^*)^T\mathbf{K}^{-1}\mathbf{k}(\mathbf{x}^*), \quad (7.14)$$

where $\mathbf{k}(\mathbf{x}^*) = [C(\mathbf{x}_1, \mathbf{x}^*) \dots C(x_N, \mathbf{x}^*)]$ is the $N \times 1$ covariance vector between the test data and the training data mentioned above and $k(\mathbf{x}^*) = C(\mathbf{x}^*, \mathbf{x}^*)$ is the auto-covariance of the test data. An illustration of the above can be found in Figure 7.1.

Interpretation

The GP model consists of the following parts:

- pairs of input/output training data (points) $\mathcal{D}$, which represent the the behaviour of the unknown system,

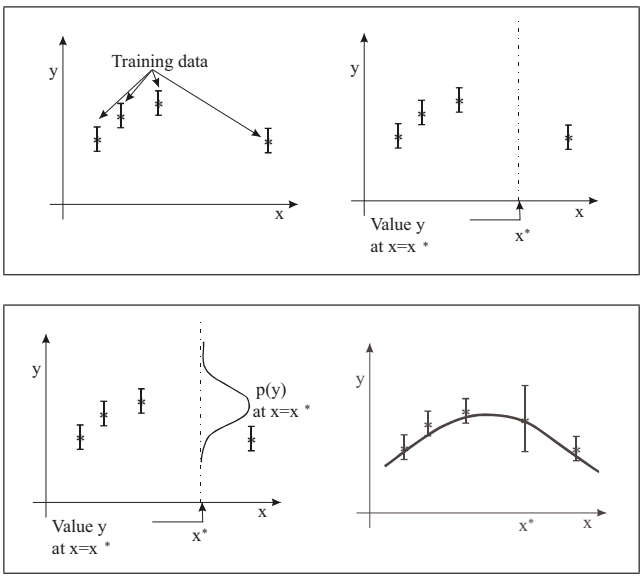

Figure 7.1: An illustration of modelling with Gaussian processes

- mean function, and
- covariance function $C(.,.)$ with known or optimised hyperparameters $\boldsymbol{\theta}$, which indicate how the data $\mathcal{D}$ are related.

Since the model GP requires information about the unknown function in the form of the training inputs and outputs even after training, the model is not parametric. The hyperparameters only indicate via a covariance function how the learning information is used for prediction, but they do not contain any information about the function/system to be described.

We can interpret the vector $\mathbf{k}(\mathbf{x}^*)^T\mathbf{K}^{-1}$ in the expression for the mean of the predicted output (7.13) as a vector of weights that determines the weighting of each training output y_i in $\mathbf{y}$ according to the distance between the training vector and the test input vector. This linear predictor can be understood as smoothing the information about the unknown system (training data) contained in the GP model. Alternatively, the prediction mean $\mu(\mathbf{x}^*)$ can be thought of as a linear combination of N basis functions, centred on the training points: $y^* = \sum_{i=1}^{N} \alpha_i \; C(\mathbf{x}^*, \mathbf{x}_i)$. The output of the system is a sample from the resulting normal distribution (7.12).

The low variance $\sigma^2(\mathbf{x}^*)$ of the predicted distribution of the output means more *confidence* in the prediction and vice versa. If we look at the expression for the variance, we see that it consists of two parts [23]. The term $\mathbf{k}(\mathbf{x}^*)^T\mathbf{K}^{-1}\mathbf{k}(\mathbf{x}^*)$ is subtracted from the first part $k(\mathbf{x}^*)$, which is the *a priori* GP variance. This represents the reduction in the *a priori* variance of GP at $\mathbf{x}^*$ due to the training data and increases as the covariance between the training and test data increases. Simply put, the test input is 'closer' if it is already known as training data, which

means that the GP model has a higher confidence in the prediction. The variance also depends on the position of the test input data relative to the training input data, which is one of the main advantages of the GP model over other models.

Comparison with other types of models

General about GP model properties

The GP model differs from most models in that it is *nonparametric*. This is because the information about the unknown system is described both by the training data $\mathcal{D}$ itself contained in the model and by the (hyper)parameters of the covariance function and the covariance function itself.

The advantages of the GP model are:

- a measure of uncertainty in the output prediction given by the variance and depending on the mutual covariance (position) of the training input vectors and the test input vectors;
- the possibility to include different types of prior knowledge, e.g., linear local models, hysteresis, prior knowledge about noise, static properties, etc.;
- relative ease of use;
- a small number of hyperparameters that simultaneously express the influence of the individual input components and noise.

In addition to these advantages, the GP model also has the following disadvantages:

- the nonparametric nature of the GP model, which limits the application methods;
- the high computational cost of training when the unknown system is described by a large number of training data.

The computational effort can be reduced in several ways. One way is to cluster the training information, meaning into local models. Another way is to speed up the training by using a smaller subset of the training set (e.g., [24]) or by approximating the inverse of the covariance matrix, which is the most computationally intensive. These and other methods are described in [21, 23] and the references therein.

Comparison

Let us review the differences between the GP model and other models obtained by experimental modelling.

The comparison with methods that first determine the structure (e.g., by theoretical modelling) and then optimise the parameters with any optimisation method is not quite appropriate, as it is a comparison between grey-box and black-box modelling methods.

It is easier to compare the GP model with neural networks in the context of modelling. It turns out that neural networks have two major problems in addition to the problem of determining a suitable structure:

- the lack of transparency: the structure of the neural network does not reflect the structure of the unknown system, and
- the curse of dimensionality, which expresses itself in two ways. As the dimension of the input space increases:
  - the demand for data increases exponentially with dimension, and
  - the number of building blocks (neurons) in a neural network also increases,

  resulting in higher computational complexity and a higher probability of terminating the optimisation at a local minimum.

The advantages of the GP model over neural networks are an indicator of confidence in the prediction of the GP model, better performance of the GP model with a smaller number of data, and a reduction in the 'curse of dimensionality'. For more on the relationship between the GP model and neural networks, see [10, 17, 19].

Fuzzy logic and local-model networks reduce the problems described for neural networks, but in certain cases the GP model has potential advantages over them. In particular, when small or incomplete data sets for identification are involved.

Models such as Support Vector Machines (SVMs) are closely related to the GP model, as the methods use kernel functions like the GP model, where kernel functions are called covariance functions. The main advantage of the GP model over these models is the confidence in the prediction of the GP model and the use of conditional probability to determine the parameters. The disadvantage is the complexity of the calculations. The Relevance Method Vector Machines (RVM) is a special form of the GP model [23].

The connection between GP models and some other models is described in more detail in [17, 23] and the references therein.

Example of identification of a static nonlinearity with GP

Let us illustrate the use of the GP model with an example. We want to identify a nonlinear function $f(x)$ as a function of the independent variable x:

$$f(x) = 4x^2 + x - 6\sin(x) + 1 + v \qquad (7.15)$$

in the interval $x \in [0, 1.2]$. The variance of the Gaussian noise v at the output is $\sigma^2 = 0.0025$. The nonlinear function is represented by eight unevenly spaced pairs of training data representing the input/output relation $x/f(x)$. The function and the training data can be seen in Figure 7.2.

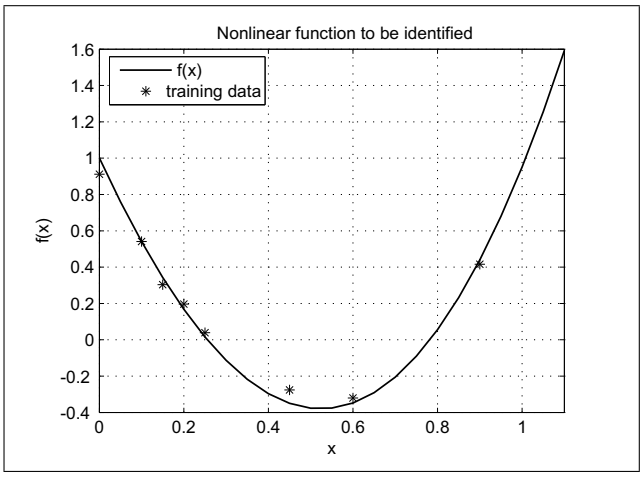

Figure 7.2: Nonlinear function

For the identification, we choose the squared-exponential covariance function (7.5). Since there is only one input, the function simplifies to:

$$C(x_i, x_j) = v_1 \exp\left[-\frac{1}{2}w(x_i - x_j)^2\right] + v_0\delta_{ij}. \qquad (7.16)$$

The optimisation leads to three hyperparameters, which have the following values $v_1 = 13.8$, $w = 4.2$ and $v_0 = 0.0065$. The identification results are shown in Figure 7.3. It can be observed that the model poorly describes the unknown function in a region not described by the training data ($x > 1$) and that the prediction in a region with sparse data (i.e., described by only a few points), namely $x > 0.7$. A good feature of the GP model is that it alerts us to a less well-described region by an increased variance in Figure 7.3, which is particularly visible for $x > 1$. It is less noticeable because of the lower noise is the second property of the GP model, specifically the smoothing of the training information it contains, where the model smooths the noise contained in the training data to predict the new output.

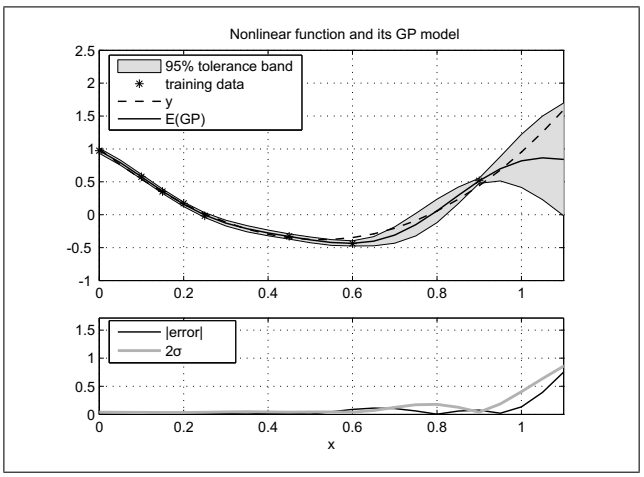

Figure 7.3: Output of model (solid curve) with uncertainty (grey area) and training data (stars)

7.2 Identification of dynamic systems with GP

The following sections summarise the good features of the GP model when used for the identification of dynamic systems:

- the prediction is given in the form of a Gaussian distribution, and the variance can be seen as a measure of confidence in the prediction; this depends on the quality and location of the training data in the domain;[3]
- the number of hyperparameters that have to be optimised in the identification is relatively small;
- the model is quite robust, as it works relatively well even with a small amount of training data, e.g., in the nonequilibrium region;
- it is possible to incorporate different types of prior knowledge, such as knowledge about static features, local models, etc.;
- the curse of dimensionality does not increase exponentially with the amount of data but with the third power [22];
- the model does not use scheduling variables nor is it necessary to divide the operational domain into sub-domains, as in local-model networks.

Of course, the GP model also has some disadvantages:

[3] The 'confidence measure' in the predicted output is not exclusively the domain of GP models. How it can be determined for fuzzy models is described, for example, in [14]. In general, it is a property of Bayesian modelling

- it is a nonparametric model and, as such, useless when a parametric model is needed, for example, for control design;
- it is computationally expensive to use the model in the case of a large training dataset; this is especially true for the hyperparameter-optimisation phase.

So, when should one use the GP model for the identification of dynamic systems instead of basis-function identification methods? It should be done when:

- the information about the system is available in the form of input/output data;
- the available data is poor, contains a lot of noise, measurement errors, missing and unevenly distributed data;
- we need an indication of the uncertainty of the prediction;
- we have a relatively small, but not too small, number of data relative to the number of regressors.

Identification procedure

In this subsection, we will first outline the identification procedure of dynamic systems with the GP modelling [2]. In system identification, the model is obtained from measurements, but of course we can also use our prior knowledge of the system if we have it. Roughly speaking, identification with the GP model consists of the following steps:

1. defining the purpose of the model;
2. GP model set-up;
3. design of the experiment using a priori knowledge or *a priori* measurements;
4. experiment and signal preprocessing to obtain data for training and validation;
5. training of the GP model, i.e., optimisation of the hyperparameters; and
6. validation of the GP model.

At each of these steps, we can decide whether to proceed to the next identification step or return to one of the previous ones. The whole identification procedure is rarely done at once. It is usually an iterative procedure. System identification is complete when we decide, based on validation, that the model is sufficient for its purpose.

In the following, we will first briefly outline the different steps of the identification process. Most of the steps in the identification procedure are the same as in other system identification methods [15], which we described in Chapter 3. We will highlight the steps of the identification procedure that differ significantly from the same steps in other types of models due to the characteristics of the GP model.

Definition of the purpose of the model

The purpose of identification is to obtain a model of a system. The model is not an end in itself, but it is to be used for a specific task. This task defines the criteria that the model must fulfil in order to be accepted as a satisfactory model. We can use the model for simulation, response prediction, control design, system analysis, fault diagnosis, and other functions.

Model set-up

The next step in the identification process is to build the model. We set up the GP model based on a priori knowledge about the process, previous measurements, or *validation results* in the iterative system identification process. The construction contains:

- the selection of an appropriate covariance function,
- the selection of regressors, and
- the decision whether to include prior knowledge.

The choice of regressors when modelling dynamic systems simultaneously determines the order of the model.

The design of experiment

The next step in the identification process is the design of the experiment, which is done on the basis of knowledge about the process gained from prior knowledge or previous measurements and according to the model setup.

The design process is similar to other methods in the following aspects:

1. the selection of inputs and outputs, taking care to measure all influencing variables;
2. the selection of the appropriate sampling time;
3. the selection of an appropriate excitation signal so that the process is described over the entire operating range of interest; this is a feature of the GP model that the model interpolates well in the region where we have training data and extrapolates poorly everywhere else.

In designing the experiment, we need to consider constraints such as the following:

- disturbances of the measured signals;
- limitations of the input signal amplitude for physical reasons, safety reasons and limitations of the actuators;
- the limited time available to perform the identification, e.g., when measuring in an industrial environment.

Experiment and data processing

The experiment is carried out as we planned it in the previous section. The result of the experiment is measured input and output signals of the process that represent its behaviour.

The input to GP is not the signals but the sampled values that determine the behaviour of the system at certain values of the regressors. These samples are obtained by sampling the input and output signals of the system being modelled. The sampling time is determined in the design phase of the experiment.

An example of a possible selection of input/output training pairs (with index i) for a system with input signal u and output signal y, which we would like to describe, for example, with a second-order NARX model:

- training output i: the output of the process at time $t = kT$: $y_i = y(k)$;
- corresponding training input: $\mathbf{x}_i = [y(k-1)\ y(k-2)\ u(k-1)\ u(k-2)]$,

where T is the sampling time of the input and output signals. The test data is constructed in the same way as the training data.

A good feature of the GP model is that the significance of each regressor is reflected by the value of the corresponding hyperparameter. This is called automatic relevance detection (ARD). For a squared exponential covariance function (7.5), for example, the larger the value of the hyperparameter w_d, the more significant the corresponding regressor d. This property can only be used if the resulting training samples are *normalised* before being used for training, so that the sample values of the individual regressors are in the same size class.

In the case of missing regressor values in a given regression vector, we can either replace these values, for example, with the average of the previous and subsequent samples or discard such a regression vector or replace it with another one if we have enough data available.

Model training

Training a GP model is the same whether it is a model that is static or one that describes dynamic systems with the NARX model. During training, we optimise the hyperparameters $\boldsymbol{\theta}$. These are not known in advance but must be determined from the training data.

Usually, the maximum-marginal-likelihood optimisation is used because it is simple but gives good results. To determine the most probable values of the hyperparameters, we can generally use any of the optimisation methods. [5]

Since there is a possibility that the optimisation process becomes stuck at the local minimum, we usually repeat the training process several times for different initial values of the hyperparameters and validate the resulting models.

Model validation

The purpose of validation is to check the fit of the mathematical model and the system under consideration [18]. Although validation is a highly important step in the identification process that provides information about how good the resulting model is, it is often not given sufficient attention.

The quality of a model can be measured in several ways, the most important aspects being:

- model plausibility (model verification): here we are interested in the consistency of the model with prior knowledge;
- model falseness: here we are usually interested in the match between the model's and the system's responses; and
- model purposiveness: here we check whether the model is appropriate for the intended task.

An overview of validation methods can be found in [18, 8], for example.

Very often, in addition to the one-step-ahead prediction of the model, we use the results of the simulation, which are compared to the behaviour of the system for validation. We validate the NARX model as an output-error model. In the next subsection, we describe the simulation of the GP model.

Simulation of dynamic GP models

Because of its form, the GP model is used as an input/output discrete-time dynamic model, often as the NARX model [25]. At time instant k, the input vector $\mathbf{x}_k$ contains the

past values of the inputs to the system u and the outputs of the system y:

$$\mathbf{x}_k = [y(k-1) \dots y(k-L) \;\; y(k-1) \dots u(k-L)]. \quad (7.17)$$

In general, there are two possibilities for the realisation of a multi-step prediction:

- *direct method*, by which we learn multiple models, for each prediction horizon separately, or
- *iterative method*, by which we learn one model for one-step-ahead prediction, which we repeat iteratively.

The problem with the direct method is that we have to choose the prediction horizon in advance. If we change it, we have to learn a new model. Another problem with this method is that for strongly nonlinear systems and large horizons, a large number of data must be available [5]. Since the simulation is a multi-step-ahead prediction for an infinite horizon, the direct method is limited.

In contrast, the model for a one-step-ahead prediction can be built relatively easily, and by iteratively feeding back the output values, it is possible to obtain a model for any desired prediction horizon [5]. This method is also used in basis-function models, for example, in the simulation of systems with neural networks.

The problem with the iterative simulation method is error accumulation as we move forward in time. One possible solution to the problem is to eliminate the systematic error that arises from successive one-step-ahead predictions [26, 9]. However, using the GP model, there is also the possibility that we do not eliminate the error, but use the variance of the prediction for the evaluation of response [5, 13].

Simulation procedure

Suppose we know the response history of the dynamic system of order L up to step k. Then, at step $k+1$, we know the complete input vector of the GP model:

$$\mathbf{x}_{k+1} = [y(k) \dots y(k-L+1) \;\; u(k) \dots u(k-L+1)] \quad (7.18)$$

In step $k+2$, the new input vector for the GP model is:

$$\mathbf{x}_{k+2} = [y(k+1)\, y(k) \dots y(k-L+2) \;\; u(k+1) \dots u(k-L+2)]. \quad (7.19)$$

The problem here is that we do not know the value of the output at time $k+1$. We do something similar to neural-network models; instead of the real value of the output of the system $y(k+1)$, we use its estimated value $\hat{y}(k+1) = f(\mathbf{x}_k)$, calculated with the GP model. This is how we proceed in all further steps.

We have two simulation options:

- *simulation without uncertainty propagation* or the 'naive' method, with which, similar to neural networks, we only take the mean value of the the predicted value and
- *simulation with uncertainty propagation*, in which all or part of the information about the predicted distribution is used. The procedure is described in detail in [5], and a shortened version is given in [13, 4]. Since we cannot calculate the exact distributions at the output of the model, we must again resort to approximations. We have two possible approximations [5, 17]:
  - analytical approximation of the statistical moments of the output distribution, where the output with *non-Gaussian distribution* is approximated by a Gaussian distribution with the same mean and variance, which is called a statistical-moments-matching method (Figure 7.9), and
  - numerical Monte-Carlo method.

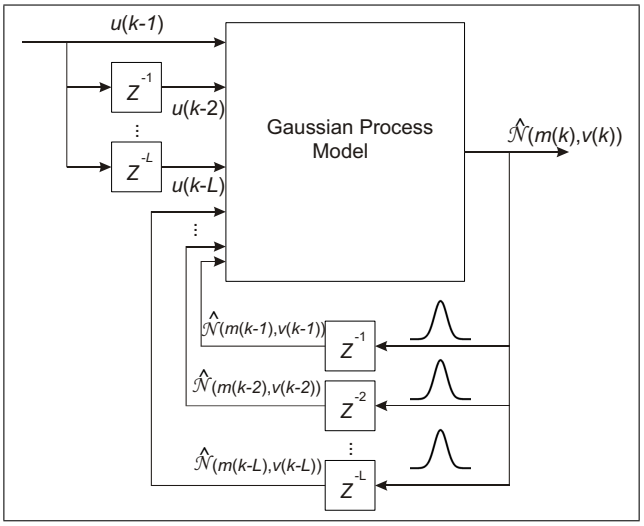

Figure 7.4: Simulation schema for the identified GP model of the dynamic process

So, when should one use the first or the second method?

The uncertainty propagation method provides a more accurate measure of confidence in the prediction, but it is more computationally intensive for both the analytical and numerical approximations. In contrast, the method without uncertainty propagation is fast and simple and still provides a meaningful measure of confidence in the prediction. Admittedly, this measure is too optimistic because the cumulative error from the previous steps is not taken into account in the prediction variance.

Variance propagation not only leads to more accurate prediction output, but also affects the mean of the predicted distribution. Depending on the nonlinearity of the system, these differences can be larger or smaller. Since both the 'naive' and uncertainty-propagation simulations are

approximations, we can only guess by how much the results of the latter are more accurate. However, the model uncertainty needs to be validated.

The 'naive' method is therefore used when we are interested in speed and simplicity, and the uncertainty propagation when we are interested in a more accurate level of confidence in the prediction and the simulation time is not too limited.

Example of identification of a dynamic system

Let us illustrate the method described by identifying a first-order nonlinear dynamical system (3.9), which we will model as a first order model:

$$y(k+1) = f(y(k), u(k)). \tag{7.20}$$

The function f is a GP, and we have a regression vector with two variables $D = 2$. This means that we use the squared-exponential covariance function to identify the hyperparameters v_0, v_1, w_1, w_2.

Input and output signals

The main characteristics of the selected input and output signals are as follows:

- The input signal is defined in the range [-1.3, 1.3].
- The sampling time selected according to the system dynamics is 0.5 s.
- Input signal for identification (as in Chapter 3):
  - is generated by a random number generator,
  - the amount of data determines the dimension of the covariance matrix,
  - the size of the covariance matrix determines the computational load.
- input signal for validation (the same as in Chapter 3):
  - generated by a random-number generator, but with different sampling and amplitude in the range [-1.2,1.2],
  - a second validation signal with an amplitude outside the identified region, namely in the range [-1,3,1,5].

The response of the simulation to the input identification signal is shown as follows in Figure 7.5, and a comparison of the model prediction with that of the original system is

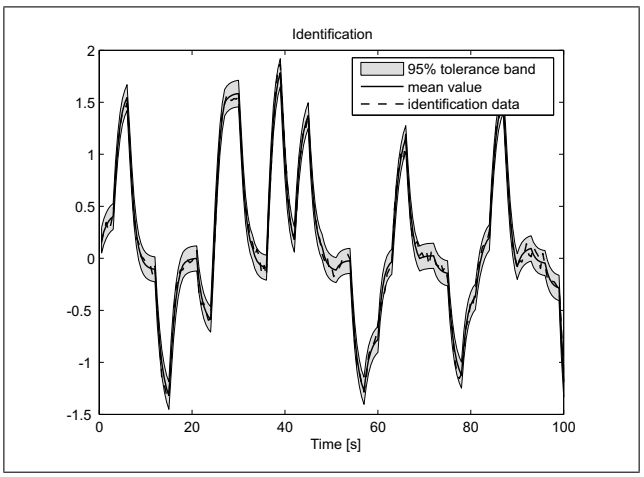

Figure 7.5: Simulation response to identification signal

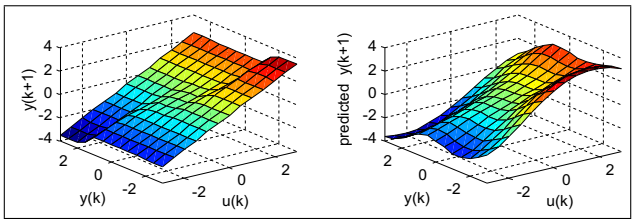

Figure 7.6: Nonlinearity of the system y(k+1)=f(u(k),y(k)) (left figure) and GP model (right figure)

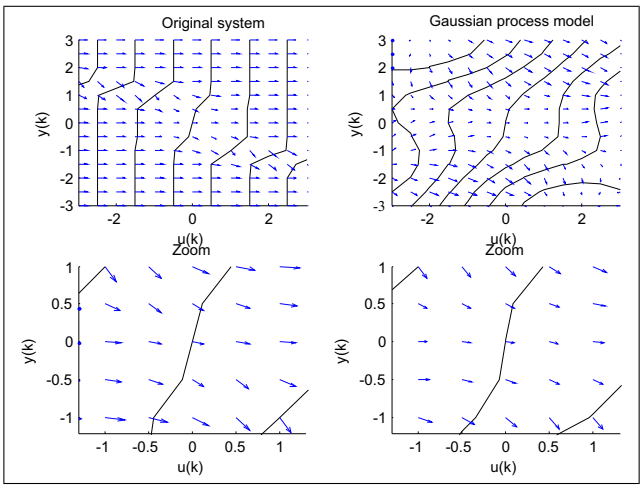

Figure 7.7: Comparison of prediction countours and gradients

shown in Figures 7.6 and 7.7. The analytical approximation with statistical moment matching was implemented for the simulation.

The validation simulation is shown in Figures 7.8 to 7.11.

Responses can be quantitatively assessed using various statistical measures. Particularly common are the mean

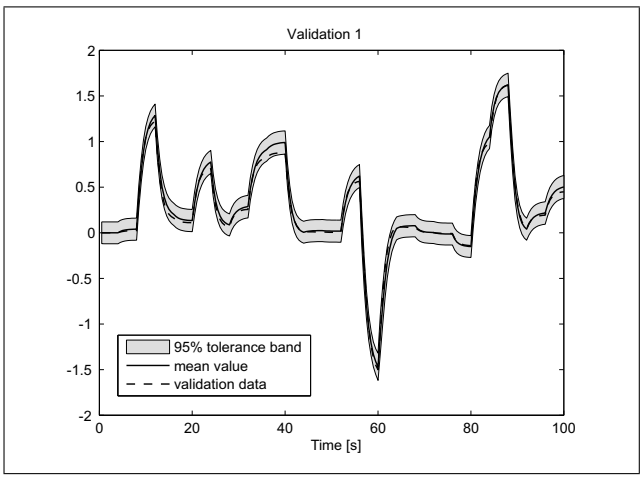

Figure 7.8: Simulation response to the first validation signal

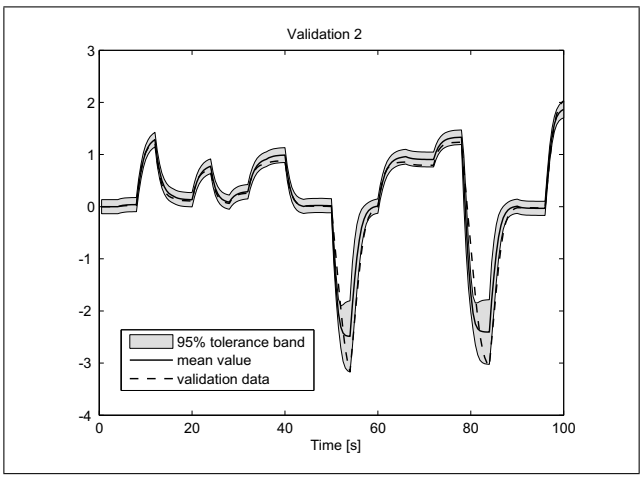

Figure 7.9: Simulation response to second validation signal

absolute error and especially the mean of the square of error, which is especially suitable for all methods for which the least-squares method is used for optimisation. An example of a statistical measure that takes the entire distribution into account is the logarithm of the error of the prediction density (LD), which gives more weight to the error of those predictions where the model with a lower variance has a higher confidence in the prediction. For all three, the lower the value, the better the model.

These statistical measures are summarised in the following equations and results for our example:

$$AE = \frac{i=1}{N}\sum|\hat{y}_i - y_i| = 0.028,$$

$$SE = \frac{i=1}{N}\sum(\hat{y}_i - y_i)^2 = 0.0016,$$

$$LD = \frac{1}{2N}\sum_i(\log(2\pi) + \log(\sigma_i^2) + \frac{(\hat{y}_i - y_i)^2}{\sigma_i^2}) = -1.3842,$$

where: AE - average value of the absolute error
SE - mean value of the square of the error
LD - logarithm of the error of the predictive density

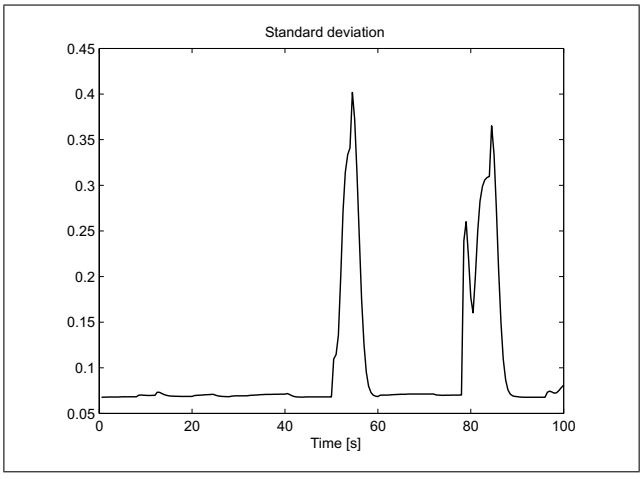

Figure 7.10: Standard deviation of the simulation response to the second signal for validation

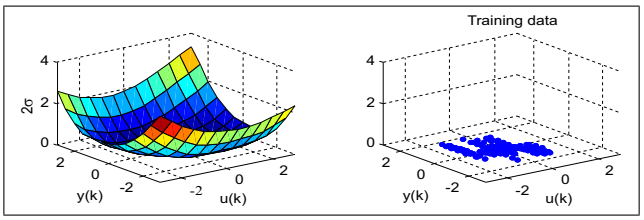

Figure 7.11: Uncertainty surface (left image) $y(k+1) = f(u(k), y(k))$ for GP model and position of training data (right figure)

7.3 Example of the identification of the pH-neutralisation process

Let us consider another example of the identification of the pH-neutralisation process (2.20) [10], which we will model with a higher order model. We have already used it to show the modelling of neural networks in Chapter 3.

Identification

We iteratively selected the following model:

- covariance function:

$$C(\mathbf{x}^p, \mathbf{x}^q) = v_1 \exp\left[-\frac{1}{2}\sum_{d=1}^{D} w_d(x_d^p - x_d^q)^2\right] + v_0\delta_{pq}.$$

- Regressors: $y(k-1), ..., y(k-4), u(k-1), ..., u(k-4)$, implying ten hyperparameters to be optimised.
- This implies a higher computational cost. In our experience, it is about 110 times higher than for

the first-order system, which depends on the amount of data used or the data required to model such a higher-order system.

- Optimisation method: conjugate gradients.
- Simulation with the statistical-moments-matching method.

Model validation with simulation is shown in Figures 7.12 to 7.14. The values of the statistical measures are $AE = 0.1494$, $SE = 0.0512$ and $LD = 27.87$.

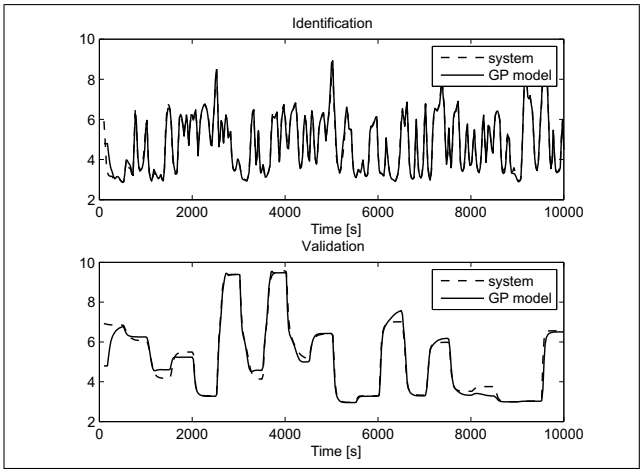

Figure 7.12: Simulation response of the system and mean values of the model

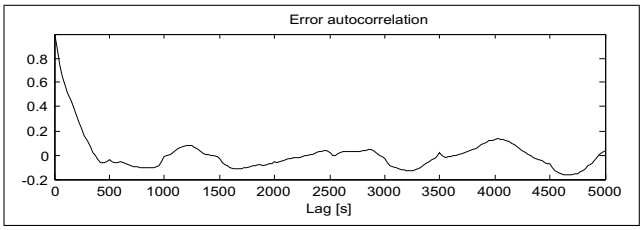

Figure 7.13: Simulation-error autocorrelation

In Figure 7.14, the areas with higher variance can be seen, which means lower confidence in the prediction. This is due to the model being validated with a signal that is dynamically very different from the signal we used to identify it. This means that the model is excited outside or at the edge of the region where it was identified. Therefore, we validate the system with a signal that also excites the system in the region where it was identified (Figures 7.15 and 7.16), which gives better validation results.

7.4 Control design

As explained earlier, the GP model is a nonparametric probabilistic model. The control-design methods that can use this type of model are more or less the same as those

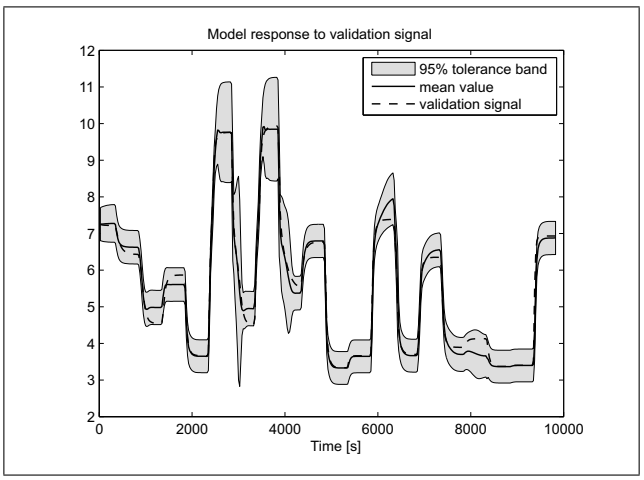

Figure 7.14: Simulation response to the validation signal together with the tolerance band

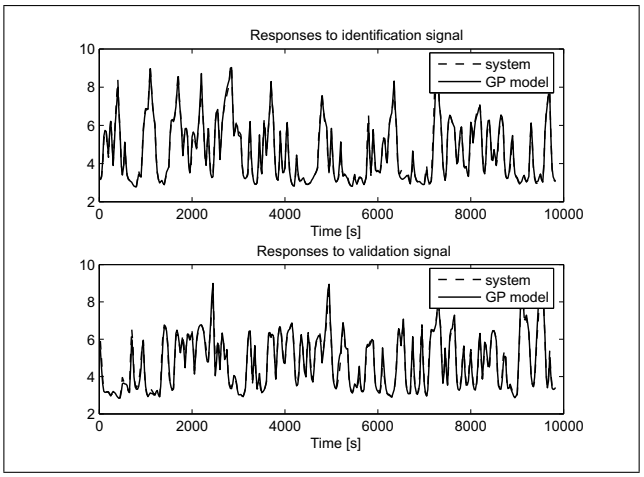

Figure 7.15: Validation with the second input signal

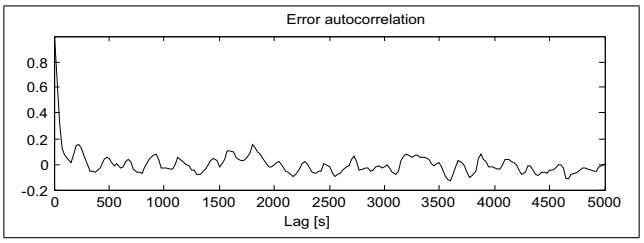

Figure 7.16: Simulation-error autocorrelation

described for use with neural networks. We will illustrate the case of predictive control using a GP model for a first-order system (3.9) [11] and for the pH-neutralisation process [12] in the same way as in Chapter 4.

We have the predictive functional control with the cost function

$$J = \min_{\mathbf{U}(k)} \left[r(k+P) - \hat{y}(k+P) \right]^2 .$$

The chosen prediction horizon is eight samples long and

the control horizon is one sample. When optimising the control signal, we could consider the amplitude and rate constraints at the input, output, and states, but we only considered the constraint on variance $\text{var}\,\hat{y}(k+P) \leq k_v$, which implicitly accounts for other constraints. In the range where the system is constrained, we cannot identify a good model or even any model at all, which is indicated by the large value of the variance.

The results of the setpoint tracking of the first-order system (3.9) without considering the constraints are shown in Figures 7.17 and 7.18, and with the constraint in Figures 7.19 and 7.20.

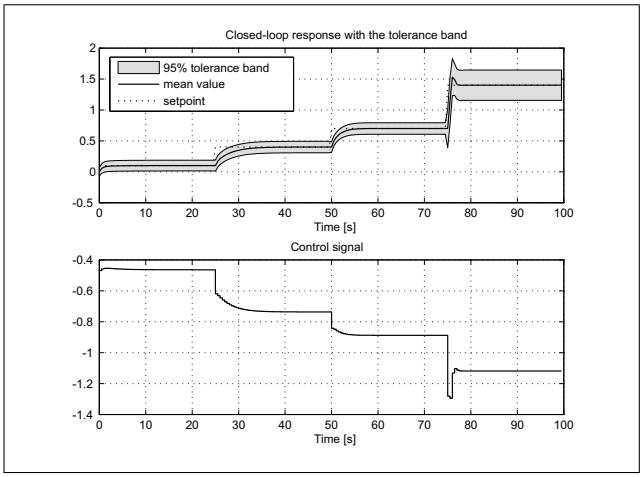

Figure 7.17: Closed-loop response - the unconstrained case

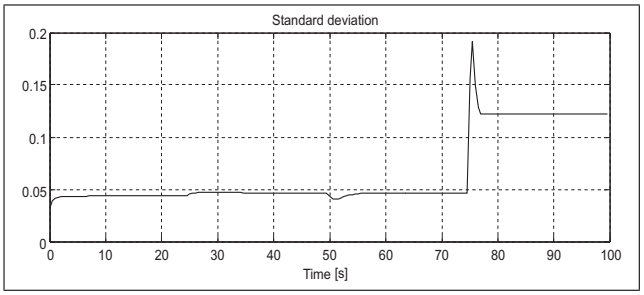

Figure 7.18: Standard deviation

The same control principle was applied to the process of pH neutralisation. The results of the setpoint tracking without considering the constraint are shown in Figures 7.21 and 7.22, and with the constraint in Figures 7.23 and 7.24.

A simple conclusion that can be drawn from the results shown is that predictive control can make sense of the information about model uncertainty. Predictive control does not allow the process to enter a region where the prediction deviation is greater than a prescribed threshold, and in this way it maintains the safety of the control.

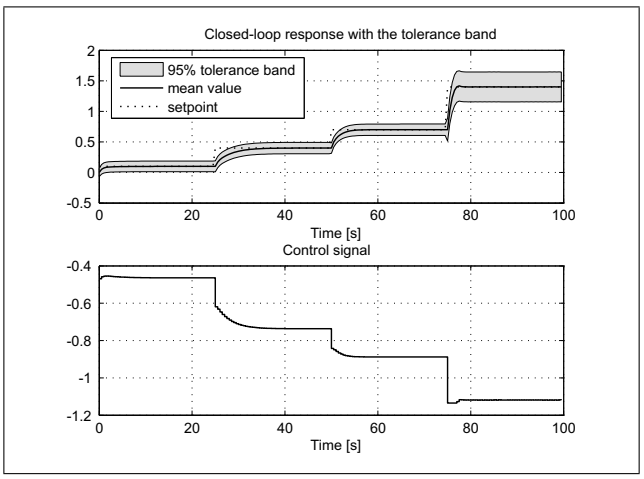

Figure 7.19: Closed-loop response: the case with the variance constraint at 0.13^2

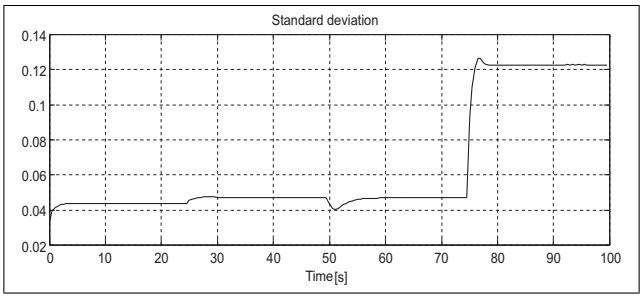

Figure 7.20: Standard deviation

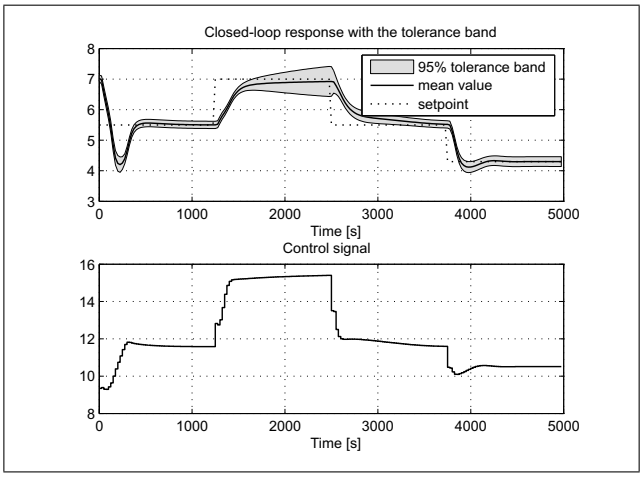

Figure 7.21: pH-process: closed-loop response – the unconstrained case

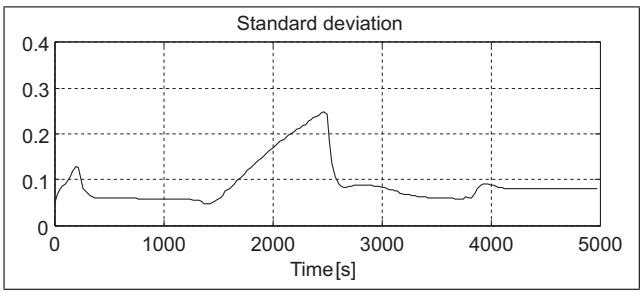

Figure 7.22: Standard deviation

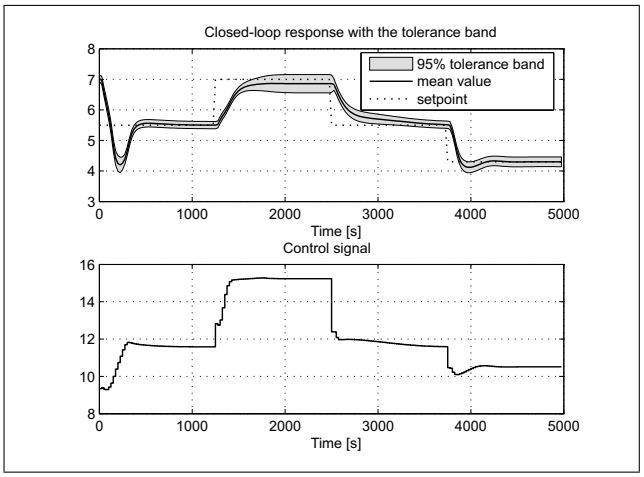

Figure 7.23: pH process: closed-loop response – the case with variance constraint at 0.15^2

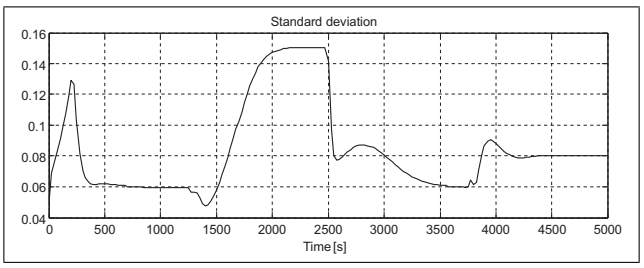

Figure 7.24: Standard deviation

Bibliography

[1] K. Ažman, J. Kocijan (2005): An example of Gaussian process model identification, L. Budin, S. Ribarić, (eds.), Proceedings of 28th International conference MIPRO, CIS – Intelligent Systems, Opatija, 79-84.

[2] K. Ažman, J. Kocijan (2006): Gaussian process model validation: biotechnological case studies, I. Troch, F. Breitenecker, (eds.) Proceedings of the 5th Vienna Symposium on Mathematical Modeling – MathMod, Dunaj.

[3] M. N. Gibbs (1997): Bayesian Gaussian processes for regression and classification, PhD dissertation, Cambridge University, Cambridge.

[4] A. Girard, C.E. Rasmussen, R. Murray-Smith (2002): Gaussian process priors with uncertain inputs: multiple-step-ahead prediction, Technical report DCS TR-2002-119, University of Glasgow, Glasgow.

[5] A. Girard (2004): Approximate methods for propagation of uncertainty with Gaussian process models, PhD dissertation, University of Glasgow, Glasgow.

[6] G. Gregorčič, G. Lightbody (2002): Gaussian processes for modelling of dynamic non-linear systems, Proceedings in Irish Signals and Systems Conference, Cork.

[7] M. A. Henson and D. E. Seborg (1994): Adaptive nonlinear control of a pH neutralization process, IEEE Trans. Control System Technology, Vol. 2, No. 3, 169-183.

[8] N. Hvala, S. Strmčnik, D. Šel, S. Milanić, B. Banko (2005): Influence of model validation on proper selection of process models — an industrial case study, Computers and Chemical Engineering, Vol. 29, 1507-1522.

[9] K. Judd, M. Small (2000): Towards long-term prediction, *Physica D*, Vol. 136, No. 1-2, 31-44.

[10] J. Kocijan, B. Banko, B. Likar, A. Girard, R. Murray-Smith, C.E. Rasmussen (2003): A case based comparison of identification with neural networks and Gaussian process models, Proceedings in IFAC ICONS conference, Faro, 137-142.

[11] J. Kocijan, R. Murray-Smith, C. E. Rasmussen, B. Likar (2003): Predictive control with Gaussian process models. V: B. Zajc (edt.), M. Tkalčič (edt.). Proceedings the IEEE Region 8 EUROCON 2003: computer as a tool, Vol. A, Ljubljana, 352-356.

[12] J. Kocijan, R. Murray-Smith (2005): Nonlinear predictive control with a Gaussian process model. V: R. Murray-Smith (edt.), R. Shorten(edt.). Switching and learning in feedback systems, Lecture notes in computer science, Vol. 3355, Springer, Heidelberg, 185-200.

[13] J. Kocijan, A. Girard, B. Banko, R. Murray-Smith (2005): Dynamic systems identification with Gaussian processes, Mathematical and Computer Modelling of Dynamic Systems, Vol. 11, No. 4, 411-424.

[14] I. Škrjanc, S. Blažič, O. Agamenonni (2005): Identification of dynamical systems with a robust interval fuzzy model, Automatica, Vol. 41, 327-332.

[15] L. Ljung (1999): System identification – theory for the user, Prentice Hall, New Jersey, the second edition.

[16] D. J. C. MacKay (1998): Introduction to Gaussian processes, C. M. Bishop (edt.), Neural networks and machine learning, NATO ASI Series – F 168, Springer-Verlag, Berlin, 133-166.

[17] D. J. C. MacKay (2003): Information theory, inference and learning algorithms, Chapter Gaussian Processes, Cambridge University Press, Cambridge, 535-548.

[18] D. J. Murray-Smith (1998): Methods for the external validation of continuous system simulation models: a review, Mathematical and Computer Modelling of Dynamical Systems, Vol. 4, 5-31.

[19] R. M. Neal (1996): Bayesian learning for neural networks, Lecture Notes in Statistics, Vol. 118, Springer-Verlag, New York.

[20] V. Peterka (1981): Bayesian system identification, P. Eykhoff (edt.), Trends and Progress in System Identification, Pergamon Press, Oxford, 239-304.

[21] J. Quiñonero-Candela, C. E. Rasmussen (2005): Analysis of some methods for reduced rank Gaussian process regression, R. Murray-Smith, (edt.),

R. Shorten, (edt.), Switching and Learning in Feedback Systems, Lecture Notes in Computer Science, Vol. 3355, Springer, Heidelberg, 98-127.

[22] C. E. Rasmussen (1996): Evaluation of Gaussian processes and other methods for nonlinear regresion, PhD dissertation, University of Toronto.

[23] C. E. Rasmussen, C. K. I. Williams (2006): Gaussian processes for machine learning, The MIT Press, Cambridge, MA.

[24] M. Seeger, C. K. I. Williams, N. D. Lawrence (2003): Fast forward selection to speed up sparse Gaussian process regression, C.M. Bishop, (edt.), B.J. Frey, (edt.), Proceedings of the Ninth International Workshop on AI and Statistics, Key West, FL.

[25] J. Sjöberg et al. (1995): Nonlinear black-box modeling in system identification: a unified overwiev, Automatica, Vol. 31, No. 12, 1691-1724.

[26] M. Small, K. Judd (1999): Variable prediction steps and long term prediction, Technical report, University of Western Australia, Dept. of Mathematics and Statistics.

[27] C. K. I. Williams (1998): Prediction with Gaussian processes: from linear regression and beyond, M. I. Jordan, (edt.), Learning in graphical models, Vol. 89 of *Nato Science Series D*, Springer, Berlin, 599-621.

[28] C. K. I. Williams, M. Seeger (2001): Using the Nyström method to speed up kernel machines, T. K. Leen, (edt.), T. G. Diettrich, (edt.),V. Tresp, (edt.), Proceedings of Advances in Neural Information Processing Systems conference, Vol. 13, The MIT Press, MA, 682-688.

Index